AIRCO

TECHNICAL REPORT

Version 1.0

Start Date: **10/03 /2022**

Authors : **Anson Go Guang Ping, s3767707**
Thomas Jukes, s3783616
Manav Shrivastav, s3785704
Duc Tran, s3752703
Jason Song, s3744335

DOCUMENT CONTROL

Version #	Implemented By	Implementation Date	Reviewed By	Approval Date	Reason
1	Development Team	30/05/2022	Dhirendra Singh	31/05/2022	Submission
2					
3					
4					
5					

TABLE OF CONTENTS

1 EXECUTIVE SUMMARY

Our product AirCo has so far satisfied many of the requirements of our client. We were successful in all but one of our goals, namely the potential to include user input to receive a tailored response from the system.

Despite that, we have created a product that works, and can be replicated on any number of AWAIR compatible sensors, and Linux capable systems. Though the setup requires a degree of technical knowledge, our Readme and documentation is extensive. It can connect to IAQ sensors, enable users to view the information and set the update intervals from a variety of choices; as well as viewing different parameters for the sensors to view.

When certain parameters exceed safe levels, such as CO2, VOCs, or PM2.5s, a warning and general suggestions are presented to the user, protecting their health.

To do this, we used the Java to code all aspects of the back end, along with JavaScript and HTML for the front-end. We used the Spring-Boot framework and REST API to interface between the front and the back end. We used React to build our User Interface, and Chart.JS to design and represent the data the sensors pick up. We also made use of the AWAIR APIs to interact with the sensors and access their data.

We used IntelliJ and Visual Studio Code as IDEs to develop the product.

In summation, what we've built can be used to achieve the stated goal, which is create a service for people to use to access more detailed, granular data for people with compatible IAQ sensors.

2 INTRODUCTION

Digital technologies such as smartphone apps that access real-time data to improve health outcomes are becoming more prevalent. For instance, a near real time smartphone app "AirRater" (https://airrater.org/) provides air quality information to vulnerable individuals such as asthmatics and the general public. AirRater can track user symptoms, monitor air quality conditions, and help reduce personal exposure to air pollution - such as smoke from bushfires. This App has helped users avoid exposure and effects of smoke by advising them on precautionary measures, such as rescheduling or planning outdoor activities, and informing decisions on the use of asthma medication.

The aim of this project is to develop an indoor air quality (IAQ) App that accesses real time data and key user information to help create healthier indoor environments. In the wake of recent bushfires, and the ever-growing pollution issues, an easy-to-use product like AirCo would be useful for anyone looking to monitor the quality of their indoor environments.
This app will:

- Communicate with IAQ sensors (e.g., https://www.getawair.com/products/omni) that are currently in use in our research.
- Enable users to view sensor data in a user-friendly way, with granular updates allowing them to see the information in real time.

- Provide information about how to improve the air quality in the indoor environment (e.g., increasing ventilation, switching a product, changing an activity, or installing a technology such as an air filter).

- Enable users to enter key information about
  (i) their indoor environment,
  (ii) possible sources of air pollution,
  (iii) and any health symptoms that are exacerbated by exposure to poor IAQ.

Our stakeholders are us, the Developing team; and anyone who cares about monitoring their indoor environment, their air quality, and their health.

3 REQUIREMENTS

USE CASES

We have decided to not include use case diagrams as we are able to articulate our functional requirements without them.

FUNCTIONAL REQUIREMENTS

Airco is an online indoor air quality web application; it includes multiple modules providing APIs to allow integration into the overall application.

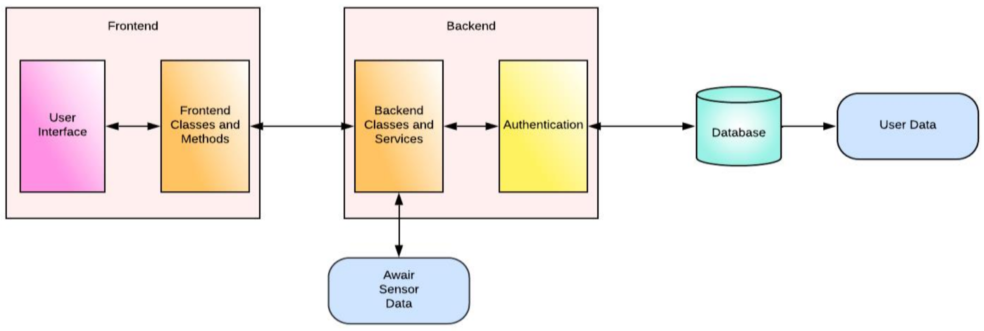
Req. ID	Description	Priority
R1	The system should display a home page with login functionality. The program must authenticate and authorise users upon login.	2
R2	The system should display a home page with register functionality. The program must authenticate user information in order to meet system sign up requirements. User ID must be unique, and password must be at least 6 characters.	2
R3	User can login and upon successful login, the user is able to choose a sensor device corresponding to their current location, and check for the indoor air quality data.	1
R4	User can read IAQ score and other pollutant levels at their dashboard. Colours are implemented to aid the identification of air quality	2

	conditions.	
R5	User can receive notifications about IAQ readings and other pollutant levels and their relative consequences and potential improvements.	1
R6	The app can update the IAQ data itself every one minute.	3
R7	User can read the complete data guide on how the app grades the pollutant levels and IAQ score.	1
R8	User can gain insight of the data in a certain time interval. The data should be in any sort of visual presentations.	1
R9	User can export current dataset to various file formats to allow further analysis of raw data or visualisation.	1
R10	The app can allow user feedback to be provided and stored as data points for future reference. This information was to be maintained and evaluated by the app, including qualities like a user's health symptoms, possible pollutants such as open paint cans, etc. and then return advice to the user.	4

NON-FUNCTIONAL REQUIREMENTS SPECIFICATION

Our main objective was to produce the functional requirements in a way that was maintainable with little developer knowledge and high levels of usability.

4 ARCHITECTURE



We implemented an N-tier architecture style for our web application, consisting of two microservices on the backend and React for the user interface.

Users required storing, retrieving and maintaining a collection of users, and this architecture provided a good distinction between the physical tiers and logical layers of the application. This allows the backend services to be deployed on a separate machine if required and makes it easier to manage dependencies on the front end and database. It also means if this application were to be deployed to a cloud service, managing and migrating it would require minimal refactoring.

5 TECHNICAL FRAMEWORK

5.1 BACK END

5.1.1 Environment

IntelliJ

IntelliJ IDEA is an integrated development environment (IDE) written in Java for developing computer software written in Java, Kotlin, Groovy, and other JAR based languages. It is developed by JetBrains (formerly known as IntelliJ) and is available as an Apache 2 Licensed community edition, and in a proprietary commercial edition. Both can be used for commercial development.

5.1.2 Language

Java

Java is a high-level, class-based, object-oriented programming language that is designed to have as few implementation dependencies as possible. It is a general-purpose programming language intended to let programmers write once, run anywhere meaning that compiled Java code can run on all platforms that support Java without the need to recompile. Java applications are typically compiled to bytecode that can run on any Java virtual machine (JVM) regardless of the underlying computer architecture.

The entire back end is written using Java and its many libraries

5.1.3 Framework

Spring boot

Spring Boot is an open source, microservice-based Java web framework. The Spring Boot framework creates a fully production-ready environment that is completely configurable using its prebuilt code within its codebase. The microservice architecture provides developers with a fully enclosed application, including embedded application servers.

In this project, we use spring boot to create microservice application, access and modify the database

REST API
A REST API (also known as RESTful API) is an application programming interface (API or web API) that conforms to the constraints of REST architectural style and allows for interaction with RESTful web services. REST stands for representational state transfer and was created by computer scientist Roy Fielding.

In this project, we use REST API to receive and response to Front-end request.

Awair API
Awair API is a proprietary library which allow user to access sensor data.

5.2 FRONTEND

5.2.1 Environment

Visual Studio Code
Visual Studio Code is a lightweight but powerful source code editor which runs on your desktop and is available for Windows, macOS and Linux. It comes with built-in support for JavaScript, TypeScript and Node.js and has a rich ecosystem of extensions for other languages (such as C++, C#, Java, Python, PHP, Go) and runtimes (such as .NET and Unity) (VSCode 2022, para.1).

In this project, VSCode is the main IDE used for frontend code editing. It is chosen as it supports language like JavaScript and JavaScript libraries like React. Besides that, Visual Studio Code includes built-in JavaScript IntelliSense, debugging, formatting, code navigation, refactoring's, and many other advanced language features (VSCode 2022). Nevertheless, it has a lot of extensions which makes code editing easier and more efficient. Here is a list of extensions that were used within development:

i. Prettier - Code formatter v9.5.0
Prettier is an opinionated code formatter. It enforces a consistent style by parsing your code and re-printing it with its own rules that take the maximum line length into account, wrapping code when necessary (VSCode 2022).

ii. Path Intellisense v2.8.0
VSCode plugin that autocompletes filenames (VSCode 2022).

iii. Node.js Modules Intellisense v1.5.0
Visual Studio Code plugin that autocompletes JavaScript / TypeScript modules in import statements. This plugin was inspired by npm Intellisense and AutoFileName (VSCode 2022).

iv. ES7 React/Redux/GraphQL/React-Native snippets v1.9.3
This extension provides you JavaScript and React/Redux snippets in ES7 with Babel plugin features for VS Code (VSCode 2022).

v. [Deprecated] Bracket Pair Colorizer 2 v0.2.4
This extension allows matching brackets to be identified with colours. The user can define which tokens to match, and which colours to use (VSCode 2022).

5.2.2 Language

JavaScript

JavaScript is a crucial programming language for any web developer that specialize in front-end, back-end, or full-stack development. JavaScript is used as the main programming language to create dynamic and interactive web pages that have become the standard user experience.

JavaScript allows us to implement features like:

- Showing and hiding menus or information
- Adding hover effects
- Creating drop down and hamburger-style menus

HTML

HTML is the standard markup language for creating Web pages and describes the structure of a Web page. It consists of a series of elements which tell the browser how to display the content.

All our pages are created using HTML. One example is HTML form is used to collect user input for login and registration purpose. The user input is then sent to backend by calling their responding API for processing.

CSS

Cascading Style Sheets (CSS) is a stylesheet language used to describe the presentation of a document written in HTML. CSS describes how elements should be rendered on screen, on paper, in speech, or on other media (MDN web docs, 2022).

In this project, CSS is used to style and layout web pages — for example, to alter the font, colour, size, and spacing of our content such as grid layout and margin.

5.2.3 Framework

React

React is a free and open-source front-end JavaScript library for building user interfaces based on UI components. It is maintained by Meta and a community of individual developers and companies makes it easier to create interactive UIs (Wikipedia, 2022).

We decided to use React as our JavaScript framework as we all has some experience in React prior to this project. It helps us design simple views for each state in our application and React will efficiently update and render just the right components when data changes. Besides that, encapsulated components are built to manage their own state which are further composed to make complex UIs.

Redux

Redux is an open-source JavaScript library which serve as a predictable state container for JavaScript apps. The whole global state of our app is stored in an object tree inside a single store. We create an action, an object describing what happened which is the only way to change the state tree and dispatch it to the store. We also write pure reducer functions that calculate a new state based on the old state and the action to specify how state gets updated in response to an action. (Redux, 2022).

In a typical Redux app, there is just a single store with a single root reducing function (Redux, 2022). As our app grows, we split the root reducer into smaller reducers independently operating on the different parts of the state tree. This is exactly like how there is just one root component in a React app, but it is composed out of many small components.

For Redux specifically, "thunks" are a pattern of writing functions with logic inside that can interact with a Redux store's dispatch and getState methods. Thunks are commonly used for data fetching and are the standard approach for writing async logic in Redux apps (Redux, 2022).

Axios

Axios is a promise-based HTTP Client for node.js and the browser (Axios, 2022). Our projects on the web need to interface with a REST API at some stage in the development. Axios is therefore used to take advantage of JavaScript's async and await for more readable asynchronous code.

React-Bootstrap

React-Bootstrap is a complete re-implementation of the Bootstrap components using React. It has no dependency on either bootstrap.js or jQuery. If you have React setup and React-Bootstrap installed, you have everything you need (React-bootstrap).

React-bootstrap is used in our project to import react components like alert, button or form that are used for login and register page.

ApexChart

ApexCharts is a modern charting library that helps developers to create beautiful and interactive visualizations for web pages. It is an open-source project licensed under MIT and is free to use in commercial applications (APEXCHART, 2022).

One of our project specifications is to implement chart to display the trends of AQI data within a duration. A line chart is essential to meet this requirement while ApexCharts provide an easy way to create a line chart with modern design. It has built-in functions for data export in csv or png format, legends and tooltips which makes development much easier.

6 IMPLEMENTATION

6.1 FRONTEND:

ReactJS

i. Getting Started

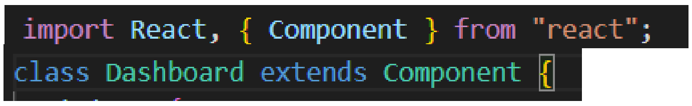
Run VSCode and locate to the directory you want to start. Run:

```
>npx create-react-app frontend
>cd frontend
>npm start
```

Create React App doesn't handle backend logic or databases; it just creates a frontend build pipeline, so you can use it with any backend you want (React, 2022).

ii. Components and Props

Conceptually, components are like JavaScript functions. They accept arbitrary inputs (called "props") and return React elements describing what should appear on the screen (React, 2022). First, we define components as classes. In this project each class represents a page or a component in a page.



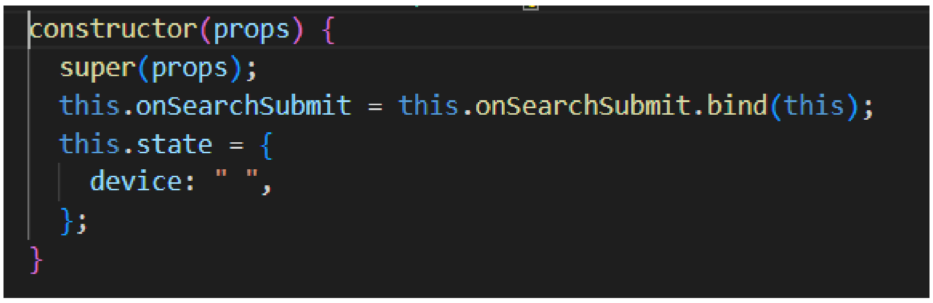
```
import React, { Component } from "react";
class Dashboard extends Component {
```

The constructor for a React component is called before it is mounted. When implementing the constructor for a React.Component subclass, you should call super(props) before any other statement. Otherwise, this.props will be undefined in the constructor, which can lead to bugs.

Typically, in React constructors are only used for two purposes:

- Initializing local state by assigning an object to this.state.
- Binding event handler methods to an instance.

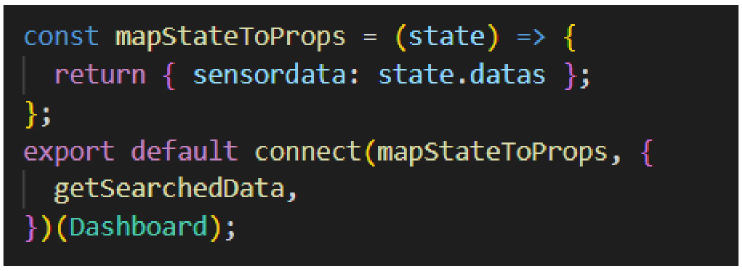
The state contains data specific to this component that may change over time. The state is user-defined, and it should be a plain JavaScript object (React, 2022). In this case, we are initializing a state named device in the constructor which is then updated later when user want to choose their location to render the indoor AQI values. The event handler function onSearchSubmit is triggered by user if they clicked on the submit button beside the dropdown bar.



```
constructor(props) {
  super(props);
  this.onSearchSubmit = this.onSearchSubmit.bind(this);
  this.state = {
    device: " ",
  };
}
```

iii. connect() and mapStateToProps() from Redux

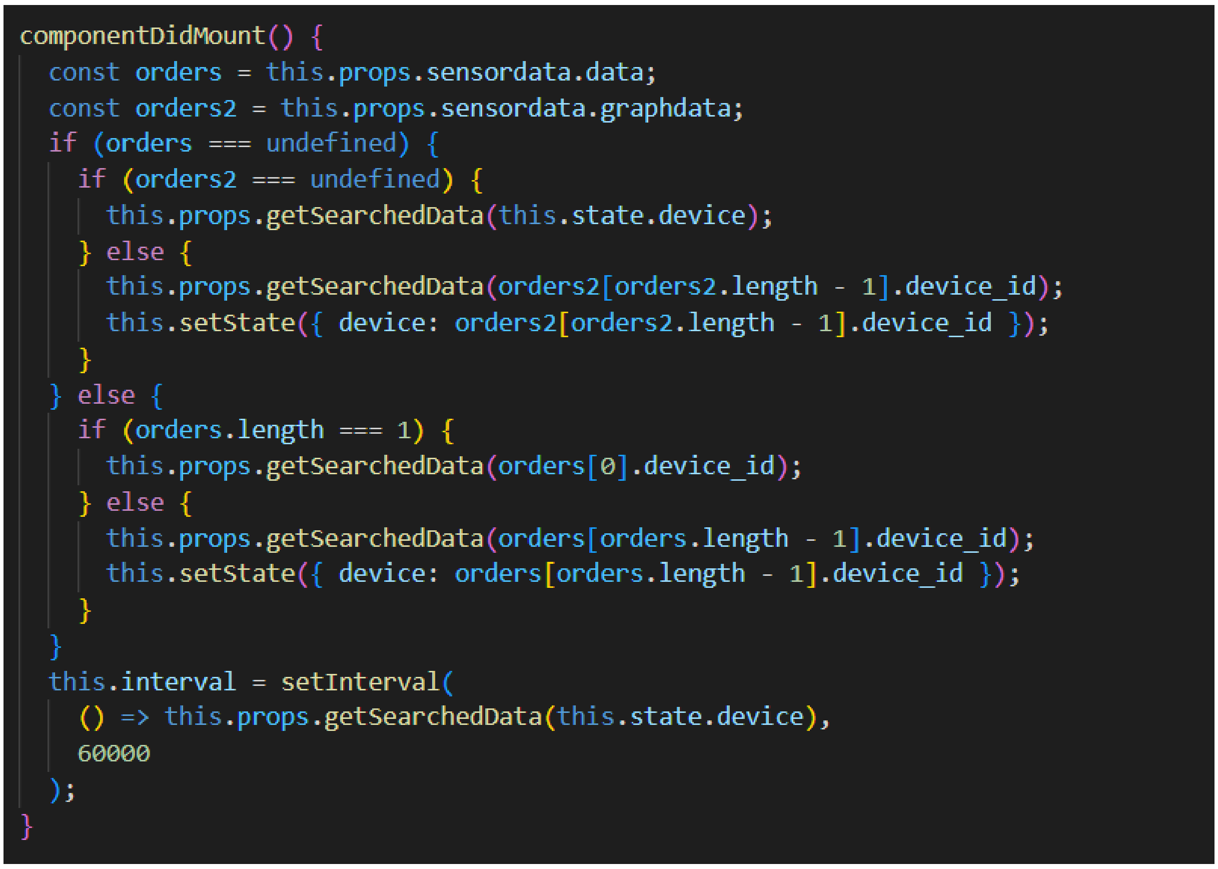
The connect() function connects a React component to a Redux store. It provides Dashboard with the data collected from the sensor (an array retrieved from json file after calling an API from the backend) it needs from the store, and the functions it can use to dispatch actions to the store. The mapStateToProps function is declared as taking one parameter (state.datas), it will be called whenever the store state changes, and given the store state as the only parameter.



```
const mapStateToProps = (state) => {
  return { sensordata: state.datas };
};
export default connect(mapStateToProps, {
  getSearchedData,
})(Dashboard);
```

iv. componentDidMount()

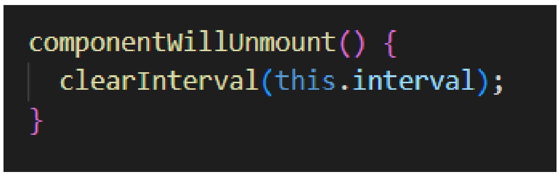
componentDidMount() is invoked immediately after a component is mounted (inserted into the tree) (React, 2022). It is used to set the state “device” which is used to pass as a parameter to call the getSearchedData() function which send an API to retrieve sensor data whenever user enters the dashboard page. When user first enters dashboard page after their login attempt, the device is undefined since user has not set the state through event handling. However, after user set this.state.device, it can be pass between page and the device information will not loss after user comes back from other page. As a bonus, we think that it could be amazing that our dashboard can update pulls the data from the sensor every one minute without user’s concern and displays the changes. This feature is implemented by running the setInterval() method that calls the getSearchedData() function every one minute. Whenever the properties is updated, the component is mounted into the tree and componentDidMount() is invoked immediately.



```
componentDidMount() {
  const orders = this.props.sensordata.data;
  const orders2 = this.props.sensordata.graphdata;
  if (orders === undefined) {
    if (orders2 === undefined) {
      this.props.getSearchedData(this.state.device);
    } else {
      this.props.getSearchedData(orders2[orders2.length - 1].device_id);
      this.setState({ device: orders2[orders2.length - 1].device_id });
    }
  } else {
    if (orders.length === 1) {
      this.props.getSearchedData(orders[0].device_id);
    } else {
      this.props.getSearchedData(orders[orders.length - 1].device_id);
      this.setState({ device: orders[orders.length - 1].device_id });
    }
  }
  this.interval = setInterval(
    () => this.props.getSearchedData(this.state.device),
    60000
  );
}
```

v. componentWillUnmount()

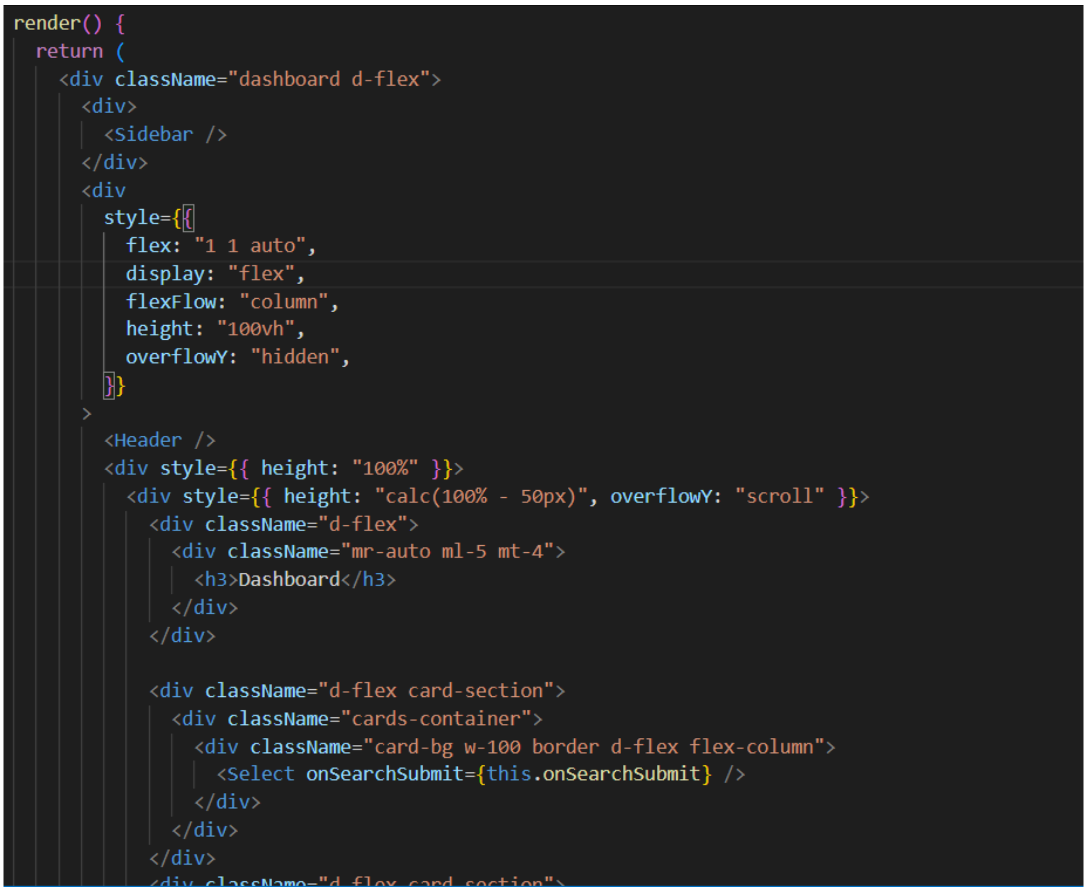
componentWillUnmount() is invoked immediately before a component is unmounted and destroyed (React, 2022). It is used to a cleanup of this.interval that was created in componentDidMount() whenever user moves to another page.



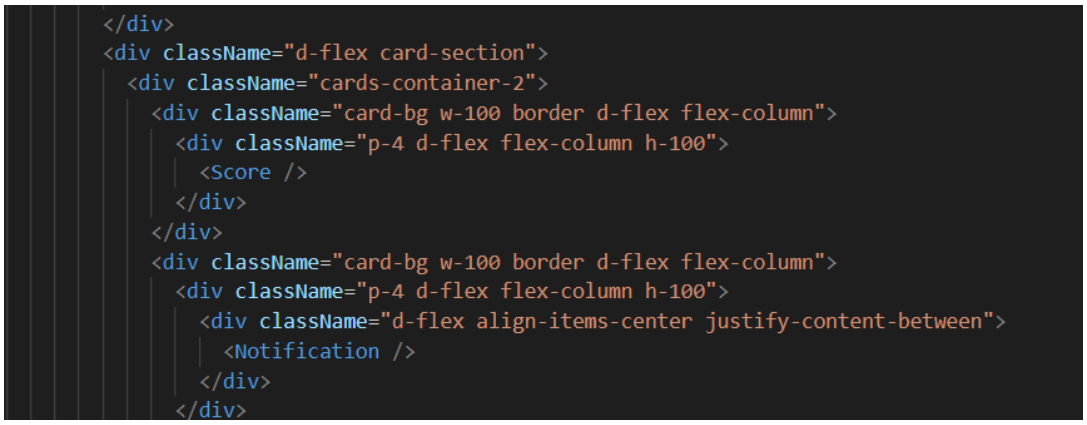
```
componentWillUnmount() {
  clearInterval(this.interval);
}
```

vi. render()

The render() method is the only required method in a class component. It is used to render HTML that builds up the page and props that is received if yes (getSearchedData() function is called with valid parameters).

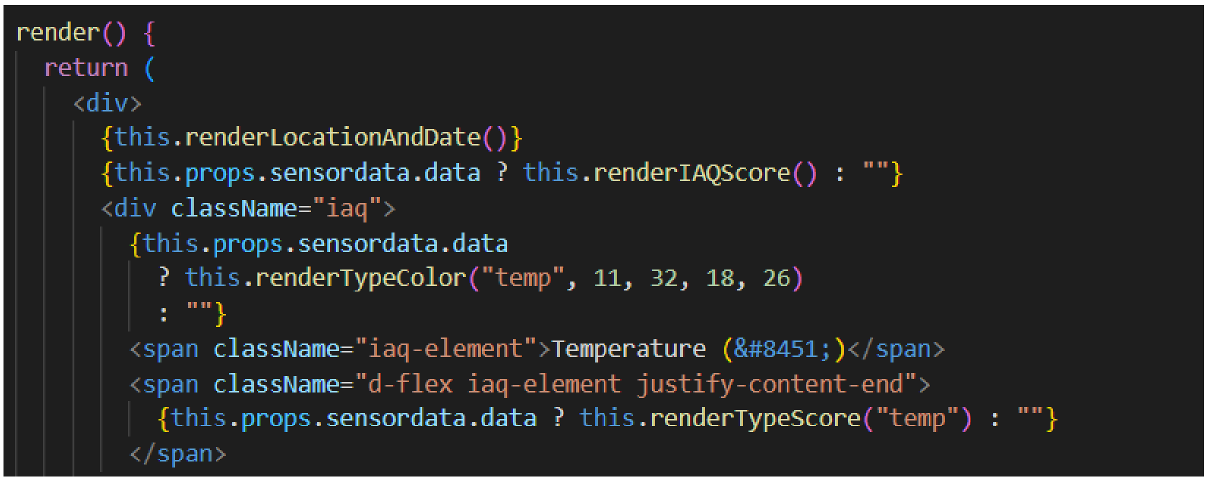


```
render() {
  return (
    <div className="dashboard d-flex">
      <div>
        <Sidebar />
      </div>
      <div
        style={{
          flex: "1 1 auto",
          display: "flex",
          flexFlow: "column",
          height: "100vh",
          overflowY: "hidden",
        }}
      >
        <Header />
        <div style={{ height: "100%" }}>
          <div style={{ height: "calc(100% - 50px)", overflowY: "scroll" }}>
            <div className="d-flex">
              <div className="mr-auto ml-5 mt-4">
                <h3>Dashboard</h3>
              </div>
            </div>

            <div className="d-flex card-section">
              <div className="cards-container">
                <div className="card-bg w-100 border d-flex flex-column">
                  <Select onSearchSubmit={this.onSearchSubmit} />
                </div>
              </div>
            </div>
            <div className="d-flex card-section">
```

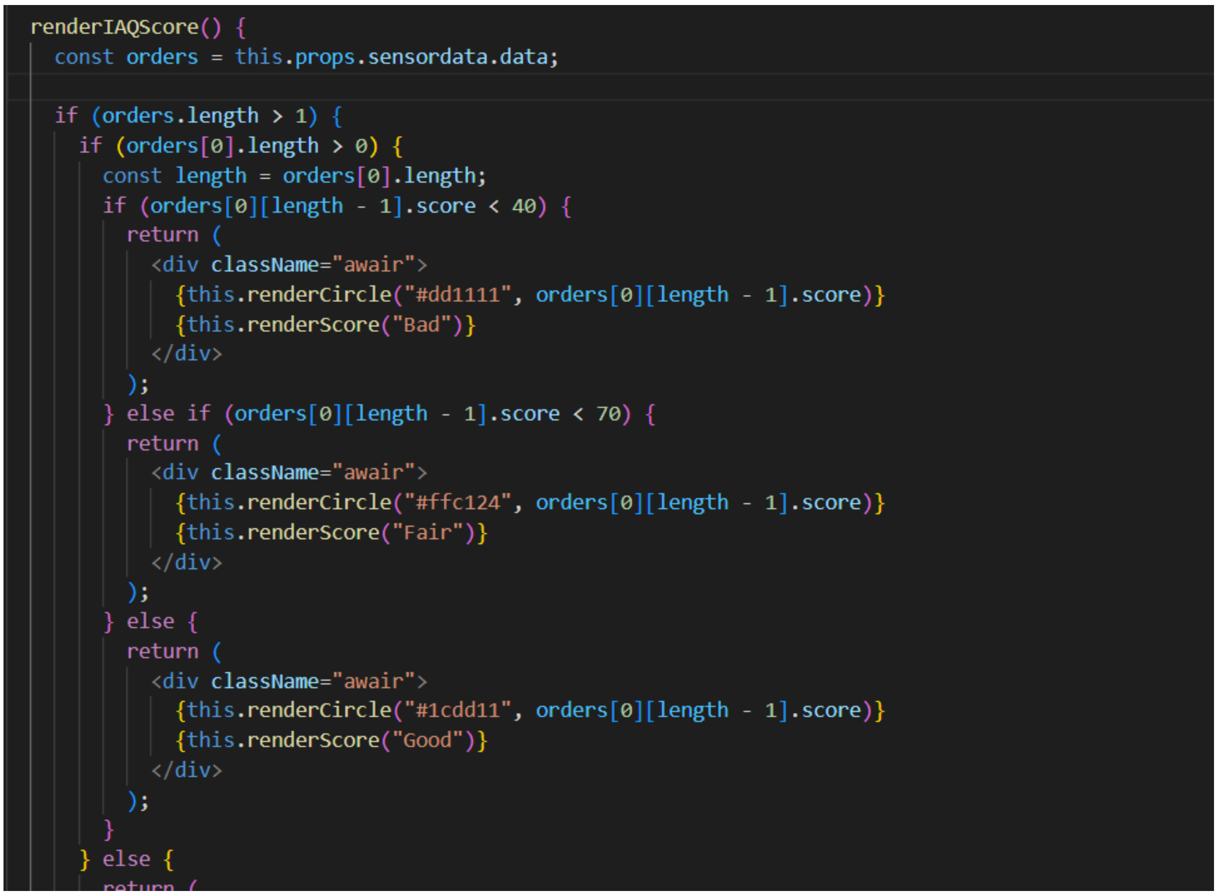


```
            </div>
            <div className="d-flex card-section">
              <div className="cards-container-2">
                <div className="card-bg w-100 border d-flex flex-column">
                  <div className="p-4 d-flex flex-column h-100">
                    <Score />
                  </div>
                </div>
                <div className="card-bg w-100 border d-flex flex-column">
                  <div className="p-4 d-flex flex-column h-100">
                    <div className="d-flex align-items-center justify-content-between">
                      <Notification />
                    </div>
                  </div>
```

<Score /> is a child class of the Dashboard class and can be passed into the render method. In the score class, functions that return HTML components using conditions is created. A constant named orders is created to store the data of this.props.sensordata and is then pass into the render method. this.props contains the props that were defined by the caller of this component.



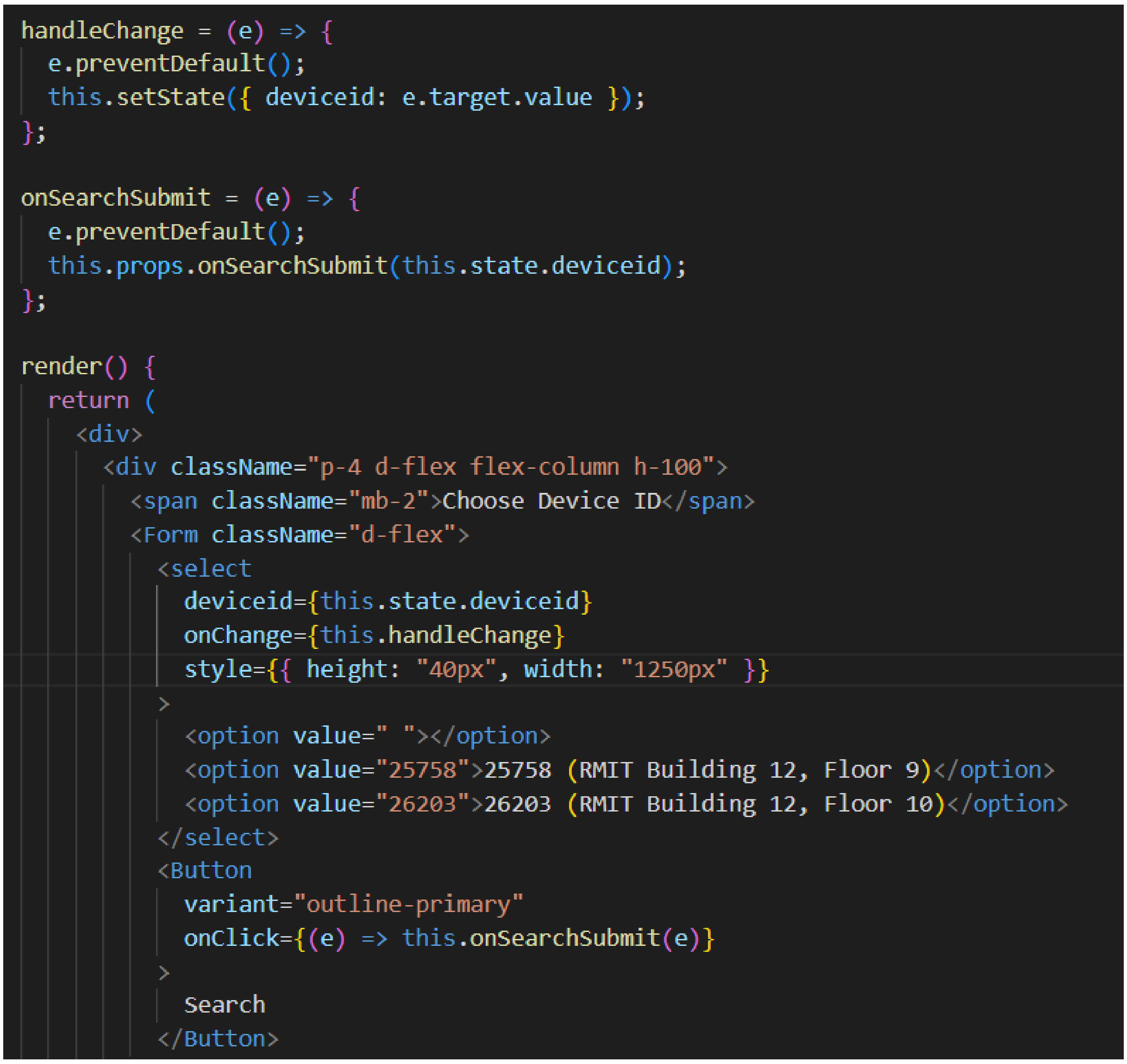
```
render() {
  return (
    <div>
      {this.renderLocationAndDate()}
      {this.props.sensordata.data ? this.renderIAQScore() : ""}
      <div className="iaq">
        {this.props.sensordata.data
          ? this.renderTypeColor("temp", 11, 32, 18, 26)
          : ""}
        <span className="iaq-element">Temperature (&#8451;)</span>
        <span className="d-flex iaq-element justify-content-end">
          {this.props.sensordata.data ? this.renderTypeScore("temp") : ""}
        </span>
```



```
renderIAQScore() {
  const orders = this.props.sensordata.data;

  if (orders.length > 1) {
    if (orders[0].length > 0) {
      const length = orders[0].length;
      if (orders[0][length - 1].score < 40) {
        return (
          <div className="awair">
            {this.renderCircle("#dd1111", orders[0][length - 1].score)}
            {this.renderScore("Bad")}
          </div>
        );
      } else if (orders[0][length - 1].score < 70) {
        return (
          <div className="awair">
            {this.renderCircle("#ffc124", orders[0][length - 1].score)}
            {this.renderScore("Fair")}
          </div>
        );
      } else {
        return (
          <div className="awair">
            {this.renderCircle("#1cdd11", orders[0][length - 1].score)}
            {this.renderScore("Good")}
          </div>
        );
      }
    } else {
      return (
```

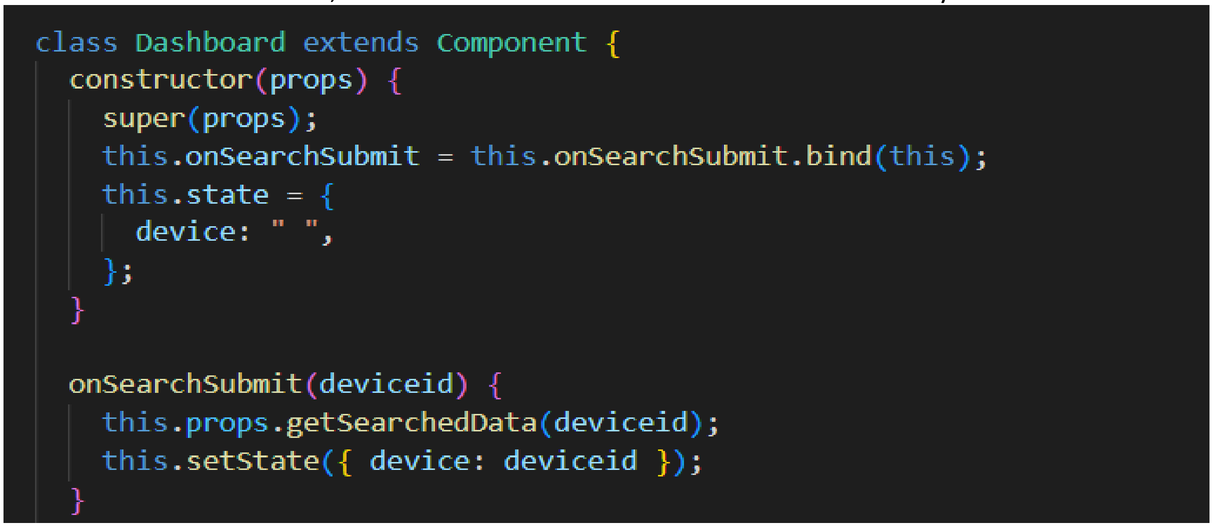
vii. Event

When using React, you generally don’t need to call addEventListener to add listeners to a DOM element after it is created. Instead, just provide a listener when the element is initially rendered (React, 2022). A select dropdown is created where user can choose their options and the value is passed as e in the handleChange() function. The e.target.value is then use to set state “deviceid” by calling setState() function. onClick() is a mouse event that triggers whenever user click the submit button. It is assigned to onSearchSubmit() function which calls onSearchSubmit() with the parameter “this.state.deviceid”.



```
handleChange = (e) => {
  e.preventDefault();
  this.setState({ deviceid: e.target.value });
};

onSearchSubmit = (e) => {
  e.preventDefault();
  this.props.onSearchSubmit(this.state.deviceid);
};

render() {
  return (
    <div>
      <div className="p-4 d-flex flex-column h-100">
        <span className="mb-2">Choose Device ID</span>
        <Form className="d-flex">
          <select
            deviceid={this.state.deviceid}
            onChange={this.handleChange}
            style={{ height: "40px", width: "1250px" }}
          >
            <option value=" "></option>
            <option value="25758">25758 (RMIT Building 12, Floor 9)</option>
            <option value="26203">26203 (RMIT Building 12, Floor 10)</option>
          </select>
          <Button
            variant="outline-primary"
            onClick={(e) => this.onSearchSubmit(e)}
          >
            Search
          </Button>
```

In JavaScript, class methods are not bound by default (React, 2022). The onSearchSubmit() is then bind this.onSearchSubmit() in the parent class and pass it to onSearchSubmit(). If the function is not bind, this will be undefined when the function is actually called.



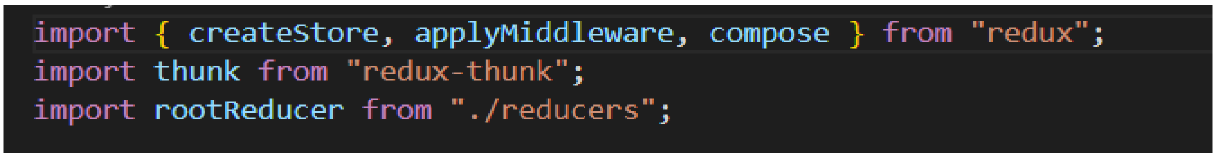
```
class Dashboard extends Component {
  constructor(props) {
    super(props);
    this.onSearchSubmit = this.onSearchSubmit.bind(this);
    this.state = {
      device: " ",
    };
  }

  onSearchSubmit(deviceid) {
    this.props.getSearchedData(deviceid);
    this.setState({ device: deviceid });
  }
```

vi. redux-thunk

For redux-thunk, we needto write an action creator that returns a function instead of creating an object. This function passed the getState and dispatch functions from Redux. First, install the redux-thunk library:

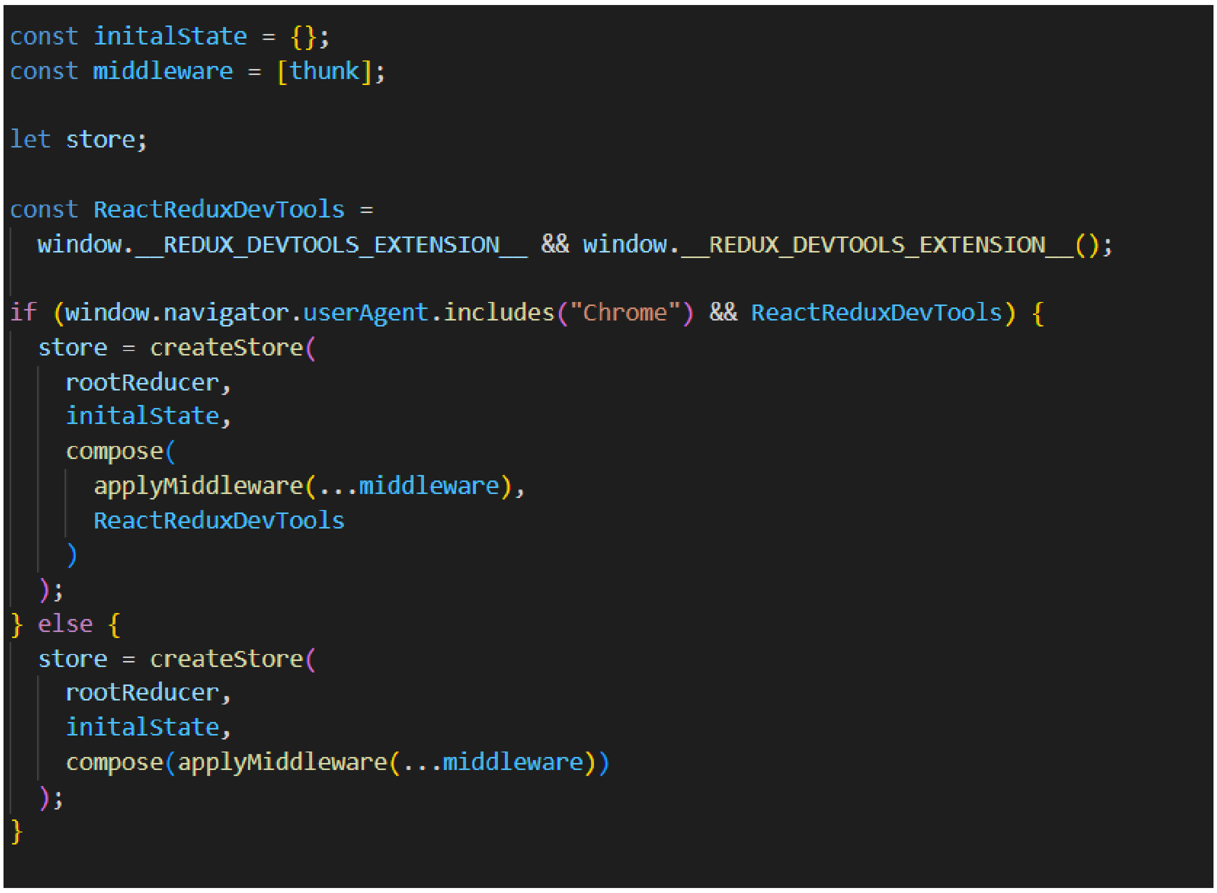
```
>npm I redux-thunk
```

To make the library work as expected, it has to be applied as middleware.



```
import { createStore, applyMiddleware, compose } from "redux";
import thunk from "redux-thunk";
import rootReducer from "./reducers";
```

We create a Redux store that holds the complete state tree of your app with createStore(reducer, [preloadedState], [enhancer]) function. The reducer is a reducing function that returns the next state tree, given the current state tree and an action to handle. The preloadedState is an initial state that is specified to hydrate the state from the server in universal apps, or to restore a previously serialized user session. The store enhancer is specified to enhance the store with middleware. The only store enhancer that ships with Redux is applyMiddleware().
...middleware (arguments): Functions that conform to the Redux middleware API. Each middleware receives Store's dispatch and getState functions as named arguments, and returns a function (Redux, 2022).



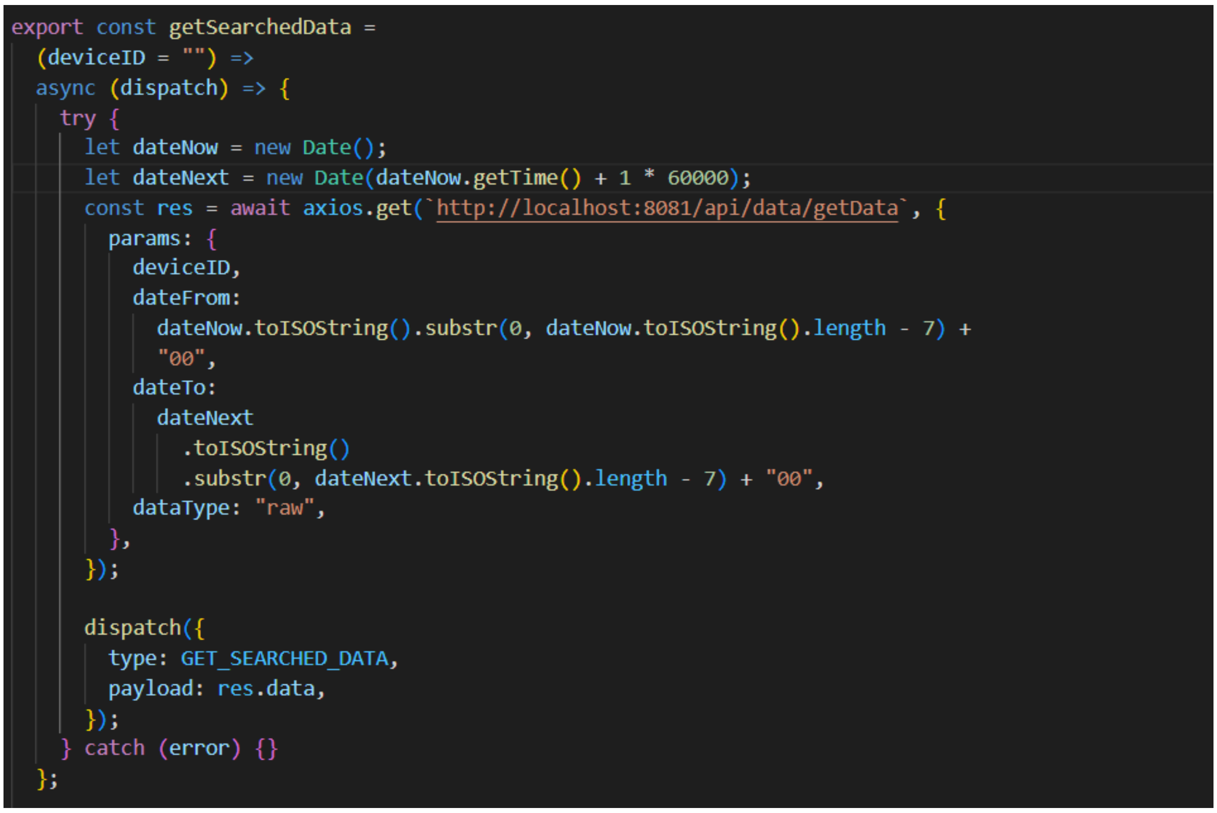
```
const initalState = {};
const middleware = [thunk];

let store;

const ReactReduxDevTools =
  window.__REDUX_DEVTOOLS_EXTENSION__ && window.__REDUX_DEVTOOLS_EXTENSION__();

if (window.navigator.userAgent.includes("Chrome") && ReactReduxDevTools) {
  store = createStore(
    rootReducer,
    initalState,
    compose(
      applyMiddleware(...middleware),
      ReactReduxDevTools
    )
  );
} else {
  store = createStore(
    rootReducer,
    initalState,
    compose(applyMiddleware(...middleware))
  );
}
```

vii. Axios

To use axios, make sure Node.js version 10.16.0 installed on your computer. Then run this command to install Axios:

>npm install axios@0.24.0

We use axios.get(url) with a URL from an API endpoint to get a promise which returns a response object. Inside the response object, there is data that is then assigned the value of data. So we use await to return the resolved value from the promise.



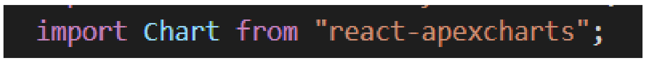
```
export const getSearchedData =
  (deviceID = "") =>
  async (dispatch) => {
    try {
      let dateNow = new Date();
      let dateNext = new Date(dateNow.getTime() + 1 * 60000);
      const res = await axios.get(`http://localhost:8081/api/data/getData`, {
        params: {
          deviceID,
          dateFrom:
            dateNow.toISOString().substr(0, dateNow.toISOString().length - 7) +
            "00",
          dateTo:
            dateNext
              .toISOString()
              .substr(0, dateNext.toISOString().length - 7) + "00",
          dataType: "raw",
        },
      });

      dispatch({
        type: GET_SEARCHED_DATA,
        payload: res.data,
      });
    } catch (error) {}
  };
```

viii.   APEXCHART.js

First install apexchart via npm:

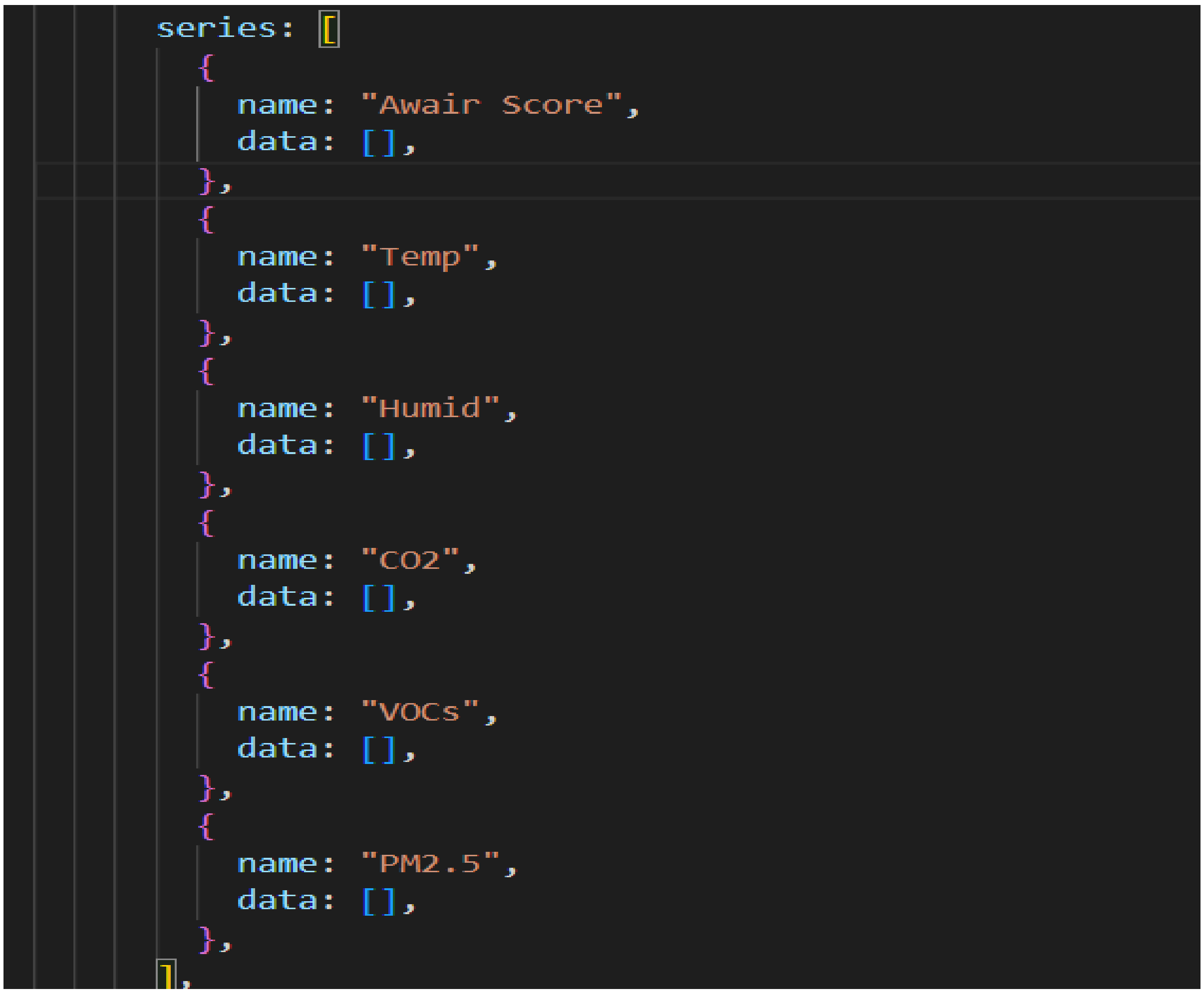
>npm I apexchart

Usage:



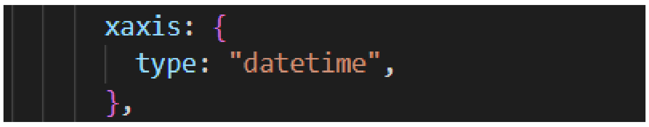
```
import Chart from "react-apexcharts";
```

As you can see below, the ApexCharts Component holds 2 props, options and series in the constructor in class Chart.js. The series is a set of data. In our project, there are six categories where where all the values in the data array indicates y axes values.

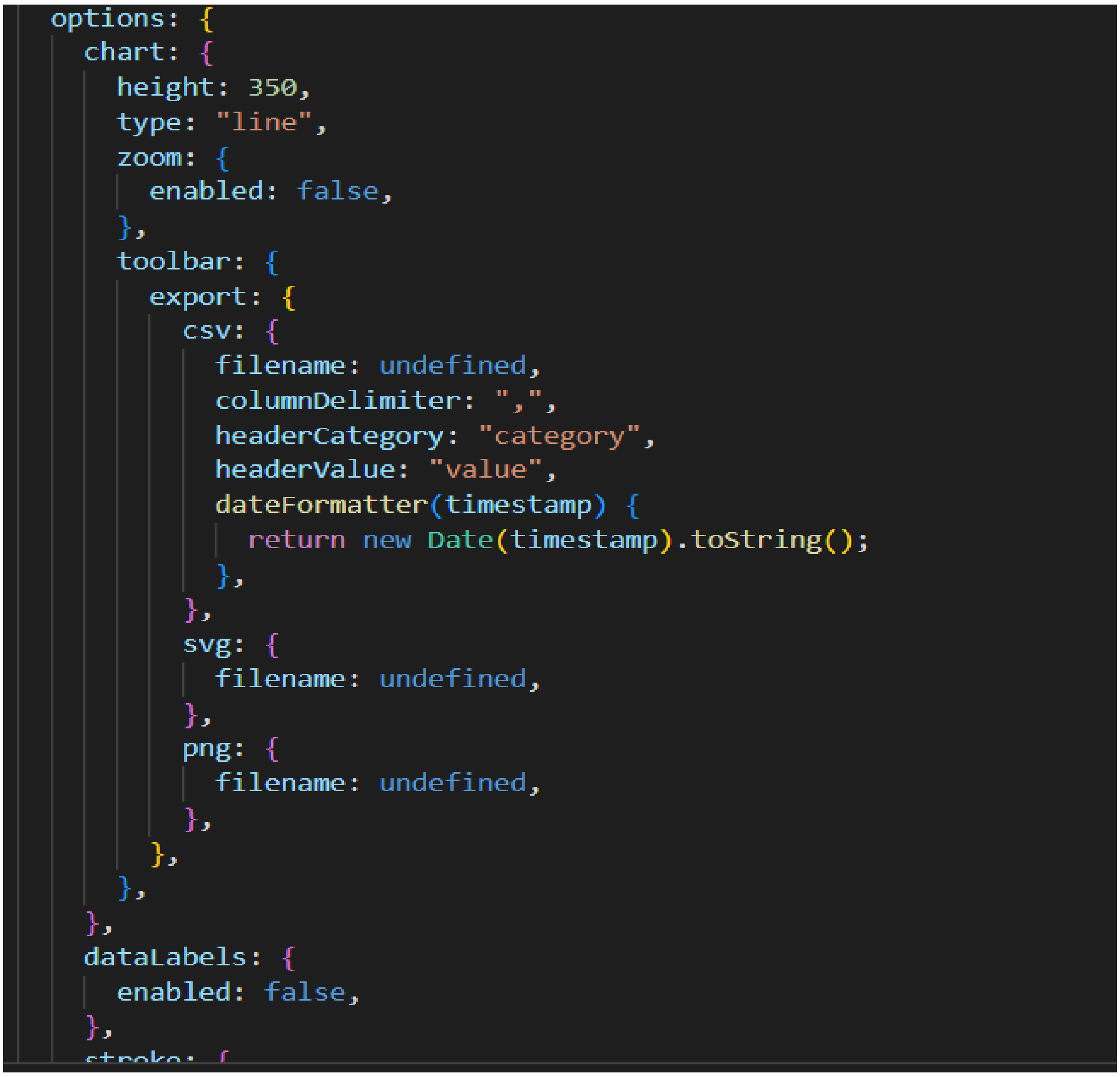


```
series: [
  {
    name: "Awair Score",
    data: [],
  },
  {
    name: "Temp",
    data: [],
  },
  {
    name: "Humid",
    data: [],
  },
  {
    name: "CO2",
    data: [],
  },
  {
    name: "VOCs",
    data: [],
  },
  {
    name: "PM2.5",
    data: [],
  },
],
```

The option is where you design and personalize your chart. It allows you to set the titles of the graph, enable tooltips, and so on. You can find the full documentation at https://apexcharts.com/docs/installation/ . Since we are using timestamp for the x-axis of our chart, we need to configure the type of x-axis in the options props.

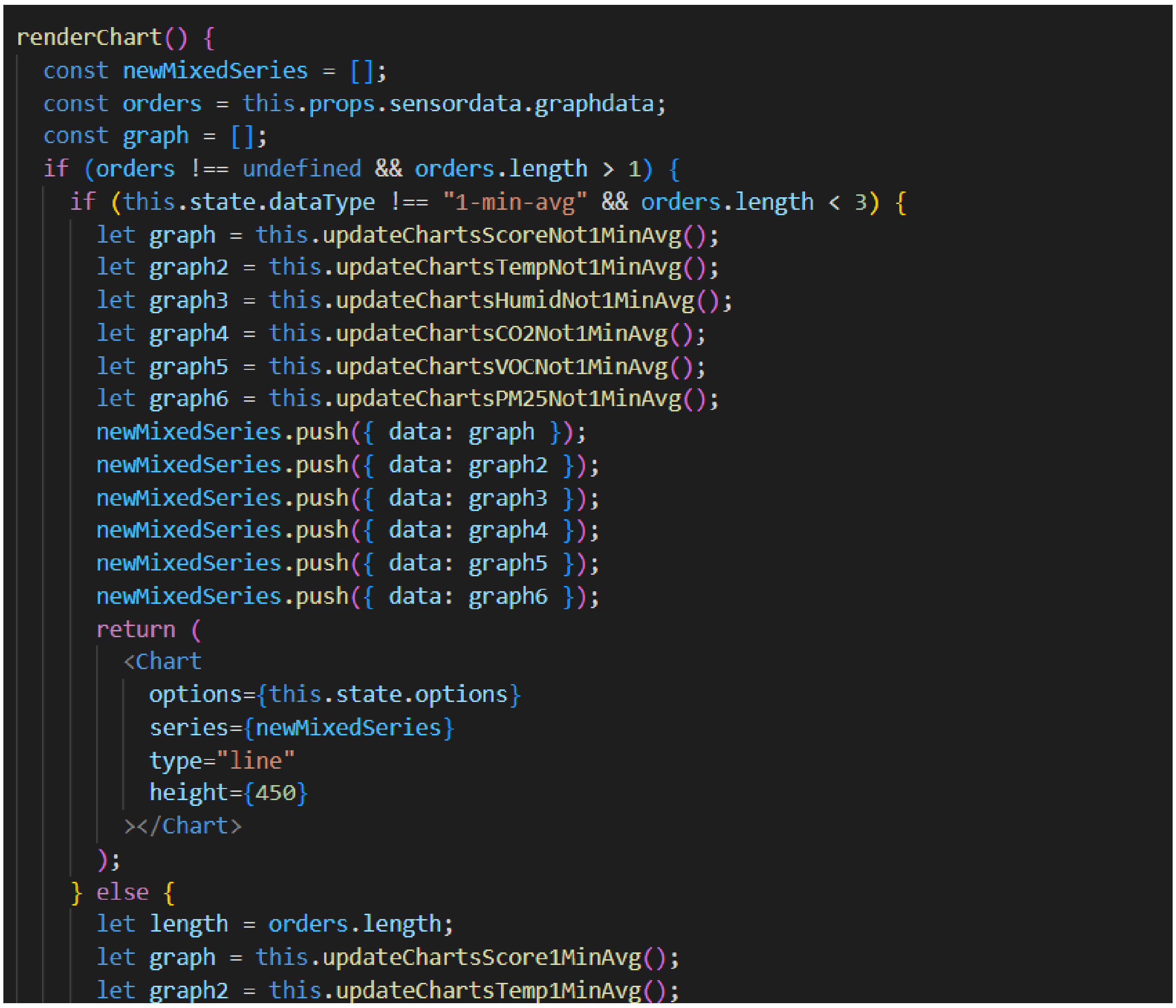


```
xaxis: {
  type: "datetime",
},
```



```
  options: {
    chart: {
      height: 350,
      type: "line",
      zoom: {
        enabled: false,
      },
      toolbar: {
        export: {
          csv: {
            filename: undefined,
            columnDelimiter: ",",
            headerCategory: "category",
            headerValue: "value",
            dateFormatter(timestamp) {
              return new Date(timestamp).toString();
            },
          },
          svg: {
            filename: undefined,
          },
          png: {
            filename: undefined,
          },
        },
      },
    },
    dataLabels: {
      enabled: false,
    },
    stroke: {
```

Updating the React chart data is simple. We just have to update the series prop and it will automatically trigger event to update the chart.



```
renderChart() {
  const newMixedSeries = [];
  const orders = this.props.sensordata.graphdata;
  const graph = [];
  if (orders !== undefined && orders.length > 1) {
    if (this.state.dataType !== "1-min-avg" && orders.length < 3) {
      let graph = this.updateChartsScoreNot1MinAvg();
      let graph2 = this.updateChartsTempNot1MinAvg();
      let graph3 = this.updateChartsHumidNot1MinAvg();
      let graph4 = this.updateChartsCO2Not1MinAvg();
      let graph5 = this.updateChartsVOCNot1MinAvg();
      let graph6 = this.updateChartsPM25Not1MinAvg();
      newMixedSeries.push({ data: graph });
      newMixedSeries.push({ data: graph2 });
      newMixedSeries.push({ data: graph3 });
      newMixedSeries.push({ data: graph4 });
      newMixedSeries.push({ data: graph5 });
      newMixedSeries.push({ data: graph6 });
      return (
        <Chart
          options={this.state.options}
          series={newMixedSeries}
          type="line"
          height={450}
        ></Chart>
      );
    } else {
      let length = orders.length;
      let graph = this.updateChartsScore1MinAvg();
      let graph2 = this.updateChartsTemp1MinAvg();
```

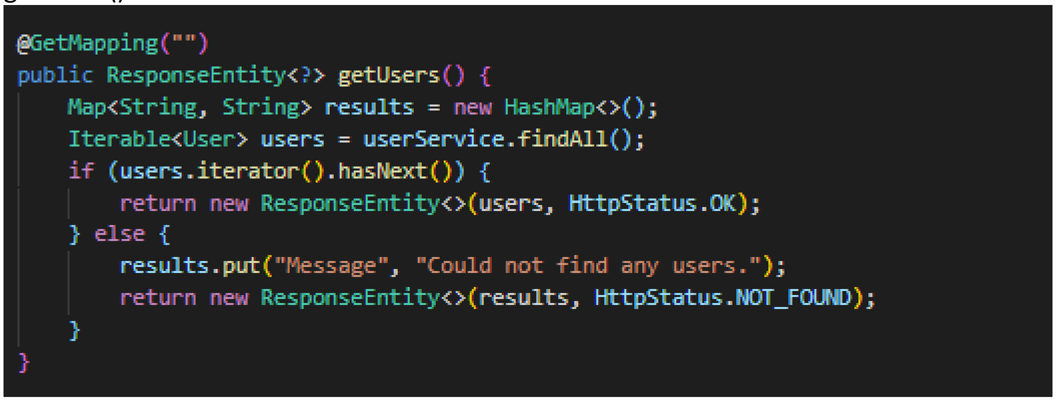
6.2 BACKEND:

We utilised a microservice architecture for our backend, separating small, independent and loosely coupled services. This included the User service and Data service.

6.2.1 User services

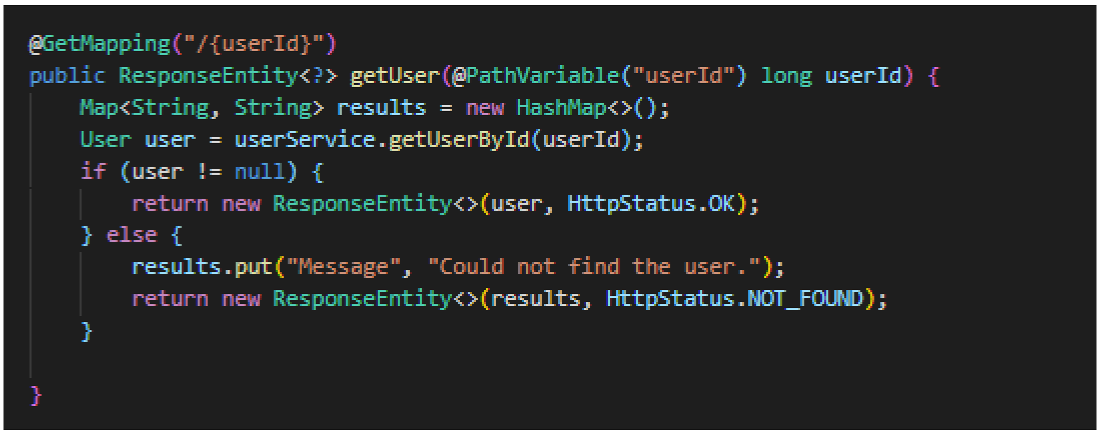
User Controller

getUsers()



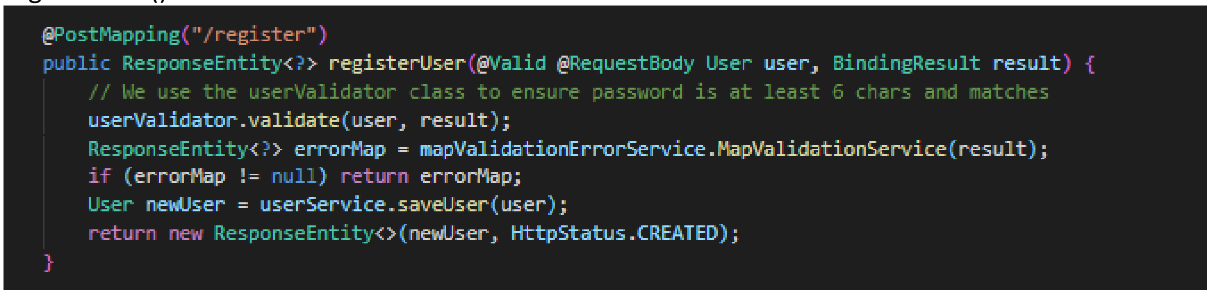
```
@GetMapping("")
public ResponseEntity<?> getUsers() {
    Map<String, String> results = new HashMap<>();
    Iterable<User> users = userService.findAll();
    if (users.iterator().hasNext()) {
        return new ResponseEntity<>(users, HttpStatus.OK);
    } else {
        results.put("Message", "Could not find any users.");
        return new ResponseEntity<>(results, HttpStatus.NOT_FOUND);
    }
}
```

getUser()



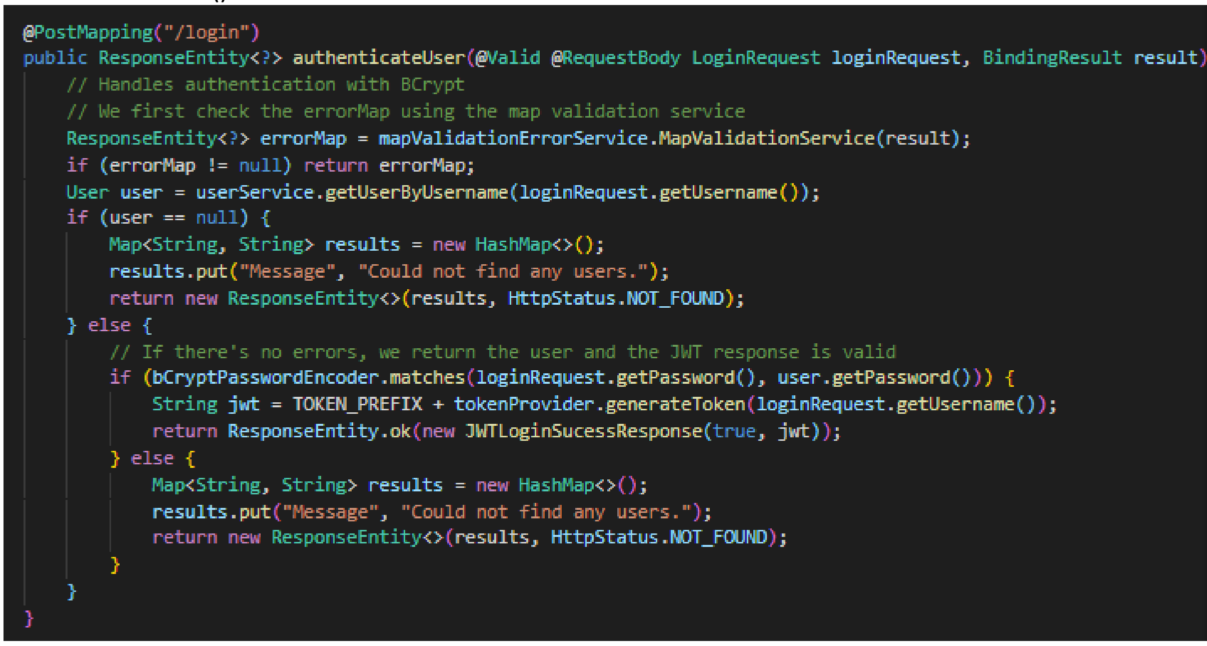
```java
@GetMapping("/{userId}")
public ResponseEntity<?> getUser(@PathVariable("userId") long userId) {
    Map<String, String> results = new HashMap<>();
    User user = userService.getUserById(userId);
    if (user != null) {
        return new ResponseEntity<>(user, HttpStatus.OK);
    } else {
        results.put("Message", "Could not find the user.");
        return new ResponseEntity<>(results, HttpStatus.NOT_FOUND);
    }

}
```

registerUser()



```java
@PostMapping("/register")
public ResponseEntity<?> registerUser(@Valid @RequestBody User user, BindingResult result) {
    // We use the userValidator class to ensure password is at least 6 chars and matches
    userValidator.validate(user, result);
    ResponseEntity<?> errorMap = mapValidationErrorService.MapValidationService(result);
    if (errorMap != null) return errorMap;
    User newUser = userService.saveUser(user);
    return new ResponseEntity<>(newUser, HttpStatus.CREATED);
}
```

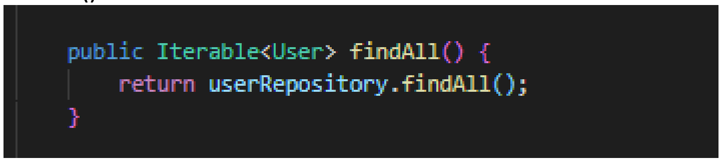
authenticateUser()



```java
@PostMapping("/login")
public ResponseEntity<?> authenticateUser(@Valid @RequestBody LoginRequest loginRequest, BindingResult result)
    // Handles authentication with BCrypt
    // We first check the errorMap using the map validation service
    ResponseEntity<?> errorMap = mapValidationErrorService.MapValidationService(result);
    if (errorMap != null) return errorMap;
    User user = userService.getUserByUsername(loginRequest.getUsername());
    if (user == null) {
        Map<String, String> results = new HashMap<>();
        results.put("Message", "Could not find any users.");
        return new ResponseEntity<>(results, HttpStatus.NOT_FOUND);
    } else {
        // If there's no errors, we return the user and the JWT response is valid
        if (bCryptPasswordEncoder.matches(loginRequest.getPassword(), user.getPassword())) {
            String jwt = TOKEN_PREFIX + tokenProvider.generateToken(loginRequest.getUsername());
            return ResponseEntity.ok(new JWTLoginSucessResponse(true, jwt));
        } else {
            Map<String, String> results = new HashMap<>();
            results.put("Message", "Could not find any users.");
            return new ResponseEntity<>(results, HttpStatus.NOT_FOUND);
        }
    }
}
```

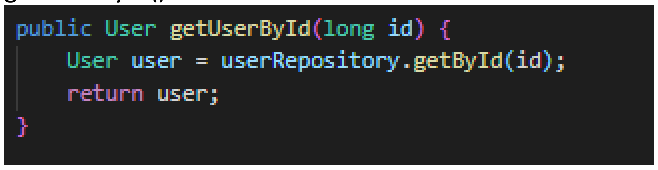
User Service

findAll()



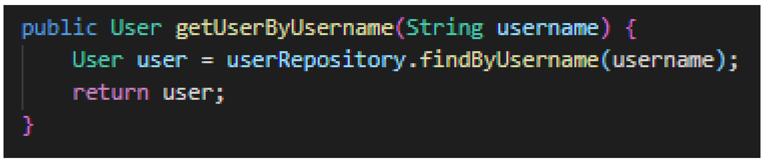
```java
public Iterable<User> findAll() {
    return userRepository.findAll();
}
```

getUserById()



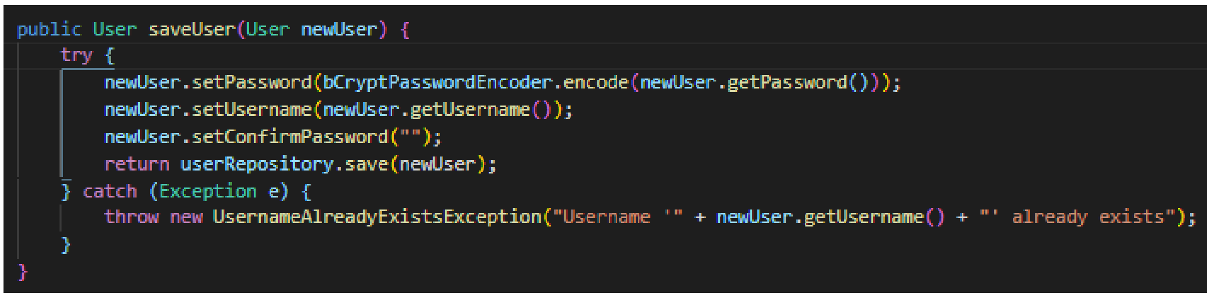
```java
public User getUserById(long id) {
    User user = userRepository.getById(id);
    return user;
}
```

getUserByUsername()



```java
public User getUserByUsername(String username) {
    User user = userRepository.findByUsername(username);
    return user;
}
```

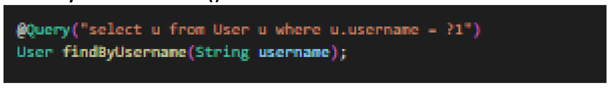
saveUser()



```java
public User saveUser(User newUser) {
    try {
        newUser.setPassword(bCryptPasswordEncoder.encode(newUser.getPassword()));
        newUser.setUsername(newUser.getUsername());
        newUser.setConfirmPassword("");
        return userRepository.save(newUser);
    } catch (Exception e) {
        throw new UsernameAlreadyExistsException("Username '" + newUser.getUsername() + "' already exists");
    }
}
```

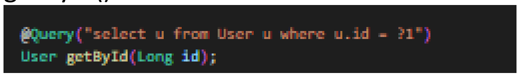
User Repository

findByUserName()



```java
@Query("select u from User u where u.username = ?1")
User findByUsername(String username);
```

getById()

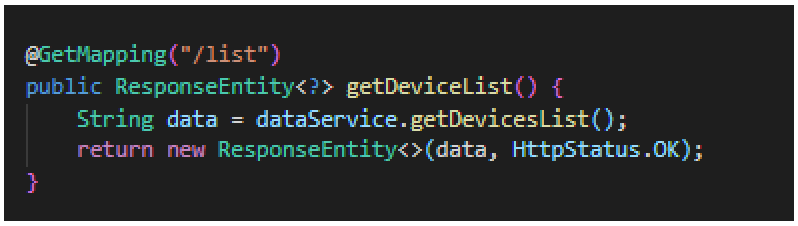


```java
@Query("select u from User u where u.id = ?1")
User getById(Long id);
```

6.2.2 Data services

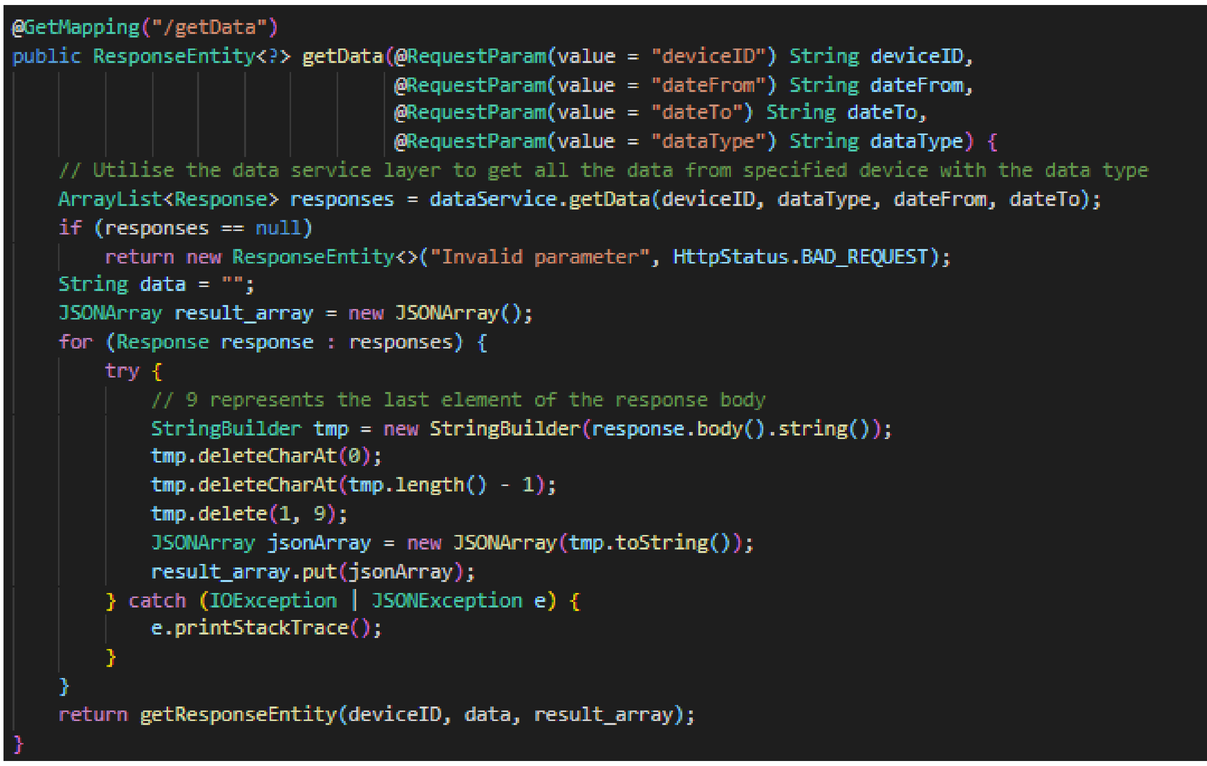
Data Controller

getDeviceList()



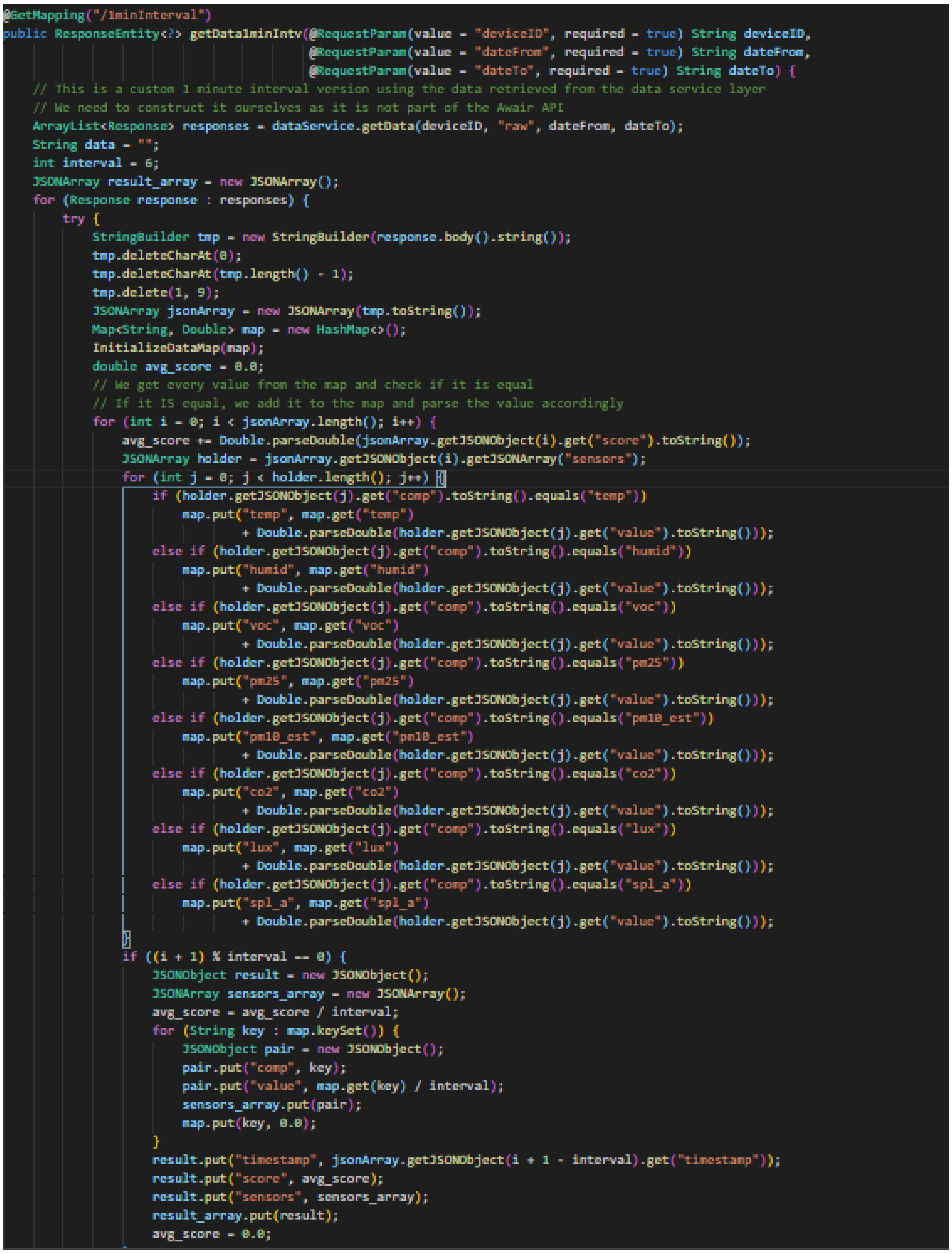
```java
@GetMapping("/list")
public ResponseEntity<?> getDeviceList() {
    String data = dataService.getDevicesList();
    return new ResponseEntity<>(data, HttpStatus.OK);
}
```

getData()



```
@GetMapping("/getData")
public ResponseEntity<?> getData(@RequestParam(value = "deviceID") String deviceID,
                                 @RequestParam(value = "dateFrom") String dateFrom,
                                 @RequestParam(value = "dateTo") String dateTo,
                                 @RequestParam(value = "dataType") String dataType) {
    // Utilise the data service layer to get all the data from specified device with the data type
    ArrayList<Response> responses = dataService.getData(deviceID, dataType, dateFrom, dateTo);
    if (responses == null)
        return new ResponseEntity<>("Invalid parameter", HttpStatus.BAD_REQUEST);
    String data = "";
    JSONArray result_array = new JSONArray();
    for (Response response : responses) {
        try {
            // 9 represents the last element of the response body
            StringBuilder tmp = new StringBuilder(response.body().string());
            tmp.deleteCharAt(0);
            tmp.deleteCharAt(tmp.length() - 1);
            tmp.delete(1, 9);
            JSONArray jsonArray = new JSONArray(tmp.toString());
            result_array.put(jsonArray);
        } catch (IOException | JSONException e) {
            e.printStackTrace();
        }
    }
    return getResponseEntity(deviceID, data, result_array);
}
```

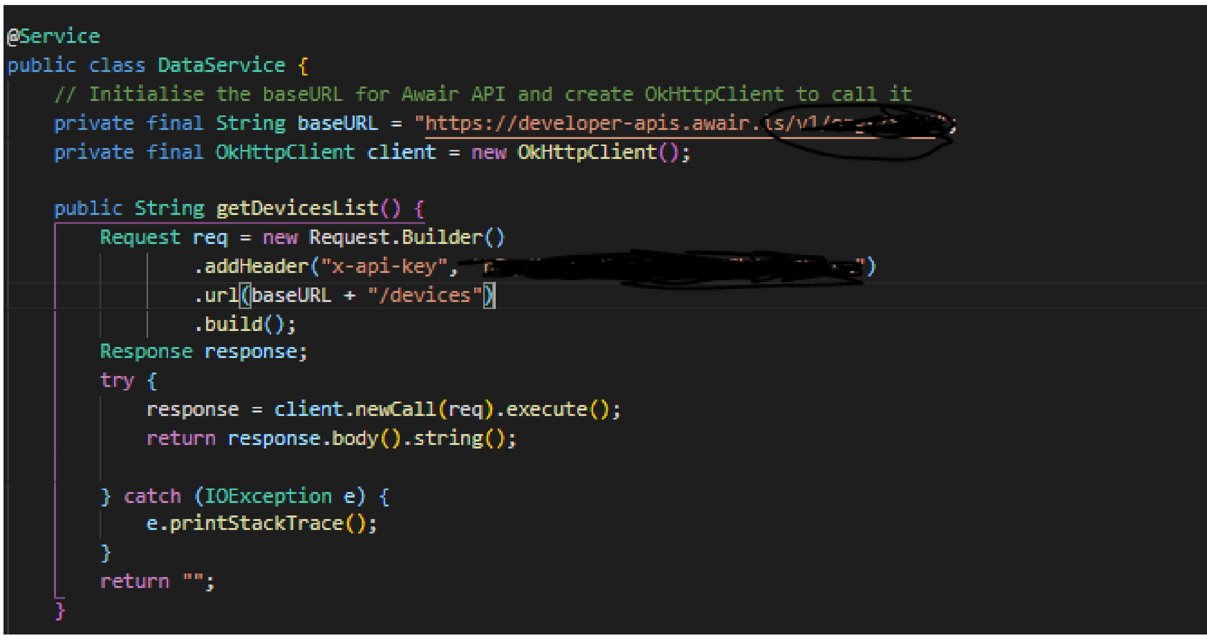
getData1minIntv()



```java
@GetMapping("/1minInterval")
public ResponseEntity<?> getData1minIntv(@RequestParam(value = "deviceID", required = true) String deviceID,
                                         @RequestParam(value = "dateFrom", required = true) String dateFrom,
                                         @RequestParam(value = "dateTo", required = true) String dateTo) {
    // This is a custom 1 minute interval version using the data retrieved from the data service layer
    // We need to construct it ourselves as it is not part of the Awair API
    ArrayList<Response> responses = dataService.getData(deviceID, "raw", dateFrom, dateTo);
    String data = "";
    int interval = 6;
    JSONArray result_array = new JSONArray();
    for (Response response : responses) {
        try {
            StringBuilder tmp = new StringBuilder(response.body().string());
            tmp.deleteCharAt(0);
            tmp.deleteCharAt(tmp.length() - 1);
            tmp.delete(1, 9);
            JSONArray jsonArray = new JSONArray(tmp.toString());
            Map<String, Double> map = new HashMap<>();
            InitializeDataMap(map);
            double avg_score = 0.0;
            // We get every value from the map and check if it is equal
            // If it IS equal, we add it to the map and parse the value accordingly
            for (int i = 0; i < jsonArray.length(); i++) {
                avg_score += Double.parseDouble(jsonArray.getJSONObject(i).get("score").toString());
                JSONArray holder = jsonArray.getJSONObject(i).getJSONArray("sensors");
                for (int j = 0; j < holder.length(); j++) {
                    if (holder.getJSONObject(j).get("comp").toString().equals("temp"))
                        map.put("temp", map.get("temp")
                                + Double.parseDouble(holder.getJSONObject(j).get("value").toString()));
                    else if (holder.getJSONObject(j).get("comp").toString().equals("humid"))
                        map.put("humid", map.get("humid")
                                + Double.parseDouble(holder.getJSONObject(j).get("value").toString()));
                    else if (holder.getJSONObject(j).get("comp").toString().equals("voc"))
                        map.put("voc", map.get("voc")
                                + Double.parseDouble(holder.getJSONObject(j).get("value").toString()));
                    else if (holder.getJSONObject(j).get("comp").toString().equals("pm25"))
                        map.put("pm25", map.get("pm25")
                                + Double.parseDouble(holder.getJSONObject(j).get("value").toString()));
                    else if (holder.getJSONObject(j).get("comp").toString().equals("pm10_est"))
                        map.put("pm10_est", map.get("pm10_est")
                                + Double.parseDouble(holder.getJSONObject(j).get("value").toString()));
                    else if (holder.getJSONObject(j).get("comp").toString().equals("co2"))
                        map.put("co2", map.get("co2")
                                + Double.parseDouble(holder.getJSONObject(j).get("value").toString()));
                    else if (holder.getJSONObject(j).get("comp").toString().equals("lux"))
                        map.put("lux", map.get("lux")
                                + Double.parseDouble(holder.getJSONObject(j).get("value").toString()));
                    else if (holder.getJSONObject(j).get("comp").toString().equals("spl_a"))
                        map.put("spl_a", map.get("spl_a")
                                + Double.parseDouble(holder.getJSONObject(j).get("value").toString()));
                }
                if ((i + 1) % interval == 0) {
                    JSONObject result = new JSONObject();
                    JSONArray sensors_array = new JSONArray();
                    avg_score = avg_score / interval;
                    for (String key : map.keySet()) {
                        JSONObject pair = new JSONObject();
                        pair.put("comp", key);
                        pair.put("value", map.get(key) / interval);
                        sensors_array.put(pair);
                        map.put(key, 0.0);
                    }
                    result.put("timestamp", jsonArray.getJSONObject(i + 1 - interval).get("timestamp"));
                    result.put("score", avg_score);
                    result.put("sensors", sensors_array);
                    result_array.put(result);
                    avg_score = 0.0;
```

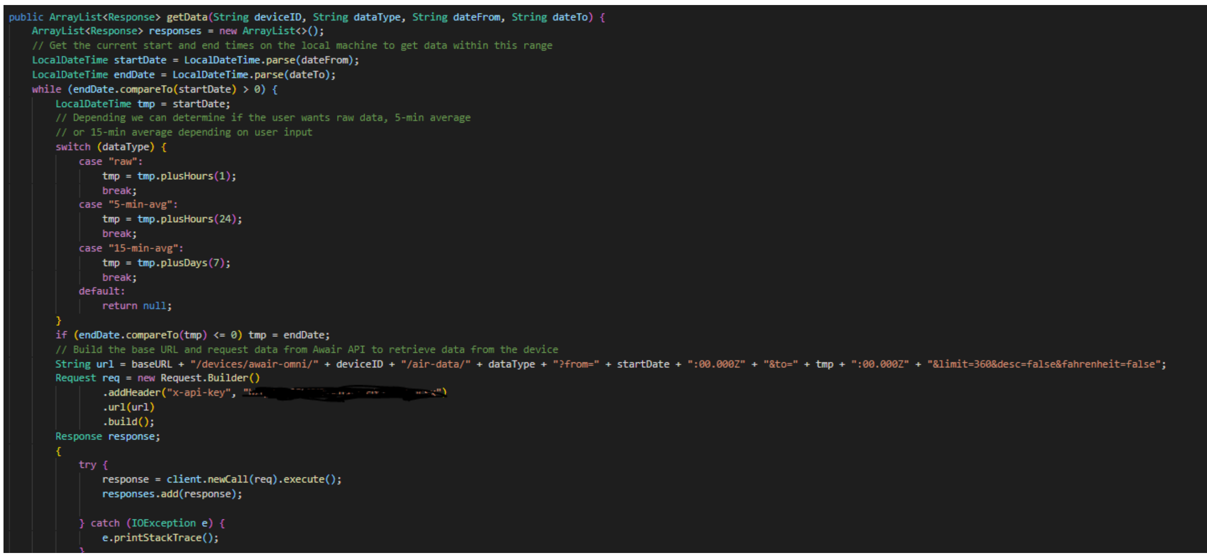
Data Services

getDeviceList()



```java
@Service
public class DataService {
    // Initialise the baseURL for Awair API and create OkHttpClient to call it
    private final String baseURL = "https://developer-apis.awair.is/v1/[redacted];
    private final OkHttpClient client = new OkHttpClient();

    public String getDevicesList() {
        Request req = new Request.Builder()
                .addHeader("x-api-key", [redacted]")
                .url(baseURL + "/devices")
                .build();
        Response response;
        try {
            response = client.newCall(req).execute();
            return response.body().string();

        } catch (IOException e) {
            e.printStackTrace();
        }
        return "";
    }
```

getData()



```java
public ArrayList<Response> getData(String deviceID, String dataType, String dateFrom, String dateTo) {
    ArrayList<Response> responses = new ArrayList<>();
    // Get the current start and end times on the local machine to get data within this range
    LocalDateTime startDate = LocalDateTime.parse(dateFrom);
    LocalDateTime endDate = LocalDateTime.parse(dateTo);
    while (endDate.compareTo(startDate) > 0) {
        LocalDateTime tmp = startDate;
        // Depending we can determine if the user wants raw data, 5-min average
        // or 15-min average depending on user input
        switch (dataType) {
            case "raw":
                tmp = tmp.plusHours(1);
                break;
            case "5-min-avg":
                tmp = tmp.plusHours(24);
                break;
            case "15-min-avg":
                tmp = tmp.plusDays(7);
                break;
            default:
                return null;
        }
        if (endDate.compareTo(tmp) <= 0) tmp = endDate;
        // Build the base URL and request data from Awair API to retrieve data from the device
        String url = baseURL + "/devices/awair-omni/" + deviceID + "/air-data/" + dataType + "?from=" + startDate + ":00.000Z" + "&to=" + tmp + ":00.000Z" + "&limit=360&desc=false&fahrenheit=false";
        Request req = new Request.Builder()
                .addHeader("x-api-key", "[redacted]")
                .url(url)
                .build();
        Response response;
        {
            try {
                response = client.newCall(req).execute();
                responses.add(response);

            } catch (IOException e) {
                e.printStackTrace();
```

7 DEPLOYMENT INSTRUCTIONS

Please refer to the README.md for deployment and setup instructions. This is located on the main branch of the GitHub repository.
Link: https://github.com/s3783616/P107IAQ

8 TEST SPECIFICATIONS

FRONTEND

Enzyme

Enzyme is a JavaScript Testing utility for React that makes it easier to assert, manipulate, and traverse React Components' output (Vijay, T 2017). It is not a unit testing framework and does not have a test runner or an assertion library.

Jest

Jest is a JavaScript unit testing framework, used by Facebook to test services and React applications (Vijay, T 2017). Jest is a framework and not a library which comes with a test runner, assertion library, and good mocking support. It is built on top of Jasmine and is able to test react components in a novel way: Snapshot testing. If the output of the current test run matches the snapshot of the previous test run, the test passes.

BACKEND

JUnit

JUnit 5 is the next generation of JUnit. The goal is to create an up-to-date foundation for developer-side testing on the JVM. This includes focusing on Java 8 and above, as well as enabling many different styles of testing.

Mockito

Mockito is a mocking framework that tastes good. It lets you write beautiful tests with a clean & simple API. Mockito doesn't give you hangover because the tests are very readable, and they produce clean verification errors. Read more about features & motivations.

Postman

Postman is an API platform for building and using APIs. Postman simplifies each step of the API lifecycle and streamlines collaboration so you can create better APIs—faster.

Test Case ID	Req. covered	Test Objective	Preconditions	Steps	Test data	Expected result
1	R1, R2	Able to find a user if the username exists	Username = “username”	Given a new user, when saving to the user repository we assert that the expected username is equal to the mocked user	User userRepository Mock	User is equal to user
2	R1, R2	Able to find a user by ID	Create an ID of type LONG Create a mock username	Given a new user, when saving to the user repository we assert that the user expected username is equal to the mocked user	User userRepository Mock	User is equal to user
3	R1, R2	Able to get a user by username	fake username	Given a fakeusername, when calling the mocked userservice, verify that the repository is called	Fake Username	Verify user repository is called
4	R2	Find all users	N/A	When calling the userService.findAll	Mocked user service	Verify user repository is called

				method, we verify that the user repository is called		
5	R2	Register User	Able to register a user	Given a user, when we perform a POST request with an ID, we expect the status to be 200/OK and a User to be registered into the repository	The User to be added Mocked user service	We expect the status to be OK/200 and a user to be registered into the database
6	R1	Get a user	Able to get a user from their ID	Given a user's ID, when we perform a get request with an ID, we expect the status to be 200/OK and a User to be returned	A new user Mocked user service	We expect the status to be OK/200 and return the requested user from their ID
7	R2	Get all users	Able to retrieve a list of users	When we perform a get request for a list of users, we expect the status to be 200/OK and a list	Mocked user service	We expect the status to be OK/200 and a list of users to be returned. Note that the list

				of users to be returned		can be empty.
8	R8	getData1mininterval	Retrieve a string of data for the 1 minute interval period	When we mock the dataService to return a certain string for the 1 min interval, we perform a GET request on "/getData" and expect the status to be OK/200 and a string of data for 1 min interval to be returned	Mocked data service	We expect the status to be OK/200 and a string of data for 1 min interval to be returned
9	R3	getDeviceList	Get a list of the devices from the AWAIR API	When we mock the dataservice and the API call, we perform a GET request on "/api/data/list" and expect the status to be OK/200 and a list of devices to be	Mocked dataService	We expect the status to be OK/200 and a list of devices to be returned

				returned		
10	R4	getData()	Retrieves data based on the device, data type and date range	When we mock the contructor values and the API call, we perform a GET request and expect the status to be OK/200 and a Response object with the data to be returned	Mocked dataservice	We expect the status to be OK/200 and a Response object with the data to be returned
11	R3	App displays suitable UI for user to read IAQ data	User has an account and had login into his/her account	Given a user login to the dashboard page, we expect there is a dropdown list and search button for user to search for IAQ data in their current location.	Select.js file	We expect there is a dropdown list and submit button at the dashboard page

9 TESTING RESULTS

We were able to create tests for the N-Tier architecture, namely tests for the user and data microservices' controllers, services and repository classes. All tests passed with no errors, meaning that we can reliably modify code with confidence that unit tests passed. We also did a complete acceptance test that covers the app full functionality in DS-107-IAQ-demo.pdf.

10 OTHER CONSIDERATIONS

AirCo might not work directly with IAQ sensors that aren't AWAIR compatible. Thus, the possibility of future efforts to make AirCo system agnostic is high!

11 REFERENCES

1. Visual Studio Code, 2022, Getting Started, viewed 28 May 2022, <https://code.visualstudio.com/docs>.
2. Visual Studio Code, 2022, Getting Started, viewed 28 May 2022, <https://marketplace.visualstudio.com/items?itemName=esbenp.prettier-vscode>
3. Visual Studio Code, 2022, Getting Started, viewed 28 May 2022, <https://marketplace.visualstudio.com/items?itemName=christian-kohler.path-intellisense>
4. Visual Studio Code, 2022, Getting Started, viewed 28 May 2022, <https://marketplace.visualstudio.com/items?itemName=leizongmin.node-module-intellisense>
5. Visual Studio Code, 2022, Getting Started, viewed 28 May 2022, <https://marketplace.visualstudio.com/items?itemName=rodrigovallades.es7-react-js-snippets>
6. Visual Studio Code, 2022, Getting Started, viewed 28 May 2022, <https://marketplace.visualstudio.com/items?itemName=CoenraadS.bracket-pair-colorizer-2>
7. Mdn web docs, 2022, CSS: Cascading Style Sheets, viewed 28 May 2022, <https://developer.mozilla.org/en-US/docs/Web/CSS >
8. Wikipedia, the free encyclopedia, 2022, React (JavaScript library), viewed 28 May 2022, <https://en.wikipedia.org/wiki/React_(JavaScript_library)>
9. Redux, 2022, Getting Started with Redux, viewed 28 May 2022, <https://redux.js.org/introduction/getting-started>
10. Redux, 2022, Writing Logic with Thunks, viewed 28 May 2022, <https://redux.js.org/usage/writing-logic-thunks>
11. Axios, 2022, Getting Started, viewed 28 May 2022, <https://axios-http.com/docs/intro>
12. React-bootstrap, 2022, Why React-Bootstrap?, viewed 28 May 2022, <https://react-bootstrap.github.io/getting-started/why-react-bootstrap/>
13. APEXCHART, 2022, What is ApexCharts?, viewed 28 May 2022, <https://apexcharts.com/>
14. React, 2022, React.Component, viewed 28 May 2022, <https://reactjs.org/docs/react-component.html#componentwillunmount>
15. React, 2022, Handling Events, viewed 28 May 2022, <https://reactjs.org/docs/handling-events.html>
16. Vijay, T 2017, Unit Testing React Components: Jest or Enzyme?, codementor, viewed 16 June 2022, < https://www.codementor.io/@vijayst/unit-testing-react-components-jest-or-enzyme-du1087lh8 >