AWS + C3.ai Application Development Analysis Compared to Developing Using Native AWS Services

Source Code

Prepared by:
Cloud Native System Integrator

Table of Contents

Introduction

This document is a companion to the report entitled, “AWS + C3.ai Application Development Analysis Compared to Developing Using Native AWS Services.”

That report was produced by a third-party system integrator that was commissioned by C3.ai to develop a Predictive Maintenance application for a network of devices, designed to run on the AWS cloud. The system integrator was given a Product Specification and asked to develop the same application using two approaches:

1. Build the application using only AWS native services;

2. Build the application using the C3 AI Suite in conjunction with AWS services.

This document provides the source code for each version of the Device Predictive Maintenance application built by the third-party system integrator.

Readers can download a copy of the Product Specification document here.

C3 AI Suite –
Device Predictive Maintenance Source Code

UI

Pages

```
ui module lightbulb {
  page Home   {
      "id": "lightbulb.Home",
      "name": "Home",
      "template": [
         "lightbulb.DashboardTemplate"
      ],
      "thumbnail": "template-thumbs/120/ui.Dashboard2x2.gif",
      "title": "Home",
      "url": "home",
      "components": [
         {
            "id": "HistogramChart",
            "component": "UIViewChartMetricsHistogram",
            "icon": "bar-chart",
            "name": "HistogramChart",
            "box": true,
            "title": "Failure Risk",
            "numberPoints": "-1",
            "numberBins": "10",
            "renderTo": "#RightColRow1",
            "height": 556,
            "titleUrl": null,
            "data": {
               "itemsDataSource": "lightbulb.Bulbs"
            },
            "metricNameTitle": true,
            "metricNameTooltip": true,
            "chartingType": "count",
            "binningType": "value",
            "labelDecimals": null,
            "worstPerformingLegendText": null,
            "period": null,
            "start": null,
            "pointPath": null,
```

```
    "pointWidth": "25",
    "nameField": null,
    "countText": null,
    "maxValueText": null,
    "nameText": null,
    "valueText": null,
    "percentileText": null,
    "drilldownBackText": null,
    "noHistogramDataMessageText": null,
    "noItemDataMessageText": null,
    "errorDataMessageText": null,
    "ignoreZeroValues": false,
    "displayXAxisLegend": false,
    "tooltipBeneathColumns": false,
    "currentMomentEval": false,
    "enableDrilldown": false,
    "histogramMetric": {
      "metricName": "RiskScore",
      "metricTitle": "RiskScore"
    },
    "colors": [
      "#4f6e04",
      "#31b82e",
      "#74f03a",
      "#f5df16",
      "#ecf029",
      "#f6f79c",
      "#ffa462",
      "#f27201",
      "#ff6d4d",
      "#e34400"
    ]
  },
  {
    "id": "Map",
    "component": "UIViewMap",
    "icon": "map-marker",
    "name": "Map",
    "box": true,
    "title": "Status Map",
    "autoLoadData": true,
    "lat": "latitude",
    "lng": "longitude",
    "autocenter": true,
```

```
    "cluster": false,
    "mapConfig": {
      "mapTypeId": "roadmap"
    },
    "components": [
      {
        "component": "UIViewTooltip",
        "id": "PhantomTooltip",
        "renderTo": ".phantom-tooltip"
      }
    ],
    "renderTo": "#RightColRow2",
    "height": 556,
    "markerColors": [
      {
        "id": "1",
        "color": "#fc3d00",
        "display": "if 'currentPrediction.prediction > 0.5' then color = #fc3d00",
        "field": "currentPrediction.prediction",
        "value": "0.5",
        "comparator": ">"
      }
    ],
    "useInfoWindowTpl": false,
    "data": {
      "collection": "lightbulb.Bulbs"
    },
    "allowGeoFilter": false,
    "showNumberOfRecords": false
  },
  {
    "id": "Grid",
    "component": "UIViewKendoGrid",
    "icon": "align-justify",
    "name": "Grid",
    "box": true,
    "paginate": true,
    "title": "Lightbulb Details",
    "data": {
      "collection": "lightbulb.Bulbs"
    },
    "filterable": false,
    "renderTo": "#RightColRow3",
    "height": 556,
```

```
    "titleUrl": null,
    "aggregate": null,
    "small": false,
    "checkBoxes": false,
    "columns": [
      {
        "id": "id",
        "field": "id",
        "label": "Bulb ID"
      },
      {
        "id": "bulbType",
        "field": "bulbType",
        "label": "Bulb Type"
      },
      {
        "id": "manufacturer.id",
        "field": "manufacturer.id",
        "label": "Manufacturer"
      },
      {
        "id": "startDate",
        "field": "startDate",
        "label": "Start Date",
        "format": "time"
      },
      {
        "id": "currentPrediction.prediction",
        "field": "currentPrediction.prediction",
        "label": "Risk Score",
        "format": "humanize"
      }
    ]
  },
  {
    "id": "FilterPanel",
    "component": "UIViewFilterPanel",
    "icon": "lightbulb",
    "name": "FilterPanel",
    "box": false,
    "title": "Filter",
    "filterForm": {
      "component": "UIViewFilterForm",
      "filters": [
```

```
                {
                  "id": "manufacturer.id",
                  "field": "manufacturer.id",
                  "label": "Manufacturer",
                  "dataType": "string",
                  "comparator": false,
                  "component": "UIViewFieldText"
                },
                {
                  "id": "bulbType",
                  "field": "bulbType",
                  "label": "Bulb type",
                  "dataType": "string",
                  "comparator": false,
                  "component": "UIViewFieldText"
                }
              ],
              "id": "FilterForm"
            },
            "renderTo": "#LeftCol",
            "height": 556,
            "data": {
              "collection": "lightbulb.Bulbs"
            },
            "geoFilterText": "Geographical Filtering",
            "allowGeoFilter": false
          }
        ]
    }
}
```

Templates

```
{{ div.container-fluid.c3-page.c3-page-dashboard({cls: cls}) do }}
 <div class="row">
 <div class="c3-component-socket col-xs-12 col-sm-3" id="LeftCol"></div>
 <div class="col-xs-12 col-sm-9" id="RightCol">
 <div class="row">
 <div class="c3-component-socket col-xs-12" id="RightColRow1"></div>
 </div>
 <div class="row">
 <div class="c3-component-socket col-xs-12" id="RightColRow2"></div>
 </div>
 <div class="row">
```

```
 <div class="c3-component-socket col-xs-12" id="RightColRow3"></div>
 </div>
 </div>
 </div>
{{ end }}

ui module lightbulb {
  dataSource Bulbs   {
    "collection":true,
    "c3function":"fetch",
    "c3arguments":{
      "spec":{
        "limit":"-1",
        "include":"currentPrediction.prediction, manufacturer, bulbType, startDate,
manufacturer, bulbStatus, latitude, longitude"
      }
    },
    "c3type":"SmartBulb",
    "name":"Bulbs",
    "responseSelector":"objs",
    "record":false,
    "responseTransform":{
      "combine":false,
      "rootProperty":false,
      "firstItem":false,
      "order":false,
      "tuples":false,
      "fields":false
    },
    "id":"lightbulb.Bulbs"
  }
}
```

DataSources

```
ui module lightbulb {
 application lightbulb {
 "name": "lightbulb",
 "humanName": "Smart Bulb Failure Prediction",
 "showDesignerComponents": true,
 "url": "lightbulb",
 "id": "lightbulb.lightbulb",
 "theme": "base",
 "package": "lightbulbTraining",
```

```
 "defaultPage": "Home",
 "thumbnail": "template-thumbs/120/ui.Dashboard2x2.gif",
 "noneditable": false,
 "moduleName": "lightbulb"
 }
}
```

SRC

types

```
/**
* SmartBulbToFixtureRelation.c3typ
* Describes a timed relationship between a smart bulb and a fixture.
* For more information, see {@link TimedIntervalRelation}.
*/
@db(index=['to', 'from'])
entity type SmartBulbToFixtureRelation mixes TimedIntervalRelation<SmartBulb,
Fixture> schema name "SMRT_BLB_FXTR_RLTN"

/*
* Copyright 2009-2018 C3 IoT, Inc. All Rights Reserved.
* This material, including without limitation any software, is the confidential trade
secret
* and proprietary information of C3 IoT and its licensors. Reproduction, use and/or
distribution
* of this material in any form is strictly prohibited except as set forth in a written
* license agreement with C3 IoT and/or its authorized distributors.
* This product may be covered by one or more U.S. patents or pending patent
applications.
~*/

/**
 * A prediction made for a single {@link SmartBulb}.
 */
 @db(compactType=true,
  datastore='cassandra',
  partitionKeyField='smartBulb',
  persistenceOrder='timestamp',
  persistDuplicates=false,
  shortId=true,
  shortIdReservationRange=100000)
entity type SmartBulbPrediction schema name "SMRT_BLB_PRDCTN" {
```

```
 /**
  * The calculated risk score for this SmartBulb
  */
 prediction: double

 /**
  * The {@link SmartBulb} for which this prediction was made.
  */
 smartBulb: SmartBulb

 /**
  * The time at which this prediction was made.
  */
 timestamp: datetime

}

/*
* A series of measurements taken from a single {@link SmartBulb}.
*/
entity type SmartBulbMeasurementSeries mixes
TimeseriesHeader<SmartBulbMeasurement> schema name "SMRT_BLB_MSRMNT_SRS"
{

 // The {@link SmartBulb} for which measurements were taken.
 smartBulb: SmartBulb
 // The aggregation/disaggregation {@link Treatment} to use for the measurements.
 treatment: string enum('previous', 'rate', 'integral')
}

/**
* A single measurement taken from a single {@link SmartBulb}.
*/
@db(datastore='cassandra',
 partitionKeyField='parent',
 persistenceOrder='start',
 persistDuplicates=false,
 compactType=true,
 shortIdReservationRange=100000)
entity type SmartBulbMeasurement mixes
TimeseriesDataPoint<SmartBulbMeasurementSeries>  schema name
'SMRT_BLB_MSRMNT' {
 // The measured number of lumens.
```

```
 @ts(treatment='rate')
 lumens: double
 // The measured power consumption.
 @ts(treatment='rate')
 power: double
 // The measured temperature.
 @ts(treatment='rate')
 temperature: double
 // The measured voltage.
 @ts(treatment='rate')
 voltage: double
 // The status of the smart bulb (on or off).
 @ts(treatment='previous')
 status: int
}

/*
* Copyright 2009-2018 C3 IoT, Inc. All Rights Reserved.
* This material, including without limitation any software, is the confidential trade
secret
* and proprietary information of C3 IoT and its licensors. Reproduction, use and/or
distribution
* of this material in any form is strictly prohibited except as set forth in a written
* license agreement with C3 IoT and/or its authorized distributors.
* This product may be covered by one or more U.S. patents or pending patent
applications.
~*/

/*
 * Returns the expected lumens of a light bulb based on wattage and bulbType
 */
function expectedLumens(wattage, bulbType) {
  if(bulbType == 'LED')
    return wattage * 84;
  if(bulbType == 'INCAN')
    return wattage * 14;
  if(bulbType == 'CFL')
    return wattage * 62;
}

/*
 * Returns the lifespan of a smart bulb in years.
 */
function lifeSpanInYears(bulbId){
```

```
  var bulb, startTime, defectFilter, defectDatum, defectTime, lifespan, conversionFactor,
lifeSpanInYears;
  bulb = SmartBulb.get({id:bulbId});
  startTime = bulb.startDate;
  defectFilter = "status == 1 && lumens == 0 && parent.id == '" + 'SBMS_serialNo_' +
bulb.id + "'";
  defectDatum = SmartBulbMeasurement.fetch({filter:defectFilter});
  defectTime = defectDatum.objs[0].end;
  lifespan = defectTime - startTime;
  conversionFactor = 1000*60*60*24*365;
  lifeSpanInYears = lifespan / conversionFactor;
  return lifeSpanInYears;
}

/*
 * Return a string with the id of the SmartBulb shortest life span.
 */
function shortestLifeSpanBulb(){
   // declare variables to use later
   var lightbulbs, shortestId, shortestVal, span, bulbId, lifespans = [];
   //perform a fetch of SmartBulbs, include just the "id" field, set the limit to unlimited (-
1), and select the objs object in the response
   lightbulbs = SmartBulb.fetch({include:"id",limit:-1}).objs;
   //Set an arbitrary shortest starting point
   shortestVal = 1000;
   //loop through our array, calling lifeSpanInYears() on each id, and performing a
bubble sort
   for (var i = 0; i < lightbulbs.length; i++) {
     bulbId = lightbulbs[i].id;
     span = SmartBulb.lifeSpanInYears(bulbId);
     lifespans.push(span);
     //bubblesort
     if (span < shortestVal){
       shortestVal = span;
       shortestId = bulbId;
     }
   }
   //return the shortestId
   return shortestId;
 }

 /*
  * Return a string with the id of the SmartBulb longest life span.
  */
```

```
 function longestLifeSpanBulb(){
   // declare variables to use later
   var lightbulbs, longestId, longestVal, span, bulbId, lifespans = [];
   //perform a fetch of SmartBulbs, include just the "id" field, set the limit to unlimited (-
1), and select the objs object in the response
   lightbulbs = SmartBulb.fetch({include:"id",limit:-1}).objs;
   //Set 0 as the longest so far
   longestVal = 0;
   //loop through our array, calling lifeSpanInYears() on each id, and performing a
bubble sort
   for (var i = 0; i < lightbulbs.length; i++) {
     bulbId = lightbulbs[i].id;
     span = SmartBulb.lifeSpanInYears(bulbId);
     lifespans.push(span);
     //bubblesort
     if (span > longestVal){
       longestVal = span;
       longestId = bulbId;
     }
   }
   //return the longestId
   return longestId;
 }

 /*
  * Returns the average life span of all smart bulbs.
  */
 function averageLifeSpan(){
   // declare variables to use later
   var lightbulbs, sum, avg, span, bulbId, lifespans = [];
   //perform a fetch of SmartBulbs, include just the "id" field, set the limit to unlimited (-
1), and select the objs object in the response
   lightbulbs = SmartBulb.fetch({include:"id",limit:-1}).objs;
   //Set 0 as the avg so far
   sum = 0;
   //loop through our array, calling lifeSpanInYears() on each id, and performing a
bubble sort
   for (var i = 0; i < lightbulbs.length; i++) {
     bulbId = lightbulbs[i].id;
     span = SmartBulb.lifeSpanInYears(bulbId);
     lifespans.push(span);
     //add all lifepans
     sum += span;
   }
```

```
   // calc the average
   avg = sum / lightbulbs.length;

   // return the average
   return avg;
  }


/**
* SmartBulbEvent.c3typ
* An event that happens to a single {@link SmartBulb}.
*/
entity type SmartBulbEvent schema name "SMRT_BLB_EVNT" {
 // The time at which this event ended.
 end: datetime
 // The time at which this event began.
 start: datetime
 // The event code used to distinguish events.
 eventCode: string
 // The type of this event.
 eventType: string
 // The {@link SmartBulb} connected to this event.
 smartBulb: SmartBulb
}

/**
* PowerGridStatusSet.c3typ
* The status of the power grid for a {@link Building} at a specific time.
*/
@db(datastore='cassandra',
 partitionKeyField='parent',
 persistenceOrder='timestamp',
 persistDuplicates=false,
 compactType=true,
 shortId=true,
 shortIdReservationRange=100000)
entity type PowerGridStatusSet mixes TimedValueHistory<Building>, Integer schema
name 'PWR_GRD_STTS_ST'

/**
* SmartBulb.c3typ
* A bulb capable of storing power consumption, light output, location, and more.
*/
entity type SmartBulb extends LightBulb mixes MetricEvaluatable type key "SMRT_BLB"
```

```
{
 // The latitude of this bulb.
 latitude: double
 // The longitude of this bulb.
 longitude: double
 // The unit of measurement used for this bulb's light output measurements.
 lumensUOM: Unit
 // The unit of measurement used for this bulb's power consumption measurements.
 powerUOM: Unit
 // The unit of measurement used for this bulb's temperature measurements.
 temperatureUOM: Unit
 // The unit of measurement used for this bulb's voltage measurements.
voltageUOM: Unit
 // This bulb's historical events.
 bulbEvents: [SmartBulbEvent](smartBulb)
// This bulb's historical measurements.
 bulbMeasurements: [SmartBulbMeasurementSeries](smartBulb)
// The collection of relations that pertain to this smart bulb.
@db(order='descending(start), descending(end)')
fixtureHistory: [SmartBulbToFixtureRelation](from)
// The current {@link Fixture} to which this smart bulb is attached.
currentFixture: Fixture stored calc 'fixtureHistory[0].(end == null).to'
/**
* Returns the life span of this smartBulb
 */
 lifeSpanInYears: function(bulbId: string): double js server

 /**
* Returns the longest life span of any smartBulb
 */
 longestLifeSpanBulb: function(): string js server

 /**
* Returns the shortest life span of any smartBulb
 */
 shortestLifeSpanBulb: function(): string js server

 /**
* Returns the avg life span of all smartBulbs
 */
 averageLifeSpan: function(): double js server

 /**
  * This bulb's historical predictions.
```

```
 */
 @db(order='descending(timestamp)')
 bulbPredictions: [SmartBulbPrediction](smartBulb)

 /**
  * This bulb's latest prediction.
  */
 currentPrediction: SmartBulbPrediction stored calc "bulbPredictions[0]"


}

/**
* PowerGridStatus.c3typ
* The status of the power grid for a {@link Building} at a specific time.
*/
type PowerGridStatus mixes TimedValue, Integer

/**
* Manufacturer.c3typ
* A company that manufactures lightbulbs.
*/
entity type Manufacturer mixes MetricEvaluatable schema name "MNFCT"

/**
* LightBulb.c3typ
* A single light bulb.
*/
extendable entity type LightBulb schema name "LGHT_BLB" {
 // The type of this light bulb.
 bulbType: string enum('LED', 'INCAN', 'CFL')
 // The light bulb's wattage
 wattage: decimal

 // The name of this light bulb's manufacturer.
 manufacturer: Manufacturer
 // The time at which this light bulb was manufactured.
 startDate: datetime
}

/**
* Fixture.c3typ
* A single fixture in which a single {@link SmartBulb} may be connected.
*/
```

```
entity type Fixture mixes MetricEvaluatable schema name "FXTR" {
   // The {@link Apartment} in which this fixture is located.
   apartment: Apartment
   // The collection of relations that map to this fixture.
   @db(order='descending(toUTC(start)), descending(toUTC(end))')
   bulbHistory: [SmartBulbToFixtureRelation](to)
   // The current {@link SmartBulb} attached to this fixture.
   currentBulb: SmartBulb stored calc 'bulbHistory[0].(end == null).from'

}

/**
* Apartment.c3typ
* A single apartment unit in a {@link Building}.
*/
entity type Apartment mixes MetricEvaluatable schema name "APRTMNT" {
 // The {@link Building} in which this apartment is located.
 building: Building
 // An array of fixtures that exist in this apartment.
 fixtures: [Fixture](apartment)
}

/**
* Apartment.c3typ
* A single building containing many {@link Apartment}s.
*/
entity type Building mixes MetricEvaluatable schema name "BLDNG"{

 // The apartments that are inside this building.
 apartments: [Apartment](building)

// The collection of statuses relevant to this {@link Building}.
@db(order='descending(timestamp)')
gridStatusSet: [PowerGridStatusSet](parent)
// The current status of the power grid for this {@link Building}.
@db(timedValueHistoryField='gridStatusSet')
gridStatus: PowerGridStatus

}
```

Transforms

```
/**
```

```
* This type encapsulates the data flow from the {@link CanonicalSmartBulb} to the
{@link SmartBulbMeasurementSeries} type.
*/
type TransformCanonicalSmartBulbToSmartBulbMeasurementSeries mixes
SmartBulbMeasurementSeries transforms CanonicalSmartBulb {
// We avoid transforming from the measurement data to the headers to prevent
creating/updating the headers an excessive number of times. This is very important at
large scales.

/**
 * The format of the id will be "SBMS_serialNo_serialNumber" to ensure that no
two series will be duplicates.
 */
   id: ~ expression "concat('SBMS_serialNo_',SN)"
   smartBulb: ~ expression { id: "SN" }
   interval: ~ expression "'HOUR'"
   treatment: ~ expression "'rate'"
}

/**
* This type encapsulates the data flow from the {@link CanonicalSmartBulb} to the
{@link SmartBulb} type.
*/
type TransformCanonicalSmartBulbToSmartBulb mixes SmartBulb transforms
CanonicalSmartBulb {
 id: ~ expression "SN"
 manufacturer: ~ expression {id: "Manufacturer"}
 bulbType: ~ expression "BulbType"
 wattage: ~ expression "Wattage"
 startDate: ~ expression "StartDate"
 latitude: ~ expression "Latitude"
 longitude: ~ expression "Longitude"
 lumensUOM: ~ expression "{ \"id\": \"'lumen'\" }"
 powerUOM: ~ expression "{ \"id\": \"'watt'\" }"
 temperatureUOM: ~ expression "{ \"id\": \"'degrees_fahrenheit'\" }"
 voltageUOM: ~ expression "{ \"id\": \"'volt'\" }"
}

/**
* This type encapsulates the data flow from the {@link CanonicalSmartBulb} to the
{@link Manufacturer} type.
*/
type TransformCanonicalSmartBulbToManufacturer mixes Manufacturer transforms
CanonicalSmartBulb {
```

```
 id: ~ expression "Manufacturer"
}

/**
* This type encapsulates the data flow from the {@link
CanonicalSmartBulbToFixtureRelation} to the {@link SmartBulbToFixtureRelation}
type.
*/
type TransformCanonicalSmartBulbToFixtureRelationToSmartBulbToFixtureRelation
mixes SmartBulbToFixtureRelation transforms CanonicalSmartBulbToFixtureRelation {
 id: ~ expression "concat(SN,'_',fixture)"
 to: ~ expression { id: "fixture" }
 from: ~ expression { id: "SN" }
 end: ~ expression "end"
 start: ~ expression "start"
}

/**
* This type encapsulates the data flow from the {@link
CanonicalSmartBulbMeasurement} to the {@link SmartBulbMeasurement} type.
*/
type TransformCanonicalSmartBulbPredictionToSmartBulbPrediction mixes
SmartBulbPrediction transforms CanonicalSmartBulbPrediction {


 prediction: ~ expression "prediction"

 smartBulb: ~ expression {id: "idSmartbulb"}

 timestamp: ~ expression "timestamp"
}

/**
* This type encapsulates the data flow from the {@link
CanonicalSmartBulbMeasurement} to the {@link SmartBulbMeasurement} type.
*/
type TransformCanonicalSmartBulbMeasurementToSmartBulbMeasurement mixes
SmartBulbMeasurement transforms CanonicalSmartBulbMeasurement {
 /**
 * Use the serial number field ("SN") of the canonical to create a properlyformatted
pointer to the parent {@link SmartBulbMeasurementSeries} object.
 */
 parent: ~ expression { id: "concat('SBMS_serialNo_',SN)" }
 lumens: ~ expression "Lumens"
```

```
 power: ~ expression "Watts"
 voltage: ~ expression "Voltage"
 temperature: ~ expression "Temp"
 start: ~ expression "TS"
 end: ~ expression "TS + period(1, 'HOUR')"
 status: ~ expression "Status"
}

/**
* This type encapsulates the data flow from the {@link CanonicalSmartBulbEvent} to
the {@link SmartBulbEvent} type.
*/
type TransformCanonicalSmartBulbEventToSmartBulbEvent mixes SmartBulbEvent
transforms CanonicalSmartBulbEvent {
 id: ~ expression "id"
 end: ~ expression "end"
 start: ~ expression "start"
 eventCode: ~ expression "eventCode"
 eventType: ~ expression "eventType"
 smartBulb: ~ expression { id: "bulb" }
}

/**
* This type encapsulates the data flow from the {@link CanonicalPowerGridStatus}
to the {@link PowerGridStatusSet} type.
*/
type TransformCanonicalPowerGridStatusToBuilding mixes Building transforms
CanonicalPowerGridStatus {
 id: ~ expression "Building"
 gridStatus: ~ expression { timestamp: "TS", value: "Status"}
}

/**
* This type encapsulates the data flow from the {@link CanonicalFixture} to the
{@link Apartment} type.
*/
type TransformCanonicalFixtureToFixture mixes Fixture transforms CanonicalFixture {

 id: ~ expression "fixture"
 apartment: ~ expression { id: "apartment" }
}

/**
* This type encapsulates the data flow from the {@link CanonicalFixture} to the
```

```
{@link Building} type.
*/
type TransformCanonicalFixtureToBuilding mixes Building transforms CanonicalFixture
{
 id: ~ expression "building"
}

/**
* This type encapsulates the data flow from the {@link CanonicalFixture} to the
{@link Apartment} type.
*/
type TransformCanonicalFixtureToApartment mixes Apartment transforms
CanonicalFixture {
 id: ~ expression "apartment"
 building: ~ expression { id: "building" }
}
```

Jobs

```
/*
* Copyright 2009-2018 C3 IoT, Inc. All Rights Reserved.
* This material, including without limitation any software, is the confidential trade
secret
* and proprietary information of C3 IoT and its licensors. Reproduction, use and/or
distribution
* of this material in any form is strictly prohibited except as set forth in a written
* license agreement with C3 IoT and/or its authorized distributors.
* This product may be covered by one or more U.S. patents or pending patent
applications.
~*/

var logger = C3.logger("lightbulb.AggPower");

/*
 * Process a set of light bulbs and return the aggregate Power for a given manufacturer.
 */
function map(batch, objs, job) {
  // For a batch of Lightbulbs, calculate the aggregate power for a manufacturer and
  // return a map with the key the manufacturer and the value the aggregate power.
  var manufacturerConsumptionMap = {},
      consumptionMetricName = "AveragePower",
      currentSmartBulbConsumptionTimeseries,
      currentSmartBulbConsumptionSumData;
```

```
  // calculate the energy usage for each lightbulb
  var averageConsumptionMetricResults = SmartBulb.evalMetrics({
    ids:         objs.pluck("id"),
    expressions:  [consumptionMetricName],
    start:       job.startDate,
    end:         job.endDate,
    interval:    job.interval
  });

  // loop over each smart bulb, calculating its aggregate energy consumption
  objs.each(function(smartBulb) {
    // retrieve the timeseries referring to the current smart bulb
    currentSmartBulbConsumptionTimeseries =
averageConsumptionMetricResults.result[smartBulb.id][consumptionMetricName];

    // calculate the total energy consumption over the time period in question for the
current smart bulb
    currentSmartBulbConsumptionSumData =
currentSmartBulbConsumptionTimeseries.aggregate("SUM").data();

    // as long as currentSmartBulbConsumptionTimeseries is not empty, this condition
should always pass, but just to be safe...
    if (!_.isEmpty(currentSmartBulbConsumptionSumData)) {
      // update the manufacturer map with the current smart bulb's total concumption
      if (!_.has(manufacturerConsumptionMap, smartBulb.manufacturer)) {
        manufacturerConsumptionMap[smartBulb.manufacturer] = 0;
      }
      manufacturerConsumptionMap[smartBulb.manufacturer] +=
currentSmartBulbConsumptionSumData[0];
    }
  });

  return manufacturerConsumptionMap;
}

/*
 * Persists the aggregate power of lightbulbs for a manufacturer for a given time range
 */
function reduce(key, objs, job) {
  // sum the consumptions of the current manufacturer's bulbs calculated from all
batches
  var consumptionSum = _.reduce(objs, function(result, current) { return result + current;
}, 0);
  var manufacturer = Manufacturer.get(key);
```

```
  // create the AggregateConsumptionByManufacturer object to record this job's results
  var consumptionByManufacturer = AggregateConsumptionByManufacturer.make({
    parent: manufacturer,
    start: job.startDate,
    end: job.endDate,
    aggregateConsumption: consumptionSum
  });
  consumptionByManufacturer.create();

  // return the results to store them in the job object as well (accessible using
job.results())
  return [consumptionSum];
}

/*
* Copyright 2009-2018 C3 IoT, Inc. All Rights Reserved.
* This material, including without limitation any software, is the confidential trade
secret
* and proprietary information of C3 IoT and its licensors. Reproduction, use and/or
distribution
* of this material in any form is strictly prohibited except as set forth in a written
* license agreement with C3 IoT and/or its authorized distributors.
* This product may be covered by one or more U.S. patents or pending patent
applications.
~*/

/**
 * Map reduce job that returns the aggregage power consumption per manufacturer
 */
entity type AggregateConsumptionByManufacturerJob extends MapReduce<SmartBulb,
string, double, double> type key 'AGG_CNS_MNFCT' {

  /**
   * Start of the evaluation interval
   */
  startDate: datetime

  /**
   * End of the evaluation interval
   */
  endDate: datetime

  /**
```

```
  * Metric evaluation interval
  */
 interval: string

 /**
  * Map function
  */
 map:      ~ js server

 /**
  * Reduce function
  */
 reduce:    ~ js server

}

/*
* Copyright 2009-2018 C3 IoT, Inc. All Rights Reserved.
* This material, including without limitation any software, is the confidential trade
secret
* and proprietary information of C3 IoT and its licensors. Reproduction, use and/or
distribution
* of this material in any form is strictly prohibited except as set forth in a written
* license agreement with C3 IoT and/or its authorized distributors.
* This product may be covered by one or more U.S. patents or pending patent
applications.
~*/

/**
 * A single aggregate power measurement for a manufacturer
 */
@db(compactType=true,
 datastore='cassandra',
 partitionKeyField='parent',
 persistenceOrder='start',
 persistDuplicates=false,
 shortId=true,
 shortIdReservationRange=100000)
entity type AggregateConsumptionByManufacturer schema name "AGG_CNS_MNFCT" {
 parent: Manufacturer
 start: datetime
 end: datetime
 aggregateConsumption: double
}
```

Canonicals

```
/**
* This type represents the raw data that will represent {@link
SmartBulbToFixtureRelation} information.
*/
type CanonicalSmartBulbToFixtureRelation mixes
Canonical<CanonicalSmartBulbToFixtureRelation> {

 // This represents the id of a {@link SmartBulb}
 SN: string

 // This represents the id of a {@link Fixture}
 fixture: string

// The date time to use for the start of a {@link SmartBulbToFixtureRelation}
 start: datetime

 // The date time to use for the end of a {@link SmartBulbToFixtureRelation}
 end: datetime
}

/**
* This type represents the raw data that will represent {@link Fixture} information.
*/
type CanonicalSmartBulbPrediction mixes Canonical<CanonicalSmartBulbPrediction> {
 id2: string

 prediction: double

 idSmartbulb: string

 timestamp: datetime
}

/**
* This type represents the raw data that will represent {@link SmartBulb}information.
*/
type CanonicalSmartBulb mixes Canonical<CanonicalSmartBulb> {
 // This represents the manufacturer of a {@link LightBulb}
 Manufacturer: string
 // This represents the bulb type of a {@link LightBulb}
 BulbType: string
```

```
 // This represents the wattage of a {@link LightBulb}
 Wattage: decimal
 // This represents the id of a {@link LightBulb}
 SN: string
 // This represents the start date of a {@link LightBulb}
 StartDate: datetime
 // This represents the latitude of a {@link SmartBulb}
 Latitude: double
 // This represents the longitude of a {@link SmartBulb}
 Longitude: double
}

/**
* This type represents the raw data that will represent {@link SmartBulbEvent}
information.
*/
type CanonicalSmartBulbEvent mixes Canonical<CanonicalSmartBulbEvent> {
 // This represents the id for a {@link SmartBulbEvent}
 id: string
 // This represents the id of a {@link SmartBulb}
 bulb: string
 // This represents the eventCode of a {@link SmartBulbEvent}
 eventCode: string
 // This represents the eventType of a {@link SmartBulbEvent}
 eventType: string
 // This represents a datetime to use as the start of a {@link SmartBulbEvent}
 start: datetime
 // This represents a datetime to use as the end of a {@link SmartBulbEvent}
 end: datetime
}

/**
* This type represents the raw data that will represent {@link PowerGridStatus}
information.
*/
type CanonicalPowerGridStatus mixes Canonical<CanonicalPowerGridStatus> {

 // This represents the datetime to use for the start of {@link PowerGridStatus}
 TS: datetime
 // This represents the id of the {@link Building}
 Building: string
 // This represents the status ON or OFF (1 or 0) for the power grid of a {@Building}
 Status: int
}
```

```
/**
* This type represents the raw data that will represent {@link Fixture} information.
*/
type CanonicalFixture mixes Canonical<CanonicalFixture> {
 // This represents the id of a {@link Fixture}
 fixture: string
 // This represents the id of an {@link Apartment}
 apartment: string
 // This represents the id of a {@link Building}
 building: string
}
```

Analytics

```
@DFE(period='Hour', interval='Hour', flattenWindows=true)
type SmartBulbOverheatInput mixes CompoundDataFlowEvent<SmartBulb> {
        temperature: SmartBulbOverheat
}

type SmartBulbOverheatAlert mixes Analytic<SmartBulbOverheatInput,
SmartBulbEvent> {
        process : ~ js server
}

@DFE(period='Hour', interval='Hour', metric='AverageTemperature', includeSpec: 'this')
type SmartBulbOverheat mixes TSDataFlowEvent<SmartBulb>

@DFE(period='Hour', interval='Hour', flattenWindows=true)
type SmartBulbLongLifeInput mixes CompoundDataFlowEvent<SmartBulb> {
 bulbLife: SmartBulbLongLife
}

var log = C3.logger("SmartBulbLongLifeAlert");

var DURATION_THRESHOLD = 10500;

function process(input) {
  var data = input.bulbLife.data(),
     dates = input.bulbLife.dates();
  for (var i = 0; i < data.length; i++) {
    // If the duration is greater than the threshold then we need to update the source
object
    if (data.at(i) > DURATION_THRESHOLD) {
```

```
    return SmartBulbEvent.make({
      smartBulb: input.source,
      eventCode: "LONGLIFE",
      eventType: "Health",
      start: dates.at(i),
      end: dates.at(i+1)
    });
   }
  }
}


type SmartBulbLongLifeAlert mixes Analytic<SmartBulbLongLifeInput, SmartBulbEvent>
{
 process : ~ js server
}

@DFE(period='Hour', interval='Hour', metric='DurationOnInHours',
flattenWindows=true)
type SmartBulbLongLife mixes TSDataFlowEvent<SmartBulb>

@DFE(period='Hour', interval='Hour', flattenWindows=true)
type SmartBulbDefectiveInput  mixes CompoundDataFlowEvent<SmartBulb> {
        defective: SmartBulbDefective
}

type SmartBulbDefectiveAlert mixes Analytic<SmartBulbDefectiveInput,
SmartBulbEvent> {
        process : ~ js server
}

var log = C3.logger("SmartBulbDefectiveAlert");

var FAILED = 1;

function process(input) {
  var data  = input.defective.data(),
     dates = input.defective.dates();
  for (var i = 0; i < data.length; i++) {
    // If defective is 1 then we need to update the source object
    if (data.at(i) == FAILED) {
      return SmartBulbEvent.make({
        smartBulb: input.source,
        eventCode: "DEFECTIVE",
```

```
      eventType: "Health",
      start: dates.at(i),
      end: dates.at(i+1)
    });
   }
  }
}

@DFE(period='Hour', interval='Hour', metric='HasEverFailed')
type SmartBulbDefective mixes TSDataFlowEvent<SmartBulb>
```

Seed

UIConfig

```
{
   "id": "uiconfig",
   "applications": [{
   "id": "lightbulb"
   }],
   "environments": [{
   "id": "development",
   "session": {
   "cache": "LocalStorage"
   },
   "locales": [
   {
   "id": "en",
   "name": "English"
   }
   ],
   "site": {
   "renderLogo": false,
   "hideNavigation": true
   },
   "connection": {
   "cache": false
   },
   "designer": true,
   "c3Tools": true,
   "acl": false
   }]
```

```
  }
```

SimpleMetric

```
{
 "id": "Status_SmartBulb",
 "name": "Status",
 "description": "Status (1 or 0) indicating ON or OFF of the SmartBulb over time",
 "path": "bulbMeasurements",
 "expression": "max(max(normalized.data.status))",
 "srcType": "SmartBulb"
}

{
 "id" : "RiskScore_SmartBulb",
 "name" : "RiskScore",
 "description": "Percentage value indicating risk of SmartBulb failure",
 "srcType" : "SmartBulb",
 "expression" : "identity(currentPrediction.prediction/365)"
}

{
 "id": "PowerGridStatus_SmartBulb",
 "name": "PowerGridStatus",
 "description": "Status (1 or 0) indicating ON or OFF of the PowerGrid over time",
 "srcType": "SmartBulb",
 "path": "fixtureHistory.to.apartment.building",
 "tsDecl": {
  "data": "gridStatusSet",
  "value": "value",
  "treatment": "PREVIOUS",
  "start": "timestamp"
 }
}

{
 "id": "PowerGridStatus_Building",
 "name": "PowerGridStatus",
 "description": "Status (1 or 0) indicating ON or OFF of the PowerGrid over time",
 "srcType" : "Building",
 "tsDecl": {
  "data": "gridStatusSet",
  "value": "value",
  "treatment": "PREVIOUS",
```

```
    "start": "timestamp"
  }
}

{
  "id": "AverageVoltage_SmartBulb",
  "name": "AverageVoltage",
  "description": "Average voltage over time of the SmartBulb",
  "path": "bulbMeasurements",
  "expression": "avg(avg(normalized.data.voltage))",
  "srcType": "SmartBulb"
}

{
  "id": "AverageTemperature_SmartBulb",
  "name": "AverageTemperature",
  "description": "Average temperature over time of the SmartBulb",
  "path": "bulbMeasurements",
  "expression": "avg(avg(normalized.data.temperature))",
  "srcType": "SmartBulb"
}

{
  "id": "AverageLumens_SmartBulb",
  "name": "AverageLumens",
  "description": "Average lumens over time of the SmartBulb",
  "path": "bulbMeasurements",
  "expression": "avg(avg(normalized.data.lumens))",
  "srcType": "SmartBulb"
}

{
  "id": "AveragePower_SmartBulb",
  "name": "AveragePower",
  "description": "Average power over time of the SmartBulb",
  "path": "bulbMeasurements",
  "expression": "avg(avg(normalized.data.power))",
  "srcType": "SmartBulb"
}
```

Role

```
{
    "id": "GeneralAnalystRole",
```

```
  "name": "GeneralAnalystRole",
  "permissions": [
    "allow:*:read:"
  ]
 }

{
  "id": "ExportDataPullRole",
  "name": "ExportDataPullRole",
  "permissions": [
    "deny:Export::remove",
    "deny:Export::removeAll",
    "allow:Export::*",
    "allow:BatchExportSpec::make",
    "allow:OAuth::token",
    "allow:S3File::*",
    "allow:File::writeEncodedStream"
  ]
 }

{
  "id": "BuildingFetchOnlyAnalystRole",
  "name": "BuildingFetchOnlyAnalystRole",
  "permissions": [
    "allow:Building::fetch"
  ]
 }

{
  "id": "Building1AnalystRole",
  "name": "Building1AnalystRole",
  "permissions": [
    "allow:*:read:",
    "deny:SmartBulbMeasurement::fetch"
  ],
  "actionConditions": [
    "Building:read::(intersects(id,['bld1']))",
    "Apartment:read::(intersects(building.id, ['bld1']))",
    "Fixture:read::(intersects(apartment.building.id,['bld1']))",

    "SmartBulbToFixtureRelation:read::(intersects(to.apartment.building.id,['bld1']))",

    "SmartBulb:read::(intersects(fixtureHistory.to.apartment.building.id,['bld1']))",
```

"SmartBulbMeasurementSeries:read::(intersects(smartBulb.fixtureHistory.to.apartment
.building.id,['bld1']))"
  ]
 }

FileSourceCollection

{
  "id": "CanonicalSmartBulbToFixtureRelation",
  "name": "CanonicalSmartBulbToFixtureRelation",
  "source": {
    "typeName": "CanonicalSmartBulbToFixtureRelation"
  },
  "sourceSystem": {
      "id": "Legacy"
  }
 }

{
  "id": "CanonicalSmartBulbPrediction",
  "name": "CanonicalSmartBulbPrediction",
  "source": {
    "typeName": "CanonicalSmartBulbPrediction"
  },
  "sourceSystem": {
      "id": "Legacy"
  }
 }

{
  "id": "CanonicalSmartBulbMeasurement",
  "name": "CanonicalSmartBulbMeasurement",
  "source": {
    "typeName": "CanonicalSmartBulbMeasurement"
  },
  "sourceSystem": {
      "id": "Legacy"
  }
 }

{
  "id": "CanonicalSmartBulbEvent",


```
  "name": "CanonicalSmartBulbEvent",
  "source": {
    "typeName": "CanonicalSmartBulbEvent"
  },
  "sourceSystem": {
      "id": "Legacy"
  }
 }


{
  "id": "CanonicalSmartBulb",
  "name": "CanonicalSmartBulb",
  "source": {
    "typeName": "CanonicalSmartBulb"
  },
  "sourceSystem": {
      "id": "Legacy"
  }
 }


{
  "id": "CanonicalPowerGridStatus",
  "name": "CanonicalPowerGridStatus",
  "source": {
    "typeName": "CanonicalPowerGridStatus"
  },
  "sourceSystem": {
      "id": "Legacy"
  }
 }

{
  "id": "CanonicalFixture",
  "name": "CanonicalFixture",
  "source": {
    "typeName": "CanonicalFixture"
  },
  "sourceSystem": {
      "id": "Legacy"
  }
 }
```

CompoundMetric

```
{
   "name": "WillFailNextMonth",
   "id": "WillFailNextMonth",
   "description" : "Indicator (1 or 0) if the lightbulb died within the next 30 days",
   "expression": "window('MAX', IsDefective, 0, 30) > 0"
}

{
   "name": "SwitchCount",
   "id": "SwitchCount",
   "description": "The number of times a lightbulb is switched on or off beginning at the
'start' given",
   "expression": "sum(eval('HOUR', abs(rollingDiff(Status))))"
}

{
   "name": "DurationOnInHours",
   "id": "DurationOnInHours",
   "description": "The total amount of time a lightbulb is switched on up to the interval",
   "expression": "rolling('SUM',sum(eval('HOUR', Status)))"
}

{
   "name": "IsDefective",
   "id": "IsDefective",
   "description": "Indicator (1 or 0) if the lightbulb died",
   "expression": "or(eval('HOUR', AverageLumens == 0 && Status == 1 &&
PowerGridStatus == 1)) ? 1 : 0"
}

{
   "name": "SwitchCountPreviousWeek",
   "id": "SwitchCountPreviousWeek",
   "description": "The total number of times a lightbulb is switched on or off up in the
previous week of the interval",
   "expression": "window('SUM', SwitchCount, -7, 7)"
}

{
   "name": "HasEverFailed",
   "id": "HasEverFailed",
   "description": "Indicates (1 or 0) if the SmartBulb has failed during evaluated interval",
```

```
  "expression": "rolling('SUM',IsDefective) ? 1 : 0"
}
```

Admin Group

```
{
  "id" : "BuildingFetchOnlyGroup",
  "name" : "BuildingFetchOnlyGroup",
  "roles" : [
    {
      "id": "BuildingFetchOnlyAnalystRole"
    },
    {
      "id": "ExportDataPullRole"
    }
  ]
 }

{
  "id" : "Building1AnalystGroup",
  "name" : "Building1AnalystGroup",
  "roles" : [
    {
      "id": "Building1AnalystRole"
    },
    {
      "id": "ExportDataPullRole"
    }
  ]
 }

{
  "id" : "GeneralAnalystGroup",
  "name" : "GeneralAnalystGroup",
  "roles" : [
    {
      "id": "GeneralAnalystRole"
    },
    {
      "id": "ExportDataPullRole"
    }
  ]
 }
```

```
{
  "name": "lightbulbTraining",
  "description": "Application Training",
  "author": "C3 IoT",
  "dependencies": ["standardDependencies", "machineLearning"]
}
```

AWS Device Predictive Maintenance Source Code

"FILENAME: C:\dev\c3\rawRepos\c3iot-web\angular.json"

```
{
  "$schema": "./node_modules/@angular/cli/lib/config/schema.json",
  "version": 1,
  "newProjectRoot": "projects",
  "projects": {
    "c3iot-web": {
      "root": "",
      "sourceRoot": "src",
      "projectType": "application",
      "prefix": "app",
      "schematics": {},
      "architect": {
        "build": {
          "builder": "@angular-devkit/build-angular:browser",
          "options": {
            "outputPath": "dist/c3iot-web",
            "index": "src/index.html",
            "main": "src/main.ts",
            "polyfills": "src/polyfills.ts",
            "tsConfig": "src/tsconfig.app.json",
            "assets": [
              "src/favicon.ico",
              "src/assets"
            ],
            "styles": [
              "src/c3iot-theme.scss",
              "src/styles.scss"
            ],
            "scripts": []
          },
          "configurations": {
            "production": {
              "fileReplacements": [
                {
                  "replace": "src/environments/environment.ts",
                  "with": "src/environments/environment.prod.ts"
                }
              ],
              "optimization": true,
              "outputHashing": "all",
              "sourceMap": false,
              "extractCss": true,
              "namedChunks": false,
              "aot": true,
              "extractLicenses": true,
              "vendorChunk": false,
              "buildOptimizer": true
            }
          }
```

```
    },
    "serve": {
      "builder": "@angular-devkit/build-angular:dev-server",
      "options": {
        "browserTarget": "c3iot-web:build"
      },
      "configurations": {
        "production": {
          "browserTarget": "c3iot-web:build:production"
        }
      }
    },
    "extract-i18n": {
      "builder": "@angular-devkit/build-angular:extract-i18n",
      "options": {
        "browserTarget": "c3iot-web:build"
      }
    },
    "test": {
      "builder": "@angular-devkit/build-angular:karma",
      "options": {
        "main": "src/test.ts",
        "polyfills": "src/polyfills.ts",
        "tsConfig": "src/tsconfig.spec.json",
        "karmaConfig": "src/karma.conf.js",
        "styles": [
          "src/styles.css"
        ],
        "scripts": [],
        "assets": [
          "src/favicon.ico",
          "src/assets"
        ]
      }
    },
    "lint": {
      "builder": "@angular-devkit/build-angular:tslint",
      "options": {
        "tsConfig": [
          "src/tsconfig.app.json",
          "src/tsconfig.spec.json"
        ],
        "exclude": [
          "**/node_modules/**"
        ]
      }
    }
  }
},
"c3iot-web-e2e": {
  "root": "e2e/",
  "projectType": "application",
  "architect": {
    "e2e": {
      "builder": "@angular-devkit/build-angular:protractor",
      "options": {
```

```
          "protractorConfig": "e2e/protractor.conf.js",
          "devServerTarget": "c3iot-web:serve"
        },
        "configurations": {
          "production": {
            "devServerTarget": "c3iot-web:serve:production"
          }
        }
      },
      "lint": {
        "builder": "@angular-devkit/build-angular:tslint",
        "options": {
          "tsConfig": "e2e/tsconfig.e2e.json",
          "exclude": [
            "**/node_modules/**"
          ]
        }
      }
    }
  },
  "defaultProject": "c3iot-web"
}
"FILENAME: C:\dev\c3\rawRepos\c3iot-web\package.json"

{
  "name": "c3iot-web",
  "version": "0.0.0",
  "scripts": {
    "ng": "ng",
    "start": "ng serve",
    "build": "ng build",
    "test": "ng test",
    "lint": "ng lint",
    "e2e": "ng e2e"
  },
  "private": true,
  "dependencies": {
    "@agm/core": "^1.0.0-beta.5",
    "@angular/animations": "^6.1.9",
    "@angular/cdk": "^6.4.7",
    "@angular/common": "^6.1.0",
    "@angular/compiler": "^6.1.0",
    "@angular/core": "^6.1.0",
    "@angular/flex-layout": "^6.0.0-beta.18",
    "@angular/forms": "^6.1.0",
    "@angular/http": "^6.1.0",
    "@angular/material": "^6.4.7",
    "@angular/platform-browser": "^6.1.0",
    "@angular/platform-browser-dynamic": "^6.1.0",
    "@angular/router": "^6.1.0",
    "core-js": "^2.5.4",
    "date-fns": "^1.30.1",
    "highcharts": "^6.2.0",
    "highcharts-angular": "^2.3.1",
    "rxjs": "~6.2.0",
```

```
    "rxjs-compat": "^6.3.3",
    "zone.js": "~0.8.26"
  },
  "devDependencies": {
    "@angular-devkit/build-angular": "~0.8.0",
    "@angular/cli": "~6.2.4",
    "@angular/compiler-cli": "^6.1.0",
    "@angular/language-service": "^6.1.0",
    "@types/date-fns": "^2.6.0",
    "@types/googlemaps": "^3.30.16",
    "@types/jasmine": "~2.8.8",
    "@types/jasminewd2": "~2.0.3",
    "@types/node": "~8.9.4",
    "codelyzer": "~4.3.0",
    "jasmine-core": "~2.99.1",
    "jasmine-spec-reporter": "~4.2.1",
    "karma": "~3.0.0",
    "karma-chrome-launcher": "~2.2.0",
    "karma-coverage-istanbul-reporter": "~2.0.1",
    "karma-jasmine": "~1.1.2",
    "karma-jasmine-html-reporter": "^0.2.2",
    "protractor": "~5.4.0",
    "ts-node": "~7.0.0",
    "tslint": "~5.11.0",
    "typescript": "~2.9.2"
  }
}

"FILENAME: C:\dev\c3\rawRepos\c3iot-web\tsconfig.json"

{
  "compileOnSave": false,
  "compilerOptions": {
    "baseUrl": "./",
    "outDir": "./dist/out-tsc",
    "sourceMap": true,
    "declaration": false,
    "moduleResolution": "node",
    "emitDecoratorMetadata": true,
    "experimentalDecorators": true,
    "target": "es5",
    "typeRoots": [
      "node_modules/@types"
    ],
    "lib": [
      "es2017",
      "dom"
    ]
  }
}

"FILENAME: C:\dev\c3\rawRepos\c3iot-web\tslint.json"

{
  "rulesDirectory": [
    "node_modules/codelyzer"
```

```
  ],
  "rules": {
    "arrow-return-shorthand": true,
    "callable-types": true,
    "class-name": true,
    "comment-format": [
      true,
      "check-space"
    ],
    "curly": true,
    "deprecation": {
      "severity": "warn"
    },
    "eofline": true,
    "forin": true,
    "import-blacklist": [
      true,
      "rxjs/Rx"
    ],
    "import-spacing": true,
    "indent": [
      true,
      "spaces"
    ],
    "interface-over-type-literal": true,
    "label-position": true,
    "max-line-length": [
      true,
      140
    ],
    "member-access": false,
    "member-ordering": [
      true,
      {
        "order": [
          "static-field",
          "instance-field",
          "static-method",
          "instance-method"
        ]
      }
    ],
    "no-arg": true,
    "no-bitwise": true,
    "no-console": [
      true,
      "debug",
      "info",
      "time",
      "timeEnd",
      "trace"
    ],
    "no-construct": true,
    "no-debugger": true,
    "no-duplicate-super": true,
    "no-empty": false,
```

```
"no-empty-interface": true,
"no-eval": true,
"no-inferrable-types": [
  true,
  "ignore-params"
],
"no-misused-new": true,
"no-non-null-assertion": true,
"no-redundant-jsdoc": true,
"no-shadowed-variable": true,
"no-string-literal": false,
"no-string-throw": true,
"no-switch-case-fall-through": true,
"no-trailing-whitespace": true,
"no-unnecessary-initializer": true,
"no-unused-expression": true,
"no-use-before-declare": true,
"no-var-keyword": true,
"object-literal-sort-keys": false,
"one-line": [
  true,
  "check-open-brace",
  "check-catch",
  "check-else",
  "check-whitespace"
],
"prefer-const": true,
"quotemark": [
  true,
  "single"
],
"radix": true,
"semicolon": [
  true,
  "always"
],
"triple-equals": [
  true,
  "allow-null-check"
],
"typedef-whitespace": [
  true,
  {
    "call-signature": "nospace",
    "index-signature": "nospace",
    "parameter": "nospace",
    "property-declaration": "nospace",
    "variable-declaration": "nospace"
  }
],
"unified-signatures": true,
"variable-name": false,
"whitespace": [
  true,
  "check-branch",
  "check-decl",
```

```
      "check-operator",
      "check-separator",
      "check-type"
    ],
    "no-output-on-prefix": true,
    "use-input-property-decorator": true,
    "use-output-property-decorator": true,
    "use-host-property-decorator": true,
    "no-input-rename": true,
    "no-output-rename": true,
    "use-life-cycle-interface": true,
    "use-pipe-transform-interface": true,
    "component-class-suffix": true,
    "directive-class-suffix": true
  }
}

"FILENAME: C:\dev\c3\rawRepos\c3iot-web\cloudformation-templates\web-hosting-infra.template"

{
    "AWSTemplateFormatVersion": "2010-09-09",
    "Description": "Creates the infrastructure needed to host the web application",
    "Parameters": {

    },

    "Resources": {
        "LightbulbDashboardS3Bucket": {
            "Type": "AWS::S3::Bucket",
            "DeletionPolicy" : "Retain",
            "Properties": {
                "BucketName": "lightbulb-dashboard",
                "AccessControl": "BucketOwnerFullControl"
            }
        },
        "OriginAccessId": {
            "Type": "AWS::CloudFront::CloudFrontOriginAccessIdentity",
            "Properties": {
                "CloudFrontOriginAccessIdentityConfig": {
                    "Comment": "Origin Access ID used to give CloudFront access to the S3 bucket"
                }
            }
        },
        "BucketPolicy": {
            "Type": "AWS::S3::BucketPolicy",
            "Properties": {
                "PolicyDocument": {
                    "Version": "2008-10-17",
                    "Id": "PolicyForCloudFrontPrivateContent",
                    "Statement": [
                        {
                            "Sid": "1",
                            "Effect": "Allow",
                            "Principal": {
                                "AWS": {
```

```
                        "Fn::Sub": "arn:aws:iam::cloudfront:user/CloudFront Origin Access Identity
${OriginAccessId}"
                      }
                    },
                    "Action": "s3:GetObject",
                    "Resource": "arn:aws:s3:::lightbulb-dashboard/*"
                  }
                ]
              },
              "Bucket": {
                "Ref": "LightbulbDashboardS3Bucket"
              }
            }
          },
          "LightbulbDashboardDistribution" : {
            "Type" : "AWS::CloudFront::Distribution",
            "Properties" : {
              "DistributionConfig" : {
                "Origins" : [ {
                  "DomainName" : {
                    "Fn::GetAtt": [
                     "LightbulbDashboardS3Bucket",
                     "DomainName"
                   ]
                  },
                  "Id" : { "Ref": "LightbulbDashboardS3Bucket" },
                  "S3OriginConfig" : {
                    "OriginAccessIdentity" : {
                      "Fn::Sub": "origin-access-identity/cloudfront/${OriginAccessId}"
                    }
                  }
                }],
                "Enabled" : "true",
                "Comment" : "",
                "DefaultRootObject" : "index.html",
                "DefaultCacheBehavior" : {
                  "AllowedMethods" : [ "GET", "HEAD" ],
                  "TargetOriginId" : { "Ref": "LightbulbDashboardS3Bucket" },
                  "ForwardedValues" : {
                    "QueryString" : "false",
                    "Cookies" : { "Forward" : "none" }
                  },
                  "ViewerProtocolPolicy" : "allow-all"
                },
                "PriceClass" : "PriceClass_100",
                "Restrictions" : {
                  "GeoRestriction" : {
                    "RestrictionType" : "none"
                  }
                },
                "ViewerCertificate" : { "CloudFrontDefaultCertificate" : "true" }
              }
            }
          }
        },
```

```
    "Outputs" : {
        "LightbulbDashboardS3BucketName":{
            "Description": "Lightbulb Dashboard S3 Bucket Full Name",
            "Value" : {"Ref" : "LightbulbDashboardS3Bucket"}
        }

    }
}
"FILENAME: C:\dev\c3\rawRepos\c3iot-web\e2e\protractor.conf.js"

// Protractor configuration file, see link for more information
// https://github.com/angular/protractor/blob/master/lib/config.ts

const { SpecReporter } = require('jasmine-spec-reporter');

exports.config = {
  allScriptsTimeout: 11000,
  specs: [
    './src/**/*.e2e-spec.ts'
  ],
  capabilities: {
    'browserName': 'chrome'
  },
  directConnect: true,
  baseUrl: 'http://localhost:4200/',
  framework: 'jasmine',
  jasmineNodeOpts: {
    showColors: true,
    defaultTimeoutInterval: 30000,
    print: function() {}
  },
  onPrepare() {
    require('ts-node').register({
      project: require('path').join(__dirname, './tsconfig.e2e.json')
    });
    jasmine.getEnv().addReporter(new SpecReporter({ spec: { displayStacktrace: true } }));
  }
};
"FILENAME: C:\dev\c3\rawRepos\c3iot-web\e2e\tsconfig.e2e.json"

{
  "extends": "../tsconfig.json",
  "compilerOptions": {
    "outDir": "../out-tsc/app",
    "module": "commonjs",
    "target": "es5",
    "types": [
      "jasmine",
      "jasminewd2",
      "node"
    ]
  }
}
"FILENAME: C:\dev\c3\rawRepos\c3iot-web\e2e\src\app.e2e-spec.ts"

import { AppPage } from './app.po';
```

```
describe('workspace-project App', () => {
  let page: AppPage;

  beforeEach(() => {
    page = new AppPage();
  });

  it('should display welcome message', () => {
    page.navigateTo();
    expect(page.getParagraphText()).toEqual('Welcome to c3iot-web!');
  });
});

"FILENAME: C:\dev\c3\rawRepos\c3iot-web\e2e\src\app.po.ts"

import { browser, by, element } from 'protractor';

export class AppPage {
  navigateTo() {
    return browser.get('/');
  }

  getParagraphText() {
    return element(by.css('app-root h1')).getText();
  }
}

"FILENAME: C:\dev\c3\rawRepos\c3iot-web\src\c3iot-theme.scss"

@import '~@angular/material/theming';
// Plus imports for other components in your app.

// Include the common styles for Angular Material. We include this here so that you only
// have to load a single css file for Angular Material in your app.
// Be sure that you only ever include this mixin once!
@include mat-core();

$custom-typography: mat-typography-config(
    $font-family: 'gotham-light'
);
@include angular-material-typography($custom-typography);
"FILENAME: C:\dev\c3\rawRepos\c3iot-web\src\index.html"

<!doctype html>
<html lang="en">
<head>
  <meta charset="utf-8">
  <meta http-equiv="cache-control" content="no-cache, must-revalidate, post-check=0, pre-check=0">
  <meta http-equiv="expires" content="0">
  <meta http-equiv="pragma" content="no-cache">
  <title>Lightbulb Dashboard</title>
  <base href="/">

  <meta name="viewport" content="width=device-width, initial-scale=1">
  <link rel="icon" type="image/x-icon" href="favicon.ico">
```

```
  <link href="https://fonts.googleapis.com/icon?family=Material+Icons" rel="stylesheet">
</head>
<body>
  <app-root></app-root>
</body>
</html>

"FILENAME: C:\dev\c3\rawRepos\c3iot-web\src\karma.conf.js"

// Karma configuration file, see link for more information
// https://karma-runner.github.io/1.0/config/configuration-file.html

module.exports = function (config) {
  config.set({
    basePath: '',
    frameworks: ['jasmine', '@angular-devkit/build-angular'],
    plugins: [
      require('karma-jasmine'),
      require('karma-chrome-launcher'),
      require('karma-jasmine-html-reporter'),
      require('karma-coverage-istanbul-reporter'),
      require('@angular-devkit/build-angular/plugins/karma')
    ],
    client: {
      clearContext: false // leave Jasmine Spec Runner output visible in browser
    },
    coverageIstanbulReporter: {
      dir: require('path').join(__dirname, '../coverage'),
      reports: ['html', 'lcovonly'],
      fixWebpackSourcePaths: true
    },
    reporters: ['progress', 'kjhtml'],
    port: 9876,
    colors: true,
    logLevel: config.LOG_INFO,
    autoWatch: true,
    browsers: ['Chrome'],
    singleRun: false
  });
};
"FILENAME: C:\dev\c3\rawRepos\c3iot-web\src\main.ts"

import { enableProdMode } from '@angular/core';
import { platformBrowserDynamic } from '@angular/platform-browser-dynamic';

import { AppModule } from './app/app.module';
import { environment } from './environments/environment';

if (environment.production) {
  enableProdMode();
}

platformBrowserDynamic().bootstrapModule(AppModule)
  .catch(err => console.error(err));
```

```
"FILENAME: C:\dev\c3\rawRepos\c3iot-web\src\polyfills.ts"

/**
 * This file includes polyfills needed by Angular and is loaded before the app.
 * You can add your own extra polyfills to this file.
 *
 * This file is divided into 2 sections:
 *   1. Browser polyfills. These are applied before loading ZoneJS and are sorted by browsers.
 *   2. Application imports. Files imported after ZoneJS that should be loaded before your main
 *      file.
 *
 * The current setup is for so-called "evergreen" browsers; the last versions of browsers that
 * automatically update themselves. This includes Safari >= 10, Chrome >= 55 (including Opera),
 * Edge >= 13 on the desktop, and iOS 10 and Chrome on mobile.
 *
 * Learn more in https://angular.io/docs/ts/latest/guide/browser-support.html
 */

/***************************************************************************************************
 * BROWSER POLYFILLS
 */

/** IE9, IE10 and IE11 requires all of the following polyfills. **/
// import 'core-js/es6/symbol';
// import 'core-js/es6/object';
// import 'core-js/es6/function';
// import 'core-js/es6/parse-int';
// import 'core-js/es6/parse-float';
// import 'core-js/es6/number';
// import 'core-js/es6/math';
// import 'core-js/es6/string';
// import 'core-js/es6/date';
// import 'core-js/es6/array';
// import 'core-js/es6/regexp';
// import 'core-js/es6/map';
// import 'core-js/es6/weak-map';
// import 'core-js/es6/set';

/** IE10 and IE11 requires the following for NgClass support on SVG elements */
// import 'classlist.js';  // Run `npm install --save classlist.js`.

/** IE10 and IE11 requires the following for the Reflect API. */
// import 'core-js/es6/reflect';


/** Evergreen browsers require these. **/
// Used for reflect-metadata in JIT. If you use AOT (and only Angular decorators), you can remove.
import 'core-js/es7/reflect';


/**
 * Web Animations `@angular/platform-browser/animations`
 * Only required if AnimationBuilder is used within the application and using IE/Edge or Safari.
 * Standard animation support in Angular DOES NOT require any polyfills (as of Angular 6.0).
 **/
```

```
// import 'web-animations-js';  // Run `npm install --save web-animations-js`.

/**
 * By default, zone.js will patch all possible macroTask and DomEvents
 * user can disable parts of macroTask/DomEvents patch by setting following flags
 */

 // (window as any).__Zone_disable_requestAnimationFrame = true; // disable patch
requestAnimationFrame
 // (window as any).__Zone_disable_on_property = true; // disable patch onProperty such as onclick
 // (window as any).__zone_symbol__BLACK_LISTED_EVENTS = ['scroll', 'mousemove']; // disable
patch specified eventNames

 /*
 * in IE/Edge developer tools, the addEventListener will also be wrapped by zone.js
 * with the following flag, it will bypass `zone.js` patch for IE/Edge
 */
// (window as any).__Zone_enable_cross_context_check = true;

/***************************************************************************************************
 * Zone JS is required by default for Angular itself.
 */
import 'zone.js/dist/zone';  // Included with Angular CLI.



/***************************************************************************************************
 * APPLICATION IMPORTS
 */

"FILENAME: C:\dev\c3\rawRepos\c3iot-web\src\styles.scss"

/* You can add global styles to this file, and also import other style files */
@import "../node_modules/@angular/material/prebuilt-themes/deeppurple-amber.css";
@import "app/styles/_colors";
@import "app/styles/_styles";
@import "app/styles/material-icons";
"FILENAME: C:\dev\c3\rawRepos\c3iot-web\src\test.ts"

// This file is required by karma.conf.js and loads recursively all the .spec and framework files

import 'zone.js/dist/zone-testing';
import { getTestBed } from '@angular/core/testing';
import {
  BrowserDynamicTestingModule,
  platformBrowserDynamicTesting
} from '@angular/platform-browser-dynamic/testing';

declare const require: any;

// First, initialize the Angular testing environment.
getTestBed().initTestEnvironment(
  BrowserDynamicTestingModule,
  platformBrowserDynamicTesting()
```

```
);
// Then we find all the tests.
const context = require.context('./', true, /\.spec\.ts$/);
// And load the modules.
context.keys().map(context);
```

"FILENAME: C:\dev\c3\rawRepos\c3iot-web\src\tsconfig.app.json"

```
{
  "extends": "../tsconfig.json",
  "compilerOptions": {
    "outDir": "../out-tsc/app",
    "types": ["googlemaps"]
  },
  "exclude": [
    "test.ts",
    "**/*.spec.ts"
  ]
}
```

"FILENAME: C:\dev\c3\rawRepos\c3iot-web\src\tsconfig.spec.json"

```
{
  "extends": "../tsconfig.json",
  "compilerOptions": {
    "outDir": "../out-tsc/spec",
    "types": [
      "jasmine",
      "node",
      "googlemaps"
    ]
  },
  "files": [
    "test.ts",
    "polyfills.ts"
  ],
  "include": [
    "**/*.spec.ts",
    "**/*.d.ts"
  ]
}
```

"FILENAME: C:\dev\c3\rawRepos\c3iot-web\src\tslint.json"

```
{
    "extends": "../tslint.json",
    "rules": {
        "directive-selector": [
            true,
            "attribute",
            "app",
            "camelCase"
        ],
        "component-selector": [
            true,
            "element",
```

```
          "app",
          "kebab-case"
        ]
    }
}
```

"FILENAME: C:\dev\c3\rawRepos\c3iot-web\src\app\angular-material.module.ts"

```
import { NgModule } from '@angular/core';
import { MatAutocompleteModule,  MatButtonModule, MatCheckboxModule, MatToolbarModule,
MatDatepickerModule,
  MatFormFieldModule, MatInputModule, MatSelectModule, MatGridListModule, MatTableModule,
  MatRadioModule, MatNativeDateModule, MatPaginatorModule, MatSnackBarModule,
MatDialogModule, MatProgressBarModule,
  MatProgressSpinnerModule, MatIconModule, MatButtonToggleModule, MatSidenavModule,
MatListModule,
  MatExpansionModule, MatSortModule, MatSlideToggleModule, MatCardModule, MatTabsModule,
MatTooltipModule,
  MatDividerModule, MatStepperModule, MatChipsModule} from '@angular/material';

@NgModule({
  imports: [
    MatAutocompleteModule,
    MatButtonModule,
    MatCheckboxModule,
    MatToolbarModule,
    MatDatepickerModule,
    MatFormFieldModule,
    MatInputModule,
    MatSelectModule,
    MatGridListModule,
    MatTableModule,
    MatRadioModule,
    MatNativeDateModule,
    MatPaginatorModule,
    MatSnackBarModule,
    MatDialogModule,
    MatProgressBarModule,
    MatProgressSpinnerModule,
    MatIconModule,
    MatButtonToggleModule,
    MatSidenavModule,
    MatListModule,
    MatExpansionModule,
    MatCardModule,
    MatSortModule,
    MatSlideToggleModule,
    MatTabsModule,
    MatTooltipModule,
    MatDividerModule,
    MatStepperModule,
    MatChipsModule
  ],
  exports: [
    MatAutocompleteModule,
    MatButtonModule,
```

```
    MatCheckboxModule,
    MatToolbarModule,
    MatDatepickerModule,
    MatFormFieldModule,
    MatInputModule,
    MatSelectModule,
    MatGridListModule,
    MatTableModule,
    MatRadioModule,
    MatPaginatorModule,
    MatSnackBarModule,
    MatDialogModule,
    MatProgressBarModule,
    MatProgressSpinnerModule,
    MatIconModule,
    MatButtonToggleModule,
    MatSidenavModule,
    MatListModule,
    MatExpansionModule,
    MatCardModule,
    MatSortModule,
    MatSlideToggleModule,
    MatTabsModule,
    MatTooltipModule,
    MatDividerModule,
    MatStepperModule,
    MatChipsModule
  ]
})
export class AngularMaterialModule { }

"FILENAME: C:\dev\c3\rawRepos\c3iot-web\src\app\app-dependencies.ts"

import { ComponentsModule } from './components/components.module';
import { ServicesModule } from './services/services.module';
import { RepositoriesModule } from './repositories/repositories.module';
import { RxjsModule } from './rxjs.module';


export const AppDependencies = {

    declarations: [],
    imports: [
        ServicesModule,
        RepositoriesModule,
        ComponentsModule,
        RxjsModule
    ],
    providers: [

    ]
};

"FILENAME: C:\dev\c3\rawRepos\c3iot-web\src\app\app-routing.module.ts"

import { RouterModule, Routes } from '@angular/router';
```

```
import { NgModule } from '@angular/core';
import { DashboardComponent } from './components/dashboard/dashboard.component';
import { BulbDetailComponent } from './components/bulb-detail/bulb-detail.component';

const appRoutes: Routes = [
  { path: 'bulb-detail/:id', component:BulbDetailComponent},
  { path: '', component: DashboardComponent }
];

@NgModule({
  imports: [
    RouterModule.forRoot(
      appRoutes,
      {
        enableTracing: false // <-- debugging purposes only
      }
    )
  ],
  exports: [
    RouterModule
  ]
})
export class AppRoutingModule { }

// ng test --single-run false
// test components, the dto inputs, and methods
"FILENAME: C:\dev\c3\rawRepos\c3iot-web\src\app\app.component.html"

<app-navbar></app-navbar>
"FILENAME: C:\dev\c3\rawRepos\c3iot-web\src\app\app.component.scss"


"FILENAME: C:\dev\c3\rawRepos\c3iot-web\src\app\app.component.spec.ts"

import { TestBed, async } from '@angular/core/testing';
import { AppComponent } from './app.component';

describe('AppComponent', () => {
  beforeEach(async(() => {
    TestBed.configureTestingModule({
      declarations: [
        AppComponent
      ],
    }).compileComponents();
  }));

  it('should create the app', () => {
    const fixture = TestBed.createComponent(AppComponent);
    const app = fixture.debugElement.componentInstance;
    expect(app).toBeTruthy();
  });

  it(`should have as title 'c3iot-web'`, () => {
    const fixture = TestBed.createComponent(AppComponent);
    const app = fixture.debugElement.componentInstance;
    expect(app.title).toEqual('c3iot-web');
```

```
  });

  it('should render title in a h1 tag', () => {
    const fixture = TestBed.createComponent(AppComponent);
    fixture.detectChanges();
    const compiled = fixture.debugElement.nativeElement;
    expect(compiled.querySelector('h1').textContent).toContain('Welcome to c3iot-web!');
  });
});

"FILENAME: C:\dev\c3\rawRepos\c3iot-web\src\app\app.component.ts"

import { Component } from '@angular/core';

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.scss']
})
export class AppComponent {
  title = 'Lightbulb Dashboard';
}

"FILENAME: C:\dev\c3\rawRepos\c3iot-web\src\app\app.module.ts"

import { BrowserModule } from '@angular/platform-browser';
import { NgModule } from '@angular/core';
import { AppComponent } from './app.component';
import { AppDependencies } from './app-dependencies';
import { AppRoutingModule } from './app-routing.module';

@NgModule({
  declarations: AppDependencies.declarations,
  imports: [
    AppRoutingModule,AppDependencies.imports
  ],
  providers: AppDependencies.providers,
  bootstrap: [AppComponent]
})
export class AppModule { }

"FILENAME: C:\dev\c3\rawRepos\c3iot-web\src\app\rxjs.module.ts"

import { NgModule } from '@angular/core';
import { CommonModule } from '@angular/common';
import 'rxjs/add/operator/finally';
import 'rxjs/add/operator/do';
import 'rxjs/add/operator/catch';
import 'rxjs/add/observable/throw';
import 'rxjs/add/operator/takeUntil';
import 'rxjs/add/operator/mergeMap';
import 'rxjs/add/operator/map';
import 'rxjs/add/operator/share';
import 'rxjs/add/observable/of';
import 'rxjs/add/observable/forkJoin';
import 'rxjs/add/operator/filter';
```

```
import 'rxjs/Observable';
import 'rxjs/add/observable/empty';

@NgModule({
  imports: [
    CommonModule
  ],
  declarations: []
})
export class RxjsModule { }
"FILENAME: C:\dev\c3\rawRepos\c3iot-web\src\app\components\components.module.ts"

import { NgModule } from '@angular/core';
import { AppComponent } from '../app.component';
import { CommonModule } from '@angular/common';
import { AngularMaterialModule } from '../angular-material.module';
import { FormsModule, ReactiveFormsModule } from '@angular/forms';
import { FlexLayoutModule } from '@angular/flex-layout';
import { BrowserModule } from '@angular/platform-browser';
import { HttpClientModule } from '@angular/common/http';
import { BrowserAnimationsModule } from '@angular/platform-browser/animations';
import { AgmCoreModule } from '@agm/core';
import { HighchartsChartModule } from 'highcharts-angular';
import { AppNavbarComponent } from './app-navbar/app-navbar.component';
import { DashboardComponent } from './dashboard/dashboard.component';
import { SearchFormComponent } from './search-form/search-form.component';
import { BulbSearchFormComponent } from './bulb-search-form/bulb-search-form.component';
import { OverviewSummaryComponent } from './overview-summary/overview-summary.component';
import { RiskChartComponent } from './risk-chart/risk-chart.component';
import { StatusMapComponent } from './status-map/status-map.component';
import { DetailsTableComponent } from './details-table/details-table.component';
import { ViewHeaderComponent } from './view-header/view-header.component';
import { AppRoutingModule } from '../app-routing.module';
import { BulbDetailComponent } from './bulb-detail/bulb-detail.component';
import { BulbDetailsTableComponent } from './bulb-details-table/bulb-details-table.component';
import { BulbChartComponent } from './bulb-chart/bulb-chart.component';
import { BulbOverviewSummaryComponent } from './bulb-overview-summary/bulb-overview-
summary.component';

@NgModule({
   imports: [
      CommonModule,
      AppRoutingModule,
      AngularMaterialModule,
      FormsModule,
      FlexLayoutModule,
      BrowserModule,
      HttpClientModule,
      BrowserAnimationsModule,
      ReactiveFormsModule,
      HighchartsChartModule,
      AgmCoreModule.forRoot({
         apiKey: 'AIzaSyCcUaxZecW8I2EQACVP-WzRoktoOMahsdI'
      })
   ],
   declarations: [
```

```
        AppComponent,
        AppNavbarComponent,
        BulbChartComponent,
        BulbDetailComponent,
        BulbDetailsTableComponent,
        BulbOverviewSummaryComponent,
        BulbSearchFormComponent,
        DashboardComponent,
        SearchFormComponent,
        OverviewSummaryComponent,
        RiskChartComponent,
        StatusMapComponent,
        DetailsTableComponent,
        ViewHeaderComponent
    ]
})
export class ComponentsModule { }
```

"FILENAME: C:\dev\c3\rawRepos\c3iot-web\src\app\components\app-navbar\app-navbar.component.html"

```
<mat-sidenav-container class="lightbulb-sidenav">
    <mat-sidenav mode="side" opened>
        <a routerLink="/" routerLinkActive="active noUnder">
            <mat-icon class="lightbulb-sidenav__home">home</mat-icon>
            <div class="lightbulb-sidenav__title">Dashboard</div>
        </a>
    </mat-sidenav>
    <mat-sidenav-content>
        <router-outlet></router-outlet>
    </mat-sidenav-content>
</mat-sidenav-container>
```

"FILENAME: C:\dev\c3\rawRepos\c3iot-web\src\app\components\app-navbar\app-navbar.component.scss"

```
@import "../../styles/_colors";

.lightbulb-sidenav
{
    position: absolute;
    top: 0;
    bottom: 0;
    left: 0;
    right: 0;

    mat-sidenav {
        width: 60px;
        background-color: map-get($colors-secondary, dark-grey);
        padding-top: 20px;
    }

    mat-sidenav-content {
        background-color: map-get($colors-secondary, light-grey);
    }

    &__title {
```

```
        text-align: center;
        font-weight: 900;
        color: map-get($colors-primary, accent);
        font-size:9px;
    }

    &__home {
        color: map-get($colors-primary, accent);
        display:block;
        margin:auto;
    }

    .noUnder {
        text-decoration: none;
    }

    .search-container {
        width: 240px;
        margin: 0 0 0 20px;
    }

    .main-container {
        margin: 25px;
    }

    .dash-component {
        border: 1px solid rgba(0,0,0,.03);
        border-radius: 3px;
        box-shadow: 0 2px 2px rgba(0,0,0,.24), 0 0 2px rgba(0,0,0,.12);
    }
}
"FILENAME: C:\dev\c3\rawRepos\c3iot-web\src\app\components\app-navbar\app-navbar.component.ts"

import { Component } from '@angular/core';
import { MatSidenav } from '@angular/material';

@Component({
  selector: 'app-navbar',
  templateUrl: 'app-navbar.component.html',
  styleUrls: ['./app-navbar.component.scss']
})

export class AppNavbarComponent { }

"FILENAME: C:\dev\c3\rawRepos\c3iot-web\src\app\components\base\component.base.ts"

import { OnDestroy } from '@angular/core';
import { Subject } from 'rxjs/Subject';

export class ComponentBase implements OnDestroy {

  public ngUnsubscribe: Subject<any> = new Subject();

  constructor() {  }

  ngOnDestroy() {
```

```
    this.ngUnsubscribe.next();
    this.ngUnsubscribe.complete();
  }
}
"FILENAME: C:\dev\c3\rawRepos\c3iot-web\src\app\components\bulb-chart\bulb-chart.component.html"

<div fxLayout="column" fxFlexFill class="risk-chart__container">
    <div class="section-title">
        <div class="section-title__title-copy">
            <span>Lightbulb Measurements Chart</span>
        </div>
    </div>
    <div class="spinner-container" *ngIf="loading" fxLayoutAlign="center center">
        <mat-spinner [diameter]="30"></mat-spinner>
    </div>
    <div class="risk-chart" *ngIf="(showChart && !loading)" >
        <highcharts-chart
            [Highcharts]="Highcharts"

            [constructorType]="chartConstructor"
            [options]="chartOptions"
            [callbackFunction]="chartCallback"

            [(update)]="updateFlag"
            [oneToOne]="oneToOneFlag"
            [runOutsideAngular]="runOutsideAngularFlag"

            style="width: 100%; height: 450px; display: block;"
        ></highcharts-chart>
    </div>
</div>
"FILENAME: C:\dev\c3\rawRepos\c3iot-web\src\app\components\bulb-chart\bulb-chart.component.scss"

@import "../../styles/_colors";

.section-title {
    border-style: solid;
    border-color: map-get($colors-secondary, grey);
    border-width: 0 0 1px 0;

    &__title-copy {
        margin-block-start: 1.33em;
        margin-block-end: 1.33em;
        margin-inline-start: 20px;
        margin-inline-end: 0px;
        font-family: 'gotham-bold', Arial, sans-serif;
        font-weight: bold;
        font-size: 12px;
        line-height: 100%;
        color: map-get($colors-secondary, medium-dark-grey);

        &--lighter {
            color: map-get($colors-secondary, medium-light-grey);
        }
    }
}
```

```
.spinner-container {
   width: 100%;
   padding: 220px;
   background-color: map-get($colors-secondary, light-grey);
}

.risk-chart {
   margin: 20px 30px 0 10px;

   &__container {
      background-color: map-get($colors-secondary, white);
   }
}
"FILENAME: C:\dev\c3\rawRepos\c3iot-web\src\app\components\bulb-chart\bulb-chart.component.ts"

import { Component, OnInit } from '@angular/core';
import { ActivatedRoute } from '@angular/router';
import { SmartBulbMapDTO } from '../../dto/smart-bulb-map.dto';
import { ComponentBase } from '../base/component.base';
import { LightbulbDetailService } from '../../services/lightbulb-detail.service';
import * as Highcharts from 'highcharts';

@Component({
  selector: 'bulb-chart',
  templateUrl: 'bulb-chart.component.html',
  styleUrls: ['./bulb-chart.component.scss']
})

export class BulbChartComponent  extends ComponentBase implements OnInit {
  lightbulbDetailItems: SmartBulbMapDTO;
  currentLightbulbDetailItems: SmartBulbMapDTO = new SmartBulbMapDTO();
  histoData: Array<number>;
  showChart: boolean;
  smartBulbName: string = '';
  chartIntervalInHours: number;
  loading: boolean = true;

  Highcharts = Highcharts; // required
  chartConstructor = 'chart'; // optional string, defaults to 'chart'
  chartOptions = {
    plotOptions: {
      series: {
         marker: {
            enabled: false
         }
      },
      alignTicks: false
    },
    legend:{
      borderRadius: 5,
      padding: 9,
      backgroundColor: '#FFFFFF',
      floating: true,
      y:-8,
      shadow: true,
```

```
  align: 'center',
  verticalAlign: 'top',
  layout: 'horizontal'
},
title: '',
series:
[
  {
    visible:true,
    name:'Temperature',
    color: '#FFB428',
    data: [],
    pointStart: Date.UTC(2010, 0, 1),
    pointInterval: 3600 * 1000,
    yAxis:0
  },
  {
    visible:true,
    name:'Voltage',
    color: '#75B809',
    data: [],
    pointStart: Date.UTC(2010, 0, 1),
    pointInterval: 3600 * 1000,
    yAxis:1
  },
  {
    visible:true,
    name:'Power',
    color: '#4169B9',
    data: [],
    pointStart: Date.UTC(2010, 0, 1),
    pointInterval: 3600 * 1000,
    yAxis:2
  },
  {
    visible:true,
    name:'Lumens',
    color: '#00B4DC',
    data: [],
    pointStart: Date.UTC(2010, 0, 1),
    pointInterval: 3600 * 1000,
    yAxis:3
  },
  {
    visible:true,
    name:'Risk',
    color: '#C80000',
    data: [],
    pointStart: Date.UTC(2010, 0, 1),
    pointInterval: 3600 * 1000,
    yAxis:4
  }
],
xAxis: {
  type: 'datetime',
  title: {text:'Date',
```

```
    style: {'font-weight': 'bolder'}
  },
  dateTimeLabelFormats: {
     second: '%Y-%m-%d<br/>%H:%M:%S',
     minute: '%Y-%m-%d<br/>%H:%M',
     hour: '%Y-%m-%d<br/>%H:%M',
     day: '%Y<br/>%m-%d',
     week: '%Y<br/>%m-%d',
     month: '%Y-%m',
     year: '%Y'
  }
},
yAxis:
[
  {
    description:'Temperature',
    title: { text: 'Temperature (C)',
      style: {
        'font-weight': 'bolder',
        'color': '#FFB428'
      }
    },
    labels: {
      formatter: function() {
          return this.value +'°C';
      },
      style: {
        color: '#FFB428'
      }
    },
    opposite: true,
    showEmpty:false
  },
  {
    description:'Voltage',
    title: { text: 'Voltage (V)',
      style: {
        'font-weight': 'bolder',
        'color': '#75B809'
      }
    },
    labels: {
      formatter: function() {
          return this.value +' V';
      },
      style: {
        color: '#75B809'
      }
    },
    opposite: false,
    showEmpty:false
  },
  {
    description:'Power',
    title: { text: 'Power (W)',
      style: {
```

```
          'font-weight': 'bolder',
          'color': '#4169B9'
        }
      },
      labels: {
        formatter: function() {
            return this.value +' W';
        },
        style: {
          color: '#4169B9'
        }
      },
      opposite: true,
      showEmpty:false
    },
    {
      description:'Lumens',
      title: { text: 'Lumens',
        style: {
          'font-weight': 'bolder',
          'color': '#00B4DC'
        }
      },
      labels: {
        formatter: function() {
            return this.value;
        },
        style: {
          color: '#00B4DC'
        }
      },
      opposite: true,
      showEmpty:false
    },
    {
      description:'Risk',
      title: { text: 'Risk Score %',
        style: {
          'font-weight': 'bolder',
          'color': '#C80000'
        }
      },
      labels: {
        formatter: function() {
            return this.value +'%';
        },
        style: {
          color: '#C80000'
        }
      },
      showEmpty:false
    }
  ]
}; // required
updateFlag:boolean = false; // optional boolean
oneToOneFlag = true; // optional boolean, defaults to false
```

```
runOutsideAngular = false; // optional boolean, defaults to false


constructor(private route: ActivatedRoute, private lightbulbDetailService: LightbulbDetailService) {
  super();
}

ngOnInit() {
  this.route.params.subscribe(params => {
    this.smartBulbName = params['id'];
    this.lightbulbDetailService.currentChartIntervalValue.subscribe(
      (result) => {
        this.chartIntervalInHours = result;
      }
    )
  });

  this.lightbulbDetailService.currentLoadingValue.subscribe(
    (result) => {
      this.loading = result;
    }
  )

  setTimeout(() => {this.getlightbulbDetailItems()}, 0);
}

getlightbulbDetailItems() {
  this.lightbulbDetailService.currentValue.subscribe(
    (result) => {
      if (!(Object.keys(result).length === 0)) {
        this.lightbulbDetailItems = result.map;
        this.reverseArrays();
        this.spreadArray();
        this.getHistoData();
      } else {
        this.clearData();
      }
    }
  )
}

clearData() {
  this.chartOptions.series[0].data=null;
  this.chartOptions.series[1].data=null;
  this.chartOptions.series[2].data=null;
  this.chartOptions.series[3].data=null;
  this.chartOptions.series[4].data=null;

  this.showChart=true;
  this.updateFlag = true;
}

getHistoData() {
  this.getTimestamps();
  this.chartOptions.series[0].data=this.currentLightbulbDetailItems.temperature;
  this.chartOptions.series[1].data=this.currentLightbulbDetailItems.voltage;
```

```
  this.chartOptions.series[2].data=this.currentLightbulbDetailItems.power;
  this.chartOptions.series[3].data=this.currentLightbulbDetailItems.lumens;
  this.chartOptions.series[4].data=this.currentLightbulbDetailItems.riskscore;

  this.showChart=true;
  this.updateFlag = true;
 }

 getTimestamps(){
  let arr=this.lightbulbDetailItems.timestamp;
  var startDate = new Date(arr[0]);
  this.chartOptions.series.forEach(element => {
    element.pointStart = Number(startDate);
    element.pointInterval = this.chartIntervalInHours*3600000;
  });
 }

 reverseArrays(){
  this.lightbulbDetailItems.lumens=this.lightbulbDetailItems.lumens.reverse();
  this.lightbulbDetailItems.power=this.lightbulbDetailItems.power.reverse();
  this.lightbulbDetailItems.riskscore=this.lightbulbDetailItems.riskscore.reverse();
  this.lightbulbDetailItems.status=this.lightbulbDetailItems.status.reverse();
  this.lightbulbDetailItems.temperature=this.lightbulbDetailItems.temperature.reverse();
  this.lightbulbDetailItems.timestamp=this.lightbulbDetailItems.timestamp.reverse();
  this.lightbulbDetailItems.voltage=this.lightbulbDetailItems.voltage.reverse();
 }

 spreadArray(){
  let lumensArr=[];
  let powerArr=[];
  let riskArr=[];
  let statusArr=[];
  let tempArr=[];
  let timeArr=[];
  let volArr=[];

  const intervalInHours = this.chartIntervalInHours;
  for(let i=0;i<this.lightbulbDetailItems.timestamp.length;i++){
    if(i%intervalInHours==0){
      lumensArr.push(this.lightbulbDetailItems.lumens[i]);
      powerArr.push(this.lightbulbDetailItems.power[i]);
      riskArr.push(this.lightbulbDetailItems.riskscore[i]);
      statusArr.push(this.lightbulbDetailItems.status[i]);
      tempArr.push(this.lightbulbDetailItems.temperature[i]);
      timeArr.push(this.lightbulbDetailItems.timestamp[i]);
      volArr.push(this.lightbulbDetailItems.voltage[i]);
    }
  }

  this.currentLightbulbDetailItems.lumens=lumensArr;
  this.currentLightbulbDetailItems.power=powerArr;
  this.currentLightbulbDetailItems.riskscore=riskArr;
  this.currentLightbulbDetailItems.status=statusArr;
  this.currentLightbulbDetailItems.temperature=tempArr;
  this.currentLightbulbDetailItems.timestamp=timeArr;
  this.currentLightbulbDetailItems.voltage=volArr;
```

```
  }
}
"FILENAME: C:\dev\c3\rawRepos\c3iot-web\src\app\components\bulb-detail\bulb-detail.component.html"

<div fxLayout="column">
   <view-header></view-header>
</div>
<div fxLayout="row">
  <div class="search-container">
       <bulb-search-form></bulb-search-form>
  </div>
  <div fxLayout="column" fxLayoutGap="20px" class="main-container" fxFlex>
    <bulb-overview-summary class="dash-component"></bulb-overview-summary>
    <bulb-chart class="dash-component"></bulb-chart>
    <bulb-details-table class="dash-component"></bulb-details-table>
  </div>
</div>


"FILENAME: C:\dev\c3\rawRepos\c3iot-web\src\app\components\bulb-detail\bulb-detail.component.scss"

@import "../../styles/_colors";

.dash-component {
   border: 1px solid rgba(0,0,0,.03);
   border-radius: 3px;
   box-shadow: 0 2px 2px rgba(0,0,0,.24), 0 0 2px rgba(0,0,0,.12);
}

.main-container {
   margin: 25px;
}

.search-container {
   width: 240px;
   margin: 0 0 0 20px;
}

"FILENAME: C:\dev\c3\rawRepos\c3iot-web\src\app\components\bulb-detail\bulb-detail.component.ts"

import { Component } from '@angular/core';
import { ActivatedRoute } from '@angular/router';
import { LightbulbDetailService } from 'src/app/services/lightbulb-detail.service';

@Component({
  selector: 'bulb-detail',
  templateUrl: 'bulb-detail.component.html',
  styleUrls: ['./bulb-detail.component.scss']
})

export class BulbDetailComponent {
  smartBulbName: string;

  constructor(private route: ActivatedRoute, private lightbulbDetailService: LightbulbDetailService) {}

  private ngOnInit() {
```

```
   this.route.params.subscribe(params => {
     this.smartBulbName = params['id'];
     console.log(this.smartBulbName);
     this.lightbulbDetailService.getAllLightbulbDetailItems(this.smartBulbName);
   });
  }

}
```

"FILENAME: C:\dev\c3\rawRepos\c3iot-web\src\app\components\bulb-details-table\bulb-details-table.component.html"

```
<div fxLayout="column" fxFlexFill class="details-table__container">
   <div class="section-title">
      <div class="section-title__title-copy">
         <h4>Lightbulb Measurements Table</h4>
      </div>
      <mat-form-field appearance="outline" class="matForm">
         <mat-icon  matSuffix>filter_list</mat-icon>
         <mat-label>Filter</mat-label>
         <input matInput (keyup)="applyFilter($event.target.value)" placeholder="Search...">
         <mat-icon  matPrefix>search</mat-icon>
      </mat-form-field>
   </div>
   <div class="details-table" fxFlex>
      <mat-table mat-table [dataSource]="dataSource" matSort>

         <!--- Note that these columns can be defined in any order.
               The actual rendered columns are set as a property on the row definition" -->

         <!-- Bulb ID Column -->
         <ng-container matColumnDef="timestamp">
            <mat-header-cell *matHeaderCellDef mat-sort-header="timestamp"> Timestamp </mat-header-cell>
            <mat-cell *matCellDef="let element"> {{element.timestamp}} </mat-cell>
         </ng-container>

         <!-- Risk Score Column -->
         <ng-container matColumnDef="riskscore">
            <mat-header-cell *matHeaderCellDef mat-sort-header="riskscore"> Risk Score % </mat-header-cell>
            <mat-cell *matCellDef="let element">
               <div class="risk-score-container">
                  <mat-progress-spinner [color]="getCircleColor(element)" [mode]="mode" [value]="element.riskscore" [diameter]="diameter"></mat-progress-spinner>
                  <div class="risk-score-number">{{element.riskscore}}</div>
               </div>
            </mat-cell>
         </ng-container>

         <!-- Status Column -->
         <ng-container matColumnDef="status">
            <mat-header-cell *matHeaderCellDef mat-sort-header="status"> Status </mat-header-cell>
            <mat-cell *matCellDef="let element">
```

```
            <mat-icon class="material-icons color_green" *ngIf="element.status ===
'On'">check_circle</mat-icon>
            <mat-icon class="material-icons color_red" *ngIf="element.status == 'Off'">cancel</mat-
icon>
            {{element.status}}
          </mat-cell>
        </ng-container>

        <!-- Lumens Column -->
        <ng-container matColumnDef="lumens">
          <mat-header-cell *matHeaderCellDef mat-sort-header="lumens"> Lumens </mat-header-cell>
          <mat-cell *matCellDef="let element"> {{element.lumens}} </mat-cell>
        </ng-container>

        <!-- Voltage Column -->
        <ng-container matColumnDef="voltage">
          <mat-header-cell *matHeaderCellDef mat-sort-header="voltage"> Voltage (V) </mat-header-
cell>
          <mat-cell *matCellDef="let element"> {{element.voltage}} </mat-cell>
        </ng-container>

        <!-- Power Column -->
        <ng-container matColumnDef="power">
          <mat-header-cell *matHeaderCellDef mat-sort-header="power"> Power (W) </mat-header-cell>
          <mat-cell *matCellDef="let element"> {{element.power}} </mat-cell>
        </ng-container>

        <!-- Temperature Column -->
        <ng-container matColumnDef="temperature">
          <mat-header-cell *matHeaderCellDef mat-sort-header="temperature"> Temperature (C) </mat-
header-cell>
          <mat-cell *matCellDef="let element"> {{element.temperature}} </mat-cell>
        </ng-container>

        <mat-header-row *matHeaderRowDef="displayedColumns"></mat-header-row>
        <mat-row *matRowDef="let row; columns: displayedColumns;"></mat-row>
      </mat-table>
      <mat-paginator [pageSizeOptions]="pageSizeOptions"></mat-paginator>
    </div>
  </div>
  "FILENAME: C:\dev\c3\rawRepos\c3iot-web\src\app\components\bulb-details-table\bulb-details-
table.component.scss"

  @import "../../styles/_colors";

  .section-title {
    border-style: solid;
    border-color: map-get($colors-secondary, grey);
    border-width: 0 0 1px 0;

    &__title-copy {
      margin: 0 0 0 20px;
    }
  }

  .details-table {
```

```
    &__container {
        background-color: map-get($colors-secondary, white);
    }
}

.risk-score-container {
  position: relative;
  text-align: center;
  margin-left: 10%;
}

.risk-score-number {
  position: absolute;
  top: 50%;
  left: 50%;
  transform: translate(-50%, -50%);
}

mat-icon {
  font-size: 18px;
  line-height: 130%;
}

.matForm{
  margin: 0 0 0 20px;
}

mat-table {
    text-align: left;
    max-height: 600px;
    overflow: auto;
    width: 100%;

    .mat-cell,
    .mat-header-cell {
      padding: 0 4px;
    }

    .mat-row,
    .mat-header-row {
      padding: 0 10px;
    }

    &.hover {
      mat-row {
        cursor: pointer;

        &.selected {
          background-color: rgb(181, 233, 244);
        }
      }
    }

    .mat-cell {
      text-overflow: ellipsis;
      white-space: nowrap;
```

```
    }

    .mat-header-row {
      position: sticky;
      position: -webkit-sticky;
      top: 0;
      background-color: inherit;
      z-index: 5;
    }
  }
"FILENAME: C:\dev\c3\rawRepos\c3iot-web\src\app\components\bulb-details-table\bulb-details-
table.component.ts"

import { Component, ViewChild } from '@angular/core';
import { ComponentBase } from '../base/component.base';
import { LightbulbDetailService } from '../../services/lightbulb-detail.service';
import {MatTableDataSource, MatPaginator, MatSort} from '@angular/material';
import { ActivatedRoute } from '@angular/router';
import { SmartBulbTableDTO } from 'src/app/dto/smart-bulb-table.dto';


@Component({
  selector: 'bulb-details-table',
  templateUrl: 'bulb-details-table.component.html',
  styleUrls: ['./bulb-details-table.component.scss']
})

export class BulbDetailsTableComponent extends ComponentBase {
  mode: string = 'determinate';
  diameter: number = 40;

  pageSizeOptions: number[] = [10, 25, 50, 100];
  smartBulbName: string = '';

  displayedColumns: string[] = ['timestamp', 'riskscore', 'status', 'lumens', 'voltage', 'power', 'temperature'];
  dataSource: MatTableDataSource<SmartBulbTableDTO>;

  @ViewChild(MatPaginator) paginator: MatPaginator;
  @ViewChild(MatSort) sort: MatSort;

  constructor(private route: ActivatedRoute, private lightbulbDetailService: LightbulbDetailService) {
    super();
    this.route.params.subscribe(params => {
      this.smartBulbName = params['id'];
      this.getLightbulbDetailItems();
    });
  }

  applyFilter(filterValue: string) {
    this.dataSource.filter = filterValue.trim().toLowerCase();

    if (this.dataSource.paginator) {
      this.dataSource.paginator.firstPage();
    }
  }
```

```
  getLightbulbDetailItems() {

    this.lightbulbDetailService.currentValue.subscribe(
      (result) => {
        if (!(Object.keys(result).length === 0)
            && result.table
            && result.table.length > 0
            && result.table[0].smartBulbName === this.smartBulbName) {
          if (result.table.length > 0) {
            this.dataSource=new MatTableDataSource(result.table);
            this.dataSource.paginator = this.paginator;
            this.dataSource.sort = this.sort;
          }
        }
      }
    )
  }

  getCircleColor(element){
    if (element.riskscore>=80){
      return 'warn';
    }
    else{
      return 'accent';
    }
  }
}
"FILENAME: C:\dev\c3\rawRepos\c3iot-web\src\app\components\bulb-overview-summary\bulb-
overview-summary.component.html"

<div fxLayout="column" fxFlexFill class="overview-data__container">
    <div class="section-title">
        <div class="section-title__title-copy">
            <h4>Key Metrics</h4>
        </div>
    </div>
    <div class="overview-data spinner-container" *ngIf="loading" fxLayoutAlign="center center">
            <mat-spinner [diameter]="30"></mat-spinner>
    </div>
    <div class="overview-data" fxLayout="row" *ngIf="!loading">
        <div class="overview-data__lightbulbs-at-risk overview-data__value-section" fxFlex="16.6"
fxLayout="column" fxAlign="start">
            <div class="overview-data__label">
                Risk Score
            </div>
            <div class="number-of-lightbulbs">{{bulbDetails?.riskscore}}%</div>
            <div fxLayout="row">
                <mat-icon class="material-icons {{arrowColor}}">{{arrowType}}</mat-icon>
                <div class="overview-data__value--small {{arrowColor}}">{{riskChange}}%</div>
            </div>
            <div>
                <div class="risk-chart" *ngIf="showChart">
                    <highcharts-chart
                        [Highcharts]="Highcharts"

                        [constructorType]="chartConstructor"
```

```
                [options]="chartOptions"
                [callbackFunction]="chartCallback"

                [(update)]="updateFlag"
                [oneToOne]="oneToOneFlag"
                [runOutsideAngular]="runOutsideAngularFlag"

                style="width: 100%; height: 50px; display: block; margin-left: -10px;"
              ></highcharts-chart>
            </div>
            <div class="overview-data__value--small">Last 24 hours</div>
          </div>
        </div>
        <div class="overview-data__total-lightbulbs overview-data__value-section" fxFlex="16.6"
fxLayout="column" fxAlign="start">
          <div class="overview-data__label">Current Status</div>
          <div fxLayout="row">
            <div class="overview-data__value">{{bulbDetails?.status}}</div>
            <div fxFlexAlign="center" class="overview-data__icon">
              <mat-icon class="material-icons color_green"
*ngIf="showBulbOnIcon(bulbDetails?.status)">check_circle</mat-icon>
              <mat-icon class="material-icons color_red"
*ngIf="!showBulbOnIcon(bulbDetails?.status)">cancel</mat-icon>
            </div>
          </div>
        </div>
        <div class="overview-data__total-lightbulbs overview-data__value-section" fxFlex="16.6"
fxLayout="column" fxAlign="start">
          <div class="overview-data__label">Lumens</div>
          <div class="overview-data__value">{{bulbDetails?.lumens}}</div>
        </div>
        <div class="overview-data__total-lightbulbs overview-data__value-section" fxFlex="16.6"
fxLayout="column" fxAlign="start">
          <div class="overview-data__label">Voltage</div>
          <div class="overview-data__value">{{bulbDetails?.voltage}}</div>
        </div>
        <div class="overview-data__total-lightbulbs overview-data__value-section" fxFlex="16.6"
fxLayout="column" fxAlign="start">
          <div class="overview-data__label">Power (W)</div>
          <div class="overview-data__value">{{bulbDetails?.power}}</div>
        </div>
        <div class="overview-data__failures-ytd overview-data__value-section" fxFlex="16.6"
fxLayout="column" fxAlign="start">
          <div class="overview-data__label">Temperature (C)</div>
          <div class="overview-data__value">{{bulbDetails?.temperature}}</div>
        </div>
      </div>
    </div>
    "FILENAME: C:\dev\c3\rawRepos\c3iot-web\src\app\components\bulb-overview-summary\bulb-
    overview-summary.component.scss"

    @import "../../styles/_colors";

    .section-title {
      border-style: solid;
      border-color: map-get($colors-secondary, grey);
```

```
    border-width: 0 0 1px 0;

    &__title-copy {
      margin: 0 0 0 20px;
    }
}

.spinner-container {
    width: 100%;
    padding: 60px;
    background-color: map-get($colors-secondary, light-grey);
}

.overview-data {
    mat-icon {
      line-height: 100%;
    }

    &__container {
      background-color: map-get($colors-secondary, white);
    }

    &__label {
      font-size: 10px;
      font-family: 'gotham-bold';
      text-transform: uppercase;
      color: map-get($colors-secondary, medium-light-grey);
    }

    &__icon {
      margin-left: 7px;
    }

    &__value {
      font-size: 42px;
      line-height: 100%;
      font-family: 'gotham-light';

      &--small {
        font-size: 10px;
        font-family: 'gotham-bold';
        line-height: 24px;
        color: map-get($colors-secondary, medium-light-grey);
        &.color_green {
          color: map-get($colors-primary, success);
        }
        &.color_red {
          color: map-get($colors-primary, danger);
        }
        &.color_yellow {
          color: map-get($colors-primary, warn);
        }
      }
    }

    &__value-section {
```

```
      padding: 10px 0 10px 15px;
    }

    &__lightbulbs-at-risk {
      border-style: solid;
      border-color: map-get($colors-secondary, grey);
      border-width: 0 1px 0 0;
    }

    &__total-lightbulbs {
      border-style: solid;
      border-color: map-get($colors-secondary, grey);
      border-width: 0 1px 0 0;
    }

    .number-of-lightbulbs {
      font-size: 42px;
      line-height: 100%;
      font-family: 'gotham-light';

      &__added {
        line-height: 100%;
        height: 24px;
      }
    }
}
"FILENAME: C:\dev\c3\rawRepos\c3iot-web\src\app\components\bulb-overview-summary\bulb-
overview-summary.component.ts"

import { Component, OnInit } from '@angular/core';
import { ActivatedRoute } from '@angular/router';
import { LightbulbDashboardDTO } from '../../dto/lightbulb-dashboard.dto';
import { ComponentBase } from '../base/component.base';
import { LightbulbDetailService } from '../../services/lightbulb-detail.service';
import { SmartBulbTableDTO } from 'src/app/dto/smart-bulb-table.dto';
import * as Highcharts from 'highcharts';

@Component({
  selector: 'bulb-overview-summary',
  templateUrl: 'bulb-overview-summary.component.html',
  styleUrls: ['./bulb-overview-summary.component.scss']
})

export class BulbOverviewSummaryComponent extends ComponentBase implements OnInit {
  bulbDetails: SmartBulbTableDTO;
  arrowType: string;
  arrowColor: string;
  riskChange: number;
  smartBulbName: string = '';
  loading: boolean = true;

  Highcharts = Highcharts; // required
  chartConstructor = 'chart'; // optional string, defaults to 'chart'
  riskChartData: Array<number>;
  showChart: boolean;
  updateFlag:boolean = false; // optional boolean
```

```
oneToOneFlag = true; // optional boolean, defaults to false
runOutsideAngular = false; // optional boolean, defaults to false

constructor(private route: ActivatedRoute, private lightbulbDetailService: LightbulbDetailService) {
 super();
}

chartOptions = {
 chart: {
  type: 'area'
 },
 legend:{
  enabled: false
 },
 title: {
    text: ''
 },
 xAxis: {
  title: {
   text: ''
  },
  visible: false
 },
 yAxis: {
  title: {
     text: ''
  },
  visible: false
 },
 plotOptions: {
    area: {
       //pointStart: 1940,
       fillColor: {
         linearGradient: {
            x1: 0,
            y1: 0,
            x2: 0,
            y2: 1
         },
         stops: [
            [0, Highcharts.getOptions().colors[8]],
            [1, Highcharts.Color(Highcharts.getOptions().colors[8]).setOpacity(0).get('rgba')]
         ]
      }
    }
 },
 series: [{
  data: [],
  color: 'red',
  pointStart: Date.UTC(2010, 0, 1),
  pointInterval: 3600 * 1000,
  marker: {
   enabled: false
 }
 }]
};
```

```
ngOnInit() {
  this.route.params.subscribe(params => {
    this.smartBulbName = params['id'];
    this.getLatestBulbDetails();
  });

  this.lightbulbDetailService.currentLoadingValue.subscribe(
    (result) => {
      this.loading = result;
    }
  )
}

getLatestBulbDetails() {
  this.lightbulbDetailService.currentValue.subscribe(
    (result) => {
      console.log("bulb summary result", result);
      if (!(Object.keys(result).length === 0)) {
        this.bulbDetails = result.table[0];

        if (result.table.length > 0) {
          this.getRiskChartData(result.map.riskscore);
          this.calculateRiskChange(result.table);
        }
      } else {
        this.clearData();
      }
    }
  )
}

clearData() {
  this.bulbDetails = null;
  this.riskChange = null;

  this.chartOptions.series[0].data = null;
  this.showChart=true;
  this.updateFlag = true;
}

getRiskChartData(riskScoreData: Array<number>) {
  this.riskChartData = riskScoreData.slice(0, 23);
  this.chartOptions.series[0].data=this.riskChartData.reverse();
  this.showChart=true;
  this.updateFlag = true;
}

setArrow(riskChange) {
  if (riskChange > 0) {
    this.arrowType = 'arrow_drop_up';
    this.arrowColor = 'color_red';
  }
  else if(this.riskChange===0){
    this.arrowType = 'remove';
    this.arrowColor = 'color_yellow';
```

```
    }
    else {
      this.arrowType = 'arrow_drop_down';
      this.arrowColor = 'color_green';
    }
  }

  getStatusString(status: number) {
    return status === 1 ? "On" : "Off";
  }

  showBulbOnIcon(status: string) {
    if(typeof status === 'string'){
      return status.toLowerCase() === 'on' ? true : false;
    }
  }

  calculateRiskChange(results) {
    // currently assuming hourly interval
    let compareIndex = Math.min(23, results.length-1);

    let currentRiskScore = parseInt(results[0].riskscore, 10);
    let compareRiskScore = parseInt(results[compareIndex].riskscore, 10);

    this.riskChange = currentRiskScore-compareRiskScore;
    if (isNaN(this.riskChange)){
      this.riskChange=0;
    }
    this.setArrow(this.riskChange);
    this.riskChange = Math.abs(this.riskChange);
  }

}
"FILENAME: C:\dev\c3\rawRepos\c3iot-web\src\app\components\bulb-search-form\bulb-search-
form.component.html"

<div fxLayout="column" class="search-form">

    <div fxLayout="row" class="search-form__title">
        <div fxFlex class="search-form__title-copy">Filter</div>
    </div>
    <form class="search-form__container" fxLayout="row wrap">
    <mat-form-field fxFlex>
            <input type="text" (blur)="bulbIdOptionSelect($event.target.value)" (keyup) =
"bulbIdOptionSelect($event.target.value)" placeholder="Bulb ID" aria-label="Number" matInput
[formControl]="bulbIdControl" [matAutocomplete]="autoBulbId">
            <mat-autocomplete #autoBulbId="matAutocomplete">
                <mat-option *ngFor="let bulbIdOption of filteredBulbIdOptions | async"
[value]="bulbIdOption">
                {{bulbIdOption}}
                </mat-option>
            </mat-autocomplete>
        </mat-form-field>

        <mat-form-field fxFlex>
```

```
        <input [(ngModel)]="startDate" matInput [matDatepicker]="startDatePicker" name="startDate"
placeholder="Start date">
        <mat-datepicker-toggle matSuffix [for]="startDatePicker"></mat-datepicker-toggle>
        <mat-datepicker #startDatePicker></mat-datepicker>
      </mat-form-field>

      <mat-form-field fxFlex>
          <input [(ngModel)]="endDate" matInput [matDatepicker]="endDatePicker" name="endDate"
placeholder="End date">
          <mat-datepicker-toggle matSuffix [for]="endDatePicker"></mat-datepicker-toggle>
          <mat-datepicker #endDatePicker></mat-datepicker>
      </mat-form-field>

      <mat-form-field fxFlex>
        <mat-select [(value)]="selectedFrequency" matNativeControl placeholder="Frequency">
          <mat-option value="1">Hourly</mat-option>
          <mat-option value="24">Daily</mat-option>
          <mat-option value="168">Weekly</mat-option>
        </mat-select>
      </mat-form-field>

      <div class="search-form__button-container" fxFlex>
        <div class="search-form__search-buttons" fxLayout="row" fxLayoutGap="15px">
          <button class="search-button search-button--primary" fxFlex="50"
(click)="filterResults()">Filter</button>
          <button class="search-button search-button--secondary" fxFlex="50"
(click)="clearAllFilterValues()">Reset</button>
        </div>
      </div>

  </form>
</div>
"FILENAME: C:\dev\c3\rawRepos\c3iot-web\src\app\components\bulb-search-form\bulb-search-
form.component.scss"

@import "../../styles/_colors";

.search-form {
  background-color: map-get($colors-secondary, white);
  margin-top: 25px;
  border: 1px solid rgba(0,0,0,.03);
  border-radius: 3px;
  box-shadow: 0 2px 2px rgba(0,0,0,.24), 0 0 2px rgba(0,0,0,.12);

  &__title {
    border-style: solid;
    border-color: map-get($colors-secondary, grey);
    border-width: 0 0 1px 0;
  }

  &__title-copy {
    margin-left: 15px;
    color: map-get($colors-secondary, medium-dark-grey);
    font-family: 'gotham-bold';
    font-size: 12px;
    line-height: 50px;
```

```
    }

    &__container {
        margin-top: 10px;
        mat-form-field {
            font-size: 10px;
            padding: 0 15px 0 15px;
        }
    }

    &__button-container
    {
        margin-top: 300px;
        border-style: solid;
        border-color: map-get($colors-secondary, grey);
        border-width: 1px 0 0 0;
    }

    &__search-buttons {
        padding: 15px;
        .search-button {
            font-size:10px;
            font-family: 'gotham-bold';
            border-style: solid;
            border-color: map-get($colors-secondary, blue);
            border-width: 2px;
            border-radius: 4px;

            &--primary {
                background-color: map-get($colors-secondary, blue);
                color: white;
            }

            &--secondary {
                background-color: white;
                color: map-get($colors-secondary, blue);
            }
        }
    }

    mat-autocomplete {
        font-size: 14px;
    }
}
```

"FILENAME: C:\dev\c3\rawRepos\c3iot-web\src\app\components\bulb-search-form\bulb-search-form.component.ts"

```
import { Component, OnInit } from '@angular/core';
import { FormControl } from '@angular/forms';
import { ActivatedRoute } from '@angular/router';
import { ComponentBase } from '../base/component.base';
import { Observable } from 'rxjs';
import { map, startWith } from 'rxjs/operators';
import { LightbulbDetailService } from '../../services/lightbulb-detail.service';
import { LightbulbDetailAutocompleteService } from '../../services/lightbulb-detail-autocomplete.service';
import { parse, differenceInDays } from 'date-fns';
```

```
@Component({
  selector: 'bulb-search-form',
  templateUrl: 'bulb-search-form.component.html',
  styleUrls: ['./bulb-search-form.component.scss']
})

export class BulbSearchFormComponent extends ComponentBase implements OnInit {
  selectedBulbId: string = '';
  bulbIdControl = new FormControl();
  bulbIdOptions: string[] = [];
  filteredBulbIdOptions: Observable<string[]>;
  startDate: Date;
  selectedFrequency: string;
  endDate: Date;


  constructor(private route: ActivatedRoute,
    private lightbulbDetailAutocompleteService: LightbulbDetailAutocompleteService,
    private LightbulbDetailService: LightbulbDetailService) {
    super();
  }

  ngOnInit() {
    this.getDashboardAutocompleteItems();
    this.route.params.subscribe(params => {
      this.selectedBulbId = params['id'];
      this.bulbIdControl.setValue(this.selectedBulbId);
    });
  }

  private _bulbIdFilter(value: string): string[] {
    const filterValue = value.toLowerCase();
    return this.bulbIdOptions.filter(option => option.toLowerCase().includes(filterValue));
  }

  private _setupFilters() {
    this.filteredBulbIdOptions = this.bulbIdControl.valueChanges
      .pipe(
        startWith(''),
        map(value => this._bulbIdFilter(value))
      );
  }

  filterResults() {
    console.log("startDate", this.startDate);
    console.log("endDate", this.endDate);
    console.log("frequency", this.selectedFrequency);

    let bulbId: string;
    let endDate: string;
    let daysBack: string;

    this.LightbulbDetailService.setLoading(true);

    if (this.selectedBulbId && this.selectedBulbId.length > 0) {
```

```
    bulbId = this.selectedBulbId;
   }

   if (this.endDate) {
    endDate = this.endDate.toISOString();

    if (this.startDate) {
     daysBack = differenceInDays(this.endDate, this.startDate).toString();
    }
   } else if (this.startDate) {
    endDate = parse(Date.now()).toISOString();
    daysBack = differenceInDays(endDate, this.startDate).toString();
   }

   if (this.selectedFrequency) {
    this.LightbulbDetailService.setChartInterval(parseInt(this.selectedFrequency, 10));
   }

   this.getLightbulbDashboardItems(bulbId, endDate, daysBack);
  }

  bulbIdOptionSelect(value) {
   this.selectedBulbId = value;
  }

  clearAllFilterValues() {
   this.startDate = null;
   this.endDate = null;
   this.selectedFrequency = "168";
  }

  getDashboardAutocompleteItems() {
   this.lightbulbDetailAutocompleteService.getLightbulbDetailAutocompleteItems()
    .takeUntil(this.ngUnsubscribe)
    .finally(() => console.log("Got Lightbulb Detail Autocomplete Items"))
    .subscribe(
     (result) => {
      this.bulbIdOptions = result.bulbNames;
      this._setupFilters();
     }
    );
  }

  getLightbulbDashboardItems(smartBulbName: string, timestamp?: string, daysBack?: string) {
   this.LightbulbDetailService.getAllLightbulbDetailItems(smartBulbName, timestamp, daysBack);
  }
 }
"FILENAME: C:\dev\c3\rawRepos\c3iot-web\src\app\components\dashboard\dashboard.component.html"

<div fxLayout="column">
   <view-header></view-header>
</div>
<div fxLayout="row">
   <div class="search-container">
      <search-form></search-form>
  </div>
```

```
  <div fxLayout="column" fxLayoutGap="20px" class="main-container" fxFlex>
    <overview-summary class="dash-component"></overview-summary>
    <risk-chart class="dash-component"></risk-chart>
    <status-map class="dash-component"></status-map>
    <details-table class="dash-component"></details-table>
  </div>
</div>
```

"FILENAME: C:\dev\c3\rawRepos\c3iot-web\src\app\components\dashboard\dashboard.component.scss"

```
@import "../../styles/_colors";

.dash-component {
    border: 1px solid rgba(0,0,0,.03);
    border-radius: 3px;
    box-shadow: 0 2px 2px rgba(0,0,0,.24), 0 0 2px rgba(0,0,0,.12);
}

.main-container {
    margin: 25px;
}

.search-container {
    width: 240px;
    margin: 0 0 0 20px;
}
```

"FILENAME: C:\dev\c3\rawRepos\c3iot-web\src\app\components\dashboard\dashboard.component.ts"

```
import { Component } from '@angular/core';

@Component({
  selector: 'dashboard',
  templateUrl: 'dashboard.component.html',
  styleUrls: ['./dashboard.component.scss']
})

export class DashboardComponent { }
```

"FILENAME: C:\dev\c3\rawRepos\c3iot-web\src\app\components\details-table\details-table.component.html"

```
<div fxLayout="column" fxFlexFill class="details-table__container">
    <div class="section-title">
        <div class="section-title__title-copy">
            <h4>Lightbulbs</h4>
        </div>
    </div>
    <div class="spinner-container" *ngIf="loading" fxLayoutAlign="center center">
            <mat-spinner [diameter]="30"></mat-spinner>
        </div>
    <div class="details-table" fxFlex *ngIf="!loading">
        <mat-table [dataSource]="dataSource" matSort (matSortChange)="sortData($event)">

            <!--- Note that these columns can be defined in any order.
```

```
                The actual rendered columns are set as a property on the row definition" -->

        <!-- Bulb ID Column -->
        <ng-container matColumnDef="bulbName">
          <mat-header-cell *matHeaderCellDef mat-sort-header="smartBulbName"> Bulb ID </mat-
header-cell>
          <mat-cell *matCellDef="let element"> <a [routerLink]="['bulb-detail', element.bulbName]"
routerLinkActive="active" class="noUnder">{{element.bulbName}}</a> </mat-cell>
        </ng-container>

        <!-- Risk Score Column -->
        <ng-container matColumnDef="riskScore">
          <mat-header-cell *matHeaderCellDef mat-sort-header="riskScore"> Risk Score % </mat-header-
cell>
          <mat-cell *matCellDef="let element">
            <div class="risk-score-container">
              <mat-progress-spinner [color]="getCircleColor(element)" [mode]="mode" [value]=
"element.riskScore" [diameter]="diameter"></mat-progress-spinner>
              <div class="risk-score-number">{{element.riskScore}}</div>
            </div>
          </mat-cell>
        </ng-container>

        <!-- Bulb Type Column -->
        <ng-container matColumnDef="bulbType">
          <mat-header-cell *matHeaderCellDef mat-sort-header="bulbType"> Bulb Type </mat-header-
cell>
          <mat-cell *matCellDef="let element"> {{element.bulbType}} </mat-cell>
        </ng-container>

        <!-- Manufacturer Column -->
        <ng-container matColumnDef="manufacturer">
          <mat-header-cell *matHeaderCellDef mat-sort-header="manufacturer"> Manufacturer </mat-
header-cell>
          <mat-cell *matCellDef="let element"> {{element.manufacturer}} </mat-cell>
        </ng-container>

        <!-- Date Started Column -->
        <ng-container matColumnDef="startDate">
          <mat-header-cell *matHeaderCellDef mat-sort-header="startDate"> Date Started </mat-header-
cell>
          <mat-cell *matCellDef="let element"> {{element.startDate}} </mat-cell>
        </ng-container>

        <mat-header-row *matHeaderRowDef="displayedColumns"></mat-header-row>
        <mat-row *matRowDef="let row; columns: displayedColumns;"></mat-row>
      </mat-table>
      <mat-paginator #paginator [pageSize]="pageSize" [pageSizeOptions]="pageSizeOptions"
[showFirstLastButtons]="true" [length]="length"
        [pageIndex]="currentPage" (page)="pageEvent = handleTable($event)">
      </mat-paginator>
    </div>
</div>
"FILENAME: C:\dev\c3\rawRepos\c3iot-web\src\app\components\details-table\details-
table.component.scss"
```

```
@import "../../styles/_colors";

.section-title {
    border-style: solid;
    border-color: map-get($colors-secondary, grey);
    border-width: 0 0 1px 0;

    &__title-copy {
        margin: 0 0 0 20px;
    }
}

.spinner-container {
  width: 100%;
  padding: 250px;
  background-color: map-get($colors-secondary, light-grey);
}

.details-table {
    &__container {
        background-color: map-get($colors-secondary, white);
    }
}

.noUnder{
  text-decoration-line: none;
}

.risk-score-container {
  position: relative;
  text-align: center;
}

.risk-score-number {
  position: absolute;
  top: 50%;
  left: 50%;
  transform: translate(-50%, -50%);
}

mat-table {
    text-align: left;
    max-height: 600px;
    overflow: auto;
    width: 100%;

    .mat-cell,
    .mat-header-cell {
      padding: 0 4px;
    }

    .mat-row,
    .mat-header-row {
      padding: 0 10px;
    }
```

```
  /*
  .mat-row:nth-child(even) {
    background-color: #F3F3F3;
  }
  */

  &.hover {
    mat-row {
      cursor: pointer;

      &.selected {
        background-color: rgb(181, 233, 244);
      }
    }
  }

  .mat-cell {
    text-overflow: ellipsis;
    white-space: nowrap;
  }

  .mat-header-row {
    position: sticky;
    position: -webkit-sticky;
    top: 0;
    background-color: inherit;
    z-index: 5;
  }
 }
"FILENAME: C:\dev\c3\rawRepos\c3iot-web\src\app\components\details-table\details-table.component.ts"

import { Component, OnInit } from '@angular/core';
import { LightbulbDashboardDTO } from '../../dto/lightbulb-dashboard.dto';
import { ComponentBase } from '../base/component.base';
import { LightbulbDashboardService } from '../../services/lightbulb-dashboard.service';
import {PageEvent} from '@angular/material';
import {Sort} from '@angular/material';
import { SmartBulbDTO } from 'src/app/dto/smart-bulb.dto';

@Component({
 selector: 'details-table',
 templateUrl: 'details-table.component.html',
 styleUrls: ['./details-table.component.scss']
})

export class DetailsTableComponent extends ComponentBase implements OnInit {
 mode: string = 'determinate';
 diameter: number = 40;

 length: number = 100;
 pageSize: number = 10;
 pageSizeOptions: number[] = [5, 10, 25, 50, 100];
 currentPage: number = 0;
 pageEvent: PageEvent;
 loading: boolean = true;
```

```
  displayedColumns: string[] = ['bulbName', 'riskScore', 'bulbType', 'manufacturer', 'startDate'];
  dataSource: LightbulbDetail[];

  dashboardItems: LightbulbDashboardDTO;
  sortedData: SmartBulbDTO[];

  constructor(private lightbulbDashboardService: LightbulbDashboardService) {
    super();
  }

  ngOnInit() {
    this.lightbulbDashboardService.currentLoadingValue.subscribe(
      (result) => {
        this.loading = result;
      }
    )

    this.getDashboardItems();
  }

  getDashboardItems() {
    this.lightbulbDashboardService.currentValue.subscribe(
      (result) => {
        if (!(Object.keys(result).length === 0)) {
          this.dashboardItems = result;
          if (this.dashboardItems.smartBulbs) {
            this.length=this.dashboardItems.smartBulbs.length;
          } else {
            this.length = 0;
          }
          this.sortData({active: "riskScore", direction: "desc"});
        }
      }
    )
  }

  updateTable() {
    const array = this.sortedData;
    const start = this.currentPage * this.pageSize;
    let currTableDisplay=[];
    for(let i = 0; i < this.pageSize; i++){
      const curr = start+i;
      if (curr < array.length) {
        currTableDisplay[i] = {bulbName: array[curr].smartBulbName, riskScore: array[curr].riskScore,
        bulbType: array[curr].bulbType, manufacturer: array[curr].manufacturer, startDate:
array[curr].startDate };
      }
    }

    this.dataSource=[...currTableDisplay];
  }

  handleTable(e: any) {
    this.currentPage = e.pageIndex;
    this.pageSize = e.pageSize;
    this.updateTable();
```

```
  }

  sortData(sort: Sort) {
    const data = this.dashboardItems.smartBulbs;
    if (!sort.active || sort.direction === '') {
      this.sortedData = data;
      return;
    }

    this.sortedData = data.sort((a, b) => {
      const isAsc = sort.direction === 'asc';
      switch (sort.active) {
        case 'smartBulbName':
          var aNum = Number(a.smartBulbName.replace(/\D/g, ""));
          var bNum = Number(b.smartBulbName.replace(/\D/g, ""));
          return compare(aNum, bNum, isAsc);
        case 'riskScore': return compare(a.riskScore, b.riskScore, isAsc);
        case 'bulbType': return compare(a.bulbType, b.bulbType, isAsc);
        case 'manufacturer': return compare(a.manufacturer, b.manufacturer, isAsc);
        case 'startDate': return compare(a.startDate, b.startDate, isAsc);
        default: return 0;
      }
    });
    this.updateTable();
  }

  getCircleColor(element){
    if (element.riskScore>=80){
      return 'warn';
    }
    else{
      return 'accent';
    }
  }
}

function compare(a: number | string, b: number | string, isAsc: boolean) {
  return (a < b ? -1 : 1) * (isAsc ? 1 : -1);
}

export interface LightbulbDetail {
  bulbName: string;
  riskScore: number;
  bulbType: string;
  manufacturer: string;
  startDate: string;
}
"FILENAME: C:\dev\c3\rawRepos\c3iot-web\src\app\components\overview-summary\overview-
summary.component.html"

<div fxLayout="column" fxFlexFill class="overview-data__container">
    <div class="section-title">
        <div class="section-title__title-copy">
            <h4>Key Metrics</h4>
        </div>
    </div>
```

```
  <div class="spinner-container" *ngIf="loading" fxLayoutAlign="center center">
    <mat-spinner [diameter]="30"></mat-spinner>
  </div>
  <div class="overview-data" fxLayout="row" *ngIf="!loading">
    <div class="overview-data__lightbulbs-at-risk overview-data__value-section" fxFlex="33"
fxLayout="column" fxAlign="start">
      <div class="overview-data__label">
        Lightbulbs at Risk
      </div>
      <div class="number-of-lightbulbs">{{dashboardItems?.bulbsAtRisk}}</div>
      <div fxLayout="row">
        <mat-icon class="material-icons {{arrowColor}}">{{arrowType}}</mat-icon>
        <div class="overview-data__value--small
{{arrowColor}}">{{dashboardItems?.bulbsAtRisk24}} in last 24 hours</div>
      </div>
    </div>
    <div class="overview-data__total-lightbulbs overview-data__value-section" fxFlex="33"
fxLayout="column" fxAlign="start">
      <div class="overview-data__label">Total Lightbulbs</div>
      <div class="overview-data__value">{{smartBulbCount}}</div>
    </div>
    <div class="overview-data__failures-ytd overview-data__value-section" fxFlex="33"
fxLayout="column" fxAlign="start">
      <div class="overview-data__label">Failures (YTD)</div>
      <div class="overview-data__value">{{dashboardItems?.failuresYTD}}</div>
    </div>
  </div>
</div>
"FILENAME: C:\dev\c3\rawRepos\c3iot-web\src\app\components\overview-summary\overview-
summary.component.scss"

@import "../../styles/_colors";

.section-title {
  border-style: solid;
  border-color: map-get($colors-secondary, grey);
  border-width: 0 0 1px 0;

  &__title-copy {
    margin: 0 0 0 20px;
  }
}

.spinner-container {
  width: 100%;
  padding: 50px;
  background-color: map-get($colors-secondary, light-grey);
}

.overview-data {
  mat-icon {
    line-height: 100%;
  }

  &__container {
    background-color: map-get($colors-secondary, white);
```

```
}

&__label {
  font-size: 10px;
  font-family: 'gotham-bold';
  text-transform: uppercase;
  color: map-get($colors-secondary, medium-light-grey);
}

&__value {
  font-size: 42px;
  line-height: 100%;
  font-family: 'gotham-light';

  &--small {
    font-size: 10px;
    font-family: 'gotham-bold';
    line-height: 24px;
    &.color_green {
      color: map-get($colors-primary, success);
    }
    &.color_red {
      color: map-get($colors-primary, danger);
    }
    &.color_yellow {
      color: map-get($colors-primary, warn);
    }
  }
}

&__value-section {
  padding: 10px 0 10px 15px;
}

&__lightbulbs-at-risk {
  border-style: solid;
  border-color: map-get($colors-secondary, grey);
  border-width: 0 1px 0 0;
}

&__total-lightbulbs {
  border-style: solid;
  border-color: map-get($colors-secondary, grey);
  border-width: 0 1px 0 0;
}

.number-of-lightbulbs {
  font-size: 42px;
  line-height: 100%;
  font-family: 'gotham-light';

  &__added {
    line-height: 100%;
    height: 24px;
  }
}
```

```
}
"FILENAME: C:\dev\c3\rawRepos\c3iot-web\src\app\components\overview-summary\overview-
summary.component.ts"

import { Component, OnInit } from '@angular/core';
import { LightbulbDashboardDTO } from '../../dto/lightbulb-dashboard.dto';
import { ComponentBase } from '../base/component.base';
import { LightbulbDashboardService } from '../../services/lightbulb-dashboard.service';

@Component({
  selector: 'overview-summary',
  templateUrl: 'overview-summary.component.html',
  styleUrls: ['./overview-summary.component.scss']
})

export class OverviewSummaryComponent extends ComponentBase implements OnInit {
  dashboardItems: LightbulbDashboardDTO;
  smartBulbCount: number;
  arrowType: string;
  arrowColor: string;
  loading: boolean = true;


  constructor(private lightbulbDashboardService: LightbulbDashboardService) {
    super();
  }

  ngOnInit() {
    this.lightbulbDashboardService.currentLoadingValue.subscribe(
      (result) => {
        this.loading = result;
      }
    )

    this.getDashboardItems();
  }

  getDashboardItems() {
    this.lightbulbDashboardService.currentValue.subscribe(
      (result) => {
        if (!(Object.keys(result).length === 0)) {
          this.dashboardItems = result;
          this.smartBulbCount = this.dashboardItems.smartBulbs.length;
          this.getArrow();
        }
      }
    )
  }

  getArrow(){
    if (this.dashboardItems.bulbsAtRisk24>0){
      this.arrowType = 'arrow_drop_up';
      this.arrowColor = 'color_green';
    }
    else if(this.dashboardItems.bulbsAtRisk24===0){
      this.arrowType = 'remove';
```

```
      this.arrowColor = 'color_yellow';
    }
    else{
      this.arrowType = 'arrow_drop_down';
      this.arrowColor = 'color_red';
    }
  }

}
```

"FILENAME: C:\dev\c3\rawRepos\c3iot-web\src\app\components\risk-chart\risk-chart.component.html"

```
<div fxLayout="column" fxFlexFill class="risk-chart__container">
    <div class="section-title">
        <div class="section-title__title-copy">
            <span>Failure Risk</span><span class="section-title__title-copy--lighter"> (next 30 days)</span>
        </div>
    </div>
    <div class="spinner-container" *ngIf="loading" fxLayoutAlign="center center">
            <mat-spinner [diameter]="30"></mat-spinner>
        </div>
    <div class="risk-chart" *ngIf="showChart && !loading">
        <highcharts-chart
            [Highcharts]="Highcharts"

            [constructorType]="chartConstructor"
            [options]="chartOptions"
            [callbackFunction]="chartCallback"

            [(update)]="updateFlag"
            [oneToOne]="oneToOneFlag"
            [runOutsideAngular]="runOutsideAngularFlag"

            style="width: 100%; height: 300px; display: block;"
        ></highcharts-chart>
    </div>
</div>
```

"FILENAME: C:\dev\c3\rawRepos\c3iot-web\src\app\components\risk-chart\risk-chart.component.scss"

```
@import "../../styles/_colors";

.section-title {
    border-style: solid;
    border-color: map-get($colors-secondary, grey);
    border-width: 0 0 1px 0;

    &__title-copy {
        margin-block-start: 1.33em;
        margin-block-end: 1.33em;
        margin-inline-start: 20px;
        margin-inline-end: 0px;
        font-family: 'gotham-bold', Arial, sans-serif;
        font-weight: bold;
        font-size: 12px;
        line-height: 100%;
        color: map-get($colors-secondary, medium-dark-grey);
```

```
        &--lighter {
            color: map-get($colors-secondary, medium-light-grey);
        }
    }
}

.spinner-container {
    width: 100%;
    padding: 130px;
    background-color: map-get($colors-secondary, light-grey);
}

.risk-chart {
    margin: 20px 30px 0 10px;

    &__container {
        background-color: map-get($colors-secondary, white);
    }
}
"FILENAME: C:\dev\c3\rawRepos\c3iot-web\src\app\components\risk-chart\risk-chart.component.ts"

import { Component, OnInit } from '@angular/core';
import { LightbulbDashboardDTO } from '../../dto/lightbulb-dashboard.dto';
import { ComponentBase } from '../base/component.base';
import { LightbulbDashboardService } from '../../services/lightbulb-dashboard.service';
import * as Highcharts from 'highcharts';

@Component({
  selector: 'risk-chart',
  templateUrl: 'risk-chart.component.html',
  styleUrls: ['./risk-chart.component.scss']
})

export class RiskChartComponent  extends ComponentBase implements OnInit {
  dashboardItems: LightbulbDashboardDTO;
  histoData: Array<number>;
  showChart: boolean;
  loading: boolean = true;

  Highcharts = Highcharts; // required
  chartConstructor = 'chart'; // optional string, defaults to 'chart'
  chartOptions = {
    legend:{
      borderRadius: 5,
      padding: 9,
      backgroundColor: '#FFFFFF',
      floating: true,
      x:-10,
      y:3,
      shadow: true,
      align: 'right',
      verticalAlign: 'top',
      layout: 'horizontal'
    },
    title: '',
    series: [{
```

```
  name:'Low',
  type: "column",
  color: '#167526',
  data: [0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
  grouping: false,
  zoneAxis: 'x',
  zones: [{
    value: 1,
    color:'#167526'
  }, {
    value: 2,
    color:'#31921A'
  }, {
    value: 3,
    color:'#71AD22'
  }, {
    value: 4,
    color:'#BCC526'
  }]
 },
 {
  name:'Medium',
  type: "column",
  color: '#D8AA28',
  data: [0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
  grouping: false,
  zoneAxis: 'x',
  zones: [{
    value: 5,
    color:'#D8AA28'
  }, {
    value: 6,
    color:'#E6882C'
  }, {
    value: 7,
    color:'#F16C2D'
  }]
 },
 {
  name:'High',
  type: "column",
  color: '#F8562E',
  data: [0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
  grouping: false,
  zoneAxis: 'x',
  zones: [{
    value: 8,
    color:'#F8562E'
  }, {
    value: 9,
    color:'#FD472F'
  }, {
    value: 10,
    color:'#F03B2C'
  }]
 }],
```

```
    xAxis: {
      title: { text: 'Risk Score',
        style: {'font-weight': 'bolder'}
      },
      categories: ['0-10%', '10-20%', '20-30%', '30-40%', '40-50%', '50-60%', '60-70%', '70-80%', '80-90%',
'90-100%']
    },
    yAxis: {
      title: { text: '# of Bulbs',
        style: {'font-weight': 'bolder'}
      }
    }
  }; // required
  updateFlag:boolean = false; // optional boolean
  oneToOneFlag = true; // optional boolean, defaults to false
  runOutsideAngular = false; // optional boolean, defaults to false


  constructor(private lightbulbDashboardService: LightbulbDashboardService) {
    super();
  }

  ngOnInit() {
    this.lightbulbDashboardService.currentLoadingValue.subscribe(
      (result) => {
        this.loading = result;
      }
    )

    this.getDashboardItems();
  }

  getDashboardItems() {
    this.lightbulbDashboardService.currentValue.subscribe(
      (result) => {
        if (!(Object.keys(result).length === 0)) {
          this.dashboardItems = result;
          this.getHistoData();
        }
      }
    )
  }

  getHistoData() {
    this.histoData=this.dashboardItems.riskDistribution;
    this.chartOptions.series[0].data=[this.histoData[0], this.histoData[1], this.histoData[2], this.histoData[3],
0, 0, 0, 0, 0, 0];
    this.chartOptions.series[1].data=[0, 0, 0, 0, this.histoData[4], this.histoData[5], this.histoData[6], 0, 0, 0];
    this.chartOptions.series[2].data=[0, 0, 0, 0, 0, 0, 0, this.histoData[7], this.histoData[8], this.histoData[9]];
    this.showChart=true;
    this.updateFlag = true;
  }
}
"FILENAME: C:\dev\c3\rawRepos\c3iot-web\src\app\components\search-form\search-
form.component.html"
```

```
<div fxLayout="column" class="search-form">

   <div fxLayout="row" class="search-form__title">
      <div fxFlex class="search-form__title-copy">Filter</div>
   </div>
   <form class="search-form__container" fxLayout="row wrap">
      <mat-form-field fxFlex>
         <input type="text" (blur)="buildingOptionSelect($event.target.value)" (keyup) =
"buildingOptionSelect($event.target.value)" placeholder="Building" aria-label="Number" matInput
[formControl]="buildingControl" [matAutocomplete]="autoBuilding">
         <mat-autocomplete #autoBuilding="matAutocomplete">
            <mat-option *ngFor="let buildingOption of filteredBuildingOptions | async"
[value]="buildingOption">
            {{buildingOption}}
            </mat-option>
         </mat-autocomplete>
      </mat-form-field>

      <mat-form-field fxFlex>
         <input type="text" (blur)="apartmentOptionSelect($event.target.value)" (keyup) =
"apartmentOptionSelect($event.target.value)" placeholder="Apartment" aria-label="Number" matInput
[formControl]="apartmentControl" [matAutocomplete]="autoApartment">
         <mat-autocomplete #autoApartment="matAutocomplete">
            <mat-option *ngFor="let apartmentOption of filteredApartmentOptions | async"
[value]="apartmentOption">
            {{apartmentOption}}
            </mat-option>
         </mat-autocomplete>
      </mat-form-field>

      <mat-form-field fxFlex>
         <input type="text" (blur)="manufacturerOptionSelect($event.target.value)" (keyup) =
"manufacturerOptionSelect($event.target.value)" placeholder="Manufacturer" aria-label="Number"
matInput [formControl]="manufacturerControl" [matAutocomplete]="autoManufacturer">
         <mat-autocomplete #autoManufacturer="matAutocomplete">
            <mat-option *ngFor="let manufacturerOption of filteredManufacturerOptions | async"
[value]="manufacturerOption">
            {{manufacturerOption}}
            </mat-option>
         </mat-autocomplete>
      </mat-form-field>

      <mat-form-field fxFlex>
         <input type="text" (blur)="bulbTypeOptionSelect($event.target.value)" (keyup) =
"bulbTypeOptionSelect($event.target.value)" placeholder="Bulb Type" aria-label="Number" matInput
[formControl]="bulbTypeControl" [matAutocomplete]="autoBulbType">
         <mat-autocomplete (optionSelected)="bulbTypeOptionSelect($event.option.value)"
#autoBulbType="matAutocomplete">
            <mat-option *ngFor="let bulbTypeOption of filteredBulbTypeOptions | async"
[value]="bulbTypeOption">
            {{bulbTypeOption}}
            </mat-option>
         </mat-autocomplete>
      </mat-form-field>

      <div class="search-form__button-container" fxFlex>
```

```
          <div class="search-form__search-buttons" fxLayout="row" fxLayoutGap="15px">
            <button class="search-button search-button--primary" fxFlex="50"
(click)="filterResults()">Filter</button>
            <button class="search-button search-button--secondary" fxFlex="50"
(click)="clearAllFilterValues()">Clear All</button>
          </div>
        </div>

    </form>
</div>
"FILENAME: C:\dev\c3\rawRepos\c3iot-web\src\app\components\search-form\search-
form.component.scss"

@import "../../styles/_colors";

.search-form {
    background-color: map-get($colors-secondary, white);
    margin-top: 25px;
    border: 1px solid rgba(0,0,0,.03);
    border-radius: 3px;
    box-shadow: 0 2px 2px rgba(0,0,0,.24), 0 0 2px rgba(0,0,0,.12);

    &__title {
        border-style: solid;
        border-color: map-get($colors-secondary, grey);
        border-width: 0 0 1px 0;
    }

    &__title-copy {
        margin-left: 15px;
        color: map-get($colors-secondary, medium-dark-grey);
        font-family: 'gotham-bold';
        font-size: 12px;
        line-height: 50px;
    }

    &__container {
        margin-top: 10px;
        mat-form-field {
            font-size: 10px;
            padding: 0 15px 0 15px;
        }
    }

    &__button-container
    {
        margin-top: 300px;
        border-style: solid;
        border-color: map-get($colors-secondary, grey);
        border-width: 1px 0 0 0;
    }

    &__search-buttons {
        padding: 15px;
        .search-button {
            font-size:10px;
```

```
        font-family: 'gotham-bold';
        border-style: solid;
        border-color: map-get($colors-secondary, blue);
        border-width: 2px;
        border-radius: 4px;

        &--primary {
          background-color: map-get($colors-secondary, blue);
          color: white;
        }

        &--secondary {
          background-color: white;
          color: map-get($colors-secondary, blue);
        }
      }
    }

    mat-autocomplete {
      font-size: 14px;
    }
}
"FILENAME: C:\dev\c3\rawRepos\c3iot-web\src\app\components\search-form\search-form.component.ts"

import { Component, OnInit } from '@angular/core';
import { FormControl } from '@angular/forms';
import { ComponentBase } from '../base/component.base';
import { Observable } from 'rxjs';
import { map, startWith } from 'rxjs/operators';
import { LightbulbDashboardService } from '../../services/lightbulb-dashboard.service';
import { LightbulbDashboardAutocompleteService } from '../../services/lightbulb-dashboard-
autocomplete.service';

@Component({
  selector: 'search-form',
  templateUrl: 'search-form.component.html',
  styleUrls: ['./search-form.component.scss']
})

export class SearchFormComponent extends ComponentBase implements OnInit {
  selectedBuilding: string = '';
  selectedApartment: string = '';
  selectedManufacturer: string = '';
  selectedBulbType: string = '';
  buildingControl = new FormControl();
  apartmentControl = new FormControl();
  manufacturerControl = new FormControl();
  bulbTypeControl = new FormControl();
  apartmentOptions: string[] = [];
  buildingOptions: string[] = [];
  manufacturerOptions: string[] = [];
  bulbTypeOptions: string[] = [];
  filteredApartmentOptions: Observable<string[]>;
  filteredBuildingOptions: Observable<string[]>;
  filteredManufacturerOptions: Observable<string[]>;
  filteredBulbTypeOptions: Observable<string[]>;
```

```
  constructor(private lightbulbDashboardAutocompleteService: LightbulbDashboardAutocompleteService,
private LightbulbDashboardService: LightbulbDashboardService) {
    super();
  }

  ngOnInit() {
    this.getDashboardAutocompleteItems();
  }

  private _apartmentFilter(value: string): string[] {
    const filterValue = value.toLowerCase();
    return this.apartmentOptions.filter(option => option.toLowerCase().includes(filterValue));
  }

  private _buildingFilter(value: string): string[] {
    const filterValue = value.toLowerCase();
    return this.buildingOptions.filter(option => option.toLowerCase().includes(filterValue));
  }

  private _manufacturerFilter(value: string): string[] {
    const filterValue = value.toLowerCase();
    return this.manufacturerOptions.filter(option => option.toLowerCase().includes(filterValue));
  }

  private _bulbTypeFilter(value: string): string[] {
    const filterValue = value.toLowerCase();
    return this.bulbTypeOptions.filter(option => option.toLowerCase().includes(filterValue));
  }

  private _setupFilters() {
    this.filteredApartmentOptions = this.apartmentControl.valueChanges
      .pipe(
        startWith(''),
        map(value => this._apartmentFilter(value))
      );

    this.filteredBuildingOptions = this.buildingControl.valueChanges
      .pipe(
        startWith(''),
        map(value => this._buildingFilter(value))
      );

    this.filteredManufacturerOptions = this.manufacturerControl.valueChanges
      .pipe(
        startWith(''),
        map(value => this._manufacturerFilter(value))
      );

    this.filteredBulbTypeOptions = this.bulbTypeControl.valueChanges
      .pipe(
        startWith(''),
        map(value => this._bulbTypeFilter(value))
      );
  }
```

```
  filterResults() {
    this.LightbulbDashboardService.setLoading(true);
    this.getLightbulbDashboardItems(this.selectedBuilding, this.selectedApartment,
this.selectedManufacturer, this.selectedBulbType);
  }

  buildingOptionSelect(value) {
    this.selectedBuilding = value;
  }

  apartmentOptionSelect(value) {
    this.selectedApartment = value;
  }

  manufacturerOptionSelect(value) {
    this.selectedManufacturer = value;
  }

  bulbTypeOptionSelect(value) {
    this.selectedBulbType = value;
  }

  clearAllFilterValues() {
    this.selectedBuilding = '';
    this.selectedApartment = '';
    this.selectedManufacturer = '';
    this.selectedBulbType = '';

    this.buildingControl.setValue('');
    this.apartmentControl.setValue('');
    this.manufacturerControl.setValue('');
    this.bulbTypeControl.setValue('');
  }

  getDashboardAutocompleteItems() {
    this.lightbulbDashboardAutocompleteService.getDashboardAutocompleteItems()
      .takeUntil(this.ngUnsubscribe)
      .finally(() => console.log("Got Dashboard Autocomplete Items"))
      .subscribe(
        (result) => {
          this.apartmentOptions = result.apartments;
          this.buildingOptions = result.buildings;
          this.manufacturerOptions = result.manufacturers;
          this.bulbTypeOptions = result.bulbTypes;

          this._setupFilters();
        }
      );
  }

  getLightbulbDashboardItems(building?: string, apartment?: string, manufacturer?: string, bulbType?:
string) {
    this.LightbulbDashboardService.getAllLightbulbDashboardItems(building, apartment, manufacturer,
bulbType);
  }
```

```
 }
"FILENAME: C:\dev\c3\rawRepos\c3iot-web\src\app\components\status-map\status-map.component.html"

<div fxFlexFill>
   <div class="spinner-container" *ngIf="loading" fxLayoutAlign="center center">
        <mat-spinner [diameter]="30"></mat-spinner>
   </div>
   <agm-map [latitude]="lat" [longitude]="lng" [zoom]="zoom" [scrollwheel]="false" *ngIf="!loading">
      <agm-marker *ngFor="let i of agmMarkers"
         [latitude]="i.lat" [longitude]="i.lng" [iconUrl]="i.icn" (markerClick)="goToPage(i.name)">
      </agm-marker>
   </agm-map>
</div>
"FILENAME: C:\dev\c3\rawRepos\c3iot-web\src\app\components\status-map\status-map.component.scss"

@import "../../styles/_colors";

agm-map {
   height: 600px;
}

.spinner-container {
   width: 100%;
   padding: 200px;
   background-color: map-get($colors-secondary, light-grey);
}
"FILENAME: C:\dev\c3\rawRepos\c3iot-web\src\app\components\status-map\status-map.component.ts"

import { Component, OnInit } from '@angular/core';
import { LightbulbDashboardDTO } from '../../dto/lightbulb-dashboard.dto';
import { ComponentBase } from '../base/component.base';
import { LightbulbDashboardService } from '../../services/lightbulb-dashboard.service';
import { Router } from '@angular/router';

@Component({
 selector: 'status-map',
 templateUrl: 'status-map.component.html',
 styleUrls: ['./status-map.component.scss']
})

export class StatusMapComponent extends ComponentBase implements OnInit {
 dashboardItems: LightbulbDashboardDTO;
 lat: number = 37.4852444;
 lng: number = -122.23752;
 zoom: number = 15.2;
 loading: boolean = true;

 private greenIcon = {
  url: 'https://upload.wikimedia.org/wikipedia/commons/0/0e/Ski_trail_rating_symbol-green_circle.svg',
  scaledSize: {
    width: 10,
    height: 10
  }
 };

 private redIcon = {
```

```
  url: 'https://www.freeiconspng.com/uploads/red-circle-png-transparent-2.png',
  scaledSize: {
    width: 8,
    height: 8
  }
};

private agmMarkers: agmmarker[]

constructor(private lightbulbDashboardService: LightbulbDashboardService, private router: Router) {
  super();
}

ngOnInit() {
  this.lightbulbDashboardService.currentLoadingValue.subscribe(
    (result) => {
      this.loading = result;
    }
  )

  this.getDashboardItems();
}

getDashboardItems() {
  this.lightbulbDashboardService.currentValue.subscribe(
    (result) => {
      if (!(Object.keys(result).length === 0)) {
        this.dashboardItems = result;
        this.updateMarkers();
      }
    }
  );
}

updateMarkers(){
  this.agmMarkers = [];
   for (let entry of this.dashboardItems.smartBulbs) {
     let icn;
     if (Number(entry.riskScore)>50){
       icn=this.redIcon;
     }
     else{
      icn=this.greenIcon;
     }
     this.agmMarkers.push({
       lat: entry.latitude,
       lng: entry.longitude,
       icn: icn,
       name: entry.smartBulbName
     });
   }
}

goToPage(pageName:string){
  this.router.navigate(['bulb-detail',`${pageName}`]);
}
```

```
}

interface agmmarker {
  lat?: number;
  lng?: number;
  icn?: Object;
  name?: string;
}
"FILENAME: C:\dev\c3\rawRepos\c3iot-web\src\app\components\view-header\view-
header.component.html"

<div fxLayout="column" fxFlexFill class="view-header">
  <div fxLayout="row" class="view-header__title">
     <div class="view-header__title-copy">
        <h4>Lightbulb Predictive Maintenance</h4>
     </div>
     <div class="view-header__remaining-space"></div>
  </div>
  <div fxLayout="row" class="breadcrumb">
       <a routerLink="/" routerLinkActive="active noUnder"><div [ngClass]="{'breadcrumb__content':
dashboardOn == true, 'breadcrumb__lightContent': dashboardOn == false}">Dashboard</div></a>
       <span class="breadcrumb__lightContent" *ngIf="!dashboardOn">></span>
       <span class="breadcrumb__content" *ngIf="!dashboardOn">{{smartBulbName}}</span>
     <div class="nextCont"><a (click)="setSpinner()" [routerLink]="['../', next]" routerLinkActive="active"
class="breadcrumb__link" *ngIf="!dashboardOn">Next ></a></div>
  </div>
</div>
"FILENAME: C:\dev\c3\rawRepos\c3iot-web\src\app\components\view-header\view-
header.component.scss"

@import "../../styles/_colors";

.view-header {
  background-color: map-get($colors-secondary, white);

  &__title {
    border-style: solid;
    border-color: map-get($colors-secondary, grey);
    border-width: 0 0 1px 0;
  }

  &__title-copy {
    padding: 0 10px 0 15px;
    border-style: solid;
    border-color: map-get($colors-secondary, grey);
    border-width: 0 1px 0 0;
  }
}

.noUnder {
  text-decoration: none;
  color: map-get($colors-secondary, black);
}

.nextCont{
  width: 100%;
```

```
}

.breadcrumb {
  border: 0 0 1px 0 solid map-get($colors-secondary, grey);
  box-shadow: 0 2px 0px rgba(0,0,0,.12), 0 0 2px rgba(0,0,0,.12);

  &__content {
    white-space: nowrap;
    padding: 0 0 0 20px;
    font-size: 12px;
    line-height: 30px;
    font-family: 'gotham-bold';
    color: map-get($colors-secondary, medium-dark-grey);
  }

  &__lightContent {
    white-space: nowrap;
    padding: 0 0 0 20px;
    font-size: 12px;
    line-height: 30px;
    font-family: 'gotham-bold';
    color: map-get($colors-secondary, medium-light-grey);
  }

  &__link {
    float: right;
    text-align: right;
    text-decoration-line: none;
    font-size: 12px;
    line-height: 30px;
    font-family: 'gotham-bold';
    margin-right: 30px;
  }
}
"FILENAME: C:\dev\c3\rawRepos\c3iot-web\src\app\components\view-header\view-header.component.ts"

import { Component } from '@angular/core';
import { ActivatedRoute } from '@angular/router';
import { LightbulbDetailService } from '../../services/lightbulb-detail.service';

@Component({
  selector: 'view-header',
  templateUrl: 'view-header.component.html',
  styleUrls: ['./view-header.component.scss']
})

export class ViewHeaderComponent {
  smartBulbName: string;
  next: string;
  dashboardOn: boolean;

  constructor(private lightbulbDetailService: LightbulbDetailService, private route: ActivatedRoute) {}

  private ngOnInit() {
    this.route.params.subscribe(params => {
      console.log(params)
```

```
    if(params.id){
      this.smartBulbName = params['id'];
      this.dashboardOn=false;
      this.next=this.getNextNum();
    }
    else{
      this.dashboardOn=true;
    }

   });
  }

  setSpinner() {
   this.lightbulbDetailService.setLoading(true);
  }

  getNextNum(){
    let thenum = Number(this.smartBulbName.match(/\d+/)[0]);
    if(thenum===100){
      thenum=1;
    }else thenum+=1;
    return 'SMBLB'+ thenum;
  }
}

"FILENAME: C:\dev\c3\rawRepos\c3iot-web\src\app\dto\lightbulb-dashboard-autocomplete.dto.ts"

export class LightbulbDashboardAutocompleteDTO {
  Buildings: string[];
  Apartments: string[];
}

"FILENAME: C:\dev\c3\rawRepos\c3iot-web\src\app\dto\lightbulb-dashboard.dto.ts"

import { SmartBulbDTO } from './smart-bulb.dto';

export class LightbulbDashboardDTO {
    bulbsAtRisk: number;
    bulbsAtRisk24: number;
    bulbsSwitchedOn: number;
    failuresYTD: number;
    riskDistribution: number[];
    smartBulbs: SmartBulbDTO[];
 }
"FILENAME: C:\dev\c3\rawRepos\c3iot-web\src\app\dto\smart-bulb-detail-autocomplete.dto.ts"

export class SmartBulbDetailAutocompleteDTO {
  BulbNames: string[];
}

"FILENAME: C:\dev\c3\rawRepos\c3iot-web\src\app\dto\smart-bulb-detail.dto.ts"

import { SmartBulbMapDTO } from 'src/app/dto/smart-bulb-map.dto';
import { SmartBulbTableDTO } from 'src/app/dto/smart-bulb-table.dto';

export class SmartBulbDetailDTO {
```

```
  map: SmartBulbMapDTO;
  table: SmartBulbTableDTO[];
}

"FILENAME: C:\dev\c3\rawRepos\c3iot-web\src\app\dto\smart-bulb-map.dto.ts"

export class SmartBulbMapDTO {
  timestamp: string[];
  riskscore: number[];
  status: number[];
  lumens: number[];
  voltage: number[];
  power: number[];
  temperature: number[];
}

"FILENAME: C:\dev\c3\rawRepos\c3iot-web\src\app\dto\smart-bulb-table.dto.ts"

export class SmartBulbTableDTO {
  timestamp: string;
  smartBulbName: string;
  riskscore: number;
  status: number;
  lumens: number;
  voltage: number;
  power: number;
  temperature: number;
  weatherData: JSON;
  switchCountWeek: number;
  hoursOn: number;
  lightbulb:object;
}

"FILENAME: C:\dev\c3\rawRepos\c3iot-web\src\app\dto\smart-bulb.dto.ts"

export class SmartBulbDTO {
  smartBulbName: string;
  manufacturer: string;
  bulbType: string;
  latitude: number;
  longitude: number;
  fixtureName: string;
  apartmentName: string;
  buildingName: string;
  status: number;
  startDate: string;
  riskScore: number;
 }
"FILENAME: C:\dev\c3\rawRepos\c3iot-web\src\app\repositories\lightbulb-dashboard-
autocomplete.repository.ts"

import { Injectable } from '@angular/core';
import { BaseRepository } from './repository';
import { Observable } from 'rxjs';
import { Http, Headers, RequestOptions } from '@angular/http';
```

```
import { environment } from '../../environments/environment';

// Here we inject dto that are necessary for this repositories calls
import { LightbulbDashboardAutocompleteDTO } from '../dto/lightbulb-dashboard-autocomplete.dto';

@Injectable()
export class LightbulbDashboardAutocompleteRepository {
  private baseUrl: string;

  constructor(private baseRepository: BaseRepository) {
    this.baseUrl = 'dashboardautocomplete';
  }

  /**
   * @returns Observable
   */
  getAllLightbulbDashboardAutocompleteItems(): Observable<LightbulbDashboardAutocompleteDTO> {
    return this.baseRepository.get<LightbulbDashboardAutocompleteDTO>(this.baseUrl);
  }
}
```

"FILENAME: C:\dev\c3\rawRepos\c3iot-web\src\app\repositories\lightbulb-dashboard.repository.ts"

```
import { Injectable } from '@angular/core';
import { BaseRepository } from './repository';
import { Observable } from 'rxjs';
import { Http, Headers, RequestOptions } from '@angular/http';
import { HttpParams } from '@angular/common/http';

import { environment } from '../../environments/environment';

// Here we inject dto that are necessary for this repositories calls
import { LightbulbDashboardDTO } from '../dto/lightbulb-dashboard.dto';

@Injectable()
export class LightbulbDashboardRepository {
  private baseUrl: string;

  constructor(private baseRepository: BaseRepository) {
    this.baseUrl = 'dashboard';
  }

  /**
   * @returns Observable
   */
  getAllLightbulbDashboardItems(params?: HttpParams): Observable<LightbulbDashboardDTO> {
    return this.baseRepository.get<LightbulbDashboardDTO>(this.baseUrl, null, params);
  }
}
```

"FILENAME: C:\dev\c3\rawRepos\c3iot-web\src\app\repositories\lightbulb-detail-autocomplete.repository.ts"

```
import { Injectable } from '@angular/core';
import { BaseRepository } from './repository';
import { Observable } from 'rxjs';
```

```
// Here we inject dto that are necessary for this repositories calls
import { SmartBulbDetailAutocompleteDTO } from '../dto/smart-bulb-detail-autocomplete.dto';

@Injectable()
export class LightbulbDetailAutocompleteRepository {
  private baseUrl: string;

  constructor(private baseRepository: BaseRepository) {
    this.baseUrl = 'lightbulbautocomplete';
  }

  /**
   * @returns Observable
   */
  getAllLightbulbDetailAutocompleteItems(): Observable<SmartBulbDetailAutocompleteDTO> {
    return this.baseRepository.get<SmartBulbDetailAutocompleteDTO>(this.baseUrl);
  }
}

"FILENAME: C:\dev\c3\rawRepos\c3iot-web\src\app\repositories\lightbulb-detail.repository.ts"

import { Injectable } from '@angular/core';
import { BaseRepository } from './repository';
import { Observable } from 'rxjs';
import { Http, Headers, RequestOptions } from '@angular/http';
import { HttpParams } from '@angular/common/http';

import { environment } from '../../environments/environment';

// Here we inject dto that are necessary for this repositories calls
import { SmartBulbDetailDTO } from '../dto/smart-bulb-detail.dto';

@Injectable()
export class LightbulbDetailRepository {
  private baseUrl: string;

  constructor(private baseRepository: BaseRepository) {
    this.baseUrl = 'lightbulb';
  }

  /**
   * @returns Observable
   */
  getAllLightbulbDetailItems(params?: HttpParams): Observable<SmartBulbDetailDTO> {
    return this.baseRepository.get<SmartBulbDetailDTO>(this.baseUrl, null, params);
  }
}

"FILENAME: C:\dev\c3\rawRepos\c3iot-web\src\app\repositories\repositories.module.ts"

import { NgModule } from '@angular/core';
import { BaseRepository } from './repository';
import { LightbulbDashboardRepository } from './lightbulb-dashboard.repository';
import { LightbulbDashboardAutocompleteRepository } from './lightbulb-dashboard-
autocomplete.repository';
```

```
import { LightbulbDetailRepository } from './lightbulb-detail.repository';
import { LightbulbDetailAutocompleteRepository } from './lightbulb-detail-autocomplete.repository';

@NgModule({
   providers: [
      BaseRepository,
      LightbulbDashboardRepository,
      LightbulbDashboardAutocompleteRepository,
      LightbulbDetailRepository,
      LightbulbDetailAutocompleteRepository
   ]
 })
 export class RepositoriesModule {}

"FILENAME: C:\dev\c3\rawRepos\c3iot-web\src\app\repositories\repository.ts"

import { Injectable } from '@angular/core';
import { HttpClient, HttpHeaders, HttpParams } from '@angular/common/http';
import { Observable, throwError, of } from 'rxjs';
import { map, catchError } from 'rxjs/operators';

import { environment } from '../../environments/environment';

@Injectable()
export class BaseRepository {
 private httpOptions: Object = {
    headers: new HttpHeaders({
      'x-api-key': environment.apiKey
    })
 };

 constructor(private http: HttpClient) {}

 /**
  * @param  {string} routePrefix
  * @param  {string} route?
  * @returns string
  */
 private buildUrl(routePrefix: string, route?: string): string {
   const url =  `${environment.apiEndpoint}/${environment.apiStage}/${routePrefix}`;
   if (route) {
     return url + `/${route}`;
   } else {
     return url;
   }
 }

 /**
  * @param  {string} routePrefix
  * @param  {string} route?
  * @returns Observable
  */
 get<T>(routePrefix: string, route?: string, params?: HttpParams): Observable<T> {
   return this.http.get(this.buildUrl(routePrefix, route),
                        { headers: new HttpHeaders({
                           'x-api-key': environment.apiKey
```

```
                    }),
                    params: params
                  }
      )
      .pipe(
        map(response => {
          return response as T
        }),
        catchError(errors => {
          return throwError(errors)
        })
      );
  }
}
```

"FILENAME: C:\dev\c3\rawRepos\c3iot-web\src\app\services\lightbulb-dashboard-autocomplete.service.ts"

```
import { Injectable } from '@angular/core';
import { Observable } from 'rxjs';
import { publishLast, refCount } from 'rxjs/operators';
import { HttpClient, HttpHeaders } from '@angular/common/http';
import { LightbulbDashboardAutocompleteRepository } from '../repositories/lightbulb-dashboard-autocomplete.repository';
import { LightbulbDashboardAutocompleteDTO } from '../dto/lightbulb-dashboard-autocomplete.dto';

@Injectable()
export class LightbulbDashboardAutocompleteService {
   constructor (private repository: LightbulbDashboardAutocompleteRepository) {
      this.subject$ = this.getAllLightbulbDashboardAutocompleteItems();
   }

   private subject$ = new Observable<any>();

   getDashboardAutocompleteItems(): Observable<any> {
      return this.subject$;
   }
   /**
    * @returns Observable
    */
   private getAllLightbulbDashboardAutocompleteItems(): Observable<LightbulbDashboardAutocompleteDTO> {
      return this.repository.getAllLightbulbDashboardAutocompleteItems()
         .pipe(
            publishLast(),
            refCount()
         );
   }
}
```

"FILENAME: C:\dev\c3\rawRepos\c3iot-web\src\app\services\lightbulb-dashboard.service.ts"

```
import { Injectable } from '@angular/core';
import { Observable, BehaviorSubject } from 'rxjs';
import { publishLast, publishReplay, publishBehavior, refCount, publish } from 'rxjs/operators';
import { LightbulbDashboardRepository } from '../repositories/lightbulb-dashboard.repository';
```

```
import { LightbulbDashboardDTO } from '../dto/lightbulb-dashboard.dto';
import { HttpParams } from '@angular/common/http';

@Injectable()
export class LightbulbDashboardService {
   constructor (private repository: LightbulbDashboardRepository) {
      this.initializeDashboardItems();
   }

   private valueSource = new BehaviorSubject<LightbulbDashboardDTO>(new
LightbulbDashboardDTO());
   private loadingValueSource =  new BehaviorSubject<boolean>(true);

   currentValue = this.valueSource.asObservable();
   currentLoadingValue = this.loadingValueSource.asObservable();

   initializeDashboardItems() {
      this.repository.getAllLightbulbDashboardItems().subscribe(
         (result) => {
            console.log("result", result);
            this.valueSource.next(result);
            this.setLoading(false);
         }
      )
   }
   /**
    * @returns Observable
    */
   getAllLightbulbDashboardItems(building?: string, apartment?: string, manufacturer?: string, bulbType?:
string) {
      let params = new HttpParams();

      if (building && building.length > 0) {
         params = params.append('building', building);
      }

      if (apartment && apartment.length > 0) {
         params = params.append('apt', apartment);
      }

      if (manufacturer && manufacturer.length > 0) {
         params = params.append('manuf', manufacturer);
      }

      if (bulbType && bulbType.length > 0) {
         params = params.append('bulbtype', bulbType);
      }

      this.repository.getAllLightbulbDashboardItems(params).subscribe(
         (result) => {
            console.log("result", result);
            this.valueSource.next(result);
            this.setLoading(false);
         }
      )
   }
```

```
    setLoading(value: boolean) {
        this.loadingValueSource.next(value);
    }
}

"FILENAME: C:\dev\c3\rawRepos\c3iot-web\src\app\services\lightbulb-detail-autocomplete.service.ts"

import { Injectable } from '@angular/core';
import { Observable } from 'rxjs';
import { publishLast, refCount } from 'rxjs/operators';
import { HttpClient, HttpHeaders } from '@angular/common/http';
import { LightbulbDetailAutocompleteRepository } from '../repositories/lightbulb-detail-
autocomplete.repository';
import { SmartBulbDetailAutocompleteDTO } from '../dto/smart-bulb-detail-autocomplete.dto';

@Injectable()
export class LightbulbDetailAutocompleteService {
    constructor (private repository: LightbulbDetailAutocompleteRepository) {
        this.subject$ = this.repository.getAllLightbulbDetailAutocompleteItems();
    }

    private subject$ = new Observable<any>();

    getLightbulbDetailAutocompleteItems(): Observable<any> {
        return this.subject$;
    }
    /**
     * @returns Observable
     */
    private getAllLightbulbDetailAutocompleteItems(): Observable<SmartBulbDetailAutocompleteDTO> {
        return this.repository.getAllLightbulbDetailAutocompleteItems()
            .pipe(
                publishLast(),
                refCount()
            );
    }
}

"FILENAME: C:\dev\c3\rawRepos\c3iot-web\src\app\services\lightbulb-detail.service.ts"

import { Injectable } from '@angular/core';
import { Observable, BehaviorSubject } from 'rxjs';
import { publishLast, publishReplay, publishBehavior, refCount, publish } from 'rxjs/operators';
import { LightbulbDetailRepository } from '../repositories/lightbulb-detail.repository';
import { SmartBulbDetailDTO } from '../dto/smart-bulb-detail.dto';
import { HttpParams } from '@angular/common/http';

@Injectable()
export class LightbulbDetailService {
    constructor (private repository: LightbulbDetailRepository) {
    }

    private valueSource = new BehaviorSubject<SmartBulbDetailDTO>(new SmartBulbDetailDTO());
    private chartIntervalValueSource = new BehaviorSubject<number>(168);
    private loadingValueSource =  new BehaviorSubject<boolean>(true);
```

```
    currentValue = this.valueSource.asObservable();
    currentChartIntervalValue = this.chartIntervalValueSource.asObservable();
    currentLoadingValue = this.loadingValueSource.asObservable();
    /**
     * @returns Observable
     */
    getAllLightbulbDetailItems(smartBulbName: string, timestamp?: string, daysBack?: string) {
        let params = new HttpParams();

        params = params.append('smartbulbname', smartBulbName);

        if (timestamp) {
            params = params.append('timestamp', timestamp);
        }

        if (daysBack) {
            params = params.append('daysback', daysBack);
        }
        this.repository.getAllLightbulbDetailItems(params).subscribe(
            (result) => {
                console.log("result", result);
                this.valueSource.next(result);
                this.setLoading(false);
            }
        )
    }

    setChartInterval(intervalInHours: number) {
        this.chartIntervalValueSource.next(intervalInHours);
    }

    setLoading(value: boolean) {
        this.loadingValueSource.next(value);
    }
}

"FILENAME: C:\dev\c3\rawRepos\c3iot-web\src\app\services\services.module.ts"

import { NgModule } from '@angular/core';
import { LightbulbDashboardService} from './lightbulb-dashboard.service';
import { LightbulbDashboardAutocompleteService} from './lightbulb-dashboard-autocomplete.service';
import { LightbulbDetailService } from './lightbulb-detail.service';
import { LightbulbDetailAutocompleteService} from './lightbulb-detail-autocomplete.service';

@NgModule({
    providers: [
        LightbulbDashboardService,
        LightbulbDashboardAutocompleteService,
        LightbulbDetailService,
        LightbulbDetailAutocompleteService,
    ],
    imports: [

    ]
  })
```

```
 export class ServicesModule {}
```

"FILENAME: C:\dev\c3\rawRepos\c3iot-web\src\app\styles\material-icons.scss"

```
@font-face {
   font-family: 'Material Icons';
   font-style: normal;
   font-weight: 400;
   src: url('/assets/fonts/MaterialIcons-Regular.eot'); /* For IE6-8 */
   src: local('Material Icons'),
     local('MaterialIcons-Regular'),
     url('/assets/fonts/MaterialIcons-Regular.woff2') format('woff2'),
     url('/assets/fonts/MaterialIcons-Regular.woff') format('woff'),
     url('/assets/fonts/MaterialIcons-Regular.ttf') format('truetype');
 }

 .material-icons {
   font-family: 'Material Icons';
   font-weight: normal;
   font-style: normal;
   font-size: 24px;  /* Preferred icon size */
   display: inline-block;
   line-height: 1;
   text-transform: none;
   letter-spacing: normal;
   word-wrap: normal;
   white-space: nowrap;
   direction: ltr;

   /* Support for all WebKit browsers. */
   -webkit-font-smoothing: antialiased;
   /* Support for Safari and Chrome. */
   text-rendering: optimizeLegibility;

   /* Support for Firefox. */
   -moz-osx-font-smoothing: grayscale;

   /* Support for IE. */
   font-feature-settings: 'liga';
 }
```

"FILENAME: C:\dev\c3\rawRepos\c3iot-web\src\app\styles_colors.scss"

```
$colors-primary: (
   primary: #001E78,
   danger: #C80000,
   warn: #FFB428,
   accent: #00B4DC,
   bright-blue: #4169B9,
   orange: #FA8214,
   success: #75B809
);

$colors-secondary: (
   black: #000000,
   teal: #2DA087,
   yellow: #FFFF55,
```

```
    dark-blue: #0F0F55,
    blue: #498DE3,
    light-blue: #5A78FF,
    dark-grey: #202834,
    medium-dark-grey: rgba(0,0,0,.6),
    grey: #9D9B9C,
    medium-light-grey: rgba(0,0,0,.4),
    light-grey: #F5F7F8,
    white: #FFFFFF,
    red: #FAE6E6
);
"FILENAME: C:\dev\c3\rawRepos\c3iot-web\src\app\styles\_styles.scss"

@import "_colors";

@font-face {
    font-family: 'gotham-light';
    src: url('/assets/fonts/Gotham-Light.eot');
    src: url('/assets/fonts/Gotham-Light.woff2') format('woff2'),
         url('/assets/fonts/Gotham-Light.woff') format('woff'),
         url('/assets/fonts/Gotham-Light.ttf') format('truetype'),
         url('/assets/fonts/Gotham-Light.svg#Gotham-Light') format('svg'),
         url('/assets/fonts/Gotham-Light.eot?#iefix') format('embedded-opentype');
    font-weight: normal;
    font-style: normal;
  }

  @font-face {
    font-family: 'gotham-black';
    src: url('/assets/fonts/Gotham-Black.eot');
    src: url('/assets/fonts/Gotham-Black.woff2') format('woff2'),
         url('/assets/fonts/Gotham-Black.woff') format('woff'),
         url('/assets/fonts/Gotham-Black.ttf') format('truetype'),
         url('/assets/fonts/Gotham-Black.svg#Gotham-Light') format('svg'),
         url('/assets/fonts/Gotham-Black.eot?#iefix') format('embedded-opentype');
    font-weight: normal;
    font-style: normal;
  }

  @font-face {
    font-family: 'gotham-bold';
    src: url('/assets/fonts/Gotham-Bold.eot');
    src: url('/assets/fonts/Gotham-Bold.woff2') format('woff2'),
         url('/assets/fonts/Gotham-Bold.woff') format('woff'),
         url('/assets/fonts/Gotham-Bold.ttf') format('truetype'),
         url('/assets/fonts/Gotham-Bold.svg#Gotham-Light') format('svg'),
         url('/assets/fonts/Gotham-Bold.eot?#iefix') format('embedded-opentype');
    font-weight: normal;
    font-style: normal;
  }

h1 {
    font-family: 'gotham-light', Arial, sans-serif;
    font-weight: bold;
    font-size: 22px;
    line-height: 30px;
```

```
    color: map-get($colors-secondary, dark-blue);
}

h2 {
    font-family: 'gotham-light', Arial, sans-serif;
    font-weight: bold;
    font-size: 18px;
    line-height: 26px;
    color: map-get($colors-secondary, dark-blue);
}

h3 {
    font-family: 'gotham-light', Arial, sans-serif;
    font-weight: bold;
    font-size: 14px;
    line-height: 20px;
    color: map-get($colors-secondary, dark-blue);
}

h4 {
    font-family: 'gotham-bold', Arial, sans-serif;
    font-weight: bold;
    font-size: 12px;
    line-height: 18px;
    color: map-get($colors-secondary, medium-dark-grey);
}

body {
    font-family: 'gotham-light', Arial, sans-serif;
    font-weight: normal;
    font-size: 12px;
    line-height: 18px;
    color: map-get($colors-secondary, dark-blue);
}

a {
    font-family: 'gotham-light', Arial, sans-serif;
    font-weight: normal;
    text-decoration: underline;
    font-size: 12px;
    line-height: 18px;
    color: map-get($colors-secondary, light-blue);
}

button {
    font-family: 'gotham-light', Arial, sans-serif;
    font-weight: normal;
    font-size: 14px;
    line-height: 20px;
    color: map-get($colors-secondary, white);
}

label {
    font-family: 'gotham-light', Arial, sans-serif;
    font-weight: normal;
    font-size: 12px;
```

```
    line-height: 18px;
    color: map-get($colors-secondary, dark-blue);
}

.small-text {
    font-family: 'gotham-light', Arial, sans-serif;
    font-weight: normal;
    font-size: 10px;
    line-height: 14px;
    color: map-get($colors-secondary, dark-blue);
}

.picture-subtitles {
    font-family: 'gotham-light', Arial, sans-serif;
    font-weight: normal;
    font-size: 11px;
    line-height: 16px;
    color: map-get($colors-secondary, light-grey);
}

.error-message {
    font-family: 'gotham-light', Arial, sans-serif;
    font-weight: bold;
    font-size: 12px;
    line-height: 18px;
    color: map-get($colors-secondary, red);
}

.confirmation-message {
    font-family: 'gotham-light', Arial, sans-serif;
    font-weight: bold;
    font-size: 12px;
    line-height: 18px;
    color: map-get($colors-secondary, teal);
}

.nav-item {
    font-family: 'gotham-light', Arial, sans-serif;
    font-weight: normal;
    font-size: 18px;
    color: map-get($colors-secondary, white);
}

.nav-item.active {
    font-family: 'gotham-light', Arial, sans-serif;
    font-weight: normal;
    font-size: 18px;
    color: map-get($colors-secondary, yellow);
}

.material-icons
{
    &.color_green {
        color: map-get($colors-primary, success);
    }
    &.color_red {
```

```
      color: map-get($colors-primary, danger);
    }
    &.color_yellow {
      color: map-get($colors-primary, warn);
    }
}
```

"FILENAME: C:\dev\c3\rawRepos\c3iot-web\src\environments\environment.prod.ts"

```
export const environment = {
  production: true
};
```

"FILENAME: C:\dev\c3\rawRepos\c3iot-web\src\environments\environment.ts"

```
// This file can be replaced during build by using the `fileReplacements` array.
// `ng build --prod` replaces `environment.ts` with `environment.prod.ts`.
// The list of file replacements can be found in `angular.json`.

export const environment = {
  production: false,
  apiEndpoint: 'https://cdd4imb26g.execute-api.us-east-1.amazonaws.com',
  apiPrefix: 'api',
  apiStage: 'LATEST',
  apiKey: 'MUbQdkj5aSkHgLQi8Ytd4jOgSVmc18u6mIqttzk2'
};

/*
 * For easier debugging in development mode, you can import the following file
 * to ignore zone related error stack frames such as `zone.run`, `zoneDelegate.invokeTask`.
 *
 * This import should be commented out in production mode because it will have a negative impact
 * on performance if an error is thrown.
 */
// import 'zone.js/dist/zone-error';  // Included with Angular CLI.
```

"FILENAME: C:\dev\c3\rawRepos\c3iot-api\functions\source\aws-helpers\index.js"

```
var AWS = require('aws-sdk');
var async = require('async');

let decrypted;
module.exports = {
        getDecryptedKmsKey: function(awsRegion, encryptedKey) {
                return new Promise((resolve, reject) => {
                        if (decrypted) {
                                resolve(decrypted);
                        } else {
                                const kms = new AWS.KMS({region: awsRegion});
                                kms.decrypt({ CiphertextBlob: new Buffer(encryptedKey, 'base64') },
(err, data) => {
                                        if (err) {
                                                console.log('Decrypt error:', err);
                                                reject(err);
                                        }
```

```
                                        decrypted = data.Plaintext.toString('ascii');
                                        resolve(decrypted);
                                });
                        }
                });
        },

        sendDataToFirehose: function (awsRegion, firehoseStream, data, callback) {
                var firehose = new AWS.Firehose({region: awsRegion});
                var params = {
                   DeliveryStreamName: firehoseStream,
                   Record: {Data: data }

                };
                firehose.putRecord(params, function(err, data) {
                   if (err){
                      console.log("Failed to insert data onto Firehose: ", JSON.stringify(err));
                      callback(err);
                   }
                   else{
                                console.log("Successfully placed data in onto the following firehose
stream: " + firehoseStream);
                                callback(null);
                   }
                });
        },
        sendDataToKinesisAsync: function (awsRegion, outputStream, partitionKey, data) {
           var kinesis = new AWS.Kinesis({region: awsRegion});
           var params = {
             Data: data,
             PartitionKey:  partitionKey,
             StreamName: outputStream
                };
                console.log("Putting data to Kinesis: "+data);
           return new Promise((resolve, reject) => {
                        kinesis.putRecord(params, function(err, data) {
                                if (err) {
                                        console.log("Failed to insert data onto Kinesis: ",
JSON.stringify(err));
                                        reject(err);
                                }
                                console.log("Successfully placed data onto the following stream: " +
outputStream);
                                resolve("Kinesis put success.");
                        })
                });
        },
        sendDataToS3: function (awsRegion, bucketName, key, data, callback) {
           var s3 = new AWS.S3({region: awsRegion});
                var params = {
                        Body: data,
                        Bucket: bucketName,
                        Key: key,
                        ServerSideEncryption: "AES256"
                };
                s3.putObject(params, function(err, data) {
```

```
                        if(err){
                    console.log("Failed to insert data into S3: ", JSON.stringify(err));
                    callback(err);
                }
                else{
                                        console.log("Successfully placed data in S3 at s3://" + bucketName +
"/" + key);
                                        callback(null);
                }
                    });
        },

        listAllS3Objects: function (awsRegion, bucketName, prefix, allS3Objects, token, callback)
        {
        var s3 = new AWS.S3({region: awsRegion});

                    var opts = {
                        Bucket: bucketName,
                        Prefix: prefix
                    };
                    if(token) opts.ContinuationToken = token;

                    s3.listObjectsV2(opts, function(err, data){
                        if (err) console.log(err, err.stack);
                        else {
                           allS3Objects = allS3Objects.concat(data.Contents);

                           if(data.IsTruncated)
                              listAllS3Objects(s3, bucketName, prefix, allS3Objects,
data.NextContinuationToken, callback);
                           else
                              callback(allS3Objects);
                        }
                    });
        },

        readAllS3Objects: function (awsRegion, bucketName, allS3Objects, callback)
        {
        var s3 = new AWS.S3({region: awsRegion});

                    var readS3Object = this.readS3Object;
           var readAsync = function(s3Object, callback){
              readS3Object(awsRegion, bucketName, s3Object.Key, callback);
           }

           async.concat(allS3Objects, readAsync, function(err, objects){
             if (err){
                console.log(err, err.stack);
                callback(err);
             }else{
                callback(null, objects);
             }
           });
        },

        readS3Object: function (awsRegion, bucketName, key, callback)
```

```
{
        var s3 = new AWS.S3({region: awsRegion});

   s3.getObject({ Bucket: bucketName, Key: key }, function(err, data)
   {
      if (err){
         console.log(err, err.stack);
         callback(err);
      }else{
         callback(null, data.Body.toString());
      }
   });
},

sendDataToSNS: function (awsRegion, message, subject, topicArn){

        var sns = new AWS.SNS({region:awsRegion});

        // Create publish parameters
        var params = {
                Message: message, /* required */
                TopicArn: topicArn,
                Subject: subject
        };

        return new Promise((resolve, reject) => {
                sns.publish(params, (err, data) => {
                        if (err) {
                                console.log("Failed to publish to SNS: ", JSON.stringify(err));
                                reject(err);
                        }
                        console.log("MessageID is " + data.MessageId);
                        resolve(data);
                })
        });
},

putItemInDynamoDB: function(awsRegion, tableName, item) {
        return new Promise((resolve,reject) => {
                var documentClient = new AWS.DynamoDB.DocumentClient({region:
awsRegion});
                var params = {
                        TableName: tableName,
                        Item: item
                };
                documentClient.put(params, function(err, data) {
             if (err){
                console.log(err, err.stack);
                reject(err);
             } else {
                resolve(data);
             }
                });
        });
},
```

```
	updateItemInDynamoDB: function(awsRegion, params) {
		return new Promise((resolve,reject) => {
			var documentClient = new AWS.DynamoDB.DocumentClient({region:
awsRegion});

			documentClient.update(params, function(err, data) {
				if (err) {
					console.log(err);
					reject(err);
				}
				else {
					resolve(data);
				}
			});
		});
	},

	getItemInDynamoDB: function(awsRegion, params) {
		return new Promise((resolve,reject) => {
			var documentClient = new AWS.DynamoDB.DocumentClient({region:
awsRegion});

			documentClient.get(params, function(err, data) {
				if (err) {
					console.log(err);
					reject(err);
				}
				else {
					resolve(data);
					return data;
				}
			});
		});
	},

	queryDynamoDB: function(awsRegion, params) {
		return new Promise((resolve,reject) => {
			let allData = [];
			var documentClient = new AWS.DynamoDB.DocumentClient({region:
awsRegion});
			documentClient.query(params, function paginateCallback(err, data) {
		if (err){
		   console.log(err, err.stack);
		   reject(err);
		} else {
					allData.push(data);
		   if(data.LastEvaluatedKey) {
						params.ExclusiveStartKey = data.LastEvaluatedKey;
						documentClient.query(params, paginateCallback);
					} else {
						const result = allData.reduce((accumulator,
currentValue) => accumulator.concat(currentValue.Items), []);
						resolve(result);
					};
				}
			});
```

```
                    });
            }
}
```

"FILENAME: C:\dev\c3\rawRepos\c3iot-api\functions\source\aws-helpers\package.json"

```
{
   "name": "aws-helpers",
   "version": "0.0.1",
   "private": true,
   "main": "index.js",
   "description": "A library of aws helper functions",
   "license": "MIT",
   "devDependencies": {
      "aws-sdk": "^2.331.0"
   },
   "dependencies": {
      "async": "~2.6.0"
   }
}
```

"FILENAME: C:\dev\c3\rawRepos\c3iot-api\functions\source\dashboard-api-lambda\.eslintrc.js"

```
module.exports = {
  env: {
    es6: true,
    node: true,
    mocha: true,
    jest: true
  },
  extends: ['airbnb-base'],
  rules: {
    'arrow-parens': ['error', 'as-needed'],
    'comma-dangle': ['error', 'never'],
    'function-paren-newline': 'off',
    'no-underscore-dangle': 'off',
    'prefer-destructuring': 'off',
    'no-console': 'off',
    'import/no-extraneous-dependencies': [
      'warn',
      { devDependencies: false }
    ]
  },
  overrides: [
    {
      files: '**/*.spec.js',
      rules: {
        'no-unused-expressions': 'off',
        'prefer-arrow-callback': 'off',
        'import/no-extraneous-dependencies': [
          'error',
          { devDependencies: true }
        ]
      }
    }
  ]
```

```
};

"FILENAME: C:\dev\c3\rawRepos\c3iot-api\functions\source\dashboard-api-lambda\index.js"

const dbHelpers = require('database-helpers');

function DashboardDTO(bulbsAtRisk, bulbsAtRisk24, bulbsSwitchedOn, failuresYTD,
riskDistributionArray, smartBulbInfoArray) {
  return {
    bulbsAtRisk: bulbsAtRisk,
    bulbsAtRisk24: bulbsAtRisk24,
    bulbsSwitchedOn: bulbsSwitchedOn,
    failuresYTD: failuresYTD,
    riskDistribution: riskDistributionArray,
    smartBulbs: smartBulbInfoArray
  };
}

function SmartBulbInfo(bulb) {
  return {
    smartBulbName: bulb.smartBulbName,
    manufacturer: bulb.manufacturer,
    bulbType: bulb.bulbType,
    latitude: bulb.latitude,
    longitude: bulb.longitude,
    fixtureName: bulb.fixtureName,
    apartmentName: bulb.apartmentName,
    buildingName: bulb.buildingName,
    startDate: bulb.startDate,
    riskScore: bulb.prediction
  };
}

async function getBulbsAtRisk(smartBulbInfoArray) {
  let count = 0;
  smartBulbInfoArray.forEach(record => {
    if (record.riskScore > 50) {
      count++;
    }
  });

  return count;
}

async function getBulbsAtRisk24(building, apartment, manufacturer, bulbType) {
  let sqlParmArray = [];

  const pool = await dbHelpers.getPool();
  let queryString ='SELECT COUNT(prediction.prediction) AS currentRisk,
COUNT(previousRisk.prediction) AS previousRisk'
  + ' FROM smart_bulb as bulb'
  + ' LEFT JOIN smart_bulb_fixture as sb_fix on sb_fix.smart_bulb_name = bulb.`name`'
  + ' LEFT JOIN fixture as fix on sb_fix.fixture_name = fix.`name`'
  + ' LEFT JOIN apartment as apt on fix.apartment_name = apt.`name`'
  + ' LEFT JOIN ('
  + ' SELECT t.smart_bulb_name, t.time_stamp, t.prediction'
```

```
  + ' FROM smart_bulb_prediction t'
  + ' INNER JOIN ('
  + ' SELECT smart_bulb_name, MAX(time_stamp) AS MaxDate, prediction'
  + ' FROM smart_bulb_prediction'
  + ' WHERE prediction > .5'
  + ' GROUP BY smart_bulb_name'
  + ' ) t2 ON t.smart_bulb_name = t2.smart_bulb_name AND t.time_stamp = t2.MaxDate'
  + ' GROUP BY smart_bulb_name'
  + ' ) AS prediction on prediction.smart_bulb_name = bulb.`name`'
  + ' LEFT JOIN('
  + ' SELECT t.smart_bulb_name, t.time_stamp, t.prediction'
  + ' FROM smart_bulb_prediction t '
  + ' INNER JOIN ('
  + ' SELECT smart_bulb_name, DATE_ADD(MAX(time_stamp), INTERVAL - 1 DAY) AS MaxDate,
prediction'
  + ' FROM smart_bulb_prediction'
  + ' WHERE prediction > .5'
  + ' GROUP BY smart_bulb_name'
  + ' ) t2 ON t.smart_bulb_name = t2.smart_bulb_name AND t.time_stamp = t2.MaxDate'
  + ' GROUP BY smart_bulb_name'
  + ' ) AS previousRisk on previousRisk.smart_bulb_name = bulb.`name`'

  if (building.length > 0 || apartment.length > 0 || manufacturer.length > 0 || bulbType.length > 0) {
    queryString += ' WHERE';
    generateQueryFilter(building, apartment, manufacturer, bulbType, (qs, sqlParms) => {
      queryString += qs;
      for (let parm of sqlParms) {
        sqlParmArray.push(parm);
      }
    });
  }

  const queryResult = await pool.query(queryString, sqlParmArray);

  return queryResult[0].currentRisk-queryResult[0].previousRisk;
}

async function getRiskDistribution(smartBulbInfoArray) {
  let riskDistributionArray = [];
  let count10 = 0;
  let count20 = 0;
  let count30 = 0;
  let count40 = 0;
  let count50 = 0;
  let count60 = 0;
  let count70 = 0;
  let count80 = 0;
  let count90 = 0;
  let count100 = 0;

  smartBulbInfoArray.forEach(record => {
    if (record.riskScore <= 10) {
      count10++;
    } else if (record.riskScore <= 20) {
      count20++;
    } else if (record.riskScore <= 30) {
```

```
      count30++;
    } else if (record.riskScore <= 40) {
      count40++;
    } else if (record.riskScore <= 50) {
      count50++;
    } else if (record.riskScore <= 60) {
      count60++;
    } else if (record.riskScore <= 70) {
      count70++;
    } else if (record.riskScore <= 80) {
      count80++;
    } else if (record.riskScore <= 90) {
      count90++;
    } else {
      count100++;
    }
  })

  riskDistributionArray.push(count10);
  riskDistributionArray.push(count20);
  riskDistributionArray.push(count30);
  riskDistributionArray.push(count40);
  riskDistributionArray.push(count50);
  riskDistributionArray.push(count60);
  riskDistributionArray.push(count70);
  riskDistributionArray.push(count80);
  riskDistributionArray.push(count90);
  riskDistributionArray.push(count100);

  return riskDistributionArray;
}

async function getBulbsSwitchedOn(smartBulbInfoArray) {
  let count = 0;
  smartBulbInfoArray.forEach(record => {
    if (record.status) {
      count++;
    }
  });

  return count;
}

async function generateQueryFilter(building, apartment, manufacturer, bulbType, callback) {
  let queryString = '';
  let sqlParmArray = [];

  if (building.length > 0) {
    queryString += ' apt.building_name = ? AND';
    sqlParmArray.push(building);
  }
  if (apartment.length > 0) {
    queryString += ' apt.`name` = ? AND';
    sqlParmArray.push(apartment);
  }
  if (manufacturer.length > 0) {
```

```
    queryString += ' bulb.manufacturer = ? AND';
    sqlParmArray.push(manufacturer);
  }
  if (bulbType.length > 0) {
    queryString += ' bulb.bulb_type = ? AND';
    sqlParmArray.push(bulbType);
  }
  queryString = queryString.slice(0, -4);

  callback(queryString, sqlParmArray);
}

async function getFailuresYTD(building, apartment, manufacturer, bulbType) {
  const pool = await dbHelpers.getPool();
  let sqlParmArray = ['DEFECTIVE'];

  let queryString = 'SELECT COUNT(sb_event.event_code) AS eventCount'
    + ' FROM smart_bulb as bulb'
    + ' LEFT JOIN smart_bulb_fixture as sb_fix on sb_fix.smart_bulb_name = bulb.`name`'
    + ' LEFT JOIN fixture as fix on sb_fix.fixture_name = fix.`name`'
    + ' LEFT JOIN apartment as apt on fix.apartment_name = apt.`name`'
    + ' INNER JOIN smart_bulb_event as sb_event on sb_event.smart_bulb_name = bulb.`name`'
    + ' WHERE sb_event.event_type = ?';

  if (building.length > 0 || apartment.length > 0 || manufacturer.length > 0 || bulbType.length > 0) {
    queryString += ' AND';
    generateQueryFilter(building, apartment, manufacturer, bulbType, (qs, sqlParms) => {
      queryString += qs;
      for (let parm of sqlParms) {
        sqlParmArray.push(parm);
      }
    });
  }

  console.log("getFailuresYTD SQL query string: ", queryString);

  const queryResult = await pool.query(queryString, sqlParmArray);

  return queryResult[0].eventCount;
}

async function getLightbulbData(building, apartment, manufacturer, bulbType) {
  let smartBulbInfoArray = [];
  let sqlParmArray = [];

  const pool = await dbHelpers.getPool();
  let queryString = 'SELECT bulb.`name` as smartBulbName, '
    + 'bulb.manufacturer,'
    + 'bulb.bulb_type as bulbType,'
    + 'bulb.latitude,'
    + 'bulb.longitude,'
    + 'bulb.start_date as startDate,'
    + 'fix.`name` as fixtureName,'
    + 'apt.`name` as apartmentName,'
    + 'apt.building_name as buildingName,'
    + ' prediction.prediction'
```

```
    + ' FROM smart_bulb as bulb'
    + ' LEFT JOIN ('
    + ' SELECT t.smart_bulb_name, t.time_stamp, t.prediction'
    + ' FROM smart_bulb_prediction t'
    + ' INNER JOIN ('
    + ' SELECT smart_bulb_name, MAX(time_stamp) as MaxDate, prediction'
    + ' FROM smart_bulb_prediction'
    + ' GROUP BY smart_bulb_name'
    + ' ) t2 on t.smart_bulb_name = t2.smart_bulb_name AND t.time_stamp = t2.MaxDate'
    + ' GROUP BY smart_bulb_name'
    + ' ) AS prediction ON prediction.smart_bulb_name= bulb.`name`'
    + ' LEFT JOIN smart_bulb_fixture as sb_fix on sb_fix.smart_bulb_name = bulb.`name`'
    + ' LEFT JOIN fixture as fix on sb_fix.fixture_name = fix.`name`'
    + ' LEFT JOIN apartment as apt on fix.apartment_name = apt.`name`';

  if (building.length > 0 || apartment.length > 0 || manufacturer.length > 0 || bulbType.length > 0) {
    queryString += ' WHERE';
    generateQueryFilter(building, apartment, manufacturer, bulbType, (qs, sqlParms) => {
      queryString += qs;
      for (let parm of sqlParms) {
        sqlParmArray.push(parm);
      }
    });
  }

  console.log("getLightbulbData SQL query string: ", queryString);

  const queryResult = await pool.query(queryString, sqlParmArray);

  queryResult.forEach(
    record => {
      if (record && record.prediction && record.prediction > 0) {
        record.prediction = Math.round(record.prediction * 100);
      }
      smartBulbInfoArray.push(new SmartBulbInfo(record));
    }
  );

  return smartBulbInfoArray;
}

exports.handler = async (event) => {
  console.log("event (query string parms): ", event);
  const building = event.building;
  const apartment = event.apt;
  const manufacturer = event.manuf;
  const bulbType = event.bulbtype;

  let lightbulbData = await getLightbulbData(building, apartment, manufacturer, bulbType);
  let bulbsAtRisk = await getBulbsAtRisk(lightbulbData);
  let bulbsAtRisk24 = await getBulbsAtRisk24(building, apartment, manufacturer, bulbType);
  let failuresYTD = await getFailuresYTD(building, apartment, manufacturer, bulbType);
  let bulbsSwitchedOn = await getBulbsSwitchedOn(lightbulbData);
  let riskDistributionArray = await getRiskDistribution(lightbulbData);
```

```
  return new DashboardDTO(bulbsAtRisk, bulbsAtRisk24, bulbsSwitchedOn, failuresYTD,
riskDistributionArray, lightbulbData);
};
```

"FILENAME: C:\dev\c3\rawRepos\c3iot-api\functions\source\dashboard-api-lambda\package.json"

```
{
   "name": "dashboard-lambda",
   "version": "0.0.1",
   "description": "A lambda that retrieves data for the lightbulb dashboard",
   "license": "MIT",
   "scripts": {
      "test": "nyc --check-coverage --lines 80 --functions 80 --branches 80 --reporter=cobertura mocha -R
mocha-multi-reporters --recursive test",
      "lint": "eslint . --fix",
      "prettier": "prettier --single-quote --write '**/*.js'"
   },
   "devDependencies": {
      "aws-lambda-mock-context": "^3.2.1",
      "chai": "^4.2.0",
      "chai-as-promised": "7.1.1",
      "eslint": "^5.6.1",
      "eslint-config-airbnb-base": "^13.1.0",
      "eslint-plugin-import": "^2.14.0",
      "expect.js": "^0.3.1",
      "mocha": "^5.2.0",
      "mocha-multi-reporters": "^1.1.7",
      "nyc": "^13.0.1",
      "prettier": "^1.6.1",
      "sinon": "^6.3.5"
   },
   "dependencies": {
      "async": "~2.6.0",
      "aws-helpers": "file:../aws-helpers",
      "aws-sdk": "^2.344.0",
      "database-helpers": "file:../database-helpers"
   },
   "private": true
}
```

"FILENAME: C:\dev\c3\rawRepos\c3iot-api\functions\source\dashboard-api-lambda\test\dashboard-api.spec.js"

```
const sinon = require('sinon');
const chai = require('chai');
const chaiAsPromised = require('chai-as-promised');
// const context = require('aws-lambda-mock-context');
const dashboardLambda = require('../index.js');
const dbHelpers = require('../../database-helpers');

chai.use(chaiAsPromised);
const { assert } = chai;
const { expect } = chai;

describe('Dashboard API Lambda Test', () => {
  const sandbox = sinon.createSandbox();
```

```
  beforeEach(() => {
  });

  afterEach(() => {
    sandbox.restore();
  });

  it('get bulbs at risk in the last 24 hours', async () => {
    const bulbsAtRisk24 = await dashboardLambda.getBulbsAtRisk24();
    console.log("bulbs", bulbsAtRisk24);
    assert.equal(bulbsAtRisk24, 3);
  });
});
```

"FILENAME: C:\dev\c3\rawRepos\c3iot-api\functions\source\dashboard-api-lambda\test\model.spec.js"

```
const chai = require('chai');
const chaiAsPromised = require('chai-as-promised');
// const context = require('aws-lambda-mock-context');
const dashboardLambda = require('../index.js');

chai.use(chaiAsPromised);
// const { expect } = chai;

describe('Dashboard API Lambda Test', () => {
  describe('dashboardGet', () => {
    it('test a particular thing', () => {
      // call into method to do some stuff
      dashboardLambda.handler();

      /*
      ctx.Promise.then(message => {
        expect(message).to.eq('Success');
      }).catch(err => {
        expect().fail(err, 'Expected success');
      });
      */
    });
  });
});
```

"FILENAME: C:\dev\c3\rawRepos\c3iot-api\functions\source\dashboard-autocomplete-api-lambda\.eslintrc.js"

```
module.exports = {
  env: {
    es6: true,
    node: true,
    mocha: true,
    jest: true
  },
  extends: ['airbnb-base'],
  rules: {
    'arrow-parens': ['error', 'as-needed'],
    'comma-dangle': ['error', 'never'],
```

```
    'function-paren-newline': 'off',
    'no-underscore-dangle': 'off',
    'prefer-destructuring': 'off',
    'no-console': 'off',
    'import/no-extraneous-dependencies': [
      'warn',
      { devDependencies: false }
    ]
  },
  overrides: [
    {
      files: '**/*.spec.js',
      rules: {
        'no-unused-expressions': 'off',
        'prefer-arrow-callback': 'off',
        'import/no-extraneous-dependencies': [
          'error',
          { devDependencies: true }
        ]
      }
    }
  ]
};

"FILENAME: C:\dev\c3\rawRepos\c3iot-api\functions\source\dashboard-autocomplete-api-
lambda\index.js"

const dbHelpers = require('database-helpers');

function DashboardAutocompleteDTO(buildings, apartments, manufacturers, bulbTypes) {
  return {
    buildings: buildings,
    apartments: apartments,
    manufacturers: manufacturers,
    bulbTypes: bulbTypes
  };
}

async function getBuildings() {
  let buildingsArray = [];

  const pool = await dbHelpers.getPool();
  let queryString = 'SELECT building.`name` FROM building'

  const queryResult = await pool.query(queryString);

  queryResult.forEach(
    record => {
      buildingsArray.push(record.name);
    }
  );

  return buildingsArray;
}

async function getApartments() {
```

```
  let apartmentsArray = [];

  const pool = await dbHelpers.getPool();
  let queryString = 'SELECT apartment.`name` FROM apartment'

  const queryResult = await pool.query(queryString);

  queryResult.forEach(
    record => {
      apartmentsArray.push(record.name);
    }
  );

  return apartmentsArray;
}

async function getManufacturers() {
  let manufacturersArray = [];

  const pool = await dbHelpers.getPool();
  let queryString = 'SELECT DISTINCT smart_bulb.manufacturer FROM smart_bulb'

  const queryResult = await pool.query(queryString);

  queryResult.forEach(
    record => {
      manufacturersArray.push(record.manufacturer);
    }
  );

  return manufacturersArray;
}

async function getBulbTypes() {
  let bulbTypesArray = [];

  const pool = await dbHelpers.getPool();
  let queryString = 'SELECT DISTINCT smart_bulb.bulb_type FROM smart_bulb'

  const queryResult = await pool.query(queryString);

  queryResult.forEach(
    record => {
      bulbTypesArray.push(record.bulb_type);
    }
  );

  return bulbTypesArray;
}

exports.handler = async (event) => {
  let buildings = await getBuildings();
  let apartments = await getApartments();
  let manufacturers = await getManufacturers();
  let bulbTypes = await getBulbTypes();
```

```
  return new DashboardAutocompleteDTO(buildings, apartments, manufacturers, bulbTypes);
};
```

"FILENAME: C:\dev\c3\rawRepos\c3iot-api\functions\source\dashboard-autocomplete-api-lambda\package.json"

```
{
   "name": "dashboard-lambda",
   "version": "0.0.1",
   "description": "A lambda that retrieves data for the lightbulb dashboard",
   "license": "MIT",
   "scripts": {
      "test": "nyc --check-coverage --lines 80 --functions 80 --branches 80 --reporter=cobertura mocha -R mocha-multi-reporters --recursive test",
      "lint": "eslint . --fix",
      "prettier": "prettier --single-quote --write '**/*.js'"
   },
   "devDependencies": {
      "aws-lambda-mock-context": "^3.2.1",
      "chai": "^4.2.0",
      "chai-as-promised": "7.1.1",
      "eslint": "^5.6.1",
      "eslint-config-airbnb-base": "^13.1.0",
      "eslint-plugin-import": "^2.14.0",
      "expect.js": "^0.3.1",
      "mocha": "^5.2.0",
      "mocha-multi-reporters": "^1.1.7",
      "nyc": "^13.0.1",
      "prettier": "^1.6.1",
      "sinon": "^6.3.5"
   },
   "dependencies": {
      "async": "~2.6.0",
      "aws-helpers": "file:../aws-helpers",
      "aws-sdk": "^2.344.0",
      "database-helpers": "file:../database-helpers"
   },
   "private": true
}
```

"FILENAME: C:\dev\c3\rawRepos\c3iot-api\functions\source\dashboard-autocomplete-api-lambda\test\dashboard-autocomplete-api.spec.js"

```
const sinon = require('sinon');
const chai = require('chai');
const chaiAsPromised = require('chai-as-promised');
const dbHelpers = require('../../database-helpers');
// const context = require('aws-lambda-mock-context');
const dashboardAutocompleteLambda = require('../index.js');

chai.use(chaiAsPromised);
const { assert } = chai;
const { expect } = chai;

describe('Dashboard API Lambda Test', () => {
  const sandbox = sinon.createSandbox();
```

```
beforeEach(() => {
});

afterEach(() => {
  sandbox.restore();
});

it('get buildings', async () => {
  const callSpy = sinon.spy();

  sandbox.stub(dbHelpers, 'getPool').callsFake(() => ({
    query() {
      console.log("called fake query");
      callSpy();
      return new Promise(resolve => resolve([]));
    }
  }));

  await dashboardAutocompleteLambda.getBuildings();
  expect(callSpy.calledOnce).to.be.true;
});

it('get apartments', async () => {
  const callSpy = sinon.spy();

  sandbox.stub(dbHelpers, 'getPool').callsFake(() => ({
    query() {
      console.log("called fake query");
      callSpy();
      return new Promise(resolve => resolve([]));
    }
  }));

  await dashboardAutocompleteLambda.getApartments();
  expect(callSpy.calledOnce).to.be.true;
});

it('get manufacturers', async () => {
  const callSpy = sinon.spy();

  sandbox.stub(dbHelpers, 'getPool').callsFake(() => ({
    query() {
      console.log("called fake query");
      callSpy();
      return new Promise(resolve => resolve([]));
    }
  }));

  await dashboardAutocompleteLambda.getManufacturers();
  expect(callSpy.calledOnce).to.be.true;
});

it('get bulb types', async () => {
  const callSpy = sinon.spy();
```

```
    sandbox.stub(dbHelpers, 'getPool').callsFake(() => ({
      query() {
        console.log("called fake query");
        callSpy();
        return new Promise(resolve => resolve([]));
      }
    }));

    await dashboardAutocompleteLambda.getBulbTypes();
    expect(callSpy.calledOnce).to.be.true;
  });
});
"FILENAME: C:\dev\c3\rawRepos\c3iot-api\functions\source\dashboard-autocomplete-api-
lambda\test\model.spec.js"

const chai = require('chai');
const chaiAsPromised = require('chai-as-promised');
// const context = require('aws-lambda-mock-context');
const dashboardLambda = require('../index.js');

chai.use(chaiAsPromised);
// const { expect } = chai;

describe('Dashboard API Lambda Test', () => {
  describe('dashboardGet', () => {
    it('test a particular thing', () => {
      // call into method to do some stuff
      dashboardLambda.handler();

      /*
      ctx.Promise.then(message => {
        expect(message).to.eq('Success');
      }).catch(err => {
        expect().fail(err, 'Expected success');
      });
      */
    });
  });
});

"FILENAME: C:\dev\c3\rawRepos\c3iot-api\functions\source\database-helpers\index.js"

const util = require('util')
const mysql = require('mysql')
const awsHelpers = require('aws-helpers');

const AWS_REGION = process.env.AWS_REGION || 'us-east-1';
const encryptedMySQLPass = process.env.MySQLPass || 'test';

let pool;
module.exports = {
    getPool: function() {
        return new Promise((resolve) => {
            if (pool) {
                resolve(pool);
            } else {
```

```
            try {
                awsHelpers.getDecryptedKmsKey(AWS_REGION, encryptedMySQLPass).then(key => {
                    pool = mysql.createPool({
                        connectionLimit: 10,
                        host: process.env.MySQLHost,
                        user: process.env.MySQLUser,
                        password: key,
                        database: process.env.MySQLDB
                    });
                    pool.query = util.promisify(pool.query);
                    resolve(pool);
                }).catch(err => console.log("POOL ERROR: "+err));

            } catch (err) {
                log.error("Error creating DB connection pool");
                reject(err);
            }
        }
    });
  }
};
"FILENAME: C:\dev\c3\rawRepos\c3iot-api\functions\source\database-helpers\package.json"

{
  "name": "database-helpers",
  "version": "0.0.1",
  "private": true,
  "main": "index.js",
  "description": "A library of aws helper functions",
  "license": "MIT",
  "devDependencies": {},
  "dependencies": {
    "async": "~2.6.0",
    "mysql": "^2.16.0",
    "aws-helpers": "file:../aws-helpers"
  }
}

"FILENAME: C:\dev\c3\rawRepos\c3iot-api\functions\source\lightbulb-api-lambda\.eslintrc.js"

module.exports = {
 env: {
  es6: true,
  node: true,
  mocha: true,
  jest: true
 },
 extends: ['airbnb-base'],
 rules: {
  'arrow-parens': ['error', 'as-needed'],
  'comma-dangle': ['error', 'never'],
  'function-paren-newline': 'off',
  'no-underscore-dangle': 'off',
  'prefer-destructuring': 'off',
  'no-console': 'off',
  'import/no-extraneous-dependencies': [
```

```
      'warn',
      { devDependencies: false }
    ]
  },
  overrides: [
    {
      files: '**/*.spec.js',
      rules: {
        'no-unused-expressions': 'off',
        'prefer-arrow-callback': 'off',
        'import/no-extraneous-dependencies': [
          'error',
          { devDependencies: true }
        ]
      }
    }
  ]
};

"FILENAME: C:\dev\c3\rawRepos\c3iot-api\functions\source\lightbulb-api-lambda\index.js"

const awsHelpers = require('aws-helpers');
const { subDays } = require('date-fns');

async function getDynamoDBResults(awsRegion, smartBulbName, daysBack, timestamp) {
  const lowerBoundDate = subDays(timestamp, daysBack).toISOString();
  const params = {
    TableName: 'lightbulb-app-ml-ddb',
    KeyConditionExpression: 'smartBulbName = :hkey AND #timestamp BETWEEN :r1 AND :r2',
    ExpressionAttributeNames: { '#timestamp': 'timestamp' },
    ExpressionAttributeValues: {
      ':hkey': smartBulbName,
      ':r1': lowerBoundDate,
      ':r2': timestamp
    },
    ScanIndexForward: 'false'
  };
  console.log('params',params);

  const results = await awsHelpers.queryDynamoDB(awsRegion, params);

  if (results.length > 0) {
    return results;
  }
  console.log('no results found for this bulb in this date range');
  return [];
}

function SmartBulbDetailDTO(map, table) {
  return {
    map,
    table
  };
}

exports.handler = async event => {
```

```
  const awsRegion = process.env.AWS_REGION;

  let daysBack = 300;
  const date = new Date();
  let timestamp = date.toISOString();

  const timestampArray = [];
  const riskscoreArray = [];
  const statusArray = [];
  const lumensArray = [];
  const voltageArray = [];
  const powerArray = [];
  const tempArray = [];

  const tableArray = [];
  const mapObject = {};

  if (event.daysback) { daysBack = event.daysback; }
  if (event.timestamp) { timestamp = event.timestamp; }

  const results = await getDynamoDBResults(awsRegion, event.smartbulbname, daysBack, timestamp);
  results.forEach(element => {
    const payload = JSON.parse(element.payload);
    if (payload && payload.prediction && payload.prediction>0){
      payload.riskscore = Math.round(payload.prediction * 100);
      }
    else{payload.riskscore='NA'}
    if(Number(payload.status)===1){
      payload.status='On';
    }
    else{payload.status='Off'}

    tableArray.push(payload);

    timestampArray.push(payload.timestamp);
    riskscoreArray.push(payload.riskscore);
    statusArray.push(Number(payload.status));
    lumensArray.push(Number(payload.lumens));
    voltageArray.push(Number(payload.voltage));
    powerArray.push(Number(payload.power));
    tempArray.push(Number(payload.temperature));
  });

  mapObject.timestamp = timestampArray;
  mapObject.riskscore = riskscoreArray;
  mapObject.status = statusArray;
  mapObject.lumens = lumensArray;
  mapObject.voltage = voltageArray;
  mapObject.power = powerArray;
  mapObject.temperature = tempArray;

  return new SmartBulbDetailDTO(mapObject, tableArray);
};
```

"FILENAME: C:\dev\c3\rawRepos\c3iot-api\functions\source\lightbulb-api-lambda\package.json"

```
{
   "name": "lightbulb-lambda",
   "version": "0.0.1",
   "description": "A lambda that retrieves data for an individual lightbulb",
   "license": "MIT",
   "scripts": {
      "test": "nyc --check-coverage --lines 80 --functions 80 --branches 80 --reporter=cobertura mocha -R
mocha-multi-reporters --recursive test",
      "lint": "eslint . --fix",
      "prettier": "prettier --single-quote --write '**/*.js'"
   },
   "devDependencies": {
      "aws-lambda-mock-context": "^3.2.1",
      "chai": "^4.2.0",
      "chai-as-promised": "7.1.1",
      "eslint": "^5.6.1",
      "eslint-config-airbnb-base": "^13.1.0",
      "eslint-plugin-import": "^2.14.0",
      "expect.js": "^0.3.1",
      "mocha": "^5.2.0",
      "mocha-multi-reporters": "^1.1.7",
      "nyc": "^13.0.1",
      "prettier": "^1.6.1",
      "sinon": "^6.3.5"
   },
   "dependencies": {
      "async": "~2.6.0",
      "aws-helpers": "file:../aws-helpers",
      "aws-sdk": "^2.344.0",
      "database-helpers": "file:../database-helpers",
      "date-fns": "^1.29.0"
   },
   "private": true
}

"FILENAME: C:\dev\c3\rawRepos\c3iot-api\functions\source\lightbulb-api-lambda\test\lightbulb-
api.spec.js"

const chai = require('chai');
const sinon = require('sinon');
const awsHelpers = require('aws-helpers');
const chaiAsPromised = require('chai-as-promised');
const lightbulbLamda = require('../index.js');

chai.use(chaiAsPromised);
const { assert } = chai;
const sandbox = sinon.createSandbox();

const dynamoDBMockData = [{
  payload: '{"timestamp":"2018-11-
24T12:00:00.000Z","smartBulbName":"SMBLB1","status":"0","power":"0","lumens":"0","voltage":"0","te
mperature":"68","weatherData":{},"lightbulb":{"smartBulbName":"SMBLB1","manufacturer":"Bell","bul
bType":"LED","latitude":37.48596719,"longitude":-
122.2428196,"fixtureName":"fixt1","apartmentName":"apt1","buildingName":"bld1"},"switchCountWeek
":0,"hoursOn":6364}',
  smartBulbName: 'SMBLB1',
```

```
  timestamp: '2018-11-24T12:00:00.000Z'
},
{
  payload: '{"timestamp":"2018-11-
24T11:00:00.000Z","smartBulbName":"SMBLB1","status":"1","power":"52","lumens":"234","voltage":"2
3","temperature":"67","weatherData":{},"lightbulb":{"smartBulbName":"SMBLB1","manufacturer":"Bell"
,"bulbType":"LED","latitude":37.48596719,"longitude":-
122.2428196,"fixtureName":"fixt1","apartmentName":"apt1","buildingName":"bld1"},"switchCountWeek
":0,"hoursOn":6364}',
  smartBulbName: 'SMBLB1',
  timestamp: '2018-11-24T11:00:00.000Z'
},
{
  payload: '{"timestamp":"2018-11-
24T10:00:00.000Z","smartBulbName":"SMBLB1","status":"0","power":"0","lumens":"0","voltage":"0","te
mperature":"67","weatherData":{},"lightbulb":{"smartBulbName":"SMBLB1","manufacturer":"Bell","bul
bType":"LED","latitude":37.48596719,"longitude":-
122.2428196,"fixtureName":"fixt1","apartmentName":"apt1","buildingName":"bld1"},"switchCountWeek
":0,"hoursOn":6364}',
  smartBulbName: 'SMBLB1',
  timestamp: '2018-11-24T10:00:00.000Z'
}];

beforeEach(() => {
});

afterEach(() => {
  sandbox.restore();
});

describe('Lightbulb API Lambda Test', () => {
  describe('lightbulbGet map', () => {
    it('Have handler return correct DTO with 3 DynamoDB data points, with smart bulb name and timestamp
in event', () => {
      const event = {
        timestamp: '2018-11-24T13:00:00.000Z', smartBulbName: 'SMBLB1'
      };
      sandbox.stub(awsHelpers, 'queryDynamoDB').callsFake(() => new Promise(resolve =>
resolve(dynamoDBMockData)));

      const lumensArray = [0, 234, 0];
      const timestampArray = ['2018-11-24T12:00:00.000Z', '2018-11-24T11:00:00.000Z', '2018-11-
24T10:00:00.000Z'];
      const statusArray = [0, 1, 0];
      const powerArray = [0, 52, 0];
      const voltageArray = [0, 23, 0];
      const temperatureArray = [68, 67, 67];

      lightbulbLamda.handler(event).then(function Test(results) {
        assert.deepEqual(results.map.timestamp, timestampArray);
        assert.deepEqual(results.map.lumens, lumensArray);
        assert.deepEqual(results.map.status, statusArray);
        assert.deepEqual(results.map.power, powerArray);
        assert.deepEqual(results.map.temperature, temperatureArray);
        assert.deepEqual(results.map.voltage, voltageArray);
      }).catch(function Error(error) { console.log('error', error); });
```

```
  });

  it('Have handler return correct DTO with 3 DynamoDB data points, with just smart bulb name', () => {
    const event = {
      smartBulbName: 'SMBLB1'
    };
    sandbox.stub(awsHelpers, 'queryDynamoDB').callsFake(() => new Promise(resolve =>
resolve(dynamoDBMockData)));

    const lumensArray = [0, 234, 0];
    const timestampArray = ['2018-11-24T12:00:00.000Z', '2018-11-24T11:00:00.000Z', '2018-11-
24T10:00:00.000Z'];
    const statusArray = [0, 1, 0];
    const powerArray = [0, 52, 0];
    const voltageArray = [0, 23, 0];
    const temperatureArray = [68, 67, 67];

    lightbulbLamda.handler(event).then(function Test(results) {
      assert.deepEqual(results.map.timestamp, timestampArray);
      assert.deepEqual(results.map.lumens, lumensArray);
      assert.deepEqual(results.map.status, statusArray);
      assert.deepEqual(results.map.power, powerArray);
      assert.deepEqual(results.map.temperature, temperatureArray);
      assert.deepEqual(results.map.voltage, voltageArray);
    }).catch(function Error(error) { console.log('error', error); });
  });

  it('Have handler return correct DTO with 3 DynamoDB data points, with smart bulb name and days
back', () => {
    const event = {
      smartBulbName: 'SMBLB1', daysBack: 3
    };
    sandbox.stub(awsHelpers, 'queryDynamoDB').callsFake(() => new Promise(resolve =>
resolve(dynamoDBMockData)));

    const lumensArray = [0, 234, 0];
    const timestampArray = ['2018-11-24T12:00:00.000Z', '2018-11-24T11:00:00.000Z', '2018-11-
24T10:00:00.000Z'];
    const statusArray = [0, 1, 0];
    const powerArray = [0, 52, 0];
    const voltageArray = [0, 23, 0];
    const temperatureArray = [68, 67, 67];

    lightbulbLamda.handler(event).then(function Test(results) {
      assert.deepEqual(results.map.timestamp, timestampArray);
      assert.deepEqual(results.map.lumens, lumensArray);
      assert.deepEqual(results.map.status, statusArray);
      assert.deepEqual(results.map.power, powerArray);
      assert.deepEqual(results.map.temperature, temperatureArray);
      assert.deepEqual(results.map.voltage, voltageArray);
    }).catch(function Error(error) { console.log('error', error); });
  });

  it('Have handler return an empty array with 0 DynamoDB data points, with smart bulb name', () => {
    const event = {
      smartBulbName: 'SMBLB1', daysBack: 3
```

```
    };
    const dynamoDBEmptyMockData = [];
    sandbox.stub(awsHelpers, 'queryDynamoDB').callsFake(() => new Promise(resolve =>
resolve(dynamoDBEmptyMockData)));

    const lumensArray = [];
    const timestampArray = [];
    const statusArray = [];
    const powerArray = [];
    const voltageArray = [];
    const temperatureArray = [];

    lightbulbLamda.handler(event).then(function Test(results) {
      assert.deepEqual(results.map.timestamp, timestampArray);
      assert.deepEqual(results.map.lumens, lumensArray);
      assert.deepEqual(results.map.status, statusArray);
      assert.deepEqual(results.map.power, powerArray);
      assert.deepEqual(results.map.temperature, temperatureArray);
      assert.deepEqual(results.map.voltage, voltageArray);
    }).catch(function Error(error) { console.log('error', error); });
  });
 });
 describe('lightbulbGet table', () => {
  it('Have handler return correct table DTO with 3 DynamoDB data points, with smart bulb name and days
back', () => {
    const event = {
      smartBulbName: 'SMBLB1', daysBack: 3
    };
    sandbox.stub(awsHelpers, 'queryDynamoDB').callsFake(() => new Promise(resolve =>
resolve(dynamoDBMockData)));

    lightbulbLamda.handler(event).then(function Test(results) {
      assert.deepEqual(results.table[1], JSON.parse(dynamoDBMockData[1].payload)); // This will throw
an error until we implement the risk score into the payload
    }).catch(function Error(error) { console.log('error', error); });
  });
  it('Have handler return correct table DTO with 0 DynamoDB data points, with smart bulb name and days
back', () => {
    const event = {
      smartBulbName: 'SMBLB1', daysBack: 3
    };
    const dynamoDBEmptyMockData = [];
    sandbox.stub(awsHelpers, 'queryDynamoDB').callsFake(() => new Promise(resolve =>
resolve(dynamoDBEmptyMockData)));

    lightbulbLamda.handler(event).then(function Test(results) {
      assert.deepEqual(results.table, []);
    }).catch(function Error(error) { console.log('error', error); });
  });
 });
});

"FILENAME: C:\dev\c3\rawRepos\c3iot-api\functions\source\lightbulb-autocomplete-api-
lambda\.eslintrc.js"

module.exports = {
```

```
  env: {
    es6: true,
    node: true,
    mocha: true,
    jest: true
  },
  extends: ['airbnb-base'],
  rules: {
    'arrow-parens': ['error', 'as-needed'],
    'comma-dangle': ['error', 'never'],
    'function-paren-newline': 'off',
    'no-underscore-dangle': 'off',
    'prefer-destructuring': 'off',
    'no-console': 'off',
    'import/no-extraneous-dependencies': [
      'warn',
      { devDependencies: false }
    ]
  },
  overrides: [
    {
      files: '**/*.spec.js',
      rules: {
        'no-unused-expressions': 'off',
        'prefer-arrow-callback': 'off',
        'import/no-extraneous-dependencies': [
          'error',
          { devDependencies: true }
        ]
      }
    }
  ]
};

"FILENAME: C:\dev\c3\rawRepos\c3iot-api\functions\source\lightbulb-autocomplete-api-lambda\index.js"

const dbHelpers = require('database-helpers');

function LightbulbAutocompleteDTO(bulbNames) {
  return {
    bulbNames: bulbNames
  };
}

async function getBulbNames() {
  let bulbNamesArray = [];

  const pool = await dbHelpers.getPool();
  let queryString = 'SELECT DISTINCT smart_bulb.`name` AS bulbName FROM smart_bulb'

  const queryResult = await pool.query(queryString);

  queryResult.forEach(
    record => {
      bulbNamesArray.push(record.bulbName);
    }
```

```
  );

  return bulbNamesArray;
}

exports.handler = async (event) => {
  let bulbNames = await getBulbNames();

  return new LightbulbAutocompleteDTO(bulbNames);
};
```

"FILENAME: C:\dev\c3\rawRepos\c3iot-api\functions\source\lightbulb-autocomplete-api-lambda\package.json"

```
{
   "name": "dashboard-lambda",
   "version": "0.0.1",
   "description": "A lambda that retrieves data for the lightbulb dashboard",
   "license": "MIT",
   "scripts": {
      "test": "nyc --check-coverage --lines 80 --functions 80 --branches 80 --reporter=cobertura mocha -R mocha-multi-reporters --recursive test",
      "lint": "eslint . --fix",
      "prettier": "prettier --single-quote --write '**/*.js'"
   },
   "devDependencies": {
      "aws-lambda-mock-context": "^3.2.1",
      "chai": "^4.2.0",
      "chai-as-promised": "7.1.1",
      "eslint": "^5.6.1",
      "eslint-config-airbnb-base": "^13.1.0",
      "eslint-plugin-import": "^2.14.0",
      "expect.js": "^0.3.1",
      "mocha": "^5.2.0",
      "mocha-multi-reporters": "^1.1.7",
      "nyc": "^13.0.1",
      "prettier": "^1.6.1",
      "sinon": "^6.3.5"
   },
   "dependencies": {
      "async": "~2.6.0",
      "aws-helpers": "file:../aws-helpers",
      "aws-sdk": "^2.344.0",
      "database-helpers": "file:../database-helpers"
   },
   "private": true
}
```

"FILENAME: C:\dev\c3\rawRepos\c3iot-api\functions\source\lightbulb-autocomplete-api-lambda\test\lightbulb-autocomplete-api.spec.js"

```
const sinon = require('sinon');
const chai = require('chai');
const chaiAsPromised = require('chai-as-promised');
const dbHelpers = require('../../database-helpers');
// const context = require('aws-lambda-mock-context');
```

```
const dashboardAutocompleteLambda = require('../index.js');

chai.use(chaiAsPromised);
const { assert } = chai;
const { expect } = chai;

describe('Dashboard API Lambda Test', () => {
  const sandbox = sinon.createSandbox();

  beforeEach(() => {
  });

  afterEach(() => {
    sandbox.restore();
  });

  it('get buildings', async () => {
    const callSpy = sinon.spy();

    sandbox.stub(dbHelpers, 'getPool').callsFake(() => ({
      query() {
        console.log("called fake query");
        callSpy();
        return new Promise(resolve => resolve([]));
      }
    }));

    await dashboardAutocompleteLambda.getBuildings();
    expect(callSpy.calledOnce).to.be.true;
  });

  it('get apartments', async () => {
    const callSpy = sinon.spy();

    sandbox.stub(dbHelpers, 'getPool').callsFake(() => ({
      query() {
        console.log("called fake query");
        callSpy();
        return new Promise(resolve => resolve([]));
      }
    }));

    await dashboardAutocompleteLambda.getApartments();
    expect(callSpy.calledOnce).to.be.true;
  });

  it('get manufacturers', async () => {
    const callSpy = sinon.spy();

    sandbox.stub(dbHelpers, 'getPool').callsFake(() => ({
      query() {
        console.log("called fake query");
        callSpy();
        return new Promise(resolve => resolve([]));
      }
    }));
```

```
    await dashboardAutocompleteLambda.getManufacturers();
    expect(callSpy.calledOnce).to.be.true;
  });

  it('get bulb types', async () => {
    const callSpy = sinon.spy();

    sandbox.stub(dbHelpers, 'getPool').callsFake(() => ({
      query() {
        console.log("called fake query");
        callSpy();
        return new Promise(resolve => resolve([]));
      }
    }));

    await dashboardAutocompleteLambda.getBulbTypes();
    expect(callSpy.calledOnce).to.be.true;
  });
});
"FILENAME: C:\dev\c3\rawRepos\c3iot-api\templates\api.template"

{
    "AWSTemplateFormatVersion": "2010-09-09",
    "Description": "Creates RESTful APIs to get lightbulb information",
    "Parameters": {
        "StackSuffix": {
            "Type": "String",
            "Description": "The suffix of the stack",
            "Default": "lightbulb"
        },
        "LambdaCodeS3Bucket": {
            "Type": "String",
            "Description": "Prefix of the S3 Bucket where lambda code exists. The bucket name will be the
value of this parameter.",
            "Default": "lightbulb-lambda-code"
        },
        "LambdaCodeS3KeyPrefix": {
            "Type": "String",
            "Description": "S3 Key where lambda code exists",
            "Default": "api-artifacts"
        },
        "DashboardApiLambdaCodeLocation": {
            "Type": "String",
            "Description": "S3 Path where the Dashboard API Lambda code exists",
            "Default": "dashboard-api-lambda.zip"
        },
        "DashboardAutocompleteApiLambdaCodeLocation": {
            "Type": "String",
            "Description": "S3 Path where Dashboard Autocomplete API Lambda code exists",
            "Default": "dashboard-autocomplete-api-lambda.zip"
        },
        "LightbulbApiLambdaCodeLocation": {
            "Type": "String",
            "Description": "S3 Path where Dashboard Autocomplete API Lambda code exists",
            "Default": "lightbulb-api-lambda.zip"
```

```
    },
    "LightbulbAutocompleteApiLambdaCodeLocation": {
      "Type": "String",
      "Description": "S3 Path where Lightbulb Autocomplete API Lambda code exists",
      "Default": "lightbulb-autocomplete-api-lambda.zip"
    },
    "RDSUsername": {
      "Type":"String",
      "Description":"The RDS username to use for database",
      "Default":"RDSmaster"
    },
    "EncryptedRDSPwd" : {
      "Description" : "The KMS encrypted database admin account password.",
      "Type" : "String"
    },
    "RDSEndpoint": {
      "Type":"String",
      "Description":"The RDS database endpoint",
      "Default":"rds-instance.czktrh4irmdc.us-east-1.rds.amazonaws.com"
    },
    "RDSDatabaseName": {
      "Type":"String",
      "Description":"The RDS username to use for database",
      "Default":"lightbulbAppRds"
    }
  },
  "Resources": {
    "DashboardApiLambdaRole": {
      "Type": "AWS::IAM::Role",
      "Properties": {
        "AssumeRolePolicyDocument": {
          "Version": "2012-10-17",
          "Statement": [{
            "Effect": "Allow",
            "Principal": {
              "Service": ["lambda.amazonaws.com"]
            },
            "Action": ["sts:AssumeRole"]
          }]
        },
        "Path": "/",
        "Policies": [{
          "PolicyName": {
            "Fn::Sub": "DashboardApiLambdaPolicy-${StackSuffix}"
          },
          "PolicyDocument": {
            "Version": "2012-10-17",
            "Statement": [{
                "Effect": "Allow",
                "Action": [
                  "logs:CreateLogGroup",
                  "logs:CreateLogStream",
                  "logs:PutLogEvents",
                  "ec2:CreateNetworkInterface",
                  "ec2:DescribeNetworkInterfaces",
                  "ec2:DeleteNetworkInterface",
```

```
                    "kms:Encrypt",
                    "kms:Decrypt",
                    "kms:ReEncrypt*",
                    "kms:GenerateDataKey*",
                    "kms:DescribeKey"
                ],
                "Resource": "*"
            }
        ]
      }
    }],
    "RoleName": {
      "Fn::Sub": "DashboardApiLambdaRole-${StackSuffix}"
    }
  }
},
"DashboardAutocompleteApiLambdaRole": {
  "Type": "AWS::IAM::Role",
  "Properties": {
    "AssumeRolePolicyDocument": {
      "Version": "2012-10-17",
      "Statement": [{
        "Effect": "Allow",
        "Principal": {
          "Service": ["lambda.amazonaws.com"]
        },
        "Action": ["sts:AssumeRole"]
      }]
    },
    "Path": "/",
    "Policies": [{
      "PolicyName": {
        "Fn::Sub": "DashboardAutocompleteApiLambdaPolicy-${StackSuffix}"
      },
      "PolicyDocument": {
        "Version": "2012-10-17",
        "Statement": [{
            "Effect": "Allow",
            "Action": [
              "logs:CreateLogGroup",
              "logs:CreateLogStream",
              "logs:PutLogEvents",
              "ec2:CreateNetworkInterface",
              "ec2:DescribeNetworkInterfaces",
              "ec2:DeleteNetworkInterface",
              "kms:Encrypt",
              "kms:Decrypt",
              "kms:ReEncrypt*",
              "kms:GenerateDataKey*",
              "kms:DescribeKey"
            ],
            "Resource": "*"
          }
        ]
      }
    }],
```

```
      "RoleName": {
        "Fn::Sub": "DashboardAutocompleteApiLambdaRole-${StackSuffix}"
      }
    }
  },
  "ApiGatewayCloudWatchLogsRole": {
    "Type": "AWS::IAM::Role",
    "Properties": {
      "RoleName": {
        "Fn::Sub": "ApiGatewayCloudWatchLogsRole-${StackSuffix}"
      },
      "AssumeRolePolicyDocument": {
        "Version": "2012-10-17",
        "Statement": [{
          "Effect": "Allow",
          "Principal": {
            "Service": ["apigateway.amazonaws.com"]
          },
          "Action": ["sts:AssumeRole"]
        }]
      },
      "Policies": [{
        "PolicyName": {
          "Fn::Sub": "ApiGatewayLogsPolicy-${StackSuffix}"
        },
        "PolicyDocument": {
          "Version": "2012-10-17",
          "Statement": [{
            "Effect": "Allow",
            "Action": [
              "logs:CreateLogGroup",
              "logs:CreateLogStream",
              "logs:DescribeLogGroups",
              "logs:DescribeLogStreams",
              "logs:PutLogEvents",
              "logs:GetLogEvents",
              "logs:FilterLogEvents"
            ],
            "Resource": "*"
          }]
        }
      }]
    }
  },
  "LightbulbApiLambdaRole": {
    "Type": "AWS::IAM::Role",
    "Properties": {
      "AssumeRolePolicyDocument": {
        "Version": "2012-10-17",
        "Statement": [{
          "Effect": "Allow",
          "Principal": {
            "Service": ["lambda.amazonaws.com"]
          },
          "Action": ["sts:AssumeRole"]
        }]
```

```
        },
        "Path": "/",
        "Policies": [{
          "PolicyName": {
            "Fn::Sub": "LightbulbApiLambdaPolicy-${StackSuffix}"
          },
          "PolicyDocument": {
            "Version": "2012-10-17",
            "Statement": [{
                  "Effect": "Allow",
                  "Action": [
                    "logs:CreateLogGroup",
                    "logs:CreateLogStream",
                    "logs:PutLogEvents",
                    "ec2:CreateNetworkInterface",
                    "ec2:DescribeNetworkInterfaces",
                    "ec2:DeleteNetworkInterface",
                    "dynamodb:Query"

                  ],
                  "Resource": "*"
              }
            ]
          }
        }],
        "RoleName": {
          "Fn::Sub": "LightbulbApiLambdaRole-${StackSuffix}"
        }
      }
    },
    "LightbulbAutocompleteApiLambdaRole": {
      "Type": "AWS::IAM::Role",
      "Properties": {
        "AssumeRolePolicyDocument": {
          "Version": "2012-10-17",
          "Statement": [{
            "Effect": "Allow",
            "Principal": {
              "Service": ["lambda.amazonaws.com"]
            },
            "Action": ["sts:AssumeRole"]
          }]
        },
        "Path": "/",
        "Policies": [{
          "PolicyName": {
            "Fn::Sub": "LightbulbAutocompleteApiLambdaPolicy-${StackSuffix}"
          },
          "PolicyDocument": {
            "Version": "2012-10-17",
            "Statement": [{
                  "Effect": "Allow",
                  "Action": [
                    "logs:CreateLogGroup",
                    "logs:CreateLogStream",
                    "logs:PutLogEvents",
```

```
                        "ec2:CreateNetworkInterface",
                        "ec2:DescribeNetworkInterfaces",
                        "ec2:DeleteNetworkInterface",
                        "kms:Encrypt",
                        "kms:Decrypt",
                        "kms:ReEncrypt*",
                        "kms:GenerateDataKey*",
                        "kms:DescribeKey"
                    ],
                    "Resource": "*"
                }
            ]
          }
        }],
        "RoleName": {
          "Fn::Sub": "LightbulbAutocompleteApiLambdaRole-${StackSuffix}"
        }
      }
    },
    "DashboardApiLambda": {
      "Type": "AWS::Lambda::Function",
      "Properties": {
        "FunctionName": {
          "Fn::Sub": "DashboardApiLambda"
        },
        "Handler": "index.handler",
        "Role": {
          "Fn::GetAtt": ["DashboardApiLambdaRole", "Arn"]
        },
        "Code": {
          "S3Bucket": {"Ref": "LambdaCodeS3Bucket"},
          "S3Key": {
            "Fn::Sub": "${LambdaCodeS3KeyPrefix}/${DashboardApiLambdaCodeLocation}"
          }
        },
        "Runtime": "nodejs8.10",
        "Timeout": 30,
        "MemorySize": "1024",
        "VpcConfig": {
          "SecurityGroupIds" : [ "sg-e56b3daa" ],
          "SubnetIds" : [ "subnet-29db564e", "subnet-02e66e2c" ]
        },
        "KmsKeyArn": "arn:aws:kms:us-east-1:466081484468:key/57647f6b-a8bd-4e66-80b9-
a4eb59d525b3",
        "Environment": {
          "Variables": {
            "MySQLHost" : { "Ref" : "RDSEndpoint" },
            "MySQLUser" : { "Ref" : "RDSUsername" },
            "MySQLPass" : { "Ref" : "EncryptedRDSPwd" },
            "MySQLDB" : { "Ref" : "RDSDatabaseName" }
          }
        }
      }
    },
    "DashboardAutocompleteApiLambda": {
      "Type": "AWS::Lambda::Function",
```

```
        "Properties": {
          "FunctionName": {
            "Fn::Sub": "DashboardAutocompleteApiLambda"
          },
          "Handler": "index.handler",
          "Role": {
            "Fn::GetAtt": ["DashboardAutocompleteApiLambdaRole", "Arn"]
          },
          "Code": {
            "S3Bucket": {"Ref": "LambdaCodeS3Bucket"},
            "S3Key": {
              "Fn::Sub":
"${LambdaCodeS3KeyPrefix}/${DashboardAutocompleteApiLambdaCodeLocation}"
            }
          },
          "Runtime": "nodejs8.10",
          "Timeout": 30,
          "MemorySize": "1024",
          "VpcConfig": {
            "SecurityGroupIds" : [ "sg-e56b3daa" ],
            "SubnetIds" : [ "subnet-29db564e", "subnet-02e66e2c" ]
          },
          "KmsKeyArn": "arn:aws:kms:us-east-1:466081484468:key/57647f6b-a8bd-4e66-80b9-
a4eb59d525b3",
          "Environment": {
            "Variables": {
              "MySQLHost" : { "Ref" : "RDSEndpoint" },
              "MySQLUser" : { "Ref" : "RDSUsername" },
              "MySQLPass" : { "Ref" : "EncryptedRDSPwd" },
              "MySQLDB" : { "Ref" : "RDSDatabaseName" }
            }
          }
        }
      },
      "LightbulbApiLambda": {
        "Type": "AWS::Lambda::Function",
        "Properties": {
          "FunctionName": {
            "Fn::Sub": "LightbulbApiLambda"
          },
          "Handler": "index.handler",
          "Role": {
            "Fn::GetAtt": ["LightbulbApiLambdaRole", "Arn"]
          },
          "Code": {
            "S3Bucket": {"Ref": "LambdaCodeS3Bucket"},
            "S3Key": {
              "Fn::Sub": "${LambdaCodeS3KeyPrefix}/${LightbulbApiLambdaCodeLocation}"
            }
          },
          "Runtime": "nodejs8.10",
          "Timeout": 30,
          "MemorySize": "1024"
        }
      },
      "LightbulbAutocompleteApiLambda": {
```

```
        "Type": "AWS::Lambda::Function",
        "Properties": {
          "FunctionName": {
            "Fn::Sub": "LightbulbAutocompleteApiLambda"
          },
          "Handler": "index.handler",
          "Role": {
            "Fn::GetAtt": ["LightbulbAutocompleteApiLambdaRole", "Arn"]
          },
          "Code": {
            "S3Bucket": {"Ref": "LambdaCodeS3Bucket"},
            "S3Key": {
              "Fn::Sub":
"${LambdaCodeS3KeyPrefix}/${LightbulbAutocompleteApiLambdaCodeLocation}"
            }
          },
          "Runtime": "nodejs8.10",
          "Timeout": 30,
          "MemorySize": "1024",
          "VpcConfig": {
            "SecurityGroupIds" : [ "sg-e56b3daa" ],
            "SubnetIds" : [ "subnet-29db564e", "subnet-02e66e2c" ]
          },
          "KmsKeyArn": "arn:aws:kms:us-east-1:466081484468:key/57647f6b-a8bd-4e66-80b9-
a4eb59d525b3",
          "Environment": {
            "Variables": {
              "MySQLHost" : { "Ref" : "RDSEndpoint" },
              "MySQLUser" : { "Ref" : "RDSUsername" },
              "MySQLPass" : { "Ref" : "EncryptedRDSPwd" },
              "MySQLDB" : { "Ref" : "RDSDatabaseName" }
            }
          }
        }
      },
      "ApiGatewayAccount": {
        "Type": "AWS::ApiGateway::Account",
        "Properties": {
          "CloudWatchRoleArn": {
            "Fn::GetAtt": ["ApiGatewayCloudWatchLogsRole", "Arn"]
          }
        }
      },
      "DashboardApi": {
        "Type": "AWS::ApiGateway::RestApi",
        "Properties": {
          "Name": {
            "Fn::Sub": "DashboardApi-${StackSuffix}"
          },
          "Description": "API used for retrieving dashboard information",
          "FailOnWarnings": true
        }
      },
      "DashboardApiLambdaPermission": {
        "Type": "AWS::Lambda::Permission",
        "Properties": {
```

```
        "Action": "lambda:invokeFunction",
        "FunctionName": {
          "Fn::GetAtt": ["DashboardApiLambda", "Arn"]
        },
        "Principal": "apigateway.amazonaws.com",
        "SourceArn": {
          "Fn::Sub": "arn:aws:execute-api:${AWS::Region}:${AWS::AccountId}:${DashboardApi}/*"
        }
      }
    },
    "DashboardAutocompleteApiLambdaPermission": {
      "Type": "AWS::Lambda::Permission",
      "Properties": {
        "Action": "lambda:invokeFunction",
        "FunctionName": {
          "Fn::GetAtt": ["DashboardAutocompleteApiLambda", "Arn"]
        },
        "Principal": "apigateway.amazonaws.com",
        "SourceArn": {
          "Fn::Sub": "arn:aws:execute-api:${AWS::Region}:${AWS::AccountId}:${DashboardApi}/*"
        }
      }
    },
    "LightbulbApiLambdaPermission": {
      "Type": "AWS::Lambda::Permission",
      "Properties": {
        "Action": "lambda:invokeFunction",
        "FunctionName": {
          "Fn::GetAtt": ["LightbulbApiLambda", "Arn"]
        },
        "Principal": "apigateway.amazonaws.com",
        "SourceArn": {
          "Fn::Sub": "arn:aws:execute-api:${AWS::Region}:${AWS::AccountId}:${DashboardApi}/*"
        }
      }
    },
    "LightbulbAutocompleteApiLambdaPermission": {
      "Type": "AWS::Lambda::Permission",
      "Properties": {
        "Action": "lambda:invokeFunction",
        "FunctionName": {
          "Fn::GetAtt": ["LightbulbAutocompleteApiLambda", "Arn"]
        },
        "Principal": "apigateway.amazonaws.com",
        "SourceArn": {
          "Fn::Sub": "arn:aws:execute-api:${AWS::Region}:${AWS::AccountId}:${DashboardApi}/*"
        }
      }
    },
    "DashboardApiDeployment": {
      "Type": "AWS::ApiGateway::Deployment",
      "DependsOn": ["DashboardApiGetRequest"],
      "Properties": {
        "RestApiId": {
          "Ref": "DashboardApi"
        },
```

```
        "StageName": "DummyStage"
      }
    },
    "DashboardApiStage": {
      "DependsOn": ["ApiGatewayAccount"],
      "Type": "AWS::ApiGateway::Stage",
      "Properties": {
        "DeploymentId": {
          "Ref": "DashboardApiDeployment"
        },
        "MethodSettings": [{
          "DataTraceEnabled": true,
          "HttpMethod": "GET",
          "LoggingLevel": "INFO",
          "ResourcePath": "/*"
        }],
        "RestApiId": {
          "Ref": "DashboardApi"
        },
        "StageName": "LATEST"
      }
    },
    "DashboardApiResource": {
      "Type": "AWS::ApiGateway::Resource",
      "Properties": {
        "RestApiId": {
          "Ref": "DashboardApi"
        },
        "ParentId": {
          "Fn::GetAtt": ["DashboardApi", "RootResourceId"]
        },
        "PathPart": "dashboard"
      }
    },
    "DashboardApiGetRequest": {
      "DependsOn": "DashboardApiLambdaPermission",
      "Type": "AWS::ApiGateway::Method",
      "Properties": {
        "ApiKeyRequired": true,
        "AuthorizationType": "NONE",
        "HttpMethod": "GET",
        "Integration": {
          "Type": "AWS",
          "IntegrationHttpMethod": "POST",
          "Uri": {
            "Fn::Join": ["", ["arn:aws:apigateway:", {
              "Ref": "AWS::Region"
            }, ":lambda:path/2015-03-31/functions/", {
              "Fn::GetAtt": ["DashboardApiLambda", "Arn"]
            }, "/invocations"]]
          },
          "IntegrationResponses": [{
            "StatusCode": 200,
            "ResponseParameters": {
              "method.response.header.Access-Control-Allow-Origin": "'*'"
            }
```

```
            }],
            "RequestTemplates": {
               "application/json": {
                  "Fn::Join": ["", [
                     "{",
                     "\"building\": \"$input.params('building')\"",
                     ",",
                     "\"apt\": \"$input.params('apt')\"",
                     ",",
                     "\"manuf\": \"$input.params('manuf')\"",
                     ",",
                     "\"bulbtype\": \"$input.params('bulbtype')\"",
                     "}"
                  ]]
               }
            }
         },
         "RequestParameters": {
            "method.request.querystring.building": false,
            "method.request.querystring.apt": false,
            "method.request.querystring.manuf": false,
            "method.request.querystring.bulbtype": false
         },
         "ResourceId": {
            "Ref": "DashboardApiResource"
         },
         "RestApiId": {
            "Ref": "DashboardApi"
         },
         "MethodResponses": [{
            "StatusCode": 200,
            "ResponseParameters": {
               "method.response.header.Access-Control-Allow-Origin": true
            }
         }]
      }
   },
   "DashboardApiOptionsRequest": {
      "Type": "AWS::ApiGateway::Method",
      "Properties": {
         "AuthorizationType": "NONE",
         "HttpMethod": "OPTIONS",
         "Integration": {
            "Type": "MOCK",
            "IntegrationResponses": [{
               "StatusCode": 200,
               "ResponseParameters": {
                  "method.response.header.Access-Control-Allow-Headers": "'Content-Type,X-Amz-
Date,Authorization,X-Api-Key,X-Amz-Security-Token'",
                  "method.response.header.Access-Control-Allow-Methods": "'GET,POST,OPTIONS'",
                  "method.response.header.Access-Control-Allow-Origin": "'*'"
               }
            }],
            "RequestTemplates": {
               "application/json": "{\n    \"statusCode\": 200\n}"
            }
```

```
        },
        "ResourceId": {
          "Ref": "DashboardApiResource"
        },
        "RestApiId": {
          "Ref": "DashboardApi"
        },
        "MethodResponses": [{
          "StatusCode": 200,
          "ResponseParameters": {
            "method.response.header.Access-Control-Allow-Headers": true,
            "method.response.header.Access-Control-Allow-Methods": true,
            "method.response.header.Access-Control-Allow-Origin": true
          }
        }]
      }
    },
    "DashboardAutocompleteApiResource": {
      "Type": "AWS::ApiGateway::Resource",
      "Properties": {
        "RestApiId": {
          "Ref": "DashboardApi"
        },
        "ParentId": {
          "Fn::GetAtt": ["DashboardApi", "RootResourceId"]
        },
        "PathPart": "dashboardautocomplete"
      }
    },
    "DashboardAutocompleteApiGetRequest": {
      "DependsOn": "DashboardAutocompleteApiLambdaPermission",
      "Type": "AWS::ApiGateway::Method",
      "Properties": {
        "ApiKeyRequired": true,
        "AuthorizationType": "NONE",
        "HttpMethod": "GET",
        "Integration": {
          "Type": "AWS",
          "IntegrationHttpMethod": "POST",
          "Uri": {
            "Fn::Join": ["", ["arn:aws:apigateway:", {
              "Ref": "AWS::Region"
            }, ":lambda:path/2015-03-31/functions/", {
              "Fn::GetAtt": ["DashboardAutocompleteApiLambda", "Arn"]
            }, "/invocations"]]
          },
          "IntegrationResponses": [{
            "StatusCode": 200,
            "ResponseParameters": {
              "method.response.header.Access-Control-Allow-Origin": "'*'"
            }
          }]
        },
        "ResourceId": {
          "Ref": "DashboardAutocompleteApiResource"
        },
```

```
            "RestApiId": {
               "Ref": "DashboardApi"
            },
            "MethodResponses": [{
               "StatusCode": 200,
               "ResponseParameters": {
                  "method.response.header.Access-Control-Allow-Origin": true
               }
            }]
         }
      },
      "DashboardAutocompleteApiOptionsRequest": {
         "Type": "AWS::ApiGateway::Method",
         "Properties": {
            "AuthorizationType": "NONE",
            "HttpMethod": "OPTIONS",
            "Integration": {
               "Type": "MOCK",
               "IntegrationResponses": [{
                  "StatusCode": 200,
                  "ResponseParameters": {
                     "method.response.header.Access-Control-Allow-Headers": "'Content-Type,X-Amz-
Date,Authorization,X-Api-Key,X-Amz-Security-Token'",
                     "method.response.header.Access-Control-Allow-Methods": "'GET,POST,OPTIONS'",
                     "method.response.header.Access-Control-Allow-Origin": "'*'"
                  }
               }],
               "RequestTemplates": {
                  "application/json": "{\n    \"statusCode\": 200\n}"
               }
            },
            "ResourceId": {
               "Ref": "DashboardAutocompleteApiResource"
            },
            "RestApiId": {
               "Ref": "DashboardApi"
            },
            "MethodResponses": [{
               "StatusCode": 200,
               "ResponseParameters": {
                  "method.response.header.Access-Control-Allow-Headers": true,
                  "method.response.header.Access-Control-Allow-Methods": true,
                  "method.response.header.Access-Control-Allow-Origin": true
               }
            }]
         }
      },
      "LightbulbApiResource": {
         "Type": "AWS::ApiGateway::Resource",
         "Properties": {
            "RestApiId": {
               "Ref": "DashboardApi"
            },
            "ParentId": {
               "Fn::GetAtt": ["DashboardApi", "RootResourceId"]
            },
```

```
        "PathPart": "lightbulb"
      }
    },
    "LightbulbApiGetRequest": {
      "DependsOn": "LightbulbApiLambdaPermission",
      "Type": "AWS::ApiGateway::Method",
      "Properties": {
        "ApiKeyRequired": true,
        "AuthorizationType": "NONE",
        "HttpMethod": "GET",
        "Integration": {
          "Type": "AWS",
          "IntegrationHttpMethod": "POST",
          "Uri": {
            "Fn::Join": ["", ["arn:aws:apigateway:", {
              "Ref": "AWS::Region"
            }, ":lambda:path/2015-03-31/functions/", {
              "Fn::GetAtt": ["LightbulbApiLambda", "Arn"]
            }, "/invocations"]]
          },
          "IntegrationResponses": [{
            "StatusCode": 200,
            "ResponseParameters": {
              "method.response.header.Access-Control-Allow-Origin": "'*'"
            }
          }],
          "RequestTemplates": {
            "application/json": {
              "Fn::Join": ["", [
                "{",
                "\"smartbulbname\": \"$input.params('smartbulbname')\"",
                ",",
                "\"timestamp\": \"$input.params('timestamp')\"",
                ",",
                "\"daysback\": \"$input.params('daysback')\"",
                "}"
              ]]
            }
          }
        },
        "ResourceId": {
          "Ref": "LightbulbApiResource"
        },
        "RestApiId": {
          "Ref": "DashboardApi"
        },
        "RequestParameters": {
          "method.request.querystring.smartbulbname": true,
          "method.request.querystring.timestamp": false,
          "method.request.querystring.daysback": false
          },

        "MethodResponses": [{
          "StatusCode": 200,
          "ResponseParameters": {
            "method.response.header.Access-Control-Allow-Origin": true
```

```
            }
          }]
        }
      },
      "LightbulbApiOptionsRequest": {
        "Type": "AWS::ApiGateway::Method",
        "Properties": {
          "AuthorizationType": "NONE",
          "HttpMethod": "OPTIONS",
          "Integration": {
            "Type": "MOCK",
            "IntegrationResponses": [{
              "StatusCode": 200,
              "ResponseParameters": {
                "method.response.header.Access-Control-Allow-Headers": "'Content-Type,X-Amz-
Date,Authorization,X-Api-Key,X-Amz-Security-Token'",
                "method.response.header.Access-Control-Allow-Methods": "'GET,POST,OPTIONS'",
                "method.response.header.Access-Control-Allow-Origin": "'*'"
              }
            }],
            "RequestTemplates": {
              "application/json": "{\n    \"statusCode\": 200\n}"
            }
          },
          "ResourceId": {
            "Ref": "LightbulbApiResource"
          },
          "RestApiId": {
            "Ref": "DashboardApi"
          },
          "MethodResponses": [{
            "StatusCode": 200,
            "ResponseParameters": {
              "method.response.header.Access-Control-Allow-Headers": true,
              "method.response.header.Access-Control-Allow-Methods": true,
              "method.response.header.Access-Control-Allow-Origin": true
            }
          }]
        }
      },
      "LightbulbAutocompleteApiResource": {
        "Type": "AWS::ApiGateway::Resource",
        "Properties": {
          "RestApiId": {
            "Ref": "DashboardApi"
          },
          "ParentId": {
            "Fn::GetAtt": ["DashboardApi", "RootResourceId"]
          },
          "PathPart": "lightbulbautocomplete"
        }
      },
      "LightbulbAutocompleteApiGetRequest": {
        "DependsOn": "LightbulbAutocompleteApiLambdaPermission",
        "Type": "AWS::ApiGateway::Method",
        "Properties": {
```

```
            "ApiKeyRequired": true,
            "AuthorizationType": "NONE",
            "HttpMethod": "GET",
            "Integration": {
               "Type": "AWS",
               "IntegrationHttpMethod": "POST",
               "Uri": {
                  "Fn::Join": ["", ["arn:aws:apigateway:", {
                     "Ref": "AWS::Region"
                  }, ":lambda:path/2015-03-31/functions/", {
                     "Fn::GetAtt": ["LightbulbAutocompleteApiLambda", "Arn"]
                  }, "/invocations"]]
               },
               "IntegrationResponses": [{
                  "StatusCode": 200,
                  "ResponseParameters": {
                     "method.response.header.Access-Control-Allow-Origin": "'*'"
                  }
               }]
            },
            "ResourceId": {
               "Ref": "LightbulbAutocompleteApiResource"
            },
            "RestApiId": {
               "Ref": "DashboardApi"
            },
            "MethodResponses": [{
               "StatusCode": 200,
               "ResponseParameters": {
                  "method.response.header.Access-Control-Allow-Origin": true
               }
            }]
         }
      },
      "LightbulbAutocompleteApiOptionsRequest": {
         "Type": "AWS::ApiGateway::Method",
         "Properties": {
            "AuthorizationType": "NONE",
            "HttpMethod": "OPTIONS",
            "Integration": {
               "Type": "MOCK",
               "IntegrationResponses": [{
                  "StatusCode": 200,
                  "ResponseParameters": {
                     "method.response.header.Access-Control-Allow-Headers": "'Content-Type,X-Amz-
Date,Authorization,X-Api-Key,X-Amz-Security-Token'",
                     "method.response.header.Access-Control-Allow-Methods": "'GET,POST,OPTIONS'",
                     "method.response.header.Access-Control-Allow-Origin": "'*'"
                  }
               }],
               "RequestTemplates": {
                  "application/json": "{\n    \"statusCode\": 200\n}"
               }
            },
            "ResourceId": {
               "Ref": "LightbulbAutocompleteApiResource"
```

```
            },
            "RestApiId": {
               "Ref": "DashboardApi"
            },
            "MethodResponses": [{
               "StatusCode": 200,
               "ResponseParameters": {
                  "method.response.header.Access-Control-Allow-Headers": true,
                  "method.response.header.Access-Control-Allow-Methods": true,
                  "method.response.header.Access-Control-Allow-Origin": true
               }
            }]
         }
      },
      "DashboardApiKey": {
         "Type" : "AWS::ApiGateway::ApiKey",
         "DependsOn": ["DashboardApiDeployment", "DashboardApiStage"],
         "Properties": {
            "Name": "DashboardApiKey",
            "Enabled": "true"
         }
      },
      "DashboardApiUsagePlan" : {
         "Type" : "AWS::ApiGateway::UsagePlan",
         "DependsOn": ["DashboardApiDeployment", "DashboardApiStage"],
         "Properties" : {
            "ApiStages" : [ {"ApiId" : { "Ref" : "DashboardApi" }, "Stage" : { "Ref" : "DashboardApiStage"
}} ],
            "Description" : "Dashboard API usage plan",
            "UsagePlanName" : { "Fn::Sub": "DashboardApiUsagePlan-${StackSuffix}" }
         }
      },
      "DashboardApiUsagePlanKey" : {
         "Type": "AWS::ApiGateway::UsagePlanKey",
         "DependsOn": ["DashboardApiUsagePlan", "DashboardApiKey"],
         "Properties": {
            "KeyId" : {"Ref" : "DashboardApiKey"},
            "KeyType" : "API_KEY",
            "UsagePlanId" : {"Ref" : "DashboardApiUsagePlan"}
         }
      },
      "Deployment": {
         "DependsOn": ["DashboardApiGetRequest",
            "DashboardApiOptionsRequest",
            "DashboardAutocompleteApiGetRequest",
            "DashboardAutocompleteApiOptionsRequest",
            "LightbulbAutocompleteApiGetRequest",
            "LightbulbAutocompleteApiOptionsRequest",
            "LightbulbApiGetRequest",
            "LightbulbApiOptionsRequest"
         ],
         "Type": "AWS::ApiGateway::Deployment",
         "Properties": {
            "RestApiId": { "Ref": "DashboardApi" },
            "Description": "DashboardApi Deployment",
            "StageName": "LATEST"
```

```
        }
      }
    }
}
"FILENAME: C:\dev\c3\rawRepos\c3iot-api\tools\package-all-lambda-code.sh"

./tools/package-lambda-code.sh dashboard-api-lambda

rm -rf functions/source/dashboard-api-lambda/node_modules

"FILENAME: C:\dev\c3\rawRepos\c3iot-api\tools\package-lambda-code.sh"

#!/bin/sh

lambda=$1

baseSource="functions/source"
source="$baseSource/$lambda"
filename="$lambda.zip"
package_folder="functions/packages"
destination="$package_folder/$lambda/$filename"

cd "$baseSource/aws-helpers"
rm -rf node_modules
cd "../../.."

cd $source

s3Key="$lambda/$filename"

# Clearing out node_modules so we can build a smaller package
rm -rf node_modules
npm install --only=prod
zip -r $lambda .

cd "../../.."

mkdir -p $package_folder/$lambda/
mv "$source/$filename" $destination

"FILENAME: C:\dev\c3\rawRepos\c3iot-api\tools\upload-lambda-code-s3.sh"

#!/bin/sh

#Enable the following line if running in git bash on Windows
alias aws="winpty /C/Program\ Files/Amazon/AWSCLI/bin/aws.cmd"

lambda=$1

filename="$lambda.zip"
package_folder="functions/packages"
packageLocation="$package_folder/$lambda/$filename"

aws s3api put-object --bucket lightbulb-lambda-code --key api-artifacts/$filename --body
$packageLocation
```

"FILENAME: C:\dev\c3\rawRepos\imet\data\smart-bulb.sql"

```
INSERT INTO `lightbulbAppRds`.`smart_bulb`
(`manufacturer`,
`bulb_type`,
`name`,
`latitude`,
`longitude`)
VALUES
("Bell","LED","SMBLB1",37.48596719,-122.2428196),
("GE","LED","SMBLB2",37.49115273,-122.2319523),
("GE","LED","SMBLB3",37.49099652,-122.2324551),
("Bell","LED","SMBLB4",37.4886712,-122.2348786),
("Philips","LED","SMBLB5",37.48498754,-122.2438649),
("Philips","LED","SMBLB6",37.48696518,-122.2384678),
("Bell","LED","SMBLB7",37.48473396,-122.2345444),
("Bell","LED","SMBLB8",37.48565758,-122.2412995),
("Philips","LED","SMBLB9",37.48033504,-122.2364642),
("GE","LED","SMBLB10",37.48495049,-122.2335112),
("Philips","LED","SMBLB11",37.48754556,-122.2306994),
("Bell","LED","SMBLB12",37.49240941,-122.2365051),
("GE","LED","SMBLB13",37.48139121,-122.2370073),
("Bell","LED","SMBLB14",37.49186437,-122.2329713),
("GE","LED","SMBLB15",37.49023012,-122.2358988),
("Philips","LED","SMBLB16",37.48851555,-122.2312477),
("Philips","LED","SMBLB17",37.48910108,-122.2348486),
("Philips","LED","SMBLB18",37.48178755,-122.2441439),
("Philips","LED","SMBLB19",37.48490753,-122.230813),
("Bell","LED","SMBLB20",37.48189289,-122.2398116),
("Philips","LED","SMBLB21",37.48767552,-122.2348742),
("GE","LED","SMBLB22",37.48551799,-122.2434559),
("Philips","LED","SMBLB23",37.49012407,-122.2380723),
("GE","LED","SMBLB24",37.47886653,-122.2406753),
("GE","LED","SMBLB25",37.48103057,-122.2323611),
("Philips","LED","SMBLB26",37.48727546,-122.2291541),
("Bell","LED","SMBLB27",37.47963233,-122.2371951),
("Bell","LED","SMBLB28",37.47848975,-122.2383186),
("Bell","LED","SMBLB29",37.48747083,-122.235961),
("Philips","LED","SMBLB30",37.48571808,-122.2356754),
("Bell","LED","SMBLB31",37.48202787,-122.2339956),
("GE","LED","SMBLB32",37.48092079,-122.2432991),
("Bell","LED","SMBLB33",37.48081476,-122.2333322),
("Philips","LED","SMBLB34",37.48197484,-122.2289861),
("GE","LED","SMBLB35",37.48138125,-122.2395454),
("Philips","CFL","SMBLB36",37.48219742,-122.2305021),
("Bell","CFL","SMBLB37",37.48183632,-122.2370443),
("Philips","CFL","SMBLB38",37.4850968,-122.2351568),
("GE","CFL","SMBLB39",37.48146452,-122.2433655),
("GE","CFL","SMBLB40",37.48455106,-122.2311152),
("Bell","CFL","SMBLB41",37.49159504,-122.2364119),
("Philips","CFL","SMBLB42",37.4873215,-122.2347744),
("GE","CFL","SMBLB43",37.48713356,-122.2328573),
("Philips","CFL","SMBLB44",37.48173825,-122.2364763),
("Bell","CFL","SMBLB45",37.48329163,-122.2291209),
("GE","CFL","SMBLB46",37.48096571,-122.2289799),
```

("Bell","CFL","SMBLB47",37.4782097,-122.2361898),
("GE","CFL","SMBLB48",37.48693456,-122.244686),
("GE","CFL","SMBLB49",37.48389012,-122.2290597),
("Philips","CFL","SMBLB50",37.47811504,-122.2366174),
("Bell","CFL","SMBLB51",37.48340709,-122.2438903),
("Philips","CFL","SMBLB52",37.48145738,-122.2304286),
("GE","CFL","SMBLB53",37.48149554,-122.2389685),
("Philips","CFL","SMBLB54",37.48221497,-122.2303585),
("Philips","CFL","SMBLB55",37.482093,-122.2337812),
("GE","CFL","SMBLB56",37.49085228,-122.2326171),
("GE","CFL","SMBLB57",37.48050122,-122.2370964),
("Philips","CFL","SMBLB58",37.48978151,-122.2409696),
("GE","CFL","SMBLB59",37.48612328,-122.2376058),
("Bell","CFL","SMBLB60",37.49000651,-122.2370966),
("Bell","CFL","SMBLB61",37.48690806,-122.2424938),
("GE","CFL","SMBLB62",37.4864202,-122.2330998),
("Bell","CFL","SMBLB63",37.48317552,-122.2327277),
("GE","CFL","SMBLB64",37.48441632,-122.2355749),
("GE","CFL","SMBLB65",37.48805742,-122.2379929),
("GE","CFL","SMBLB66",37.47933887,-122.2397703),
("Philips","CFL","SMBLB67",37.48013581,-122.2345651),
("GE","CFL","SMBLB68",37.48564711,-122.2451554),
("Philips","CFL","SMBLB69",37.48471346,-122.2329462),
("GE","CFL","SMBLB70",37.48165565,-122.2295085),
("GE","INCAN","SMBLB71",37.48946526,-122.2342469),
("Philips","INCAN","SMBLB72",37.48996027,-122.2372601),
("GE","INCAN","SMBLB73",37.48686505,-122.2385513),
("Philips","INCAN","SMBLB74",37.48783614,-122.2386279),
("GE","INCAN","SMBLB75",37.48041943,-122.2303494),
("GE","INCAN","SMBLB76",37.48886444,-122.2382416),
("Philips","INCAN","SMBLB77",37.48737265,-122.2332354),
("Bell","INCAN","SMBLB78",37.4853778,-122.24126),
("GE","INCAN","SMBLB79",37.48051245,-122.2328671),
("Bell","INCAN","SMBLB80",37.48983769,-122.2316233),
("Bell","INCAN","SMBLB81",37.48485456,-122.2428451),
("Philips","INCAN","SMBLB82",37.48099546,-122.2336633),
("GE","INCAN","SMBLB83",37.48961939,-122.2330229),
("GE","INCAN","SMBLB84",37.48841298,-122.2304134),
("Bell","INCAN","SMBLB85",37.48665954,-122.2384863),
("Bell","INCAN","SMBLB86",37.48192893,-122.2291017),
("Bell","INCAN","SMBLB87",37.47809215,-122.2351019),
("GE","INCAN","SMBLB88",37.48943059,-122.230806),
("Philips","INCAN","SMBLB89",37.48359668,-122.2441759),
("Bell","INCAN","SMBLB90",37.48508144,-122.2427776),
("GE","INCAN","SMBLB91",37.49182911,-122.239443),
("Philips","INCAN","SMBLB92",37.48322779,-122.2382966),
("Bell","INCAN","SMBLB93",37.48066176,-122.2338844),
("GE","INCAN","SMBLB94",37.49066877,-122.2411348),
("Philips","INCAN","SMBLB95",37.48565609,-122.2420495),
("Philips","INCAN","SMBLB96",37.48240447,-122.2338945),
("GE","INCAN","SMBLB97",37.49051868,-122.2331416),
("Bell","INCAN","SMBLB98",37.49043204,-122.2302848),
("GE","INCAN","SMBLB99",37.4897744,-122.2391418),
("GE","INCAN","SMBLB100",37.48378625,-122.2447513);

"FILENAME: C:\dev\c3\rawRepos\imet\data\tables\apartment.sql"

```
CREATE TABLE `apartment` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `name` varchar(255) DEFAULT NULL,
  `building_name` varchar(255) DEFAULT NULL,
  PRIMARY KEY (`id`),
  UNIQUE KEY `name_UNIQUE` (`name`),
  KEY `fk_apartment_building_name_idx` (`building_name`),
  CONSTRAINT `fk_apartment_building_name` FOREIGN KEY (`building_name`) REFERENCES
`building` (`name`) ON DELETE NO ACTION ON UPDATE NO ACTION
)

"FILENAME: C:\dev\c3\rawRepos\imet\data\tables\building.sql"

CREATE TABLE `building` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `name` varchar(255) DEFAULT NULL,
  PRIMARY KEY (`id`),
  UNIQUE KEY `name_UNIQUE` (`name`)
)

"FILENAME: C:\dev\c3\rawRepos\imet\data\tables\fixture.sql"

CREATE TABLE `fixture` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `name` varchar(255) DEFAULT NULL,
  `apartment_name` varchar(255) DEFAULT NULL,
  PRIMARY KEY (`id`),
  UNIQUE KEY `name_UNIQUE` (`name`),
  KEY `fk_fixture_apartment_id_idx` (`apartment_name`),
  CONSTRAINT `fk_fixture_apartment_name` FOREIGN KEY (`apartment_name`) REFERENCES
`apartment` (`name`) ON DELETE NO ACTION ON UPDATE NO ACTION
)

"FILENAME: C:\dev\c3\rawRepos\imet\data\tables\smart_bulb.sql"

CREATE TABLE `smart_bulb` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `name` varchar(255) NOT NULL,
  `manufacturer` varchar(255) DEFAULT NULL,
  `bulb_type` varchar(255) DEFAULT NULL,
  `latitude` double DEFAULT NULL,
  `longitude` double DEFAULT NULL,
  `long_life` tinyint(4) DEFAULT NULL,
  `overheat_date` datetime DEFAULT NULL,
  `failure` tinyint(4) DEFAULT NULL,
  `current_fixture_id` int(11) DEFAULT NULL,
  `lumens_uom` varchar(45) DEFAULT NULL,
  `power_uom` varchar(45) DEFAULT NULL,
  `temperature_uom` varchar(45) DEFAULT NULL,
  `voltage_uom` varchar(45) DEFAULT NULL,
  `start_date` datetime DEFAULT NULL,
  PRIMARY KEY (`id`),
  UNIQUE KEY `idx_smart_bulb_name` (`name`)
)
```

"FILENAME: C:\dev\c3\rawRepos\imet\data\tables\smart_bulb_event.sql"

```
CREATE TABLE `smart_bulb_event` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `smart_bulb_name` varchar(255) DEFAULT NULL,
  `event_code` varchar(45) DEFAULT NULL,
  `event_type` varchar(255) DEFAULT NULL,
  `start` datetime DEFAULT NULL,
  `end` datetime DEFAULT NULL,
  PRIMARY KEY (`id`),
  KEY `fk_smart_bulb_event_smart_bulb_name_idx` (`smart_bulb_name`),
  CONSTRAINT `fk_smart_bulb_event_smart_bulb_name` FOREIGN KEY (`smart_bulb_name`)
REFERENCES `smart_bulb` (`name`) ON DELETE NO ACTION ON UPDATE NO ACTION
)
```

"FILENAME: C:\dev\c3\rawRepos\imet\data\tables\smart_bulb_fixture.sql"

```
CREATE TABLE `smart_bulb_fixture` (
  `smart_bulb_name` varchar(255) NOT NULL,
  `fixture_name` varchar(255) NOT NULL,
  `start` datetime DEFAULT NULL,
  `end` datetime DEFAULT NULL,
  PRIMARY KEY (`smart_bulb_name`,`fixture_name`)
)
```

"FILENAME: C:\dev\c3\rawRepos\imet\data\tables\smart_bulb_measurement.sql"

```
CREATE TABLE `smart_bulb_measurement` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `smart_bulb_measurement_series_id` int(11) DEFAULT NULL,
  `lumens` double DEFAULT NULL,
  `power` double DEFAULT NULL,
  `temperature` double DEFAULT NULL,
  `voltage` double DEFAULT NULL,
  PRIMARY KEY (`id`),
  KEY `fk_smart_bulb_measurement_smart_bulb_measurement_series_id_idx`
(`smart_bulb_measurement_series_id`),
  CONSTRAINT `fk_smart_bulb_measurement_smart_bulb_measurement_series_id` FOREIGN KEY
(`smart_bulb_measurement_series_id`) REFERENCES `smart_bulb_measurement_series` (`id`) ON
DELETE NO ACTION ON UPDATE NO ACTION
)
```

"FILENAME: C:\dev\c3\rawRepos\imet\data\tables\smart_bulb_measurement_series.sql"

```
CREATE TABLE `smart_bulb_measurement_series` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `smart_bulb_name` varchar(255) DEFAULT NULL,
  `interval` varchar(255) DEFAULT NULL,
  `treatment` varchar(255) DEFAULT NULL,
  PRIMARY KEY (`id`),
  KEY `fk_smart_bulb_measurement_series_smart_bulb_name_idx` (`smart_bulb_name`),
  CONSTRAINT `fk_smart_bulb_measurement_series_smart_bulb_name` FOREIGN KEY
(`smart_bulb_name`) REFERENCES `smart_bulb` (`name`) ON DELETE NO ACTION ON UPDATE
NO ACTION
)
```

"FILENAME: C:\dev\c3\rawRepos\imet\data\tables\smart_bulb_prediction.sql"

```
CREATE TABLE `smart_bulb_prediction` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `prediction` double DEFAULT NULL,
  `time_stamp` datetime DEFAULT NULL,
  `smart_bulb_name` varchar(255) DEFAULT NULL,
  PRIMARY KEY (`id`),
  UNIQUE KEY `prediction_UNIQUE` (`prediction`),
  KEY `fk_smart_bulb_prediction_smart_bulb_name_idx` (`smart_bulb_name`),
  CONSTRAINT `fk_smart_bulb_prediction_smart_bulb_name` FOREIGN KEY (`smart_bulb_name`)
REFERENCES `smart_bulb` (`name`) ON DELETE NO ACTION ON UPDATE NO ACTION
)
```

"FILENAME: C:\dev\c3\rawRepos\imet\data\tables\smart_bulb_status.sql"

```
CREATE TABLE `smart_bulb_status` (
  `id` int(11) NOT NULL,
  `smart_bulb_name` varchar(255) DEFAULT NULL,
  `status` varchar(255) DEFAULT NULL,
  PRIMARY KEY (`id`),
  KEY `fk_smart_bulb_status_smart_bulb_name_idx` (`smart_bulb_name`),
  CONSTRAINT `fk_smart_bulb_status_smart_bulb_name` FOREIGN KEY (`smart_bulb_name`)
REFERENCES `smart_bulb` (`name`) ON DELETE NO ACTION ON UPDATE NO ACTION
)
```

"FILENAME: C:\dev\c3\rawRepos\imet\data\tables\smart_bulb_status_set.sql"

```
CREATE TABLE `smart_bulb_status_set` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `smart_bulb_name` varchar(255) DEFAULT NULL,
  `status` varchar(255) DEFAULT NULL,
  `time_stamp` datetime DEFAULT NULL,
  PRIMARY KEY (`id`),
  KEY `fk_smart_bulb_status_set_smart_bulb_name_idx` (`smart_bulb_name`),
  CONSTRAINT `fk_smart_bulb_status_set_smart_bulb_name` FOREIGN KEY (`smart_bulb_name`)
REFERENCES `smart_bulb` (`name`) ON DELETE NO ACTION ON UPDATE NO ACTION
)
```

"FILENAME: C:\dev\c3\rawRepos\imet\functions\source\alerts-lambda\.eslintrc.js"

```
module.exports = {
  env: {
    es6: true,
    node: true,
    mocha: true,
    jest: true
  },
  extends: ['airbnb-base'],
  rules: {
    'arrow-parens': ['error', 'as-needed'],
    'comma-dangle': ['error', 'never'],
    'function-paren-newline': 'off',
    'no-underscore-dangle': 'off',
    'prefer-destructuring': 'off',
    'no-console': 'off',
```

```
    'no-await-in-loop': 'off',
    'import/no-extraneous-dependencies': [
      'warn',
      { devDependencies: false }
    ],
    'no-restricted-syntax': [
      'error',
      {
        selector: 'ForInStatement',
        message: 'for..in loops iterate over the entire prototype chain, which is virtually never what you want.
Use Object.{keys,values,entries}, and iterate over the resulting array.',
      },
      {
        selector: 'LabeledStatement',
        message: 'Labels are a form of GOTO; using them makes code confusing and hard to maintain and
understand.',
      },
      {
        selector: 'WithStatement',
        message: '`with` is disallowed in strict mode because it makes code impossible to predict and
optimize.',
      },
    ],
  },
  overrides: [
    {
      files: '**/*.spec.js',
      rules: {
        'no-unused-expressions': 'off',
        'prefer-arrow-callback': 'off',
        'import/no-extraneous-dependencies': [
          'error',
          { devDependencies: true }
        ]
      }
    }
  ]
};

"FILENAME: C:\dev\c3\rawRepos\imet\functions\source\alerts-lambda\index.js"

const awsHelpers = require('aws-helpers');
const dbHelpers = require('database-helpers');

const Alert = function () {
  this.alertType = '';
};

Alert.prototype = {
  setStrategy(alertType) {
    this.alertType = alertType;
  },

  process(bulbHistoryRecord) {
    return this.alertType.process(bulbHistoryRecord);
  }
```

```
};

const AlertInfo = class {
  constructor(shouldSend, message, eventCode, eventType) {
    this.shouldSend = shouldSend;
    this.message = message;
    this.eventCode = eventCode;
    this.eventType = eventType;
  }
};

const isOverheat = function (bulbHistoryRecord) {
  return bulbHistoryRecord.temperature > 95;
};

const isLongLife = function (bulbHistoryRecord) {
  return bulbHistoryRecord.hoursOn > 10500;
};

const getAlertInfo = function (bulbHistoryRecord, alertType) {
  let shouldSendAlert = true;
  let message = '';
  let eventCode = 0;

  switch (alertType) {
    case 'OVERHEAT':
      message = (`${bulbHistoryRecord.smartBulbName} bulb is overheating. Timestamp:
${bulbHistoryRecord.timestamp} Temperature: ${bulbHistoryRecord.temperature}`);
      eventCode = 2;
      break;
    case 'LONGLIFE':
      message = (`${bulbHistoryRecord.smartBulbName} bulb has reached longlife. Timestamp:
${bulbHistoryRecord.timestamp}`);
      eventCode = 3;
      break;
    default:
      shouldSendAlert = false;
      console.log('Not a valid alert type');
  }

  return new AlertInfo(shouldSendAlert, message, eventCode, alertType);
};

const OverheatAlert = function () {
  this.process = function (bulbHistoryRecord) {
    let alertInfo = new AlertInfo(false, null, null, null);
    if (isOverheat(bulbHistoryRecord)) {
      alertInfo = getAlertInfo(bulbHistoryRecord, 'OVERHEAT');
    }
    return alertInfo;
  };
};

const LongLifeAlert = function () {
  this.process = function (bulbHistoryRecord) {
    let alertInfo = new AlertInfo(false, null, null, null);
```

```
    if (isLongLife(bulbHistoryRecord)) {
      alertInfo = getAlertInfo(bulbHistoryRecord, 'LONGLIFE');
    }
    return alertInfo;
  };
};

async function insertAlertEventIntoRDS(bulbHistoryRecord, eventCode, eventType) {
  console.log('Inserting alert event into database');
  const queryString = 'INSERT INTO `smart_bulb_event`'
  + ' (`smart_bulb_name`,'
  + ' `event_code`,'
  + ' `event_type`,'
  + ' `start`)'
  + ' VALUES'
  + ' (?,?,?,?);';

  const pool = await dbHelpers.getPool();
  try {
    await pool.query(queryString, [bulbHistoryRecord.smartBulbName,
      eventCode,
      eventType,
      bulbHistoryRecord.timestamp
    ]);
  } catch (err) {
    throw new Error(`Error inserting alert event into database: ${err}`);
  }
  return true;
}

async function sendAlert(awsRegion, bulbHistoryRecord, topicArn, message, eventCode, alertType) {
  console.log(`Sending ${alertType} alert`);
  try {
    await insertAlertEventIntoRDS(bulbHistoryRecord, eventCode, alertType);
    await awsHelpers.sendDataToSNS(awsRegion, message, alertType, topicArn);
  } catch (err) {
    throw new Error(`Error in sendAlert(): ${err}`);
  }
}

async function sendAlerts(awsRegion, topicArn, bulbHistoryRecord, alertInfoArray) {
  for (const alertInfo of alertInfoArray) {
    if (alertInfo.shouldSend) {
      await sendAlert(awsRegion, bulbHistoryRecord, topicArn, alertInfo.message, alertInfo.eventCode,
alertInfo.eventType);
    }
  }
}

const processAlerts = async function (event) {
  const awsRegion = process.env.AWS_REGION;
  const topicArn = process.env.TopicArn;
  console.log('Received event:', JSON.stringify(event, null, 2));

  for (const record of event.Records) {
    try {
```

```
      const dataString = Buffer.from(record.kinesis.data, 'base64').toString('utf8');
      const parsedDataString = JSON.parse(dataString);
      const bulbHistoryRecord = JSON.parse(parsedDataString.payload);

      const alert = new Alert();
      const alertInfoArray = [];

      const overheatAlert = new OverheatAlert();
      const longLifeAlert = new LongLifeAlert();

      alert.setStrategy(overheatAlert);
      alertInfoArray.push(alert.process(bulbHistoryRecord));

      alert.setStrategy(longLifeAlert);
      alertInfoArray.push(alert.process(bulbHistoryRecord));

      await sendAlerts(awsRegion, topicArn, bulbHistoryRecord, alertInfoArray);
    } catch (err) {
      console.log(`Failed to process events: ${err}`);
    }
  }

  return 'Completed alerts processing';
};

module.exports = {
  isOverheat,
  isLongLife,
  getAlertInfo,
  OverheatAlert,
  LongLifeAlert,
  sendAlerts,
  sendAlert,
  insertAlertEventIntoRDS,
  handler: processAlerts
};
```

"FILENAME: C:\dev\c3\rawRepos\imet\functions\source\alerts-lambda\package.json"

```
{
    "name": "alerts-lambda",
    "version": "0.0.1",
    "description": "lambda function that sends out messages as alerts",
    "license": "MIT",
    "scripts": {
        "test": "nyc --check-coverage --lines 80 --functions 80 --branches 80 --reporter=cobertura mocha -R
mocha-multi-reporters --recursive test",
        "lint": "eslint . --fix",
        "prettier": "prettier --single-quote --write '**/*.js'"
    },
    "devDependencies": {
        "aws-lambda-mock-context": "^3.2.1",
        "chai": "^4.2.0",
        "chai-as-promised": "7.1.1",
        "expect.js": "^0.3.1",
        "eslint": "^5.6.1",
```

```
      "eslint-config-airbnb-base": "^13.1.0",
      "eslint-plugin-import": "^2.14.0",
      "mocha": "^5.2.0",
      "mocha-multi-reporters": "^1.1.7",
      "nyc": "^13.0.1",
      "prettier": "^1.6.1",
      "sinon": "^6.3.5"
   },
   "dependencies": {
      "async": "~2.6.0",
      "aws-helpers": "file:../aws-helpers",
      "database-helpers": "file:../database-helpers",
      "request-promise-native": "^1.0.5",
      "bulb-history-record": "file:../bulb-history-record"
   },
   "prettier": {
      "singleQuote": true
   },
   "private": true
}

"FILENAME: C:\dev\c3\rawRepos\imet\functions\source\alerts-lambda\test\alerts.spec.js"

const sinon = require('sinon');
const chai = require('chai');
const chaiAsPromised = require('chai-as-promised');
const alerts = require('../index.js');
const awsHelpers = require('aws-helpers');
const dbHelpers = require('../../database-helpers');

chai.use(chaiAsPromised);
const { assert } = chai;
const { expect } = chai;

describe('Alerts Lambda Test', () => {
  const sandbox = sinon.createSandbox();

  beforeEach(() => {
  });

  afterEach(() => {
    sandbox.restore();
  });

  function getBaselineBulbHistoryRecord() {
    return {
      timestamp: '2018-11-01T13:00:00-0800',
      smartBulbName: 'SMBLB14',
      status: 1,
      power: 14,
      lumens: 100,
      voltage: 120,
      temperature: 96,
      weatherData: {},
      lightbulb: {
        name: 'apt4',
```

```
      manufacturer: 'Bell',
      bulb_type: 'LED',
      latitude: 37.49186437,
      longitude: -122.2329713,
      building_name: 'bld1'
    },
    switchCountWeek: 0,
    hoursOn: 561
  };
}

describe('alert handler', () => {

  it('isOverheat positive', () => {
    const bulbHistoryRecord = getBaselineBulbHistoryRecord();
    bulbHistoryRecord.temperature = 96;
    const isOverheatResult = alerts.isOverheat(bulbHistoryRecord);
    assert.equal(isOverheatResult, true);
  });

  it('isOverheat negative', () => {
    const bulbHistoryRecord = getBaselineBulbHistoryRecord();
    bulbHistoryRecord.temperature = 40;
    const isOverheatResult = alerts.isOverheat(bulbHistoryRecord);
    assert.equal(isOverheatResult, false);
  });

  it('isLongLife positive', () => {
    const bulbHistoryRecord = getBaselineBulbHistoryRecord();
    bulbHistoryRecord.hoursOn = 10501;
    const isLongLifeResult = alerts.isLongLife(bulbHistoryRecord);
    assert.equal(isLongLifeResult, true);
  });

  it('isLongLife negative', () => {
    const bulbHistoryRecord = getBaselineBulbHistoryRecord();
    const isLongLifeResult = alerts.isLongLife(bulbHistoryRecord);
    assert.equal(isLongLifeResult, false);
  });

  it('getAlertInfo - OVERHEAT Alert', () => {
    const bulbHistoryRecord = getBaselineBulbHistoryRecord();
    const alertInfo = alerts.getAlertInfo(bulbHistoryRecord, 'OVERHEAT');

    assert.equal(alertInfo.shouldSend, true);
    assert.equal(alertInfo.eventCode, 2);
    assert.equal(alertInfo.eventType, 'OVERHEAT');
  });

  it('getAlertInfo - LONGLIFE Alert', () => {
    const bulbHistoryRecord = getBaselineBulbHistoryRecord();
    const alertInfo = alerts.getAlertInfo(bulbHistoryRecord, 'LONGLIFE');

    assert.equal(alertInfo.shouldSend, true);
    assert.equal(alertInfo.eventCode, 3);
    assert.equal(alertInfo.eventType, 'LONGLIFE');
```

```
});

it('getAlertInfo - Invalid Alert', () => {
  const bulbHistoryRecord = getBaselineBulbHistoryRecord();
  const alertInfo = alerts.getAlertInfo(bulbHistoryRecord, 'OTHER');

  assert.equal(alertInfo.shouldSend, false);
  assert.equal(alertInfo.eventCode, 0);
});

it('OverheatAlert strategy', () => {
  const bulbHistoryRecord = getBaselineBulbHistoryRecord();
  bulbHistoryRecord.temperature = 96;
  const overheatAlert = new alerts.OverheatAlert();
  const alertInfo = overheatAlert.process(bulbHistoryRecord);

  assert.equal(alertInfo.shouldSend, true);
  assert.equal(alertInfo.eventCode, 2);
  assert.equal(alertInfo.eventType, 'OVERHEAT');
});

it('LongLifeAlert strategy', () => {
  const bulbHistoryRecord = getBaselineBulbHistoryRecord();
  bulbHistoryRecord.hoursOn = 10501;
  const longLifeAlert = new alerts.LongLifeAlert();
  const alertInfo = longLifeAlert.process(bulbHistoryRecord);

  assert.equal(alertInfo.shouldSend, true);
  assert.equal(alertInfo.eventCode, 3);
  assert.equal(alertInfo.eventType, 'LONGLIFE');
});

it('sendAlerts', async () => {
  const callSpy = sinon.spy();

  sandbox.stub(dbHelpers, 'getPool').callsFake(() => ({
    query() {
      console.log("called fake query");
      callSpy();
      return new Promise(resolve => resolve([]));
    }
  }));

  sandbox.stub(awsHelpers, 'sendDataToSNS').callsFake(
    function () {
      return new Promise(resolve => { console.log('Stubbed SNS send');
      callSpy();
      resolve();
    });
  });

  let alertInfoArray = [];
  const bulbHistoryRecord = getBaselineBulbHistoryRecord();
  alertInfoArray.push({
    shouldSend: true,
    message: "test",
```

```
      eventCode: 1,
      eventType: 'OVERHEAT'
    });

    await alerts.sendAlerts("region", "topicArn", bulbHistoryRecord, alertInfoArray);
    expect(callSpy.calledTwice).to.be.true;
  });

  it('sendAlert', async () => {
    const callSpy = sinon.spy();

    sandbox.stub(dbHelpers, 'getPool').callsFake(() => ({
      query() {
        console.log("called fake query");
        callSpy();
        return new Promise(resolve => resolve([]));
      }
    }));

    sandbox.stub(awsHelpers, 'sendDataToSNS').callsFake(
      function () {
        return new Promise(resolve => { console.log('Stubbed SNS send');
        callSpy();
        resolve();
      });
    });

    const bulbHistoryRecord = getBaselineBulbHistoryRecord();
    await alerts.sendAlert("region", "topicArn", bulbHistoryRecord, "test", 1, "OVERHEAT");

    expect(callSpy.calledTwice).to.be.true;
  });

  it('process alerts', async () => {

    const testRecords = {
      "Records": [
        {
          "kinesis": {
            "kinesisSchemaVersion": "1.0",
            "partitionKey": "s1",
            "sequenceNumber": "49568167373333333333333333333333333333333333333333333333333",
            "data":
"eyAic21hcnRCdWxiTmFtZSI6ICJTTUJMQjE0IiwKInRpbWVzdGFtcCI6ICIyMDExLTAxLTAxVDEzOjAwOjAwLTA4MDAiLAoicGF5bG9hZCI6ICJ7XCJ0aW1lc3RhbXBcIjpcIjIwMTgtMTEtMDFUMTM6MDA6MDAtMDgwMFwiLFwic21hcnRCdWxiTmFtZVwiOlwiU01CTEIxNFwiLFwic3RhdHVzXCI6MSxcInBvd2VyXCI6MTQsXCJsdW1lbnNcIjowLFwidm9sdGFnZVwiOjEyMCxcInRlbXBlcmF0dXJlXCI6NzYsXCJ3ZWF0aGVyRGF0YVwiOnt9LFwibGlnaHRidWxiXCI6e1wibmFtZVwiOlwiYXB0NFwiLFwibWFudWZhY3R1cmVyXCI6XCJCZWxsXCIsXCJidWxiX3R5cGVcIjpcIkxFRFwiLFwibGF0aXR1ZGVcIjozNy40OTE4NjQzNyxcImxvbmdpdHVkZVwiOi0xMjIuMjMyOTcxMyxcImJ1aWxkaW5nX25hbWVcIjpcImJsZDFcIn0sXCJzd2l0Y2hDb3VudFdlZWtcIjowLFwiaG91cnNPblwiOjB9IiB9",
            "approximateArrivalTimestamp": "1480641523.477"
          },
          "eventSource": "aws:kinesis",
          "eventVersion": "1.0",
```

```
        "eventID": "shardId-
000000000000:49568167373333333333333333333333333333333333333333333333333333",
        "eventName": "aws:kinesis:record",
        "invokeIdentityArn": "arn:aws:iam::123456789012:role/LambdaRole",
        "awsRegion": "us-east-1",
        "eventSourceARN": "arn:aws:kinesis:us-east-1:123456789012:stream/simple-stream"
       }
      ]
    };

    const callSpy = sinon.spy();

    sandbox.stub(dbHelpers, 'getPool').callsFake(() => ({
      query() {
        console.log("called fake query");
        callSpy();
        return new Promise(resolve => resolve([]));
      }
    }));

    sandbox.stub(awsHelpers, 'sendDataToSNS').callsFake(
      function () {
        return new Promise(resolve => { console.log('Stubbed SNS send');
        callSpy();
        resolve();
      });
    });

    await alerts.handler(testRecords);

    expect(callSpy.calledTwice).to.be.true;
   });
  });
});

"FILENAME: C:\dev\c3\rawRepos\imet\functions\source\aws-helpers\index.js"

var AWS = require('aws-sdk');
var async = require('async');

let decrypted;
module.exports = {
        getDecryptedKmsKey: function(awsRegion, encryptedKey) {
                return new Promise((resolve, reject) => {
                        if (decrypted) {
                                resolve(decrypted);
                        } else {
                                const kms = new AWS.KMS({region: awsRegion});
                                kms.decrypt({ CiphertextBlob: new Buffer(encryptedKey, 'base64') },
(err, data) => {
                                        if (err) {
                                                console.log('Decrypt error:', err);
                                                reject(err);
                                        }
                                        decrypted = data.Plaintext.toString('ascii');
                                        resolve(decrypted);
```

```
                        });
                    }
                });
        },

        getSagemakerPrediction: function(awsRegion, model, endpoint){
                return new Promise((resolve, reject) => {
                        const sagemakerRuntime = new AWS.SageMakerRuntime({region:
awsRegion});
                        const deepARBody = {
                                instances: [model],
                                configuration: {
                                  num_samples: 100,
                                  output_types: ['mean']
                                }
                          };
                        const body = JSON.stringify(deepARBody);
                        var params = {
                                Body: body,
                                EndpointName: endpoint,
                                Accept: 'application/json',
                                ContentType: 'application/json'
                        };
                        sagemakerRuntime.invokeEndpoint(params, function(err, data) {
                                if (err) {
                                        reject(err);
                                }
                                resolve(data.Body.toString());
                        });
                });
        },

        sendDataToFirehose: function (awsRegion, firehoseStream, data, callback) {
                var firehose = new AWS.Firehose({region: awsRegion});
                var params = {
                   DeliveryStreamName: firehoseStream,
                   Record: {Data: data }

                };
                firehose.putRecord(params, function(err, data) {
                   if (err){
                      console.log("Failed to insert data onto Firehose: ", JSON.stringify(err));
                      callback(err);
                   }
                   else{
                                console.log("Successfully placed data in onto the following firehose
stream: " + firehoseStream);
                                callback(null);
                   }
                });
        },
        sendDataToKinesisAsync: function (awsRegion, outputStream, partitionKey, data) {
           var kinesis = new AWS.Kinesis({region: awsRegion});
           var params = {
             Data: data,
             PartitionKey:  partitionKey,
```

```
            StreamName: outputStream
                };
                console.log("Putting data to Kinesis: "+data);
            return new Promise((resolve, reject) => {
                        kinesis.putRecord(params, function(err, data) {
                                if (err) {
                                        console.log("Failed to insert data onto Kinesis: ",
JSON.stringify(err));
                                        reject(err);
                                }
                                console.log("Successfully placed data onto the following stream: " +
outputStream);
                                resolve("Kinesis put success.");
                        })
                });
        },
        sendDataToS3: function (awsRegion, bucketName, key, data, callback) {
            var s3 = new AWS.S3({region: awsRegion});
                var params = {
                        Body: data,
                        Bucket: bucketName,
                        Key: key,
                        ServerSideEncryption: "AES256"
                };
                s3.putObject(params, function(err, data) {
                        if(err){
                console.log("Failed to insert data into S3: ", JSON.stringify(err));
                callback(err);
              }
              else{
                                console.log("Successfully placed data in S3 at s3://" + bucketName +
"/" + key);
                                callback(null);
              }
                });
        },

        listAllS3Objects: function (awsRegion, bucketName, prefix, allS3Objects, token, callback)
        {
        var s3 = new AWS.S3({region: awsRegion});

                var opts = {
                        Bucket: bucketName,
                        Prefix: prefix
                };
                if(token) opts.ContinuationToken = token;

                s3.listObjectsV2(opts, function(err, data){
                        if (err) console.log(err, err.stack);
                        else {
                            allS3Objects = allS3Objects.concat(data.Contents);

                            if(data.IsTruncated)
                                listAllS3Objects(s3, bucketName, prefix, allS3Objects,
data.NextContinuationToken, callback);
                            else
```

```
                        callback(allS3Objects);
                    }
            });
},

readAllS3Objects: function (awsRegion, bucketName, allS3Objects, callback)
{
var s3 = new AWS.S3({region: awsRegion});

            var readS3Object = this.readS3Object;
   var readAsync = function(s3Object, callback){
      readS3Object(awsRegion, bucketName, s3Object.Key, callback);
   }

   async.concat(allS3Objects, readAsync, function(err, objects){
      if (err){
         console.log(err, err.stack);
         callback(err);
       }else{
         callback(null, objects);
      }
   });
},

readS3Object: function (awsRegion, bucketName, key, callback)
{
            var s3 = new AWS.S3({region: awsRegion});

   s3.getObject({ Bucket: bucketName, Key: key }, function(err, data)
   {
      if (err){
         console.log(err, err.stack);
         callback(err);
       }else{
         callback(null, data.Body.toString());
      }
   });
},

sendDataToSNS: function (awsRegion, message, subject, topicArn){

            var sns = new AWS.SNS({region:awsRegion});

            // Create publish parameters
            var params = {
                        Message: message, /* required */
                        TopicArn: topicArn,
                        Subject: subject
            };

            return new Promise((resolve, reject) => {
                        sns.publish(params, (err, data) => {
                                    if (err) {
                                                console.log("Failed to publish to SNS: ", JSON.stringify(err));
                                                reject(err);
                                    }
```

```
                        console.log("MessageID is " + data.MessageId);
                        resolve(data);
                    })
            });
    },

    putItemInDynamoDB: function(awsRegion, tableName, item) {
            return new Promise((resolve,reject) => {
                var documentClient = new AWS.DynamoDB.DocumentClient({region:
awsRegion});
                var params = {
                    TableName: tableName,
                    Item: item
                };
                documentClient.put(params, function(err, data) {
               if (err){
                  console.log(err, err.stack);
                  reject(err);
               } else {
                  resolve(data);
               }
                    });
            });
    },

    updateItemInDynamoDB: function(awsRegion, params) {
            return new Promise((resolve,reject) => {
                var documentClient = new AWS.DynamoDB.DocumentClient({region:
awsRegion});

                documentClient.update(params, function(err, data) {
                    if (err) {
                        console.log(err);
                        reject(err);
                    }
                    else {
                        resolve(data);
                    }
                });
            });
    },

    getItemInDynamoDB: function(awsRegion, params) {
            return new Promise((resolve,reject) => {
                var documentClient = new AWS.DynamoDB.DocumentClient({region:
awsRegion});

                documentClient.get(params, function(err, data) {
                    if (err) {
                        console.log(err);
                        reject(err);
                    }
                    else {
                        resolve(data);
                        return data;
                    }
```

```
                        });
                });
        },

        queryDynamoDB: function(awsRegion, params, maxRecords) {
                return new Promise((resolve,reject) => {
                        let allData = [];
                        let allSize = 0;
                        var documentClient = new AWS.DynamoDB.DocumentClient({region:
awsRegion});
                        documentClient.query(params, function paginateCallback(err, data) {
                    if (err){
                       console.log(err, err.stack);
                       reject(err);
                    } else {
                                        allData.push(data);
                                        allSize += data.Items ? data.Items.length : 0;
                       if(data.LastEvaluatedKey && !(allSize >= maxRecords)) {
                                                params.ExclusiveStartKey = data.LastEvaluatedKey;
                                                documentClient.query(params, paginateCallback);
                                        } else {
                                                const result = allData.reduce((accumulator,
currentValue) => accumulator.concat(currentValue.Items), []);
                                                resolve(result);
                                        };
                                }
                        });
                });
        }
}

"FILENAME: C:\dev\c3\rawRepos\imet\functions\source\aws-helpers\package.json"

{
   "name": "aws-helpers",
   "version": "0.0.1",
   "private": true,
   "main": "index.js",
   "description": "A library of aws helper functions",
   "license": "MIT",
   "devDependencies": {
      "aws-sdk": "^2.331.0"
   },
   "dependencies": {
      "async": "~2.6.0"
   }
}

"FILENAME: C:\dev\c3\rawRepos\imet\functions\source\bulb-history-record\index.js"

"use strict";

module.exports = class BulbHistoryRecord {
   constructor (timestamp, smartBulbName, status, watts, lumens, voltage, temp) {
        this.timestamp = timestamp;
        this.smartBulbName = smartBulbName;
```

```
            this.status = status;
            this.power = watts;
            this.lumens = lumens;
            this.voltage = voltage;
            this.temperature = temp;
    }
}
"FILENAME: C:\dev\c3\rawRepos\imet\functions\source\bulb-history-record\package.json"

{
    "name": "bulb-history-record",
    "version": "0.0.1",
    "private": true,
    "main": "index.js",
    "description": "Data model for bulb history",
    "license": "MIT",
    "dependencies": {}
}

"FILENAME: C:\dev\c3\rawRepos\imet\functions\source\database-helpers\index.js"

const util = require('util')
const mysql = require('mysql')
const awsHelpers = require('aws-helpers');

const AWS_REGION = process.env.AWS_REGION || 'us-east-1';
const encryptedMySQLPass = process.env.MySQLPass || 'test';

let pool;
module.exports = {
    getPool: function() {
        return new Promise((resolve) => {
            if (pool) {
                resolve(pool);
            } else {
                try {
                    awsHelpers.getDecryptedKmsKey(AWS_REGION, encryptedMySQLPass).then(key => {
                        pool = mysql.createPool({
                            connectionLimit: 10,
                            connectTimeout: 5 * 60 * 1000,
                            acquireTimeout: 5 * 60 * 1000,
                            timeout: 5 * 60 * 1000,
                            host: process.env.MySQLHost,
                            user: process.env.MySQLUser,
                            password: key,
                            database: process.env.MySQLDB
                        });
                        pool.query = util.promisify(pool.query);
                        resolve(pool);
                    }).catch(err => console.log("POOL ERROR: "+err));

                } catch (err) {
                    log.error("Error creating DB connection pool");
                    reject(err);
                }
            }
```

```
        });
    }
};
"FILENAME: C:\dev\c3\rawRepos\imet\functions\source\database-helpers\package.json"

{
    "name": "database-helpers",
    "version": "0.0.1",
    "private": true,
    "main": "index.js",
    "description": "A library of aws helper functions",
    "license": "MIT",
    "devDependencies": {},
    "dependencies": {
        "async": "~2.6.0",
        "mysql": "^2.16.0",
        "aws-helpers": "file:../aws-helpers"
    }
}

"FILENAME: C:\dev\c3\rawRepos\imet\functions\source\enhance-lambda\.eslintrc.js"

module.exports = {
  env: {
    es6: true,
    node: true,
    mocha: true,
    jest: true
  },
  extends: ['airbnb-base'],
  rules: {
    'arrow-parens': ['error', 'as-needed'],
    'comma-dangle': ['error', 'never'],
    'function-paren-newline': 'off',
    'no-underscore-dangle': 'off',
    'prefer-destructuring': 'off',
    'no-console': 'off',
    'no-await-in-loop': 'off',
    'import/no-extraneous-dependencies': [
      'warn',
      { devDependencies: false }
    ],
    'no-restricted-syntax': [
      'error',
      {
        selector: 'ForInStatement',
        message: 'for..in loops iterate over the entire prototype chain, which is virtually never what you want.
Use Object.{keys,values,entries}, and iterate over the resulting array.',
      },
      {
        selector: 'LabeledStatement',
        message: 'Labels are a form of GOTO; using them makes code confusing and hard to maintain and
understand.',
      },
      {
        selector: 'WithStatement',
```

```
      message: '`with` is disallowed in strict mode because it makes code impossible to predict and
optimize.',
    },
   ],
  },
  overrides: [
   {
    files: '**/*.spec.js',
    rules: {
     'no-unused-expressions': 'off',
     'prefer-arrow-callback': 'off',
     'import/no-extraneous-dependencies': [
      'error',
      { devDependencies: true }
     ]
    }
   }
  ]
};

"FILENAME: C:\dev\c3\rawRepos\imet\functions\source\enhance-lambda\index.js"

const awsHelpers = require('aws-helpers');
const dbHelpers = require('database-helpers');
const request = require('request-promise-native');
const subMinutes = require('date-fns/sub_minutes');

async function getWeatherData(measurementRecord) {
  const options = {
   method: 'POST',
   uri: process.env.WeatherUri,
   qs: { action: 'evalMetrics' },
   headers:
   {
    'Cache-Control': 'no-cache',
    Authorization: process.env.WeatherAuth,
    'Content-Type': 'application/json',
    Accept: 'application/json'
   },
   body:
   {
    spec:
     {
      ids: [measurementRecord.smartBulbName],
      expressions: ['AmbientTemperature'],
      start: subMinutes(measurementRecord.timestamp, 15),
      end: measurementRecord.timestamp,
      interval: 'FIVE_MINUTE'
     }
   },
   json: true,
   timeout: 1000
  };

  try {
   const callData = await request(options);
```

```
    return callData.result;
  } catch (err) {
    console.log('Caught error with API POST', err);
  }

  return new Promise(resolve => resolve({}));
}

async function getLightbulbData(bulbName) {
  const queryString = 'SELECT bulb.`name` as smartBulbName, '
  + 'bulb.manufacturer,'
  + 'bulb.bulb_type as bulbType,'
  + 'bulb.latitude,'
  + 'bulb.longitude,'
  + 'fix.`name` as fixtureName,'
  + 'apt.`name` as apartmentName,'
  + 'apt.building_name as buildingName'
  + ' FROM smart_bulb as bulb'
  + ' LEFT JOIN smart_bulb_fixture as sb_fix on sb_fix.smart_bulb_name = bulb.`name`'
  + ' LEFT JOIN fixture as fix on sb_fix.fixture_name = fix.`name`'
  + ' LEFT JOIN apartment as apt on fix.apartment_name = apt.`name`'
  + ' WHERE bulb.`name` = ?';

  const pool = await dbHelpers.getPool();
  const queryResult = await pool.query(queryString, [bulbName]);
  if (queryResult.length > 0) {
    return queryResult[0];
  }
  throw new Error('No results found for given lightbulb name');
}

async function enhanceData(measurementRecord) {
  const copy = JSON.parse(JSON.stringify(measurementRecord));
  try {
    const lightbulbData = await getLightbulbData(measurementRecord.smartBulbName);
    const weatherData = await getWeatherData(measurementRecord);
    copy.weatherData = weatherData;
    copy.lightbulb = lightbulbData;
    return copy;
  } catch (err) {
    console.error('Caught error retrieving enhancement data.', err);
    return copy;
  }
}

exports.handler = async event => {
  const awsRegion = process.env.AWS_REGION;
  const outputStream = process.env.OutputStream;
  console.log('Received event: ', JSON.stringify(event, null, 2));
  for (const record of event.Records) {
    try {
      const dataString = Buffer.from(record.kinesis.data, 'base64').toString('utf8');
      const lightbulbMeasurementRecord = JSON.parse(dataString);

      const updatedData = await enhanceData(lightbulbMeasurementRecord);
      const serializedUpdatedData = JSON.stringify(updatedData);
```

```
      console.log('Enhanced data: ', serializedUpdatedData);
      const result = await awsHelpers.sendDataToKinesisAsync(awsRegion,
        outputStream,
        lightbulbMeasurementRecord.smartBulbName,
        serializedUpdatedData);
      console.log(`Completed with: ${result}`);
    } catch (err) {
      console.log(`Error enhancing modeled data: ${err}`);
      throw new Error(`Unable to complete processing: ${err}`);
    }
  }
  return 'Finished processing records';
};
```

"FILENAME: C:\dev\c3\rawRepos\imet\functions\source\enhance-lambda\package.json"

```
{
    "name": "enhance-lambda",
    "version": "0.0.1",
    "description": "A simple lambda that add appends data to price history records",
    "license": "MIT",
    "scripts": {
        "test": "nyc --check-coverage --lines 80 --functions 80 --branches 80 --reporter=cobertura mocha -R
mocha-multi-reporters --recursive test",
        "lint": "eslint . --fix",
        "prettier": "prettier --single-quote --write '**/*.js'"
    },
    "devDependencies": {
        "chai": "^4.2.0",
        "chai-as-promised": "7.1.1",
        "expect.js": "^0.3.1",
        "eslint": "^5.6.1",
        "eslint-config-airbnb-base": "^13.1.0",
        "eslint-plugin-import": "^2.14.0",
        "mocha": "^5.2.0",
        "mocha-multi-reporters": "^1.1.7",
        "nyc": "^13.0.1",
        "prettier": "^1.6.1",
        "sinon": "^6.3.5"
    },
    "dependencies": {
        "aws-helpers": "file:../aws-helpers",
        "database-helpers": "file:../database-helpers",
        "request-promise-native": "^1.0.5",
        "request": "^2.88.0",
        "date-fns": "^1.29.0"
    },
    "prettier": {
        "singleQuote": true
    },
    "private": true
}
```

"FILENAME: C:\dev\c3\rawRepos\imet\functions\source\enhance-lambda\test\enhance.spec.js"

```
const sinon = require('sinon');
```

```
const chai = require('chai');
const awsHelpers = require('aws-helpers');
const dbHelpers = require('database-helpers');
const chaiAsPromised = require('chai-as-promised');
const enhanceLambda = require('../index.js');

chai.use(chaiAsPromised);
const { expect } = chai;

describe('Enhance Lambda Test', () => {
 const sandbox = sinon.createSandbox();

 beforeEach(() => {

 });

 afterEach(() => {
  sandbox.restore();
 });

 describe('Enhance handler', () => {
  it('passes through the kinesis event to outbound kinesis', async () => {
   const callSpy = sinon.spy();

   sandbox.stub(awsHelpers, 'sendDataToKinesisAsync')
    .callsFake(function () { console.log('Stubbed kinesis put'); callSpy(); return true; });

   sandbox.stub(dbHelpers, 'getPool').callsFake(() => ({
    query() {
     return new Promise(resolve => resolve([{ latitude: 5.0, longitude: -2.0 }]));
    }
   }));

   const testRecord = {
    timestamp: '2016-10-23T14:00:00.000Z',
    smartBulbName: 'SMBLB1',
    status: 1,
    power: 123,
    lumens: 234,
    voltage: 345,
    temperature: 456
   };

   const testEvent = {
    Records: [
     {
      kinesis: {
       kinesisSchemaVersion: '1.0',
       partitionKey: 's1',
       sequenceNumber: '49568167373333333333333333333333333333333333333333333333333333333333',
       data: Buffer.from(JSON.stringify(testRecord)).toString('base64'),
       approximateArrivalTimestamp: 1480641523.477
      },
      eventSource: 'aws:kinesis',
      eventVersion: '1.0',
```

```
          eventID: 'shardId-
000000000000:49568167373333333333333333333333333333333333333333333333333333',
          eventName: 'aws:kinesis:record',
          invokeIdentityArn: 'arn:aws:iam::123456789012:role/LambdaRole',
          awsRegion: 'us-east-1',
          eventSourceARN: 'arn:aws:kinesis:us-east-1:123456789012:stream/simple-stream'
        }
      ]
    };

    const res = await enhanceLambda.handler(testEvent);
    expect(callSpy.calledOnce).to.be.true;
    expect(res).to.eq('Finished processing records');
  });

  it('fails the event if output kinesis put fails', async () => {
    sandbox.stub(awsHelpers, 'sendDataToKinesisAsync').throws('TestError');
    sandbox.stub(dbHelpers, 'getPool').callsFake(() => ({
      query() {
        return new Promise(resolve => resolve([{ latitude: 5.0, longitude: -2.0 }]));
      }
    }));

    const testEvent = {
      Records: [
        {
          kinesis: {
            kinesisSchemaVersion: '1.0',
            partitionKey: 's1',
            sequenceNumber: '49568167373333333333333333333333333333333333333333333333333333',
            data: Buffer.from(JSON.stringify({ bulbId: 123, otherField: 'this is a test' })).toString('base64'),
            approximateArrivalTimestamp: 1480641523.477
          },
          eventSource: 'aws:kinesis',
          eventVersion: '1.0',
          eventID: 'shardId-
000000000000:49568167373333333333333333333333333333333333333333333333333333',
          eventName: 'aws:kinesis:record',
          invokeIdentityArn: 'arn:aws:iam::123456789012:role/LambdaRole',
          awsRegion: 'us-east-1',
          eventSourceARN: 'arn:aws:kinesis:us-east-1:123456789012:stream/simple-stream'
        }
      ]
    };
    try {
      const res = await enhanceLambda.handler(testEvent);
      console.log("UH OK");
      expect(res).to.exist();
    } catch (err) {
      console.log(err);
      expect(err).to.be.an('error');
    }
  });
});

  it('fails the event if no bulb data is found for measurement name', async () => {
```

```
    const callSpy = sinon.spy();

    sandbox.stub(awsHelpers, 'sendDataToKinesisAsync')
      .callsFake(function () { console.log('Stubbed kinesis put'); callSpy(); return true; });
    sandbox.stub(dbHelpers, 'getPool').callsFake(() => ({
      query() {
        return new Promise(resolve => resolve([]));
      }
    }));

    const testEvent = {
      Records: [
        {
          kinesis: {
            kinesisSchemaVersion: '1.0',
            partitionKey: 's1',
            sequenceNumber: '49568167373333333333333333333333333333333333333333333333',
            data: Buffer.from(JSON.stringify({ bulbId: 123, otherField: 'this is a test' })).toString('base64'),
            approximateArrivalTimestamp: 1480641523.477
          },
          eventSource: 'aws:kinesis',
          eventVersion: '1.0',
          eventID: 'shardId-
000000000000:49568167373333333333333333333333333333333333333333333333',
          eventName: 'aws:kinesis:record',
          invokeIdentityArn: 'arn:aws:iam::123456789012:role/LambdaRole',
          awsRegion: 'us-east-1',
          eventSourceARN: 'arn:aws:kinesis:us-east-1:123456789012:stream/simple-stream'
        }
      ]
    };
    try {
      const res = await enhanceLambda.handler(testEvent);
      expect(res).to.exist();
    } catch (err) {
      expect(err).to.be.an('error');
    }
  });
});

"FILENAME: C:\dev\c3\rawRepos\imet\functions\source\ingest-lambda\.eslintrc.js"

module.exports = {
  env: {
    es6: true,
    node: true,
    mocha: true,
    jest: true
  },
  extends: ['airbnb-base'],
  rules: {
    'arrow-parens': ['error', 'as-needed'],
    'comma-dangle': ['error', 'never'],
    'function-paren-newline': 'off',
    'no-underscore-dangle': 'off',
    'prefer-destructuring': 'off',
```

```
    'no-console': 'off',
    'no-await-in-loop': 'off',
    'import/no-extraneous-dependencies': [
      'warn',
      { devDependencies: false }
    ],
    'no-restricted-syntax': [
      'error',
      {
        selector: 'ForInStatement',
        message: 'for..in loops iterate over the entire prototype chain, which is virtually never what you want.
Use Object.{keys,values,entries}, and iterate over the resulting array.',
      },
      {
        selector: 'LabeledStatement',
        message: 'Labels are a form of GOTO; using them makes code confusing and hard to maintain and
understand.',
      },
      {
        selector: 'WithStatement',
        message: '`with` is disallowed in strict mode because it makes code impossible to predict and
optimize.',
      },
    ],
  },
  overrides: [
    {
      files: '**/*.spec.js',
      rules: {
        'no-unused-expressions': 'off',
        'prefer-arrow-callback': 'off',
        'import/no-extraneous-dependencies': [
          'error',
          { devDependencies: true }
        ]
      }
    }
  ]
};

"FILENAME: C:\dev\c3\rawRepos\imet\functions\source\ingest-lambda\index.js"

const awsHelpers = require('aws-helpers');
const crypto = require('crypto');

exports.handler = async event => {
  console.log('Ingest handler received event:', event);
  const outputStream = process.env.OutputStream;
  const awsRegion = process.env.AWS_REGION;
  for (const record of event.Records) {
    try {
      if (Object.prototype.hasOwnProperty.call(record, 'body')) {
        console.log(`Processing ingest message ${record.body}`);
        const partitionKey = crypto.createHash('md5').update(new Date().toISOString()).digest('hex');
        await awsHelpers.sendDataToKinesisAsync(awsRegion, outputStream, partitionKey, record.body);
      } else {
```

```
        throw new Error(`Unexpected message format for ${JSON.stringify(record)}`);
      }
    } catch (err) {
      console.log(`Failed to process Kinesis event: ${err}`);
      throw new Error(`Failed to process Kinesis event: ${err}`);
    }
  }
  return 'Successfully processed Kinesis event';
};
```

"FILENAME: C:\dev\c3\rawRepos\imet\functions\source\ingest-lambda\package.json"

```
{
   "name": "ingest-lambda",
   "version": "0.0.1",
   "description": "A simple lambda that queries the AWS ECS Spot Pricing API and pushes the results to
Kinesis",
   "license": "MIT",
   "scripts": {
      "test": "nyc --check-coverage --lines 80 --functions 80 --branches 80 --reporter=cobertura mocha -R
mocha-multi-reporters --recursive test",
      "lint": "eslint . --fix",
      "prettier": "prettier --single-quote --write '**/*.js'"
   },
   "devDependencies": {
      "chai": "^4.2.0",
      "chai-as-promised": "7.1.1",
      "expect.js": "^0.3.1",
      "eslint": "^5.6.1",
      "eslint-config-airbnb-base": "^13.1.0",
      "eslint-plugin-import": "^2.14.0",
      "mocha": "^5.2.0",
      "mocha-multi-reporters": "^1.1.7",
      "nyc": "^13.0.1",
      "prettier": "^1.6.1",
      "sinon": "^6.3.5"
   },
   "dependencies": {
      "aws-helpers": "file:../aws-helpers"
   },
   "prettier": {
      "singleQuote": true
   },
   "private": true
}
```

"FILENAME: C:\dev\c3\rawRepos\imet\functions\source\ingest-lambda\test\ingest.spec.js"

```
const sinon = require('sinon');
const chai = require('chai');
const awsHelpers = require('aws-helpers');
const chaiAsPromised = require('chai-as-promised');
const ingestLambda = require('../index.js');

chai.use(chaiAsPromised);
const { expect } = chai;
```

```
describe('Ingest Lambda Test', () => {
  const sandbox = sinon.createSandbox();

  beforeEach(() => {
  });

  afterEach(() => {
    sandbox.restore();
  });

  describe('ingest handler', () => {
    it('passes through the sqs event to kinesis', async () => {
      const callSpy = sinon.spy();

      sandbox.stub(awsHelpers, 'sendDataToKinesisAsync')
        .callsFake(function () { console.log('Stubbed kinesis put'); callSpy(); });

      const testEvent = {
        Records:
              [{
                messageId: 'c02dc6c7-6c27-4da9-88d8-18fae87e7490',
                receiptHandle:
'AQEBRdrthlUT9x5Iu/vBgvMl01KaK+h//7YcML0F2bXNdsnM0HGjmfXTXmRRRwxU+1+esRZ5yJrpdJ
lKGRMHAw0O6BVis3QF6aQMC1FMAJOK5Z1wv10OtvTgAiEHXGN+9DbgNqF0v6fh18YY3lxORZw
p/u+0aQ5glIC2vlm+CEUMl2lH39rmNYztaqJk3fsay8tC/imvyRq35sI1CNpX5GoCLOiGYvKaH3MVnfyh
VcGF2qUiu5y5gAoRTtOTeiifzwmn8sfQanrLCQ+c8B5QTHEtVlE3T3ULj0C3CUajWJD7PXFUw4BBE
QdjyWngpNaUWjpkONs4Qwp8TD3ZS+eiUrlrznumuv0UQtmY/vst0TOwvKZOQFYA8lcPJ2ocQy3UXV
VbTC8T8zHP51b29KFfP7iXcQ==',
                body: '{"test":"This is a test body"}',
                attributes: {},
                messageAttributes: {},
                md5OfBody: '85d8601e6c7ea5d4d9b8809b2e5062ff',
                eventSource: 'aws:sqs',
                eventSourceARN: 'arn:aws:sqs:us-east-1:466081484468:BatchRequestQueue',
                awsRegion: 'us-east-1'
              }]
      };

      await ingestLambda.handler(testEvent);
      expect(callSpy.calledOnce).to.be.true;
    });

    it('does not pass through if sqs event has no body', async () => {
      const callSpy = sinon.spy();
      sandbox.stub(awsHelpers, 'sendDataToKinesis')
        .callsFake(function () { console.log('Stubbed kinesis put'); callSpy(); });

      const testEvent = {
        Records: [
          { notTheBody: JSON.stringify({ test: 'someData' }) }
        ]
      };

      try {
        const res = await ingestLambda.handler(testEvent);
```

```
      // Should throw exception and if it gets to here something went wrong
      expect(res).to.not.exist();
    } catch (err) {
      expect(err).to.be.an('error');
    }
  });
 });
});
```

"FILENAME: C:\dev\c3\rawRepos\imet\functions\source\model-lambda\.eslintrc.js"

```
module.exports = {
 env: {
  es6: true,
  node: true,
  mocha: true,
  jest: true
 },
 extends: ['airbnb-base'],
 rules: {
  'arrow-parens': ['error', 'as-needed'],
  'comma-dangle': ['error', 'never'],
  'function-paren-newline': 'off',
  'no-underscore-dangle': 'off',
  'prefer-destructuring': 'off',
  'no-console': 'off',
  'no-await-in-loop': 'off',
  'import/no-extraneous-dependencies': [
   'warn',
   { devDependencies: false }
  ],
  'no-restricted-syntax': [
   'error',
   {
    selector: 'ForInStatement',
    message: 'for..in loops iterate over the entire prototype chain, which is virtually never what you want.
Use Object.{keys,values,entries}, and iterate over the resulting array.',
   },
   {
    selector: 'LabeledStatement',
    message: 'Labels are a form of GOTO; using them makes code confusing and hard to maintain and
understand.',
   },
   {
    selector: 'WithStatement',
    message: '`with` is disallowed in strict mode because it makes code impossible to predict and
optimize.',
   },
  ],
 },
 overrides: [
  {
   files: '**/*.spec.js',
   rules: {
    'no-unused-expressions': 'off',
    'prefer-arrow-callback': 'off',
```

```
        'import/no-extraneous-dependencies': [
          'error',
          { devDependencies: true }
        ]
      }
    }
  ]
};
```

"FILENAME: C:\dev\c3\rawRepos\imet\functions\source\model-lambda\index.js"

```
const awsHelpers = require('aws-helpers');
const crypto = require('crypto');
const dbHelpers = require('database-helpers');
const BulbHistoryRecord = require('bulb-history-record');
const dateParse = require('date-fns/parse');

function modelData(ingestRecord) {
  const parsedDate = dateParse(ingestRecord.TS);
  const record = new BulbHistoryRecord(
    parsedDate,
    ingestRecord.SN,
    ingestRecord.Status,
    ingestRecord.Watts,
    ingestRecord.Lumens,
    ingestRecord.Voltage,
    ingestRecord.Temp
  );

  console.log(`Adding bulb history record: ${JSON.stringify(record)}`);
  return record;
}

async function insertMeasurementIntoRDS(modeledRecord) {
  console.log('Inserting modeled data into database');
  const queryString = 'INSERT INTO `smart_bulb_measurement`'
  + ' (`timestamp`,'
  + ' `smart_bulb_name`,'
  + ' `lumens`,'
  + ' `power`,'
  + ' `temperature`,'
  + ' `voltage`,'
  + ' `status`)'
  + ' VALUES'
  + ' (?,?,?,?,?,?,?);';

  const pool = await dbHelpers.getPool();
  try {
    await pool.query(queryString, [modeledRecord.timestamp,
      modeledRecord.smartBulbName,
      modeledRecord.lumens,
      modeledRecord.power,
      modeledRecord.temperature,
      modeledRecord.voltage,
      modeledRecord.status]);
  } catch (err) {
```

```
    throw new Error(`Error inserting modeled measurement into database: ${err}`);
  }
  return true;
}

exports.handler = async event => {
  const awsRegion = process.env.AWS_REGION;
  const outputStream = process.env.OutputStream;
  console.log('Received event:', JSON.stringify(event, null, 2));
  for (const record of event.Records) {
    try {
      const serializedData = Buffer.from(record.kinesis.data, 'base64').toString('utf8');
      const ingestRecord = JSON.parse(serializedData);

      console.log(`Raw data: ${JSON.stringify(ingestRecord)}`);

      const modeledData = modelData(ingestRecord);
      await insertMeasurementIntoRDS(modeledData);
      console.log('Sending data to kinesis');
      const result = await awsHelpers.sendDataToKinesisAsync(awsRegion,
        outputStream,
        modeledData.smartBulbName,
        JSON.stringify(modeledData));
      console.log(`Completed with: ${result}`);
    } catch (err) {
      console.error('Encountered error when modeling data', err);
    }
  }
  return 'Finished processing records';
};
```

"FILENAME: C:\dev\c3\rawRepos\imet\functions\source\model-lambda\package.json"

```
{
   "name": "model-lambda",
   "version": "0.0.1",
   "description": "A simple lambda that converts the raw ingest data into the common pipeline model",
   "license": "MIT",
   "scripts": {
      "test": "nyc --check-coverage --lines 80 --functions 80 --branches 80 --reporter=cobertura mocha -R
mocha-multi-reporters --recursive test",
      "lint": "eslint . --fix",
      "prettier": "prettier --single-quote --write '**/*.js'"
   },
   "devDependencies": {
      "chai": "^4.2.0",
      "chai-as-promised": "7.1.1",
      "expect.js": "^0.3.1",
      "eslint": "^5.6.1",
      "eslint-config-airbnb-base": "^13.1.0",
      "eslint-plugin-import": "^2.14.0",
      "mocha": "^5.2.0",
      "mocha-multi-reporters": "^1.1.7",
      "nyc": "^13.0.1",
      "prettier": "^1.6.1",
      "sinon": "^6.3.5"
```

```
  },
  "dependencies": {
    "date-fns": "^1.29.0",
    "aws-helpers": "file:../aws-helpers",
    "database-helpers": "file:../database-helpers",
    "bulb-history-record": "file:../bulb-history-record"
  },
  "private": true
}
```

"FILENAME: C:\dev\c3\rawRepos\imet\functions\source\model-lambda\test\model.spec.js"

```
const sinon = require('sinon');
const chai = require('chai');
const awsHelpers = require('aws-helpers');
const dbHelpers = require('database-helpers');
const chaiAsPromised = require('chai-as-promised');
const modelLambda = require('../index.js');

chai.use(chaiAsPromised);
const { expect } = chai;

describe('Model Lambda Test', () => {
 const sandbox = sinon.createSandbox();

 beforeEach(() => {
 });

 afterEach(() => {
  sandbox.restore();
 });

 describe('model handler', () => {
  it('passes through the kinesis event to kinesis', async () => {
   const callSpy = sinon.spy();

   sandbox.stub(awsHelpers, 'sendDataToKinesisAsync')
    .callsFake(function () { return new Promise(resolve => { console.log('Stubbed kinesis put'); callSpy();
resolve(); }); });

   sandbox.stub(dbHelpers, 'getPool').callsFake(() => ({
    query() {
     console.log('Called fake query');
     return new Promise(resolve => resolve([]));
    }
   }));

   const testEvent = {
    Records:
        [{
         kinesis: {
          kinesisSchemaVersion: '1.0',
          partitionKey: 's1',
          sequenceNumber: '49568167373333333333333333333333333333333333333333333333333333',
          data: Buffer.from(JSON.stringify({ SN: 123, TS: 'this is a test' })).toString('base64'),
          approximateArrivalTimestamp: 1480641523.477
```

```
            },
            eventSource: 'aws:kinesis',
            eventVersion: '1.0',
            eventID: 'shardId-
000000000000:49568167373333333333333333333333333333333333333333333333333333',
            eventName: 'aws:kinesis:record',
            invokeIdentityArn: 'arn:aws:iam::123456789012:role/LambdaRole',
            awsRegion: 'us-east-1',
            eventSourceARN: 'arn:aws:kinesis:us-east-1:123456789012:stream/simple-stream'

          }]
    };
    const res = await modelLambda.handler(testEvent);
    console.log(res);
    expect(callSpy.calledOnce).to.be.true;
  });

  it('does not pass through events that are not from kinesis', () => {
    const callSpy = sinon.spy();

    sandbox.stub(awsHelpers, 'sendDataToKinesisAsync')
      .callsFake(function () { return new Promise(res => { console.log('Stubbed kinesis put'); callSpy();
res(); }); });

    sandbox.stub(dbHelpers, 'getPool').callsFake(() => ({
      query() {
        console.log('Called fake query');
        return new Promise((resolve, reject) => reject(new Error('Insert failed')));
      }
    }));

    const testEvent = {
      Records:
          [{
            kinesis: {
              kinesisSchemaVersion: '1.0',
              partitionKey: 's1',
              sequenceNumber: '49568167373333333333333333333333333333333333333333333333333333',
              data: Buffer.from(JSON.stringify({ SN: 123, TS: 'this is a test' })).toString('base64'),
              approximateArrivalTimestamp: 1480641523.477
            },
            eventSource: 'aws:kinesis',
            eventVersion: '1.0',
            eventID: 'shardId-
000000000000:49568167373333333333333333333333333333333333333333333333333333',
            eventName: 'aws:kinesis:record',
            invokeIdentityArn: 'arn:aws:iam::123456789012:role/LambdaRole',
            awsRegion: 'us-east-1',
            eventSourceARN: 'arn:aws:kinesis:us-east-1:123456789012:stream/simple-stream'

          }]
    };

    modelLambda.handler(testEvent);
    expect(callSpy.notCalled).to.be.true;
  });
```

```
  });
});

"FILENAME: C:\dev\c3\rawRepos\imet\functions\source\transform-lambda\.eslintrc.js"

module.exports = {
  env: {
    es6: true,
    node: true,
    mocha: true,
    jest: true
  },
  extends: ['airbnb-base'],
  rules: {
    'arrow-parens': ['error', 'as-needed'],
    'comma-dangle': ['error', 'never'],
    'function-paren-newline': 'off',
    'no-underscore-dangle': 'off',
    'prefer-destructuring': 'off',
    'no-console': 'off',
    'no-await-in-loop': 'off',
    'import/no-extraneous-dependencies': [
      'warn',
      { devDependencies: false }
    ],
    'no-restricted-syntax': [
      'error',
      {
        selector: 'ForInStatement',
        message: 'for..in loops iterate over the entire prototype chain, which is virtually never what you want.
Use Object.{keys,values,entries}, and iterate over the resulting array.',
      },
      {
        selector: 'LabeledStatement',
        message: 'Labels are a form of GOTO; using them makes code confusing and hard to maintain and
understand.',
      },
      {
        selector: 'WithStatement',
        message: '`with` is disallowed in strict mode because it makes code impossible to predict and
optimize.',
      },
    ],
  },
  overrides: [
    {
      files: '**/*.spec.js',
      rules: {
        'no-unused-expressions': 'off',
        'prefer-arrow-callback': 'off',
        'import/no-extraneous-dependencies': [
          'error',
          { devDependencies: true }
        ]
      }
    }
```

```
  ]
};

"FILENAME: C:\dev\c3\rawRepos\imet\functions\source\transform-lambda\index.js"

const awsHelpers = require('aws-helpers');
const dbHelpers = require('database-helpers');
const hoursOnFunction = require('./features/hours-on-feature');
const weekSwitchCountFunction = require('./features/week-switch-count-feature');
const doesFailFunction = require('./features/does-fail-feature.js');
const { getTransformedData, getPredictionTransformedData } = require('./lib/transformed-data-retriever');
const { buildDeepArModel } = require('./lib/deep-ar-model-builder');
const { appendPredictionDataToModel } = require('./lib/deep-ar-request-appender');

const addDoesFail = doesFailFunction.addDoesFail;

async function updatePredictionTable(prediction, enhancedRecord) {
  const queryString = 'INSERT INTO `smart_bulb_prediction`'
  + ' (`prediction`,'
  + ' `time_stamp`,'
  + ' `smart_bulb_name`)'
  + ' VALUES'
  + ' (?,?,?);';

  const pool = await dbHelpers.getPool();
  try {
    await pool.query(queryString, [prediction,
      enhancedRecord.timestamp,
      enhancedRecord.smartBulbName]);
  } catch (err) {
    throw new Error(`Error inserting modeled measurement into database: ${err}`);
  }
  return true;
}

const getPrediction = async function (enhancedRecord) {
  const awsRegion = process.env.AWS_REGION;
  const predictionEndpoint = process.env.PredictionEndpoint;
  try {
    const getResults = await getTransformedData(enhancedRecord.smartBulbName,
enhancedRecord.timestamp);
    const predictionData = await getPredictionTransformedData(enhancedRecord.smartBulbName,
enhancedRecord.timestamp);
    const model = buildDeepArModel(getResults.bulbData, getResults.windowParameters);
    console.log("Built AR model");
    const requestModel = appendPredictionDataToModel(predictionData.bulbData,
predictionData.windowParameters, model);
    console.log("Appended prediction data to model");
    const predictionResult = await awsHelpers.getSagemakerPrediction(awsRegion, requestModel,
predictionEndpoint);
    console.log("Got prediction");
    const parsedPrediction = JSON.parse(predictionResult).predictions[0];
    // The last prediction is the most recent and represents the current timestamp
    const prediction = parsedPrediction.mean[parsedPrediction.mean.length - 1];
    return prediction;
  } catch (err) {
```

```
    console.log("Unable to build prediction.  Setting as null.");
    return null;
  }
};

async function transformMLData(enhancedRecord, prediction) {
  try {
    const hoursOnTotal = await hoursOnFunction.getHoursOn(enhancedRecord);
    const switchCountWeekTotal = await weekSwitchCountFunction(enhancedRecord);
    const payload = enhancedRecord;
    payload.switchCountWeek = switchCountWeekTotal;
    payload.hoursOn = hoursOnTotal;
    payload.prediction = prediction;
    const transformedRecord = {
      smartBulbName: enhancedRecord.smartBulbName,
      timestamp: enhancedRecord.timestamp,
      payload: JSON.stringify(payload)
    };
    return transformedRecord;
  } catch (err) {
    console.error('Caught error adding ML input data', err);
  }
  return null;
}

async function dynamoPush(table, data) {
  const awsRegion = process.env.AWS_REGION;
  try {
    await awsHelpers.putItemInDynamoDB(awsRegion, table, data);
  } catch (err) {
    console.error('Caught error pushing to dynamo', err);
  }
}

const handler = async event => {
  const dynamoDBML = process.env.DynamoDBML;
  const awsRegion = process.env.AWS_REGION;
  const outputStream = process.env.OutputStream;
  console.log('Received event:', JSON.stringify(event, null, 2));

  for (const record of event.Records) {
    try {
      const dataString = Buffer.from(record.kinesis.data, 'base64').toString('utf8');
      const enhancedRecord = JSON.parse(dataString);
      const prediction = await getPrediction(enhancedRecord);
      console.log("Got prediction index");
      if(prediction != null) {
        await updatePredictionTable(prediction, enhancedRecord);
        console.log("Updated table with prediction");
      }

      const transformedData = await transformMLData(enhancedRecord, prediction);
      console.log(`Got data: ${JSON.stringify(transformedData)}`);
      await dynamoPush(dynamoDBML, transformedData);
      console.log('Pushed measurements to ML dynamo DB');
      await addDoesFail(enhancedRecord);
```

```
      const result = await awsHelpers.sendDataToKinesisAsync(awsRegion,
        outputStream,
        enhancedRecord.smartBulbName,
        JSON.stringify(transformedData));
      console.log(`Completed with: ${result}`);
    } catch (err) {
      console.log(`Error transforming enhanced data: ${err}`);
      throw new Error(`Error transforming enhanced data: ${err}`);
    }
  }
  return 'Completed record processing';
};

module.exports = {
  handler,
  getPrediction
};
```

"FILENAME: C:\dev\c3\rawRepos\imet\functions\source\transform-lambda\package.json"

```
{
    "name": "transform-lambda",
    "version": "0.0.1",
    "description": "A simple lambda that appends history records to the SageMaker training files",
    "license": "MIT",
    "scripts": {
        "test": "nyc --check-coverage --lines 80 --functions 80 --branches 80 --reporter=cobertura mocha -R
mocha-multi-reporters --recursive test",
        "lint": "eslint . --fix",
        "prettier": "prettier --single-quote --write '**/*.js'"
    },
    "devDependencies": {
        "chai": "^4.2.0",
        "chai-as-promised": "7.1.1",
        "expect.js": "^0.3.1",
        "eslint": "^5.6.1",
        "eslint-config-airbnb-base": "^13.1.0",
        "eslint-plugin-import": "^2.14.0",
        "mocha": "^5.2.0",
        "mocha-multi-reporters": "^1.1.7",
        "nyc": "^13.0.1",
        "prettier": "^1.6.1",
        "sinon": "^6.3.5"
    },
    "dependencies": {
        "aws-helpers": "file:../aws-helpers",
        "database-helpers": "file:../database-helpers",
        "date-fns": "^1.29.0"
    },
    "private": true
}
```

"FILENAME: C:\dev\c3\rawRepos\imet\functions\source\transform-lambda\features\does-fail-feature.js"

```
const dbHelpers = require('database-helpers');
const awsHelpers = require('aws-helpers');
```

```
const subDays = require('date-fns/sub_days');

async function insertAlertEventIntoRDS(enhancedRecord, eventCode, eventType) {
  console.log('Inserting alert event into database');
  const queryString = 'INSERT INTO `smart_bulb_event`'
  + ' (`smart_bulb_name`,'
  + ' `event_code`,'
  + ' `event_type`,'
  + ' `start`)'
  + ' VALUES'
  + ' (?,?,?,?);';

  const pool = await dbHelpers.getPool();
  try {
    await pool.query(queryString, [enhancedRecord.smartBulbName,
      eventCode,
      eventType,
      enhancedRecord.timestamp
    ]);
  } catch (err) {
    throw new Error(`Error inserting alert event into database: ${err}`);
  }
  return true;
}

async function sendAlert(awsRegion, enhancedRecord, topicArn, message, eventCode, alertType) {
  console.log(`Sending ${alertType} alert`);
  try {
    await insertAlertEventIntoRDS(enhancedRecord, eventCode, alertType);
    await awsHelpers.sendDataToSNS(awsRegion, message, alertType, topicArn);
  } catch (err) {
    throw new Error(`Error in sendAlert(): ${err}`);
  }
}

async function isDefective(awsRegion, topicArn, enhancedRecord) {
  const status = parseInt(enhancedRecord.status, 10);
  const lumens = parseInt(enhancedRecord.lumens, 10);
  if (Number.isNaN(lumens) || Number.isNaN(status)) { return false; }
  if (status === 1 && lumens === 0) {
    const message = (`${enhancedRecord.smartBulbName} bulb is defective. Timestamp:
${enhancedRecord.timestamp}`);
    await sendAlert(awsRegion, enhancedRecord, topicArn, message, 1, 'DEFECTIVE');
    return true;
  }
  return false;
}

const getDBArray = async function getArray(smartBulbName, timestamp) {
  const pool = await dbHelpers.getPool();
  const queryString = 'SELECT COUNT(event_code)'
        + ' AS count'
        + ' FROM smart_bulb_event'
        + ' WHERE smart_bulb_name = ?'
        + ' AND event_code = 1'
        + ' AND start >= DATE_ADD(?, INTERVAL - 30 DAY)'
```

```
        + ' AND start <= ?';
  const results = await pool.query(queryString, [smartBulbName, timestamp, timestamp]);
  return results;
};

async function updateOldDynamoDB(payload, table, smartBulbName, timestamp, awsRegion) {
  const params = {
    TableName: table,
    Key: { smartBulbName, timestamp },
    UpdateExpression: 'set #a = :x',
    ExpressionAttributeNames: { '#a': 'MLPayload' },
    ExpressionAttributeValues: {
      ':x': payload
    }
  };
  try {
    await awsHelpers.updateItemInDynamoDB(awsRegion, params);
  } catch (err) {
    console.error('Caught error updating dynamo', err);
  }
}

async function getOldDynamoDB(table, timestamp, smartBulbName, awsRegion) {
  let payload;
  const params = {
    TableName: table,
    Key: { smartBulbName, timestamp }
  };
  try {
    payload = await awsHelpers.getItemInDynamoDB(awsRegion, params);
  } catch (err) {
    console.error('Caught error getting dynamo', err);
  }
  return payload;
}

const checkFail = async function checkFail(enhancedRecord, awsRegion, topicArn) {
  const check = await isDefective(awsRegion, topicArn, enhancedRecord);
  if (check) {
    return 1;
  }
  const queryResult = await getDBArray(enhancedRecord.smartBulbName, enhancedRecord.timestamp);
  if (queryResult.length > 0) {
    console.log(`Found event code history for ${enhancedRecord.smartBulbName}`);
  } else throw new Error('Error with query');
  if (queryResult[0].count === 0) {
    return 0;
  } return 1;
};

const addDoesFail = async function addDoesFail(enhancedRecord) {
  const awsRegion = process.env.AWS_REGION;
  const dynamoDBML = process.env.DynamoDBML;
  const topicArn = process.env.TopicArn;
  const priorDate = subDays(enhancedRecord.timestamp, 30).toISOString();
  try {
```

```
    const doesFail = await checkFail(enhancedRecord, awsRegion, topicArn);
    const item = await getOldDynamoDB(dynamoDBML,
      priorDate,
      enhancedRecord.smartBulbName,
      awsRegion);

    if (item.Item && item.Item.payload) {
      const itemPayload = JSON.parse(item.Item.payload);
      const hoursOn = itemPayload.hoursOn;
      const switchCountWeek = itemPayload.switchCountWeek;
      const MLPayload = JSON.stringify({ hoursOn, switchCountWeek, doesFail });
      await updateOldDynamoDB(MLPayload,
        dynamoDBML,
        enhancedRecord.smartBulbName,
        priorDate,
        awsRegion);

      console.log('Updated dynamoDB record from 30 days ago');
    } else console.log('No value found in dynamo 30 days ago');
  } catch (err) {
    console.error('Error updating dynamoDB record from 30 days ago', err);
  }
};

module.exports = {
  addDoesFail, checkFail, isDefective
};

"FILENAME: C:\dev\c3\rawRepos\imet\functions\source\transform-lambda\features\hours-on-feature.js"

const awsHelpers = require('aws-helpers');
const {
  isBefore, parse, subDays, subMinutes, differenceInMinutes
} = require('date-fns');

async function getDynamoDBResults(awsRegion, smartBulbName, timestamp, minDate) {
  if (isBefore(timestamp, minDate)) {
    return [];
  }

  const daysBack = 2;
  const lowerBoundDate = subDays(timestamp, daysBack).toISOString();
  const params = {
    TableName: 'lightbulb-app-ml-ddb',
    KeyConditionExpression: 'smartBulbName = :hkey AND #timestamp BETWEEN :r1 AND :r2',
    ExpressionAttributeNames: { '#timestamp': 'timestamp' },
    ExpressionAttributeValues: {
      ':hkey': smartBulbName,
      ':r1': lowerBoundDate,
      ':r2': subMinutes(parse(timestamp), 1).toISOString()
    }
  };

  const results = await awsHelpers.queryDynamoDB(awsRegion, params);

  if (results.length > 0) {
```

```
    return results;
  }

  // recursively go back in time until results are found or min date is passed
  const recursiveResult = await getDynamoDBResults(awsRegion, smartBulbName, lowerBoundDate,
minDate);
  return recursiveResult;
}

const getLastMeasurementRecord = function (results) {
  let latestRecord = results[0];

  for (const result of results) {
    if (isBefore(latestRecord.timestamp, result.timestamp)) {
      latestRecord = result;
    }
  }

  console.log('getLastMeasurementRecord|latestRecord: ', latestRecord);

  return JSON.parse(latestRecord.payload);
};

const calculateHoursOn = function (hoursOn, enhancedRecord, lastRecord) {
  console.log('calculateHoursOn|enhancedRecord.timestamp: ', enhancedRecord.timestamp);
  console.log('calculateHoursOn|lastRecord.timestamp: ', lastRecord.timestamp);

  let currentStatus = enhancedRecord.status;
  if (typeof (currentStatus) === 'number') {
    currentStatus = currentStatus.toString();
  }

  if (currentStatus === '1') {
    const interval = differenceInMinutes(enhancedRecord.timestamp, lastRecord.timestamp) / 60;
    console.log('calculateHoursOn|interval: ', interval);

    return (interval > 0 ? hoursOn + interval : hoursOn);
  }
  return hoursOn;
};

const getHoursOn = async function (enhancedRecord) {
  let hoursOn = 0;
  let lastRecord;
  const awsRegion = process.env.AWS_REGION;
  const minDate = process.env.StartDate;

  let currentStatus = enhancedRecord.status;
  if (typeof (currentStatus) === 'number') {
    currentStatus = currentStatus.toString();
  }

  const results = await getDynamoDBResults(
    awsRegion,
    enhancedRecord.smartBulbName,
    enhancedRecord.timestamp,
```

```
    minDate
  );

  if (results.length > 0 && Object.keys(results[0]).length > 0) {
    lastRecord = getLastMeasurementRecord(results);
    if (lastRecord === null) {
      return (currentStatus === '1' ? 1 : 0);
    }
    hoursOn = lastRecord.hoursOn;
  } else {
    return (currentStatus === '1' ? 1 : 0);
  }

  hoursOn = calculateHoursOn(hoursOn, enhancedRecord, lastRecord);
  return hoursOn;
};

module.exports = {
  getHoursOn,
  calculateHoursOn,
  getLastMeasurementRecord
};

"FILENAME: C:\dev\c3\rawRepos\imet\functions\source\transform-lambda\features\week-switch-count-
feature.js"

const dbHelpers = require('database-helpers');

const getDBArray = async function getArray(smartBulbName, timestamp) {
  const pool = await dbHelpers.getPool();
  const queryString = 'SELECT timestamp,'
        + 'status'
        + ' FROM smart_bulb_measurement'
        + ' WHERE smart_bulb_name = ?'
        + ' AND timestamp >='
        + ' DATE_ADD(?, INTERVAL - 7 DAY)'
        + ' AND timestamp <= ?'
        + ' ORDER BY CONVERT(timestamp,DATETIME) ASC';
  const results = await pool.query(queryString, [smartBulbName, timestamp, timestamp]);
  return results;
};

const getWeekSwitchCount = async function getWeekSwitchCount(enhancedRecord) {
  const queryResult = await getDBArray(enhancedRecord.smartBulbName, enhancedRecord.timestamp);
  if (queryResult.length > 0) {
    console.log(`Found previous week status history for bulb ${enhancedRecord.smartBulbName}`);
  } else throw new Error('No results found for given lightbulb name');
  let switchCount = 0;
  for (let i = 1; i < queryResult.length; i += 1) {
    if (queryResult[i - 1].status !== queryResult[i].status) {
      switchCount += 1;
    }
  }
  return switchCount;
};
```

```
module.exports = getWeekSwitchCount;

"FILENAME: C:\dev\c3\rawRepos\imet\functions\source\transform-lambda\lib\deep-ar-model-builder.js"

const isEqual = require('date-fns/is_equal');
const parse = require('date-fns/parse');
const subHours = require('date-fns/sub_hours');

// Ignoring tests as these are coveraged in the machine-learning repo

/* istanbul ignore next */
const getFullTimestampRange = function (endTime, trainingHoursBack) {
  let i = 0;
  const timestamps = [];
  while (i < trainingHoursBack) {
    timestamps.push(subHours(endTime, i));
    i += 1;
  }
  return timestamps;
};

const HOURS_ON_MODEL_INDEX = 0;
const SWITCH_COUNT_MODEL_INDEX = 1;
/* istanbul ignore next */
function addNewDataToModel(model, measurement) {
  const parsedMLData = JSON.parse(measurement.MLPayload);
  model.target.unshift(parsedMLData.doesFail);
  model.dynamic_feat[HOURS_ON_MODEL_INDEX].unshift(parsedMLData.hoursOn);
  model.dynamic_feat[SWITCH_COUNT_MODEL_INDEX].unshift(parsedMLData.switchCountWeek);
}
/* istanbul ignore next */
function pullForwardLastData(model) {
  model.target.unshift(model.target[0]);

model.dynamic_feat[HOURS_ON_MODEL_INDEX].unshift(model.dynamic_feat[HOURS_ON_MODEL
_INDEX][0]);

model.dynamic_feat[SWITCH_COUNT_MODEL_INDEX].unshift(model.dynamic_feat[SWITCH_COU
NT_MODEL_INDEX][0]);
}
/* istanbul ignore next */
const buildModel = function (baseModel, sortedBulbMeasurements, timestamps) {
  const model = JSON.parse(JSON.stringify(baseModel));
  let bulbCursor = 0;
  timestamps.forEach(timestamp => {
    if (sortedBulbMeasurements[bulbCursor] && isEqual(sortedBulbMeasurements[bulbCursor].timestamp,
timestamp)) {
      if (sortedBulbMeasurements[bulbCursor].MLPayload) {
        addNewDataToModel(model, sortedBulbMeasurements[bulbCursor]);
      } else {
        pullForwardLastData(model);
      }
      bulbCursor += 1;
    } else if (sortedBulbMeasurements[bulbCursor] != undefined) {
      console.log(`${JSON.stringify(sortedBulbMeasurements[bulbCursor])} did not match current
${timestamp}.  Pulling forward previous data.`);
```

```
      pullForwardLastData(model);
    } else {
      pullForwardLastData(model);
    }
  });
  return model;
};
/* istanbul ignore next */
const buildDeepArModel = function (transformedBulbMeasurements, windowParameters) {
  const endTime = parse(transformedBulbMeasurements[0].timestamp);
  const timestampsForModel = getFullTimestampRange(endTime, windowParameters.trainingHoursBack);
  const start = parse(timestampsForModel[timestampsForModel.length - 1]).toISOString(); // reverse sorted
by TS from DDB
  const baseModel = {
    start: start.replace('Z', ''), // sagemaker requires no trailing Z for timezone so removing it
    target: [],
    dynamic_feat: [[], []]
  };
  const model = buildModel(baseModel, transformedBulbMeasurements, timestampsForModel);
  return model;
};

module.exports = {
  buildDeepArModel,
  getFullTimestampRange,
  buildModel,
  HOURS_ON_MODEL_INDEX,
  SWITCH_COUNT_MODEL_INDEX
};

"FILENAME: C:\dev\c3\rawRepos\imet\functions\source\transform-lambda\lib\deep-ar-request-
appender.js"

const subHours = require('date-fns/sub_hours');
const isEqual = require('date-fns/is_equal');
const { HOURS_ON_MODEL_INDEX, SWITCH_COUNT_MODEL_INDEX } = require('./deep-ar-
model-builder');

const getFullTimestampRangeAscending = function (endTime, trainingHoursBack) {
  let i = 1;
  const timestamps = [];
  while (i < trainingHoursBack+1) {
    timestamps.push(subHours(endTime, i));
    i+=1;
  }
  // We want this in ascending order so reversing.
  return timestamps.reverse();
};

function addNewDataToModel(model, measurement) {
  const parsedMLData = JSON.parse(measurement.payload);
  model.dynamic_feat[HOURS_ON_MODEL_INDEX].push(parsedMLData.hoursOn);
  model.dynamic_feat[SWITCH_COUNT_MODEL_INDEX].push(parsedMLData.switchCountWeek);
}

function pullForwardLastData(model) {
```

```
model.dynamic_feat[HOURS_ON_MODEL_INDEX].push(model.dynamic_feat[HOURS_ON_MODEL_I
NDEX][model.dynamic_feat[HOURS_ON_MODEL_INDEX].length - 1]);

model.dynamic_feat[SWITCH_COUNT_MODEL_INDEX].push(model.dynamic_feat[SWITCH_COUNT
_MODEL_INDEX][model.dynamic_feat[SWITCH_COUNT_MODEL_INDEX].length - 1]);
}

const buildModel = function (model, sortedBulbMeasurements, timestamps) {
  let bulbCursor = 0;
  timestamps.forEach(timestamp => {
    if (sortedBulbMeasurements[bulbCursor] && isEqual(sortedBulbMeasurements[bulbCursor].timestamp,
timestamp)) {
      if (sortedBulbMeasurements[bulbCursor].payload) {
        addNewDataToModel(model, sortedBulbMeasurements[bulbCursor]);
      } else {
        pullForwardLastData(model);
      }
      bulbCursor += 1;
    } else if (sortedBulbMeasurements[bulbCursor] != undefined) {
      console.log(`${JSON.stringify(sortedBulbMeasurements[bulbCursor])} did not match current
${timestamp}.  Pulling forward previous data.`);
      pullForwardLastData(model);
    } else {
      pullForwardLastData(model);
    }
  });
  return model;
};
// TODO: Rename trainingHoursBack in windowParams to something more generic / applicable
const appendPredictionDataToModel = function (predictionBulbMeasurements, windowParameters,
model) {
  // Array of timestamps from endTime and every hour back for traininghours
  const timestampsForModel = getFullTimestampRangeAscending(windowParameters.endOfWindow,
windowParameters.trainingHoursBack);
  return buildModel(model, predictionBulbMeasurements, timestampsForModel);
};

module.exports = {
  appendPredictionDataToModel
};

"FILENAME: C:\dev\c3\rawRepos\imet\functions\source\transform-lambda\lib\transformed-data-
retriever.js"

const awsHelpers = require('aws-helpers');
const {
  startOfHour, parse, subDays, subHours
} = require('date-fns');

// Ignoring as these are covered in machine-learning and are external calls

/* istanbul ignore next */
const getTimeWindow = function (baseTimestamp, predictionDaysBack, trainingHoursNeeded) {
  const initialDate = startOfHour(parse(baseTimestamp));
  const endOfWindow = subDays(initialDate, predictionDaysBack).toISOString();
```

```
  const beginningOfWindow = subHours(endOfWindow, trainingHoursNeeded).toISOString();
  return { beginningOfWindow, endOfWindow, trainingHoursBack: trainingHoursNeeded };
};

const PREDICTION_DAYS_BACK = 30;
const TRAINING_HOURS_BACK = 720;
const RECORD_LIMIT = TRAINING_HOURS_BACK;
/* istanbul ignore next */
const getTransformedData = async function (bulbName, baseTimestamp) {
  const awsRegion = process.env.AWS_REGION;
  const MLDataTable = process.env.DynamoDBML;
  const windowParameters = getTimeWindow(baseTimestamp, PREDICTION_DAYS_BACK,
TRAINING_HOURS_BACK);
  const params = {
    TableName: MLDataTable,
    KeyConditionExpression: '#a = :hkey and #b < :endkey',
    ExpressionAttributeNames: {
      '#a': 'smartBulbName',
      '#b': 'timestamp'
    },
    ExpressionAttributeValues: {
      ':hkey': bulbName,
      ':endkey': windowParameters.endOfWindow
    },
    FilterExpression: 'attribute_exists (MLPayload)',
    ProjectionExpression: '#b,MLPayload',
    ScanIndexForward: false
  };
  const bulbData = await awsHelpers.queryDynamoDB(awsRegion, params, RECORD_LIMIT);
  if (!bulbData || bulbData.length === 0) {
    throw new Error(`No transformed bulb data found for given lightbulb ${bulbName} for dates before
${windowParameters.endOfWindow}`);
  }
  console.log(`Found ${bulbData.length} records for dates ending at
${windowParameters.endOfWindow}`);
  return { bulbData, windowParameters };
};

/* istanbul ignore next */
const getPredictionTransformedData = async function (bulbName, baseTimestamp) {
  const awsRegion = process.env.AWS_REGION;
  const MLDataTable = process.env.DynamoDBML;
  const windowParameters = getTimeWindow(baseTimestamp, 0, TRAINING_HOURS_BACK);
  const params = {
    TableName: MLDataTable,
    KeyConditionExpression: '#a = :hkey and #b BETWEEN :begKey AND :endkey',
    ExpressionAttributeNames: {
      '#a': 'smartBulbName',
      '#b': 'timestamp'
    },
    ExpressionAttributeValues: {
      ':hkey': bulbName,
      ':begKey': windowParameters.beginningOfWindow,
      ':endkey': windowParameters.endOfWindow
    },
    FilterExpression: 'attribute_exists (payload)',
```

```
    ProjectionExpression: '#b,payload',
    ScanIndexForward: true
  };
  const bulbData = await awsHelpers.queryDynamoDB(awsRegion, params, RECORD_LIMIT);
  if (!bulbData || bulbData.length === 0) {
    throw new Error(`No transformed bulb data found for given lightbulb ${bulbName} for dates before
${windowParameters.endOfWindow}`);
  }
  console.log(`Found ${bulbData.length} records for dates ending at
${windowParameters.endOfWindow}`);
  return { bulbData, windowParameters };
};

module.exports = {
  getTransformedData,
  getTimeWindow,
  getPredictionTransformedData
};

"FILENAME: C:\dev\c3\rawRepos\imet\functions\source\transform-lambda\test\deep-ar-request-
appender.spec.js"

const chai = require('chai');
const chaiAsPromised = require('chai-as-promised');
const sinon = require('sinon');
const appender = require('../lib/deep-ar-request-appender.js');

chai.use(chaiAsPromised);
const { expect } = chai;

describe('Transform Lambda Test', () => {
  const sandbox = sinon.createSandbox();

  beforeEach(() => {

  });

  afterEach(() => {
    sandbox.restore();
  });

  describe('Deep Ar Request Appender', () => {
    const model = {
      start: '2018-02-01T15:00:00.000',
      target: [], // Not relevant at this stage
      dynamic_feat: [[1, 2], [14, 15]]
    };

    const testBulbData = [
      { payload: JSON.stringify({ hoursOn: 3, switchCountWeek: 16 }), timestamp: '2018-02-
01T17:00:00.000Z' },
      { payload: JSON.stringify({ hoursOn: 4, switchCountWeek: 17 }), timestamp: '2018-02-
01T19:00:00.000Z' },
      { payload: JSON.stringify({ hoursOn: 5, switchCountWeek: 18 }), timestamp: '2018-02-
01T20:00:00.000Z' },
```

```
    { payload: JSON.stringify({ hoursOn: 6, switchCountWeek: 18 }), timestamp: '2018-02-
01T21:00:00.000Z' },
    { payload: JSON.stringify({ hoursOn: 7, switchCountWeek: 19 }), timestamp: '2018-02-
01T22:00:00.000Z' },
    { payload: JSON.stringify({ hoursOn: 8, switchCountWeek: 19 }), timestamp: '2018-02-
01T23:00:00.000Z' }];

   const windowParams = { endOfWindow: '2018-02-01T23:00:00.000Z', trainingHoursBack: 7 };
   it('add new dynamic features on top of the existing model', async () => {
    const expectedHoursOn = [1, 2, 2, 3, 3, 4, 5, 6, 7];
    const expectedSwitchCount = [14, 15, 15, 16, 16, 17, 18, 18, 19];
    const requestModel = appender.appendPredictionDataToModel(testBulbData, windowParams, model);
    console.log(requestModel);
    expect(requestModel.dynamic_feat[0]).to.deep.eq(expectedHoursOn);
    expect(requestModel.dynamic_feat[1]).to.deep.eq(expectedSwitchCount);
   });
  });
});

"FILENAME: C:\dev\c3\rawRepos\imet\functions\source\transform-lambda\test\does-fail.spec.js"

const chai = require('chai');
const chaiAsPromised = require('chai-as-promised');
const sinon = require('sinon');
const dbHelpers = require('database-helpers');
const awsHelpers = require('aws-helpers');
const subDays = require('date-fns/sub_days');
const doesFailFeature = require('../features/does-fail-feature.js');

const addDoesFail = doesFailFeature.addDoesFail;
const checkFail = doesFailFeature.checkFail;

chai.use(chaiAsPromised);
const { assert, expect } = chai;
const sandbox = sinon.createSandbox();

beforeEach(() => {
});

afterEach(() => {
  sandbox.restore();
});

describe('Does Fail Feature Test', () => {
  describe('doesFailRecord', () => {
    it('Testing with one event within the 30 days', async () => {
      const inputRecord = {
        timestamp: '2012-02-01T21:00:00.000Z', smartBulbName: 'SMBLB1', status: 1
      };
      const DB = [
        { count: 1 }];
      console.log(`Database values: ${JSON.stringify(DB)}`);
      sandbox.stub(dbHelpers, 'getPool').callsFake(() => ({
        query() {
          return new Promise(resolve => resolve(DB));
        }
```

```
    }));
    sandbox.stub(awsHelpers, 'getItemInDynamoDB').callsFake(async () => (
      {
        Item: {

          payload: '{"timestamp":"2012-01-
02T21:00:00.000Z","smartBulbName":"SMBLB1","status":1,"power":14,"lumens":1000,"voltage":120,"te
mperature":76,"weatherData":{},"lightbulb":{"smartBulbName":"SMBLB1","manufacturer":"Bell","bulbT
ype":"LED","latitude":37.48596719,"longitude":-
122.2428196,"fixtureName":"fixt1","apartmentName":"apt1","buildingName":"bld1"},"switchCountWeek
":34,"hoursOn":2961}',
          smartBulbName: 'SMBLB1',
          timestamp: '2012-01-02T21:00:00.000Z'

        }
      }));
    sandbox.stub(awsHelpers, 'updateItemInDynamoDB');
    const failTest = await checkFail(inputRecord);
    assert.equal(failTest, 1);
  });
  it('Testing with no event within the 30 days', async () => {
    const inputRecord = {
      timestamp: '2011-02-01 13:00:00', smartBulbName: 'SMBLB1', status: 1
    };
    const DB = [{ count: 0 }];
    console.log(`Database values: ${JSON.stringify(DB)}`);
    sandbox.stub(dbHelpers, 'getPool').callsFake(() => ({
      query() {
        return new Promise(resolve => resolve(DB));
      }
    }));
    sandbox.stub(awsHelpers, 'getItemInDynamoDB').callsFake(async () => (
      {
        Item: {

          payload: '{"timestamp":"2012-01-
02T21:00:00.000Z","smartBulbName":"SMBLB1","status":1,"power":14,"lumens":1000,"voltage":120,"te
mperature":76,"weatherData":{},"lightbulb":{"smartBulbName":"SMBLB1","manufacturer":"Bell","bulbT
ype":"LED","latitude":37.48596719,"longitude":-
122.2428196,"fixtureName":"fixt1","apartmentName":"apt1","buildingName":"bld1"},"switchCountWeek
":34,"hoursOn":2961}',
          smartBulbName: 'SMBLB1',
          timestamp: '2012-01-02T21:00:00.000Z'

        }
      }));
    sandbox.stub(awsHelpers, 'updateItemInDynamoDB');
    const failTest = await checkFail(inputRecord);
    assert.equal(failTest, 0);
  });
  it('Testing with no dynamodb value 30 days prior', async () => {
    const inputRecord = {
      timestamp: '2012-02-01T21:00:00.000Z', smartBulbName: 'SMBLB1', status: 1
    };
    const DB = [{ count: 0 }];
    console.log(`Database values: ${JSON.stringify(DB)}`);
```

```
    sandbox.stub(dbHelpers, 'getPool').callsFake(() => ({
      query() {
        return new Promise(resolve => resolve(DB));
      }
    }));
    sandbox.stub(awsHelpers, 'getItemInDynamoDB').callsFake(async () => (
      {}));
    const spy = sinon.spy(awsHelpers, 'updateItemInDynamoDB');
    await addDoesFail(inputRecord);
    assert(spy.notCalled);
  });
  it('Testing with query error from RDS', async () => {
    const inputRecord = {
      timestamp: '2012-02-01T21:00:00.000Z', smartBulbName: 'SMBLB1', status: 1
    };
    const DB = [];
    sandbox.stub(dbHelpers, 'getPool').callsFake(() => ({
      query() {
        return new Promise(resolve => resolve(DB));
      }
    }));
    expect(checkFail(inputRecord)).to.be.rejectedWith(Error, 'Error with query');
  });
  it('Checking to see sub days output day we want', async () => {
    const inputRecord = {
      timestamp: '2013-02-03T15:00:00.000Z', smartBulbName: 'SMBLB1', status: 1
    };
    const priorDay = subDays(inputRecord.timestamp, 30).toISOString();
    expect(priorDay).to.eq('2013-01-04T15:00:00.000Z');
  });
 });
});

"FILENAME: C:\dev\c3\rawRepos\imet\functions\source\transform-lambda\test\hours-on-feature.spec.js"

const chai = require('chai');
const chaiAsPromised = require('chai-as-promised');
const sinon = require('sinon');
const awsHelpers = require('aws-helpers');
const isEqual = require('date-fns/is_equal');
const hoursOnFeature = require('../features/hours-on-feature.js');

chai.use(chaiAsPromised);
const { assert } = chai;
const sandbox = sinon.createSandbox();

beforeEach(() => {
});

afterEach(() => {
 sandbox.restore();
});

describe('Hours on Feature Test', () => {
 describe('hoursOnRecord', () => {
   it('Expect hours on to increment by 1 hour', async () => {
```

```
    const inputRecord = {
      timestamp: '2018-11-24T13:00:00.000Z', smartBulbName: 'SMBLB1', status: '1'
    };
    const dynamoDBMockData = [{
      payload: '{"timestamp":"2018-11-
24T12:00:00.000Z","smartBulbName":"SMBLB1","status":"0","power":"0","lumens":"0","voltage":"0","te
mperature":"68","weatherData":{},"lightbulb":{"smartBulbName":"SMBLB1","manufacturer":"Bell","bul
bType":"LED","latitude":37.48596719,"longitude":-
122.2428196,"fixtureName":"fixt1","apartmentName":"apt1","buildingName":"bld1"},"switchCountWeek
":0,"hoursOn":6364}',
      smartBulbName: 'SMBLB1',
      timestamp: '2018-11-24T12:00:00.000Z'
    },
    {
      payload: '{"timestamp":"2018-11-
24T11:00:00.000Z","smartBulbName":"SMBLB1","status":"0","power":"0","lumens":"0","voltage":"0","te
mperature":"67","weatherData":{},"lightbulb":{"smartBulbName":"SMBLB1","manufacturer":"Bell","bul
bType":"LED","latitude":37.48596719,"longitude":-
122.2428196,"fixtureName":"fixt1","apartmentName":"apt1","buildingName":"bld1"},"switchCountWeek
":0,"hoursOn":6364}',
      smartBulbName: 'SMBLB1',
      timestamp: '2018-11-24T11:00:00.000Z'
    },
    {
      payload: '{"timestamp":"2018-11-
24T10:00:00.000Z","smartBulbName":"SMBLB1","status":"0","power":"0","lumens":"0","voltage":"0","te
mperature":"67","weatherData":{},"lightbulb":{"smartBulbName":"SMBLB1","manufacturer":"Bell","bul
bType":"LED","latitude":37.48596719,"longitude":-
122.2428196,"fixtureName":"fixt1","apartmentName":"apt1","buildingName":"bld1"},"switchCountWeek
":0,"hoursOn":6364}',
      smartBulbName: 'SMBLB1',
      timestamp: '2018-11-24T10:00:00.000Z'
    }];

    sandbox.stub(awsHelpers, 'queryDynamoDB').callsFake(() => new Promise(resolve =>
resolve(dynamoDBMockData))
    );

    const hoursOn = await hoursOnFeature.getHoursOn(inputRecord);

    assert.equal(hoursOn, 6365);
  });
  it('Expect hours on to increment by 1 hour, status is a number type', async () => {
    const inputRecord = {
      timestamp: '2018-11-24T13:00:00.000Z', smartBulbName: 'SMBLB1', status: 1
    };
    const dynamoDBMockData = [{
      payload: '{"timestamp":"2018-11-
24T12:00:00.000Z","smartBulbName":"SMBLB1","status":"0","power":"0","lumens":"0","voltage":"0","te
mperature":"68","weatherData":{},"lightbulb":{"smartBulbName":"SMBLB1","manufacturer":"Bell","bul
bType":"LED","latitude":37.48596719,"longitude":-
122.2428196,"fixtureName":"fixt1","apartmentName":"apt1","buildingName":"bld1"},"switchCountWeek
":0,"hoursOn":6364}',
      smartBulbName: 'SMBLB1',
      timestamp: '2018-11-24T12:00:00.000Z'
    },
```

```
    {
      payload: '{"timestamp":"2018-11-
24T11:00:00.000Z","smartBulbName":"SMBLB1","status":"0","power":"0","lumens":"0","voltage":"0","te
mperature":"67","weatherData":{},"lightbulb":{"smartBulbName":"SMBLB1","manufacturer":"Bell","bul
bType":"LED","latitude":37.48596719,"longitude":-
122.2428196,"fixtureName":"fixt1","apartmentName":"apt1","buildingName":"bld1"},"switchCountWeek
":0,"hoursOn":6364}',
      smartBulbName: 'SMBLB1',
      timestamp: '2018-11-24T11:00:00.000Z'
    },
    {
      payload: '{"timestamp":"2018-11-
24T10:00:00.000Z","smartBulbName":"SMBLB1","status":"0","power":"0","lumens":"0","voltage":"0","te
mperature":"67","weatherData":{},"lightbulb":{"smartBulbName":"SMBLB1","manufacturer":"Bell","bul
bType":"LED","latitude":37.48596719,"longitude":-
122.2428196,"fixtureName":"fixt1","apartmentName":"apt1","buildingName":"bld1"},"switchCountWeek
":0,"hoursOn":6364}',
      smartBulbName: 'SMBLB1',
      timestamp: '2018-11-24T10:00:00.000Z'
    }];

    sandbox.stub(awsHelpers, 'queryDynamoDB').callsFake(() => new Promise(resolve =>
resolve(dynamoDBMockData))
    );

    const hoursOn = await hoursOnFeature.getHoursOn(inputRecord);

    assert.equal(hoursOn, 6365);
  });
  it('Expect hours on to increment by 15 min', async () => {
    const inputRecord = {
      timestamp: '2018-11-24T12:15:00.000Z', smartBulbName: 'SMBLB1', status: '1'
    };
    const dynamoDBMockData = [{
      payload: '{"timestamp":"2018-11-
24T12:00:00.000Z","smartBulbName":"SMBLB1","status":"0","power":"0","lumens":"0","voltage":"0","te
mperature":"68","weatherData":{},"lightbulb":{"smartBulbName":"SMBLB1","manufacturer":"Bell","bul
bType":"LED","latitude":37.48596719,"longitude":-
122.2428196,"fixtureName":"fixt1","apartmentName":"apt1","buildingName":"bld1"},"switchCountWeek
":0,"hoursOn":6364.0}',
      smartBulbName: 'SMBLB1',
      timestamp: '2018-11-24T12:00:00.000Z'
    },
    {
      payload: '{"timestamp":"2018-11-
24T11:00:00.000Z","smartBulbName":"SMBLB1","status":"0","power":"0","lumens":"0","voltage":"0","te
mperature":"67","weatherData":{},"lightbulb":{"smartBulbName":"SMBLB1","manufacturer":"Bell","bul
bType":"LED","latitude":37.48596719,"longitude":-
122.2428196,"fixtureName":"fixt1","apartmentName":"apt1","buildingName":"bld1"},"switchCountWeek
":0,"hoursOn":6364.0}',
      smartBulbName: 'SMBLB1',
      timestamp: '2018-11-24T10:15:00.000Z'
    },
    {
      payload: '{"timestamp":"2018-11-
24T10:00:00.000Z","smartBulbName":"SMBLB1","status":"0","power":"0","lumens":"0","voltage":"0","te
```

```
mperature":"67","weatherData":{},"lightbulb":{"smartBulbName":"SMBLB1","manufacturer":"Bell","bul
bType":"LED","latitude":37.48596719,"longitude":-
122.2428196,"fixtureName":"fixt1","apartmentName":"apt1","buildingName":"bld1"},"switchCountWeek
":0,"hoursOn":6364.0}',
        smartBulbName: 'SMBLB1',
        timestamp: '2018-11-24T10:00:00.000Z'
      }];

      sandbox.stub(awsHelpers, 'queryDynamoDB').callsFake(() => new Promise(resolve =>
resolve(dynamoDBMockData))
      );

      const hoursOn = await hoursOnFeature.getHoursOn(inputRecord);

      assert.equal(hoursOn, 6364.25);
    });
    it('Expect hours on to increment by 2.75 hour', async () => {
      const inputRecord = {
        timestamp: '2018-11-24T14:45:00.000Z', smartBulbName: 'SMBLB1', status: '1'
      };
      const dynamoDBMockData = [{
        payload: '{"timestamp":"2018-11-
24T12:00:00.000Z","smartBulbName":"SMBLB1","status":"0","power":"0","lumens":"0","voltage":"0","te
mperature":"68","weatherData":{},"lightbulb":{"smartBulbName":"SMBLB1","manufacturer":"Bell","bul
bType":"LED","latitude":37.48596719,"longitude":-
122.2428196,"fixtureName":"fixt1","apartmentName":"apt1","buildingName":"bld1"},"switchCountWeek
":0,"hoursOn":6364}',
        smartBulbName: 'SMBLB1',
        timestamp: '2018-11-24T12:00:00.000Z'
      },
      {
        payload: '{"timestamp":"2018-11-
24T11:00:00.000Z","smartBulbName":"SMBLB1","status":"0","power":"0","lumens":"0","voltage":"0","te
mperature":"67","weatherData":{},"lightbulb":{"smartBulbName":"SMBLB1","manufacturer":"Bell","bul
bType":"LED","latitude":37.48596719,"longitude":-
122.2428196,"fixtureName":"fixt1","apartmentName":"apt1","buildingName":"bld1"},"switchCountWeek
":0,"hoursOn":6364}',
        smartBulbName: 'SMBLB1',
        timestamp: '2018-11-24T11:00:00.000Z'
      },
      {
        payload: '{"timestamp":"2018-11-
24T10:00:00.000Z","smartBulbName":"SMBLB1","status":"0","power":"0","lumens":"0","voltage":"0","te
mperature":"67","weatherData":{},"lightbulb":{"smartBulbName":"SMBLB1","manufacturer":"Bell","bul
bType":"LED","latitude":37.48596719,"longitude":-
122.2428196,"fixtureName":"fixt1","apartmentName":"apt1","buildingName":"bld1"},"switchCountWeek
":0,"hoursOn":6364}',
        smartBulbName: 'SMBLB1',
        timestamp: '2018-11-24T10:00:00.000Z'
      }];

      sandbox.stub(awsHelpers, 'queryDynamoDB').callsFake(() => new Promise(resolve =>
resolve(dynamoDBMockData))
      );

      const hoursOn = await hoursOnFeature.getHoursOn(inputRecord);
```

```
    assert.equal(hoursOn, 6366.75);
  });
  it('Expect hours on to increment by 4 hours', async () => {
    const inputRecord = {
      timestamp: '2018-11-24T13:00:00.000Z', smartBulbName: 'SMBLB1', status: '1'
    };
    const dynamoDBMockData = [{
      payload: '{"timestamp":"2018-11-
24T09:00:00.000Z","smartBulbName":"SMBLB1","status":"0","power":"0","lumens":"0","voltage":"0","te
mperature":"68","weatherData":{},"lightbulb":{"smartBulbName":"SMBLB1","manufacturer":"Bell","bul
bType":"LED","latitude":37.48596719,"longitude":-
122.2428196,"fixtureName":"fixt1","apartmentName":"apt1","buildingName":"bld1"},"switchCountWeek
":0,"hoursOn":6364}',
      smartBulbName: 'SMBLB1',
      timestamp: '2018-11-24T09:00:00.000Z'
    },
    {
      payload: '{"timestamp":"2018-11-
24T08:00:00.000Z","smartBulbName":"SMBLB1","status":"0","power":"0","lumens":"0","voltage":"0","te
mperature":"67","weatherData":{},"lightbulb":{"smartBulbName":"SMBLB1","manufacturer":"Bell","bul
bType":"LED","latitude":37.48596719,"longitude":-
122.2428196,"fixtureName":"fixt1","apartmentName":"apt1","buildingName":"bld1"},"switchCountWeek
":0,"hoursOn":6364}',
      smartBulbName: 'SMBLB1',
      timestamp: '2018-11-24T08:00:00.000Z'
    },
    {
      payload: '{"timestamp":"2018-11-
24T07:00:00.000Z","smartBulbName":"SMBLB1","status":"0","power":"0","lumens":"0","voltage":"0","te
mperature":"67","weatherData":{},"lightbulb":{"smartBulbName":"SMBLB1","manufacturer":"Bell","bul
bType":"LED","latitude":37.48596719,"longitude":-
122.2428196,"fixtureName":"fixt1","apartmentName":"apt1","buildingName":"bld1"},"switchCountWeek
":0,"hoursOn":6364}',
      smartBulbName: 'SMBLB1',
      timestamp: '2018-11-24T07:00:00.000Z'
    }];

    sandbox.stub(awsHelpers, 'queryDynamoDB').callsFake(() => new Promise(resolve =>
resolve(dynamoDBMockData))
    );

    const hoursOn = await hoursOnFeature.getHoursOn(inputRecord);
    assert.equal(hoursOn, 6368);
  });
  it('Current record is older than latest record - expect hours on to increment by 0 hours', async () => {
    const inputRecord = {
      timestamp: '2018-11-24T03:00:00.000Z', smartBulbName: 'SMBLB1', status: '1'
    };
    const dynamoDBMockData = [{
      payload: '{"timestamp":"2018-11-
24T09:00:00.000Z","smartBulbName":"SMBLB1","status":"0","power":"0","lumens":"0","voltage":"0","te
mperature":"68","weatherData":{},"lightbulb":{"smartBulbName":"SMBLB1","manufacturer":"Bell","bul
bType":"LED","latitude":37.48596719,"longitude":-
122.2428196,"fixtureName":"fixt1","apartmentName":"apt1","buildingName":"bld1"},"switchCountWeek
":0,"hoursOn":6364}',
      smartBulbName: 'SMBLB1',
```

```
      timestamp: '2018-11-24T09:00:00.000Z'
     },
     {
      payload: '{"timestamp":"2018-11-
24T08:00:00.000Z","smartBulbName":"SMBLB1","status":"0","power":"0","lumens":"0","voltage":"0","te
mperature":"67","weatherData":{},"lightbulb":{"smartBulbName":"SMBLB1","manufacturer":"Bell","bul
bType":"LED","latitude":37.48596719,"longitude":-
122.2428196,"fixtureName":"fixt1","apartmentName":"apt1","buildingName":"bld1"},"switchCountWeek
":0,"hoursOn":6364}',
      smartBulbName: 'SMBLB1',
      timestamp: '2018-11-24T08:00:00.000Z'
     },
     {
      payload: '{"timestamp":"2018-11-
24T02:00:00.000Z","smartBulbName":"SMBLB1","status":"0","power":"0","lumens":"0","voltage":"0","te
mperature":"67","weatherData":{},"lightbulb":{"smartBulbName":"SMBLB1","manufacturer":"Bell","bul
bType":"LED","latitude":37.48596719,"longitude":-
122.2428196,"fixtureName":"fixt1","apartmentName":"apt1","buildingName":"bld1"},"switchCountWeek
":0,"hoursOn":6364}',
      smartBulbName: 'SMBLB1',
      timestamp: '2018-11-24T02:00:00.000Z'
     }];

     sandbox.stub(awsHelpers, 'queryDynamoDB').callsFake(() => new Promise(resolve =>
resolve(dynamoDBMockData))
     );

     const hoursOn = await hoursOnFeature.getHoursOn(inputRecord);

     assert.equal(hoursOn, 6364);
   });
   it('Status is off - expect hours on to stay the same', async () => {
     const inputRecord = {
      timestamp: '2018-11-24T13:00:00.000Z', smartBulbName: 'SMBLB1', status: '0'
     };
     const dynamoDBMockData = [{
      payload: '{"timestamp":"2018-11-
24T09:00:00.000Z","smartBulbName":"SMBLB1","status":"0","power":"0","lumens":"0","voltage":"0","te
mperature":"68","weatherData":{},"lightbulb":{"smartBulbName":"SMBLB1","manufacturer":"Bell","bul
bType":"LED","latitude":37.48596719,"longitude":-
122.2428196,"fixtureName":"fixt1","apartmentName":"apt1","buildingName":"bld1"},"switchCountWeek
":0,"hoursOn":6364}',
      smartBulbName: 'SMBLB1',
      timestamp: '2018-11-24T09:00:00.000Z'
     },
     {
      payload: '{"timestamp":"2018-11-
24T08:00:00.000Z","smartBulbName":"SMBLB1","status":"0","power":"0","lumens":"0","voltage":"0","te
mperature":"67","weatherData":{},"lightbulb":{"smartBulbName":"SMBLB1","manufacturer":"Bell","bul
bType":"LED","latitude":37.48596719,"longitude":-
122.2428196,"fixtureName":"fixt1","apartmentName":"apt1","buildingName":"bld1"},"switchCountWeek
":0,"hoursOn":6364}',
      smartBulbName: 'SMBLB1',
      timestamp: '2018-11-24T08:00:00.000Z'
     },
     {
```

```
      payload: '{"timestamp":"2018-11-
24T07:00:00.000Z","smartBulbName":"SMBLB1","status":"0","power":"0","lumens":"0","voltage":"0","te
mperature":"67","weatherData":{},"lightbulb":{"smartBulbName":"SMBLB1","manufacturer":"Bell","bul
bType":"LED","latitude":37.48596719,"longitude":-
122.2428196,"fixtureName":"fixt1","apartmentName":"apt1","buildingName":"bld1"},"switchCountWeek
":0,"hoursOn":6364}',
       smartBulbName: 'SMBLB1',
       timestamp: '2018-11-24T07:00:00.000Z'
      }];

     sandbox.stub(awsHelpers, 'queryDynamoDB').callsFake(() => new Promise(resolve =>
resolve(dynamoDBMockData))
     );

     const hoursOn = await hoursOnFeature.getHoursOn(inputRecord);

     assert.equal(hoursOn, 6364);
   });
   it('No existing records with current status on - expect hours on to be 1', async () => {
     const inputRecord = {
       timestamp: '2018-11-24T13:00:00.000Z', smartBulbName: 'SMBLB1', status: '1'
     };
     const dynamoDBMockData = [{}];

     sandbox.stub(awsHelpers, 'queryDynamoDB').callsFake(() => new Promise(resolve =>
resolve(dynamoDBMockData))
     );

     const hoursOn = await hoursOnFeature.getHoursOn(inputRecord);

     assert.equal(hoursOn, 1);
   });
   it('No existing records with current status off - expect hours on to be 0', async () => {
     const inputRecord = {
       timestamp: '2018-11-24T13:00:00.000Z', smartBulbName: 'SMBLB1', status: '0'
     };
     const dynamoDBMockData = [{}];

     sandbox.stub(awsHelpers, 'queryDynamoDB').callsFake(() => new Promise(resolve =>
resolve(dynamoDBMockData))
     );

     const hoursOn = await hoursOnFeature.getHoursOn(inputRecord);

     assert.equal(hoursOn, 0);
   });
   it('Calculate hours on value', () => {
     const inputRecord = {
       timestamp: '2018-11-24T13:00:00.000Z', smartBulbName: 'SMBLB1', status: '1'
     };

     const lastRecord = {
       timestamp: '2018-11-24T09:00:00.000Z',
       smartBulbName: 'SMBLB1',
       status: '0',
       power: '0',
```

```
      lumens: '0',
      voltage: '0',
      temperature: '68',
      weatherData: {},
      lightbulb:
      {
        smartBulbName: 'SMBLB1',
        manufacturer: 'Bell',
        bulbType: 'LED',
        latitude: 37.48596719,
        longitude: -122.2428196,
        fixtureName: 'fixt1',
        apartmentName: 'apt1',
        buildingName: 'bld1'
      },
      switchCountWeek: 0,
      hoursOn: 6364
    };

    const hoursOn = 6364;

    const calculatedHoursOn = hoursOnFeature.calculateHoursOn(
      hoursOn,
      inputRecord,
      lastRecord
    );

    assert.equal(calculatedHoursOn, 6368);
  });
  it('Get latest record', () => {
    const dynamoDBMockData = [{
      payload: '{"timestamp":"2018-11-
24T09:00:00.000Z","smartBulbName":"SMBLB1","status":"0","power":"0","lumens":"0","voltage":"0","te
mperature":"68","weatherData":{},"lightbulb":{"smartBulbName":"SMBLB1","manufacturer":"Bell","bul
bType":"LED","latitude":37.48596719,"longitude":-
122.2428196,"fixtureName":"fixt1","apartmentName":"apt1","buildingName":"bld1"},"switchCountWeek
":0,"hoursOn":6364}',
      smartBulbName: 'SMBLB1',
      timestamp: '2018-11-24T09:00:00.000Z'
    },
    {
      payload: '{"timestamp":"2018-11-
24T14:00:00.000Z","smartBulbName":"SMBLB1","status":"0","power":"0","lumens":"0","voltage":"0","te
mperature":"67","weatherData":{},"lightbulb":{"smartBulbName":"SMBLB1","manufacturer":"Bell","bul
bType":"LED","latitude":37.48596719,"longitude":-
122.2428196,"fixtureName":"fixt1","apartmentName":"apt1","buildingName":"bld1"},"switchCountWeek
":0,"hoursOn":6364}',
      smartBulbName: 'SMBLB1',
      timestamp: '2018-11-24T14:00:00.000Z'
    },
    {
      payload: '{"timestamp":"2018-11-
24T07:00:00.000Z","smartBulbName":"SMBLB1","status":"0","power":"0","lumens":"0","voltage":"0","te
mperature":"67","weatherData":{},"lightbulb":{"smartBulbName":"SMBLB1","manufacturer":"Bell","bul
bType":"LED","latitude":37.48596719,"longitude":-
```

```
122.2428196,"fixtureName":"fixt1","apartmentName":"apt1","buildingName":"bld1"},"switchCountWeek
":0,"hoursOn":6364}',
        smartBulbName: 'SMBLB1',
        timestamp: '2018-11-24T07:00:00.000Z'
      }];

      const lastRecord = hoursOnFeature.getLastMeasurementRecord(dynamoDBMockData);

      assert.equal(true, isEqual('2018-11-24T14:00:00.000Z', lastRecord.timestamp));
    });
  });
});
```

"FILENAME: C:\dev\c3\rawRepos\imet\functions\source\transform-lambda\test\transform.spec.js"

```
// const sinon = require('sinon');
// const chai = require('chai');
// const chaiAsPromised = require('chai-as-promised');
// const transformLambda = require('../index.js');

// chai.use(chaiAsPromised);
// const { expect } = chai;

// describe('Transform Lambda Test', () => {
//   const sandbox = sinon.createSandbox();

//   beforeEach(() => {

//   });

//   afterEach(() => {
//     sandbox.restore();
//   });

//   describe('Get Prediciton', function() {
//     this.timeout(15000);
//     it('get a sagemaker prediction from the sagemaker endpoint', async () => {
//         const enhancedRecord = { smartBulbName: 'SMBLB1', timestamp: '2018-11-16 12:00:00' };
//         const res = await transformLambda.getPrediction(enhancedRecord)
//         console.log("Prediction = "+res);
//         expect(res).to.not.be.undefined;
//     });
//   })
// });
```

"FILENAME: C:\dev\c3\rawRepos\imet\functions\source\transform-lambda\test\week-switch-count-feature.spec.js"

```
const chai = require('chai');
const chaiAsPromised = require('chai-as-promised');
const sinon = require('sinon');
const dbHelpers = require('database-helpers');
const switchCountFeature = require('../features/week-switch-count-feature.js');

chai.use(chaiAsPromised);
const { expect } = chai;
```

```
const { assert } = chai;
const sandbox = sinon.createSandbox();

beforeEach(() => {
});

afterEach(() => {
  sandbox.restore();
});

describe('Week Switch Count Feature Test', () => {
  describe('weekSwitchCountRecord', () => {
    it('Expecting error after inputing in a value to week switch count, and having an empty database', () => {
      const inputRecord = {
        timestamp: '2011-01-24 13:00:00', smartBulbName: 'SMBLB1', status: 1
      };
      sandbox.stub(dbHelpers, 'getPool').callsFake(() => ({
        query() {
          return new Promise(resolve => resolve([]));
        }
      }));
      expect(switchCountFeature(inputRecord)).to.be.rejectedWith(Error, 'No results found for given
lightbulb name');
    });
    it('Testing multiple zeroes in a row, should have total of 4 switches', async () => {
      const inputRecord = {
        timestamp: '2011-01-24 13:00:00', smartBulbName: 'SMBLB1', status: 1
      };
      const DB = [{ timestamp: '2011-01-13 10:00:00', status: 1 },
        { timestamp: '2011-01-13 11:00:00', status: 0 },
        { timestamp: '2011-01-13 12:00:00', status: 0 },
        { timestamp: '2011-01-13 13:00:00', status: 1 },
        { timestamp: '2011-01-13 14:00:00', status: 0 },
        { timestamp: '2011-01-13 15:00:00', status: 1 }];
      console.log(`Database values: ${JSON.stringify(DB)}`);
      sandbox.stub(dbHelpers, 'getPool').callsFake(() => ({
        query() {
          return new Promise(resolve => resolve(DB));
        }
      }));
      const switchCount = await switchCountFeature(inputRecord);
      assert.equal(switchCount, 4);
    });
    it('Testing multiple ones in a row, should have total of 2 switches', async () => {
      const inputRecord = {
        timestamp: '2011-01-24 13:00:00', smartBulbName: 'SMBLB1', status: 1
      };
      const DB = [{ timestamp: '2011-01-13 10:00:00', status: 0 },
        { timestamp: '2011-01-13 11:00:00', status: 1 },
        { timestamp: '2011-01-13 12:00:00', status: 1 },
        { timestamp: '2011-01-13 13:00:00', status: 1 },
        { timestamp: '2011-01-13 14:00:00', status: 0 }];
      console.log(`Database values: ${JSON.stringify(DB)}`);
      sandbox.stub(dbHelpers, 'getPool').callsFake(() => ({
        query() {
          return new Promise(resolve => resolve(DB));
```

```
    }
  }));
  const switchCount = await switchCountFeature(inputRecord);
  assert.equal(switchCount, 2);
});
it('Testing only one value in database, should have total of 0 switches', async () => {
  const inputRecord = {
    timestamp: '2011-01-24 13:00:00', smartBulbName: 'SMBLB1', status: 1
  };
  const DB = [{ timestamp: '2011-01-13 10:00:00', status: 0 }];
  console.log(`Database values: ${JSON.stringify(DB)}`);
  sandbox.stub(dbHelpers, 'getPool').callsFake(() => ({
    query() {
      return new Promise(resolve => resolve(DB));
    }
  }));
  const switchCount = await switchCountFeature(inputRecord);
  assert.equal(switchCount, 0);
});
it('Testing one at start, should have total of 3 switches', async () => {
  const inputRecord = {
    timestamp: '2011-01-24 13:00:00', smartBulbName: 'SMBLB1', status: 1
  };
  const DB = [{ timestamp: '2011-01-13 10:00:00', status: 1 },
    { timestamp: '2011-01-13 11:00:00', status: 0 },
    { timestamp: '2011-01-13 12:00:00', status: 1 },
    { timestamp: '2011-01-13 13:00:00', status: 0 },
    { timestamp: '2011-01-13 14:00:00', status: 0 },
    { timestamp: '2011-01-13 15:00:00', status: 0 }];
  console.log(`Database values: ${JSON.stringify(DB)}`);
  sandbox.stub(dbHelpers, 'getPool').callsFake(() => ({
    query() {
      return new Promise(resolve => resolve(DB));
    }
  }));
  const switchCount = await switchCountFeature(inputRecord);
  assert.equal(switchCount, 3);
});
it('Testing all ones, should have total of 0 switches', async () => {
  const inputRecord = {
    timestamp: '2011-01-24 13:00:00', smartBulbName: 'SMBLB1', status: 1
  };
  const DB = [{ timestamp: '2011-01-13 10:00:00', status: 1 },
    { timestamp: '2011-01-13 11:00:00', status: 1 },
    { timestamp: '2011-01-13 12:00:00', status: 1 },
    { timestamp: '2011-01-13 13:00:00', status: 1 },
    { timestamp: '2011-01-13 14:00:00', status: 1 },
    { timestamp: '2011-01-13 15:00:00', status: 1 }];
  console.log(`Database values: ${JSON.stringify(DB)}`);
  sandbox.stub(dbHelpers, 'getPool').callsFake(() => ({
    query() {
      return new Promise(resolve => resolve(DB));
    }
  }));
  const switchCount = await switchCountFeature(inputRecord);
  assert.equal(switchCount, 0);
```

```
  });
  it('Testing all zeroes, should have total of 0 switches', async () => {
   const inputRecord = {
    timestamp: '2011-01-24 13:00:00', smartBulbName: 'SMBLB1', status: 1
   };
   const DB = [{ timestamp: '2011-01-13 10:00:00', status: 0 },
    { timestamp: '2011-01-13 11:00:00', status: 0 },
    { timestamp: '2011-01-13 12:00:00', status: 0 },
    { timestamp: '2011-01-13 13:00:00', status: 0 },
    { timestamp: '2011-01-13 14:00:00', status: 0 },
    { timestamp: '2011-01-13 15:00:00', status: 0 }];
   console.log(`Database values: ${JSON.stringify(DB)}`);
   sandbox.stub(dbHelpers, 'getPool').callsFake(() => ({
    query() {
     return new Promise(resolve => resolve(DB));
    }
   }));
   const switchCount = await switchCountFeature(inputRecord);
   assert.equal(switchCount, 0);
  });
 });
});

"FILENAME: C:\dev\c3\rawRepos\imet\templates\data-lake-infra.template"

{
  "AWSTemplateFormatVersion": "2010-09-09",
  "Description": "Creates the data lake infrastructure needed by the rest of the application",
  "Parameters": {
    "DBUser" : {
      "Default" : "RDSmaster",
      "Description" : "The database admin account username",
      "Type" : "String",
      "MinLength" : "1",
      "MaxLength" : "41",
    },
    "DBPwd" : {
      "NoEcho" : "true",
      "Description" : "The database admin account password. Has to be longer than 8 characters",
      "Type" : "String",
      "MinLength" : "1",
      "MaxLength" : "41",
    },
    "SageMakerOutputModelPrefix": {
      "Type": "String",
      "Description": "S3 Prefix where SageMaker model lives",
      "Default": "models"
    }
  },

  "Resources": {
    "DataLakeLambdaS3Bucket": {
      "Type": "AWS::S3::Bucket",
      "DeletionPolicy" : "Retain",
      "Properties": {
        "BucketName": "lightbulb-lambda-code",
```

```
      "BucketEncryption": {
        "ServerSideEncryptionConfiguration": [
          {
            "ServerSideEncryptionByDefault": {
              "SSEAlgorithm": "AES256"
            }
          }
        ]
      },
      "AccessControl": "BucketOwnerFullControl"
    }
  },
  "DataLakeMLModelS3Bucket": {
    "Type": "AWS::S3::Bucket",
    "DeletionPolicy" : "Retain",
    "Properties": {
      "BucketName": "lightbulb-ml-model",
      "BucketEncryption": {
        "ServerSideEncryptionConfiguration": [
          {
            "ServerSideEncryptionByDefault": {
              "SSEAlgorithm": "AES256"
            }
          }
        ]
      },
      "AccessControl": "BucketOwnerFullControl",
      "NotificationConfiguration": {
        "LambdaConfigurations": [{
          "Event": "s3:ObjectCreated:*",
          "Function": {
            "Fn::ImportValue" : "SageMakerEndpointUpdateLambdaRef"
          },
          "Filter": {
            "S3Key": {
              "Rules": [
                {
                  "Name": "prefix",
                  "Value": {
                    "Ref": "SageMakerOutputModelPrefix"
                  }
                }
              ]
            }
          }
        }]
      }
    }
  },
  "DataLakeMLS3Bucket": {
    "Type": "AWS::S3::Bucket",
    "DeletionPolicy" : "Retain",
    "Properties": {
      "BucketName": "lightbulb-ml-data",
      "BucketEncryption": {
        "ServerSideEncryptionConfiguration": [
```

```
                {
                   "ServerSideEncryptionByDefault": {
                      "SSEAlgorithm": "AES256"
                   }
                }
             ]
          },
          "AccessControl": "BucketOwnerFullControl"
       }
    },
    "MLDynamoDBTable": {
       "Type" : "AWS::DynamoDB::Table",
       "Properties" : {
          "AttributeDefinitions" : [ {"AttributeName": "smartBulbName", "AttributeType" : "S"},
{"AttributeName": "timestamp", "AttributeType" : "S"}],
          "KeySchema" : [ {"AttributeName":"smartBulbName", "KeyType":"HASH"},
{"AttributeName":"timestamp", "KeyType":"RANGE"}],
          "ProvisionedThroughput" : { "ReadCapacityUnits": 125, "WriteCapacityUnits": 60},
          "SSESpecification" : {"SSEEnabled" : true},
          "TableName" : "lightbulb-app-ml-ddb",
          "BillingMode" : "PAY_PER_REQUEST"
       }
    },
    "RDS": {
       "Type" : "AWS::RDS::DBInstance",
       "Properties" : {
          "AllocatedStorage" : "10",
          "DBInstanceClass" : "db.m3.medium",
          "DBInstanceIdentifier" : "RDS-Instance",
          "DBName" : "lightbulbAppRds",
          "Engine" : "MySQL",
          "MasterUsername" : { "Ref" : "DBUser" },
          "MasterUserPassword" : { "Ref" : "DBPwd" }
       }
    }
  },

  "Outputs" : {
    "DataLakeLambdaS3BucketName":{
       "Description": "Data Lake Lambda S3 Bucket Full Name",
       "Value" : {"Ref" : "DataLakeLambdaS3Bucket"}
    },
    "RDSName":{
       "Description": "RDSName Full Name",
       "Value" : {"Ref" : "RDS"}
    }

  }
}
"FILENAME: C:\dev\c3\rawRepos\imet\templates\imet.template"

{
  "AWSTemplateFormatVersion": "2010-09-09",
  "Description": "Creates the data processing pipeline to continuously ingest, model, enhance, and
transform the raw data for training models. (qs-1onlac5tb)",
  "Parameters": {
```

```
    "LambdaCodeS3Bucket": {
      "Type": "String",
      "Description": "Prefix of the S3 Bucket where lambda code exists. The bucket name will be the
value of this parameter."
    },
    "LambdaCodeS3KeyPrefix": {
      "Type": "String",
      "Description": "S3 Key where lambda code exists",
      "Default": "imet-artifacts"
    },
    "IngestLambdaCodeLocation": {
      "Type": "String",
      "Description": "S3 Path where Ingest Lambda code exists",
      "Default": "ingest-lambda.zip"
    },
    "ModelLambdaCodeLocation": {
      "Type": "String",
      "Description": "S3 Path where Model Lambda code exists",
      "Default": "model-lambda.zip"
    },
    "EnhanceLambdaCodeLocation": {
      "Type": "String",
      "Description": "S3 Path where Enhance Lambda code exists",
      "Default": "enhance-lambda.zip"
    },
    "TransformLambdaCodeLocation": {
      "Type": "String",
      "Description": "S3 Path where Transform Lambda code exists",
      "Default": "transform-lambda.zip"
    },
    "AlertsLambdaCodeLocation": {
      "Type": "String",
      "Description": "S3 Path where Alerts Lambda code exists",
      "Default": "alerts-lambda.zip"
    },
    "SubscriptionEndPoint":{
      "Type":"String",
      "Description":"The endpoint that receives notifications from the Amazon SNS topic. The endpoint
value depends on the protocol that you specify. This could be a URL or ARN"
    },
    "SubscriptionProtocol":{
      "Type":"String",
      "Description":"The subscription's protocol",
      "AllowedValues":["http","https","email","email-json","sms","sqs","application","lambda"],
      "Default":"email"
    },
    "RDSUsername": {
      "Type":"String",
      "Description":"The RDS username to use for database",
      "Default":"RDSmaster"
    },
    "EncryptedRDSPwd" : {
      "Description" : "The KMS encrypted database admin account password.",
      "Type" : "String"
    },
    "RDSEndpoint": {
```

```
        "Type":"String",
        "Description":"The RDS username to use for database",
        "Default":"rds-instance.czktrh4irmdc.us-east-1.rds.amazonaws.com"
      },
      "RDSDatabaseName": {
        "Type":"String",
        "Description":"The RDS username to use for database",
        "Default":"lightbulbAppRds"
      },
      "AlertsTopicName": {
        "Type":"String",
        "Description":"The name of the alerts topic you want created",
        "Default":"lightBulbAlerts"
      },
      "DynamoDBMeasurementsTable": {
        "Type":"String",
        "Description":"The name of the DynamoDB measurements table",
        "Default":"lightbulb-app-measurements-ddb"
      },
      "DynamoDBMLTable": {
        "Type":"String",
        "Description":"The name of the DynamoDB ML table",
        "Default":"lightbulb-app-ml-ddb"
      },
      "StartDate": {
        "Type":"String",
        "Description":"The earliest date to get lightbulb data for",
        "Default":"2018-01-01T12:00:00.000Z"
      },
      "PredictionEndpoint": {
        "Type":"String",
        "Description":"The endpoint for the sagemaker prediction API",
        "Default":"risk-score-predictions"
      },
      "WeatherUri": {
        "Type":"String",
        "Description":"The uri for the weather API"
      },
      "WeatherAuth":{
        "Type":"String",
        "Description":"Auth key to access weather API",
        "Default":"Basic
YnJlbmRhbi5oaW5ja3NAcGFyaXZlZGFzb2x1dGlvbnMuY29tOlBQM0hLU3FHVXNAIUNCeUQ="
      }
    },
    "Resources": {
      "IngestLambdaRole": {
        "Type": "AWS::IAM::Role",
        "Properties": {
          "AssumeRolePolicyDocument": {
            "Version": "2012-10-17",
            "Statement": [{
              "Effect": "Allow",
              "Principal": {
                "Service": ["lambda.amazonaws.com"]
              },
```

```
            "Action": ["sts:AssumeRole"]
          }]
        },
        "Path": "/",
        "Policies": [{
          "PolicyName": {
            "Fn::Sub": "IngestLambdaPolicy"
          },
          "PolicyDocument": {
            "Version": "2012-10-17",
            "Statement": [{
                "Effect": "Allow",
                "Action": [
                  "logs:CreateLogGroup",
                  "logs:CreateLogStream",
                  "logs:PutLogEvents"
                ],
                "Resource": "*"
              },
              {
                "Effect": "Allow",
                "Action": [
                  "kinesis:ListStreams"
                ],
                "Resource": "*"
              },
              {
                "Effect": "Allow",
                "Action": [
                  "kinesis:Put*",
                  "kinesis:DescribeStream"
                ],
                "Resource": {
                  "Fn::GetAtt": ["IngestStream", "Arn"]
                }
              },
              {
                "Effect": "Allow",
                "Action": [
                  "sqs:DeleteMessage",
                  "sqs:ReceiveMessage",
                  "sqs:GetQueueAttributes"
                ],
                "Resource": {
                  "Fn::GetAtt": ["BatchRequestQueue", "Arn"]
                }
              }
            ]
          }
        }],
        "RoleName": {
          "Fn::Sub": "IngestLambdaRole"
        }
      }
    },
    "ModelLambdaRole": {
```

```
"Type": "AWS::IAM::Role",
"Properties": {
   "AssumeRolePolicyDocument": {
      "Version": "2012-10-17",
      "Statement": [{
         "Effect": "Allow",
         "Principal": {
            "Service": ["lambda.amazonaws.com"]
         },
         "Action": ["sts:AssumeRole"]
      }]
   },
   "Path": "/",
   "Policies": [{
      "PolicyName": {
         "Fn::Sub": "ModelLambdaPolicy"
      },
      "PolicyDocument": {
         "Version": "2012-10-17",
         "Statement": [{
               "Effect": "Allow",
               "Action": [
                  "logs:CreateLogGroup",
                  "logs:CreateLogStream",
                  "logs:PutLogEvents",
                  "ec2:CreateNetworkInterface",
                  "ec2:DescribeNetworkInterfaces",
                  "ec2:DeleteNetworkInterface",
                  "kms:Encrypt",
                  "kms:Decrypt",
                  "kms:ReEncrypt*",
                  "kms:GenerateDataKey*",
                  "kms:DescribeKey"
               ],
               "Resource": "*"
            },
            {
               "Effect": "Allow",
               "Action": [
                  "kinesis:ListStreams"
               ],
               "Resource": "*"
            },
            {
               "Effect": "Allow",
               "Action": [
                  "kinesis:GetRecords",
                  "kinesis:GetShardIterator",
                  "kinesis:DescribeStream"
               ],
               "Resource": {
                  "Fn::GetAtt": ["IngestStream", "Arn"]
               }
            },
            {
               "Effect": "Allow",
```

```
                "Action": [
                    "kinesis:Put*",
                    "kinesis:DescribeStream"
                ],
                "Resource": {
                    "Fn::GetAtt": ["ModelStream", "Arn"]
                }
            }
          ]
        }
      }],
      "RoleName": {
        "Fn::Sub": "ModelLambdaRole"
      }
    }
  },
  "EnhanceLambdaRole": {
    "Type": "AWS::IAM::Role",
    "Properties": {
      "AssumeRolePolicyDocument": {
        "Version": "2012-10-17",
        "Statement": [{
          "Effect": "Allow",
          "Principal": {
            "Service": ["lambda.amazonaws.com"]
          },
          "Action": ["sts:AssumeRole"]
        }]
      },
      "Path": "/",
      "Policies": [{
        "PolicyName": {
          "Fn::Sub": "EnhanceLambdaPolicy"
        },
        "PolicyDocument": {
          "Version": "2012-10-17",
          "Statement": [{
              "Effect": "Allow",
              "Action": [
                "logs:CreateLogGroup",
                "logs:CreateLogStream",
                "logs:PutLogEvents",
                "ec2:CreateNetworkInterface",
                "ec2:DescribeNetworkInterfaces",
                "ec2:DeleteNetworkInterface",
                "kms:Encrypt",
                "kms:Decrypt",
                "kms:ReEncrypt*",
                "kms:GenerateDataKey*",
                "kms:DescribeKey"
              ],
              "Resource": "*"
            },
            {
              "Effect": "Allow",
              "Action": [
```

```
                "kinesis:ListStreams"
              ],
              "Resource": "*"
            },
            {
              "Effect": "Allow",
              "Action": [
                "kinesis:GetRecords",
                "kinesis:GetShardIterator",
                "kinesis:DescribeStream"
              ],
              "Resource": {
                "Fn::GetAtt": ["ModelStream", "Arn"]
              }
            },
            {
              "Effect": "Allow",
              "Action": [
                "kinesis:Put*",
                "kinesis:DescribeStream"
              ],
              "Resource": {
                "Fn::GetAtt": ["EnhanceStream", "Arn"]
              }
            }
          ]
        }
      }],
      "RoleName": {
        "Fn::Sub": "EnhanceLambdaRole"
      }
    }
  },
  "TransformLambdaRole": {
    "Type": "AWS::IAM::Role",
    "Properties": {
      "AssumeRolePolicyDocument": {
        "Version": "2012-10-17",
        "Statement": [{
          "Effect": "Allow",
          "Principal": {
            "Service": ["lambda.amazonaws.com"]
          },
          "Action": ["sts:AssumeRole"]
        }]
      },
      "Path": "/",
      "Policies": [{
        "PolicyName": {
          "Fn::Sub": "TransformLambdaPolicy"
        },
        "PolicyDocument": {
          "Version": "2012-10-17",
          "Statement": [{
              "Effect": "Allow",
              "Action": [
```

```
            "logs:CreateLogGroup",
            "logs:CreateLogStream",
            "logs:PutLogEvents",
            "ec2:CreateNetworkInterface",
            "ec2:DescribeNetworkInterfaces",
            "ec2:DeleteNetworkInterface",
            "kms:Encrypt",
            "kms:Decrypt",
            "kms:ReEncrypt*",
            "kms:GenerateDataKey*",
            "kms:DescribeKey",
            "dynamodb:DeleteItem",
            "dynamodb:GetItem",
            "dynamodb:PutItem",
            "dynamodb:Scan",
            "dynamodb:UpdateItem",
            "dynamodb:Query"
        ],
        "Resource": "*"
    },
    {
        "Effect": "Allow",
        "Action": [
            "kinesis:ListStreams",
            "sagemaker:InvokeEndpoint"
        ],
        "Resource": "*"
    },
    {
        "Effect": "Allow",
        "Action": [
            "kinesis:GetRecords",
            "kinesis:GetShardIterator",
            "kinesis:DescribeStream"
        ],
        "Resource": {
            "Fn::GetAtt": ["EnhanceStream", "Arn"]
        }
    },
    {
        "Effect": "Allow",
        "Action": [
            "kinesis:Put*",
            "kinesis:DescribeStream"
        ],
        "Resource": {
            "Fn::GetAtt": ["TransformStream", "Arn"]
        }
    },
    {
        "Effect":"Allow",
        "Action":[
            "sns:Publish",
            "sns:CreateTopic"
        ],
        "Resource": {"Ref":"SNSTopic"}
```

```
                }
            ]
        }
      }],
      "RoleName": {
        "Fn::Sub": "TransformLambdaRole"
      }
    }
  },
  "AlertsLambdaRole": {
    "Type": "AWS::IAM::Role",
    "Properties": {
      "AssumeRolePolicyDocument": {
        "Version": "2012-10-17",
        "Statement": [{
          "Effect": "Allow",
          "Principal": {
            "Service": ["lambda.amazonaws.com"]
          },
          "Action": ["sts:AssumeRole"]
        }]
      },
      "Path": "/",
      "Policies": [{
        "PolicyName": {
          "Fn::Sub": "AlertsLambdaPolicy"
        },
        "PolicyDocument": {
          "Version": "2012-10-17",
          "Statement": [{
              "Effect": "Allow",
              "Action": [
                "logs:CreateLogGroup",
                "logs:CreateLogStream",
                "logs:PutLogEvents",
                "ec2:CreateNetworkInterface",
                "ec2:DescribeNetworkInterfaces",
                "ec2:DeleteNetworkInterface",
                "kms:Encrypt",
                "kms:Decrypt",
                "kms:ReEncrypt*",
                "kms:GenerateDataKey*",
                "kms:DescribeKey"
              ],
              "Resource": "*"
            },
            {
              "Effect": "Allow",
              "Action": [
                "kinesis:ListStreams"
              ],
              "Resource": "*"
            },
            {
              "Effect": "Allow",
              "Action": [
```

```
                        "kinesis:GetRecords",
                        "kinesis:GetShardIterator",
                        "kinesis:DescribeStream"
                    ],
                    "Resource": {
                        "Fn::GetAtt": ["TransformStream", "Arn"]
                    }
                },
                {
                    "Effect":"Allow",
                    "Action":[
                        "sns:Publish",
                        "sns:CreateTopic"
                    ],
                    "Resource": {"Ref":"SNSTopic"}
                }
            ]
          }
        }],
        "RoleName": {
          "Fn::Sub": "AlertsLambdaRole"
        }
    }
},
"BatchRequestQueue" : {
    "Type" : "AWS::SQS::Queue",
    "Properties" : {
        "QueueName" : "BatchRequestQueue",
        "ReceiveMessageWaitTimeSeconds": 20
    }
},
"IngestStream": {
    "Type": "AWS::Kinesis::Stream",
    "Properties": {
        "Name": {
          "Fn::Sub": "IngestStream"
        },
        "RetentionPeriodHours": 24,
        "ShardCount": 2
    }
},
"ModelStream": {
    "Type": "AWS::Kinesis::Stream",
    "Properties": {
        "Name": {
          "Fn::Sub": "ModelStream"
        },
        "RetentionPeriodHours": 24,
        "ShardCount": 4
    }
},
"EnhanceStream": {
    "Type": "AWS::Kinesis::Stream",
    "Properties": {
        "Name": {
          "Fn::Sub": "EnhanceStream"
```

```
        },
        "RetentionPeriodHours": 24,
        "ShardCount": 4
    }
},
"TransformStream": {
    "Type": "AWS::Kinesis::Stream",
    "Properties": {
        "Name": {
            "Fn::Sub": "TransformStream"
        },
        "RetentionPeriodHours": 24,
        "ShardCount": 2
    }
},
"IngestLambda": {
    "Type": "AWS::Lambda::Function",
    "Properties": {
        "FunctionName": {
            "Fn::Sub": "IngestLambda"
        },
        "Handler": "index.handler",
        "Role": {
            "Fn::GetAtt": ["IngestLambdaRole", "Arn"]
        },
        "Code": {
            "S3Bucket": {"Ref": "LambdaCodeS3Bucket"},
            "S3Key": {
                "Fn::Sub": "${LambdaCodeS3KeyPrefix}/${IngestLambdaCodeLocation}"
            }
        },
        "Runtime": "nodejs8.10",
        "Timeout": 30,
        "MemorySize": "1024",
        "Environment": {
            "Variables": {
                "OutputStream": {
                    "Fn::Sub": "IngestStream"
                }
            }
        }
    }
},
"ModelLambda": {
    "Type": "AWS::Lambda::Function",
    "Properties": {
        "FunctionName": {
            "Fn::Sub": "ModelLambda"
        },
        "Handler": "index.handler",
        "Role": {
            "Fn::GetAtt": ["ModelLambdaRole", "Arn"]
        },
        "Code": {
            "S3Bucket": {"Ref": "LambdaCodeS3Bucket"},
            "S3Key": {
```

```
                "Fn::Sub": "${LambdaCodeS3KeyPrefix}/${ModelLambdaCodeLocation}"
              }
            },
            "Runtime": "nodejs8.10",
            "Timeout": 90,
            "MemorySize": "2048",
            "VpcConfig": {
              "SecurityGroupIds" : [ "sg-e56b3daa" ],
              "SubnetIds" : [ "subnet-29db564e", "subnet-02e66e2c" ]
            },
            "KmsKeyArn": "arn:aws:kms:us-east-1:466081484468:key/57647f6b-a8bd-4e66-80b9-
a4eb59d525b3",
            "Environment": {
              "Variables": {
                "OutputStream": {
                  "Fn::Sub": "ModelStream"
                },
                "MySQLHost" : { "Ref" : "RDSEndpoint" },
                "MySQLUser" : { "Ref" : "RDSUsername" },
                "MySQLPass" : { "Ref" : "EncryptedRDSPwd" },
                "MySQLDB" : { "Ref" : "RDSDatabaseName" }
              }
            }
          }
        },
        "EnhanceLambda": {
          "Type": "AWS::Lambda::Function",
          "Properties": {
            "FunctionName": {
              "Fn::Sub": "EnhanceLambda"
            },
            "Handler": "index.handler",
            "Role": {
              "Fn::GetAtt": ["EnhanceLambdaRole", "Arn"]
            },
            "Code": {
              "S3Bucket": {"Ref": "LambdaCodeS3Bucket"},
              "S3Key": {
                "Fn::Sub": "${LambdaCodeS3KeyPrefix}/${EnhanceLambdaCodeLocation}"
              }
            },
            "Runtime": "nodejs8.10",
            "Timeout": 30,
            "MemorySize": "1024",
            "VpcConfig": {
              "SecurityGroupIds" : [ "sg-e56b3daa" ],
              "SubnetIds" : [ "subnet-12520f58", "subnet-dff17f83" ]
            },
            "KmsKeyArn": "arn:aws:kms:us-east-1:466081484468:key/57647f6b-a8bd-4e66-80b9-
a4eb59d525b3",
            "Environment": {
              "Variables": {
                "OutputStream": {
                  "Fn::Sub": "EnhanceStream"
                },
                "MySQLHost" : { "Ref" : "RDSEndpoint" },
```

```
                "MySQLUser" : { "Ref" : "RDSUsername" },
                "MySQLPass" : { "Ref" : "EncryptedRDSPwd" },
                "MySQLDB" : { "Ref" : "RDSDatabaseName" },
                "WeatherUri" : { "Ref" : "WeatherUri" },
                "WeatherAuth": { "Ref" : "WeatherAuth" }
              }
            }
          }
        },
        "TransformLambda": {
          "Type": "AWS::Lambda::Function",
          "Properties": {
            "FunctionName": {
              "Fn::Sub": "TransformLambda"
            },
            "Handler": "index.handler",
            "Role": {
              "Fn::GetAtt": ["TransformLambdaRole", "Arn"]
            },
            "Code": {
              "S3Bucket": {"Ref": "LambdaCodeS3Bucket"},
              "S3Key": {
                "Fn::Sub": "${LambdaCodeS3KeyPrefix}/${TransformLambdaCodeLocation}"
              }
            },
            "Runtime": "nodejs8.10",
            "Timeout": 30,
            "MemorySize": "1024",
            "VpcConfig": {
              "SecurityGroupIds" : [ "sg-e56b3daa" ],
              "SubnetIds" : [ "subnet-29db564e", "subnet-02e66e2c" ]
            },
            "KmsKeyArn": "arn:aws:kms:us-east-1:466081484468:key/57647f6b-a8bd-4e66-80b9-
a4eb59d525b3",
            "Environment": {
              "Variables": {
                "OutputPrefix": "transform",
                "OutputStream": {
                  "Fn::Sub": "TransformStream"
                },
                "TopicArn" : { "Ref" : "SNSTopic" },
                "MySQLHost" : { "Ref" : "RDSEndpoint" },
                "MySQLUser" : { "Ref" : "RDSUsername" },
                "MySQLPass" : { "Ref" : "EncryptedRDSPwd" },
                "MySQLDB" : { "Ref" : "RDSDatabaseName" },
                "DynamoDBMeasurements" : { "Ref" : "DynamoDBMeasurementsTable" },
                "DynamoDBML" : { "Ref": "DynamoDBMLTable" },
                "StartDate" : { "Ref": "StartDate" },
                "PredictionEndpoint" : { "Ref": "PredictionEndpoint"}
              }
            }
          }
        },
        "AlertsLambda": {
          "Type": "AWS::Lambda::Function",
          "Properties": {
```

```
            "FunctionName": {
               "Fn::Sub": "AlertsLambda"
            },
            "Handler": "index.handler",
            "Role": {
               "Fn::GetAtt": ["AlertsLambdaRole", "Arn"]
            },
            "Code": {
               "S3Bucket": {"Ref": "LambdaCodeS3Bucket"},
               "S3Key": {
                  "Fn::Sub": "${LambdaCodeS3KeyPrefix}/${AlertsLambdaCodeLocation}"
               }
            },
            "Runtime": "nodejs8.10",
            "Timeout": 90,
            "MemorySize": "2048",
            "VpcConfig": {
               "SecurityGroupIds" : [ "sg-e56b3daa" ],
               "SubnetIds" : [ "subnet-29db564e", "subnet-02e66e2c" ]
            },
            "KmsKeyArn": "arn:aws:kms:us-east-1:466081484468:key/57647f6b-a8bd-4e66-80b9-
a4eb59d525b3",
            "Environment": {
               "Variables": {
                  "TopicArn":{"Ref":"SNSTopic"},
                  "MySQLHost" : { "Ref" : "RDSEndpoint" },
                  "MySQLUser" : { "Ref" : "RDSUsername" },
                  "MySQLPass" : { "Ref" : "EncryptedRDSPwd" },
                  "MySQLDB" : { "Ref" : "RDSDatabaseName" }
               }
            }
         }
      },
      "IngestLambdaEventSource": {
         "Type": "AWS::Lambda::EventSourceMapping",
         "Properties": {
            "BatchSize": 10,
            "Enabled": true,
            "EventSourceArn": {
               "Fn::GetAtt": ["BatchRequestQueue", "Arn"]
            },
            "FunctionName": {
               "Fn::GetAtt": ["IngestLambda", "Arn"]
            }
         },
         "DependsOn": ["IngestLambdaRole"]
      },
      "ModelLambdaEventSource": {
         "Type": "AWS::Lambda::EventSourceMapping",
         "Properties": {
            "BatchSize": 1,
            "Enabled": true,
            "EventSourceArn": {
               "Fn::GetAtt": ["IngestStream", "Arn"]
            },
            "FunctionName": {
```

```
            "Fn::GetAtt": ["ModelLambda", "Arn"]
          },
          "StartingPosition": "LATEST"
      },
      "DependsOn": ["ModelLambdaRole"]
    },
    "EnhanceLambdaEventSource": {
      "Type": "AWS::Lambda::EventSourceMapping",
      "Properties": {
          "BatchSize": 1,
          "Enabled": true,
          "EventSourceArn": {
              "Fn::GetAtt": ["ModelStream", "Arn"]
          },
          "FunctionName": {
              "Fn::GetAtt": ["EnhanceLambda", "Arn"]
          },
          "StartingPosition": "LATEST"
      },
      "DependsOn": ["EnhanceLambdaRole"]
    },
    "TransformLambdaEventSource": {
      "Type": "AWS::Lambda::EventSourceMapping",
      "Properties": {
          "BatchSize": 2,
          "Enabled": true,
          "EventSourceArn": {
              "Fn::GetAtt": ["EnhanceStream", "Arn"]
          },
          "FunctionName": {
              "Fn::GetAtt": ["TransformLambda", "Arn"]
          },
          "StartingPosition": "LATEST"
      },
      "DependsOn": ["TransformLambdaRole"]
    },
    "AlertsLambdaEventSource": {
      "Type": "AWS::Lambda::EventSourceMapping",
      "Properties": {
          "BatchSize": 1,
          "Enabled": true,
          "EventSourceArn": {
              "Fn::GetAtt": ["TransformStream", "Arn"]
          },
          "FunctionName": {
              "Fn::GetAtt": ["AlertsLambda", "Arn"]
          },
          "StartingPosition": "LATEST"
      },
      "DependsOn": ["AlertsLambdaRole"]
    },
    "SNSTopic":{
      "Type" : "AWS::SNS::Topic",
      "Properties": {
        "Subscription" : [
          {
```

```
            "Endpoint" : {"Ref":"SubscriptionEndPoint"},
            "Protocol" : {"Ref":"SubscriptionProtocol"}
          }
        ],
        "TopicName":{"Ref":"AlertsTopicName"}
      }

    }

  }
}
"FILENAME: C:\dev\c3\rawRepos\imet\tools\data-load.py"

#!/usr/bin/python

import csv
import json
from datetime import datetime, timedelta
import sys
import boto3


def convertfully(csvfilename, jsonfilename, fullbulbs, bulbbuffer):
    start = datetime.now()
    jsonfile = open(jsonfilename, 'w')
    jsonfile.truncate(0)
    if fullbulbs == 0:
        return start

    fullbulbs += bulbbuffer
    with open(csvfilename, "rb") as csvfile:
        next(csvfile)
        datareader = csv.reader(csvfile)

        count = 0
        bulbcount = bulbbuffer + 1
        name = 'SMBLB' + str(bulbcount)
        jsonlist = []
        jsonobj = {}
        jsonobj[name] = jsonlist


        for row in datareader:
            if row[1] == name and bulbcount <= fullbulbs:
                newdate = str((start - timedelta(hours=(1613 * 24) - count)).replace(microsecond=0, second=0,
minute=0).isoformat()) + '.000Z' # Hardcoded 1500 for now

                jsoninsert = {}
                jsoninsert['TS'] = newdate
                jsoninsert['SN'] = row[1]
                jsoninsert['Status'] = row[2]
                jsoninsert['Watts'] = row[3]
                jsoninsert['Lumens'] = row[4]
                jsoninsert['Voltage'] = row[5]
                jsoninsert['Temp'] = row[6]
```

```
            jsonobj[name].append(jsoninsert)

            count += 1

        elif row[1] != name and int(filter(str.isdigit, row[1])) > bulbbuffer:
            print('Finished fully converting SMBLB'+str(bulbcount))
            sys.stdout.flush()
            bulbcount += 1
            if bulbcount > fullbulbs:
                json.dump(jsonobj, jsonfile)
                return start

            name = 'SMBLB' + str(bulbcount)
            jsonlist = []
            jsonobj[name] = jsonlist
            count = 0



def convert(csvfilename, jsonfilename, days, totalbulbs, fullbulbs, bulbbuffer, start):
    if fullbulbs >= totalbulbs:
        return

    if fullbulbs == 0:
        data = {}
    else:
        with open(jsonfilename) as f:
            data = json.load(f)


    totalbulbs += bulbbuffer
    fullbulbs += bulbbuffer

    with open(csvfilename, "rb") as csvfile:
        datareader = csv.reader(csvfile)

        count = 0
        bulbcount = 1 + fullbulbs
        name = 'SMBLB' + str(bulbcount)
        jsonlist = []
        jsonobj = {}
        jsonobj[name] = jsonlist


        for row in datareader:
            if row[1] == name and count < (days*24) and bulbcount <= totalbulbs:
                newdate = str((start - timedelta(hours=(days * 24) - count)).replace(microsecond=0, second=0,
minute=0).isoformat()) + '.000Z'

                jsoninsert = {}
                jsoninsert['TS'] = newdate
                jsoninsert['SN'] = row[1]
                jsoninsert['Status'] = row[2]
                jsoninsert['Watts'] = row[3]
                jsoninsert['Lumens'] = row[4]
                jsoninsert['Voltage'] = row[5]
```

```
            jsoninsert['Temp'] = row[6]

            jsonobj[name].append(jsoninsert)

            count += 1

        elif count >= (days*24):
            print('Finished converting SMBLB'+str(bulbcount))
            sys.stdout.flush()
            bulbcount += 1
            name = 'SMBLB' + str(bulbcount)
            jsonlist = []
            jsonobj[name] = jsonlist
            count = 0

    data.update(jsonobj)

    with open(jsonfilename, 'w') as f:
        json.dump(data, f)

def sendtosqs(jsonfilename, totalbulbs, bulbbuffer):
    sqs = boto3.resource('sqs', region_name='us-east-1')
    queue_name = 'BatchRequestQueue'
    queue = sqs.get_queue_by_name(QueueName=queue_name)
    totalbulbs += bulbbuffer

    with open(jsonfilename, 'r') as json_file:
        data = json.load(json_file)
        length = -1

        for bulb in range((bulbbuffer+1),(totalbulbs+1)):
            name = 'SMBLB' + str(bulb)
            if len(data[name])>length:
                length=len(data[name])

        for time in range(0,length):
            for bulb in range((bulbbuffer+1), (totalbulbs+1)):
                name = 'SMBLB' + str(bulb)
                if len(data[name])>time:
                    (queue.send_message(MessageBody=json.dumps(data[name][time])))



def main():
    # arg list: csvfilename, jsonfilename
    print('How many days of data (excluding full history bulbs) for each bulb:')
    sys.stdout.flush()
    days = input()
    print('Total amount of bulbs to be input:')
    sys.stdout.flush()
    totalbulbs = input()
    print('Number of bulbs with entire history:')
    sys.stdout.flush()
    fullbulbs = input()
    print('Number of bulbs to skip for input (bulbbuffer):')
    sys.stdout.flush()
```

```
    bulbbuffer = input()

    start = convertfully(sys.argv[1], sys.argv[2], fullbulbs, bulbbuffer)
    convert(sys.argv[1], sys.argv[2], days, totalbulbs, fullbulbs, bulbbuffer, start)
    sendtosqs(sys.argv[2], totalbulbs, bulbbuffer)


if __name__ == "__main__":
    main()
```

"FILENAME: C:\dev\c3\rawRepos\imet\tools\package-lambda-code.sh"

```
#!/bin/sh

lambda=$1

baseSource="functions/source"
source="$baseSource/$lambda"
filename="$lambda.zip"
package_folder="functions/packages"
destination="$package_folder/$lambda/$filename"

cd "$baseSource/aws-helpers"
rm -rf node_modules
npm install --only=prod
cd "../../.."

cd $source

s3Key="$lambda/$filename"

# Clearing out node_modules so we can build a smaller package
rm -rf node_modules
npm install --only=prod
zip -r -q $lambda .

cd "../../.."

mkdir -p $package_folder/$lambda/
mv "$source/$filename" $destination
```

"FILENAME: C:\dev\c3\rawRepos\imet\tools\package-upload-all-lambda-code.sh"

```
./tools/package-lambda-code.sh ingest-lambda
./tools/upload-lambda-code-s3.sh ingest-lambda
./tools/package-lambda-code.sh model-lambda
./tools/upload-lambda-code-s3.sh model-lambda
./tools/package-lambda-code.sh enhance-lambda
./tools/upload-lambda-code-s3.sh enhance-lambda
./tools/package-lambda-code.sh transform-lambda
./tools/upload-lambda-code-s3.sh transform-lambda
./tools/package-lambda-code.sh alerts-lambda
./tools/upload-lambda-code-s3.sh alerts-lambda

rm -rf functions/source/ingest-lambda/node_modules
rm -rf functions/source/model-lambda/node_modules
```

```
rm -rf functions/source/enhance-lambda/node_modules
rm -rf functions/source/transform-lambda/node_modules
rm -rf functions/source/alerts-lambda/node_modules
"FILENAME: C:\dev\c3\rawRepos\imet\tools\upload-lambda-code-s3.sh"

#!/bin/sh

#Enable the following line if running in git bash on Windows
alias aws="winpty /C/Program\ Files/Amazon/AWSCLI/bin/aws.cmd"

lambda=$1

filename="$lambda.zip"
package_folder="functions/packages"
packageLocation="$package_folder/$lambda/$filename"

aws s3api put-object --bucket lightbulb-lambda-code --key artifacts/$filename --body $packageLocation


"FILENAME: C:\dev\c3\rawRepos\machine-learning\functions\source\aws-helpers\index.js"

var AWS = require('aws-sdk');
var async = require('async');

let decrypted;
module.exports = {
	getDecryptedKmsKey: function(awsRegion, encryptedKey) {
		return new Promise((resolve, reject) => {
			if (decrypted) {
				resolve(decrypted);
			} else {
				const kms = new AWS.KMS({region: awsRegion});
				kms.decrypt({ CiphertextBlob: new Buffer(encryptedKey, 'base64') },
(err, data) => {
					if (err) {
						console.log('Decrypt error:', err);
						reject(err);
					}
					decrypted = data.Plaintext.toString('ascii');
					resolve(decrypted);
				});
			}
		});
	},

	sendDataToFirehose: function (awsRegion, firehoseStream, data, callback) {
		var firehose = new AWS.Firehose({region: awsRegion});
		var params = {
		   DeliveryStreamName: firehoseStream,
		   Record: {Data: data }

		};
		firehose.putRecord(params, function(err, data) {
		   if (err){
		      console.log("Failed to insert data onto Firehose: ", JSON.stringify(err));
		      callback(err);
```

```
                    }
                    else{
                                console.log("Successfully placed data in onto the following firehose
stream: " + firehoseStream);
                                callback(null);
                    }
                });
        },
        sendDataToKinesisAsync: function (awsRegion, outputStream, partitionKey, data) {
          var kinesis = new AWS.Kinesis({region: awsRegion});
          var params = {
            Data: data,
            PartitionKey:  partitionKey,
            StreamName: outputStream
                };
                console.log("Putting data to Kinesis: "+data);
          return new Promise((resolve, reject) => {
                        kinesis.putRecord(params, function(err, data) {
                                if (err) {
                                        console.log("Failed to insert data onto Kinesis: ",
JSON.stringify(err));
                                        reject(err);
                                }
                                console.log("Successfully placed data onto the following stream: " +
outputStream);
                                resolve("Kinesis put success.");
                        })
                });
        },
        sendDataToS3: function (awsRegion, bucketName, key, data) {
                const s3 = new AWS.S3({region: awsRegion});
                const params = {
                        Body: data,
                        Bucket: bucketName,
                        Key: key,
                        ServerSideEncryption: "AES256"
                 };
                return new Promise((resolve, reject) => {
                        s3.putObject(params, function(err, data) {
                                if(err){
                                  console.log("Failed to insert data into S3: ", JSON.stringify(err));
                                  reject(err);
                         }
                         else{
                                 console.log("Successfully placed data in S3 at s3://" + bucketName +
"/" + key);
                                 resolve(data);
                         }
                        });
                });
        },

        listAllS3Objects: function (awsRegion, bucketName, prefix, allS3Objects, token, callback)
        {
        var s3 = new AWS.S3({region: awsRegion});
```

```
                var opts = {
                        Bucket: bucketName,
                        Prefix: prefix
                };
                if(token) opts.ContinuationToken = token;

                s3.listObjectsV2(opts, function(err, data){
                        if (err) console.log(err, err.stack);
                        else {
                           allS3Objects = allS3Objects.concat(data.Contents);

                           if(data.IsTruncated)
                              listAllS3Objects(s3, bucketName, prefix, allS3Objects,
data.NextContinuationToken, callback);
                           else
                              callback(allS3Objects);
                        }
                });
        },

        readAllS3Objects: function (awsRegion, bucketName, allS3Objects, callback)
        {
        var s3 = new AWS.S3({region: awsRegion});

                var readS3Object = this.readS3Object;
           var readAsync = function(s3Object, callback){
              readS3Object(awsRegion, bucketName, s3Object.Key, callback);
           }

           async.concat(allS3Objects, readAsync, function(err, objects){
             if (err){
                console.log(err, err.stack);
                callback(err);
              }else{
                callback(null, objects);
              }
           });
        },

        readS3Object: function (awsRegion, bucketName, key, callback)
        {
                var s3 = new AWS.S3({region: awsRegion});

           s3.getObject({ Bucket: bucketName, Key: key }, function(err, data)
           {
              if (err){
                console.log(err, err.stack);
                callback(err);
              }else{
                callback(null, data.Body.toString());
              }
           });
        },

        sendDataToSNS: function (awsRegion, message, subject, topicArn){
```

```
		var sns = new AWS.SNS({region:awsRegion});

		// Create publish parameters
		var params = {
			Message: message, /* required */
			TopicArn: topicArn,
			Subject: subject
		};

		return new Promise((resolve, reject) => {
			sns.publish(params, (err, data) => {
				if (err) {
					console.log("Failed to publish to SNS: ", JSON.stringify(err));
					reject(err);
				}
				console.log("MessageID is " + data.MessageId);
				resolve(data);
			})
		});
	},

	putItemInDynamoDB: function(awsRegion, tableName, item) {
		return new Promise((resolve,reject) => {
			var documentClient = new AWS.DynamoDB.DocumentClient({region:
awsRegion});
			var params = {
				TableName: tableName,
				Item: item
			};
			documentClient.put(params, function(err, data) {
		   if (err){
		      console.log(err, err.stack);
		      reject(err);
		   } else {
		      resolve(data);
		   }
			});
		});
	},

	updateItemInDynamoDB: function(awsRegion, params) {
		return new Promise((resolve,reject) => {
			var documentClient = new AWS.DynamoDB.DocumentClient({region:
awsRegion});

			documentClient.update(params, function(err, data) {
				if (err) {
					console.log(err);
					reject(err);
				}
				else {
					resolve(data);
				}
			});
		});
	},
```

```
	getItemInDynamoDB: function(awsRegion, params) {
		return new Promise((resolve,reject) => {
			var documentClient = new AWS.DynamoDB.DocumentClient({region:
awsRegion});

			documentClient.get(params, function(err, data) {
				if (err) {
					console.log(err);
					reject(err);
				}
				else {
					resolve(data);
					return data;
				}
			});
		});
	},

	queryDynamoDB: function(awsRegion, params, maxRecords) {
		return new Promise((resolve,reject) => {
			let allData = [];
			let allSize = 0;
			var documentClient = new AWS.DynamoDB.DocumentClient({region:
awsRegion});
			documentClient.query(params, function paginateCallback(err, data) {
		    if (err){
		       console.log(err, err.stack);
		       reject(err);
		    } else {
					allData.push(data);
					allSize += data.Items ? data.Items.length : 0;
		       if(data.LastEvaluatedKey && !(allSize >= maxRecords)) {
						params.ExclusiveStartKey = data.LastEvaluatedKey;
						documentClient.query(params, paginateCallback);
					} else {
						const result = allData.reduce((accumulator,
currentValue) => accumulator.concat(currentValue.Items), []);
						resolve(result);
					};
				}
			});
		});
	}
}

"FILENAME: C:\dev\c3\rawRepos\machine-learning\functions\source\aws-helpers\package.json"

{
  "name": "aws-helpers",
  "version": "0.0.1",
  "private": true,
  "main": "index.js",
  "description": "A library of aws helper functions",
  "license": "MIT",
  "devDependencies": {
```

```
      "aws-sdk": "^2.331.0"
    },
    "dependencies": {
      "async": "~2.6.0"
    }
}

"FILENAME: C:\dev\c3\rawRepos\machine-learning\functions\source\data-prep-lambda\.eslintrc.js"

module.exports = {
  env: {
    es6: true,
    node: true,
    mocha: true,
    jest: true
  },
  extends: ['airbnb-base'],
  rules: {
    'arrow-parens': ['error', 'as-needed'],
    'comma-dangle': ['error', 'never'],
    'function-paren-newline': 'off',
    'no-underscore-dangle': 'off',
    'prefer-destructuring': 'off',
    'no-console': 'off',
    'no-await-in-loop': 'off',
    'import/no-extraneous-dependencies': [
      'warn',
      { devDependencies: false }
    ],
    'no-restricted-syntax': [
      'error',
      {
        selector: 'ForInStatement',
        message: 'for..in loops iterate over the entire prototype chain, which is virtually never what you want.
Use Object.{keys,values,entries}, and iterate over the resulting array.',
      },
      {
        selector: 'LabeledStatement',
        message: 'Labels are a form of GOTO; using them makes code confusing and hard to maintain and
understand.',
      },
      {
        selector: 'WithStatement',
        message: '`with` is disallowed in strict mode because it makes code impossible to predict and
optimize.',
      },
    ],
  },
  overrides: [
    {
      files: '**/*.spec.js',
      rules: {
        'no-unused-expressions': 'off',
        'prefer-arrow-callback': 'off',
        'import/no-extraneous-dependencies': [
          'error',
```

```
        { devDependencies: true }
       ]
      }
     }
    ]
};

"FILENAME: C:\dev\c3\rawRepos\machine-learning\functions\source\data-prep-lambda\index.js"

const awsHelpers = require('aws-helpers');
const { getTransformedData } = require('./lib/transformed-data-retriever');
const { buildDeepArModel } = require('./lib/deep-ar-model-builder');
const { parse } = require('date-fns');
const BATCH_SIZE = 25;

function buildBulbNameSet(batchNum) {
  let bulbNameSet = [];
  let bulbNum = BATCH_SIZE * batchNum;
  while (bulbNum < BATCH_SIZE * (batchNum+1)) {
    bulbNameSet.push(`SMBLB${bulbNum+1}`);
    bulbNum = bulbNum+1;
  }
  return bulbNameSet;
}

const RECORDS_TO_USE_IN_TEST_SET = 2;
async function putModelInBucket(modelArray, startTime, batchNum) {
  const awsRegion = process.env.AWS_REGION;
  const bucketName = process.env.OutputBucketName;
  const formattedOutputString = parse(startTime).toISOString();
  const testModelArray = modelArray.slice(0,RECORDS_TO_USE_IN_TEST_SET);
  const trainModelArray = modelArray.splice(RECORDS_TO_USE_IN_TEST_SET, modelArray.length);
  const testingJsonLines = testModelArray.reduce((acc, modeledItem) => acc +=
JSON.stringify(modeledItem)+"\n", "");
  const trainingJsonLines= trainModelArray.reduce((acc, modeledItem) => acc +=
JSON.stringify(modeledItem)+"\n", "");

  const testKey = `${formattedOutputString}/test/${batchNum}.jsonl`;
  const trainKey = `${formattedOutputString}/train/${batchNum}.jsonl`;
  await awsHelpers.sendDataToS3(awsRegion, bucketName, testKey, testingJsonLines);
  await awsHelpers.sendDataToS3(awsRegion, bucketName, trainKey, trainingJsonLines);
}

exports.handler = async (event) => {
  console.log('Received event: ', JSON.stringify(event, null, 2));
  const startTime = event.startTime ? parse(event.startTime) : parse(new Date());
  const batchNum = event.batchNum ? event.batchNum : 0;
  const bulbSet = buildBulbNameSet(batchNum);
  const fileData = [];
  for (let bulb of bulbSet) {
    try {
      const transformedData = await getTransformedData(bulb, startTime);
      const model = buildDeepArModel(transformedData.bulbData, transformedData.windowParameters);
      fileData.push(model);
    } catch (err) {
      console.error(`Error processing ${bulb}: ${err}`);
```

```
    }
  }
  await putModelInBucket(fileData, startTime, batchNum);

  //Lets the next step know where to start
  return { batchNum: batchNum+1, startTime: startTime };
};
```

"FILENAME: C:\dev\c3\rawRepos\machine-learning\functions\source\data-prep-lambda\package.json"

```
{
   "name": "enhance-lambda",
   "version": "0.0.1",
   "description": "A simple lambda that add appends data to price history records",
   "license": "MIT",
   "scripts": {
      "test": "nyc --check-coverage --lines 80 --functions 80 --branches 80 --reporter=cobertura mocha -R
mocha-multi-reporters --recursive test",
      "lint": "eslint . --fix",
      "prettier": "prettier --single-quote --write '**/*.js'"
   },
   "devDependencies": {
      "chai": "^4.2.0",
      "chai-as-promised": "7.1.1",
      "expect.js": "^0.3.1",
      "eslint": "^5.6.1",
      "eslint-config-airbnb-base": "^13.1.0",
      "eslint-plugin-import": "^2.14.0",
      "mocha": "^5.2.0",
      "mocha-multi-reporters": "^1.1.7",
      "nyc": "^13.0.1",
      "prettier": "^1.6.1",
      "sinon": "^6.3.5"
   },
   "dependencies": {
      "aws-helpers": "file:../aws-helpers",
      "date-fns": "^1.29.0"
   },
   "prettier": {
      "singleQuote": true
   },
   "private": true
}
```

"FILENAME: C:\dev\c3\rawRepos\machine-learning\functions\source\data-prep-lambda\lib\deep-ar-model-builder.js"

```
const isEqual = require('date-fns/is_equal');
const parse = require('date-fns/parse');
const subHours = require('date-fns/sub_hours');

const getFullTimestampRange = function(endTime, trainingHoursBack) {
   let i = 0;
   let timestamps = [];
   while (i < trainingHoursBack) {
      timestamps.push(subHours(endTime, i));
```

```
      i++;
   }
   return timestamps;
}

const HOURS_ON_MODEL_INDEX = 0;
const SWITCH_COUNT_MODEL_INDEX = 1;

function addNewDataToModel(model, measurement) {
   const parsedMLData = JSON.parse(measurement.MLPayload);
   model.target.unshift(parsedMLData.doesFail);
   model.dynamic_feat[HOURS_ON_MODEL_INDEX].unshift(parsedMLData.hoursOn);
   model.dynamic_feat[SWITCH_COUNT_MODEL_INDEX].unshift(parsedMLData.switchCountWeek);
}

function pullForwardLastData(model) {
   model.target.unshift(model.target[0]);

model.dynamic_feat[HOURS_ON_MODEL_INDEX].unshift(model.dynamic_feat[HOURS_ON_MODEL
_INDEX][0]);

model.dynamic_feat[SWITCH_COUNT_MODEL_INDEX].unshift(model.dynamic_feat[SWITCH_COU
NT_MODEL_INDEX][0]);
}

const buildModel = function(baseModel, sortedBulbMeasurements, timestamps) {
   let model = JSON.parse(JSON.stringify(baseModel));
   let bulbCursor = 0;
   timestamps.forEach((timestamp) => {
      if (sortedBulbMeasurements[bulbCursor] &&
isEqual(sortedBulbMeasurements[bulbCursor].timestamp, timestamp)){
         console.log(`${JSON.stringify(sortedBulbMeasurements[bulbCursor])} matched with
${timestamp}`);
         if(sortedBulbMeasurements[bulbCursor].MLPayload) {
            console.log("Found payload.  Adding to model");
            addNewDataToModel(model, sortedBulbMeasurements[bulbCursor]);
         } else {
            console.log("No payload found.  Pulling forward data from previous item.");
            pullForwardLastData(model);
         }
         bulbCursor=bulbCursor+1;
      } else {
         console.log(`${JSON.stringify(sortedBulbMeasurements[bulbCursor])} did not match current
${timestamp}.  Pulling forward previous data.`);
         pullForwardLastData(model);
      }
   });
   return model;
}

const buildDeepArModel = function(transformedBulbMeasurements, windowParameters) {
   const endTime = parse(transformedBulbMeasurements[0].timestamp);
   const timestampsForModel = getFullTimestampRange(endTime,
windowParameters.trainingHoursBack);
   const start = parse(timestampsForModel[timestampsForModel.length -1]).toISOString(); //reverse sorted
by TS from DDB
```

```
    const baseModel = {
        "start": start.substring(0, start.length - 1), //sagemaker requires no trailing Z for timezone so removing
it
        "target":[],
        "dynamic_feat":[[],[]]
    };

    return buildModel(baseModel, transformedBulbMeasurements, timestampsForModel);
}

module.exports = {
    buildDeepArModel: buildDeepArModel,
    getFullTimestampRange: getFullTimestampRange,
    buildModel: buildModel
};
"FILENAME: C:\dev\c3\rawRepos\machine-learning\functions\source\data-prep-lambda\lib\transformed-
data-retriever.js"

const awsHelpers = require('aws-helpers');
const { startOfHour, parse, subDays, subHours } = require('date-fns');

const getTrainingWindow = function(baseTimestamp, predictionDaysBack, trainingHoursNeeded) {
    const initialDate = startOfHour(parse(baseTimestamp));
    const endOfWindow = subDays(initialDate, predictionDaysBack).toISOString();
    return { endOfWindow: endOfWindow, trainingHoursBack: trainingHoursNeeded };
}

const PREDICTION_DAYS_BACK = 30;
const TRAINING_HOURS_BACK = 920;
const RECORD_LIMIT = TRAINING_HOURS_BACK;
const getTransformedData = async function(bulbName, baseTimestamp) {
    const awsRegion = process.env.AWS_REGION;
    const MLDataTable = process.env.DynamoDBML;
    const windowParameters = getTrainingWindow(baseTimestamp, PREDICTION_DAYS_BACK,
TRAINING_HOURS_BACK);
    const params = {
        TableName: MLDataTable,
        KeyConditionExpression: '#a = :hkey and #b < :endkey',
        ExpressionAttributeNames: {
            '#a': 'smartBulbName',
            '#b': 'timestamp'
        },
        ExpressionAttributeValues: {
          ':hkey': bulbName,
          ':endkey': windowParameters.endOfWindow
        },
        FilterExpression: "attribute_exists (MLPayload)",
        ProjectionExpression : "#b,MLPayload",
        Limit: RECORD_LIMIT,
        ScanIndexForward: false
    };
    const bulbData = await awsHelpers.queryDynamoDB(awsRegion, params, RECORD_LIMIT);
    if (!bulbData || bulbData.length === 0) {
        throw new Error(`No transformed bulb data found for given lightbulb ${bulbName} for dates before
${windowParameters.endOfWindow}`);
    }
```

```
   console.log(`Found ${bulbData.length} records for dates ending at
${windowParameters.endOfWindow}`);
   return { bulbData, windowParameters };
}

module.exports = {
   getTransformedData: getTransformedData,
   getTrainingWindow: getTrainingWindow
};
"FILENAME: C:\dev\c3\rawRepos\machine-learning\functions\source\data-prep-lambda\test\data-
prep.spec.js"

const sinon = require('sinon');
const chai = require('chai');
const chaiAsPromised = require('chai-as-promised');
const awsHelpers = require('aws-helpers');
const retriever = require('../lib/transformed-data-retriever');
// const { startOfHour, parse, subHours, isBefore} = require('date-fns');

chai.use(chaiAsPromised);
const { expect } = chai;

describe('Transformed Data Retriever Test', () => {
 const sandbox = sinon.createSandbox();

 sandbox.stub(awsHelpers, 'queryDynamoDB')
      .callsFake(function () { console.log('Stubbed kinesis put'); return new Promise((res) => res([]))});


 beforeEach(() => {

 });

 afterEach(() => {
  sandbox.restore();
 });

 describe('Retriever', () => {
  const timestamp = '2018-11-30T21:32:19.738Z';
  const daysToGoBack = 1;
  const windowResult = retriever.getTrainingWindow(timestamp, daysToGoBack, 24);
  it(`build a window that rounds to nearest hour and can subtract ${daysToGoBack} days from the base
time`, async () => {
    expect(windowResult.endOfWindow).to.eq('2018-11-29T21:00:00.000Z');
  });

  it('throw an error if no data is returned from the ddb call', async () => {
    sandbox.stub(awsHelpers, 'queryDynamoDB')
      .callsFake(function () { console.log('Stubbed kinesis put'); return new Promise((res) => res([]))});

    expect(retriever.getTransformedData("someBulb", '2018-11-
30T21:32:19.738Z')).to.eventually.throw();
  })
 });
});
```

```
"FILENAME: C:\dev\c3\rawRepos\machine-learning\functions\source\data-prep-lambda\test\deep-ar-model-builder.spec.js"

const sinon = require('sinon');
const chai = require('chai');
// const awsHelpers = require('aws-helpers');
const chaiAsPromised = require('chai-as-promised');
const builder = require('../lib/deep-ar-model-builder');
const parse = require('date-fns/parse');

chai.use(chaiAsPromised);
const { expect } = chai;

describe('Model Builder Lib Test', () => {
   const sandbox = sinon.createSandbox();

   beforeEach(() => {

   });

   afterEach(() => {
      sandbox.restore();
   });

   describe('Model Builder', () => {
     const earliestTimestamp = "2018-02-01T17:00:00.000Z";
     const testDDBData = [
        {
          "MLPayload": "{\"hoursOn\":5,\"switchCountWeek\":9,\"doesFail\":1}",
          "timestamp": "2018-02-01T23:00:00.000Z"
        },
        {
          "MLPayload": "{\"hoursOn\":4,\"switchCountWeek\":9,\"doesFail\":0}",
          "timestamp": "2018-02-01T22:00:00.000Z"
        },
        {
          "MLPayload": "{\"hoursOn\":3,\"switchCountWeek\":8,\"doesFail\":0}",
          "timestamp": "2018-02-01T21:00:00.000Z"
        },
        {
          "MLPayload": "{\"hoursOn\":3,\"switchCountWeek\":8,\"doesFail\":0}",
          "timestamp": "2018-02-01T20:00:00.000Z"
        },
        {
          "MLPayload": "{\"hoursOn\":2,\"switchCountWeek\":7,\"doesFail\":0}",
          "timestamp": "2018-02-01T19:00:00.000Z"
        },
        {
          "MLPayload": "{\"hoursOn\":1,\"switchCountWeek\":6,\"doesFail\":0}",
          "timestamp": earliestTimestamp
        }
       ];

       const hours = 7;
       const modelResult = builder.buildDeepArModel(testDDBData, { endOfWindow: parse("2018-02-01T23:00:00.000Z"), trainingHoursBack: hours});
```

```
      // console.log(JSON.stringify(modelResult));

    it('correctly set the start time to earliest records', async () => {
      expect(modelResult.start).to.eq(earliestTimestamp);
    });

    it('have a model with equal length target and feature arrays', async () => {
     expect(modelResult.target.length).to.eq(modelResult.dynamic_feat[0].length)
    });
  });
});
"FILENAME: C:\dev\c3\rawRepos\machine-learning\functions\source\predict-api-lambda\.eslintrc.js"

module.exports = {
 env: {
  es6: true,
  node: true,
  mocha: true,
  jest: true
 },
 extends: ['airbnb-base'],
 rules: {
  'arrow-parens': ['error', 'as-needed'],
  'comma-dangle': ['error', 'never'],
  'function-paren-newline': 'off',
  'no-underscore-dangle': 'off',
  'prefer-destructuring': 'off',
  'no-console': 'off',
  'no-await-in-loop': 'off',
  'import/no-extraneous-dependencies': [
   'warn',
   { devDependencies: false }
  ],
  'no-restricted-syntax': [
   'error',
   {
    selector: 'ForInStatement',
    message: 'for..in loops iterate over the entire prototype chain, which is virtually never what you want.
Use Object.{keys,values,entries}, and iterate over the resulting array.',
   },
   {
    selector: 'LabeledStatement',
    message: 'Labels are a form of GOTO; using them makes code confusing and hard to maintain and
understand.',
   },
   {
    selector: 'WithStatement',
    message: '`with` is disallowed in strict mode because it makes code impossible to predict and
optimize.',
   },
  ],
 },
 overrides: [
  {
   files: '**/*.spec.js',
   rules: {
```

```
        'no-unused-expressions': 'off',
        'prefer-arrow-callback': 'off',
        'import/no-extraneous-dependencies': [
          'error',
          { devDependencies: true }
        ]
      }
    }
  ]
};

"FILENAME: C:\dev\c3\rawRepos\machine-learning\functions\source\predict-api-lambda\index.js"

const AWS = require('aws-sdk');
const {
  addHours
} = require('date-fns');

exports.handler = (event, context, callback) => {
  console.log('Received event:', JSON.stringify(event, null, 2));

  const intervalInHours = parseInt(process.env.SageMakerIntervalInHours, 10);
  if (Number.isNaN(intervalInHours)) {
    var error = new Error("SageMakerIntervalInHours not set to a valid value");
    callback(error);
  }

  const historicalsStart = event.start;
  const target = event.target;
  const dynamicFeatures = event.dynamic_feat;
  const numIntervals = target.length;

  const historicals = {
    start: historicalsStart,
    target,
    dynamic_feat: dynamicFeatures
  };

  // make start date the next interval after the time corresponding to the last target
  const startDate = addHours(historicalsStart, numIntervals * intervalInHours + intervalInHours);

  executeSageMakerDeepAR(process.env.SageMakerEndpointName, historicals, startDate,
intervalInHours, callback);
};

function executeSageMakerDeepAR(endpointName, historicals, startDate, intervalInHours, callback) {
  const sagemakerruntime = new AWS.SageMakerRuntime();
  historicals.start = historicals.start.toString().replace('Z', '');

  const deepARBody = {
    instances: [historicals],
    configuration: {
      num_samples: 100,
      output_types: ['mean']
    }
  };
```

```
  const body = JSON.stringify(deepARBody);

  const params = {
    Body: body,
    EndpointName: endpointName,
    Accept: 'application/json',
    ContentType: 'application/json'
  };
  console.log('executing DeepAR endpoint %s with body %s', endpointName, body);
  sagemakerruntime.invokeEndpoint(params, (err, data) => {
    if (err) console.log(err, err.stack); // an error occurred
    else {
      const body = data.Body.toString();
      console.log('SageMaker predictions: ', body);
      const predictions = JSON.parse(body);
      const sageMakerResults = predictions.predictions[0];
      const meanValues = sageMakerResults.mean;
      const results = [];

      for (let i = 0; i < meanValues.length; i++) {
        results[i] = {
          when: addHours(startDate, i * intervalInHours).toISOString(),
          mean: meanValues[i]
        };
      }
      console.log('results: %s', JSON.stringify(results));
      callback(null, results);
    }
  });
}
```

"FILENAME: C:\dev\c3\rawRepos\machine-learning\functions\source\predict-api-lambda\package.json"

```
{
   "name": "enhance-lambda",
   "version": "0.0.1",
   "description": "A simple lambda that add appends data to price history records",
   "license": "MIT",
   "scripts": {
      "test": "nyc --check-coverage --lines 80 --functions 80 --branches 80 --reporter=cobertura mocha -R
mocha-multi-reporters --recursive test",
      "lint": "eslint . --fix",
      "prettier": "prettier --single-quote --write '**/*.js'"
   },
   "devDependencies": {
      "chai": "^4.2.0",
      "chai-as-promised": "7.1.1",
      "expect.js": "^0.3.1",
      "eslint": "^5.6.1",
      "eslint-config-airbnb-base": "^13.1.0",
      "eslint-plugin-import": "^2.14.0",
      "mocha": "^5.2.0",
      "mocha-multi-reporters": "^1.1.7",
      "nyc": "^13.0.1",
      "prettier": "^1.6.1",
      "sinon": "^6.3.5"
```

```
    },
    "dependencies": {
        "aws-helpers": "file:../aws-helpers",
        "async": "~2.6.0",
        "date-fns": "^1.29.0"
    },
    "prettier": {
        "singleQuote": true
    },
    "private": true
}
```

"FILENAME: C:\dev\c3\rawRepos\machine-learning\functions\source\sagemaker-endpoint-update-lambda\.eslintrc.js"

```
module.exports = {
  env: {
    es6: true,
    node: true,
    mocha: true,
    jest: true
  },
  extends: ['airbnb-base'],
  rules: {
    'arrow-parens': ['error', 'as-needed'],
    'comma-dangle': ['error', 'never'],
    'function-paren-newline': 'off',
    'no-underscore-dangle': 'off',
    'prefer-destructuring': 'off',
    'no-console': 'off',
    'no-await-in-loop': 'off',
    'import/no-extraneous-dependencies': [
      'warn',
      { devDependencies: false }
    ],
    'no-restricted-syntax': [
      'error',
      {
        selector: 'ForInStatement',
        message: 'for..in loops iterate over the entire prototype chain, which is virtually never what you want. Use Object.{keys,values,entries}, and iterate over the resulting array.',
      },
      {
        selector: 'LabeledStatement',
        message: 'Labels are a form of GOTO; using them makes code confusing and hard to maintain and understand.',
      },
      {
        selector: 'WithStatement',
        message: '`with` is disallowed in strict mode because it makes code impossible to predict and optimize.',
      },
    ],
  },
  overrides: [
    {
```

```
    files: '**/*.spec.js',
    rules: {
      'no-unused-expressions': 'off',
      'prefer-arrow-callback': 'off',
      'import/no-extraneous-dependencies': [
        'error',
        { devDependencies: true }
      ]
    }
   }
  ]
};

"FILENAME: C:\dev\c3\rawRepos\machine-learning\functions\source\sagemaker-endpoint-update-
lambda\index.js"

var AWS = require('aws-sdk');
var async = require('async');

exports.handler = (event, context, callback) => {
            var awsRegion = process.env.AWS_REGION;
            console.log('Received event:', JSON.stringify(event, null, 2));
    async.each(event.Records, function(record, callback){
                        createUpdateEndpoint(awsRegion, record, callback);
            }, function(err){
                        if(err) {
          console.log("Failed to process events.");
          context.fail("Failed to process events" + err);
       }
       else {
          console.log("Successfully processed " + event.Records.length + " records.");
          context.succeed("Successfully processed " + event.Records.length + " records.");
       }
            });
};

function createUpdateEndpoint(awsRegion, record, callback){
            var s3Bucket = record.s3.bucket.name;
            var key = record.s3.object.key;
            var s3Url = `s3://${s3Bucket}/${key}`;
            var keySplit = key.split("/");
            var dateString = keySplit[1];
            createModel(awsRegion, s3Url, dateString, function(err){
                        if(err){
                                    callback(err);
                        }
                        else{
                                    createEndpointConfiguration(awsRegion, dateString, function(err){
                                                if(err){
                                                            callback(err);
                                                }
                                                else{
                                                            checkForEndpoint(awsRegion, function(err, endpointName){
                                                                        if(err){
                                                                                    callback(err);
                                                                        }
```

```
                                                else if(endpointName){
                                                        updateEndpoint(awsRegion, dateString,
function(err){
                                                                if(err){
                                                                        callback(err);
                                                                }
                                                                else{
                                                                        callback(null);
                                                                }
                                                        });
                                                }
                                                else{
                                                        createEndpoint(awsRegion, dateString,
function(err){
                                                                if(err){
                                                                        callback(err);
                                                                }
                                                                else{
                                                                        callback(null);
                                                                }
                                                        })
                                                }
                                        });
                                }
                        });
                }
        });
}

function getModelCreationParams(s3Url, dateString){
        var modelName = getModelName(dateString);
        var modelExecutionRoleArn = process.env.ModelRoleArn;
        var modelImage = process.env.TrainingImage;
        var params = {
                ExecutionRoleArn: modelExecutionRoleArn,
                ModelName: modelName,
                PrimaryContainer: {
                Image: modelImage,
                ModelDataUrl: s3Url
                }
        };
        return params;
}


function getModelName(dateString){
        var modelNamePrefix = "risk-score-model-";
        return modelNamePrefix + dateString;
}

function getEndpointConfigurationName(dateString){
        var endpointConfigurationPrefix = "risk-score-endpoint-config-";
        return endpointConfigurationPrefix + dateString;
}

function getEndpointName(){
```

```
	return process.env.SageMakerEndpointName;
}

function createModel(awsRegion, s3Url, dateString, callback){
	var modelCreationParams = getModelCreationParams(s3Url, dateString);
	var name= {ModelName:getModelName(dateString)};
	var sagemaker = new AWS.SageMaker({region: awsRegion});
	sagemaker.describeModel(name,function(err,response){

		if(err){
			console.log(`model doesnt exist creating: ${JSON.stringify(err)}`);
			sagemaker.createModel(modelCreationParams, function(err, response){
			if(err){
				console.log(`Failed to create model: ${JSON.stringify(err)}`);
				callback(err);
			}
			else{
				console.log(`Successfully created model:
${JSON.stringify(response)}`);
				callback(null);
			}
		});

		}
		else{
			console.log(`Model already exists: ${JSON.stringify(response)}`);
			callback(null);
		}
	});

}

function getEndpointConfigurationParams(dateString){
	var endpointConfigName = getEndpointConfigurationName(dateString);
	var modelName = getModelName(dateString);
	var initialInstanceCount = parseInt(process.env.InitialEndpointInstanceCount);
	var InstanceType = process.env.EndpointInstanceType;
	var params = {
		EndpointConfigName: endpointConfigName,
		ProductionVariants: [
		{
			InitialInstanceCount: initialInstanceCount,
			InstanceType: InstanceType,
			ModelName: modelName,
			VariantName: 'AllTraffic'
		}
		]
	};
	return params;
}

function createEndpointConfiguration(awsRegion, dateString, callback){
	var endpointConfigurationParams = getEndpointConfigurationParams(dateString);
	var name= {EndpointConfigName:getEndpointConfigurationName(dateString)};
	var sagemaker = new AWS.SageMaker({region: awsRegion});
	sagemaker.describeEndpointConfig(name,function(err,response){
```

```
                if(err){
                        console.log(`endpointConfig doesnt exist creating: ${JSON.stringify(err)}`);
                        sagemaker.createEndpointConfig(endpointConfigurationParams, function(err,
response){
                        if(err){
                                console.log(`Failed to create endpoint configuration:
${JSON.stringify(err)}`);
                                callback(err);
                        }
                        else{
                                console.log(`Successfully created endpoint configuration:
${JSON.stringify(response)}`);
                                callback(null);
                        }
                });
                }
                else{
                        console.log(`Endpoint config already exists: ${JSON.stringify(response)}`);
                        callback(null);
                }
        });

}

function getDescribeEndpointParams(){
        var endpointName = getEndpointName();
        var params = {
                EndpointName: endpointName
        };
        return params;
}

function checkForEndpoint(awsRegion, callback){
        var describeEndpointParams = getDescribeEndpointParams();
        var sagemaker = new AWS.SageMaker({region: awsRegion});
        sagemaker.describeEndpoint(describeEndpointParams, function(err, response){
                if(err){
                        var errorString = JSON.stringify(err);
                        if(errorString.indexOf("Could not find endpoint") > -1){
                                console.log("No endpoint found. Will create new endpoint.");
                                callback(null, null);
                        }
                        else{
                                console.log(`Failed to describe endpoint: ${errorString}`);
                                callback(err);
                        }
                }
                else{
                        console.log(`Found existing endpoint: ${JSON.stringify(response)}`);
                        callback(null, response.EndpointName);
                }
        });
}

function getCreateUpdateEndpointParams(endpointConfigName){
        var endpointName = getEndpointName();
```

```
	var params = {
		EndpointConfigName: endpointConfigName,
		EndpointName: endpointName
	};
	return params;
}

function createEndpoint(awsRegion, dateString, callback){
	var endpointConfigName = getEndpointConfigurationName(dateString);
	var createUpdateEndpointParams = getCreateUpdateEndpointParams(endpointConfigName);
	console.log(`Creating endpoint with params: ${JSON.stringify(createUpdateEndpointParams)}`);
	var sagemaker = new AWS.SageMaker({region: awsRegion});
	sagemaker.createEndpoint(createUpdateEndpointParams, function(err, response){
		if(err){
			console.log(`Failed to create endpoint: ${JSON.stringify(err)}`);
			callback(err);
		}
		else{
			console.log(`Successfully created endpoint: ${JSON.stringify(response)}`);
			callback(null);
		}
	});
}

function updateEndpoint(awsRegion, dateString, callback){
	var endpointConfigName = getEndpointConfigurationName(dateString);
	var createUpdateEndpointParams = getCreateUpdateEndpointParams(endpointConfigName);
	console.log(`Updating endpoint with params: ${JSON.stringify(createUpdateEndpointParams)}`);
	var sagemaker = new AWS.SageMaker({region: awsRegion});
	sagemaker.updateEndpoint(createUpdateEndpointParams, function(err, response){
		if(err){
			console.log(`Failed to update endpoint: ${JSON.stringify(err)}`);
			callback(err);
		}
		else{
			console.log(`Successfully updated endpoint: ${JSON.stringify(response)}`);
			callback(null);
		}
	});
}
"FILENAME: C:\dev\c3\rawRepos\machine-learning\functions\source\sagemaker-endpoint-update-lambda\package.json"

{
  "name": "enhance-lambda",
  "version": "0.0.1",
  "description": "A simple lambda that add appends data to price history records",
  "license": "MIT",
  "scripts": {
    "test": "nyc --check-coverage --lines 80 --functions 80 --branches 80 --reporter=cobertura mocha -R mocha-multi-reporters --recursive test",
    "lint": "eslint . --fix",
    "prettier": "prettier --single-quote --write '**/*.js'"
  },
  "devDependencies": {
    "chai": "^4.2.0",
```

```
        "chai-as-promised": "7.1.1",
        "expect.js": "^0.3.1",
        "eslint": "^5.6.1",
        "eslint-config-airbnb-base": "^13.1.0",
        "eslint-plugin-import": "^2.14.0",
        "mocha": "^5.2.0",
        "mocha-multi-reporters": "^1.1.7",
        "nyc": "^13.0.1",
        "prettier": "^1.6.1",
        "sinon": "^6.3.5"
    },
    "dependencies": {
        "async": "~2.6.0"
    },
    "prettier": {
        "singleQuote": true
    },
    "private": true
}

"FILENAME: C:\dev\c3\rawRepos\machine-learning\functions\source\sagemaker-training-kickoff-lambda\.eslintrc.js"

module.exports = {
  env: {
    es6: true,
    node: true,
    mocha: true,
    jest: true
  },
  extends: ['airbnb-base'],
  rules: {
    'arrow-parens': ['error', 'as-needed'],
    'comma-dangle': ['error', 'never'],
    'function-paren-newline': 'off',
    'no-underscore-dangle': 'off',
    'prefer-destructuring': 'off',
    'no-console': 'off',
    'no-await-in-loop': 'off',
    'import/no-extraneous-dependencies': [
      'warn',
      { devDependencies: false }
    ],
    'no-restricted-syntax': [
      'error',
      {
        selector: 'ForInStatement',
        message: 'for..in loops iterate over the entire prototype chain, which is virtually never what you want. Use Object.{keys,values,entries}, and iterate over the resulting array.',
      },
      {
        selector: 'LabeledStatement',
        message: 'Labels are a form of GOTO; using them makes code confusing and hard to maintain and understand.',
      },
      {
```

```
      selector: 'WithStatement',
      message: '`with` is disallowed in strict mode because it makes code impossible to predict and
optimize.',
    },
   ],
  },
  overrides: [
   {
    files: '**/*.spec.js',
    rules: {
     'no-unused-expressions': 'off',
     'prefer-arrow-callback': 'off',
     'import/no-extraneous-dependencies': [
      'error',
      { devDependencies: true }
     ]
    }
   }
  ]
};

"FILENAME: C:\dev\c3\rawRepos\machine-learning\functions\source\sagemaker-training-kickoff-
lambda\index.js"

var AWS = require('aws-sdk');

exports.handler = (event, context, callback) => {
        var awsRegion = process.env.AWS_REGION;
        console.log('Received event:', JSON.stringify(event, null, 2));
        kickoffTrainingJob(awsRegion, event, callback);
};

function kickoffTrainingJob(awsRegion, record, callback){
        getParams(awsRegion, record, function(err, params){
                if(err){
                        callback(err);
                }
                else{
                        var sagemaker = new AWS.SageMaker({region: awsRegion});
                        sagemaker.createTrainingJob(params, function(err, response) {
                                if (err){
                                        console.log(`Failed to create training job: ${err}`);
                                        callback(err);
                                }
                                else{
                                        console.log(`Successfully created training job:
${JSON.stringify(response)}`);
                                        callback(null);
                                }
                        });
                }
        });
}

function getParams(awsRegion, record, callback){
  var s3Bucket = process.env.DataLakeMLS3Bucket;
```

```
  var startTime = record.startTime ? record.startTime : new Date();
        var trainingKey = startTime + "/train/";
        var testKey = trainingKey.replace("train/", "test/");
        var hyperParameters = JSON.parse(process.env.HyperParameters);
        var dateString = new Date().toISOString().replace(":", "-").replace(":", "-").slice(0, -5);
        var instanceCount = parseInt(process.env.TrainingInstanceCount);
        var instanceType = process.env.TrainingInstanceType;
        var instanceVolumeSize = parseInt(process.env.TrainingInstanceVolumeSize);
        var inputDataConfig = getInputDataConfig(s3Bucket, trainingKey, testKey);
        var outputBucket = process.env.OutputBucketName;
        var outputPrefix = process.env.SageMakerOutputModelPrefix;
        var outputDataConfig = getOutputDataConfig(outputBucket, outputPrefix, dateString);
        var resourceConfig = getResourceConfig(instanceCount, instanceType, instanceVolumeSize);
        var trainingJobName = getTrainingJobName(dateString);
        var roleArn = process.env.TrainingRoleArn;
        var trainingImage = process.env.TrainingImage;
        var algorithmSpecification = getAlgorithmSpecification(trainingImage);
        var maxRuntimeSeconds = process.env.TrainingMaxRuntimeSeconds;
        var stoppingCondition = getStoppingCondition(maxRuntimeSeconds);
        var params = {
                AlgorithmSpecification: algorithmSpecification,
                InputDataConfig: inputDataConfig,
                OutputDataConfig: outputDataConfig,
                ResourceConfig: resourceConfig,
                RoleArn: roleArn,
                StoppingCondition: stoppingCondition,
                TrainingJobName: trainingJobName,
                HyperParameters: hyperParameters
        };
        callback(null,  params);
}


function getInputDataConfig(bucketName, trainingDataKey, testDataKey){
        var trainingDataUri = `s3://${bucketName}/${trainingDataKey}`;
        var testDataUri = `s3://${bucketName}/${testDataKey}`;
        var channels = [
           {
            ChannelName: 'train',
            DataSource: {
              S3DataSource: {
                S3DataType: 'S3Prefix',
                S3Uri: trainingDataUri
              }
            },
            ContentType: 'json'
           },
           {
            ChannelName: 'test',
            DataSource: {
              S3DataSource: {
                S3DataType: 'S3Prefix',
                S3Uri: testDataUri
              }
            },
            ContentType: 'json'
```

```
            }
        ];
        return channels;
}

function getOutputDataConfig(outputBucket, outputPrefix, dateString){
        var outputDataUri = `s3://${outputBucket}/${outputPrefix}/${dateString}`;
        var outputDataConfig = {
        S3OutputPath: outputDataUri
        };
        return outputDataConfig;
}

function getResourceConfig(instanceCount, instanceType, instanceVolumeSize){
        var resourceConfig = {
        InstanceCount: instanceCount,
        InstanceType: instanceType,
        VolumeSizeInGB: instanceVolumeSize
        };
        return resourceConfig;
}

function getTrainingJobName(dateString){
        return `training-${dateString}`;
}

function getAlgorithmSpecification(trainingImage){
        var algorithmSpecification = {
                TrainingImage: trainingImage,
        TrainingInputMode: 'File'
        };
   return algorithmSpecification;
}

function getStoppingCondition(maxRuntimeSeconds){
        var stoppingCondition = {
                MaxRuntimeInSeconds: maxRuntimeSeconds
        };
        return stoppingCondition;
}
```

"FILENAME: C:\dev\c3\rawRepos\machine-learning\functions\source\sagemaker-training-kickoff-lambda\package.json"

```
{
   "name": "enhance-lambda",
   "version": "0.0.1",
   "description": "A simple lambda that add appends data to price history records",
   "license": "MIT",
   "scripts": {
      "test": "nyc --check-coverage --lines 80 --functions 80 --branches 80 --reporter=cobertura mocha -R
mocha-multi-reporters --recursive test",
      "lint": "eslint . --fix",
      "prettier": "prettier --single-quote --write '**/*.js'"
   },
   "devDependencies": {
```

```
      "chai": "^4.2.0",
      "chai-as-promised": "7.1.1",
      "expect.js": "^0.3.1",
      "eslint": "^5.6.1",
      "eslint-config-airbnb-base": "^13.1.0",
      "eslint-plugin-import": "^2.14.0",
      "mocha": "^5.2.0",
      "mocha-multi-reporters": "^1.1.7",
      "nyc": "^13.0.1",
      "prettier": "^1.6.1",
      "sinon": "^6.3.5"
   },
   "dependencies": {
      "aws-helpers": "file:../aws-helpers"
   },
   "prettier": {
      "singleQuote": true
   },
   "private": true
}

"FILENAME: C:\dev\c3\rawRepos\machine-learning\templates\deliver.template"

{
   "AWSTemplateFormatVersion": "2010-09-09",
   "Description": "Creates RESTful APIs to get historical and predicted values from the data stream. (qs-
1onlac5tl)",
   "Parameters": {
      "LambdaCodeS3Bucket": {
         "Type": "String",
         "Description": "S3 Bucket where lambda code exists",
         "Default": "lightbulb-lambda-code"
      },
      "LambdaCodeS3KeyPrefix": {
         "Type": "String",
         "Description": "S3 Bucket where lambda code exists",
         "Default": "ml-artifacts"
      },
      "PredictApiLambdaCodeLocation": {
         "Type": "String",
         "Description": "S3 Path where Deliver API Lambda code exists",
         "Default": "predict-api-lambda.zip"
      },
      "SageMakerEndpointName": {
         "Type": "String",
         "Description": "Name of the SageMaker endpoint that will be executed to generate predictions",
         "Default": "risk-score-predictions"
      },
      "SageMakerIntervalInHours": {
         "Type": "String",
         "Description": "The interval in hours of the time series that SageMaker uses for the model and
predictions",
         "Default": "1"
      },
      "DataLakeS3BucketName": {
         "Type": "String",
```

```
      "Description": "S3 Bucket where data lake lives",
      "Default": "lightbulb-ml-data"
    }
  },
  "Resources": {
    "PredictApiLambdaRole": {
      "Type": "AWS::IAM::Role",
      "Properties": {
        "AssumeRolePolicyDocument": {
          "Version": "2012-10-17",
          "Statement": [{
            "Effect": "Allow",
            "Principal": {
              "Service": ["lambda.amazonaws.com"]
            },
            "Action": ["sts:AssumeRole"]
          }]
        },
        "Path": "/",
        "Policies": [{
          "PolicyName": {
            "Fn::Sub": "PredictApiLambdaPolicy"
          },
          "PolicyDocument": {
            "Version": "2012-10-17",
            "Statement": [
              {
                "Effect": "Allow",
                "Action": [
                  "s3:ListBucket"
                ],
                "Resource": { "Fn::Sub": "arn:aws:s3:::${DataLakeS3BucketName}" }
              },
              {
                "Effect": "Allow",
                "Action": [
                  "s3:GetObject"
                ],
                "Resource": { "Fn::Sub": "arn:aws:s3:::${DataLakeS3BucketName}/transform/*"}
              },
              {
                "Action": [
                  "sagemaker:InvokeEndpoint",
                  "sagemaker:DescribeEndpoint",
                  "sagemaker:DescribeEndpointConfiguration"
                ],
                "Effect": "Allow",
                "Resource": {
                  "Fn::Sub": "arn:aws:sagemaker:*:*:endpoint/${SageMakerEndpointName}"
                },
                "Sid": "GetPredictions"
              },
              {
                "Action": [
                  "logs:CreateLogGroup",
                  "logs:CreateLogStream",
```

```
                    "logs:PutLogEvents"
                ],
                "Effect": "Allow",
                "Resource": "*",
                "Sid": "Logs"
              }
            ]}
          }],
          "RoleName": {
            "Fn::Sub": "PredictApiLambdaRole"
          }
        }
      },
    "ApiGatewayCloudWatchLogsRole": {
      "Type": "AWS::IAM::Role",
      "Properties": {
        "RoleName": {
          "Fn::Sub": "ApiGatewayCloudWatchLogsRole"
        },
        "AssumeRolePolicyDocument": {
          "Version": "2012-10-17",
          "Statement": [{
            "Effect": "Allow",
            "Principal": {
              "Service": ["apigateway.amazonaws.com"]
            },
            "Action": ["sts:AssumeRole"]
          }]
        },
        "Policies": [{
          "PolicyName": {
            "Fn::Sub": "ApiGatewayLogsPolicy"
          },
          "PolicyDocument": {
            "Version": "2012-10-17",
            "Statement": [{
              "Effect": "Allow",
              "Action": [
                "logs:CreateLogGroup",
                "logs:CreateLogStream",
                "logs:DescribeLogGroups",
                "logs:DescribeLogStreams",
                "logs:PutLogEvents",
                "logs:GetLogEvents",
                "logs:FilterLogEvents"
              ],
              "Resource": "*"
            }]
          }
        }]
      }
    },
    "PredictApiLambda": {
      "Type": "AWS::Lambda::Function",
      "Properties": {
        "FunctionName": {
```

```
            "Fn::Sub": "PredictApiLambda"
        },
        "Handler": "index.handler",
        "Role": {
            "Fn::GetAtt": ["PredictApiLambdaRole", "Arn"]
        },
        "Code": {
            "S3Bucket": {"Ref" : "LambdaCodeS3Bucket"},
            "S3Key": {
                "Fn::Sub": "${LambdaCodeS3KeyPrefix}/${PredictApiLambdaCodeLocation}"
            }
        },
        "Runtime": "nodejs8.10",
        "Timeout": 30,
        "Environment": {
            "Variables": {
                "OutputBucketName": {
                    "Fn::Sub": "${DataLakeS3BucketName}"
                },
                "ObjectPrefix": "transform",
                "SageMakerEndpointName": {
                    "Fn::Sub": "${SageMakerEndpointName}"
                },
                "SageMakerIntervalInHours": {
                    "Ref": "SageMakerIntervalInHours"
                },
            }
        }
    }
},
"ApiGatewayAccount": {
    "Type": "AWS::ApiGateway::Account",
    "Properties": {
        "CloudWatchRoleArn": {
            "Fn::GetAtt": ["ApiGatewayCloudWatchLogsRole", "Arn"]
        }
    }
},
"DeliverApi": {
    "Type": "AWS::ApiGateway::RestApi",
    "Properties": {
        "Name": {
            "Fn::Sub": "DeliverApi"
        },
        "Description": "API used for Delivering Transform Data",
        "FailOnWarnings": true
    }
},
"PredictApiLambdaPermission": {
    "Type": "AWS::Lambda::Permission",
    "Properties": {
        "Action": "lambda:invokeFunction",
        "FunctionName": {
            "Fn::GetAtt": ["PredictApiLambda", "Arn"]
        },
        "Principal": "apigateway.amazonaws.com",
```

```
        "SourceArn": {
          "Fn::Sub": "arn:aws:execute-api:${AWS::Region}:${AWS::AccountId}:${DeliverApi}/*"
        }
      }
    },
    "DeliverApiDeployment": {
      "Type": "AWS::ApiGateway::Deployment",
      "DependsOn": ["PredictApiPostRequest"],
      "Properties": {
        "RestApiId": {
          "Ref": "DeliverApi"
        },
        "StageName": "DummyStage"
      }
    },
    "DeliverApiStage": {
      "DependsOn": ["ApiGatewayAccount"],
      "Type": "AWS::ApiGateway::Stage",
      "Properties": {
        "DeploymentId": {
          "Ref": "DeliverApiDeployment"
        },
        "MethodSettings": [{
          "DataTraceEnabled": true,
          "HttpMethod": "GET",
          "LoggingLevel": "INFO",
          "ResourcePath": "/*"
        }],
        "RestApiId": {
          "Ref": "DeliverApi"
        },
        "StageName": "LATEST"
      }
    },
    "PredictApiResource": {
      "Type": "AWS::ApiGateway::Resource",
      "Properties": {
        "RestApiId": {
          "Ref": "DeliverApi"
        },
        "ParentId": {
          "Fn::GetAtt": ["DeliverApi", "RootResourceId"]
        },
        "PathPart": "predict"
      }
    },
    "PredictApiModel": {
      "Type": "AWS::ApiGateway::Model",
      "Properties": {
        "RestApiId": { "Ref": "DeliverApi" },
        "ContentType": "application/json",
        "Description": "Schema for PredictApi",
        "Name": "PredictApiModel",
        "Schema": {
          "$schema": "http://json-schema.org/draft-04/schema#",
          "title": "PredictApiModel",
```

```
                "type": "object",
                "properties": {
                  "start": { "type": "string" },
                  "target": { "type": "array"},
                  "dynamic_feat": {"type": "array"}
                }
              }
            }
        },
        "PredictApiPostRequest": {
          "DependsOn": "PredictApiLambdaPermission",
          "Type": "AWS::ApiGateway::Method",
          "Properties": {
            "AuthorizationType": "NONE",
            "HttpMethod": "POST",
            "Integration": {
              "Type": "AWS",
              "IntegrationHttpMethod": "POST",
              "Uri": {
                "Fn::Join": ["", ["arn:aws:apigateway:", {
                  "Ref": "AWS::Region"
                }, ":lambda:path/2015-03-31/functions/", {
                  "Fn::GetAtt": ["PredictApiLambda", "Arn"]
                }, "/invocations"]]
              },
              "IntegrationResponses": [{
                "StatusCode": 200
              }]
            },
            "ResourceId": {
              "Ref": "PredictApiResource"
            },
            "RestApiId": {
              "Ref": "DeliverApi"
            },
            "MethodResponses": [{
              "StatusCode": 200
            }]
          }
        }
    },
    "Outputs": {
      "PredictEndpointUrl": {
        "Description": "URL for the predict endpoint",
        "Value": {"Fn::Sub": "https://${DeliverApi}.execute-
api.${AWS::Region}.amazonaws.com/${DeliverApiStage}/predict"}
      }
    }
}
"FILENAME: C:\dev\c3\rawRepos\machine-learning\templates\prep.template"

{
    "AWSTemplateFormatVersion": "2010-09-09",
    "Description": "Creates the infrastructure for the data prep lambda",
    "Parameters": {
      "LambdaCodeS3Bucket": {
```

```
        "Type": "String",
        "Description": "Prefix of the S3 Bucket where lambda code exists. The bucket name will be the
value of this parameter.",
        "Default":"lightbulb-lambda-code"
      },
      "LambdaCodeS3KeyPrefix": {
        "Type": "String",
        "Description": "S3 Key where lambda code exists",
        "Default": "ml-artifacts"
      },
      "DataPrepLambdaCodeLocation": {
        "Type": "String",
        "Description": "S3 Path where data prep lambda code exists",
        "Default": "data-prep-lambda.zip"
      },
      "DataLakeMLDDB" :{
        "Type": "String",
        "Description": "DDB where transformed ml data lives",
        "Default": "lightbulb-app-ml-ddb"
      },
      "DataLakeMLS3Bucket": {
        "Type": "String",
        "Description": "S3 Bucket where ml data lives",
        "Default": "lightbulb-ml-data"
      }
    },
    "Resources": {
          "DataPrepLambdaRole": {
        "Type": "AWS::IAM::Role",
        "Properties": {
          "AssumeRolePolicyDocument": {
            "Version": "2012-10-17",
            "Statement": [{
              "Effect": "Allow",
              "Principal": {
                "Service": ["lambda.amazonaws.com"]
              },
              "Action": ["sts:AssumeRole"]
            }]
          },
          "Path": "/",
          "Policies": [{
            "PolicyName": {
              "Fn::Sub": "DataPrepLambdaPolicy"
            },
            "PolicyDocument": {
              "Version": "2012-10-17",
              "Statement": [{
                    "Effect": "Allow",
                    "Action": [
                      "logs:CreateLogGroup",
                      "logs:CreateLogStream",
                      "logs:PutLogEvents"
                    ],
                    "Resource": "*"
                  },
```

```
                {
                  "Effect": "Allow",
                  "Action": [
                    "s3:ListBucket"
                  ],
                  "Resource": {
                     "Fn::Sub": "arn:aws:s3:::${DataLakeMLS3Bucket}"
                  }
                },
                {
                  "Effect": "Allow",
                  "Action": [
                    "dynamodb:Query"
                  ],
                  "Resource": {
                     "Fn::Sub": "arn:aws:dynamodb:us-east-
1:466081484468:table/${DataLakeMLDDB}"
                  }
                },
                {
                  "Effect": "Allow",
                  "Action": [
                    "s3:Put*"
                  ],
                  "Resource": [
                    {
                      "Fn::Join": [
                        "/",
                        [
                          {
                            "Fn::Sub": "arn:aws:s3:::${DataLakeMLS3Bucket}"
                          },
                          "*"
                        ]
                      ]
                    }
                  ]
                }
              ]
            }
          }],
          "RoleName": {
            "Fn::Sub": "DataPrepLambdaRole"
          }
        }
      },
      "DataPrepLambda": {
        "Type": "AWS::Lambda::Function",
        "Properties": {
          "FunctionName": {
            "Fn::Sub": "DataPrepLambda"
          },
          "Handler": "index.handler",
          "Role": {
            "Fn::GetAtt": ["DataPrepLambdaRole", "Arn"]
          },
```

```
        "Code": {
          "S3Bucket": {
            "Ref": "LambdaCodeS3Bucket"
          },
          "S3Key": {
            "Fn::Sub": "${LambdaCodeS3KeyPrefix}/${DataPrepLambdaCodeLocation}"
          }
        },
        "Runtime": "nodejs8.10",
        "Timeout": 300,
        "MemorySize": 1024,
        "Environment": {
          "Variables": {
            "OutputBucketName": {
              "Fn::Sub": "${DataLakeMLS3Bucket}"
            },
            "DynamoDBML": {
              "Fn::Sub": "${DataLakeMLDDB}"
            }
          }
        }
      }
    },
    "DataPrepTriggerRule": {
      "Type": "AWS::Events::Rule",
      "Properties": {
        "Name": "DataPrepTriggerRule",
        "ScheduleExpression": "cron(0 19 ? * WED *)",
        "Targets": [{
          "Id": "DataPrepTriggerScheduler",
          "Arn": { "Ref" : "DataPrepStateMachine"},
          "RoleArn": {
            "Fn::GetAtt": ["DataPrepTriggerRuleRole", "Arn"]
          }
        }]
      },
      "DependsOn" : "DataPrepTriggerRuleRole"
    },
    "DataPrepTriggerRuleRole": {
      "Type": "AWS::IAM::Role",
      "Properties": {
        "AssumeRolePolicyDocument": {
          "Version": "2012-10-17",
          "Statement": [
            {
              "Effect": "Allow",
              "Principal": {
                "Service": [
                  {
                    "Fn::Sub": "states.${AWS::Region}.amazonaws.com"
                  },
                  "events.amazonaws.com"
                ]
              },
              "Action": "sts:AssumeRole"
            }
```

```
            ]
         },
         "Path": "/",
         "Policies": [
            {
               "PolicyName": "DataPrepTriggerRulePolicy",
               "PolicyDocument": {
                  "Version": "2012-10-17",
                  "Statement": [
                     {
                        "Effect": "Allow",
                        "Action": [
                           "states:DescribeStateMachine",
                           "states:StartExecution",
                           "states:DeleteStateMachine",
                           "states:ListExecutions",
                           "states:UpdateStateMachine"
                        ],
                        "Resource": { "Ref" : "DataPrepStateMachine" }
                     }
                  ]
               }
            }
         ],
         "RoleName": {
            "Fn::Sub": "DataPrepTriggerRuleRole"
         }
      }
   },
   "StatesExecutionRole": {
      "Type": "AWS::IAM::Role",
      "Properties": {
         "AssumeRolePolicyDocument": {
            "Version": "2012-10-17",
            "Statement": [
               {
                  "Effect": "Allow",
                  "Principal": {
                     "Service": [
                        {
                           "Fn::Sub": "states.${AWS::Region}.amazonaws.com"
                        },
                        "events.amazonaws.com"
                     ]
                  },
                  "Action": "sts:AssumeRole"
               }
            ]
         },
         "Path": "/",
         "Policies": [
            {
               "PolicyName": "StatesExecutionPolicy",
               "PolicyDocument": {
                  "Version": "2012-10-17",
                  "Statement": [
```

```
                    {
                      "Effect": "Allow",
                      "Action": [
                        "lambda:InvokeFunction"
                      ],
                      "Resource": "*"
                    }
                  ]
                }
              }
            ],
            "RoleName": {
              "Fn::Sub": "StatesExecutionRole"
            }
          }
        },
        "DataPrepStateMachine": {
          "Type": "AWS::StepFunctions::StateMachine",
          "Properties": {
            "StateMachineName": "DataPrepStateMachine",
            "DefinitionString": {
              "Fn::Join": [
                "",
                ["{\"StartAt\": \"FirstFunction\",",
                "    \"States\" : {",
                "       \"FirstFunction\" : {",
                "          \"Type\" : \"Task\", ",
                {"Fn::Sub" : ["          \"Resource\" : \"${lambdaArn}\",", { "lambdaArn": { "Fn::GetAtt":
["DataPrepLambda", "Arn"]}}]},
                "          \"Next\": \"SecondFunction\"",
                "       },",
                "       \"SecondFunction\" : {",
                "          \"Type\" : \"Task\", ",
                {"Fn::Sub" : ["          \"Resource\" : \"${lambdaArn}\",", { "lambdaArn": { "Fn::GetAtt":
["DataPrepLambda", "Arn"]}}]},
                "          \"Next\": \"ThirdFunction\"",
                "       },",
                "       \"ThirdFunction\" : {",
                "          \"Type\" : \"Task\", ",
                {"Fn::Sub" : ["          \"Resource\" : \"${lambdaArn}\",", { "lambdaArn": { "Fn::GetAtt":
["DataPrepLambda", "Arn"]}}]},
                "          \"Next\": \"FourthFunction\"",
                "       },",
                "       \"FourthFunction\" : {",
                "          \"Type\" : \"Task\", ",
                {"Fn::Sub" : ["          \"Resource\" : \"${lambdaArn}\",", { "lambdaArn": { "Fn::GetAtt":
["DataPrepLambda", "Arn"]}}]},
                "          \"Next\": \"TrainingKickoff\"",
                "       },",
                "       \"TrainingKickoff\" : {",
                "          \"Type\" : \"Task\", ",
                {"Fn::Sub" : ["          \"Resource\" : \"${lambdaArn}\",", { "lambdaArn": {
"Fn::ImportValue" : "SageMakerTrainingKickoffLambda"}}]},
                "          \"End\" : true",
                "       }",
                "    }",
```

```
                "}"]
            ]
          },
          "RoleArn": {
            "Fn::GetAtt": [
              "StatesExecutionRole",
              "Arn"
            ]
          }
        }
      }
    }
}
"FILENAME: C:\dev\c3\rawRepos\machine-learning\templates\sagemaker.template"

{
  "AWSTemplateFormatVersion": "2010-09-09",
  "Description": "Creates the resources needed to train and get predictions from SageMaker Models.
Lambda functions are used to create training jobs and prediction endpoints based on changes in S3 data.
(qs-1onlac5tg)",
  "Parameters": {
    "LambdaCodeS3Bucket": {
      "Type": "String",
      "Description": "S3 Bucket where lambda code exists",
      "Default":"lightbulb-lambda-code"
    },
    "LambdaCodeS3KeyPrefix": {
      "Type": "String",
      "Description": "S3 Key where lambda code exists",
      "Default": "ml-artifacts"
    },
    "SageMakerTrainingKickoffLambdaCodeLocation": {
      "Type": "String",
      "Description": "S3 Path where SageMaker Training Kickoff Lambda code exists",
      "Default": "sagemaker-training-kickoff-lambda.zip"
    },
    "SageMakerEndpointUpdateLambdaCodeLocation": {
       "Type": "String",
      "Description": "S3 Path where SageMaker Endpoint Update Lambda code exists",
      "Default": "sagemaker-endpoint-update-lambda.zip"
    },
    "DataLakeMLS3Bucket": {
      "Type": "String",
      "Description": "S3 Bucket where data lake lives",
      "Default": "lightbulb-ml-data"
    },
    "DataLakeMLModelS3Bucket": {
      "Type": "String",
      "Description": "S3 Bucket where ML model lives",
      "Default": "lightbulb-ml-model"
    },
    "SageMakerEndpointName": {
      "Type": "String",
      "Description": "Name of the SageMaker endpoint that will be executed to generate predictions",
      "Default": "risk-score-predictions"
    },
```

```
      "HyperParameters": {
         "Type": "String",
         "Description": "The hyperparameters that will be passed to the SageMaker training algorithm, in
JSON format.  See the HyperParameters section under the algorithm you are using from
https://docs.aws.amazon.com/sagemaker/latest/dg/algos.html",
         "Default":
"{\"time_freq\":\"H\",\"context_length\":\"200\",\"prediction_length\":\"720\",\"epochs\":\"10\",\"likelihood
\":\"beta\"}"
      },
      "TrainingInstanceCount": {
         "Type": "String",
         "Description": "The number of instances to train the SageMaker model with",
         "Default": "1"
      },
      "TrainingInstanceType": {
         "Type": "String",
         "Description": "The instance type to train the SageMaker model with",
         "Default": "ml.c4.2xlarge"
      },
      "TrainingInstanceVolumeSize": {
         "Type": "String",
         "Description": "The EBS volume size (in GB) to attach to each instance training the SageMaker
model",
         "Default": "20"
      },
      "TrainingMaxRuntimeSeconds": {
         "Type": "String",
         "Description": "The maximum number of seconds to allow a SageMaker training job to execute",
         "Default": "3600"
      },
      "InitialEndpointInstanceCount": {
         "Type": "String",
         "Description": "The number of instances the SageMaker endpoint autoscaling group begins with",
         "Default": "1"
      },
      "EndpointInstanceType": {
         "Type": "String",
         "Description": "The instance type to run the SageMaker endpoint on",
         "Default": "ml.t2.medium"
      },
      "SageMakerInputDataPrefix": {
         "Type": "String",
         "Description": "S3 Prefix where SageMaker training/testing data lives",
         "Default": "sagemaker-input-data"
      },
      "SageMakerOutputModelPrefix": {
         "Type": "String",
         "Description": "S3 Prefix where SageMaker model lives",
         "Default": "models"
      }
   },
   "Resources": {
      "SageMakerTrainingKickoffLambdaRole": {
         "Type": "AWS::IAM::Role",
         "Properties": {
            "AssumeRolePolicyDocument": {
```

```
            "Version": "2012-10-17",
            "Statement": [{
               "Effect": "Allow",
               "Principal": {
                  "Service": ["lambda.amazonaws.com"]
               },
               "Action": ["sts:AssumeRole"]
            }]
         },
         "Path": "/",
         "Policies": [{
            "PolicyName": {
               "Fn::Sub": "SageMakerTrainingKickoffLambdaPolicy"
            },
            "PolicyDocument": {
               "Version": "2012-10-17",
               "Statement": [{
                     "Effect": "Allow",
                     "Action": [
                        "logs:CreateLogGroup",
                        "logs:CreateLogStream",
                        "logs:PutLogEvents"
                     ],
                     "Resource": "*"
                  },
                  {
                     "Effect": "Allow",
                     "Action": [
                        "s3:GetObject"
                     ],
                     "Resource": {
                        "Fn::Sub": "arn:aws:s3:::${DataLakeMLS3Bucket}",
                        "Fn::Sub": "arn:aws:s3:::${DataLakeMLS3Bucket}/*"
                     }
                  },
                  {
                     "Effect": "Allow",
                     "Action": [
                        "s3:ListBucket"
                     ],
                     "Resource": {
                        "Fn::Sub": "arn:aws:s3:::${DataLakeMLS3Bucket}"
                     }
                  },
                  {
                     "Effect": "Allow",
                     "Action": [
                        "sagemaker:CreateTrainingJob"
                     ],
                     "Resource": {
                        "Fn::Sub": "arn:aws:sagemaker:${AWS::Region}:${AWS::AccountId}:training-
job/*training-*"
                     }
                  },
                  {
                     "Effect": "Allow",
```

```
                "Action": [
                    "iam:PassRole"
                ],
                "Resource": {
                    "Fn::GetAtt": ["SageMakerTrainingExecutionRole", "Arn"]
                },
                "Condition": {
                    "StringEquals": {
                        "iam:PassedToService": "sagemaker.amazonaws.com"
                    }
                }
            }
        ]
      }
    }],
    "RoleName": {
      "Fn::Sub": "SageMakerTrainingKickoffLambdaRole"
    }
  }
},
"SageMakerEndpointUpdateLambdaRole": {
  "Type": "AWS::IAM::Role",
  "Properties": {
    "AssumeRolePolicyDocument": {
      "Version": "2012-10-17",
      "Statement": [{
        "Effect": "Allow",
        "Principal": {
          "Service": ["lambda.amazonaws.com"]
        },
        "Action": ["sts:AssumeRole"]
      }]
    },
    "Path": "/",
    "Policies": [{
      "PolicyName": {
        "Fn::Sub": "SageMakerEndpointUpdateLambdaPolicy"
      },
      "PolicyDocument": {
        "Version": "2012-10-17",
        "Statement": [
          {
            "Effect": "Allow",
            "Action": [
              "logs:CreateLogGroup",
              "logs:CreateLogStream",
              "logs:PutLogEvents"
            ],
            "Resource": "*"
          },
          {
            "Effect": "Allow",
            "Action": [
              "sagemaker:CreateModel",
              "sagemaker:DescribeModel"
            ],
```

```
                "Resource": {
                    "Fn::Sub":
"arn:aws:sagemaker:${AWS::Region}:${AWS::AccountId}:model/*model-*"
                }
            },
            {
                "Effect": "Allow",
                "Action": [
                    "sagemaker:CreateEndpointConfig",
                    "sagemaker:DescribeEndpointConfig"
                ],
                "Resource": {
                    "Fn::Sub": "arn:aws:sagemaker:${AWS::Region}:${AWS::AccountId}:endpoint-
config/*endpoint-config-*"
                }
            },
            {
                "Effect": "Allow",
                "Action": [
                    "sagemaker:DescribeEndpoint"
                ],
                "Resource": {
                    "Fn::Sub":
"arn:aws:sagemaker:${AWS::Region}:${AWS::AccountId}:endpoint/*${SageMakerEndpointName}*"
                }
            },
            {
                "Effect": "Allow",
                "Action": [
                    "sagemaker:CreateEndpoint",
                    "sagemaker:UpdateEndpoint"
                ],
                "Resource": [
                    {
                        "Fn::Sub":
"arn:aws:sagemaker:${AWS::Region}:${AWS::AccountId}:endpoint/*${SageMakerEndpointName}*"
                    },
                    {
                        "Fn::Sub": "arn:aws:sagemaker:${AWS::Region}:${AWS::AccountId}:endpoint-
config/*config-*"
                    }
                ]
            },
            {
                "Effect": "Allow",
                "Action": [
                    "iam:PassRole"
                ],
                "Resource": {
                    "Fn::GetAtt": ["SageMakerEndpointRole", "Arn"]
                },
                "Condition": {
                    "StringEquals": {
                        "iam:PassedToService": "sagemaker.amazonaws.com"
                    }
                }
```

```
                }
            ]
        }
      }],
      "RoleName": {
        "Fn::Sub": "SageMakerEndpointUpdateLambdaRole"
      }
    }
  },
  "SageMakerTrainingExecutionRole": {
    "Type": "AWS::IAM::Role",
    "Properties": {
      "AssumeRolePolicyDocument": {
        "Version": "2012-10-17",
        "Statement": [{
          "Effect": "Allow",
          "Principal": {
            "Service": ["sagemaker.amazonaws.com"]
          },
          "Action": ["sts:AssumeRole"]
        }]
      },
      "Path": "/",
      "Policies": [{
        "PolicyName": {
          "Fn::Sub": "SageMakerTrainingExecutionPolicy"
        },
        "PolicyDocument": {
          "Version": "2012-10-17",
          "Statement": [{
              "Effect": "Allow",
              "Action": "ec2:Describe*",
              "Resource": "*"
            },
            {
              "Effect": "Allow",
              "Action": "elasticloadbalancing:Describe*",
              "Resource": "*"
            },
            {
              "Effect": "Allow",
              "Action": [
                "cloudwatch:ListMetrics",
                "cloudwatch:GetMetricStatistics",
                "cloudwatch:Describe*"
              ],
              "Resource": "*"
            },
            {
              "Effect": "Allow",
              "Action": "autoscaling:Describe*",
              "Resource": "*"
            },
            {
              "Effect": "Allow",
              "Action": [
```

```
            "sagemaker:*"
        ],
        "Resource": "*"
    },
    {
        "Effect": "Allow",
        "Action": [
            "ecr:GetAuthorizationToken",
            "ecr:GetDownloadUrlForLayer",
            "ecr:BatchGetImage",
            "ecr:BatchCheckLayerAvailability",
            "cloudwatch:PutMetricData",
            "cloudwatch:PutMetricAlarm",
            "cloudwatch:DescribeAlarms",
            "cloudwatch:DeleteAlarms",
            "application-autoscaling:DeleteScalingPolicy",
            "application-autoscaling:DeleteScheduledAction",
            "application-autoscaling:DeregisterScalableTarget",
            "application-autoscaling:DescribeScalableTargets",
            "application-autoscaling:DescribeScalingActivities",
            "application-autoscaling:DescribeScalingPolicies",
            "application-autoscaling:DescribeScheduledActions",
            "application-autoscaling:PutScalingPolicy",
            "application-autoscaling:PutScheduledAction",
            "application-autoscaling:RegisterScalableTarget",
            "logs:CreateLogGroup",
            "logs:CreateLogStream",
            "logs:DescribeLogStreams",
            "logs:GetLogEvents",
            "logs:PutLogEvents"
        ],
        "Resource": "*"
    },
    {
        "Effect": "Allow",
        "Action": [
            "s3:GetObject"
        ],
        "Resource": {
            "Fn::Sub": "arn:aws:s3:::${DataLakeMLS3Bucket}/*"
        }
    },
    {
        "Effect": "Allow",
        "Action": [
            "s3:ListBucket"
        ],
        "Resource": {
            "Fn::Sub": "arn:aws:s3:::${DataLakeMLS3Bucket}"
        }
    },
    {
        "Effect": "Allow",
        "Action": [
            "s3:Put*"
        ],
```

```
                        "Resource": {
                           "Fn::Sub":
"arn:aws:s3:::${DataLakeMLModelS3Bucket}/${SageMakerOutputModelPrefix}/*"
                        }
                     }
                  ]
               }
            }],
            "RoleName": {
               "Fn::Sub": "SageMakerTrainingExecutionRole"
            }
         }
      },
      "SageMakerEndpointRole": {
         "Type": "AWS::IAM::Role",
         "Properties": {
            "AssumeRolePolicyDocument": {
               "Version": "2012-10-17",
               "Statement": [{
                  "Effect": "Allow",
                  "Principal": {
                     "Service": ["sagemaker.amazonaws.com"]
                  },
                  "Action": ["sts:AssumeRole"]
               }]
            },
            "Path": "/",
            "Policies": [{
               "PolicyName": {
                  "Fn::Sub": "SageMakerEndpointRole"
               },
               "PolicyDocument": {
                  "Version": "2012-10-17",
                  "Statement": [{
                        "Effect": "Allow",
                        "Action": "ec2:Describe*",
                        "Resource": "*"
                     },
                     {
                        "Effect": "Allow",
                        "Action": "elasticloadbalancing:Describe*",
                        "Resource": "*"
                     },
                     {
                        "Effect": "Allow",
                        "Action": [
                           "cloudwatch:ListMetrics",
                           "cloudwatch:GetMetricStatistics",
                           "cloudwatch:Describe*"
                        ],
                        "Resource": "*"
                     },
                     {
                        "Effect": "Allow",
                        "Action": "autoscaling:Describe*",
                        "Resource": "*"
```

```
                    },
                    {
                       "Effect": "Allow",
                       "Action": [
                          "sagemaker:*"
                       ],
                       "Resource": "*"
                    },
                    {
                       "Effect": "Allow",
                       "Action": [
                          "ecr:GetAuthorizationToken",
                          "ecr:GetDownloadUrlForLayer",
                          "ecr:BatchGetImage",
                          "ecr:BatchCheckLayerAvailability",
                          "cloudwatch:PutMetricData",
                          "cloudwatch:PutMetricAlarm",
                          "cloudwatch:DescribeAlarms",
                          "cloudwatch:DeleteAlarms",
                          "application-autoscaling:DeleteScalingPolicy",
                          "application-autoscaling:DeleteScheduledAction",
                          "application-autoscaling:DeregisterScalableTarget",
                          "application-autoscaling:DescribeScalableTargets",
                          "application-autoscaling:DescribeScalingActivities",
                          "application-autoscaling:DescribeScalingPolicies",
                          "application-autoscaling:DescribeScheduledActions",
                          "application-autoscaling:PutScalingPolicy",
                          "application-autoscaling:PutScheduledAction",
                          "application-autoscaling:RegisterScalableTarget",
                          "logs:CreateLogGroup",
                          "logs:CreateLogStream",
                          "logs:DescribeLogStreams",
                          "logs:GetLogEvents",
                          "logs:PutLogEvents"
                       ],
                       "Resource": "*"
                    },
                    {
                       "Effect": "Allow",
                       "Action": [
                          "s3:GetObject"
                       ],
                       "Resource": {
                          "Fn::Sub":
"arn:aws:s3:::${DataLakeMLModelS3Bucket}/${SageMakerOutputModelPrefix}/*"
                       }
                    }
                 ]
              }
           }],
           "RoleName": {
              "Fn::Sub": "SageMakerEndpointRole"
           }
        }
     },
     "S3BucketInvokeEndpointUpdateLambdaPerm": {
```

```
      "Type": "AWS::Lambda::Permission",
      "Properties": {
        "Action": "lambda:InvokeFunction",
        "FunctionName": {
          "Ref": "SageMakerEndpointUpdateLambda"
        },
        "Principal": "s3.amazonaws.com",
        "SourceAccount": {
          "Ref": "AWS::AccountId"
        },
        "SourceArn": {
          "Fn::Sub": "arn:aws:s3:::${DataLakeMLModelS3Bucket}"
        }

      }
    },
    "SageMakerTrainingKickoffLambda": {
      "Type": "AWS::Lambda::Function",
      "Properties": {
        "FunctionName": {
          "Fn::Sub": "SageMakerTrainingKickoffLambda"
        },
        "Handler": "index.handler",
        "Role": {
          "Fn::GetAtt": ["SageMakerTrainingKickoffLambdaRole", "Arn"]
        },
        "Code": {
          "S3Bucket": {
            "Ref": "LambdaCodeS3Bucket"
          },
          "S3Key": {
            "Fn::Sub":
"${LambdaCodeS3KeyPrefix}/${SageMakerTrainingKickoffLambdaCodeLocation}"
          }
        },
        "Runtime": "nodejs8.10",
        "Timeout": 300,
        "Environment": {
          "Variables": {
            "OutputBucketName": {
              "Fn::Sub": "${DataLakeMLModelS3Bucket}"
            },
            "SageMakerOutputModelPrefix": {
              "Ref": "SageMakerOutputModelPrefix"
            },
            "HyperParameters": {
              "Ref": "HyperParameters"
            },
            "TrainingInstanceCount": {
              "Ref": "TrainingInstanceCount"
            },
            "TrainingInstanceType": {
              "Ref": "TrainingInstanceType"
            },
            "TrainingInstanceVolumeSize": {
              "Ref": "TrainingInstanceVolumeSize"
```

```
                },
                "TrainingMaxRuntimeSeconds": {
                   "Ref": "TrainingMaxRuntimeSeconds"
                },
                "TrainingImage": "522234722520.dkr.ecr.us-east-1.amazonaws.com/forecasting-
deepar:latest",
                "TrainingRoleArn": {
                   "Fn::GetAtt": ["SageMakerTrainingExecutionRole", "Arn"]
                },
                "DataLakeMLS3Bucket": {
                   "Ref": "DataLakeMLS3Bucket"
                }
             }
          }
       }
    },
    "SageMakerEndpointUpdateLambda": {
       "Type": "AWS::Lambda::Function",
       "Properties": {
          "FunctionName": {
             "Fn::Sub": "SageMakerEndpointUpdateLambda"
          },
          "Handler": "index.handler",
          "Role": {
             "Fn::GetAtt": ["SageMakerEndpointUpdateLambdaRole", "Arn"]
          },
          "Code": {
             "S3Bucket": {
                "Ref": "LambdaCodeS3Bucket"
             },
             "S3Key": {
                "Fn::Sub":
"${LambdaCodeS3KeyPrefix}/${SageMakerEndpointUpdateLambdaCodeLocation}"
             }
          },
          "Runtime": "nodejs8.10",
          "Timeout": 300,
          "Environment": {
             "Variables": {
                "TrainingImage": "522234722520.dkr.ecr.us-east-1.amazonaws.com/forecasting-
deepar:latest",
                "ModelRoleArn": {
                   "Fn::GetAtt": ["SageMakerEndpointRole", "Arn"]
                },
                "InitialEndpointInstanceCount": {
                   "Ref": "InitialEndpointInstanceCount"
                },
                "EndpointInstanceType": {
                   "Ref": "EndpointInstanceType"
                },
                "SageMakerEndpointName": {
                   "Fn::Sub": "${SageMakerEndpointName}"
                }
             }
          }
       }
```

```
      }
   },
   "Outputs":{
      "SageMakerEndpointUpdateLambdaRef" : {
         "Value" :  { "Fn::GetAtt": ["SageMakerEndpointUpdateLambda", "Arn"] },
         "Export" : { "Name" : "SageMakerEndpointUpdateLambdaRef"}
      },
      "SageMakerTrainingLambdaRef" : {
         "Value" :  { "Fn::GetAtt": ["SageMakerTrainingKickoffLambda", "Arn"] },
         "Export" : { "Name" : "SageMakerTrainingKickoffLambda"}
      }
   }
}
"FILENAME: C:\dev\c3\rawRepos\machine-learning\tools\package-all-lambda-code.sh"

./tools/package-lambda-code.sh ingest-lambda
./tools/package-lambda-code.sh model-lambda
./tools/package-lambda-code.sh enhance-lambda
./tools/package-lambda-code.sh transform-lambda

rm -rf functions/source/ingest-lambda/node_modules
rm -rf functions/source/model-lambda/node_modules
rm -rf functions/source/enhance-lambda/node_modules
rm -rf functions/source/transform-lambda/node_modules
"FILENAME: C:\dev\c3\rawRepos\machine-learning\tools\package-lambda-code.sh"

#!/bin/sh

lambda=$1

baseSource="functions/source"
source="$baseSource/$lambda"
filename="$lambda.zip"
package_folder="functions/packages"
destination="$package_folder/$lambda/$filename"

cd "$baseSource/aws-helpers"
rm -rf node_modules
npm install --only=prod
cd "../../.."

cd $source

s3Key="$lambda/$filename"

# Clearing out node_modules so we can build a smaller package
rm -rf node_modules
npm install --only=prod
zip -r -q $lambda .

cd "../../.."

mkdir -p $package_folder/$lambda/
mv "$source/$filename" $destination

"FILENAME: C:\dev\c3\rawRepos\machine-learning\tools\upload-lambda-code-s3.sh"
```

```
#!/bin/sh

#Enable the following line if running in git bash on Windows
alias aws="winpty /C/Program\ Files/Amazon/AWSCLI/bin/aws.cmd"

lambda=$1

filename="$lambda.zip"
package_folder="functions/packages"
packageLocation="$package_folder/$lambda/$filename"

aws s3api put-object --bucket lightbulb-lambda-code --key ml-artifacts/$filename --body $packageLocation
```