FIRE ALERT

Distributes Systems | SE3020 – Assignment 2 | REST API

Contributors

IT18009132	Dilanka R.M.T.
IT18001112	Tharaka W.C.M.K.
IT18007848	Rathnayake R.H.C.S.
IT18006476	Gunarathna G.K.N.L.

Table of Contents

Table of Figures

Introduction

Fire Alert is the fire alarm monitoring system that enables to the user monitor and controls the Fire Sensor details remotely. It provides an interface to monitor the smoke level and the co2 level of each room that has fixed a Fire Sensor inside. In case of an emergency that the smoke level or the co2 level goes up than usual, the administrator will be notified via SMS and an Email immediately.

Abstract view of the system

The fire alarm monitoring system that comprises the following features. In an abstract view of the system user, there are three main components that the user directly interacts with.

Fire Sensor

The Fire Alarm sensor is the responsible component to measure the co2 level and smoke level of the nearby environment. Each fire sensor has a unique id which enables it to identify the sensor uniquely. Each Fire Sensor should be registered to the system before fixing it in some location. The system administrator is the person who has privileges to manage the Fire Alarm Sensors data, and the desktop client application provides an interface to control and monitor them remotely.

Desktop Client

The desktop client is an application that runs on a PC, and it enables to control and monitor the Fire Sensor details. It provides two privilege levels as guests and administrators. According to the privilege level, allowed features as follows.

Guest Login

- Monitor the Fire Sensor readings and status.
  (Smoke Level, CO2 Level, Whether the Fire Sensor is on or off)

Administrator Login

- Monitor the Fire Sensor readings and status.
  (Smoke Level, CO2 Level, Whether the Fire Sensor is on or off)

- Control Fire Sensor details
  (Register new Fire Sensors, Update existing Fire Sensor details, Remove existing Fire Sensors)

Web Client

The web client is an application hosted on a web server. It is accessible anywhere through the internet using any mobile device or desktop device that facilitates to access the internet. It also provides an

interface to monitor the Fire sensor current readings and status. However, it has neither privilege levels nor Fire Sensor controlling capability.

High level overview of the system architecture

The system has been developed based on Service-oriented architecture (SOA). Hence, this system consists of multiple, loosely coupled, reusable components to archive the service abstraction that describes in the previous section. Services are provided to clients by application components, through the HTTP protocol over the internet. Here is the high-level system architecture diagram of implemented system.

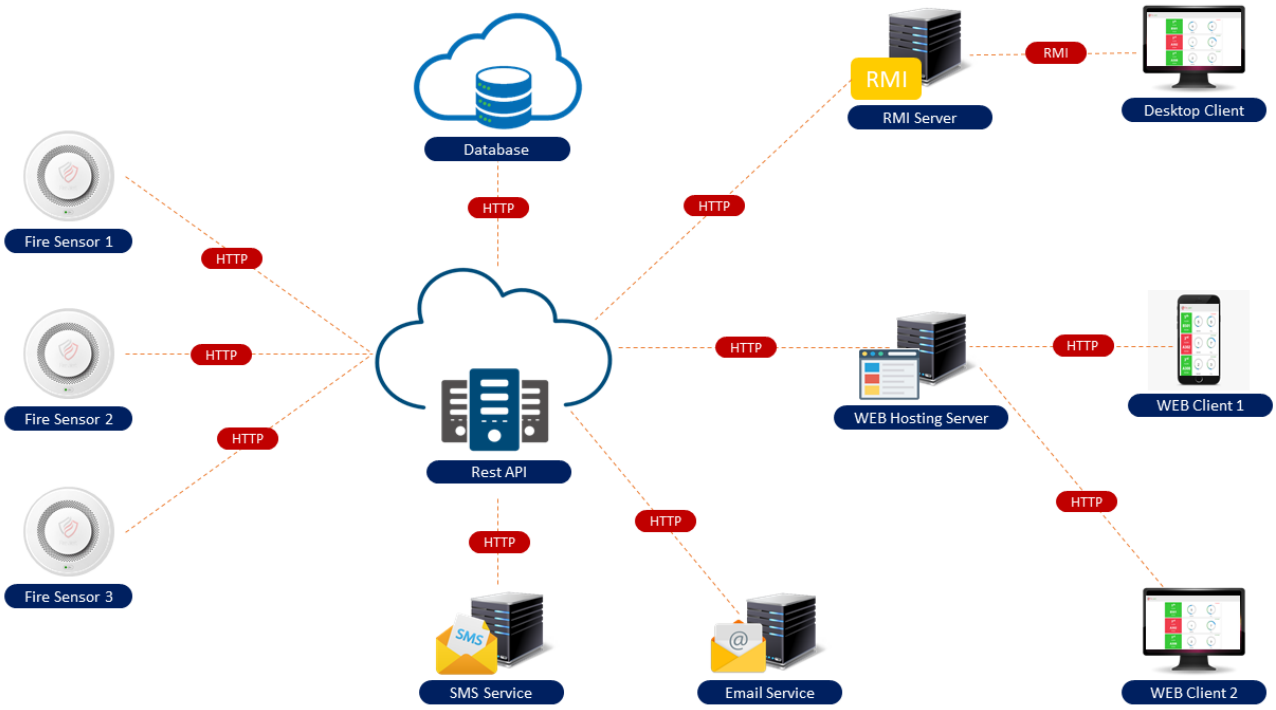


Figure 1: Fire Alert System Architecture

The main service provider component of the system is the REST API. It provides several endpoints to get the service to outside. All the Fire Sensors are connected, to the REST API over the Internet. There may have any number of sensors located in different locations. Fire Sensors send the smoke level, and the carbon dioxide level of it's the nearby environment repeatedly. These sensor readings are stores in a remote database which connected to the REST API, over the Internet. To simulate this real-world Fire Sensor, a client application has been implemented with the same behavior mentioned above.

Other major service of the system is the RMI server. It also connected to the REST API, over the Internet. RMI server is the responsible component to provide services to the Desktop client application and monitoring emergencies. In this system, when the smoke level or carbon dioxide level goes up more than

five out of ten, it is considering as an emergency. When an emergency occurs, it will send requests to the constrained API endpoints to notify the Administrator via SMS alert and Email notification.

As shown in figure 1, there are two third-party services connected to the REST API to handle Email and SMS sending functionality.

The Desktop client that connected to the RMI server is enabled to control Fire Sensor details with valid authentication. And also, it displays the current smoke level and the carbon dioxide level of each location that has a fixed Fire Sensor.

As shown in figure 1, there is another server named Web hosting server. That is the server that hosted the web client application. It is directly connected to the REST API. It also displays the current smoke level and the carbon dioxide level of each location that has a fixed Fire Sensor same as the desktop client. However, the significant characteristic of this is it is accessible from any device that enables to access the Internet.

Specification of Services and Clients

This chapter describes the detailed specification of services and client applications of the system. Figure 2 shows the full breakdown of the system components.

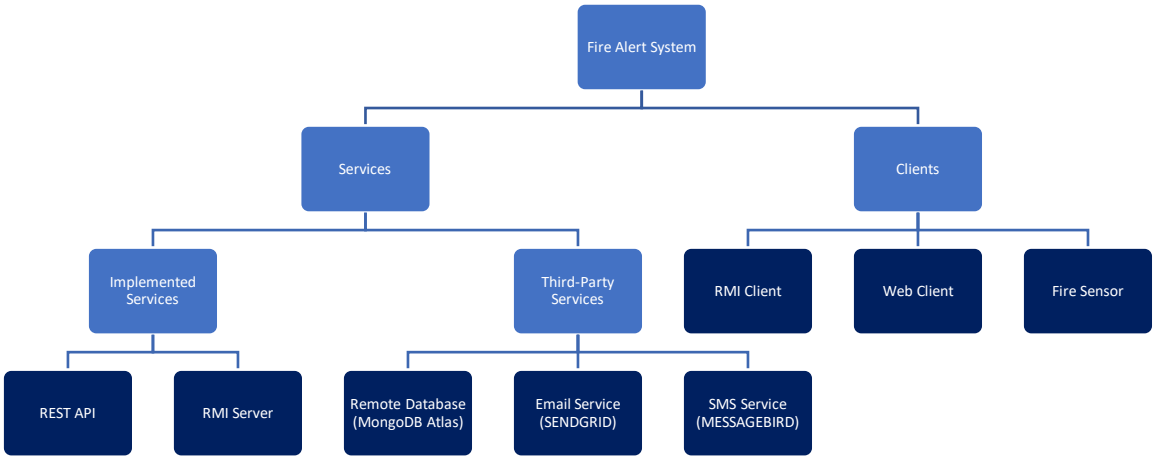


Figure 2: System Component Breakdown

REST API

Repository: https://github.com/ThamalDilanka/fire-alert-api

Hosted API: https://fire-alert-solution.herokuapp.com/

Overview and Technologies

RESTful web service has been developed as the primary service provider, using Node.js. This API handles all the service requests through the exposed endpoints. This API exposes several public endpoints and private endpoints to obtain services from it. The API is deployed in Heroku platform.

Express.js framework and Mongoose framework has been used, to handle the routing and database manipulations, respectively. To implement the authentication mechanism, the JSON Web Token (JWT), has been used. For automated processes like SMS and Email notification sending, a customized authentication mechanism has been used. Following sections provides a detailed description about the authentication and all the endpoints.

Exposed Interfaces to Outside

Since this API is hosted, https://fire-alert-solution.herokuapp.com/ URL could be used for access all the endpoints.

Open endpoints

Public endpoints do not require authentication to get services.

admin

POST URL/ api/v1/admin/signup

This endpoint responsible for register the admin to the system. The request must have the admin object with following attributes. Email should be a *valid email address* and the password should contain *at least 8 characters*. The following is a sample Admin Object that expected by the server

```
{
        "name": "Admin Name",
        "email": "adminemail@gmail.com",
        "password": "mypassword",
        "passwordConfirm": "mypassword"
}
```

POST URL/ api/v1/admin/login

This is the end point should be accessed in the administrator login. It Returns an access token if the given credentials are valid. The request must have an object with following attributes.

Sample Login Request Object

```
{
        "email": "adminemail@gmail.com",
        "password": "mypassword"
}
```

GET URL/ api/v1/admin/

This endpoint just returns admins *Name* and the *Email address*. It doesn't contain the encrypted password.

Sensors

GET	URL/	api/v1/sensors/

This endpoint returns all the sensor documents that have registered into the system.

GET	URL/	api/v1/sensors/<sensor-id>

This endpoint returns a one sensor document that specified as a query parameter.

sensorReadings

POST	URL/	api/v1/sensorReadings/<sensor-id>

This is the endpoint that responsible to add sensor readings to the database. The sensor id must be mentioned as a query parameter. The required attributes as follows for sensor reading object.

```
{
        "sensor": "5e8a13e64bc0b91a18ab6903",
        "reading": {
                "smokeLevel": 3,
                "co2Level": 5,
                "time": "2020-04-05T17:37:39.281Z"
        }
}
```

GET	URL/	api/v1/sensorReadings/<sensor-id>

This endpoint returns all the readings of one sensor that specified as a query parameter.

Protected endpoints

To access this end points, user should pass valid token in the request header along with the request. Once you successfully logged in to the system, it will send you a valid token. You may set that token to the request header as key value pair as follows. It must accompany this format.

KEY Authorization | **VALUE** Bearer<space><the-token-that-received-when-logged-in>

admin

PATCH URL/ api/v1/admin/<admin-id>

This endpoint allows only to update the Name and the Email Address of the admin.

Sensors

POST URL/ api/v1/sensors

A new sensor could be registered to the system by sending post request to this endpoint. The request body should contain a JSON object with following attributes as shown in the following sample document.

```
{
        "_id": "<sensor-id-that-mentioned-in-the-sensor>",
        "activated": true,
        "floor": "4th",
        "room": "X111"
}
```

PATCH URL/ api/v1/ sensors/<sensor-id>

Registered sensor details could be updated using this endpoint. The sensor id must be specified as a query parameter.

DELETE URL/ api/v1/sensors/<sensor-id>

This endpoint is responsible to remove the registered sensors from the system. The sensor id must be specified as a query parameter.

Special endpoints

Even though these endpoints are protected, these are accessed by automated processes in this system. Hence, it uses static authentication token that is pre-defined in the API. These endpoints only accept POST requests and the token must be included in the request header in following format.

KEY Authorization | **VALUE** Bearer<space><email-sending-token>

SMS

POST URL/ api/v1/sms

This endpoint is responsible to send a SMS to the given mobile number. It accepts JSON object with following attributes. The mobile number must start with the country code of the mobile.

```
{
        "to": "94xxxxxxxxx",
        "sensor": "5e8a13e64bc0b91a18ab6903"
}
```

The following is the sample text message sent from the API.

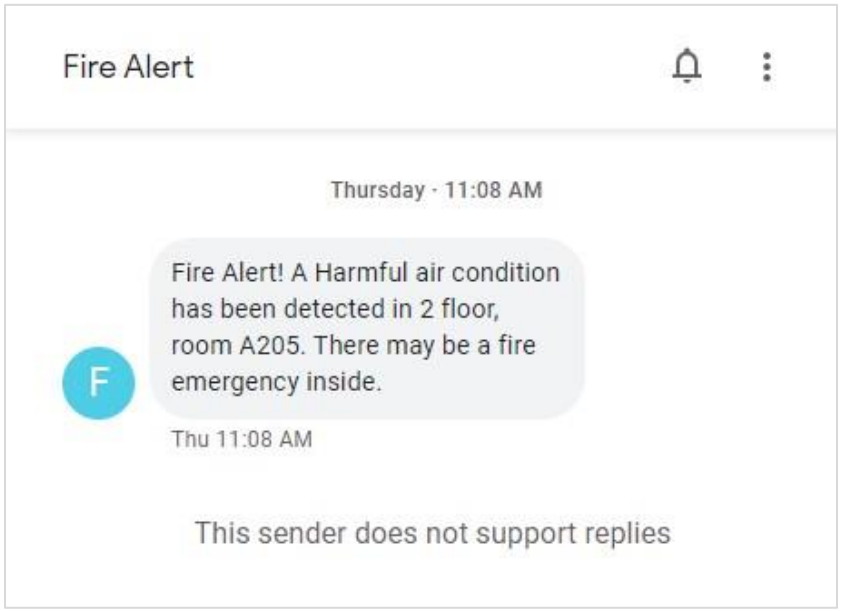


Figure 3: Sample SMS Alert

Email

POST	URL/	api/v1/email

This endpoint enables to send detailed email to the admin in case of emergency. It accepts a JSON object with following attributes.

```
{
        "to": "recipient@email.com",
        "sensor": "5e8a13e64bc0b91a18ab6903",
        "reading": {
                "smokeLevel": 4,
                "co2Level": 8,
                "time": "2020-04-05T17:37:39.281Z"
        }
}
```

A sample email that sent from the API is follows.

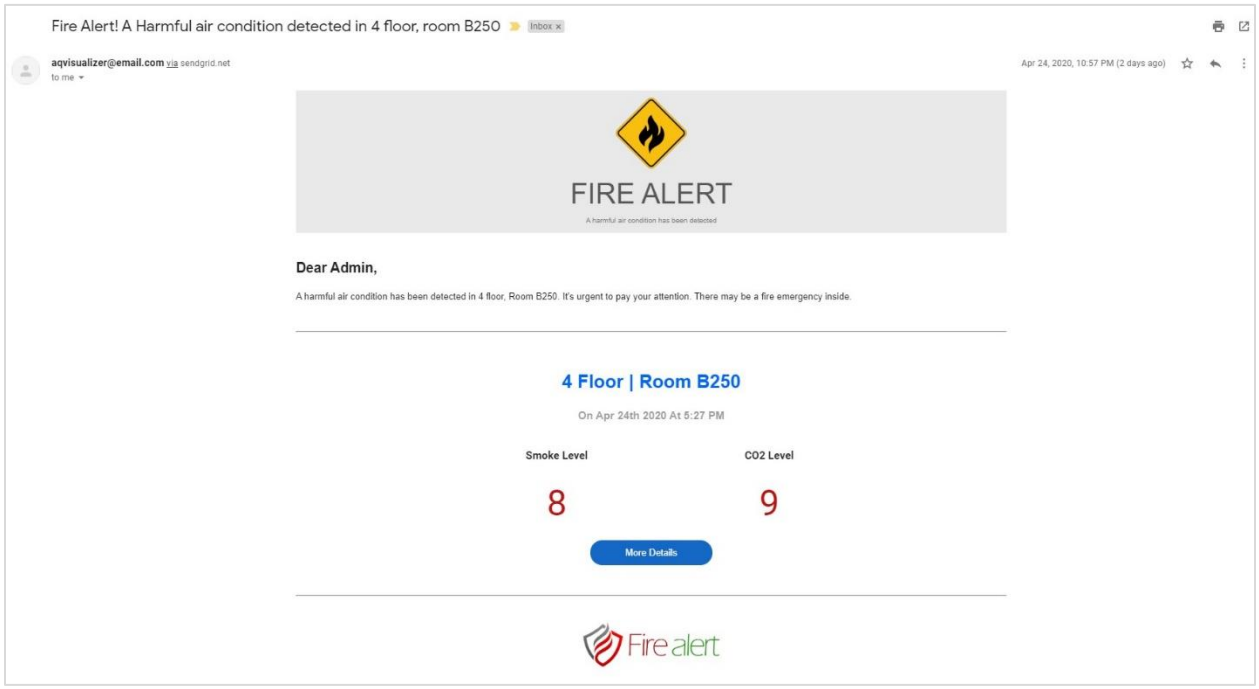


Figure 4: Sample Email

API Security Implementation

The API authentication has been implemented using JSON Web Tokens. All the protected endpoints require a valid token to get the service. The token should be included in the header of HTTP request in the following format. The token could be stored in a local storage and a token valid only 30 days.

KEY Authorization | **VALUE** Bearer<space><the-token-that-received-when-logged-in>

Email and SMS sending endpoints are used by the automated processes and they should work immediately in an emergency. By considering that facts, Email and SMS sending endpoints have been categorized as special endpoints of the API. These endpoints use pre-defined static token for authentication. It is required to include the token in the header of the HTTP request in the following format.

KEY Authorization | **VALUE** Bearer<space><email-sending-token>

RMI Server

Repository: https://github.com/Kavindu-Tharaka/fire-alert-RMI

Overview and Technologies

RMI server has been built using java as the programming language and java RMI as the RMI framework. As other main packages 'org.apache.http' , 'java.net.http' are used. Public Services of RMI server are exposed to outside through a public interface named 'RMIService' (given in below). An instance of RMI server is bound to RMI registry with the name 'AirSensorService' and RMI Server will be run on 'rmi://localhost:5099/AirSensorService' .

Apart from above mentioned public services, RMI Server has its own private services as well. Those services are not exposed through above mentioned public interface. Therefore, desktop client cannot access those services. RMI server is the one which dealing directly with the REST API to get, set, update, delete data.

RMI server checks the statuses of the fire alarm sensors repeatedly. The interval of repeating is 5 seconds. If the CO_2 level or Smoke level goes above level 5, RMI server will send emails and SMSs to the admin through third party services. RMI server will call to REST API and REST API will call to email and SMS services. Desktop client get details of the fire alarm sensors for each 30 seconds through RMI Server by invoking the public services that are declared in the above-mentioned public interface.

Exposed Interfaces to Outside

```
public interface RMIService extends java.rmi.Remote{
    public String getAllSensorDetails() throws java.rmi.RemoteException;
    public String loginValidator(String email, String password);
    public boolean addSensor(String id, int floor, String room);
    public boolean editSensor(String id, int floor, String room);
    public boolean deleteSensor(String id) throws java.rmi.RemoteException;
}
```

Above mentioned public services are used by desktop client.

Method	**Description**
getAllSensorDetails()	Get fire alarm sensor details
loginValidator()	Authenticate admin login credentials
addSensor()	Register new fire alarm sensor
editSensor()	Edit an existing fire alarm sensor's detail
deleteSensor()	Delete an existing fire alarm sensor

RMI Desktop Client

Repository: https://github.com/Kavindu-Tharaka/fire-alert-RMI

Overview

Public Services of RMI server are exposed to outside through a public interface are accessible here. Example of desktop client accessing those public services of RMI server is given below as a code snippet.

```
RMIService service;
String result = null;

try {
    service = (RMIService) Naming.lookup("rmi://localhost:5099/AirSensorService");
    result = service.loginValidator(email, password);

} catch (MalformedURLException | RemoteException | NotBoundException ex) {
    //handle exceptions here
}
```

As the main functionality, more similarly to the web client, desktop client also get details of the fire alarm sensors for each 30 seconds through RMI Server by invoking the public services that are declared in a publicly exposed interface of the RMI server named 'RMIService'.

There are two user levels.

1. Admin Login
   With the admin login user can register a new sensor and edit, delete existing sensors.
2. Guest Login
   With the guest login user can only see the details of sensors.

Technologies

Desktop client has been built using java as the programming language and java RMI as the RMI framework. Java swing is used to create the User Interfaces of the desktop client.

Sample user interfaces of the desktop client application are follows.

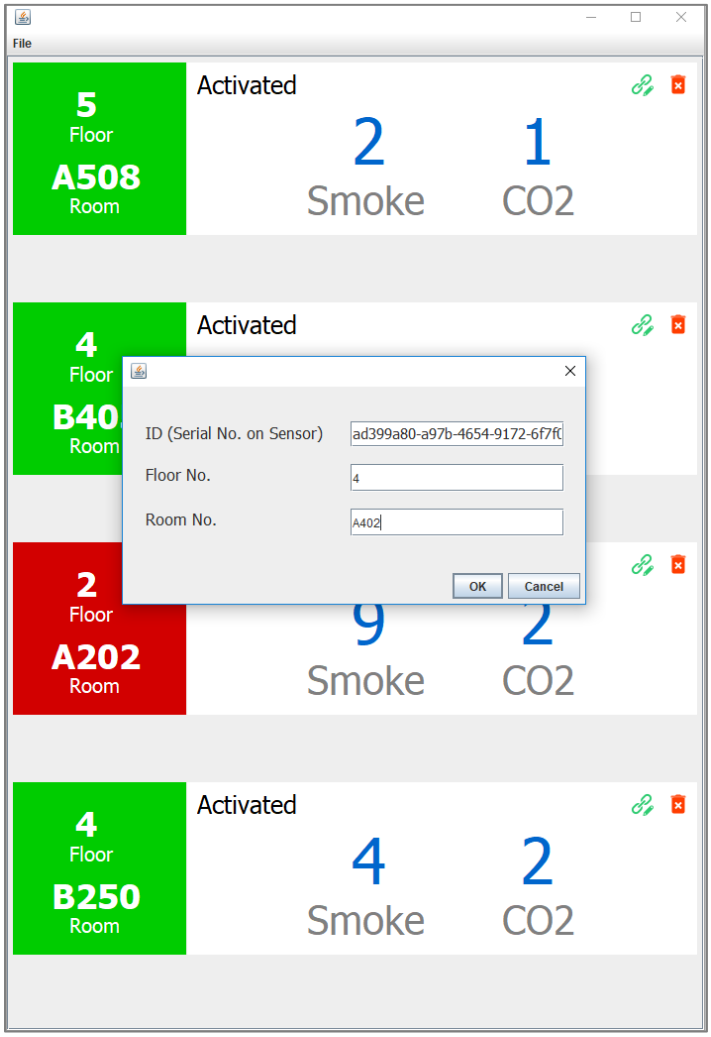


Figure 7: Desktop Client Dashboard

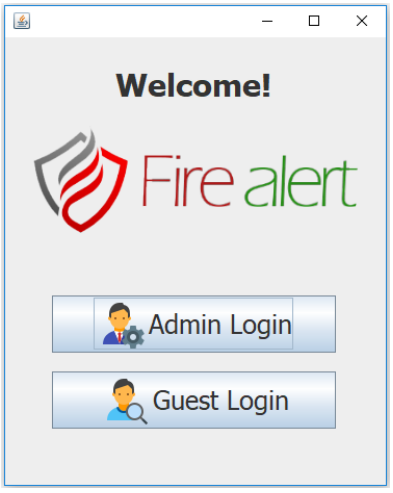


Figure 6: Desktop Client Login

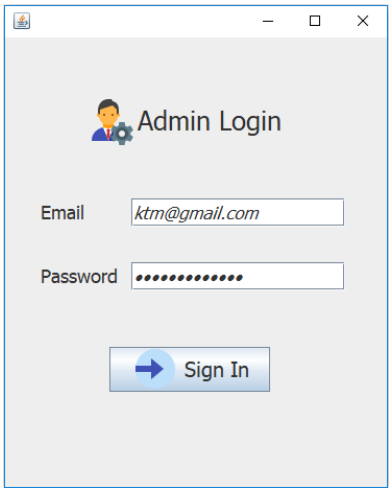


Figure 5: Admin Login

Web Client

Repository: https://github.com/ThamalDilanka/fire-alert-web

Hosted Web Client: https://thamaldilanka.github.io/fire-alert-web/

Overview

As specified in the requirements, the Web Client provides an interface to monitor the registered Fire Alarm Readings and status of them. It updates every 40 seconds asynchronously. The following is the UI of the web client application.

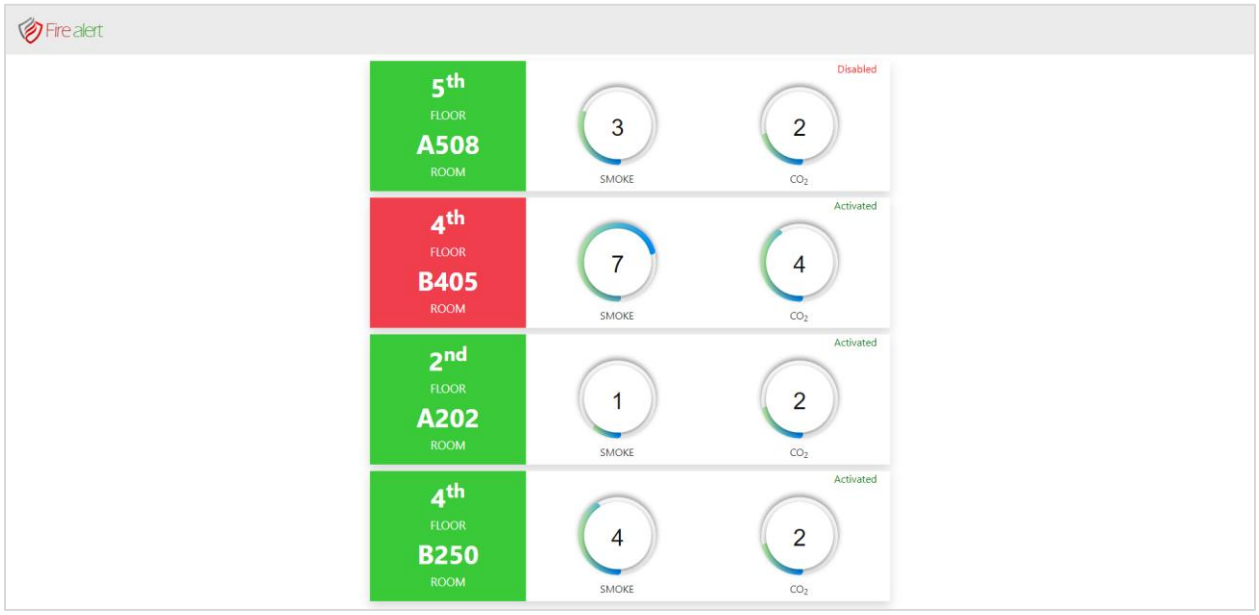


Figure 8: Web Client Application UI

Technologies

The web client application has been implemented using React library. It uses React version 16.13.1. To make HTTP requests from the API, *axios* library has been used. Bootstrap and Apex chart dependencies are used to styling and smoke and carbon dioxide level graphical visualization, respectively. The live version of the Web client has been deployed in GitHub Pages platform.

Fire Sensor Simulator

Repository: https://github.com/ThamalDilanka/fire-alert-sensor

Note: Packaged version (.exe file) of the fire sensor simulator application, is contains in *dist* directory of the repository.

Overview

A desktop client application has been developed to simulate a real Fire Sensor. All the sensor application has a unique ID and when the sensor is registered to the system, the fire sensor sends readings the API in every 10 seconds. It enables to change the smoke level and carbon dioxide level manually. The following figure shows the UI of the fire sensor simulator application.

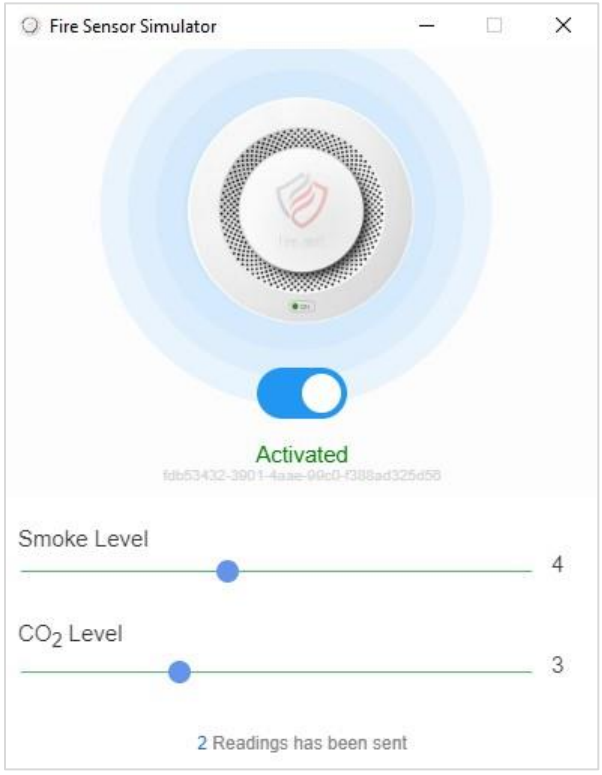


Figure 10:Fire Sensor Simulator - Activated

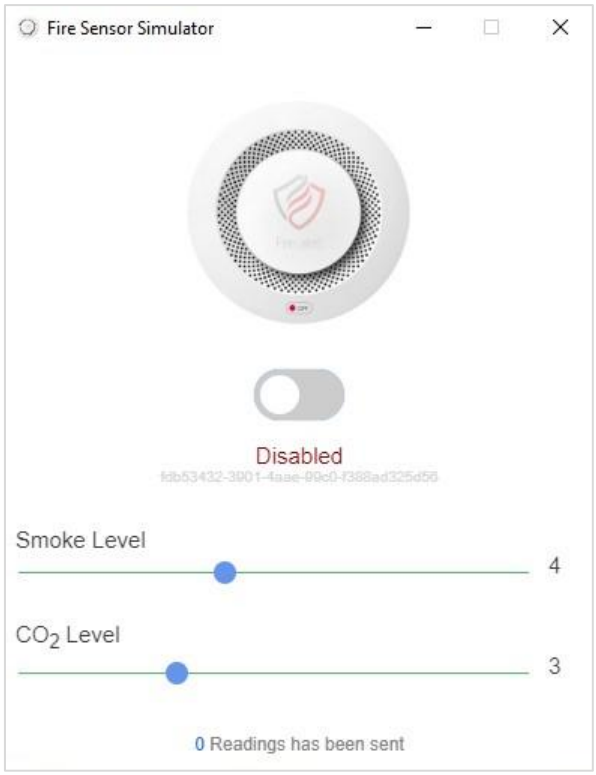


Figure 9: Fire Sensor Simulator - Disabled

Technologies

Electron framework is used to implement the fire sensor simulator. It enables to build cross platform desktop applications with JavaScript. HHTP requests are handled using axios library. Electron-Packager library used for the packaging the application.

Third Party Services

In the implementation, third party services have been used, to store data, send Emails, and SMS notifications to the user. These services are integrated, into the REST API. The following chapter describes the service details and the integration.

Cloud Database

MongoDB Atlas: https://www.mongodb.com/

This system is required to store the data generated by the system in some database, and it should have quick and concurrent data access. Hence, MongoDB has been used to store data. MongoDB is the most popular No-SQL database, and they provide an online platform called MongoDB Atlas to manage the databases. The System API connects to the MongoDB Atlas database though the Internet. Here is the essential integration logic of a MongoDB atlas database with mongoose framework.

```
// Importing mongoose library
const mongoose = require('mongoose');
// Connecting to the database
mongoose.connect(DB, {
    useNewUrlParser: true,
    useCreateIndex: true,
    useFindAndModify: false,
    useUnifiedTopology: true,
})
.then(() => console.log('Connected to the mongoDB successfully'))
.catch((err) => {
    console.log(
        'Something went wrong. Check your connection and credentials'
    );
});
```

Since MongoDB is No-SQL type database, Generated data stores in three main collections of the database as follows. Even the collections are natively consisting with JSON objects, by considering the convenient of referencing, collections have been presented in tabular format instead of using JSON objects.

- admins
- sensors
- Sensor-readings

_id ObjectId	name String	email String	password String
“5e9db379ebd24500178d972b”	"Thamal Dilanka"	"dev.thamal@gmail.com"	"$2a$12$XhSQgCPlEPv3joxPDSuzR.dSuYvYddyfXp4hyp9eiavG7nULF9/TW"

Table 1: admins collection

_id ObjectId	floor Int32	room String	activated Boolean	lastReading Object
"1ghb6ec3-7b69-4ea1-8c1a-8bb78e1e59d"	5	“A503”	true	"smokeLevel": 4, "co2Level": 2, "time": "Sun Apr 26 2020 09:13:53 GMT+0000 (Coordinated Universal Time)"
"d4eb52c1-139e-4ac6-a2ee-e5552f0f96c4"	6	“B608”	true	"smokeLevel": 2, "co2Level": 4, "time": "Sun Apr 26 2020 09:13:53 GMT+0000 (Coordinated Universal Time)"
"78gjhgjhjjgfj-7b69-4ea1-8c1a-8bb78e1e59d"	3	“A302”	false	"smokeLevel": 1, "co2Level": 2, "time": "Sun Apr 26 2020 09:13:53 GMT+0000 (Coordinated Universal Time)"
"gbfdrbdsdt47-139e-4ac6-a2ee-e5552f0f96c4"	4	“B409”	true	"smokeLevel": 5, "co2Level": 1, "time": "Sun Apr 26 2020 09:13:53 GMT+0000 (Coordinated Universal Time)"

Table 2: sensors collection

_id ObjectId	sensor ObjectId	reading Object
"1ghb6ec3-7b69-4egr-8c1a-8bb78e1e5dr"	"1ghb6ec3-7b69-4ea1-8c1a-8bb78e1e59d"	"smokeLevel": 4, "co2Level": 2, "time": "Sun Apr 26 2020 09:13:53 GMT+0000 (Coordinated Universal Time)"
"d4eb52c1-139e-4ac6-a2ee-e5552f0f96cn"	"d4eb52c1-139e-4ac6-a2ee-e5552f0f96c4"	"smokeLevel": 2, "co2Level": 2, "time": "Sun Apr 26 2020 09:13:53 GMT+0000 (Coordinated Universal Time)"
"gfd4eb52c1-139e-4ac6-a2ee-e5552f0fgfdgr"	"4hfdrbdsdt47-139e-4ac6-a2ee-e5552f0f9rbtd"	"smokeLevel": 3, "co2Level": 2, "time": "Sun Apr 26 2020 09:13:53 GMT+0000 (Coordinated Universal Time)"
"gfdr52c1-139e-4ac6-a2ee-e55gfdsgdf43"	"gbfdrbdsdt47-139e-4ac6-a2ee-e5552f0f9gfdr"	"smokeLevel": 2, "co2Level": 2, "time": "Sun Apr 26 2020 09:13:53 GMT+0000 (Coordinated Universal Time)"
"d4eb52c1-139e-4ac6-a2ee-e5552f0fgftr4"	"b5fdrbdsdt47-139e-4ac6-a2ee-e5552f0f965g"	"smokeLevel": 5, "co2Level": 2, "time": "Sun Apr 26 2020 09:13:53 GMT+0000 (Coordinated Universal Time)"

Table 3: sensor-readings collection

Email Service

 SendGrid: https://sendgrid.com/

This system uses SendGrid email service to handle the Email Notifications. It provides the full optimized email sending solution as an API. And also, it provides their library thought the Node Package Manger (NPM). The following is the integration code, using Twilio SendGrid's v3 Node.js library, that used in the Fire Alert REST API.

```
const sendGrid = require('@sendgrid/mail');
sendGrid.setApiKey(process.env.SENDGRID_API_KEY);
const msg = {
    to: 'admin@email.com',
    from: 'aqvisualizer@email.com',
    subject: 'Fire Alert',
    text: message,
    html: '<div>Template<div>',
};
sendGrid.send(msg);
```

SMS Service

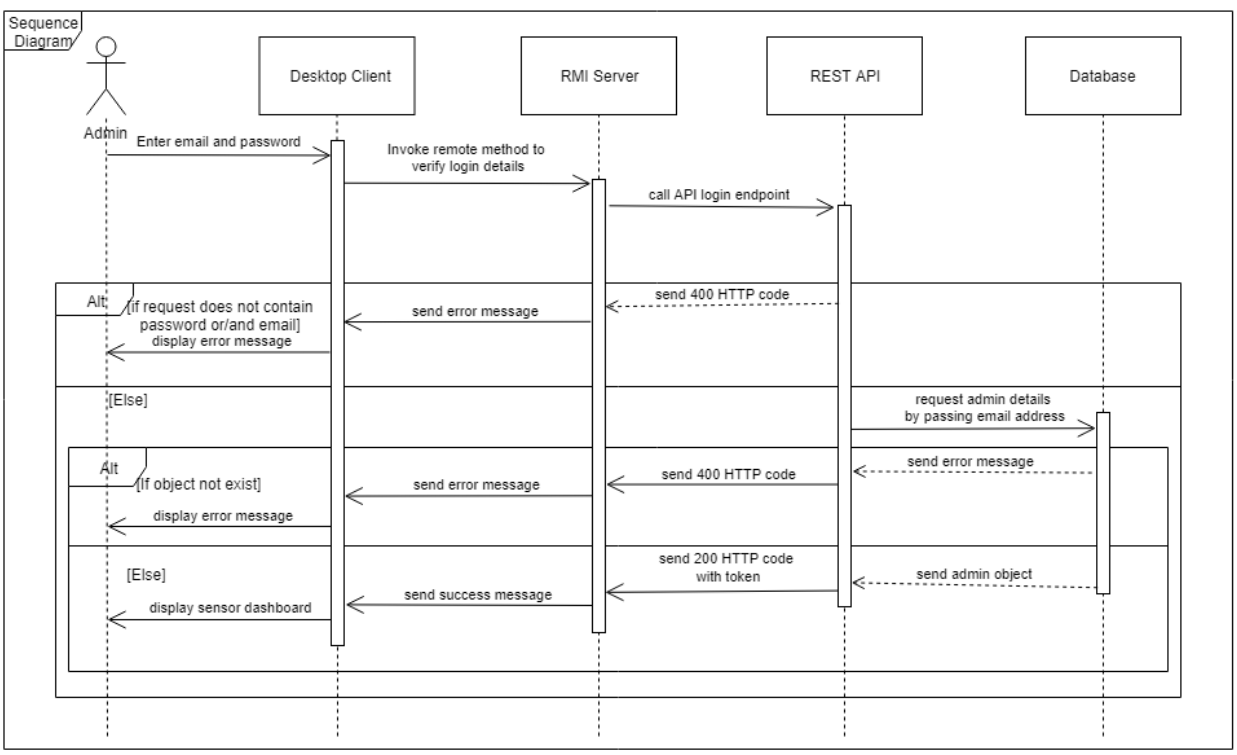
 MessageBird: https://www.messagebird.com/en/

Message Bird service is used to send SMS notifications in the system. This is a complete solution that contains customer service software, developer APIs, and global carrier connectivity to power business communications. In this project uses their API and node library thought the Node Package Manger (NPM) for integration. The following is the integration logic which uses *messagebird* NPM library that used in Fire Alert API.

```
const messagebird = require('messagebird')(process.env.ACCESS_KEY);
messagebird.messages.create({
    originator : 'Fire Alert',
    recipients : [ '94xxxxxxxx' ],
    body : 'message'
});
```

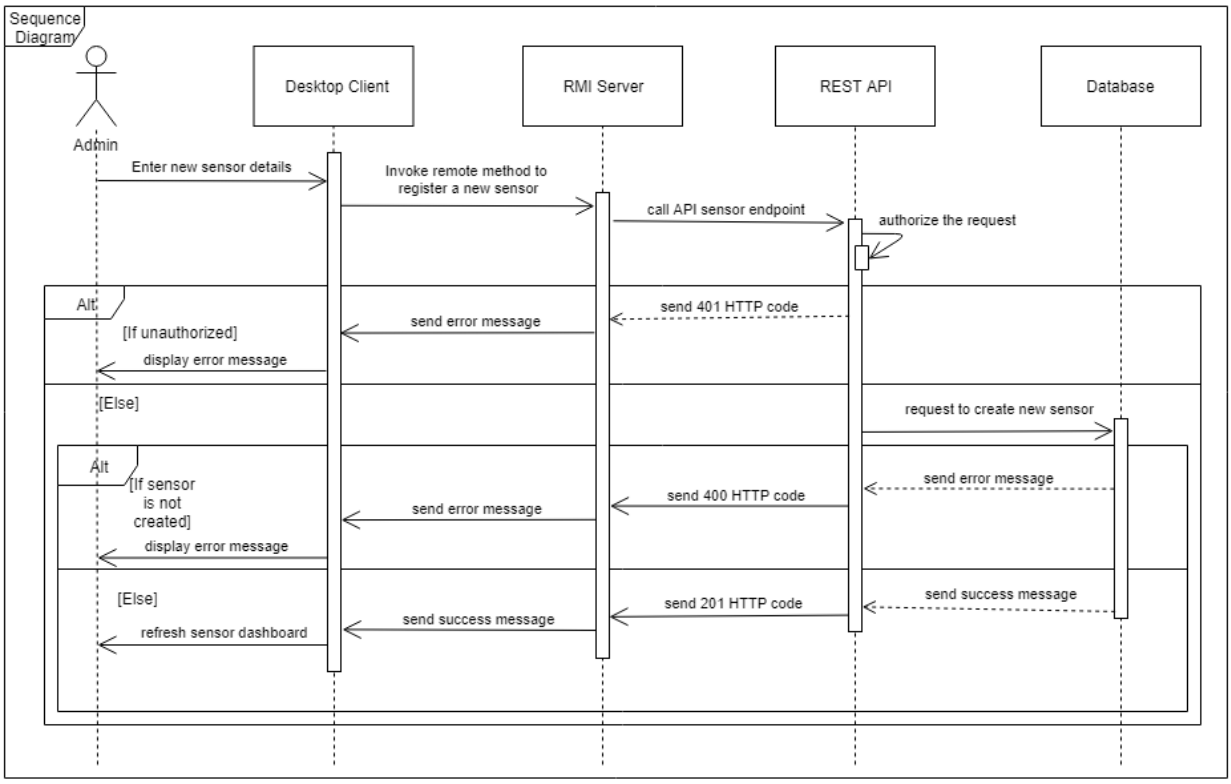
Workflows of The System

Admin login



1. Admin enters the login details. (email, password)
2. Desktop client invokes the remote method that are implemented in RMI server which is related with verify admin login. → `loginValidator(String email, String password)`
3. RMI Server calls REST API endpoint as a POST request → `https://fire-alert-solution.herokuapp.com/api/v1/admin/login`
4. If request does not contain password or/and email, then
    a. REST API send 400 bad request status code to the RMI Server.
    b. RMI Server sends an error message to the desktop client.
    c. Desktop client shows the error message to the admin.
5. if request does not contain password or/and email, then
    a. REST API requests admin details by passing email to the database.
    b. If the admin object is existing, then
        i. Database sends success message to the REST API.
        ii. REST API sends 200 OK status code to the RMI SERVER.
        iii. RMI Server sends a success message to the desktop client.
        iv. Desktop client shows the sensor dashboard to the admin.
    c. If the admin object is not existing, then
        i. Database sends error message to the REST API.
        ii. REST API sends 400 bad request status code to the RMI SERVER.
        iii. RMI Server sends an error message to the desktop client.
        iv. Desktop client shows the error message to the admin.

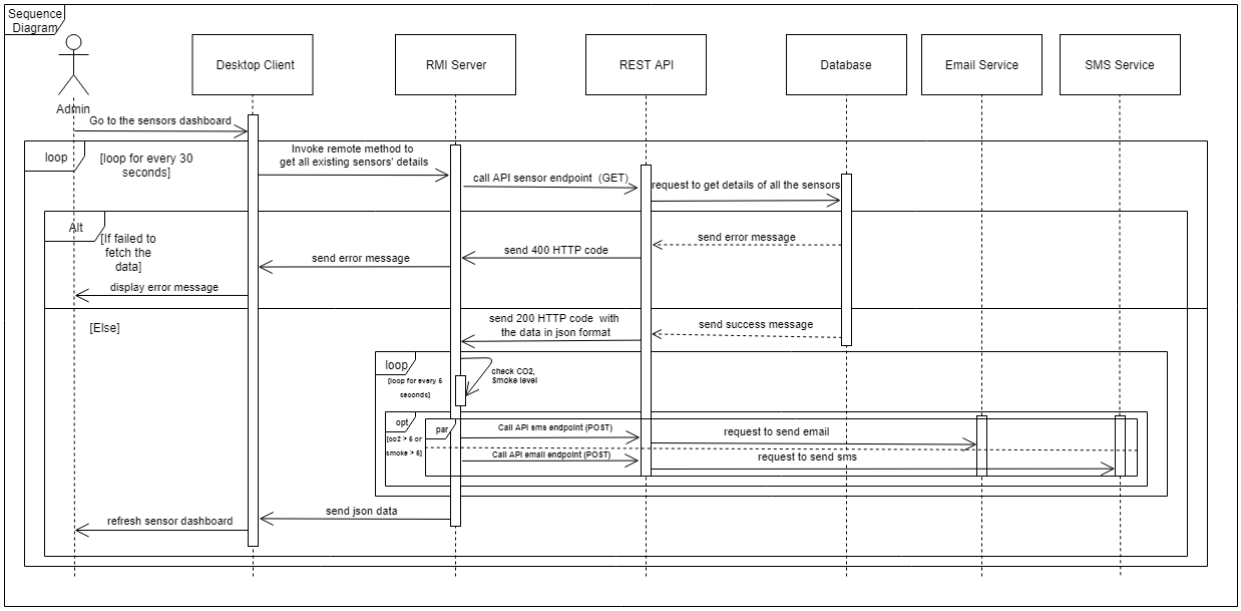
Register a new Fire Sensor



1. Admin enters the details of the new sensor. (serial number of the sensor as the ID, Floor and Room)
2. Desktop client invokes the remote method that are implemented in RMI server which is related with registering a new sensor. → `addSensor(String id, int floor, String room)`
3. RMI Server calls REST API endpoint as a POST request → `https://fire-alert-solution.herokuapp.com/api/v1/sensors`
4. REST API call a method that is inside REST API to verify the request → `protect = async (req, res, next)`
5. If the request is unauthorized, then
    a. REST API send 401 unauthorized status code to the RMI Server.
    b. RMI Server sends an error message to the desktop client.
    c. Desktop client shows the error message to the admin.
6. If the request is authorized, then
    a. REST API requests to create a new server to the database. → `Sensor.create(req.body)`
    b. If the sensor object is created, then
        i. Database sends success message to the REST API.
        ii. REST API sends 201 created status code to the RMI SERVER.
        iii. RMI Server sends a success message to the desktop client.
        iv. Desktop client refreshes the sensor dashboard.
    c. If the sensor object is not created, then

i. Database sends error message to the REST API.
ii. REST API sends 400 bad request status code to the RMI SERVER.
iii. RMI Server sends an error message to the desktop client.
iv. Desktop client shows the error message to the admin.

Display Fire Sensor details in the RMI desktop client



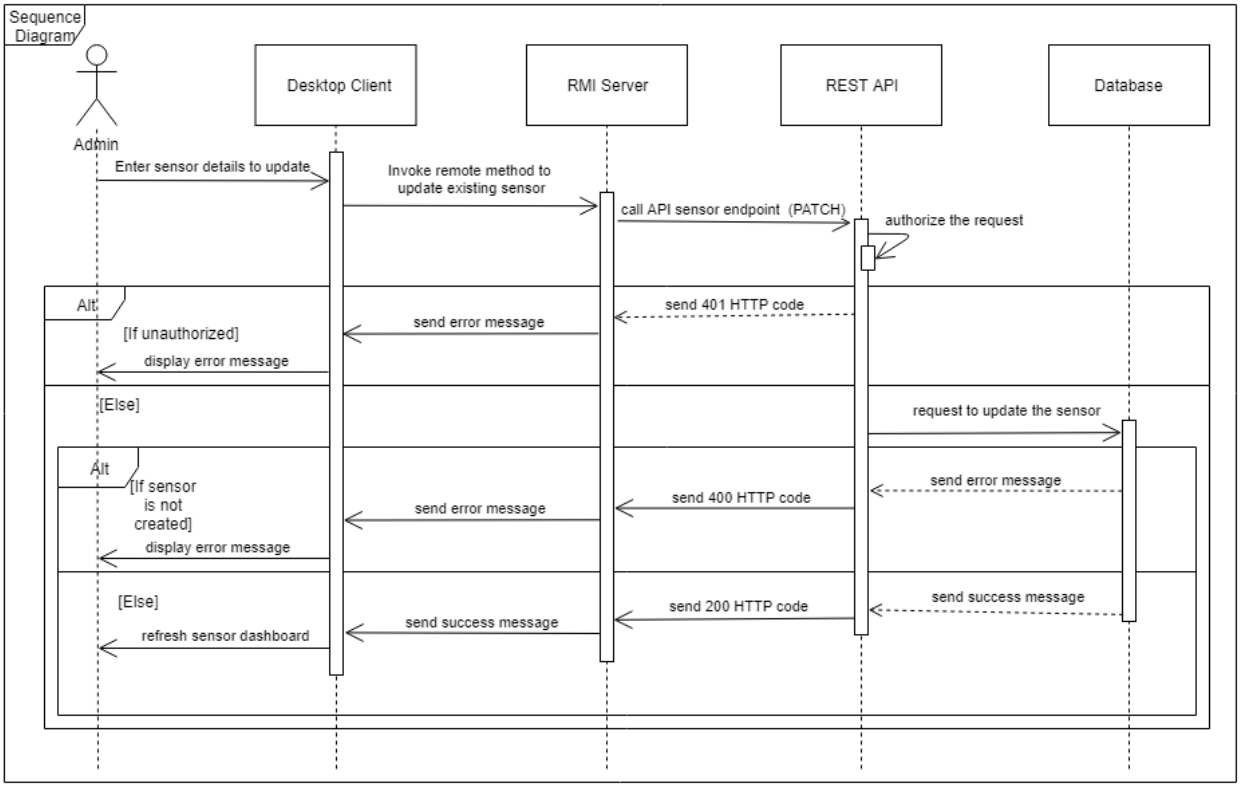
1. Admin opens sensor dashboard.
2. Desktop client invokes remote method that is related with get all existing sensors' details which is implemented in the RMI Server.(once every 30 seconds) → `getAllSensorDetails()`
3. RMI Server calls REST API endpoint as a GET request → `https://fire-alert-solution.herokuapp.com/api/v1/sensors/`
4. REST API requests to get all sensors' details, to the database. → `Sensor.find(req.query)`
5. If failed to fetch data form the database, then
    a. Database sends error message to the REST API.
    b. REST API sends 400 bad request status code to the RMI SERVER.
    c. RMI Server sends an error message to the desktop client.
    d. Desktop client shows the error message to the admin.
6. If fetch data form the database, then
    a. Database sends success message to the REST API.
    b. REST API sends 200 OK status code with the data in json format to the RMI SERVER.
    c. RMI Server checks for whether CO2 or/and smoke level is above level 5.(once every 5 seconds)
        i. If CO2 or/and smoke level is above level 5, then
            1. RMI Server calls two REST API endpoints as a POST requests →
                a. `https://fire-alert-solution.herokuapp.com/api/v1/email`

b. https://fire-alert-solution.herokuapp.com/api/v1/sms

d. RMI Server sends json data to the desktop client

e. Desktop client populate the sensor dashboard → `populateSensorComponents(String responseBody)`

Update a registered Fire Sensor

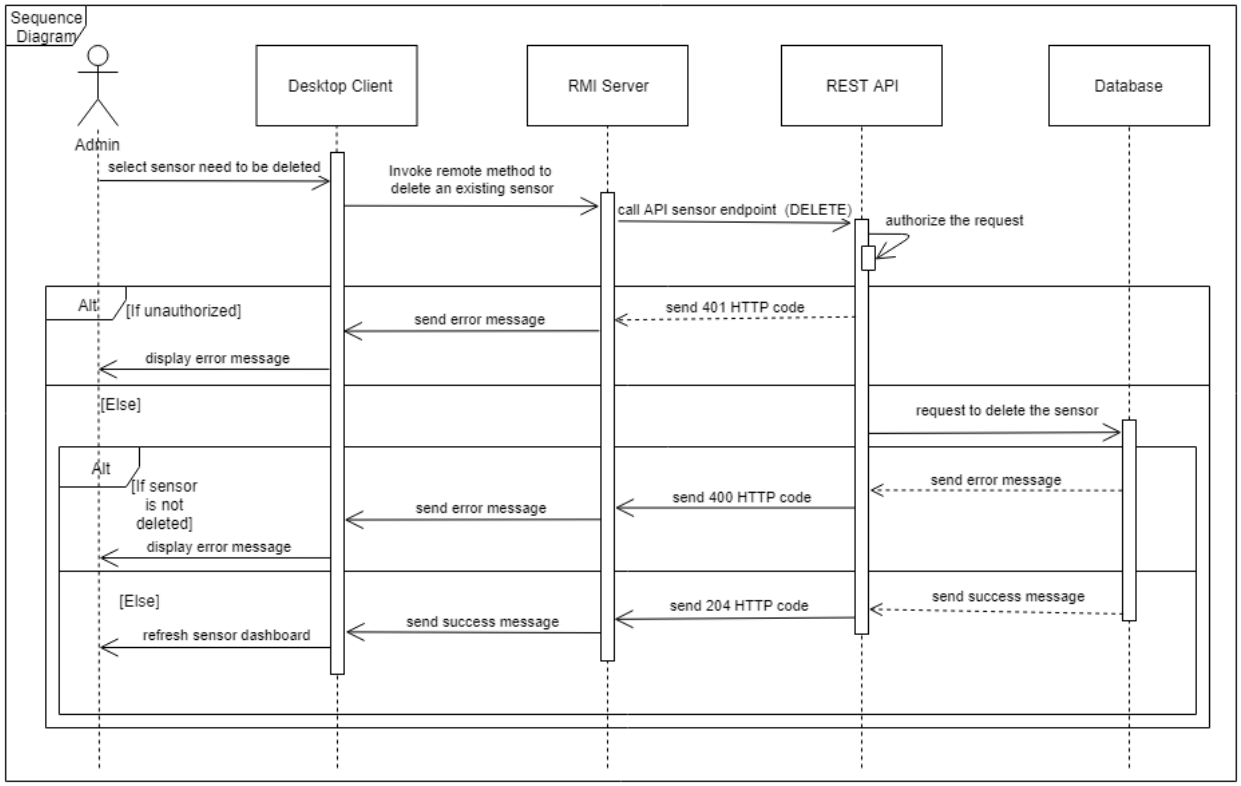


1. Admin selects the sensor that is needed to be updated.
2. Desktop client invokes the remote method that are implemented in RMI server which is related with deleting a sensor. → `editSensor(String id, int floor, String room)`
3. RMI Server calls REST API endpoint as a PATCH request → https://fire-alert-solution.herokuapp.com/api/v1/sensors/<sensor-id>
4. REST API calls a method that is inside REST API to verify the request → `protect = async (req, res, next)`
5. If the request is unauthorized, then
    a. REST API send 401 unauthorized status code to the RMI Server.
    b. RMI Server sends an error message to the desktop client.
    c. Desktop client shows the error message to the admin.
6. If the request is authorized, then
    a. REST API requests to create a new server to the database. → `Sensor.findByIdAndUpdate(req.params.id, req.body)`
    b. If the sensor object is updated, then

i. Database sends success message to the REST API.
ii. REST API sends 200 OK status code to the RMI SERVER.
iii. RMI Server sends a success message to the desktop client.
iv. Desktop client refreshes the sensor dashboard.

c. If the sensor object is not updated, then
i. Database sends error message to the REST API.
ii. REST API sends 400 bad request status code to the RMI SERVER.
iii. RMI Server sends an error message to the desktop client.
iv. Desktop client shows the error message to the admin.

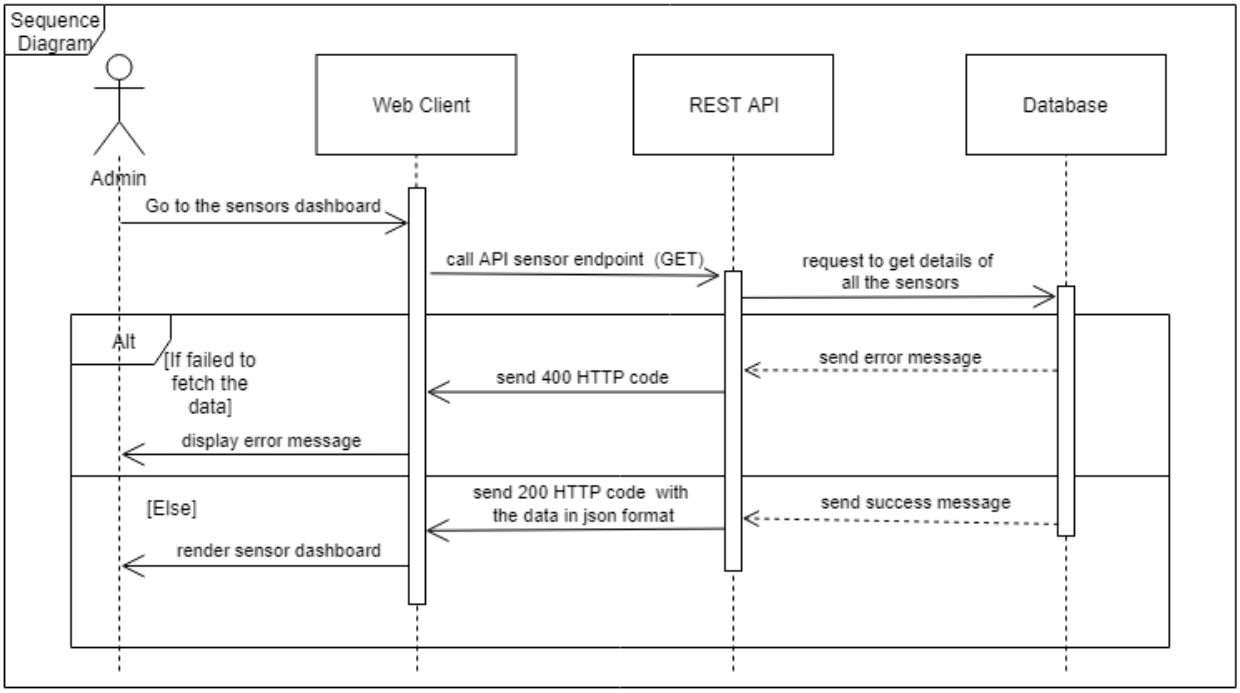
Delete a registered Fire Sensor



1. Admin selects the sensor that is needed to be deleted.
2. Desktop client invokes the remote method that are implemented in RMI server which is related with deleting a sensor. → `deleteSensor(String id)`
3. RMI Server calls REST API endpoint as a DELETE request → `https://fire-alert-solution.herokuapp.com/api/v1/sensors/<sensor-id>`
4. REST API calls a method that is inside REST API to verify the request → `protect = async (req, res, next)`
5. If the request is unauthorized, then
   a. REST API send 401 unauthorized status code to the RMI Server.
   b. RMI Server sends an error message to the desktop client.
   c. Desktop client shows the error message to the admin.

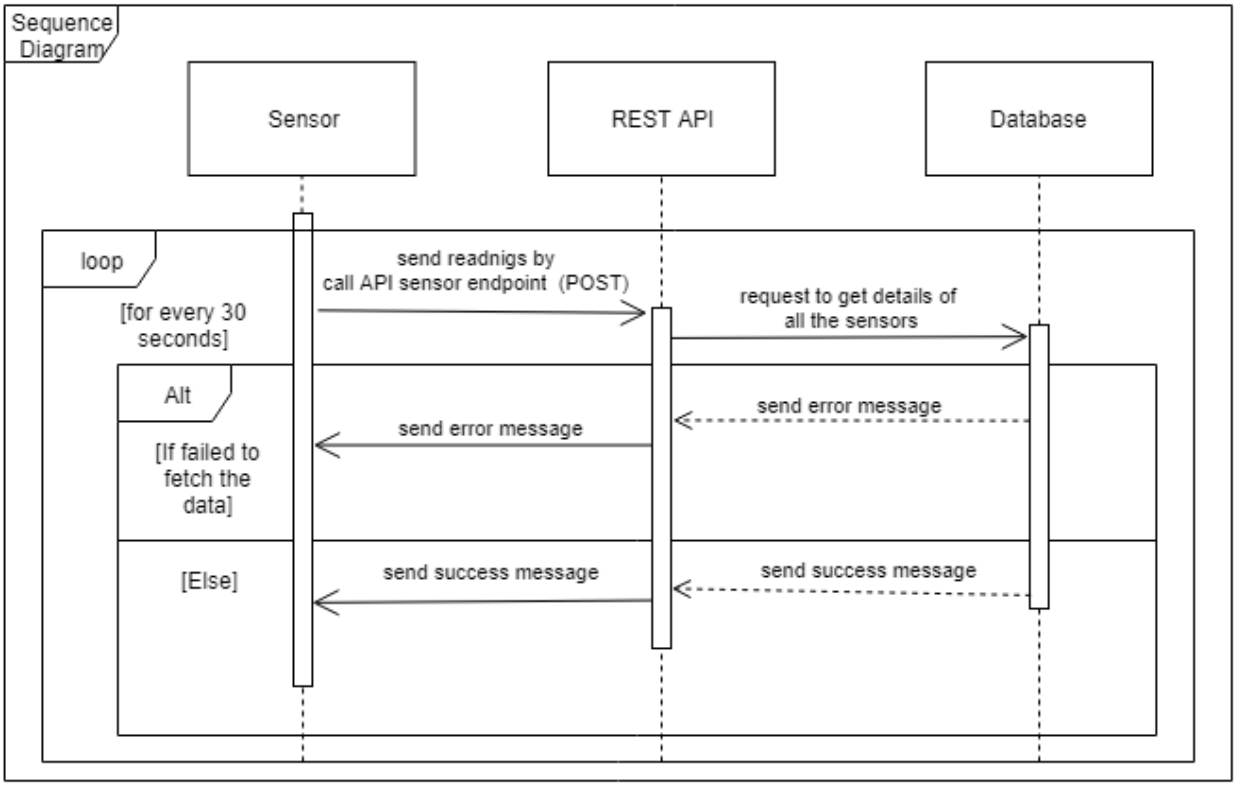
6. If the request is authorized, then
    a. REST API requests to create a new server to the database. → `Sensor.findByIdAndDelete(req.params.id)`
    b. If the sensor object is deleted, then
        i. Database sends success message to the REST API.
        ii. REST API sends 204 No content status code to the RMI SERVER.
        iii. RMI Server sends a success message to the desktop client.
        iv. Desktop client refreshes the sensor dashboard.
    c. If the sensor object is not deleted, then
        i. Database sends error message to the REST API.
        ii. REST API sends 400 bad request status code to the RMI SERVER.
        iii. RMI Server sends an error message to the desktop client.
        iv. Desktop client shows the error message to the admin.

Display Fire Sensor details in the web client



1. Admin open the web client application though the web browser
2. The web client application calls the REST API endpoint as a GET request -> → `https://fire-alert-solution.herokuapp.com/api/v1/sensors`
3. The REST API call the FIND method to retrieve the data set from the remote database.
4. If the request was success
    a. Database sends JSON array with sensor details to the REST API
    b. REST API sends JSON array with the sensor details to the Web client application
5. If the request was unsuccessful
    a. REST API sends error message to the Web Client application.

Sending Fire Sensor readings to the API



1. The Fire Sensor Application calls the REST API endpoint as a POST request -> → `https://fire-alert-solution.herokuapp.com/api/v1/sensors/<sensor-id>`
2. The REST API call the SAVE method to insert the data set into the remote database.
3. If the request was success
    b. Database sends JSON object with stored sensor reading object
    c. REST API sends JSON object with stored sensor reading object to the fire alert application
4. If the request was unsuccessful
    d. REST API sends error message to the Web Client application.

Appendix

REST API

adminController.js

```
const Admin = require('../models/Admin');

// Returns the admin object without password
exports.getAdmin = async (req, res, next) => {
    try {
        const admin = await Admin.find();

        res.status(201).json({
            status: 'success',
            data: {
                admin,
            },
        });
    } catch (err) {
        res.status(400).json({
            status: 'failed',
            message: err.message,
        });
    }
};

// Updates the admin details
exports.updateAdmin = async (req, res, next) => {
    try {
        const admin = await Admin.findByIdAndUpdate(req.params.id, req.body, {
            new: true,
            runValidators: true,
        });

        res.status(200).json({
            status: 'success',
            data: {
                admin,
            },
        });
    } catch (err) {
        res.status(400).json({
            status: 'failed',
            message: err.message,
```

```
        });
    }
}
```

authController.js

```
const { promisify } = require('util');
const Admin = require('./../models/Admin');
const jwt = require('jsonwebtoken');

// Build a JWT
const signToken = (id) => {
    return jwt.sign({ id }, process.env.JWT_SECRET, {
        expiresIn: process.env.JWT_EXPIRES_IN,
    });
};

// signup endpoint handler
exports.signup = async (req, res, next) => {
    try {
        let newAdmin = await Admin.create({
            name: req.body.name,
            email: req.body.email,
            password: req.body.password,
            passwordConfirm: req.body.passwordConfirm,
        });

        // Get the sign token
        const token = signToken(newAdmin._id);

        // Remove password from admin
        newAdmin.password = undefined;

        // Send back the response
        res.status(201).json({
            status: 'success',
            token,
            data: {
                admin: newAdmin,
            },
        });
    } catch (err) {
        res.status(400).json({
            status: 'failed',
```

```
            message: err.message,
        });
    }
};

// login endpoint handler
exports.login = async (req, res, next) => {
    try {
        // getting email and password from the request body
        const { email, password } = req.body;

        // Check email is exists in request
        if (!email || !password) {
            res.status(400).json({
                status: 'failed',
                message: 'Please enter email and password',
            });

            // Call the next middleware and terminate the function
            return next();
        }

        // Find the admin from the database with given email
        const admin = await Admin.findOne({ email }).select('+password');

        // Check the hashed password with given password and assign the result to
 correct variable
        const correct = await admin.checkPassword(password, admin.password);

        // Check whether the existence of admin or correct values
        if (!admin || !correct) {
            // Sending response to the user if given credentials are invalid
            res.status(400).json({
                status: 'failed',
                message: 'Invalid Credentials',
            });

            // Call the next middleware and terminate the function
            return next();
        }

        // Send token to user
        const token = signToken(admin._id);

        // Sending response with Json Web Token to the user
```

```
        res.status(200).json({
            status: 'success',
            token,
        });
    } catch (err) {
        // Send error message if error occurs
        res.status(400).json({
            status: 'failed',
            message: err.message,
        });
    }
};

// The protect middle ware that assignable for any route
exports.protect = async (req, res, next) => {
    try {
        // Get the token from request header
        let token = getTokenFromRequest(req);

        // Check whether is in correct format
        if (!token) {
            // Sending response if token is not valid
            res.status(401).json({
                status: 'failed',
                message: 'You are not logged in',
            });

            // terminate the function
            return;
        }

        // Verify token
        try {
            // Decode the token
            const decoded = await promisify(jwt.verify)(
                token,
                process.env.JWT_SECRET
            );
        } catch (err) {
            // Handle the custom exceptions
            if (err.name === 'JsonWebTokenError') {
                res.status(401).json({
                    status: 'failed',
                    message: 'Invalid Token',
                });
```

```
            }

            if (err.name === 'TokenExpiredError') {
                res.status(401).json({
                    status: 'failed',
                    message: 'Token has expired',
                });
            }

            // terminate the function
            return;
        }

        // Call the next middleware if authentication succeeded
        next();
    } catch (err) {
        // Sending error message if error occurs
        res.status(401).json({
            status: 'failed',
            message: err.message,
        });
    }
};

// Extract the token from the request
const getTokenFromRequest = (req) => {
    // Separate the JWT from the header
    if (
        req.headers.authorization &&
        req.headers.authorization.startsWith('Bearer')
    ) {
        token = req.headers.authorization.split(' ')[1];
    }
    return token;
};
```

emailController.js

```
const Email = require('../utils/email');
const Sensor = require('../models/Sensor');
const pug = require('pug');
const Moment = require('moment');

exports.emailHandler = async (req, res) => {
```

```
    try {
        // Extracting the values from the request body
        const { to, sensor, reading } = req.body;

        // Check whether all the variable has passed
        if (
            !to ||
            !sensor ||
            !reading ||
            !req.headers.authorization ||
            req.headers.authorization !== process.env.EMAIL_SENDING_ACCESS_TOKEN
        ) {
            // Throw an error if any condition unsatisfied
            throw {
                name: 'InvalidRequestBodyError',
                message: 'Bad request! Check the body object and the password',
            };
        }

        // Find the sensor object from the data base and assign it to a variable
        const sensorObject = await Sensor.findById(sensor);

        // Create a local time object with momentJS
        const time = Moment(sensorObject.time);

        // Email Subject
        const subject = `Fire Alert! A Harmful air condition detected in ${sensor
Object.floor} floor, room ${sensorObject.room}`;

        // Render the email template
        const html = pug.renderFile(`${__dirname}/../Views/emails/alert.pug`, {
            floor: sensorObject.floor,
            room: sensorObject.room,
            smokeLevel: reading.smokeLevel,
            co2Level: reading.co2Level,
            date: time.format('MMM Do YYYY'),
            time: time.format('LT'),
            url: 'https://www.google.com',
        });

        // Sending the email
        await Email.sendEmail(to, subject, html);

        // Sending the response when email sending succeed
        res.status(201).json({
```

```
            status: 'success',
            data: {
                subject,
            },
        });
    } catch (err) {
        // Sending error message when error occurs
        res.status(400).json({
            status: 'failed',
            message: err.message,
        });
    }
};
```

sensorController.js

```
const Sensor = require('../models/Sensor');

// Add new sensor
exports.createSensor = async (req, res) => {
    try {
        // Create a new sensor in the data base and get it to a variable
        const newSensor = await Sensor.create(req.body);

        // Sending back the response with added sensor object
        res.status(201).json({
            status: 'success',
            data: {
                sensor: newSensor,
            },
        });
    } catch (err) {
        // Sending error message when error occurs
        res.status(400).json({
            status: 'failed',
            message: err.message,
        });
    }
};

// Returns all the sensors
exports.getAllSensors = async (req, res) => {
    try {
        // Getting the all sensor objects from the database
```

```
        const sensors = await Sensor.find(req.query);

        // Sending back the response with all the sensor objects
        res.status(200).json({
            status: 'success',
            results: sensors.length,
            data: {
                sensors,
            },
        });
    } catch (err) {
        // Sending error message when error occurs
        res.status(400).json({
            status: 'failed',
            message: err.message,
        });
    }
};

// Returns a sensor object matches with the given id
exports.getSensor = async (req, res) => {
    try {
        // Find the sensor from database and assign it to a variable
        const sensor = await Sensor.findById(req.params.id);

        // Sending back the response with the sensor object
        res.status(200).json({
            status: 'success',
            data: {
                sensor,
            },
        });
    } catch (err) {
        // Sending the error message when error occurs
        res.status(400).json({
            status: 'failed',
            message: err.message,
        });
    }
};

// Updating the sensor details
exports.updateSensor = async (req, res) => {
    try {
        // Update the database and getting updated sensor object
```

```
        const sensor = await Sensor.findByIdAndUpdate(req.params.id, req.body, {
            new: true,
            runValidators: true,
        });

        // Sending response back to the user with updated sensor object
        res.status(200).json({
            status: 'success',
            data: {
                sensor,
            },
        });
    } catch (err) {
        // Sending error message when error occurs
        res.status(400).json({
            status: 'failed',
            message: err.message,
        });
    }
};

// Deleting the sensor from the database
exports.deleteSensor = async (req, res) => {
    try {
        // Remove the sensor from the database
        await Sensor.findByIdAndDelete(req.params.id);

        // Sending response the success message
        res.status(204).json({
            status: 'success',
            data: null,
        });
    } catch (err) {
        // Sending a error message when error occurs
        res.status(400).json({
            status: 'failed',
            message: err.message,
        });
    }
};
```

sensorReadingController.js

```
const SensorReading = require('../models/SensorReading');
const Sensor = require('../models/Sensor');
const Moment = require('moment');

exports.addSensorReading = async (req, res) => {
    try {
        // Insert the new sensor document in to SensorReadings collection
        const newSensorReading = await SensorReading.create({
            sensor: req.params.id,
            reading: {
                smokeLevel: req.body.reading.smokeLevel,
                co2Level: req.body.reading.co2Level,
            },
        });

        // Update the sensor's last reading in the Sensors collection
        let lastReading = newSensorReading.reading;
        lastReading.time = Moment(lastReading.time).format(
            'MMMM Do YYYY, h:mm:ss a'
        );

        // Update the sensor details
        await Sensor.findByIdAndUpdate(
            req.params.id,
            { lastReading },
            {
                new: true,
                runValidators: true,
            }
        );

        // Send response to the client
        res.status(201).json({
            status: 'success',
            data: {
                sensorReading: newSensorReading,
            },
        });
    } catch (err) {
        res.status(400).json({
            status: 'failed',
            message: err.message,
        });
```

```
    }
};

// Returns all the sensor readings
exports.getSensorReadings = async (req, res) => {
    try {
        // find the readings that matches with the given id
        const sensorReadings = await SensorReading.find({
            sensor: req.params.id,
        });

        // Sending the response with sensor readings
        res.status(200).json({
            status: 'success',
            results: sensorReadings.length,
            data: {
                sensorReadings,
            },
        });
    } catch (err) {
        // Sending error message when error occurs
        res.status(400).json({
            status: 'failed',
            message: err.message,
        });
    }
};
```

smsController.js

```
const messagebird = require('messagebird')(
    `${process.env.MESSAGEBIRD_ACCESSKEY}`
);
const Sensor = require('../models/Sensor');
const Moment = require('moment');

exports.smsHandler = async (req, res) => {
    try {
        // Extracting the values from the request body
        const { to, sensor, reading } = req.body;

        // Check whether the request body has valid sending token
        if (
            !to ||
```

```
            !sensor ||
            req.headers.authorization !== process.env.EMAIL_SENDING_ACCESS_TOKEN
        ) {
            throw {
                name: 'InvalidRequestBodyError',
                message: 'Bad request! Check the body object and the password',
            };
        }

        // Find the sensor object from the data base and assign it to a variable
        const sensorObject = await Sensor.findById(sensor);

        // SMS Text
        const message = `Fire Alert! A Harmful air condition has been detected in
 ${sensorObject.floor} floor, room ${sensorObject.room}. There may be a fire emer
gency inside.`;

        // Build the SMS Object
        let params = {
            originator: 'Fire Alert',
            recipients: [`${to}`],
            body: message,
        };

        // Sending the SMS
        messagebird.messages.create(params, function (err, response) {
            if (err) {
                // Sending the error message if sms sending failed
                res.status(400).json({
                    status: 'failed',
                    message: err,
                });
                return;
            }
            // Sending success message
            res.status(201).json({
                status: 'success',
                data: response,
            });
        });
    } catch (err) {
        // Sending error message when error occurs
        res.status(400).json({
            status: 'failed',
            message: err.message,
```

```
        });
    }
};
```

Admin.js

```
const mongoose = require('mongoose');
const validator = require('validator');
const bcrypt = require('bcryptjs');

// Build the admin schema
const adminSchema = new mongoose.Schema({
    name: {
        type: String,
        required: [true, 'Please tell us your name!'],
    },
    email: {
        type: String,
        required: [true, 'Please provide your email'],
        unique: true,
        lowercase: true,
        validate: [validator.isEmail, 'Please provide a valid email'],
    },
    password: {
        type: String,
        required: [true, 'Please provide a password'],
        minlength: 8,
        select: false,
    },
    passwordConfirm: {
        type: String,
        required: [true, 'Please confirm your password'],
        validate: {
            validator: function (el) {
                return el === this.password;
            },
            message: "Confirmed Password doesn't matched",
        },
    },
});

// Check the password for login
adminSchema.methods.checkPassword = async function (
    candidatePassword,
```

```
    adminPassword
) {
    return await bcrypt.compare(candidatePassword, adminPassword);
};

// Password encryption
adminSchema.pre('save', async function (next) {
    // Add condition to check password change
    if (!this.isModified('password')) return next();

    this.password = await bcrypt.hash(this.password, 12);
    this.passwordConfirm = undefined;
    next();
});

// Create and export admin schema
const Admin = mongoose.model('admins', adminSchema);
module.exports = Admin;
```

Sensor.js

```
const mongoose = require('mongoose');

// Build the sensor schema
const sensorSchema = new mongoose.Schema({
    _id: {
        type: String,
        required: true,
    },
    floor: {
        type: Number,
        required: true,
    },
    room: {
        type: String,
        required: true,
    },
    activated: {
        type: Boolean,
        default: true,
    },
    lastReading: {
        smokeLevel: {
            type: Number,
```

```
            default: 0,
        },
        co2Level: {
            type: Number,
            default: 0,
        },
        time: {
            type: String,
            default: Date.now,
        },
    },
});

// Create and export the sensor schema
const Sensor = mongoose.model('sensors', sensorSchema);
module.exports = Sensor;
```

SensorReading.js

```
const mongoose = require('mongoose');

// Build sensors reading schema
const sensorReadingSchema = new mongoose.Schema({
    sensor: {
        type: String,
        ref: 'Sensor',
    },
    reading: {
        smokeLevel: {
            type: Number,
            required: true,
        },
        co2Level: {
            type: Number,
            required: true,
        },
        time: {
            type: Date,
            default: Date.now,
        },
    },
});

// Create and export sensor readings schema
```

```
const SensorReading = mongoose.model('sensorReadings', sensorReadingSchema);
module.exports = SensorReading;
```

email.js

```
// Import the module that gives from Send Grid email service
const sgMail = require('@sendgrid/mail');
const htmlToText = require('html-to-text');

// Email sending method
exports.sendEmail = async (to, subject, html) => {
    // Setting API key
    sgMail.setApiKey(process.env.SENDGRID_API_KEY);

    // Build the email object
    const email = {
        to,
        from: process.env.EMAIL_FROM,
        subject,
        text: htmlToText.fromString(html),
        html,
    };

    // Sending email
    await sgMail.send(email);
};
```

alert.pug

```
doctype html
html(lang="en")
  head
    meta(charset="UTF-8")
    meta(name="viewport" content="width=device-width, initial-scale=1.0")
    title Email
  body(style="font-family: Arial, Helvetica, sans-serif; margin-bottom: 100px;")
    div(style="width: 100%;")
      div(style="\
      max-width: 1000px;\
      margin-left: auto;\
      margin-right: auto;\
      ")
        div(style="\
```

```
        width: 100%;\
        height: 200px;\
        background-color: rgb(233, 233, 233);\
        text-align: center;\
        ")
          img(src="https://i.ibb.co/xD3y5VZ/fire-alert-
sign.png" alt="warning" style="width: 100px; margin: 10px auto 0px auto;")
          p(style="\
          font-size: 40px;\
          color: rgb(87, 87, 87);\
          margin: 0;\
          ")
            | FIRE ALERT
          p(style="font-size: 10px; color: gray; margin: 0;")
            | A harmful air condition has been detected
        br
        h2 Dear Admin,
        p
          | A harmful air condition has been detected in #{floor} floor,
          | Room #{room}. It's urgent to pay your attention. There may be a fire
emergency inside.
        br
        hr
        div(style="width: 100%; text-align: center; margin-top: 50px;")
          h1(style="color: rgb(0, 102, 255);")
            | #{floor} Floor | Room #{room}
          h3(style="\
          color: rgba(112, 112, 112, 0.692);\
          margin-top: 0;\
          ")
            | On #{date} At #{time}
          table(style="width: 60%; margin: auto;")
            tr
              td
                h3 Smoke Level
              td
                h3 CO2 Level
            tr
              td
                span(style="\
                font-size: 50px;\
                color: rgb(187, 19, 19);\
                ") #{smokeLevel}
              td
                span(style="\
```

```
                font-size: 50px;\
                color: rgb(187, 19, 19);\
                ") #{co2Level}
        div(style="\
        width: 100%;\
        text-align: center;\
        margin-top: 20px;\
        margin-bottom: 50px;\
        ")
          a(href=`${url}` target="_blank" style="\
          text-decoration: none;\
          width: 50%;\
          height: 100px;\
          background-color: rgb(22, 104, 199);\
          padding: 10px 50px;\
          color: white;\
          border-radius: 20px;\
          margin: auto;\
          ") More Details
        hr
        div(style="width: 100%; text-align: center; margin-top: 30px;")
          img(src="https://i.ibb.co/N7L2Gn3/fire-alert-logo-name.png" alt="aq-
visualizer-logo" style="width: 200px;")
```

app.js

```
const express = require('express');
const morgan = require('morgan');
const bodyParser = require('body-parser');
const path = require('path');
const cors = require('cors');

// Importing routes from the routes folder
const sensorRoutes = require('./routes/sensorRoutes');
const sensorReadingRoutes = require('./routes/sensorReadingRoutes');
const adminRoutes = require('./routes/adminRoutes');
const emailRoutes = require('./routes/emailRoutes');
const smsRoutes = require('./routes/smsRoutes');

// Creating a Express application
const app = express();

// CORS support
app.use(cors());
```

```
// Setting body parser to get access of request.body
app.use(bodyParser.json());

// Logging all the requests to the console in development environment
if (process.env.NODE_ENV === 'development') {
    app.use(morgan('dev'));
}

// Setting the routes to the app as middlewares
app.use('/api/v1/sensors', sensorRoutes);
app.use('/api/v1/sensorReadings', sensorReadingRoutes);
app.use('/api/v1/admin', adminRoutes);
app.use('/api/v1/email', emailRoutes);
app.use('/api/v1/sms', smsRoutes);

// Setting static webpage
app.use(express.static('./client'));
app.get('*', (req, res) => {
    res.sendFile(path.resolve(__dirname, 'client', 'index.html'));
});

module.exports = app;
```

server.js

```
const dotenv = require('dotenv').config({ path: './config.env' });
const mongoose = require('mongoose');
const port = process.env.PORT || 8000;
const app = require('./app');

// Build the connection string
const DB = process.env.DATABASE.replace(
    '<PASSWORD>',
    process.env.DATABASE_PASSWORD
);

// Connecting to the database
mongoose
    .connect(DB, {
        useNewUrlParser: true,
        useCreateIndex: true,
        useFindAndModify: false,
        useUnifiedTopology: true,
```

```
    })
    .then(() => console.log('Connected to the mongoDB successfully'))
    .catch((err) => {
        console.log(
            'Something went wrong. Check your connection and credentials'
        );
    });

// Starting the express server
app.listen(port, () => {
    console.log(`Server starts running on port ${port}`);
});
```

adminRoutes.js

```
const express = require('express');
const adminController = require('./../controllers/adminController');
const authController = require('./../controllers/authController');

const router = express.Router();

router.post('/signup', authController.signup); // Signup route
router.post('/login', authController.login); // Login route

router.route('/').get(adminController.getAdmin); // admin root route
router.route('/:id').patch(authController.protect, adminController.updateAdmin);
// admin root route with query parameter

module.exports = router;
```

emailRoutes.js

```
const express = require('express');
const emailController = require('./../controllers/emailController');

const router = express.Router();

router.route('/').post(emailController.emailHandler); // email root route

module.exports = router;
```

sensorReadingRoutes.js

```
const express = require('express');
const sensorReadingController = require('./../controllers/sensorReadingController
');

const router = express.Router();

router.route('/').get(sensorReadingController.getSensorReadings); // sensorReadin
gs root route
router.route('/:id').post(sensorReadingController.addSensorReading); // sensorRea
dings root route with query parameter

module.exports = router;
```

sensorRoutes.js

```
const express = require('express');
const sensorController = require('./../controllers/sensorController');
const authController = require('./../controllers/authController');

const router = express.Router();

// sensor root route
router
    .route('/')
    .get(sensorController.getAllSensors)
    .post(authController.protect, sensorController.createSensor);

// sensor root route with query parameter
router
    .route('/:id')
    .get(sensorController.getSensor)
    .patch(sensorController.updateSensor)
    .delete(authController.protect, sensorController.deleteSensor);

module.exports = router;
```

smsRoute.js

```
const express = require('express');
const smsController = require('./../controllers/smsController');

const router = express.Router();
```

```
router.route('/').post(smsController.smsHandler); // sms root route

module.exports = router;
```

index.html

```
<!DOCTYPE html>
<html lang="en">
    <head>
        <meta charset="UTF-8" />
        <meta name="viewport" content="width=device-width, initial-scale=1.0" />
        <title>Fire Alert</title>
        <link
            rel="icon"
            href="https://i.ibb.co/6Nb0BJD/fire-alert-logo.png"
            type="image/png"
        />

        <style>
            .center-screen {
                position: absolute;
                top: 50%;
                left: 50%;
                margin-right: -50%;
                transform: translate(-50%, -50%);
                height: 100px;
                width: 400px;
            }

            img {
                width: 100%;
            }
        </style>
    </head>
    <body>
        <div class="center-screen">
            <img
                src="https://i.ibb.co/N7L2Gn3/fire-alert-logo-name.png"
                alt="aq-visualizer-logo"
            />
        </div>
    </body>
</html>
```

RMI Server and RMI Desktop Client Application

RMIServer.java

```
public class RMIServer extends UnicastRemoteObject implements RMIService {

    public static void main(String[] args) throws RemoteException, AlreadyBoundException, IOEx
ception {

        Registry registry = LocateRegistry.createRegistry(5099);
        registry.bind("AirSensorService", new RMIServer());

        Timer t = new Timer(0, null);

        t.addActionListener(new ActionListener() {

            @Override
            public void actionPerformed(ActionEvent e) {
                try {
                    checkStateRepeatedly();
                } catch (Exception e1) {
                    e1.printStackTrace();
                }
            }
        });

        t.setRepeats(true);
        t.setDelay(5000); // repeat every 15 sec
        t.start();

    }

    protected RMIServer() throws RemoteException {
        super();
    }

    /*
     * used to retrieve sensor readings and other details of all sensors
     */
    @Override
    public String getAllSensorDetails() throws RemoteException {
        HttpClient client = HttpClient.newHttpClient();
        // prepare a HTTP request to send to API
        HttpRequest request = HttpRequest
                .newBuilder(URI.create("https://fire-alert-
solution.herokuapp.com/api/v1/sensors/")).build();
        return client.sendAsync(request, HttpResponse.BodyHandlers.ofString()).thenApply(HttpR
esponse::body)
                .thenApply((responseBody) -> parse(responseBody)).join();
    }

    public static String parse(String responseBody) {
        return responseBody;
    }
```

```
    /*
     * used to retrieve authenticate Admin login credentials
     */
    @Override
    public String loginValidator(String email, String password) throws RemoteException {

        JSONObject json = new JSONObject();
        json.put("email", email);
        json.put("password", password);

        String res = null;

        CloseableHttpClient httpClient = HttpClientBuilder.create().build();

        try {
            // prepare a HTTP request to send to API
            HttpPost request = new HttpPost("https://fire-alert-
solution.herokuapp.com/api/v1/admin/login");
            StringEntity params = new StringEntity(json.toString());
            // add headers to the request
            request.addHeader("content-type", "application/json");
            request.setEntity(params);
            org.apache.http.HttpResponse response = httpClient.execute(request);

            // check the response
            if (response.getStatusLine().toString().equalsIgnoreCase("HTTP/1.1 200 OK")) {
                res = "success";
            } else {
                res = "failed";
            }

        } catch (Exception ex) {
            ex.printStackTrace();
        } finally {
            try {
                httpClient.close();
            } catch (IOException ex) {
                ex.printStackTrace();
            }
        }
        return res;
    }

    /*
     * used to check co2 and smoke levels repeatedly
     *
     */
    public static void checkStateRepeatedly() {
        HttpClient client = HttpClient.newHttpClient();
        // prepare a HTTP request to send to API
        HttpRequest request = HttpRequest
                .newBuilder(URI.create("https://fire-alert-
solution.herokuapp.com/api/v1/sensors/")).build();
```

```
        client.sendAsync(request, HttpResponse.BodyHandlers.ofString()).thenApply(HttpResponse
::body)
                .thenApply((responseBody) -> checkCo2andSmokeLevel(responseBody)).join();
    }

    private static String checkCo2andSmokeLevel(String responseBody) {

        JSONObject res = new JSONObject(responseBody);
        JSONObject data = res.getJSONObject("data");
        JSONArray sensors = data.getJSONArray("sensors");

        for (int i = 0; i < sensors.length(); i++) {

            JSONObject obj = sensors.getJSONObject(i);

            JSONObject lastReading = obj.getJSONObject("lastReading");

            int co2Level = lastReading.getInt("co2Level");
            int smokeLevel = lastReading.getInt("smokeLevel");
            String _id = obj.getString("_id");

            if (co2Level > 5 || smokeLevel > 5) {

                // create JSON object to send with Email API call
                JSONObject jsonReadingEmail = new JSONObject();
                jsonReadingEmail.put("smokeLevel", smokeLevel);
                jsonReadingEmail.put("co2Level", co2Level);

                JSONObject jsonEmail = new JSONObject();
                jsonEmail.put("to", "kavindu.ktm@gmail.com");
                jsonEmail.put("sensor", _id);
                jsonEmail.put("reading", jsonReadingEmail);

                // create JSON object to send with SMS API call
                JSONObject jsonReadingSms = new JSONObject();
                jsonReadingSms.put("smokeLevel", smokeLevel);
                jsonReadingSms.put("co2Level", co2Level);

                JSONObject jsonSms = new JSONObject();
                jsonSms.put("to", "+94711334645");
                jsonSms.put("sensor", _id);
                jsonSms.put("reading", jsonReadingSms);

                CloseableHttpClient httpClient = HttpClientBuilder.create().build();

                try {
                    // prepare a HTTP request to send to API to send email
                    HttpPost requestEmail = new HttpPost("https://fire-alert-
solution.herokuapp.com/api/v1/email");
                    StringEntity paramsEmail = new StringEntity(jsonEmail.toString());
                    // add headers to the request
                    requestEmail.addHeader("content-type", "application/json");
                    requestEmail.addHeader("Authorization", "agfYjhdioJK5ghiH46dHr8gfg857yfrJY
uit57vf");
```

```
                    requestEmail.setEntity(paramsEmail);
                    org.apache.http.HttpResponse responseEmail = httpClient.execute(requestEma
il);

                    // prepare a HTTP request to send to API to send SMS
                    HttpPost requestSms = new HttpPost("https://fire-alert-
solution.herokuapp.com/api/v1/sms");
                    StringEntity paramsSms = new StringEntity(jsonSms.toString());
                    // add headers to the request
                    requestSms.addHeader("content-type", "application/json");
                    requestSms.addHeader("Authorization", "agfYjhdioJK5ghiH46dHr8gfg857yfrJYui
t57vf");
                    requestSms.setEntity(paramsSms);
                    org.apache.http.HttpResponse responseSms = httpClient.execute(requestSms);

                    // check the responses
                    System.out.println(responseEmail.getStatusLine().toString().equalsIgnoreCa
se("HTTP/1.1 201 Created")
                            ? "Email has Sent"
                            : "Email has not Sent");
                    System.out.println(responseSms.getStatusLine().toString().equalsIgnoreCase
("HTTP/1.1 201 Created")
                            ? "Sms has Sent"
                            : "Sms has not Sent");

                } catch (Exception ex) {
                    ex.printStackTrace();
                } finally {
                    try {
                        httpClient.close();
                    } catch (IOException e) {
                        e.printStackTrace();
                    }
                }
            }
        }

        return null;
    }

    /*
     * used to register a new sensor
     */
    @Override
    public boolean addSensor(String id, int floor, String room) throws RemoteException {

        boolean res = false;

        JSONObject json = new JSONObject();
        json.put("_id", id);
        json.put("floor", floor);
        json.put("room", room);

        CloseableHttpClient httpClient = HttpClientBuilder.create().build();
```

```
        try {
            // prepare a HTTP request to send to API
            HttpPost request = new HttpPost("https://fire-alert-
solution.herokuapp.com/api/v1/sensors");
            StringEntity params = new StringEntity(json.toString());
            // add headers to the request
            request.addHeader("content-type", "application/json");
            request.addHeader("Authorization",
                    "Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6IjVlOWRiYzlmZTE3NTBiM
DAxN2Q1OGRiOSIsImlhdCI6MTU4NzM5NTc0NCwiZXhwIjoxNTg5OTg3NzQ0fQ.4MZXhOVMVkiMboNhoGyiCDeuY6yfysrg
H70PB1nAKok");
            request.setEntity(params);
            org.apache.http.HttpResponse response = httpClient.execute(request);

            System.out.println(response.getStatusLine().toString().equalsIgnoreCase("HTTP/1.1
201 Created"));

            // check the response
            res = response.getStatusLine().toString().equalsIgnoreCase("HTTP/1.1 201 Created")
;

        } catch (Exception ex) {
            ex.printStackTrace();
        } finally {
            try {
                httpClient.close();
            } catch (IOException e) {
                e.printStackTrace();
            }
        }

        return res;
    }

    /*
     * used to edit existing sensor details
     */
    @Override
    public boolean editSensor(String id, int floor, String room) throws RemoteException {

        boolean res = false;

        JSONObject json = new JSONObject();
        json.put("_id", id);
        // json.put("activated", activated);
        json.put("floor", floor);
        json.put("room", room);

        CloseableHttpClient httpClient = HttpClientBuilder.create().build();

        try {
            // prepare a HTTP request to send to API
```

```
            HttpPatch request = new HttpPatch("https://fire-alert-
solution.herokuapp.com/api/v1/sensors/" + id);
            StringEntity params = new StringEntity(json.toString());
            // add headers to the request
            request.addHeader("content-type", "application/json");
            request.addHeader("Authorization",
                    "Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6IjVlOWRiYzlmZTE3NTBiM
DAxN2Q1OGRiOSIsImlhdCI6MTU4NzM5NTc0NCwiZXhwIjoxNTg5OTg3NzQ0fQ.4MZXhOVMVkiMboNhoGyiCDeuY6yfysrg
H70PB1nAKok");
            request.setEntity(params);
            org.apache.http.HttpResponse response = httpClient.execute(request);

            System.out.println(response.getStatusLine().toString().equalsIgnoreCase("HTTP/1.1
200 OK"));

            // check the response
            res = response.getStatusLine().toString().equalsIgnoreCase("HTTP/1.1 200 OK");

        } catch (Exception ex) {
            System.out.println(ex);
        } finally {
            try {
                httpClient.close();
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
        return res;
    }

    /*
     * used to delete existing sensor
     */
    @Override
    public boolean deleteSensor(String id) throws RemoteException {

        boolean res = false;

        CloseableHttpClient httpClient = HttpClientBuilder.create().build();

        try {
            // prepare a HTTP request to send to API
            HttpDelete request = new HttpDelete("https://fire-alert-
solution.herokuapp.com/api/v1/sensors/" + id);
            // add headers to the request
            request.addHeader("content-type", "application/json");
            request.addHeader("Authorization",
                    "Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6IjVlOWRiYzlmZTE3NTBiM
DAxN2Q1OGRiOSIsImlhdCI6MTU4NzM5NTc0NCwiZXhwIjoxNTg5OTg3NzQ0fQ.4MZXhOVMVkiMboNhoGyiCDeuY6yfysrg
H70PB1nAKok");
            org.apache.http.HttpResponse response = httpClient.execute(request);

            // check the response
```

```
            res = response.getStatusLine().toString().equalsIgnoreCase("HTTP/1.1 204 No Conten
t");

        } catch (Exception ex) {
            System.out.println(ex);
        } finally {
            try {
                httpClient.close();
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
        return res;
    }
}
```

RMIService.java

```
public interface RMIService extends Remote {

    /*
     * methods that are exposed to desktop client
     * */

    public String getAllSensorDetails() throws RemoteException;

    public String loginValidator(String email, String password) throws RemoteException;

    public boolean addSensor(String id, int floor, String room) throws RemoteException;

    public boolean editSensor(String id, int floor, String room) throws RemoteException;

    public boolean deleteSensor(String id) throws RemoteException;

}
```

DashBoardFrm.java

```
public class DashBoardFrm extends JFrame {

    private static JPanel contentPane;
    private static String responseBody;
    private static DashBoardFrm frame;
    private static boolean isAdmin = false;

    /**
     * Launch the application.
     */
    public static void main(String[] args) {
        EventQueue.invokeLater(new Runnable() {
```

```
            public void run() {
                try {
                    frame = new DashBoardFrm(isAdmin);
                    frame.setVisible(true);
                } catch (Exception e) {
                    e.printStackTrace();
                }

                RMIService service;

                try {

                    // find the service
                    service = (RMIService) Naming.lookup("rmi://localhost:5099/AirSensorServic
e");


                    Timer t = new Timer(0, null);

                    t.addActionListener(new ActionListener() {

                        @Override
                        public void actionPerformed(ActionEvent e) {
                            try {
                                // call remote method and get the response to a variable
                                responseBody = service.getAllSensorDetails();
                            } catch (RemoteException e1) {
                                e1.printStackTrace();
                            }
                            populateSensorComponents(responseBody);
                        }
                    });

                    t.setRepeats(true);
                    t.setDelay(30000); // repeat every 30 sec
                    t.start();

                } catch (MalformedURLException | RemoteException | NotBoundException e) {
                    e.printStackTrace();
                }
            }
        });
    }

    /**
     * Create the frame.
     */
    public DashBoardFrm(boolean isAdminn) {

        JMenuItem mntmNewMenuItem = new JMenuItem("Add New Sensor");
        // add event listner to menu item
        mntmNewMenuItem.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                frame.dispose();
                AddSensorForm addSensorForm = new AddSensorForm();
```

```
                addSensorForm.setVisible(true);
            }
        });

        JMenuItem mntmNewMenuItem_1 = new JMenuItem("Exit");
        // add event listner to menu item
        mntmNewMenuItem_1.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                System.exit(EXIT_ON_CLOSE);
            }
        });
        mnNewMenu.add(mntmNewMenuItem_1);
    }

    /**
     * used to populate sensor details
     */
    public static void populateSensorComponents(String responseBody) {

        contentPane.removeAll();

        JSONObject res = new JSONObject(responseBody);
        JSONObject data = res.getJSONObject("data");
        JSONArray sensors = data.getJSONArray("sensors");

        // loop through the response body and add each sensor's details to the user interface
        for (int i = 0; i < sensors.length(); i++) {

            JSONObject obj = sensors.getJSONObject(i);

            JSONObject lastReading = obj.getJSONObject("lastReading");

            int co2Level = lastReading.getInt("co2Level");
            int smokeLevel = lastReading.getInt("smokeLevel");
            String time = lastReading.getString("time");

            boolean activated = obj.getBoolean("activated");
            String _id = obj.getString("_id");
            int floor = obj.getInt("floor");
            String room = obj.getString("room");

            SensorDetailComponent sensorDetailComponent = new SensorDetailComponent(_id, floor
, room, activated,
                    co2Level, smokeLevel, isAdmin, frame);
            sensorDetailComponent.setVisible(true);
            contentPane.add(sensorDetailComponent);
        }
        contentPane.validate();
        contentPane.repaint();
    }
}
```

WelcomeForm.java

```
public class WelcomeForm {
    private JFrame frame;
    /**
     * Launch the application.
     */
    public static void main(String[] args) {
        EventQueue.invokeLater(new Runnable() {
            public void run() {
                try {
                    WelcomeForm window = new WelcomeForm();
                    window.frame.setVisible(true);
                } catch (Exception e) {
                    e.printStackTrace();
                }
            }
        });
    }

    /**
     * Create the application.
     */
    public WelcomeForm() {
        initialize();
    }
    /**
     * Initialize the contents of the frame.
     */
    private void initialize() {


        JButton btnNewButton = new JButton("Admin Login");
        btnNewButton.setIcon(new ImageIcon(WelcomeForm.class.getResource("/img/admin.png")));
        btnNewButton.setFont(new Font("Tahoma", Font.PLAIN, 25));
        btnNewButton.addActionListener(new ActionListener() {
            // add event listner to the button
            public void actionPerformed(ActionEvent e) {
                frame.dispose();
                // open Admin login form
                AdminLoginForm adminLoginForm = new AdminLoginForm();
                adminLoginForm.setVisible(true);
            }
        });


        JButton btnNewButton_1 = new JButton("Guest Login");
        btnNewButton_1.setIcon(new ImageIcon(WelcomeForm.class.getResource("/img/user.png")));
        btnNewButton_1.setFont(new Font("Tahoma", Font.PLAIN, 25));
        btnNewButton_1.addActionListener(new ActionListener() {
            // add event listner to the button
```

```
            public void actionPerformed(ActionEvent e) {
                frame.dispose();
                // open Dashboard to guest user
                DashBoardFrm dashboardForm = new DashBoardFrm(false);
                dashboardForm.main(null);
            }
        });
    }
}
```

AdminLoginForm.java

```
public class AdminLoginForm extends JFrame {

    private JPanel contentPane;
    private JTextField txtemail;
    private JTextField txtpassword;
    private static AdminLoginForm frame;

    public static void main(String[] args) {
        EventQueue.invokeLater(new Runnable() {
            public void run() {
                try {
                    frame = new AdminLoginForm();
                    frame.setVisible(true);
                } catch (Exception e) {
                    e.printStackTrace();
                }
            }
        });
    }
    public AdminLoginForm() {
        JButton btnsignin = new JButton("Sign In");
        btnsignin.setIcon(new ImageIcon(AdminLoginForm.class.getResource("/img/signin_.png")))
;
        btnsignin.setFont(new Font("Tahoma", Font.PLAIN, 20));
        btnsignin.addActionListener(new ActionListener() {
            // add event listner to the button
            public void actionPerformed(ActionEvent e) {

                String email = txtemail.getText();
                String password = txtpassword.getText();

                RMIService service;
                String result = null;
                try {
                    // find the remote service
                    service = (RMIService) Naming.lookup("rmi://localhost:5099/AirSensorServic
e");
                    // invoke the remote method
                    result = service.loginValidator(email, password);
```

```
                } catch (MalformedURLException | RemoteException | NotBoundException ex) {
                    ex.printStackTrace();
                }

                System.out.println(result); // write the result on the console

                if (result.equalsIgnoreCase("success")) {
                    // if valid Admin credential, open the dashboard
                    frame.dispose();
                    DashBoardFrm dashboardForm = new DashBoardFrm(true);
                    dashboardForm.main(null);
                } else {
                    // if invalid Admin credential, display error msg
                    lblerror.setVisible(true);
                }
            }
        });
    }
}
```

AddSensorForm.java

```
public class AddSensorForm extends JDialog {

    private final JPanel contentPanel = new JPanel();
    private JTextField txtsensorid;
    private JTextField txtfloorno;
    private JTextField txtroomno;
    private boolean res;

    /**
     * Launch the application.
     */
    public static void main(String[] args) {
        try {
            AddSensorForm dialog = new AddSensorForm();
            dialog.setDefaultCloseOperation(JDialog.DISPOSE_ON_CLOSE);
            dialog.setVisible(true);
        } catch (Exception e) {
            e.printStackTrace();
        }
    }

    /**
     * Create the dialog.
     */
    public AddSensorForm() {
        JLabel lblerromsg = new JLabel("Same ID " + txtsensorid.getText() + " is already exist
!");
        lblerromsg.setFont(new Font("Tahoma", Font.PLAIN, 13));
        lblerromsg.setForeground(Color.RED);
        lblerromsg.setBounds(153, 161, 204, 19);
        lblerromsg.setVisible(false);
```

```
        contentPanel.add(lblerromsg);
        {
            JPanel buttonPane = new JPanel();
            buttonPane.setLayout(new FlowLayout(FlowLayout.RIGHT));
            getContentPane().add(buttonPane, BorderLayout.SOUTH);
            {
                JButton okButton = new JButton("OK");
                okButton.addActionListener(new ActionListener() {
                    public void actionPerformed(ActionEvent e) {

                        String id = txtsensorid.getText();
                        int floor = Integer.parseInt(txtfloorno.getText());
                        String room = txtroomno.getText();

                        RMIService service;

                        try {
                            //find the service
                            service = (RMIService) Naming.lookup("rmi://localhost:5099/AirSens
orService");


                            try {
                                //invoke remote method and assign to a variable
                                res = service.addSensor(id, floor, room);

                            } catch (RemoteException e1) {
                                e1.printStackTrace();
                            }
                        } catch (MalformedURLException | RemoteException | NotBoundException e
x) {
                            ex.printStackTrace();
                        }

                        if (res) {

                            DashBoardFrm dashboardForm = new DashBoardFrm(true);
                            dashboardForm.main(null);
                        } else {
                            lblerromsg.setVisible(true);
                        }

                    }
                });
                okButton.setActionCommand("OK");
                buttonPane.add(okButton);
                getRootPane().setDefaultButton(okButton);
            }
            {
                JButton cancelButton = new JButton("Cancel");
                cancelButton.addActionListener(new ActionListener() {
                    public void actionPerformed(ActionEvent e) {
                        System.exit(EXIT_ON_CLOSE);
                    }
                });
```

```
                cancelButton.setActionCommand("Cancel");
                buttonPane.add(cancelButton);
            }
        }
    }
}
```

EditSensorForm.java

```
public class EditSensorForm extends JDialog {
    private final JPanel contentPanel = new JPanel();
    private boolean res;
    private JTextField txtFloorNo;
    private JTextField txtRoomNo;

    /**
     * Launch the application.
     */
    public static void main(String[] args) {
        try {
            EditSensorForm dialog = new EditSensorForm(null, null, null, null);
            dialog.setDefaultCloseOperation(JDialog.DISPOSE_ON_CLOSE);
            dialog.setVisible(true);
        } catch (Exception e) {
            e.printStackTrace();
        }
    }

    /**
     * Create the dialog.
     */
    public EditSensorForm(String lblfloornumber, String lblroomnumber, String lblsensorid, JFr
ame frame) {
            JPanel buttonPane = new JPanel();
            buttonPane.setLayout(new FlowLayout(FlowLayout.RIGHT));
            getContentPane().add(buttonPane, BorderLayout.SOUTH);
            {
                JButton okButton = new JButton("OK");
                txtFloorNo.setText(lblfloornumber);
                txtRoomNo.setText(lblroomnumber);
                okButton.addActionListener(new ActionListener() {
                    public void actionPerformed(ActionEvent e) {

                        RMIService service;

                        try {
                            //find the service
                          service = (RMIService) Naming.lookup("rmi://localhost:5099/AirSensor
Service");

                          try {
```

```
                            //invoke remote method and assign to a variable
                        res = service.editSensor(lblsensorid, Integer.parseInt(txtFloorNo.g
etText()),
                                    txtRoomNo.getText());

                         } catch (RemoteException e1) {
                            e1.printStackTrace();
                         }
                    } catch (MalformedURLException | RemoteException | NotBoundException e
x) {
                        ex.printStackTrace();
                    }

                    if (res) {
                        frame.dispose();
                        DashBoardFrm dashboardForm = new DashBoardFrm(true);
                        dashboardForm.main(null);

                    } else {
                        System.out.println("Error");
                    }
                }
            });
            okButton.setActionCommand("OK");
            buttonPane.add(okButton);
            getRootPane().setDefaultButton(okButton);
        }
        {
            JButton cancelButton = new JButton("Cancel");
            cancelButton.setActionCommand("Cancel");
            buttonPane.add(cancelButton);
        }
    }
}
```

SensorDetailComponent.java

```
public class SensorDetailComponent extends JPanel {

    private String sensorId;
    private int floorNumber;
    private String roomNumber;
    private boolean status;
    private int co2Level;
    private int smokeLevel;
    private JLabel lblsensorid;
    private boolean isAdminn;
    private JFrame frame;

    public SensorDetailComponent(String sensorId, int floorNumber, String roomNumber, boolean
status, int co2level,
```

```
            int smokelevel, boolean isAdminn, JFrame frame) {
        this.sensorId = sensorId;
        this.floorNumber = floorNumber;
        this.roomNumber = roomNumber;
        this.status = status;
        this.co2Level = co2level;
        this.smokeLevel = smokelevel;
        this.isAdminn = isAdminn;
        this.frame = frame;

        JPanel panel = new JPanel();
        // set Red color if the smoke or CO2 level goes above level 5
        panel.setBackground(this.co2Level > 5 || this.smokeLevel > 5 ? new Color(210, 0, 0) :
new Color(0, 204, 0));
        panel.setBounds(0, 0, 88, 100);
        add(panel);
        panel.setLayout(null);

        JLabel lblsensorstatus = new JLabel(this.status ? "Activated" : "Deactivated");
        lblsensorstatus.setFont(new Font("Tahoma", Font.PLAIN, 14));
        lblsensorstatus.setBounds(10, 10, 93, 13);
        panel_1.add(lblsensorstatus);

        JButton btnNewButton = new JButton("");
        btnNewButton.setVisible(isAdminn);
        btnNewButton.addActionListener(new ActionListener() {
            // add event listner to the button
            public void actionPerformed(ActionEvent e) {
                EditSensorForm editSensorForm = new EditSensorForm(lblfloornumber.getText(), l
blroomnumber.getText(),
                        lblsensorid.getText(), frame);
                editSensorForm.setVisible(true);
            }
        });
        JButton btnNewButton_1 = new JButton("");
        btnNewButton_1.setVisible(isAdminn);
        btnNewButton_1.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {

                RMIService service;
                boolean res = false;
                try {
                    // find the remote service
                    service = (RMIService) Naming.lookup("rmi://localhost:5099/AirSensorServic
e");


                    try {
                        // call the reomte method and get response to a variable
                        res = service.deleteSensor(lblsensorid.getText());

                    } catch (RemoteException e1) {
                        e1.printStackTrace();
                    }
                } catch (MalformedURLException | RemoteException | NotBoundException ex) {
```

```
                    ex.printStackTrace();
                }

                if (res) {
                    frame.dispose();
                    DashBoardFrm dashboardForm = new DashBoardFrm(true);
                    dashboardForm.main(null);
                } else {
                    System.out.println("Error");
                }
            }
        });
    }
}
```

Web Client Application

RadiaBar.js

```
import React, { Component } from 'react';
import ReactApexChart from 'react-apexcharts';

class RadialBar extends Component {
    constructor(props) {
        super(props);

        this.state = {
            series: [0],
            options: {
                chart: {
                    height: 350,
                    type: 'radialBar',
                    toolbar: {
                        show: false,
                    },
                },
                plotOptions: {
                    radialBar: {
                        startAngle: -180,
                        endAngle: 180,
                        hollow: {
                            margin: 0,
                            size: '70%',
                            background: '#fff',
                            image: undefined,
                            imageOffsetX: 0,
                            imageOffsetY: 0,
                            position: 'front',
                            dropShadow: {
                                enabled: true,
                                top: 3,
                                left: 0,
                                blur: 4,
                                opacity: 0.24,
                            },
                        },
                        track: {
                            background: '#fff',
                            strokeWidth: '67%',
                            margin: 0, // margin is in pixels
                            dropShadow: {
```

```
                        enabled: true,
                        top: -3,
                        left: 0,
                        blur: 4,
                        opacity: 0.35,
                    },
                },

                dataLabels: {
                    show: true,
                    name: {
                        offsetY: -10,
                        show: false,
                        color: '#888',
                        fontSize: '17px',
                    },
                    value: {
                        formatter: function (val) {
                            return parseInt(val) / 10;
                        },
                        color: '#111',
                        fontSize: '36px',
                        show: true,
                    },
                },
            },
        },
        fill: {
            type: 'gradient',
            gradient: {
                shade: 'dark',
                type: 'horizontal',
                shadeIntensity: 0.5,
                gradientToColors: ['#ABE5A1'],
                inverseColors: true,
                opacityFrom: 1,
                opacityTo: 1,
                stops: [0, 100],
            },
        },
        stroke: {
            lineCap: 'round',
        },
        labels: [''],
    },
```

```
        };
    }

    render() {
        return (
            <div id='card'>
                <div id='chart'>
                    <ReactApexChart
                        options={this.state.options}
                        series={[this.props.series * 10]}
                        type='radialBar'
                        height={160}
                    />
                </div>
            </div>
        );
    }
}

export default RadialBar;
```

navbar.js

```
import React, { Component } from 'react';

class Navbar extends Component {
    render() {
        return (
            <nav
                className='navbar navbar-light sticky-top'
                style={{ backgroundColor: '#ebebeb' }}
            >
                <a
                    className='navbar-brand'
                    href='https://ThamalDilanka.github.io/fire-alert-web'
                >
                    <img
                        src='https://i.ibb.co/N7L2Gn3/fire-alert-logo-name.png'
                        alt='aq-visualizer-logo'
                        className='d-inline-block align-top'
                        height='50'
                    />
                </a>
            </nav>
```

```
        );
    }
}

export default Navbar;
```

sensor.js

```
import React, { Component } from 'react';
import NumberToWord from 'number-to-words';
import RadialBarChart from '../../Chart/RadialBar';
import '../Sensor/sensor.css';

class Sensor extends Component {

    getClass = () => {
        if(this.props.smokeLevel > 5 || this.props.co2Level > 5) {
            return 'sensor-container-left-warning'
        } else {
            return 'sensor-container-left-safe'
        }
    }

    render() {
        return (
            <div className='sensor-container-main shadow'>
                <div className={this.getClass()}>
                    {this.getFloor(this.props.floor)}
                    <p className='location-normal'>FLOOR</p>
                    <p className='location-highlight'>{this.props.room}</p>
                    <p className='location-normal'>ROOM</p>
                </div>

                <div className='sensor-container-right'>
                    <div className='active-status-container'>
                        {this.props.activated ? (
                            <p className='active-status-true'>Activated</p>
                        ) : (
                            <p className='active-status-false'>Disabled</p>
                        )}
                    </div>
                    <div className='chart-container'>
                        <div className='chart-container-left'>
                            <div className='chart'>
```

```
                            <RadialBarChart series={this.props.smokeLevel} />
                        </div>
                        <div className='name'>
                            <p>SMOKE</p>
                        </div>
                    </div>
                    <div className='chart-container-right'>
                        <div className='chart'>
                            <RadialBarChart series={this.props.co2Level} />
                        </div>
                        <div className='name'>
                            <p>
                                CO<sub>2</sub>
                            </p>
                        </div>
                    </div>
                </div>
            </div>
        </div>
        );
    }

    getFloor = (floor) => {
        const ordinal = NumberToWord.toOrdinal(parseInt(floor));
        const word = ordinal.slice(-2);
        return (
            <p className='location-highlight'>
                {floor}
                <sup>{word}</sup>
            </p>
        );
    };
}

export default Sensor;
```

sensors.js

```
import React, { Component } from 'react';
import Sensor from './Sensor/Sensor';

class Sensors extends Component {
    render() {
        const sensors = this.props.sensors;
```

```
        return (
            <div>
                {sensors.map((sensor) => {
                    return (
                        <Sensor
                            key={sensor._id}
                            id={sensor._id}
                            floor={sensor.floor}
                            room={sensor.room}
                            smokeLevel={sensor.lastReading.smokeLevel}
                            co2Level={sensor.lastReading.co2Level}
                            activated={sensor.activated}
                        />
                    );
                })}
            </div>
        );
    }
}

export default Sensors;
```

index.js

```
import React from 'react';
import ReactDOM from 'react-dom';
import './index.css';
import App from './App';
import * as serviceWorker from './serviceWorker';
import 'bootstrap/dist/css/bootstrap.css';

ReactDOM.render(
    <React.StrictMode>
        <App />
    </React.StrictMode>,
    document.getElementById('root')
);

serviceWorker.unregister();
```

Fire Sensor Simulator

Index.js

```
const Axios = require('axios'); // Importing library that enables to send http re
quests

const sensorId = 'ad399a80-a97b-4654-9172-
6f7f0cbc30f3'; // The unique id of the sensor 1
//const sensorId = 'fdb53432-3901-4aae-99c0-
f388ad325d56'; // The unique id of the sensor 2
//const sensorId = 'd4eb52c1-139e-4ac6-a2ee-
e5552f0f96c4'; // The unique id of the sensor 3
//const sensorId = 'e1720832-efab-4f12-a2ac-
d08827505c7d'; // The unique id of the sensor 4
//const sensorId = 'ba92cdb7-420e-4f42-a914-
fde77c612e28'; // The unique id of the sensor 5

const proxy = 'https://fire-alert-
solution.herokuapp.com/'; // The hosted rest api url

// API end points
const endpointAddReading = `${proxy}api/v1/sensorReadings/${sensorId}`;
const endpointUpdateSensor = `${proxy}api/v1/sensors/${sensorId}`;

// DOM elements initialization
const sliderSmoke = document.getElementById('smokeLevel');
const sliderCo2 = document.getElementById('co2Level');
const outputSmoke = document.getElementById('smokeLevelIndicator');
const outputCo2 = document.getElementById('co2LevelIndicator');
const toggleSwitch = document.getElementById('toggleSwitch');
const statusIndicator = document.getElementById('statusIndicator');
const displayImage = document.getElementById('display-image');
const readingCountIndicator = document.getElementById('reading-count-value');
const sensorIdIndicator = document.getElementById('sensor-id');
const rippleImage = document.getElementById('display-ripple');

// Initialize sensor reading parameters
let smokeLevel = 5,
    co2Level = 5,
    readingCount = 0;

outputSmoke.innerHTML = sliderSmoke.value;
outputCo2.innerHTML = sliderCo2.value;

sensorIdIndicator.innerHTML = sensorId;
```

```
displayImage.ondragstart = () => {
    return false;
};

sliderSmoke.oninput = function () {
    outputSmoke.innerHTML = this.value;
    smokeLevel = this.value;
};
sliderCo2.oninput = function () {
    outputCo2.innerHTML = this.value;
    co2Level = this.value;
};

// Sensor on/off functionality
toggleSwitch.addEventListener('change', () => {
    if (toggleSwitch.checked) {
        statusIndicator.innerHTML = 'Activated';
        statusIndicator.className = 'statusIndicator-active';
        rippleImage.style.visibility = 'visible';
        updateSensorStatus(true); // Update the sensor state of the database
        displayImage.src = 'sensor-on.png';
    } else {
        statusIndicator.innerHTML = 'Disabled';
        statusIndicator.className = 'statusIndicator-deactive';
        rippleImage.style.visibility = 'hidden';
        displayImage.src = 'sensor-off.png';
        updateSensorStatus(false); // Update the sensor state of the database
    }
    readingCount = 0;
    readingCountIndicator.innerHTML = readingCount;
});

// Sending sensor readings to the api
postSensorReading = () => {
    Axios.post(endpointAddReading, {
        sensor: sensorId,
        reading: {
            smokeLevel: smokeLevel,
            co2Level: co2Level,
        },
    }).then((res) => {
        readingCount++;
        readingCountIndicator.innerHTML = readingCount;
        console.log(res);
```

```
    });
};

// Updating the sensor status
updateSensorStatus = (status) => {
    Axios.patch(endpointUpdateSensor, {
        activated: status,
    }).then((res) => {
        console.log(res);
    });
};

// Set timing to send readings in every 10 seconds
postSensorReading();
readingCountIndicator.innerHTML = readingCount;
setInterval(function () {
    if (toggleSwitch.checked) {
        postSensorReading();
        readingCountIndicator.innerHTML = readingCount;
    }
}, 10000);
```

main.js

```
const { app, BrowserWindow } = require('electron');

function createWindow() {
    // Create the browser window.
    const win = new BrowserWindow({
        width: 410,
        height: 520,
        webPreferences: {
            nodeIntegration: true,
        },
    });

    // and load the index.html of the app.
    win.loadFile('./src/index.html');

    // Hide the menu bar
    win.setMenuBarVisibility(false);

    win.resizable = false;
```

```
    //win.webContents.openDevTools();
}

// This method will be called when Electron has finished
// initialization and is ready to create browser windows.
// Some APIs can only be used after this event occurs.
app.whenReady().then(createWindow);

// Quit when all windows are closed.
app.on('window-all-closed', () => {
    // On macOS it is common for applications and their menu bar
    // to stay active until the user quits explicitly with Cmd + Q
    if (process.platform !== 'darwin') {
        app.quit();
    }
});

app.on('activate', () => {
    // On macOS it's common to re-create a window in the app when the
    // dock icon is clicked and there are no other windows open.
    if (BrowserWindow.getAllWindows().length === 0) {
        createWindow();
    }
});

// In this file you can include the rest of your app's specific main process
// code. You can also put them in separate files and require them here.
```

Index.html

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="UTF-8" />
        <title>Fire Sensor Simulator</title>
        <link rel="stylesheet" href="index.css" />
        <meta
            http-equiv="Content-Security-Policy"
            content="script-src 'self' 'unsafe-inline';"
        />
    </head>
    <body>
        <div class="toggle-container">
            <img id="display-image" src="sensor-on.png" alt="sending" />
```

```
            <img id="display-ripple" src="ripple.gif" alt="sending" />

            <br />
            <label class="toggle-switch">
                <input id="toggleSwitch" type="checkbox" checked />
                <span class="toggle-slider toggle-round"></span>
            </label>
            <p id="statusIndicator" class="statusIndicator-active">Activated</p>
        </div>

        <p id="sensor-id">Sensor ID:</p>

        <br />

        <div class="slider-container">
            <p>Smoke Level</p>
            <div class="slider-container-left">
                <div class="slidecontainer">
                    <input
                        type="range"
                        min="0"
                        max="10"
                        value="5"
                        class="slider"
                        id="smokeLevel"
                    />
                </div>
            </div>
            <div class="slider-container-right">
                <p><span id="smokeLevelIndicator"></span></p>
            </div>
        </div>

        <br /><br />

        <div class="slider-container">
            <p>CO<sub>2</sub> Level</p>
            <div class="slider-container-left">
                <div class="slidecontainer">
                    <input
                        type="range"
                        min="0"
                        max="10"
                        value="5"
                        class="slider"
```

```
                        id="co2Level"
                    />
                </div>
            </div>
            <div class="slider-container-right">
                <p><span id="co2LevelIndicator"></span></p>
            </div>
        </div>

        <br /><br /><br />

        <p id="reading-count-text">
            <span id="reading-count-value">0</span> Readings has been sent
        </p>

        <script src="index.js"></script>
    </body>
</html>
```

[End of the Document]