A picture containing drawing

Description automatically generated

Sri Lanka Institute of Information Technology

Assignment 02

REST API

Distributed Systems (SE3020)

Group members:

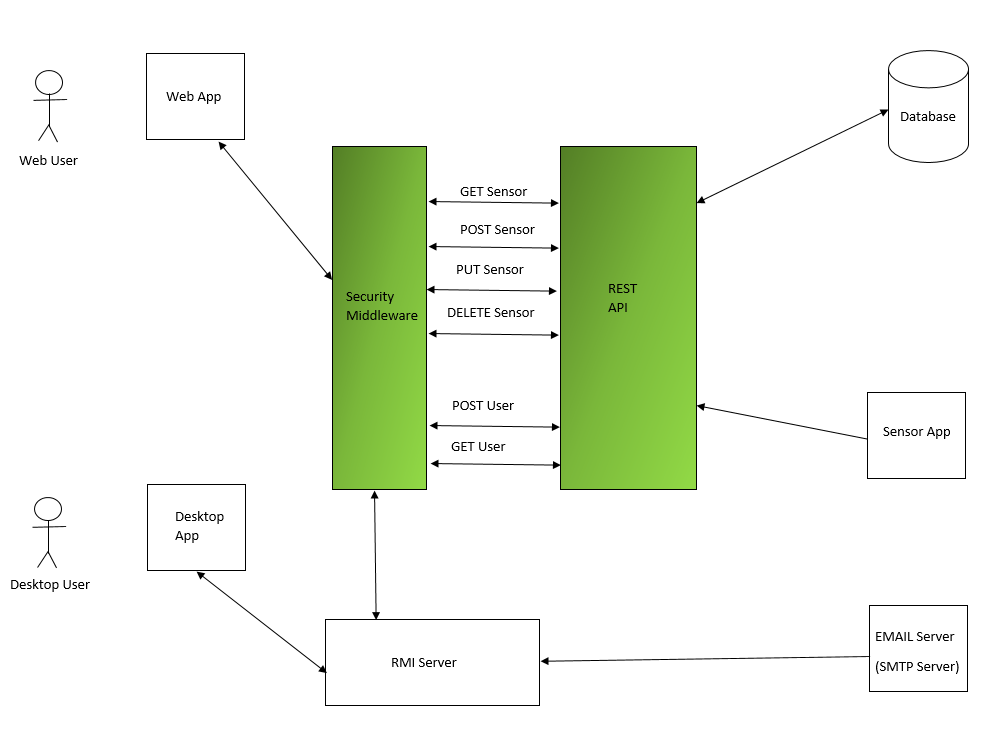
IT18140576 - Pinnawala N. M

IT18129922 - Wijerathna W. A. K. T

IT18136784 - Senevirathna B. W. H. K

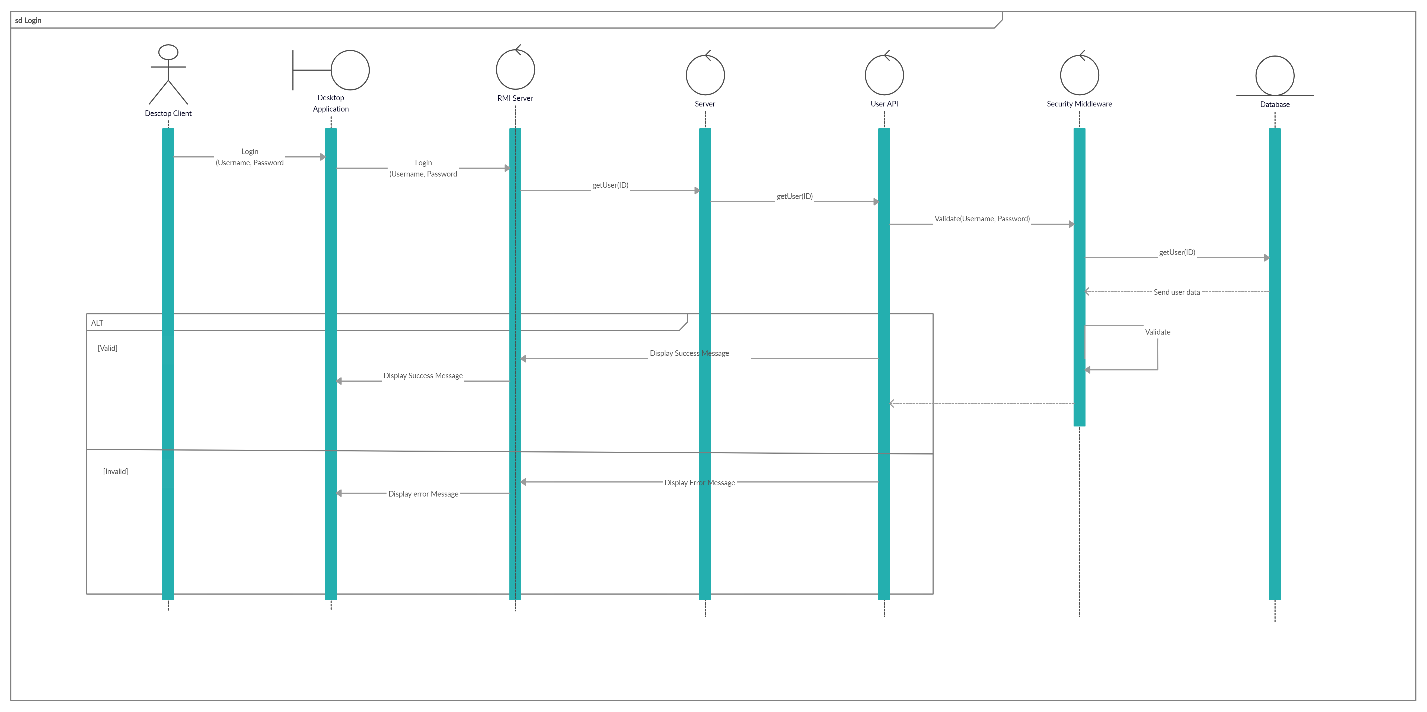
IT18163278 - Thilakaratne B. M. S. A

**High Level Architectural Diagram**



**Figure 1: High Level Architectural Diagram**

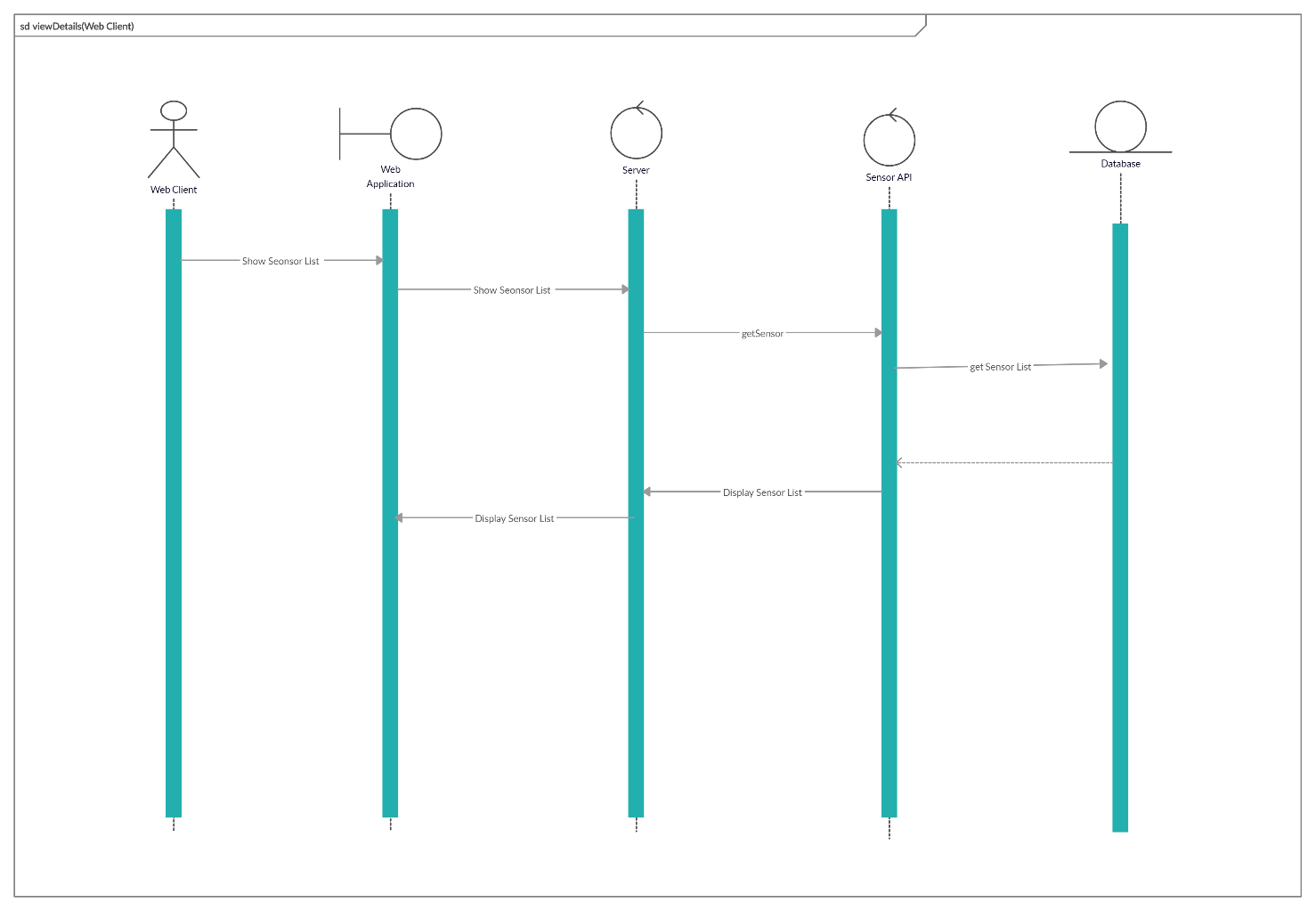
**Service Interfaces**

* **POST User**
* **GET User**

**Figure 2: POST User & GET User Sequential Diagram**

The administrator enters the credentials calling the login method of the desktop application interface. These credentials will be passed on till the User API via the RMI Server and the intermediary server which connects the desktop application with the User API. The User API calls the method to validate credentials of the security middleware. This security middleware searches the database for matching credentials. If the authentication is successful (if the JSON web token matches with that of a record stored in the database), it notifies the User API that the credentials are valid and thereby, the User API will send a success message to display on the application interface. If the authentication fails, the User API sends an error message to the application interface.

* **GET Sensor**
* Web Client

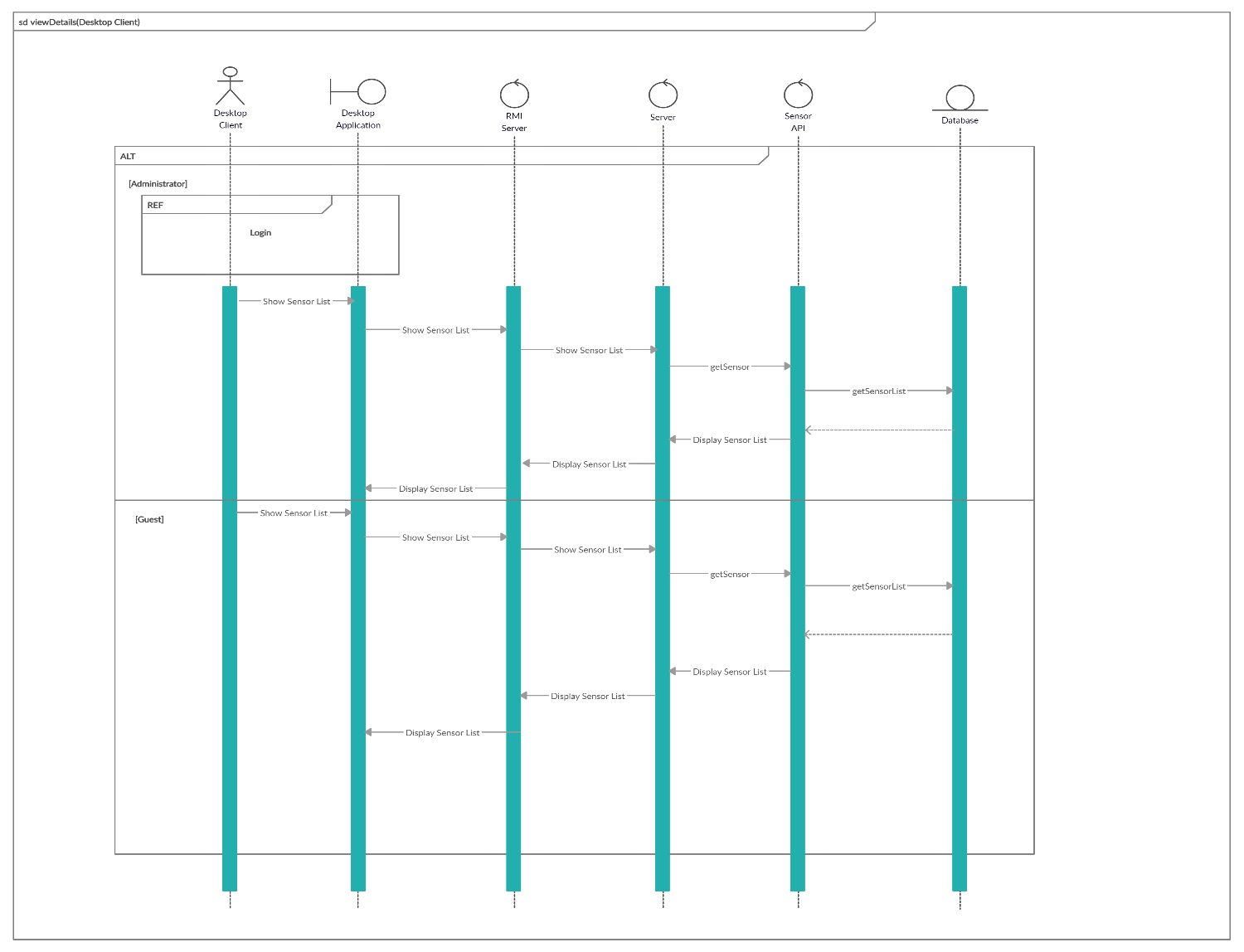


**Figure 3: GET Sensor (Web Client) Sequential Diagram**

The web application can be viewed by anyone without logging into the system.

Once the user opens the web application the view sensor method in the desktop interface will be called. It will then pass this message invoking the view sensor method of the intermediary server. Then the intermediary server invokes the GET route of the Sensor API. The Sensor API will retrieve the sensor list from the database and send them back to the web application interface via the intermediary server.

* Desktop Client

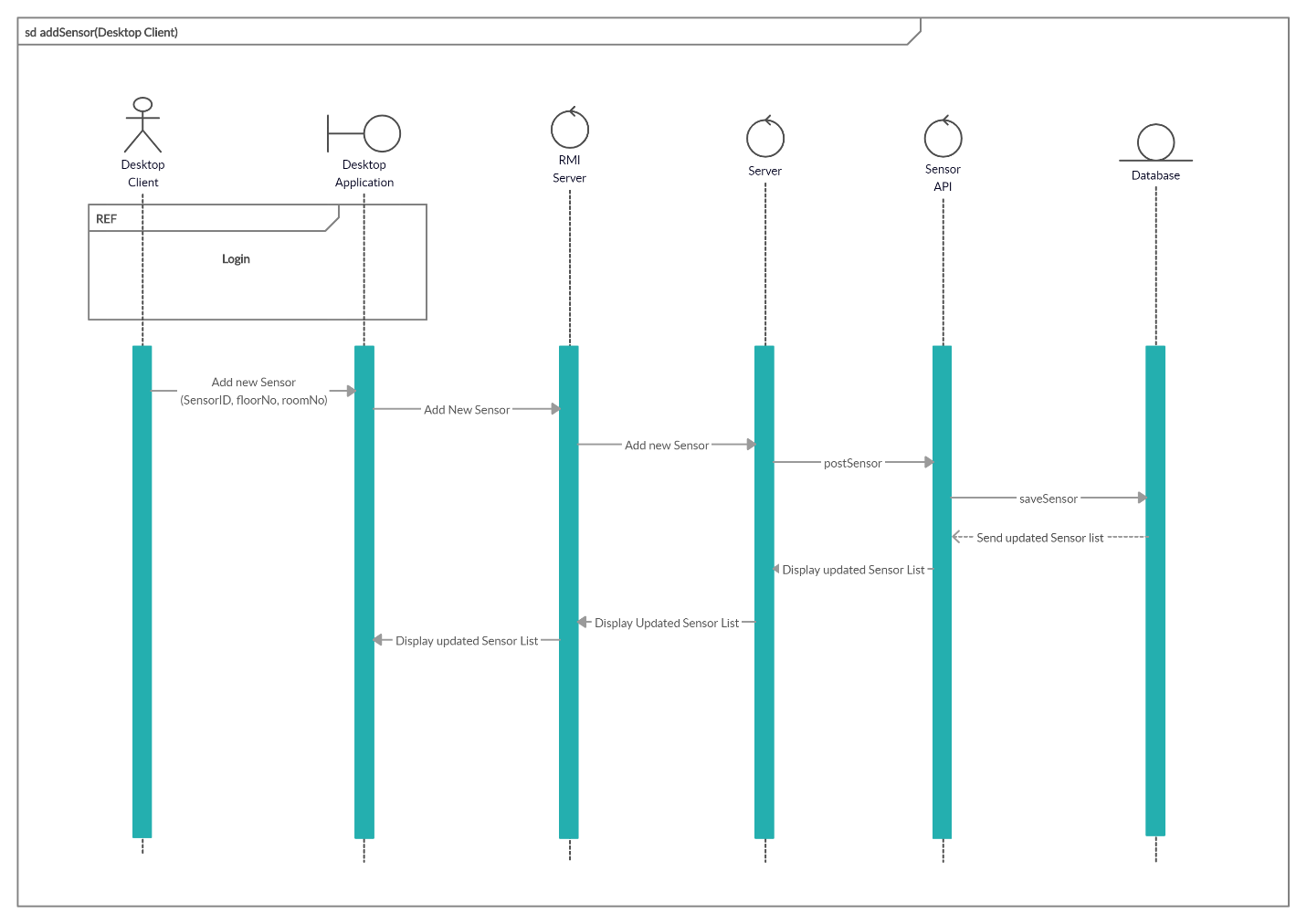


**Figure 4: GET Sensor (Desktop Client) Sequential Diagram**

The desktop application can be viewed by two types of users. They are the administrators and the guests.

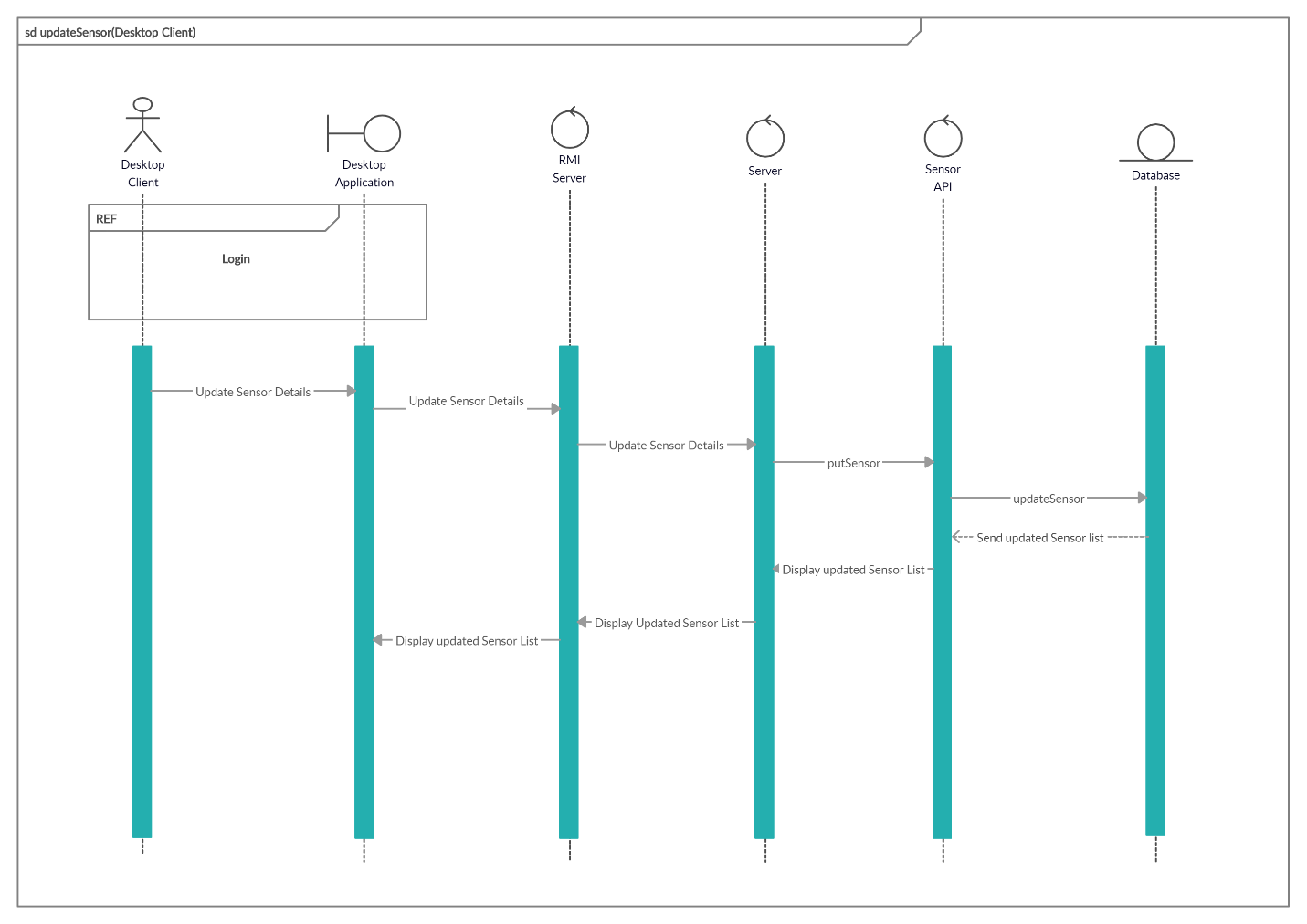
The guests can view the sensor list without logging into the system. If a guest calls the view sensor method in the desktop interface, it will pass this message invoking methods of the RMI server and the intermediary server. Then the intermediary server invokes the GET route of the Sensor API. The Sensor API will retrieve the sensor list from the database and send them back to the desktop interface via the intermediary servers.

Administrators can log into the system. Once an administrator logs into the system, the view sensor method of the desktop application will be invoked. Then the application sends this message requesting sensor details via the RMI server and the intermediary server. The intermediary server invokes the GET route of the Sensor API. Then the Sensor API will retrieve the sensor list from the database and send them back to the desktop interface via the intermediary servers. The administrator is allowed to update or delete sensors in this view.

* **POST Sensor**

**Figure 5: POST Sensor Sequential Diagram**

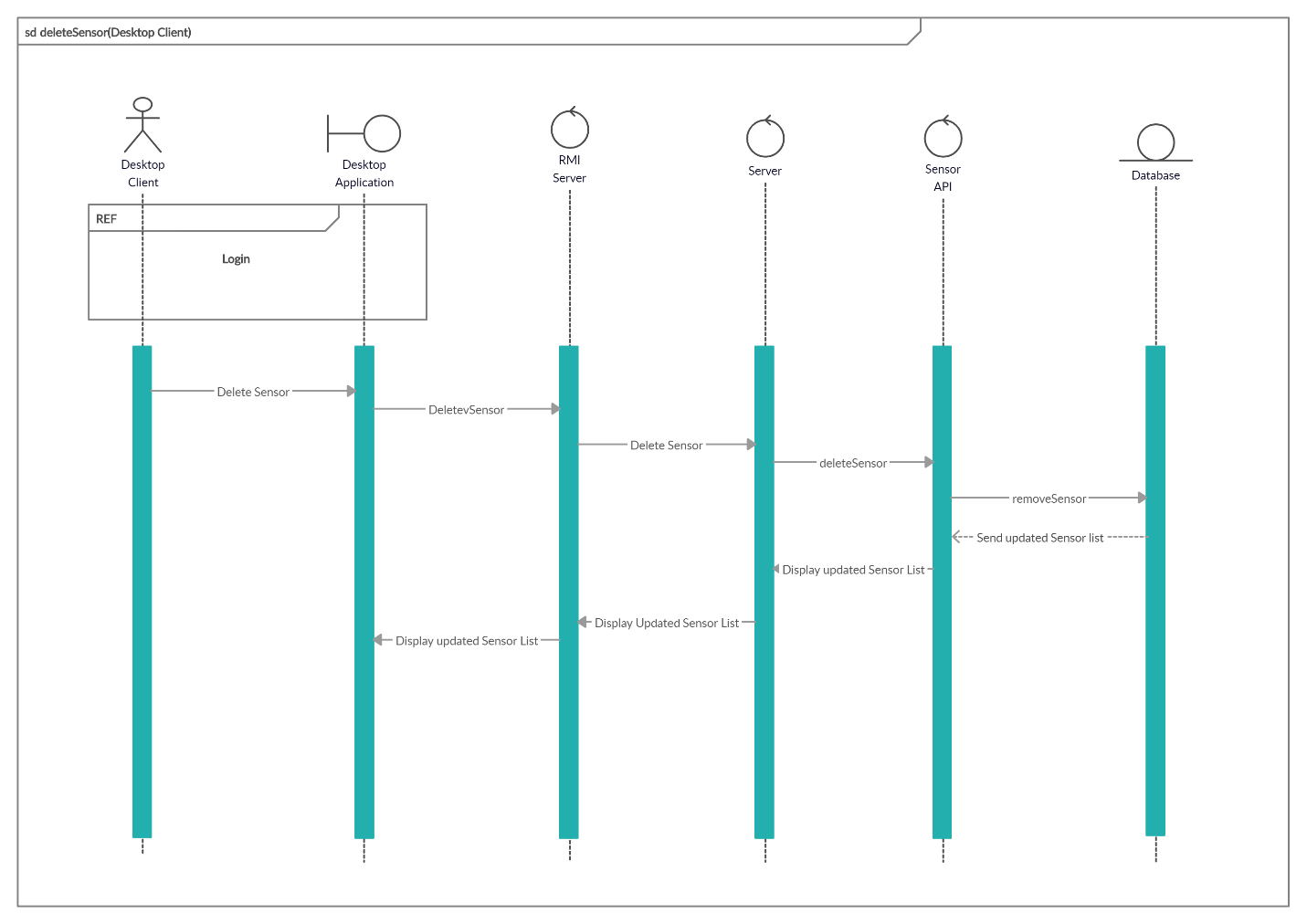
Only an administrator can add new sensors to the system and also the administrator has to be logged into the system. Once an administrator logs into the system, they can invoke the add new sensor method of the desktop application interface including all the data of the new sensor. Then the application will send the data via the RMI server and the intermediary server. The intermediary server will call the POST route of the Sensor API. Then the Sensor API sends and saves data of the new sensor in the database. The database will send back the updated sensor list to the Sensor API. Thereby, the Sensor API calls the method to display the updated sensor list on the application interfaces via the intermediary servers.

* **PUT Sensor**

**Figure 6: PUT Sensor Sequential Diagram**

Same as adding new sensors, only an administrator can update sensor details. This administrator has to be logged into the system. Once an administrator logs into the system, they can call the update sensor details method of the desktop application. The application then sends the data which need to be updated via the RMI server and the intermediary server. The intermediary server calls the PUT route in the Sensor API. Then the Sensor API will update the sensor details in the database. In return, the database sends the updated sensor list to the Sensor API. This updated list is sent to display on the application interfaces via the intermediary server and the RMI server.

* **DELETE Sensor**

****

**Figure 7: DELETE Sensor Sequential Diagram**

Same as updating sensors, only an administrator can delete sensors and this administrator has to be logged into the system. Once an administrator logs into the system, they can call the delete sensor method of the desktop application. The application then sends the message to delete the sensor via the RMI server and the intermediary server. The intermediary server calls the DELETE route in the Sensor API. Then the Sensor API will delete the sensor from the database. In return, the database sends the updated sensor list to the Sensor API. This updated list is sent to display on the application interfaces via the intermediary server and the RMI server.

**Authentication / Security Mechanisms**

This scenario consists of a client and a server. The client sends auth data (email & password) to the server. If the user is registering, this auth data is saved in the database. The password is saved as a hash created using a module called “bcrypt” to make it more secure. If the user is trying to login, the details are compared with the details saved in the database. Since RESTful API is stateless, instead of a session the server responses with a token. This token is then stored by the client and used/attached in future requests. When a request is made by the user, the token stored is sent with the request to be verified. For this we have used JSON Web Token (JWT).

Client

Storage

Request + Token

Token

Auth Data

Server

**Appendix**

1. REST API

* Index.js

const express = require("express");

const bodyParser = require("body-parser");

const cors = require("cors");

const mongoose = require("mongoose");

const routes = require("./routes/api");

const userRoutes = require("./routes/userapi");

//Setup Express

const app = express();

//Connect to DB

mongoose.connect("mongodb://localhost/sensorDB");

mongoose.Promise = global.Promise;

//Page Content

app.use(express.static("public"));

app.use(cors());

app.use(bodyParser.urlencoded({ extends: true }));

app.use(bodyParser.json());

//Initialize Routes

app.use("/api", routes);

app.use("/users", userRoutes);

//Error Handling

app.use(function (err, req, res, next) {

res.status(422).send({ error: err.message });

});

//Listening to Requests

app.listen(4000, function () {

console.log("Now listening for requests");

});

* api.js

const express = require("express");

const router = express.Router();

const Sensor = require("../models/sensor");

const checkAuth = require("../middleware/checkAuth");

//Get a list from DB

router.get("/sensors", function (req, res, next) {

Sensor.find({}).then(function (sensor) {

res.send(sensor);

});

});

//Add a new item to DB

router.post("/sensors", checkAuth, function (req, res, next) {

Sensor.create(req.body)

.then(function (sensor) {

res.send(sensor);

})

.catch(next);

});

//Update an item in DB

router.put("/sensors/:id", checkAuth, function (req, res, next) {

Sensor.findByIdAndUpdate({ \_id: req.params.id }, req.body).then(function () {

Sensor.findOne({ \_id: req.params.id }).then(function (sensor) {

res.send(sensor);

});

});

});

//Delete an item from DB

router.delete("/sensors/:id", checkAuth, function (req, res, next) {

Sensor.findByIdAndRemove({ \_id: req.params.id }).then(function (sensor) {

res.send(sensor);

});

});

module.exports = router;

* userapi.js

const express = require("express");

const router = express.Router();

const User = require("../models/user");

const bcrypt = require("bcrypt");

const jwt = require("jsonwebtoken");

//User Registration

router.post("/signup", (req, res, next) => {

User.find({ email: req.body.email })

.exec()

.then((user) => {

if (user.length >= 1) {

return res.status(409).json({

message: "Mail already exists",

});

} else {

bcrypt.hash(req.body.password, 10, (err, hash) => {

if (err) {

return res.status(500).json({

error: err,

});

} else {

const user = new User({

email: req.body.email,

password: hash,

});

user

.save()

.then((result) => {

console.log(result);

res.status(201).json({

message: "User created",

});

})

.catch((err) => {

console.log(err);

res.status(500).json({

error: err,

});

});

}

});

}

});

});

//User Login

router.post("/login", (req, res, next) => {

User.find({ email: req.body.email })

.exec()

.then((user) => {

if (user.length < 1) {

return res.status(401).json({

message: "Login failed.",

});

}

bcrypt.compare(req.body.password, user[0].password, (err, result) => {

if (err) {

return res.status(401).json({

message: "Login failed.",

});

}

if (result) {

const token = jwt.sign(

{

email: user[0].email,

},

"secret"

);

return res.status(200).json({

message: "Auth successful.",

token: token,

});

}

return res.status(401).json({

message: "Login failed.",

});

});

})

.catch((err) => {

console.log(err);

res.status(500).json({

error: err,

});

});

});

module.exports = router;

1. Database Models

* Sensor.js

const mongoose = require("mongoose");

const Schema = mongoose.Schema;

const SensorSchema = new Schema({

sensorID: {

type: String,

required: [true, "ID field is required"],

},

floorNo: {

type: Number,

required: [true, "Floor number is required"],

},

roomNo: {

type: Number,

required: [true, "Room number is required"],

},

c\_level: {

type: Number,

},

s\_level: {

type: Number,

},

status: {

type: Boolean,

default: false,

},

});

const Sensor = mongoose.model("sensor", SensorSchema);

module.exports = Sensor;

* user.js

const mongoose = require("mongoose");

const Schema = mongoose.Schema;

const UserSchema = new Schema({

email: { type: String, required: true, unique: true },

password: { type: String, required: true },

});

const User = mongoose.model("user", UserSchema);

module.exports = User;

1. Authentication

* checkAuth.js

const jwt = require("jsonwebtoken");

module.exports = (req, res, next) => {

try {

const token = req.headers.authorization.split(" ")[1];

const decoded = jwt.verify(token, "secret");

req.userData = decoded;

next();

} catch (error) {

return res.status(401).json({

message: "Auth failed.",

});

}

};

1. Web Client

* Index.html

<title>Sensor Monitor</title>

<noscript>You need to enable JavaScript to run this app.</noscript>

* App.js

import React, { Component } from "react";

import "bootstrap/dist/css/bootstrap.min.css";

import SensorTable from "./components/SensorTable";

class App extends Component {

constructor(props) {

super(props);

this.state = {

alarmId: "",

floorNo: "",

roomNo: "",

status: false,

};

}

render() {

return (

<div className="container">

<br />

<br />

<h1 className="text-center">Sensor Monitoring System</h1>

<br />

<div className="text-center">

<br />

<SensorTable />

</div>

</div>

);

}

}

export default App;

* SensorTable.js

import React, { Component } from "react";

import SensorTableRow from "./SensorTableRow";

class SensorTable extends Component {

constructor(props) {

super(props);

this.state = { sensors: [] };

}

componentDidMount = async () => {

try {

setInterval(async () => {

fetch("http://localhost:4000/api/sensors/")

.then(function (data) {

return data.json();

})

.then((json) => {

this.setState({

sensors: json,

});

});

}, 40000);

} catch (err) {

console.log(err);

}

};

tableRow() {

console.log(this.state.sensors);

return this.state.sensors.map(function (oneSensor, i) {

return <SensorTableRow oneSensor={oneSensor} key={i} />;

});

}

render() {

return (

<div className="col-12 text-center">

<div className="card">

<table className="table table-hover">

<thead>

<tr>

<th scope="col">Sensor</th>

<th scope="col">Floor Number</th>

<th scope="col">Room Number</th>

<th scope="col">Smoke Level</th>

<th scope="col">CO2 Level</th>

<th scope="col">Status</th>

</tr>

</thead>

<tbody>{this.tableRow()}</tbody>

</table>

</div>

</div>

);

}

}

export default SensorTable;

* SensorTableRow.js

import React, { Component } from "react";

class SensorTableRow extends Component {

render() {

return (

<tr>

<td>{this.props.oneSensor.sensorID}</td>

<td>{this.props.oneSensor.floorNo}</td>

<td>{this.props.oneSensor.roomNo}</td>

<td>{this.props.oneSensor.s\_level}</td>

<td>{this.props.oneSensor.c\_level}</td>

<td>

{this.props.oneSensor.s\_level > 5 ||

this.props.oneSensor.c\_level > 5 ? (

<i className="fa fa-circle" style={{ color: "red" }} />

) : (

<i className="fa fa-circle" style={{ color: "green" }} />

)}

</td>

</tr>

);

}

}

export default SensorTableRow;

1. Desktop Client & RMI

* RMI\_Server.java

package sendhttp;

import java.awt.event.ActionEvent;

import java.awt.event.ActionListener;

import java.io.IOException;

import java.net.URI;

import java.net.http.HttpClient;

import java.net.http.HttpRequest;

import java.net.http.HttpResponse;

import java.rmi.AlreadyBoundException;

import java.rmi.RemoteException;

import java.rmi.registry.LocateRegistry;

import java.rmi.registry.Registry;

import java.rmi.server.UnicastRemoteObject;

import java.util.Date;

import java.util.Properties;

import javax.mail.Message;

import javax.mail.MessagingException;

import javax.mail.PasswordAuthentication;

import javax.mail.Session;

import javax.mail.Transport;

import javax.mail.internet.AddressException;

import javax.mail.internet.InternetAddress;

import javax.mail.internet.MimeMessage;

import javax.swing.Timer;

import org.apache.http.client.methods.HttpDelete;

import org.apache.http.client.methods.HttpPost;

import org.apache.http.client.methods.HttpPut;

import org.apache.http.entity.StringEntity;

import org.apache.http.impl.client.CloseableHttpClient;

import org.apache.http.impl.client.HttpClientBuilder;

import org.json.JSONArray;

import org.json.JSONObject;

public class RMI\_Server extends UnicastRemoteObject implements RMI\_Service {

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

Registry registry = LocateRegistry.createRegistry(5099);

registry.bind("FireSensorService", new RMI\_Server());

System.out.println("Server started.");

//Timer to refresh the window

Timer t = new Timer(0, null);

t.addActionListener(new ActionListener() {

@Override

public void actionPerformed(ActionEvent e) {

try {

checkLevelsRepeat();

} catch (Exception e1) {

e1.printStackTrace();

}

}

});

t.setRepeats(true);

t.setDelay(15000);

t.start();

}

protected RMI\_Server() throws RemoteException {

super();

}

//Get Sensor

@Override

public String getSensorDetails() throws RemoteException {

HttpClient client = HttpClient.newHttpClient();

HttpRequest request = HttpRequest.newBuilder().uri(URI.create("http://localhost:4000/api/sensors/")).build();

return client.sendAsync(request, HttpResponse.BodyHandlers.ofString()).thenApply(HttpResponse::body).thenApply((responseBody) -> parse(responseBody)).join();

}

public static String parse(String responseBody) {

return responseBody;

}

//Add Sensor

@Override

public boolean addSensor(String sensorID, int floorNo, int roomNo, int c\_level, int s\_level) throws RemoteException {

boolean res = false;

JSONObject json = new JSONObject();

json.put("sensorID", sensorID);

json.put("floorNo", floorNo);

json.put("roomNo", roomNo);

json.put("c\_level", c\_level);

json.put("s\_level", s\_level);

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

try {

HttpPost request = new HttpPost("http://localhost:4000/api/sensors/");

StringEntity params = new StringEntity(json.toString());

//Headers and Authentication Token

request.addHeader("content-type", "application/json");

request.addHeader("Authorization", "Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJlbWFpbCI6ImhlbGxvQGdtYWlsLmNvbSIsImlhdCI6MTU4ODQ1NDE2N30.yG30vjE6jkIXr\_WMQ3PIPfloT7xtya-pJVLqkfZza2s");

request.setEntity(params);

org.apache.http.HttpResponse response = httpClient.execute(request);

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

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;

}

//Edit Sensor

@Override

public boolean editSensor(String id, String sensorID, int floorNo, int roomNo, int c\_level, int s\_level) throws RemoteException {

boolean res = false;

JSONObject json = new JSONObject();

json.put("sensorID", sensorID);

json.put("floorNo", floorNo);

json.put("roomNo", roomNo);

json.put("c\_level", c\_level);

json.put("s\_level", s\_level);

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

try {

HttpPut request = new HttpPut("http://localhost:4000/api/sensors/" + id);

StringEntity params = new StringEntity(json.toString());

//Headers and Authentication Token

request.addHeader("content-type", "application/json");

request.addHeader("Authorization","Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJlbWFpbCI6ImhlbGxvQGdtYWlsLmNvbSIsImlhdCI6MTU4ODQ1NDE2N30.yG30vjE6jkIXr\_WMQ3PIPfloT7xtya-pJVLqkfZza2s");

request.setEntity(params);

org.apache.http.HttpResponse response = httpClient.execute(request);

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

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;

}

//Delete Sensor

@Override

public boolean deleteSensor(String id) throws RemoteException {

boolean res = false;

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

try {

HttpDelete request = new HttpDelete("http://localhost:4000/api/sensors/" + id);

//Headers and Authentication Token

request.addHeader("content-type", "application/json");

request.addHeader("Authorization","Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJlbWFpbCI6ImhlbGxvQGdtYWlsLmNvbSIsImlhdCI6MTU4ODQ1NDE2N30.yG30vjE6jkIXr\_WMQ3PIPfloT7xtya-pJVLqkfZza2s");

org.apache.http.HttpResponse response = httpClient.execute(request);

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

} catch (Exception ex) {

System.out.println(ex);

} finally {

try {

httpClient.close();

} catch (IOException e) {

e.printStackTrace();

}

}

return res;

}

//Login

@Override

public String loginDetails(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 {

HttpPost request = new HttpPost("http://localhost:4000/users/login/");

StringEntity params = new StringEntity(json.toString());

request.addHeader("content-type", "application/json");

request.setEntity(params);

org.apache.http.HttpResponse response = httpClient.execute(request);

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;

}

//Check Smoke and CO2 Levels

public static void checkLevelsRepeat() {

HttpClient client = HttpClient.newHttpClient();

HttpRequest request = HttpRequest.newBuilder().uri(URI.create("http://localhost:4000/api/sensors/")).build();

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

}

//Call sendMail()

private static String checkLevels(String responseBody) {

JSONArray sensors = new JSONArray(responseBody);

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

JSONObject obj = sensors.getJSONObject(i);

int co2Level = obj.getInt("c\_level");

int smokeLevel = obj.getInt("s\_level");

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

//sendMail("nishith.pinnawala@gmail.com");

System.out.println("FIRE ALERT: Email Sent.");

sendMail();

}

}

return null;

}

//Send Email

public static void sendMail() {

//Dummy Email Service - Fake SMTP server

try {

InternetAddress[] distributionList = InternetAddress.parse("admin@firealert.com",false);

String from = "emergency@firealert.com";

String subject = "FIRE ALERT";

Properties props = new Properties();

props.put("mail.smtp.host", "localhost");

props.put("mail.smtp.port", "25");

Session session = Session.getDefaultInstance(props, null);

session.setDebug(false);

Message msg = new MimeMessage(session);

String message = "Fire Alert!";

msg.setContent(message, "text/html; charset=utf-8");

msg.setFrom(new InternetAddress(from));

msg.setRecipients(Message.RecipientType.TO, distributionList);

msg.setSubject(subject);

msg.setSentDate(new Date());

Transport.send(msg);

} catch (Exception ex) {

ex.printStackTrace();

}

//Uncomment to use Google Mail (Part 1)

/\*

Properties props = new Properties();

props.put("mail.smtp.auth" , "true");

props.put("mail.smtp.starttls.enable" , "true");

props.put("mail.smtp.ssl.trust", "smtp.gmail.com");

props.put("mail.smtp.host" , "smtp.gmail.com");

props.put("mail.smtp.port" , "587");

String eMail = "firesysds@gmail.com";

String password = "desktopapp123";

Session session = Session.getInstance(props,

new javax.mail.Authenticator() {

protected PasswordAuthentication getPasswordAuthentication() {

return new PasswordAuthentication(eMail , password);

}

});

Message message = prepareMessage(session , eMail , rsp);

try {

Transport.send(message);

System.out.println("Email sent.");

} catch (MessagingException e) {

e.printStackTrace();

}\*/

}

//Uncomment to use Google Email (Part 2)

/\*Create Email Content

private static Message prepareMessage(Session session, String eMail , String rsp) {

Message message = new MimeMessage(session);

try {

message.setFrom(new InternetAddress(eMail));

message.setRecipient(Message.RecipientType.TO, new InternetAddress(rsp));

message.setSubject("Fire Alert!");

message.setText("There's a fire.");

return message;

} catch (AddressException e) {

e.printStackTrace();

} catch (MessagingException e) {

e.printStackTrace();

}

return null;

}\*/

}

* RMI\_Service.java

package sendhttp;

import java.rmi.Remote;

import java.rmi.RemoteException;

public interface RMI\_Service extends Remote {

public String getSensorDetails() throws RemoteException;

public boolean addSensor(String sensorID, int floorNo, int roomNo, int c\_level, int s\_level) throws RemoteException;

public boolean editSensor(String id, String sensorID, int floorNo, int roomNo, int c\_level, int s\_level) throws RemoteException;

public boolean deleteSensor(String id) throws RemoteException;

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

}

* Login.java

package desktopClient;

import java.awt.Color;

import java.awt.EventQueue;

import java.awt.Font;

import java.awt.event.ActionEvent;

import java.awt.event.ActionListener;

import java.net.MalformedURLException;

import java.rmi.Naming;

import java.rmi.NotBoundException;

import java.rmi.RemoteException;

import javax.swing.JButton;

import javax.swing.JFrame;

import javax.swing.JLabel;

import javax.swing.JPanel;

import javax.swing.JPasswordField;

import javax.swing.JTextField;

import javax.swing.border.EmptyBorder;

import sendhttp.RMI\_Service;

public class Login extends JFrame{

private JPanel contentPane;

private JTextField txtemail;

private JPasswordField txtpassword;

private static Login frame;

public static void main(String[] args) {

EventQueue.invokeLater(new Runnable() {

public void run() {

try {

frame = new Login();

frame.setLocationRelativeTo(null);

frame.setVisible(true);

} catch (Exception e) {

e.printStackTrace();

}

}

});

}

public Login() {

setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

setBounds(350, 250, 500, 500);

contentPane = new JPanel();

contentPane.setBorder(new EmptyBorder(5, 5, 5, 5));

setContentPane(contentPane);

contentPane.setLayout(null);

JLabel lblTitle = new JLabel("Sensor Monitoring System");

lblTitle.setFont(new Font("Tahoma", Font.PLAIN, 25));

lblTitle.setBounds(105, 55, 300, 50);

contentPane.add(lblTitle);

JLabel lblEmail = new JLabel("Email:");

lblEmail.setFont(new Font("Tahoma", Font.PLAIN, 18));

lblEmail.setBounds(100, 159, 77, 16);

contentPane.add(lblEmail);

JLabel lblPass = new JLabel("Password:");

lblPass.setFont(new Font("Tahoma", Font.PLAIN, 18));

lblPass.setBounds(100, 217, 85, 22);

contentPane.add(lblPass);

JLabel lblFail = new JLabel("Login failed.");

lblFail.setForeground(Color.RED);

lblFail.setFont(new Font("Tahoma", Font.PLAIN, 13));

lblFail.setBounds(210, 382, 185, 22);

contentPane.add(lblFail);

lblFail.setVisible(false);

txtemail = new JTextField();

txtemail.setFont(new Font("Tahoma", Font.PLAIN, 16));

txtemail.setBounds(195, 154, 205, 27);

contentPane.add(txtemail);

txtemail.setColumns(10);

txtpassword = new JPasswordField();

txtpassword.setFont(new Font("Tahoma", Font.PLAIN, 16));

txtpassword.setBounds(195, 215, 205, 27);

contentPane.add(txtpassword);

txtpassword.setColumns(10);

JButton btnLogin = new JButton("Login");

btnLogin.setFont(new Font("Tahoma", Font.PLAIN, 15));

btnLogin.addActionListener(new ActionListener() {

public void actionPerformed(ActionEvent e) {

String email = txtemail.getText();

String password = txtpassword.getText();

RMI\_Service service;

String result = null;

try {

service = (RMI\_Service) Naming.lookup("rmi://localhost:5099/FireSensorService");

result = service.loginDetails(email, password);

} catch (MalformedURLException | RemoteException | NotBoundException ex) {

ex.printStackTrace();

}

System.out.println(result);

if (result.equalsIgnoreCase("success")) {

dispose();

RMI\_Test rmitest = new RMI\_Test(true);

rmitest.main(null);

} else {

lblFail.setVisible(true);

}

}

});

btnLogin.setBounds(140, 300, 100, 40);

contentPane.add(btnLogin);

JButton btnGuest = new JButton("Guest");

btnGuest.setFont(new Font("Tahoma", Font.PLAIN, 15));

btnGuest.addActionListener(new ActionListener() {

public void actionPerformed(ActionEvent e) {

dispose();

RMI\_Test rmitest = new RMI\_Test(false);

rmitest.main(null);

}

});

btnGuest.setBounds(255, 300, 100, 40);

contentPane.add(btnGuest);

}

}

* AddSensor.java

package desktopClient;

import java.awt.BorderLayout;

import java.awt.Color;

import java.awt.FlowLayout;

import java.awt.Font;

import java.awt.event.ActionEvent;

import java.awt.event.ActionListener;

import java.net.MalformedURLException;

import java.rmi.Naming;

import java.rmi.NotBoundException;

import java.rmi.RemoteException;

import javax.swing.JButton;

import javax.swing.JDialog;

import javax.swing.JLabel;

import javax.swing.JPanel;

import javax.swing.JTextField;

import javax.swing.border.EmptyBorder;

import desktopClient.AddSensor;

import desktopClient.RMI\_Test;

import sendhttp.RMI\_Service;

public class AddSensor extends JDialog {

private final JPanel contentPanel = new JPanel();

private JTextField txtsensorid;

private JTextField txtfloorno;

private JTextField txtroomno;

private JTextField txtsmokelvl;

private JTextField txtcolvl;

private boolean res;

public static void main(String[] args) {

try {

AddSensor dialog = new AddSensor();

dialog.setDefaultCloseOperation(JDialog.DISPOSE\_ON\_CLOSE);

dialog.setVisible(true);

} catch (Exception e) {

e.printStackTrace();

}

}

public AddSensor() {

setBounds(250, 300, 360, 350);

getContentPane().setLayout(new BorderLayout());

contentPanel.setBorder(new EmptyBorder(5, 5, 5, 5));

getContentPane().add(contentPanel, BorderLayout.CENTER);

contentPanel.setLayout(null);

JLabel lblSensorName = new JLabel("Sensor Name:");

lblSensorName.setFont(new Font("Tahoma", Font.PLAIN, 15));

lblSensorName.setBounds(25, 25, 100, 30);

contentPanel.add(lblSensorName);

JLabel lblFloorNo = new JLabel("Floor No:");

lblFloorNo.setFont(new Font("Tahoma", Font.PLAIN, 15));

lblFloorNo.setBounds(25, 75, 100, 30);

contentPanel.add(lblFloorNo);

JLabel lblRoomNo = new JLabel("Room No:");

lblRoomNo.setFont(new Font("Tahoma", Font.PLAIN, 15));

lblRoomNo.setBounds(25, 125, 100, 30);

contentPanel.add(lblRoomNo);

JLabel lblSmokeLevel = new JLabel("Smoke Level:");

lblSmokeLevel.setFont(new Font("Tahoma", Font.PLAIN, 15));

lblSmokeLevel.setBounds(25, 175, 100, 30);

contentPanel.add(lblSmokeLevel);

JLabel lblCOLevel = new JLabel("CO2 Level:");

lblCOLevel.setFont(new Font("Tahoma", Font.PLAIN, 15));

lblCOLevel.setBounds(25, 225, 100, 30);

contentPanel.add(lblCOLevel);

txtsensorid = new JTextField();

txtsensorid.setFont(new Font("Tahoma", Font.PLAIN, 15));

txtsensorid.setBounds(150, 25, 170, 30);

contentPanel.add(txtsensorid);

txtsensorid.setColumns(10);

txtfloorno = new JTextField();

txtfloorno.setBounds(150, 75, 170, 30);

contentPanel.add(txtfloorno);

txtfloorno.setColumns(10);

txtroomno = new JTextField();

txtroomno.setBounds(150, 125, 170, 30);

contentPanel.add(txtroomno);

txtroomno.setColumns(10);

txtsmokelvl = new JTextField();

txtsmokelvl.setBounds(150, 175, 170, 30);

contentPanel.add(txtsmokelvl);

txtsmokelvl.setColumns(10);

txtcolvl = new JTextField();

txtcolvl.setBounds(150, 225, 170, 30);

contentPanel.add(txtcolvl);

txtcolvl.setColumns(10);

{

JPanel buttonPane = new JPanel();

buttonPane.setLayout(new FlowLayout(FlowLayout.RIGHT));

getContentPane().add(buttonPane, BorderLayout.SOUTH);

{

JButton saveButton = new JButton("Save");

saveButton.addActionListener(new ActionListener() {

public void actionPerformed(ActionEvent e) {

String id = txtsensorid.getText();

int floor = Integer.parseInt(txtfloorno.getText());

int room = Integer.parseInt(txtroomno.getText());

int smoke = Integer.parseInt(txtsmokelvl.getText());

int co = Integer.parseInt(txtcolvl.getText());

RMI\_Service service;

try {

service = (RMI\_Service) Naming.lookup("rmi://localhost:5099/FireSensorService");

try {

res = service.addSensor(id, floor, room, co, smoke);

} catch (RemoteException e1) {

e1.printStackTrace();

}

} catch (MalformedURLException | RemoteException | NotBoundException ex) {

ex.printStackTrace();

}

dispose();

RMI\_Test rmiTest = new RMI\_Test(true);

rmiTest.main(null);

}

});

saveButton.setActionCommand("Save");

buttonPane.add(saveButton);

getRootPane().setDefaultButton(saveButton);

}

{

JButton cancelButton = new JButton("Cancel");

cancelButton.addActionListener(new ActionListener() {

public void actionPerformed(ActionEvent e) {

dispose();

RMI\_Test rmiTest = new RMI\_Test(true);

rmiTest.main(null);

}

});

cancelButton.setActionCommand("Cancel");

buttonPane.add(cancelButton);

}

}

}

}

* ViewSensor.java

package desktopClient;

import java.awt.Color;

import java.awt.Font;

import java.awt.event.ActionEvent;

import java.awt.event.ActionListener;

import javax.swing.JButton;

import javax.swing.JFrame;

import javax.swing.JLabel;

import javax.swing.JPanel;

import javax.swing.JScrollPane;

import javax.swing.JTable;

import desktopClient.EditSensor;

public class ViewSensor extends JPanel {

private String id;

private String sensorId;

private int floorNo;

private int roomNo;

private int c\_level;

private int s\_level;

private boolean status;

private boolean isLogged;

private JFrame contentFrame;

public ViewSensor(String id, String sensorId, int floorNo, int roomNo, int c\_level, int s\_level, boolean status, boolean isLogged, JFrame contentFrame) {

super();

setLayout(null);

this.id = id;

this.sensorId = sensorId;

this.floorNo = floorNo;

this.roomNo = roomNo;

this.c\_level = c\_level;

this.s\_level = s\_level;

this.status = status;

this.isLogged = isLogged;

this.contentFrame = contentFrame;

//Layout

JPanel panel = new JPanel();

panel.setBackground(this.c\_level > 5 || this.s\_level > 5 ? new Color(217, 83, 79) : new Color(92, 184, 92));

panel.setBounds(0, 5, 390, 130);

add(panel);

panel.setLayout(null);

//Section 1

JLabel label\_1 = new JLabel("Sensor:");

label\_1.setForeground(new Color(0, 0, 0));

label\_1.setFont(new Font("Tahoma", Font.PLAIN, 18));

label\_1.setBounds(25, 25, 200, 30);

panel.add(label\_1);

JLabel label\_sensorId = new JLabel(this.sensorId);

label\_sensorId.setForeground(new Color(0, 0, 0));

label\_sensorId.setFont(new Font("Tahoma", Font.BOLD, 18));

label\_sensorId.setBounds(100, 25, 200, 30);

panel.add(label\_sensorId);

JLabel label\_2 = new JLabel("Floor:");

label\_2.setForeground(new Color(0, 0, 0));

label\_2.setFont(new Font("Tahoma", Font.PLAIN, 18));

label\_2.setBounds(25, 50, 200, 30);

panel.add(label\_2);

JLabel label\_floorNo = new JLabel("" + this.floorNo);

label\_floorNo.setForeground(new Color(0, 0, 0));

label\_floorNo.setFont(new Font("Tahoma", Font.BOLD, 18));

label\_floorNo.setBounds(100, 50, 200, 30);

panel.add(label\_floorNo);

JLabel label\_3 = new JLabel("Room:");

label\_3.setForeground(new Color(0, 0, 0));

label\_3.setFont(new Font("Tahoma", Font.PLAIN, 18));

label\_3.setBounds(25, 75, 200, 30);

panel.add(label\_3);

JLabel label\_roomNo = new JLabel("" + this.roomNo);

label\_roomNo.setForeground(new Color(0, 0, 0));

label\_roomNo.setFont(new Font("Tahoma", Font.BOLD, 18));

label\_roomNo.setBounds(100, 75, 200, 30);

panel.add(label\_roomNo);

//Section 2

JLabel label\_4 = new JLabel("Smoke Level:");

label\_4.setForeground(new Color(0, 0, 0));

label\_4.setFont(new Font("Tahoma", Font.PLAIN, 18));

label\_4.setBounds(200, 25, 150, 30);

panel.add(label\_4);

JLabel label\_sLevel = new JLabel("" + this.s\_level);

label\_sLevel.setForeground(new Color(0, 0, 0));

label\_sLevel.setFont(new Font("Tahoma", Font.BOLD, 18));

label\_sLevel.setBounds(320, 25, 150, 30);

panel.add(label\_sLevel);

JLabel label\_5 = new JLabel("CO2 Level:");

label\_5.setForeground(new Color(0, 0, 0));

label\_5.setFont(new Font("Tahoma", Font.PLAIN, 18));

label\_5.setBounds(200, 50, 150, 30);

panel.add(label\_5);

JLabel label\_cLevel = new JLabel("" + this.c\_level);

label\_cLevel.setForeground(new Color(0, 0, 0));

label\_cLevel.setFont(new Font("Tahoma", Font.BOLD, 18));

label\_cLevel.setBounds(320, 50, 150, 30);

panel.add(label\_cLevel);

JButton btnEdit = new JButton("Edit");

btnEdit.setVisible(this.isLogged);

btnEdit.addActionListener(new ActionListener() {

public void actionPerformed(ActionEvent e) {

contentFrame.dispose();

EditSensor editSensor = new EditSensor(id, label\_sensorId.getText(), label\_floorNo.getText(), label\_roomNo.getText(), label\_cLevel.getText(), label\_sLevel.getText());

editSensor.setVisible(true);

}

});

btnEdit.setBounds(230, 82, 100, 20);

btnEdit.setBorderPainted(false);

panel.add(btnEdit);

}

public String getId() {

return id;

}

public void setId(String id) {

this.id = id;

}

public String getSensorId() {

return sensorId;

}

public void setSensorId(String sensorId) {

this.sensorId = sensorId;

}

public int getFloorNo() {

return floorNo;

}

public void setFloorNo(int floorNo) {

this.floorNo = floorNo;

}

public int getRoomNo() {

return roomNo;

}

public void setRoomNo(int roomNo) {

this.roomNo = roomNo;

}

public int getC\_level() {

return c\_level;

}

public void setC\_level(int c\_level) {

this.c\_level = c\_level;

}

public int getS\_level() {

return s\_level;

}

public void setS\_level(int s\_level) {

this.s\_level = s\_level;

}

public boolean isStatus() {

return status;

}

public void setStatus(boolean status) {

this.status = status;

}

}

* EditSensor.java

package desktopClient;

import java.awt.BorderLayout;

import java.awt.FlowLayout;

import java.awt.Font;

import java.awt.event.ActionEvent;

import java.awt.event.ActionListener;

import java.net.MalformedURLException;

import java.rmi.Naming;

import java.rmi.NotBoundException;

import java.rmi.RemoteException;

import javax.swing.JButton;

import javax.swing.JDialog;

import javax.swing.JFrame;

import javax.swing.JLabel;

import javax.swing.JPanel;

import javax.swing.JTextField;

import javax.swing.border.EmptyBorder;

import desktopClient.RMI\_Test;

import desktopClient.EditSensor;

import sendhttp.RMI\_Service;

public class EditSensor extends JDialog {

private final JPanel contentPanel = new JPanel();

private boolean res;

private JTextField txtSensorName;

private JTextField txtFloorNo;

private JTextField txtRoomNo;

private JTextField txtCOLevel;

private JTextField txtSmokeLevel;

public static void main(String[] args) {

try {

EditSensor dialog = new EditSensor(null, null, null, null, null, null);

dialog.setDefaultCloseOperation(JDialog.DISPOSE\_ON\_CLOSE);

dialog.setVisible(true);

} catch (Exception e) {

e.printStackTrace();

}

}

public EditSensor(String id, String sensorID, String floorNo, String roomNo, String c\_level, String s\_level) {

setBounds(250, 300, 360, 350);

getContentPane().setLayout(new BorderLayout());

contentPanel.setBorder(new EmptyBorder(5, 5, 5, 5));

getContentPane().add(contentPanel, BorderLayout.CENTER);

contentPanel.setLayout(null);

{

JLabel lblSensorName = new JLabel("Sensor Name: ");

lblSensorName.setFont(new Font("Tahoma", Font.PLAIN, 15));

lblSensorName.setBounds(25, 25, 100, 30);

contentPanel.add(lblSensorName);

}

{

JLabel lblFloorNo = new JLabel("Floor No:");

lblFloorNo.setFont(new Font("Tahoma", Font.PLAIN, 15));

lblFloorNo.setBounds(25, 75, 100, 30);

contentPanel.add(lblFloorNo);

}

{

JLabel lblRoomNo = new JLabel("Room No:");

lblRoomNo.setFont(new Font("Tahoma", Font.PLAIN, 15));

lblRoomNo.setBounds(25, 125, 100, 30);

contentPanel.add(lblRoomNo);

}

{

JLabel lblCOLevel = new JLabel("CO2 Level: ");

lblCOLevel.setFont(new Font("Tahoma", Font.PLAIN, 15));

lblCOLevel.setBounds(25, 175, 100, 30);

contentPanel.add(lblCOLevel);

}

{

JLabel lblSmokeLevel = new JLabel("Smoke Level: ");

lblSmokeLevel.setFont(new Font("Tahoma", Font.PLAIN, 15));

lblSmokeLevel.setBounds(25, 225, 100, 30);

contentPanel.add(lblSmokeLevel);

}

{

txtSensorName = new JTextField();

txtSensorName.setBounds(150, 25, 170, 30);

contentPanel.add(txtSensorName);

txtSensorName.setColumns(10);

}

{

txtFloorNo = new JTextField();

txtFloorNo.setBounds(150, 75, 170, 30);

contentPanel.add(txtFloorNo);

txtFloorNo.setColumns(10);

}

{

txtRoomNo = new JTextField();

txtRoomNo.setBounds(150, 125, 170, 30);

contentPanel.add(txtRoomNo);

txtRoomNo.setColumns(10);

}

{

txtCOLevel = new JTextField();

txtCOLevel.setBounds(150, 175, 170, 30);

contentPanel.add(txtCOLevel);

txtCOLevel.setColumns(10);

}

{

txtSmokeLevel = new JTextField();

txtSmokeLevel.setBounds(150, 225, 170, 30);

contentPanel.add(txtSmokeLevel);

txtSmokeLevel.setColumns(10);

}

{

JPanel buttonPane = new JPanel();

buttonPane.setLayout(new FlowLayout(FlowLayout.RIGHT));

getContentPane().add(buttonPane, BorderLayout.SOUTH);

{

JButton saveButton = new JButton("Save");

txtSensorName.setText(sensorID);

txtFloorNo.setText(floorNo);

txtRoomNo.setText(roomNo);

txtCOLevel.setText(c\_level);

txtSmokeLevel.setText(s\_level);

saveButton.addActionListener(new ActionListener() {

public void actionPerformed(ActionEvent e) {

RMI\_Service service;

try {

service = (RMI\_Service) Naming.lookup("rmi://localhost:5099/FireSensorService");

try {

res = service.editSensor(id, txtSensorName.getText(), Integer.parseInt(txtFloorNo.getText()),Integer.parseInt(txtRoomNo.getText()), Integer.parseInt(txtCOLevel.getText()), Integer.parseInt(txtSmokeLevel.getText()));

} catch (RemoteException e1) {

e1.printStackTrace();

}

} catch (MalformedURLException | RemoteException | NotBoundException ex) {

ex.printStackTrace();

}

dispose();

RMI\_Test rmiTest = new RMI\_Test(true);

rmiTest.main(null);

}

});

saveButton.setActionCommand("Save");

buttonPane.add(saveButton);

getRootPane().setDefaultButton(saveButton);

}

{

JButton deleteButton = new JButton("Delete");

deleteButton.addActionListener(new ActionListener() {

public void actionPerformed(ActionEvent e) {

RMI\_Service service;

try {

service = (RMI\_Service) Naming.lookup("rmi://localhost:5099/FireSensorService");

try {

res = service.deleteSensor(id);

} catch (RemoteException e1) {

e1.printStackTrace();

}

} catch (MalformedURLException | RemoteException | NotBoundException ex) {

ex.printStackTrace();

}

dispose();

RMI\_Test rmiTest = new RMI\_Test(true);

rmiTest.main(null);

}

});

deleteButton.setActionCommand("Delete");

buttonPane.add(deleteButton);

getRootPane().setDefaultButton(deleteButton);

}

{

JButton cancelButton = new JButton("Cancel");

cancelButton.addActionListener(new ActionListener() {

public void actionPerformed(ActionEvent e) {

dispose();

RMI\_Test rmiTest = new RMI\_Test(true);

rmiTest.main(null);

}

});

buttonPane.add(cancelButton);

}

}

}

}

* RMI\_Test.java

package desktopClient;

import java.awt.EventQueue;

import java.awt.GridLayout;

import java.awt.event.ActionEvent;

import java.awt.event.ActionListener;

import java.net.MalformedURLException;

import java.rmi.Naming;

import java.rmi.NotBoundException;

import java.rmi.RemoteException;

import javax.swing.JFrame;

import javax.swing.JMenu;

import javax.swing.JMenuBar;

import javax.swing.JMenuItem;

import javax.swing.JPanel;

import javax.swing.JScrollPane;

import javax.swing.JSeparator;

import javax.swing.Timer;

import javax.swing.border.EmptyBorder;

import org.json.JSONArray;

import org.json.JSONObject;

import desktopClient.ViewSensor;

import sendhttp.RMI\_Service;

public class RMI\_Test extends JFrame {

private static JPanel contentPane;

private static String responseBody;

private static RMI\_Test contentFrame;

private static boolean isLogged = false;

public static void main(String[] args) {

EventQueue.invokeLater(new Runnable() {

public void run() {

try {

contentFrame = new RMI\_Test(isLogged);

contentFrame.setLocationRelativeTo(null);

contentFrame.setVisible(true);

} catch (Exception e) {

e.printStackTrace();

}

Timer t = new Timer(0, null);

t.addActionListener(new ActionListener() {

@Override

public void actionPerformed(ActionEvent e) {

try {

RMI\_Service service = null;

try {

service = (RMI\_Service) Naming.lookup("rmi://localhost:5099/FireSensorService");

try {

responseBody = service.getSensorDetails();

} catch (RemoteException e1) {

e1.printStackTrace();

}

populateSensorComponents(responseBody);

} catch (NotBoundException ex) {

System.err.println(ex.getMessage());

} catch (MalformedURLException ex) {

System.err.println(ex.getMessage());

} catch (RemoteException ex) {

System.err.println(ex.getMessage());

}

System.out.println("15 seconds!");

} catch (Exception e1) {

e1.printStackTrace();

}

}

});

t.setRepeats(true);

t.setDelay(15000);

t.start();

}});

}

public RMI\_Test(boolean isLogged) {

this.isLogged = isLogged;

setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

setBounds(0, 0, 390, 800);

JMenuBar menuBar = new JMenuBar();

setJMenuBar(menuBar);

JMenu menuList = new JMenu("Options");

menuBar.add(menuList);

JMenuItem menuItem1 = new JMenuItem("Add Sensor");

menuItem1.addActionListener(new ActionListener() {

public void actionPerformed(ActionEvent e) {

contentFrame.dispose();

AddSensor addSensorForm = new AddSensor();

addSensorForm.setVisible(true);

}

});

menuList.add(menuItem1);

menuItem1.setVisible(isLogged);

JMenuItem menuItem2 = new JMenuItem("Login");

menuItem2.addActionListener(new ActionListener() {

public void actionPerformed(ActionEvent e) {

contentFrame.dispose();

Login login = new Login();

login.setVisible(true);

}

});

menuList.add(menuItem2);

menuItem2.setVisible(!isLogged);

JSeparator separator = new JSeparator();

menuList.add(separator);

JMenuItem menuItem3 = new JMenuItem("Exit");

menuItem3.addActionListener(new ActionListener() {

public void actionPerformed(ActionEvent e) {

System.exit(EXIT\_ON\_CLOSE);

}

});

menuList.add(menuItem3);

//Grid Layout

contentPane = new JPanel();

contentPane.setBorder(new EmptyBorder(5, 5, 5, 5));

GridLayout gridPane = new GridLayout();

gridPane.setColumns(1);

gridPane.setRows(0);

contentPane.setLayout(gridPane);

//Scroll Bar

JScrollPane scrollPane = new JScrollPane(contentPane, JScrollPane.VERTICAL\_SCROLLBAR\_AS\_NEEDED,

JScrollPane.HORIZONTAL\_SCROLLBAR\_NEVER);

setContentPane(scrollPane);

}

public static void populateSensorComponents(String responseBody) {

contentPane.removeAll();

JSONArray sensors = new JSONArray(responseBody);

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

JSONObject sensor = sensors.getJSONObject(i);

String id = sensor.getString("\_id");

String sensorId = sensor.getString("sensorID");

int floorNo = sensor.getInt("floorNo");

int roomNo = sensor.getInt("roomNo");

int c\_level = sensor.getInt("c\_level");

int s\_level = sensor.getInt("s\_level");

boolean status = sensor.getBoolean("status");

ViewSensor viewSensor = new ViewSensor(id, sensorId, floorNo, roomNo, c\_level, s\_level, status, isLogged, contentFrame);

viewSensor.setVisible(true);

contentPane.add(viewSensor);

}

contentPane.validate();

contentPane.repaint();

}

}

1. Dummy Sensor

import java.awt.event.ActionEvent;

import java.awt.event.ActionListener;

import java.io.IOException;

import java.net.MalformedURLException;

import java.rmi.Naming;

import java.rmi.NotBoundException;

import java.rmi.RemoteException;

import java.util.Scanner;

import javax.swing.Timer;

import org.apache.http.client.methods.HttpPut;

import org.apache.http.entity.StringEntity;

import org.apache.http.impl.client.CloseableHttpClient;

import org.apache.http.impl.client.HttpClientBuilder;

import org.json.JSONObject;

public class fireSensor {

public static void main(String[] args) throws RemoteException {

//EDIT this to match the Sensor ID

String id = "5eaf01ebae30d335b465ee66";

//int c\_level;

//int s\_level;

//Scanner input = new Scanner(System.in);

/\*System.out.print("CO2 Level: ");

c\_level = input.nextInt();

System.out.print("Smoke Level: ");

s\_level = input.nextInt();\*/

try {

while (true) {

int c\_level = (int) ((Math.random()\*((9-1)+1))+1);

int s\_level = (int) ((Math.random()\*((9-1)+1))+1);

editSensor(id, c\_level, s\_level);

System.out.println("WORKING");

Thread.sleep(10 \* 1000);

}

} catch (InterruptedException e) {

e.printStackTrace();

}

}

public static boolean editSensor(String id, double c\_level, double s\_level) throws RemoteException {

boolean res = false;

JSONObject json = new JSONObject();

json.put("c\_level", c\_level);

json.put("s\_level", s\_level);

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

try {

HttpPut request = new HttpPut("http://localhost:4000/api/sensors/" + id);

StringEntity params = new StringEntity(json.toString());

//Headers and Authentication Token

request.addHeader("content-type", "application/json");

request.addHeader("Authorization","Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJlbWFpbCI6ImhlbGxvQGdtYWlsLmNvbSIsImlhdCI6MTU4ODQ1NDE2N30.yG30vjE6jkIXr\_WMQ3PIPfloT7xtya-pJVLqkfZza2s");

request.setEntity(params);

org.apache.http.HttpResponse response = httpClient.execute(request);

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

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;

}

}