# SIT311SoftwareEngineering 3: Designing User-Centred IoT Applications

## Graded Task 6.3: **Software Engineering Project Assignment (40%)**

The assignment is a **<u>group</u> assignment** (groups of 4 students; 1-2 student must be cloud student)**.** Allocation of students to teams was proposed in the email.

The assignment has a total of **100 marks** (it is **worth 40%** of the unit's final mark).

### <u>Due Dates & Interviews:</u>

- The submission is due on **Friday Week** 10 **by 11:59pm.**
- **Demos/interviews** will be held in the Tutorials **in Week 11**. Without an interview, the assignment mark will be zero.

### <u>ParkSmart (Smart Parking Recommender): An Overview</u>

In this assignment, you will implement a web application that consumes real-time sensory data coming from parking sensors and provides a dashboard to monitor current status of on-street parking in the City of Melbourne. Furthermore, this application assists drivers in finding carparks by providing related information, that is, working on a smartphone in a web-browser.

### <u>Task 1: Invoking public web APIs (10 marks):</u>

A weather API will be used to provide the weather information (i.e. temperature and weather condition like rain, shower, cloudy, sunny) for a location. You need to call the right API methods and pass the right parameters, and then parse the JSON response to retrieve the current temperature. **(10 marks)**

The current temperature value will be displayed in the Dashboard when drivers search for a carpark.

These are some examples of weather APIs below, but you can find and use any other weather API. Make sure you use the free access.

Open Weather Map <u>http://www.openweathermap.org/api</u>

World Weather Online’s weather http://developer.worldweatheronline.com/api/

Dark Sky API https://darksky.net/dev/

## Task 2 Parking Information and Status (20 marks):

City of Melbourne has around 3000 in-ground on-street parking sensors which indicate if a parking bay is occupied or not. This information is available online through City of Melbourne's Open Data Platform[1]. Moreover, the Open Data portal also offers data about all parking signs and restrictions in Melbourne CBD. The Open Data portal provides programmatic access to these datasets including the ability to filter, query, and aggregate data.

Students should familiarise themselve with these APIs, and **develop a search method** that **accepts geo-coordinates** of a location (latitude and longitude) and **returns live information** about **up to 10** parking locations **within the radius of 500 meters** from the provided location. This method should consume the APIs provided in Table 1, and produce the following information for each parking bay **in JSON format**:

- **Bay ID**: The unique ID of the parking bay where the parking sensor is located
- **Parking Status**: The status will either display: Occupied – A car is present in the parking bay at that time. Unoccupied – The parking bay is available at that time.
- **Location**: The latitude and longitude of the parking bay
- **Restrictions**
    - **Free**: Indicates if the parking bay is free or metered.
    - **Duration**: The time that a vehicle can park in the bay (assuming they can legally park there).
    - **Effective on public holidays**: Does this restriction apply on public holidays ?
    - **Time**: The time each day when the restriction applies (i.e. start time and end time)
    - **Days**: An array of days where the restriction applies.

[1] https://data.melbourne.vic.gov.au/

*Table 1 - Open Portal APIs*

| | |
|---|---|
| On-street Parking Bay Sensors | https://data.melbourne.vic.gov.au/Transport-Movement/On-street-Parking-Bay-Sensors/vh2v-4nfs |
| On-street Car Park Bay Restrictions | https://data.melbourne.vic.gov.au/Transport-Movement/On-street-Car-Park-Bay-Restrictions/ntht-5rk7 |

The following text box represent an example of search method output. Please note in days, each day is mapped to an integer form 0(Sunday) to 6(Saturday).

```
{
  "bayID": "1005",
  "location": {
    "latitude": "-37.813104376935705",
    "longitude": "144.9625311043034"
  },
  "status": "Unoccupied",
  "restrictions": [
    {
      "isFree": false,
      "duration": {"normal": "30",disablity": "60"},
      "effectiveonph": "0",
      "time": {"start": "07:30:00","end": "18:30:00"},
      "days": [1, 2, 3, 4, 5, 6]
    },
    {
      "isFree": false,
      "duration": {"normal": "120","disablity": "240"},
      "effectiveonph": "0",
      "time": { "start": "18:30:00", "end": "20:30:00" },
      "days": [1, 2, 3, 4, 5, 6]
    },
    {
      "isFree": true,
      "duration": {"normal": "60","disablity": "120"},
      "effectiveonph": "0",
      "time": {"start": "07:30:00","end": "18:30:00"},
      "days": [0]
    }
  ]
}
```

*Figure 1 Parking information example*

**Task 3 AWS IoT Core devices (20 marks):**

a) You should create 10 devices/Things in AWS IoT Core named as parking1, parking2… parking10. Whenever you start the ParkSmart application, you should connect to these 10 devices using AWS IoT Node JS SDK. **(5 marks)**

b) When a user searches for a location and you have retrieved parking bays (up to 10) using the search method, **for each retrieved parking**, you need to send a **MQTT message** to AWS IoT Core that contains the **bay_id**, **status**, and **type of restriction** for the **current date and time (5 marks)**. For example, if the current time is Monday 10am, for the parking bay shown in Figure 1, the MQTT message should be as below:

```
{
  "bayID": "1005",
  "status": "Unoccupied",
  "restriction_ duration": 30
}
```

c) Every two minutes, the application should check the status of parking bay (by invoking the correct Open portal API) and send a MQTT message to AWS IoT core if the status changes. **(5 marks)**

d) You should define a **rule** in **AWS rule engine** to redirect **all the messages** coming from parking things to **IoT Events**. **(5 marks)**

**Task 4 AWS IoT Events (20 marks):**

a) You should build a **Detector model** in **AWS IoT Events** that **monitors** the **incoming messages** coming from **IoT Core** and **detects the current state** of each bay **(10 marks)**. The possible states that the detector should detects are:

1. **Available**: When the parking bay is not occupied.
2. **Occupied**: When the parking bay is occupied and not 3 and 4.
3. **Likely to be available**: When the car that currently occupied the parking bay must leave in **5 minutes** to **avoid overstaying**. For example, based on the type of restriction,

if the maximum duration that a car can **stay legally** in the **parking bay is 30 minutes**, the parking bay state should be **Likely to be available** if a car was **parked** in the given parking bay for **25+ minutes**.

4. **Overstayed**: If **a car is parked** in the parking bay for **longer than legal duration.**

b) The Detector should **issue a MQTT message to a correct IoT Topic** whenever **the parking bay state changes. (3 marks)**

c) The IoT devices created in task 3 should subscribe to these topics and receive messages when the parking bay state changes. **(3 marks)**

d) For **testing purposes**, you should design your detector in a way to be possible to change the state of parking bays by sending certain messages. For example, no matter of what is the current state of a given parking bay, by sending a message where status is "**goToOverstayed**", the state should be changed to ***Overstayed*** immediately. **(4 marks)**

**Hint**: For doing so, you should add an additional transition rule from each state to all the other states.

## Task 5 ParkSmart Dashboard (30 marks):

The dashboard should contain the following components:

**a) Search bar (6 marks):** The dashboard should provide a **location search** bar with **autocomplete** feature that allows a user to search for parking near a location. Moreover, the search bar should convert the address to geo-coordinates. For this task, you can use any third party library such as:

- https://community.algolia.com/places/
- https://github.com/hibiken/react-places-autocomplete

After a user presses the search button, the application should call the **search parking method.** Moreover, it should fetch the weather in the given geo-coordinates.

**b) Weather Widget (4 marks):** The dashboard should display the weather condition in the location searched by user.

c) **Map (10 marks)**: The dashboard should display the **location that user has entered in the search bar**, and also list all the **parking bays** near the provided location (**outcome of search parking method**). **(5 marks)**

   - The **marker colour** will be based on the parking bay state as below: **(3 marks)**
     - **Available**: Dark Green
     - **Likely to be available**: Light Green
     - **Occupied**: Blue
     - **Overstayed**: Red
   - When you **tap on a marker**, the **parking information** should be displayed in **Parking Details widget in the dashboard. (2 marks)**
   - To complete this task, you can use any Map library such as:
     - https://react-leaflet.js.org/
     - https://github.com/mariusandra/pigeon-maps
     - https://github.com/google-map-react/google-map-react

d) **Parking Details (6 marks):** If users **tap on a parking bay marker,** the dashboard should display all the information related to that parking. This widget should also have **4 buttons** (**available**, **occupied**, **likely to be available**, and **overstayed**), that allow users to **change the state** of the parking bay.

e) **Activity List (4 marks)**: The Dashboard should display the last 50 messages it sent to AWS IoT Core or received from AWS IoT Events **(2 marks)**. It should also provide an option to save the logs as text file **(2 marks)**.

## Additional Marking Criteria

**Only completing a task does not yield a HD mark.** There are 6 other criteria for marking each task. For a full mark of any task, all the following criteria should be achieved:

1. **High quality of design and programming/coding (e.g. high cohesion, loose coupling, detailed comments, separation of concerns, bug free, solid exception handling, following coding standards, proper and meaningful naming of variables and methods, efficient use of variables, etc)**
2. **Handling all the exceptions and user data entry validation**
3. **Excellent GUI interface and layout and navigation**
4. **Evidence of originality and creativity of student (going beyond the assignment specification),**
5. **Full functionality of all the operations (during the interview),**
6. **And student's deep understanding of their code and the program logic**

**Not meeting any of these criteria can result in mark deduction**

**Late Submission:**

Late Assignments or extensions will not be accepted unless you submit a special consideration form and provide valid documentation such as a medical certificate prior to the submission deadline (NOT after). Otherwise, there will be **5% penalty per day including the weekends**.