

**Sri Lanka Institute of Information Technology**

**Distributed System**

**(SE3020)**

**Assignment 02 – REST API**

**Fire Alarm Monitoring System Assignment Report**

**Group Details:**

|  |  |
| --- | --- |
| IT17184304 | **Jayagoda N.M.** |
| IT18044928 | **J.L. Thilini Randika** |
| IT18016444 | **I.G. Hansani Yasodya** |

**Table of Contents**

**1.Introduction** ........................................................................................ **3**

**2.High Level Architectural Diagram** ....................................................... **4**

**3.System Workflow Diagram** ................................................................. **5**

**4.System Workflow Scenario Execution** ................................................ **6**

**5.Appendix (Source Codes & Binaries)**................................................. **12**

**5.1 Web Client** ............................................................................. **13 5.2 REST API** ................................................................................. **18 5.3 RMI Server/Client** .................................................................. **34**

# 1.Introduction

This project ‘Fire Alarm monitoring system’  is a real-time monitoring system that detects the smoke and co2 level in the air due to fire. This has been developed on NodeJS ,Java and MySQL.

Users can view the status of all fire alarm sensors by using this system. System displays whether each fire alarm sensor is active. In addition to that system displays the floor number, room number, smoke level and the co2 level. Smoke level and co2 level is represented in 1 to 10 scale. If the smoke level or co2 level is above 5, they are marked in red.The sensor details are updated every 40 seconds.

In this system admin can monetarize the fire alarm system of a building. First admin must log into the system by giving correct credentials. When the smoke level or sensor level goes more than 5 then the administrator will be notified that fire alarm sensor has been activated. An email is sent to the administration to alert that occasion. In this system admin can add or register new fire alarm sensors and edit sensors or details. When registering the floor number, the room no and floor number should be given.

Our goal is to design a user friendly and ergonomically efficient fire alarm system suitable for use within the specified building. The design team will have to take into account the users of the system and the new technologies available.

# 2.High Level Architectural Diagram

A screenshot of a cell phone

Description automatically generated

# 3.System Workflow Diagram

# A close up of a map Description automatically generated

# 4.System Workflow Scenario Execution

4.1 This dashboard can be viewed by both users and administrators. In this if the co2 level or smoke level is greater than 5,it is mentioned as ‘critical’ in the alert column. When we click ‘Manage Sensors’ button we can go the admin login page.

A screenshot of a cell phone

Description automatically generated

4. In the dashboard if the alert column is ‘critical’ an email alert is sent to the admin.

A screenshot of a cell phone

Description automatically generated

4.3 This is the administration login. An administrator can log into system by giving the correct credentials. When the login is successful a popup message will appear in the interface and administrator can go to sensor registration interface.

A screenshot of a cell phone

Description automatically generated

4.4 This is the sensor registration page. In this administrator can register new sensors by giving location details. By clicking the drop-down list, administrator can see all the registered sensors. He can give the sensor ID and click ‘Fetch Sensor Data’ button, we can see the details of that specific sensor.

A screenshot of a social media post

Description automatically generated

4.5 Then the administrator can edit that sensor. After the necessary edition he can click the ‘Update’ button. Then a popup message is appeared in the system to show that the modification is successful.

A screenshot of a social media post

Description automatically generated

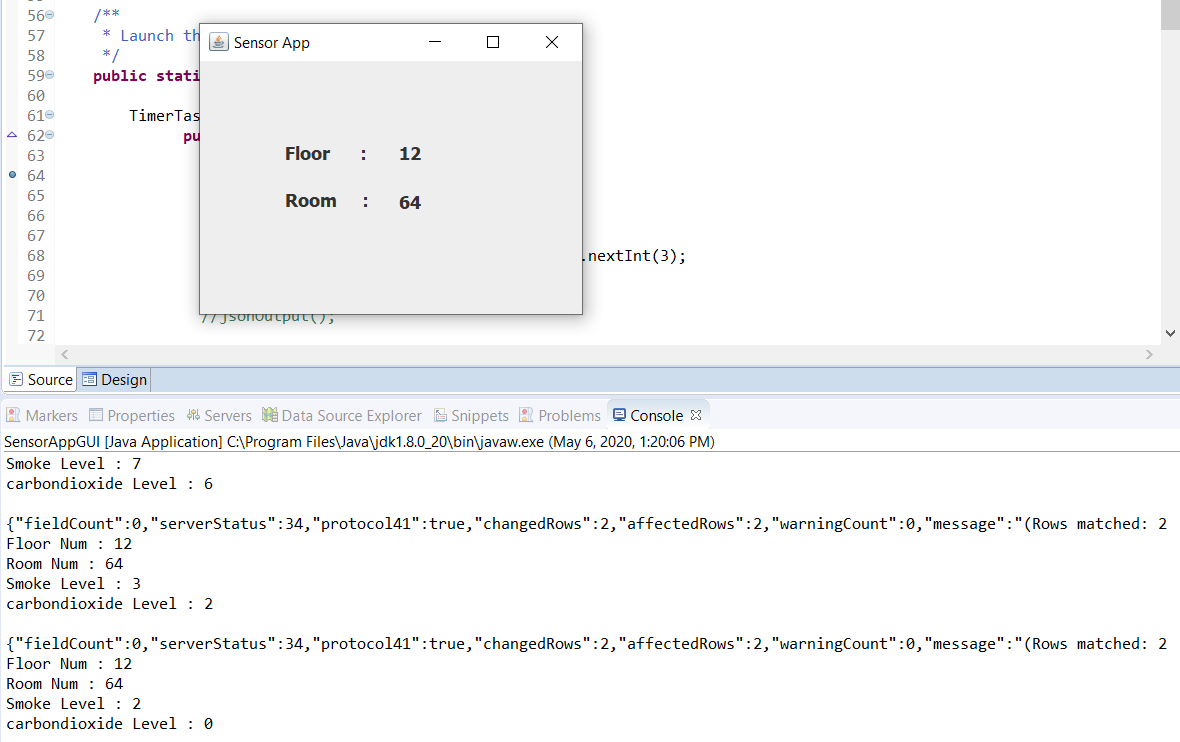
4.6 If the administrator wants to delete a specific sensor, he can click the ‘Delete’ button. Then that sensor will delete from the system. Then a popup message is appeared in the interface to show that sensor is deleted successfully.

A screenshot of a social media post

Description automatically generated

# 5. Sensor App execution process scenario.

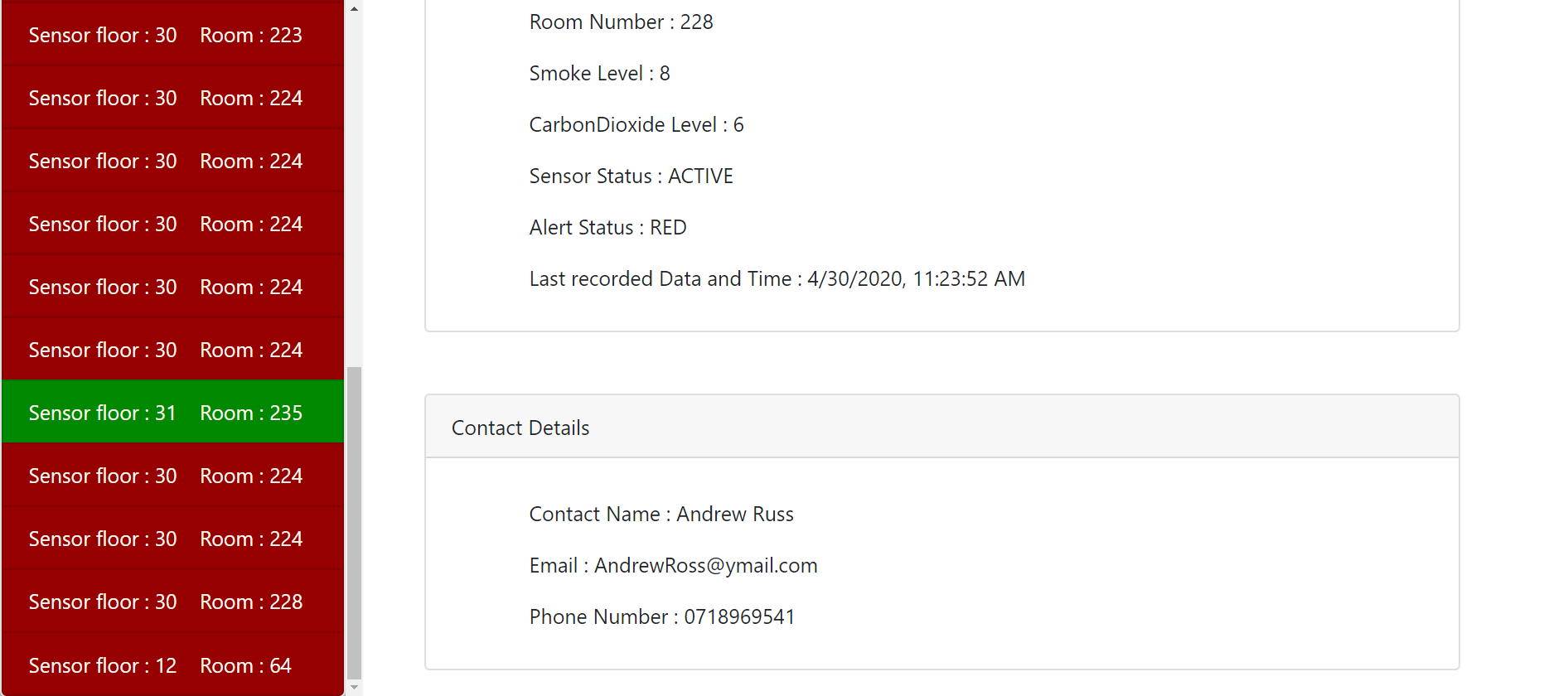
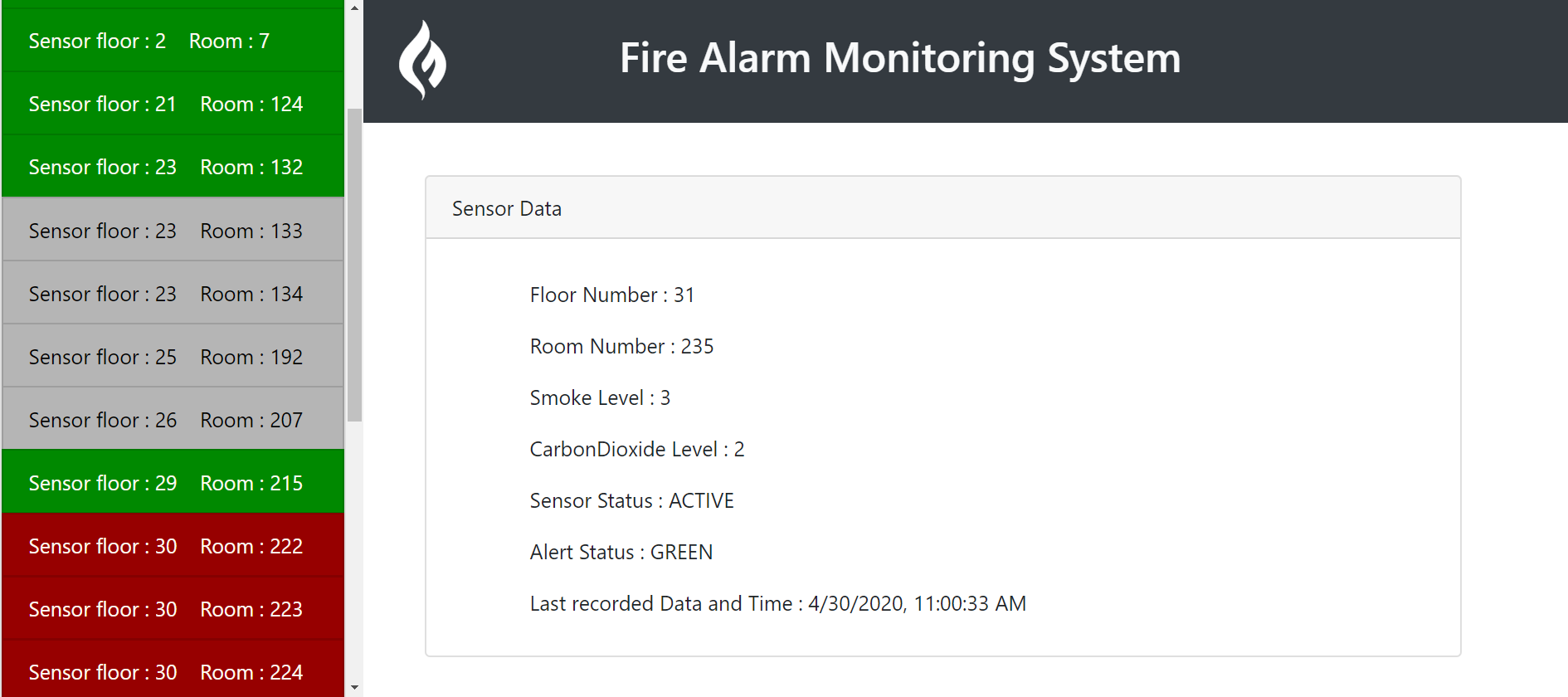
Sensor app is built in Java with Swing UI and it generates random values for carbondioxde levels and Smoke level. There is a relationship build between these two factors. Then the generated value is directly sent to the Database through the REST API every 10 seconds. I have used a library to periodically send the data from the sensor to the REST API every 10 seconds. At the Database level, the previously existed values get updated.



# 

# 6. Web Client

At Web client, you could view all the sensors available in the building and there you have a clear view of sensors which indicate a fire warning, sensors which are currently offline and the sensors under good condition ( smoke level and carbondioxide levels below level 5 ). Then the Web Client sends the request to get the fresh updated values from the database through the REST API every 15 seconds and retrieve them, and the retrieved values are used to update the components in the Document Object model (DOM).

# 7. REST API

REST API is developed using NodeJS and Express and it handles all the requests from the Web Client, Sensor App, and the RMI Server. Mainly there is get all the sensor records, get sensor by floor number and the room number, Add a new sensor to the database, update the state of the sensor including smoke level and the carbondioxide level and deleting a sensor operations.

**5.Appendix**

**Source Codes & Binaries**

## 5.3 RMI Server/Client

#### com.ds.rmi.client

##### Home.java

public class Home extends javax.swing.JFrame {

boolean emailOnceSend = false;

List<Integer>sendEmailData = new ArrayList<>();

public Home(){

initComponents();

setLocationRelativeTo(null);

serverRefresh();

SensorIndicator render = new SensorIndicator();

tblSensors.setDefaultRenderer(Object.class, render);

}

public void serverRefresh() {

//To change body of generated methods, choose Tools | Templates.

dsiplaySensorList();

Thread t = new Thread(new Runnable() {

@Override

public void run() {

//To change body of generated methods, choose Tools | Templates.

try{

Thread.sleep(5000);

serverRefresh();

}catch(Exception e){

e.printStackTrace();

}

}

});

t.start();

}

public void dsiplaySensorList(){

try {

Registry reg = LocateRegistry.getRegistry("localhost",1212);

FireAlarmInterface fai = (FireAlarmInterface)reg.lookup("FireAlarmServer");

String sensorJson = fai.sensorList();

java.lang.reflect.Type listType = new TypeToken<ArrayList<Sensor>>() {}.getType();

List<Sensor> sensorList = new Gson().fromJson(sensorJson, listType);

System.out.println("Sensor List: "+sensorList);

String[] columnNames = { "Sensor ID", "Floor No", "Room No", "Smoke Level", "CO2 Level", "Status","Alert" };

DefaultTableModel model = new DefaultTableModel(columnNames, 0);

int smk\_lvl, co2\_lvl, emailid;

for( Sensor sensor : sensorList)

{

Vector<String> row = new Vector<String>();

row.add(sensor.getSensor\_id());

row.add(sensor.getFloor\_no());

row.add(sensor.getRoom\_no());

row.add(sensor.getSmoke\_level());

row.add(sensor.getCarbondioxide\_level());

row.add(sensor.getSensor\_status());

smk\_lvl = Integer.parseInt(sensor.getSmoke\_level());

co2\_lvl = Integer.parseInt(sensor.getCarbondioxide\_level());

int id = Integer.parseInt(sensor.getSensor\_id());

if(smk\_lvl > 5 || co2\_lvl > 5){

row.add("CRITICAL");

if(!sendEmailData.contains(id)){

sendEmailData.add(id);

// JOptionPane.showMessageDialog(null, sensor.getEmail());

fai.sendMail(sensor.getEmail(),sensor.getFloor\_no(),sensor.getRoom\_no());

}

}else{

row.add("NORMAL");

if(sendEmailData.contains(id)){

sendEmailData.remove(sendEmailData.indexOf(id));

}

}

model.addRow( row );

}

JTable table = new JTable( model );

tblSensors.setModel(model);

} catch (RemoteException ex) {

Logger.getLogger(Login.class.getName()).log(Level.SEVERE, null, ex);

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

} catch (NotBoundException ex) {

Logger.getLogger(Login.class.getName()).log(Level.SEVERE, null, ex);

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

} catch (MessagingException ex) {

Logger.getLogger(Home.class.getName()).log(Level.SEVERE, null, ex);

}

}

private void btnAdminActionPerformed(java.awt.event.ActionEvent evt) {

Login obj = new Login();

obj.setVisible(true);

}

}

##### SensorIndicator.java

package com.ds.rmi.client

import java.awt.Color;

import java.awt.Component;

import javax.swing.JTable;

import javax.swing.table.DefaultTableCellRenderer;

import javax.swing.table.TableCellRenderer;

public class SensorIndicator implements TableCellRenderer{

private static final TableCellRenderer render = new DefaultTableCellRenderer();

@Override

public Component getTableCellRendererComponent(JTable table, Object value, boolean isSelected, boolean hasFocus, int row, int column) {

Component c = render.getTableCellRendererComponent(table, value, isSelected, hasFocus, row, column);

if(column == 6){

Object result = table.getModel().getValueAt(row, column);

String status = result.toString();

if(status.equals("NORMAL")){

c.setBackground(Color.GREEN);

c.setForeground(Color.black);

}else if(status.equals("CRITICAL")){

c.setBackground(Color.red);

c.setForeground(Color.yellow);

}

}else{

c.setBackground(Color.white);

c.setForeground(Color.black);

}

return c;

}

}

##### Login.java

public class Login extends javax.swing.JFrame {

public Login() {

initComponents();

setLocationRelativeTo(null);

}

private void btnLoginActionPerformed(java.awt.event.ActionEvent evt) {

try {

Registry reg = LocateRegistry.getRegistry("localhost",1212);

FireAlarmInterface fai = (FireAlarmInterface)reg.lookup("FireAlarmServer");

boolean result = fai.login(txtUsername.getText(), txtPassword.getText());

// System.out.println("pw: "+txtPassword.getText());

if(result){

JOptionPane.showMessageDialog(null, "Login Successfully");

console obj = new console();

obj.setVisible(true);

this.dispose();

}else{

JOptionPane.showMessageDialog(null, "Login Failed. Error: credentials mismatch");

}

} catch (RemoteException ex) {

Logger.getLogger(Login.class.getName()).log(Level.SEVERE, null, ex);

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

} catch (NotBoundException ex) {

Logger.getLogger(Login.class.getName()).log(Level.SEVERE, null, ex);

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

}

}

##### console.java

public class console extends javax.swing.JFrame {

public console() {

initComponents();

setLocationRelativeTo(null);

displaySensorIDs();

btnUpdate.setVisible(false);

btnDelete.setVisible(false);

}

public void displaySensorIDs(){

try {

Registry reg = LocateRegistry.getRegistry("localhost",1212);

FireAlarmInterface fai = (FireAlarmInterface)reg.lookup("FireAlarmServer");

String jsonSensorList = fai.sensorList();

if(jsonSensorList == null){

comboSensors.addItem("No Sensors Yet");

}else{

java.lang.reflect.Type listType = new TypeToken<ArrayList<Sensor>>() {}.getType();

List<Sensor> sensorList = new Gson().fromJson(jsonSensorList, listType);

comboSensors.removeAllItems();

for( Sensor s : sensorList)

{

comboSensors.addItem("Sensor ID :"+ s.getSensor\_id());

}

}

} catch (RemoteException ex) {

Logger.getLogger(console.class.getName()).log(Level.SEVERE, null, ex);

} catch (NotBoundException ex) {

Logger.getLogger(console.class.getName()).log(Level.SEVERE, null, ex);

}

}

public void getSensorData(){

try {

Registry reg = LocateRegistry.getRegistry("localhost",1212);

FireAlarmInterface fai = (FireAlarmInterface)reg.lookup("FireAlarmServer");

String id = comboSensors.getSelectedItem().toString();

id = id.substring(11);

System.out.println("id: "+id);

String sensorData = fai.getSensorData(id);

java.lang.reflect.Type listType = new TypeToken<ArrayList<Sensor>>() {}.getType();

List<Sensor> sensorList = new Gson().fromJson(sensorData, listType);

for(Sensor s : sensorList){

txtName.setText(s.getOwner\_name());

txtEmail.setText(s.getEmail());

txtPhone.setText(s.getPhone\_number());

txtFloor.setText(s.getFloor\_no());

txtRoom.setText(s.getRoom\_no());

}

btnRegister.setText("Update");

System.out.println(sensorList);

} catch (RemoteException ex) {

Logger.getLogger(console.class.getName()).log(Level.SEVERE, null, ex);

} catch (NotBoundException ex) {

Logger.getLogger(console.class.getName()).log(Level.SEVERE, null, ex);

}

}

public void register(){

String validatteMsg = "";

int floor\_no = 0, room\_no = 0;

String name=txtName.getText();

String email=txtEmail.getText();

String phone=txtPhone.getText();

if(name.isEmpty()){

validatteMsg="Name Field s Mandatory!";

}else if(email.isEmpty()){

validatteMsg="Email Field s Mandatory!";

}else if(phone.isEmpty()){

validatteMsg="Telephone Field s Mandatory!";

}else if(txtFloor.getText().isEmpty()){

validatteMsg="Floor No Field s Mandatory!";

}else if(txtRoom.getText().isEmpty()){

validatteMsg="Room No Field s Mandatory!";

}else{

try{

floor\_no = Integer.parseInt(txtFloor.getText());

}catch(Exception e){

JOptionPane.showMessageDialog(null, "FLOOR NO is Invalid!.","Sensor Registration",JOptionPane.WARNING\_MESSAGE);

}

try{

room\_no = Integer.parseInt(txtRoom.getText());

}catch(Exception e){

JOptionPane.showMessageDialog(null, "ROOM NO is Invalid!.","Sensor Registration",JOptionPane.WARNING\_MESSAGE);

}

System.out.println(name);

System.out.println(email);

System.out.println(phone);

System.out.println(floor\_no);

System.out.println(room\_no);

try{

Registry reg = LocateRegistry.getRegistry("localhost",1212);

FireAlarmInterface fai = (FireAlarmInterface)reg.lookup("FireAlarmServer");

String regResult = fai.registerSensor(name, email, phone, floor\_no, room\_no);

JOptionPane.showMessageDialog(null, regResult,"Sensor Registration",JOptionPane.INFORMATION\_MESSAGE);

}catch (RemoteException ex) {

Logger.getLogger(Login.class.getName()).log(Level.SEVERE, null, ex);

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

} catch (NotBoundException ex) {

Logger.getLogger(Login.class.getName()).log(Level.SEVERE, null, ex);

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

}

}

}

public void update(){

String sid = comboSensors.getSelectedItem().toString();

int id = Integer.parseInt(sid.substring(11));

System.out.println("id: "+id);

int floor\_no = 0, room\_no = 0;

String name=txtName.getText();

String email=txtEmail.getText();

String phone=txtPhone.getText();

try{

floor\_no = Integer.parseInt(txtFloor.getText());

}catch(Exception e){

JOptionPane.showMessageDialog(null, "FLOOR NO is Invalid!.","Sensor Registration",JOptionPane.WARNING\_MESSAGE);

}

try{

room\_no = Integer.parseInt(txtRoom.getText());

}catch(Exception e){

JOptionPane.showMessageDialog(null, "ROOM NO is Invalid!.","Sensor Registration",JOptionPane.WARNING\_MESSAGE);

}

try{

Registry reg = LocateRegistry.getRegistry("localhost",1212);

FireAlarmInterface fai = (FireAlarmInterface)reg.lookup("FireAlarmServer");

String regResult = fai.updateSensor(id,name, email, phone, floor\_no, room\_no);

JOptionPane.showMessageDialog(null, regResult,"Sensor Modification",JOptionPane.INFORMATION\_MESSAGE);

}catch (RemoteException ex) {

Logger.getLogger(Login.class.getName()).log(Level.SEVERE, null, ex);

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

} catch (NotBoundException ex) {

Logger.getLogger(Login.class.getName()).log(Level.SEVERE, null, ex);

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

}

}

public void delete(){

String id = comboSensors.getSelectedItem().toString();

id = id.substring(11);

System.out.println("id: "+id);

try{

Registry reg = LocateRegistry.getRegistry("localhost",1212);

FireAlarmInterface fai = (FireAlarmInterface)reg.lookup("FireAlarmServer");

String regResult = fai.deleteSensor(id);

JOptionPane.showMessageDialog(null, regResult,"Sensor Deletion",JOptionPane.INFORMATION\_MESSAGE);

}catch (RemoteException ex) {

Logger.getLogger(Login.class.getName()).log(Level.SEVERE, null, ex);

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

} catch (NotBoundException ex) {

Logger.getLogger(Login.class.getName()).log(Level.SEVERE, null, ex);

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

}

}

private void btnSearchActionPerformed(java.awt.event.ActionEvent evt) {

getSensorData();

btnDelete.setVisible(true);

btnUpdate.setVisible(true);

btnRegister.setVisible(false);

}

private void btnRegisterActionPerformed(java.awt.event.ActionEvent evt) {

register();

}

private void comboSensorsFocusGained(java.awt.event.FocusEvent evt) {

displaySensorIDs();

}

private void btnClearActionPerformed(java.awt.event.ActionEvent evt) {

txtName.setText(null);

txtEmail.setText(null);

txtPhone.setText(null);

txtFloor.setText(null);

txtRoom.setText(null);

btnRegister.setText("Register");

btnDelete.setVisible(false);

btnUpdate.setVisible(false);

btnRegister.setVisible(true);

}

private void btnDeleteActionPerformed(java.awt.event.ActionEvent evt) {

delete();

}

private void btnUpdateActionPerformed(java.awt.event.ActionEvent evt) {

update();

}

private void btnHomeActionPerformed(java.awt.event.ActionEvent evt) {

Home obj = new Home();

obj.setVisible(true);

this.dispose();

}

}

#### com.ds.rmi.Server

##### User.java

package com.ds.rmi.server;

public class User {

private String name;

private String email;

private String phone;

private String floor;

private String room;

public User(String name, String email, String phone, String floor, String room) {

this.name = name;

this.email = email;

this.phone = phone;

this.floor = floor;

this.room = room;

}

public String getName() {

return name;

}

public void setName(String name) {

this.name = name;

}

public String getEmail() {

return email;

}

public void setEmail(String email) {

this.email = email;

}

public String getPhone() {

return phone;

}

public void setPhone(String phone) {

this.phone = phone;

}

public String getFloor() {

return floor;

}

public void setFloor(String floor) {

this.floor = floor;

}

public String getRoom() {

return room;

}

public void setRoom(String room) {

this.room = room;

}

}

##### Sensor.java

package com.ds.rmi.server;

public class Sensor {

private String sensor\_id;

private String floor\_no;

private String room\_no;

private String smoke\_level;

private String carbondioxide\_level;

private String sensor\_status;

private String alert\_status;

private String created\_date;

private String owner\_name;

private String email;

private String phone\_number;

public Sensor(){

}

public Sensor(String sensor\_id, String floor\_no, String room\_no, String smoke\_level, String carbondioxide\_level, String sensor\_status, String alert\_status, String created\_date, String owner\_name, String email, String phone\_number) {

this.sensor\_id = sensor\_id;

this.floor\_no = floor\_no;

this.room\_no = room\_no;

this.smoke\_level = smoke\_level;

this.carbondioxide\_level = carbondioxide\_level;

this.sensor\_status = sensor\_status;

this.alert\_status = alert\_status;

this.created\_date = created\_date;

this.owner\_name = owner\_name;

this.email = email;

this.phone\_number = phone\_number;

}

public String getSensor\_id() {

return sensor\_id;

}

public void setSensor\_id(String sensor\_id) {

this.sensor\_id = sensor\_id;

}

public String getFloor\_no() {

return floor\_no;

}

public void setFloor\_no(String floor\_no) {

this.floor\_no = floor\_no;

}

public String getRoom\_no() {

return room\_no;

}

public void setRoom\_no(String room\_no) {

this.room\_no = room\_no;

}

public String getSmoke\_level() {

return smoke\_level;

}

public void setSmoke\_level(String smoke\_level) {

this.smoke\_level = smoke\_level;

}

public String getCarbondioxide\_level() {

return carbondioxide\_level;

}

public void setCarbondioxide\_level(String carbondioxide\_level) {

this.carbondioxide\_level = carbondioxide\_level;

}

public String getSensor\_status() {

return sensor\_status;

}

public void setSensor\_status(String sensor\_status) {

this.sensor\_status = sensor\_status;

}

public String getAlert\_status() {

return alert\_status;

}

public void setAlert\_status(String alert\_status) {

this.alert\_status = alert\_status;

}

public String getCreated\_date() {

return created\_date;

}

public void setCreated\_date(String created\_date) {

this.created\_date = created\_date;

}

public String getOwner\_name() {

return owner\_name;

}

public void setOwner\_name(String owner\_name) {

this.owner\_name = owner\_name;

}

public String getEmail() {

return email;

}

public void setEmail(String email) {

this.email = email;

}

public String getPhone\_number() {

return phone\_number;

}

public void setPhone\_number(String phone\_number) {

this.phone\_number = phone\_number;

}

@Override

public String toString() {

return "Sensor{" + "sensor\_id=" + sensor\_id + ", floor\_no=" + floor\_no + ", room\_no=" + room\_no + ", smoke\_level=" + smoke\_level + ", carbondioxide\_level=" + carbondioxide\_level + ", sensor\_status=" + sensor\_status + ", alert\_status=" + alert\_status + ", created\_date=" + created\_date + ", owner\_name=" + owner\_name + ", email=" + email + ", phone\_number=" + phone\_number + '}';

}

}

##### FireAlarmInterface.java

package com.ds.rmi.server;

import java.rmi.Remote;

import java.rmi.RemoteException;

import java.util.ArrayList;

import java.util.List;

import javax.mail.MessagingException;

public interface FireAlarmInterface extends Remote{

public boolean login(String user,String pass) throws RemoteException;

public String registerSensor(String name,String email,String phone, int floor, int room) throws RemoteException;

public String sensorList() throws RemoteException;

public String getSensorData(String id) throws RemoteException;

public String updateSensor(int id, String name,String email,String phone, int floor, int room) throws RemoteException;

public String deleteSensor(String id) throws RemoteException;

public void sendMail(String receiver, String floor, String room) throws RemoteException, MessagingException;

}

##### FireAlarmImpl.java

package com.ds.rmi.server;

import com.ds.rmi.server.FireAlarmInterface;

import com.google.gson.Gson;

import com.google.gson.JsonObject;

import com.google.gson.reflect.TypeToken;

import java.rmi.RemoteException;

import java.rmi.server.UnicastRemoteObject;

import java.util.ArrayList;

import java.io.BufferedReader;

import java.io.IOException;

import java.io.InputStreamReader;

import java.io.OutputStream;

import java.io.OutputStreamWriter;

import java.lang.reflect.Type;

import java.net.HttpURLConnection;

import java.net.MalformedURLException;

import java.net.URL;

import java.rmi.server.UnicastRemoteObject;

import java.util.List;

import java.util.Properties;

import java.util.logging.Level;

import java.util.logging.Logger;

import javax.mail.Authenticator;

import javax.mail.Message;

import javax.mail.MessagingException;

import javax.mail.PasswordAuthentication;

import javax.mail.Session;

import javax.mail.Transport;

import javax.mail.internet.InternetAddress;

import javax.mail.internet.MimeMessage;

public class FireAlarmImpl extends UnicastRemoteObject implements FireAlarmInterface{

public FireAlarmImpl() throws RemoteException{

}

@Override

public boolean login(String user, String pass) throws RemoteException {

//To change body of generated methods, choose Tools | Templates.

return (user.equals("admin") && pass.equals("admin"));

}

@Override

public String registerSensor(String name, String email, String phone, int floor, int room) throws RemoteException {

try {

//To change body of generated methods, choose Tools | Templates.

URL url = new URL ("http://localhost:3000/sensorRecord");

HttpURLConnection con = (HttpURLConnection)url.openConnection();

con.setRequestMethod("POST");

con.setRequestProperty("Content-Type", "application/json");

con.setRequestProperty("Accept", "application/json");

con.setDoOutput(true);

JsonObject newSensor = new JsonObject();

newSensor.addProperty("floor\_no", floor);

newSensor.addProperty("room\_no", room);

newSensor.addProperty("owner\_name", name);

newSensor.addProperty("email", email);

newSensor.addProperty("phone\_number", phone);

OutputStream os = con.getOutputStream();

os.write(newSensor.toString().getBytes());

os.flush();

if (con.getResponseCode() == 200) {

return "New Sensor Registered Successfuly.";

}else if(con.getResponseCode() != 200){

throw new RuntimeException("Failed : HTTP error code : "

+ con.getResponseCode());

}

System.out.println("byte post : "+newSensor);

return con.getResponseMessage();

} catch (MalformedURLException ex) {

Logger.getLogger(FireAlarmImpl.class.getName()).log(Level.SEVERE, null, ex);

return "Sensor Registering Failed. Error Code: "+ex.getMessage();

} catch (IOException ex) {

Logger.getLogger(FireAlarmImpl.class.getName()).log(Level.SEVERE, null, ex);

return "Sensor Registering Failed. Error Code: "+ex.getMessage();

}

}

@Override

public String updateSensor(int id, String name, String email, String phone, int floor, int room) throws RemoteException {

//To change body of generated methods, choose Tools | Templates.

try {

//To change body of generated methods, choose Tools | Templates.

URL url = new URL ("http://localhost:3000/sensorData/"+id);

HttpURLConnection con = (HttpURLConnection)url.openConnection();

con.setRequestMethod("PUT");

con.setRequestProperty("Content-Type", "application/json");

con.setRequestProperty("Accept", "application/json");

con.setDoOutput(true);

JsonObject newSensor = new JsonObject();

// newSensor.addProperty("sensor\_id", id);

newSensor.addProperty("floor\_no", floor);

newSensor.addProperty("room\_no", room);

newSensor.addProperty("owner\_name", name);

newSensor.addProperty("email", email);

newSensor.addProperty("phone\_number", phone);

OutputStream os = con.getOutputStream();

os.write(newSensor.toString().getBytes());

os.flush();

if (con.getResponseCode() == 200) {

return "Sensor Updated Successfuly.";

}else if(con.getResponseCode() != 200){

throw new RuntimeException("Failed : HTTP error code : "

+ con.getResponseCode());

}

System.out.println("byte post : "+newSensor);

return con.getResponseMessage();

} catch (MalformedURLException ex) {

Logger.getLogger(FireAlarmImpl.class.getName()).log(Level.SEVERE, null, ex);

return "Sensor Updating Failed. Error Code: "+ex.getMessage();

} catch (IOException ex) {

Logger.getLogger(FireAlarmImpl.class.getName()).log(Level.SEVERE, null, ex);

return "Sensor Updating Failed. Error Code: "+ex.getMessage();

}

}

@Override

public String deleteSensor(String id) {

//To change body of generated methods, choose Tools | Templates.

try {

//To change body of generated methods, choose Tools | Templates.

URL url = new URL ("http://localhost:3000/sensorData/"+id);

HttpURLConnection con = (HttpURLConnection)url.openConnection();

con.setRequestMethod("DELETE");

if (con.getResponseCode() == 200) {

return "ensor Updated Successfuly.";

}else if(con.getResponseCode() != 200){

throw new RuntimeException("Failed : HTTP error code : "

+ con.getResponseCode());

}

return con.getResponseMessage();

} catch (MalformedURLException ex) {

Logger.getLogger(FireAlarmImpl.class.getName()).log(Level.SEVERE, null, ex);

return "Sensor Deleting Failed. Error Code: "+ex.getMessage();

} catch (IOException ex) {

Logger.getLogger(FireAlarmImpl.class.getName()).log(Level.SEVERE, null, ex);

return "Sensor Deleting Failed. Error Code: "+ex.getMessage();

}

}

@Override

public String sensorList() throws RemoteException {

//To change body of generated methods, choose Tools | Templates.

try {

URL url = new URL("http://localhost:3000/sensorData");

HttpURLConnection conn = (HttpURLConnection) url.openConnection();

conn.setRequestMethod("GET");

conn.setRequestProperty("Accept", "application/json");

if (conn.getResponseCode() != 200) {

throw new RuntimeException("Failed : HTTP error code : "

+ conn.getResponseCode());

}

BufferedReader br = new BufferedReader(new InputStreamReader((conn.getInputStream())));

String output;

String json = "";

System.out.println("Output from Server .... \n");

while ((output = br.readLine()) != null) {

System.out.println(output);

json += output;

}

conn.disconnect();

return json;

} catch (IOException | RuntimeException ex) {

System.out.println("Error:==="+ex);

return null;

}

}

@Override

public String getSensorData(String id) throws RemoteException {

try {

//To change body of generated methods, choose Tools | Templates.

String sUrl = "http://localhost:3000/sensorData/"+id;

URL url = new URL(sUrl);

HttpURLConnection conn = (HttpURLConnection) url.openConnection();

conn.setRequestMethod("GET");

conn.setRequestProperty("Accept", "application/json");

if (conn.getResponseCode() != 200) {

throw new RuntimeException("Failed : HTTP error code : "

+ conn.getResponseCode());

}

BufferedReader br = new BufferedReader(new InputStreamReader((conn.getInputStream())));

String output;

String json = "";

System.out.println("Output from Server .... \n");

while ((output = br.readLine()) != null) {

// System.out.println("line: "+output);

json += output;

}

conn.disconnect();

System.out.println("Sensor Data Json: "+json);

return json;

} catch (MalformedURLException ex) {

Logger.getLogger(FireAlarmImpl.class.getName()).log(Level.SEVERE, null, ex);

return ex.getMessage().toString();

} catch (IOException ex) {

Logger.getLogger(FireAlarmImpl.class.getName()).log(Level.SEVERE, null, ex);

return ex.getMessage().toString();

}

}

@Override

public void sendMail(String receiver, String floor, String room) throws RemoteException, MessagingException {

//To change body of generated methods, choose Tools | Templates.

String msg = "Our system detected the Room "+room+" in Floor "+floor+" has critical issue with firing. Please be carefual and follow the rules while our services fix it. Thank you";

String to = receiver;

String host = "smtp.gmail.com";

String user = "firealarm.monitoring.system@gmail.com";

String pass = "firealarm123";

//Get the session object

Properties properties = new Properties();

properties.put("mail.smtp.host", host);

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

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

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

Session session = Session.getDefaultInstance(properties, new Authenticator() {

protected PasswordAuthentication getPasswordAuthentication(){

return new PasswordAuthentication(user, pass);

}

});

//compose the message

try{

MimeMessage message = new MimeMessage(session);

message.setFrom(new InternetAddress(user));

message.addRecipient(Message.RecipientType.TO,new InternetAddress(to));

message.setSubject("Fire Alarm Notify");

message.setText(msg);

// Send message

Transport.send(message);

System.out.println("message sent successfully....");

}catch (MessagingException mex) {

mex.printStackTrace();

}

}

}

##### FireAlarmServer.java

package com.ds.rmi.server;

import java.rmi.AccessException;

import java.rmi.AlreadyBoundException;

import java.rmi.RemoteException;

import java.rmi.registry.LocateRegistry;

import java.rmi.registry.Registry;

import javax.mail.MessagingException;

public class FireAlarmServer {

public static void main(String[] args){

try{

Registry registry = LocateRegistry.createRegistry(1212);

FireAlarmInterface fai = new FireAlarmImpl();

registry.bind("FireAlarmServer", fai);

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

}catch(RemoteException e){

e.getMessage();

}catch(AlreadyBoundException e){

e.getMessage();

}

}

}

## 5.4 EmailApp

##### Server.js

const express = require('express');

const bodyparser = require('body-parser');

const nodemailer = require('nodemailer');

const app = express();

app.use(bodyparser.urlencoded({extended: false}));

app.use(bodyparser.json());

app.post('/sendEmail', (req,res) => {

const floor = req.body.floor\_no;

const room = req.body.room\_no;

const email = req.body.email;

const sender = 'firealarm.monitoring.system@gmail.com';

let transport = nodemailer.createTransport({

host: 'smtp.gmail.com',

port: 587,

auth: {

user: sender,

pass: 'firealarm123'

}

});

const message = {

from: sender,

to: email,

subject: 'Fire Alarm Notify',

text: 'Our system detected the Room '+room+' in Floor '+floor+' has critical issue with firing. Please be careful and follow the rules while our services fix it. Thank you'

};

transport.sendMail(message, function(err, info) {

if (err) {

console.log(err)

} else {

console.log(info);

res.json(info.response);

}

});

});

app.listen(2500, () => console.log('Mail Server is Started..'));

**package.json**

{

"name": "MailApp",

"version": "1.0.0",

"description": "",

"main": "index.js",

"scripts": {

"test": "echo \"Error: no test specified\" && exit 1"

},

"keywords": [],

"author": "",

"license": "ISC",

"dependencies": {

"body-parser": "^1.19.0",

"express": "^4.17.1",

"express-handlebars": "^4.0.4",

"nodemailer": "^6.4.6",

"nodemon": "^2.0.3",

"path": "^0.12.7"

}

}

**5.5 : REST API NodeJS**

## **App.js :**

|  |
| --- |
|  |
|  | const express = require('express');  const bodyParser = require('body-parser'); |
|  | var mysql = require('mysql'); |
|  | var cors = require('cors'); |
|  | const app = express(); |
|  |  |
|  | app.use(bodyParser.json()); |
|  | app.use(cors()); |
|  |  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  | //Database connection to the MySQL database |
|  | var connection = mysql.createConnection({ |
|  | host : 'localhost', |
|  | database : 'fire\_alarm', |
|  | user : '< --username-- >', |
|  | password : '< --password-- >' |
|  | }); |
|  |  |
|  | connection.connect(function(err){ |
|  | if (err){ |
|  | console.error('Error connecting to the Database : ' + err); |
|  | return; |
|  | } |
|  |  |
|  | console.log("Connected to the MySQL database successfully..! "); |
|  | }); |
|  |  |
|  |  |
|  | //routes |
|  |  |
|  | //Get All the Sensor Data |
|  | app.get('/sensorData', async (req, res) => { |
|  | const queryString = 'SELECT \* FROM sensors'; |
|  | connection.query(queryString, function (err, rows) { |
|  | if(err){ |
|  | console.log("Error : ", err); |
|  | return; |
|  | } |
|  | res.send(rows); |
|  | }); |
|  | }); |
|  |  |
|  | //Add sensor data to the Database |
|  | app.post('/sensorRecord', async(req, res) => { |
|  | const floor\_no = req.body.floor\_no; |
|  | const room\_no = req.body.room\_no; |
|  | const smoke\_level = req.body.smoke\_level; |
|  | const carbondioxide\_level = req.body.carbondioxide\_level; |
|  | const sensor\_status = req.body.sensor\_status; |
|  | const owner\_name = req.body.owner\_name; |
|  | const email = req.body.email; |
|  | const phone\_number = req.body.phone\_number; |
|  |  |
|  | var alert\_status = ''; |
|  |  |
|  | if(sensor\_status !== 'OFFLINE'){ |
|  | //Generating the alert\_status record based on the smoke level and the carbondioxide levels, if the sensorStatus is 'ACTIVE' |
|  | if( smoke\_level > 5 || carbondioxide\_level > 5 ){ |
|  | alert\_status = 'RED'; |
|  | }else{ |
|  | alert\_status = 'GREEN'; |
|  | } |
|  | } |
|  |  |
|  | const queryString = "INSERT INTO fire\_alarm.sensors (floor\_no, room\_no, smoke\_level, carbondioxide\_level, sensor\_status, alert\_status, owner\_name, email, phone\_number) " + |
|  | "VALUES (" + floor\_no + ", " + room\_no + ", " + smoke\_level + ", " + carbondioxide\_level + ", '" + sensor\_status +"', '" + alert\_status + "', '" |
|  | + owner\_name + "', '"+ email +"', '" + phone\_number + "');"; |
|  | connection.query( queryString, function (err, rows) { |
|  | if(err){ |
|  | console.error("Error : " + err); |
|  | res.send(err); |
|  | return; |
|  | } |
|  | res.json(rows); |
|  | }); |
|  | }); |
|  |  |
|  | //Update sensor record of the Database |
|  | app.put('/sensorRecord', async(req, res) => { |
|  | const floor\_no = req.body.floor\_no; |
|  | const room\_no = req.body.room\_no; |
|  | const smoke\_level = req.body.smoke\_level; |
|  | const carbondioxide\_level = req.body.carbondioxide\_level; |
|  | const sensor\_status = req.body.sensor\_status; |
|  |  |
|  | var alertStatus = ''; |
|  |  |
|  | if(sensorStatus !== 'OFFLINE'){ |
|  | //Generating the alert\_status record based on the smoke level and the carbondioxide levels, if the sensorStatus is 'ACTIVE' |
|  | if( smokeLevel > 5 || carbondioxideLevel > 5 ){ |
|  | alertStatus = 'RED'; |
|  | }else{ |
|  | alertStatus = 'GREEN'; |
|  | } |
|  | } |
|  |  |
|  | let queryString = "UPDATE fire\_alarm.sensors " + |
|  | "SET smoke\_level = " + smoke\_level + ", " + |
|  | "carbondioxide\_level = " + carbondioxide\_level + ", " + |
|  | "sensor\_status = '" + sensor\_status + "' , " + |
|  | "alert\_status = '" + alertStatus + "' " + |
|  | "WHERE floor\_no = " + floor\_no + " AND room\_no = " + room\_no + ";"; |
|  |  |
|  | connection.query( queryString, function (err, rows) { |
|  | if(err){ |
|  | console.error("Error : " + err); |
|  | res.send(err); |
|  | return; |
|  | } |
|  | res.json(rows); |
|  | }); |
|  | }); |
|  |  |
|  |  |
|  | //Get records by the sensorId |
|  | app.get('/sensorData/:sensor\_id', async (req, res) => { |
|  | const sensor\_id = req.params.sensor\_id; |
|  | const queryString = 'SELECT \* FROM fire\_alarm.sensors where sensor\_id = ' + sensor\_id; |
|  | connection.query(queryString, function(err, rows){ |
|  | if(err){ |
|  | console.log("Error : ", err); |
|  | return; |
|  | } |
|  | res.send(rows); |
|  | }); |
|  | }); |
|  |  |
|  | //Get Records using floor number and the room number ; last record inserted - the latest update of the alert\_status |
|  | app.get('/sensorData/:floor\_no/:room\_no', async (req, res) => { |
|  | const floor\_no = req.params.floor\_no; |
|  | const room\_no = req.params.room\_no; |
|  |  |
|  | const queryString = 'SELECT room\_no, smoke\_level, carbondioxide\_level, sensor\_status, alert\_status' + ' ' + |
|  | 'FROM fire\_alarm.sensors' + ' ' + |
|  | 'WHERE floor\_no = ' + floor\_no + ' AND room\_no = ' + room\_no + ' ' + |
|  | 'ORDER BY sensor\_id DESC' + ' ' + |
|  | 'LIMIT 1;'; |
|  | connection.query(queryString, function(err, rows) { |
|  | if(err){ |
|  | console.log("Error : ", err); |
|  | res.send(err); |
|  | return; |
|  | } |
|  | res.send(rows); |
|  | }); |
|  | }); |
|  |  |
|  | // Email service |
|  | app.post('/emailService/:floor/:room', async(req, res) => { |
|  | const floor\_no = req.params.floor; |
|  | const room\_no = req.params.room; |
|  |  |
|  | const queryString = "SELECT email FROM fire\_alarm.sensors WHERE floor\_no =" + floor\_no + " AND room\_no = " + room\_no; |
|  |  |
|  | connection.query(queryString, function (err, rows) { |
|  | if(err){ |
|  | console.log("Error : ", err); |
|  | res.send(err); |
|  | return; |
|  | } |
|  | res.json(rows); |
|  | }); |
|  | }); |
|  |  |
|  | //delete sensor data |
|  | app.delete('/sensorData/:sensor\_id',(req,res) =>{ |
|  | const sensor\_id = req.params.sensor\_id; |
|  | const queryString = 'DELETE FROM fire\_alarm.sensors where sensor\_id = ' + sensor\_id; |
|  | connection.query(queryString, function(err, rows){ |
|  | if(err){ |
|  | console.log("Error : ", err); |
|  | return; |
|  | } |
|  | res.send(rows); |
|  | }); |
|  | }); |
|  |  |
|  | app.listen(3000); |

# Package.json :

|  |
| --- |
| { |
|  | "name": "firm-alarm-monitoring-system", |
|  | "version": "1.0.0", |
|  | "description": "DS Assignment 02 - Firm alarm monitoring application backend REST API development using NodeJS and Express with MySQL database", |
|  | "main": "app.js", |
|  | "scripts": { |
|  | }, |
|  | "author": "Jayagoda N.M. - IT17184304", |
|  | "license": "ISC", |
|  | "dependencies": { |
|  | "body-parser": "^1.19.0", |
|  | "cors": "^2.8.5", |
|  | "express": "^4.17.1", |
|  | "mysql": "^2.18.1", |
|  | "nodemon": "^2.0.3" |
|  | } |
|  | } |

# **5.6 Sensor App**

# **sensorApp.java :**

|  |
| --- |
| package swingGUI; |
|  |  |
|  | import java.awt.BorderLayout; |
|  | import java.awt.EventQueue; |
|  |  |
|  | import javax.swing.JFrame; |
|  | import javax.swing.JPanel; |
|  | import javax.swing.border.EmptyBorder; |
|  |  |
|  | import com.sun.javafx.iio.common.SmoothMinifier; |
|  |  |
|  | import java.awt.FlowLayout; |
|  | import javax.swing.JLabel; |
|  | import java.awt.Font; |
|  | import javax.swing.SwingConstants; |
|  | import javax.swing.JTextArea; |
|  | import java.awt.GridLayout; |
|  | import java.io.BufferedInputStream; |
|  | import java.io.BufferedReader; |
|  | import java.io.IOException; |
|  | import java.io.InputStream; |
|  | import java.io.InputStreamReader; |
|  | import java.io.OutputStream; |
|  | import java.io.Reader; |
|  | import java.net.HttpURLConnection; |
|  | import java.net.URL; |
|  | import java.net.URLEncoder; |
|  | import java.util.LinkedHashMap; |
|  | import java.util.Map; |
|  | import java.util.Random; |
|  | import java.util.Timer; |
|  | import java.util.TimerTask; |
|  |  |
|  | import org.json.JSONException; |
|  | import org.json.simple.JSONObject; |
|  | import org.json.simple.parser.JSONParser; |
|  | import org.json.simple.parser.ParseException; |
|  |  |
|  | import javax.swing.JTextField; |
|  |  |
|  | public class SensorAppGUI extends JFrame { |
|  |  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |  |
|  | private JPanel contentPane; |
|  | static int floorNum; |
|  | static int roomNum; |
|  | static int smokeLevel; |
|  | static int carbondioxideLevel; |
|  |  |
|  | public static void main(String[] args) { |
|  | TimerTask task = new TimerTask() { |
|  | public void run() { |
|  | Random rand = new Random(); |
|  | smokeLevel = rand.nextInt(9); |
|  | if (smokeLevel == 0 || smokeLevel == 1 ) { |
|  | carbondioxideLevel = 0; |
|  | } else { |
|  | carbondioxideLevel = smokeLevel - rand.nextInt(3); |
|  | } |
|  |  |
|  | //send sensorData |
|  | try { |
|  | JSONObject res = sendSensorData(); |
|  | System.out.println(res); |
|  | } catch (IOException | JSONException | ParseException e) { |
|  | e.printStackTrace(); |
|  | } |
|  | } |
|  | }; |
|  | Timer timer = new Timer(); |
|  | long delay = 0; |
|  | long intevalPeriod = 10 \* 1000; |
|  | // schedules the task to be run in an interval |
|  | timer.scheduleAtFixedRate(task, delay, intevalPeriod); |
|  |  |
|  |  |
|  | EventQueue.invokeLater(new Runnable() { |
|  | public void run() { |
|  | try { |
|  | SensorAppGUI frame = new SensorAppGUI(); |
|  | frame.setVisible(true); |
|  | } catch (Exception e) { |
|  | e.printStackTrace(); |
|  | } |
|  | } |
|  | }); |
|  |  |
|  | } |
|  |  |
|  | //send data to the REST API |
|  | public static JSONObject sendSensorData() throws IOException, JSONException, ParseException { |
|  | String query = "http://localhost:3000/sensorRecord"; |
|  | String jsonString = "{" |
|  | + "\"floor\_no\" : "+ floorNum + "," |
|  | + "\"room\_no\" : " + roomNum + "," |
|  | + "\"smoke\_level\" :" + smokeLevel + "," |
|  | + "\"carbondioxide\_level\" :" + carbondioxideLevel + "," |
|  | + "\"sensor\_status\" :" + "\"ACTIVE\"" |
|  | + "}"; |
|  |  |
|  | URL url = new URL(query); |
|  | HttpURLConnection conn = (HttpURLConnection) url.openConnection(); |
|  | conn.setConnectTimeout(5000); |
|  | conn.setRequestProperty("Content-Type", "application/json; charset=UTF-8"); |
|  | conn.setRequestProperty("Accept", "application/json"); |
|  | conn.setDoOutput(true); |
|  | conn.setDoInput(true); |
|  | conn.setRequestMethod("PUT"); |
|  |  |
|  | try (OutputStream os = conn.getOutputStream()) { |
|  | byte[] input = jsonString.getBytes("utf-8"); |
|  | os.write(input, 0, input.length); |
|  | os.close(); |
|  | } |
|  |  |
|  | // read the response |
|  | InputStream in = new BufferedInputStream(conn.getInputStream()); |
|  | String result = org.apache.commons.io.IOUtils.toString(in, "UTF-8"); |
|  | JSONParser parser = new JSONParser(); |
|  | JSONObject jsonObject = (JSONObject) parser.parse(result); |
|  |  |
|  |  |
|  | in.close(); |
|  | conn.disconnect(); |
|  |  |
|  | return jsonObject; |
|  | } |
|  |  |
|  | /\*\* |
|  | \* Create the frame. |
|  | \* @return |
|  | \*/ |
|  | public SensorAppGUI() { |
|  | setTitle("Sensor App"); |
|  | setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE); |
|  | setBounds(100, 100, 400, 300); |
|  | contentPane = new JPanel(); |
|  | contentPane.setBorder(new EmptyBorder(5, 5, 5, 5)); |
|  | setContentPane(contentPane); |
|  | contentPane.setLayout(null); |
|  |  |
|  | JLabel lblFloor = new JLabel("Floor : "); |
|  | lblFloor.setFont(new Font("Tahoma", Font.BOLD, 18)); |
|  | lblFloor.setBounds(85, 69, 107, 46); |
|  | contentPane.add(lblFloor); |
|  |  |
|  | JLabel lblRoom = new JLabel("Room : "); |
|  | lblRoom.setFont(new Font("Tahoma", Font.BOLD, 18)); |
|  | lblRoom.setBounds(85, 116, 107, 46); |
|  | contentPane.add(lblRoom); |
|  |  |
|  | JLabel floorNumlbl = new JLabel("12"); |
|  | floorNumlbl.setFont(new Font("Tahoma", Font.BOLD, 18)); |
|  | floorNumlbl.setBounds(199, 84, 56, 16); |
|  | contentPane.add(floorNumlbl); |
|  | this.floorNum = Integer.parseInt(floorNumlbl.getText()); |
|  |  |
|  | JLabel roomNumlbl = new JLabel("64"); |
|  | roomNumlbl.setFont(new Font("Tahoma", Font.BOLD, 18)); |
|  | roomNumlbl.setBounds(199, 133, 56, 16); |
|  | contentPane.add(roomNumlbl); |
|  | this.roomNum = Integer.parseInt(roomNumlbl.getText()); |
|  |  |
|  | } |
|  | } |

# **5.7 Web Client – reactJS**

## 5.7.1 [Web Client - ReactJS](https://github.com/SLIIT-DS/DS-Assignment-2_yr3_sem1/tree/Jayagoda/Web%20Client%20-%20ReactJS)/[src](https://github.com/SLIIT-DS/DS-Assignment-2_yr3_sem1/tree/Jayagoda/Web%20Client%20-%20ReactJS/src)/[components](https://github.com/SLIIT-DS/DS-Assignment-2_yr3_sem1/tree/Jayagoda/Web%20Client%20-%20ReactJS/src/components)/**NavBar.jsx:**

|  |
| --- |
| import React, {Component} from 'react'; |
|  | import './componentCSS/NavBar.css'; |
|  |  |
|  | class NavBar extends Component { |
|  | render() { |
|  | return ( |
|  | <div> |
|  | <nav className="navbar navbar-dark bg-dark navbar-expand-lg py-md-3"> |
|  | <span className='text-light'> <h2>Fire Alarm Monitoring System </h2></span> |
|  | <span className='fireIcon'><i className="fab fa-gripfire text-white fa-4x"></i></span> |
|  | </nav> |
|  | </div> |
|  | ); |
|  | } |
|  | } |
|  |  |
|  | export default NavBar; |

## 5.7.2 [Web Client - ReactJS](https://github.com/SLIIT-DS/DS-Assignment-2_yr3_sem1/tree/Jayagoda/Web%20Client%20-%20ReactJS)/[src](https://github.com/SLIIT-DS/DS-Assignment-2_yr3_sem1/tree/Jayagoda/Web%20Client%20-%20ReactJS/src)/[components](https://github.com/SLIIT-DS/DS-Assignment-2_yr3_sem1/tree/Jayagoda/Web%20Client%20-%20ReactJS/src/components)/**SensorsList.jsx :**

|  |
| --- |
|  |
|  | import React, {Component} from 'react';  import Sensor from "./sensor"; |
|  | import './componentCSS/SensorList.css'; |
|  | import {SensorContext} from "../contexts/sensorContext"; |
|  | import shortId from 'shortid' |
|  |  |
|  | class SensorsList extends Component { |
|  | static contextType = SensorContext |
|  | render() { |
|  |  |
|  | const { sensors } = this.context; |
|  | let sensorList = sensors.map( (sensor) => { |
|  | return <Sensor key={shortId.generate()} floorNum={sensor.floorNum} roomNum={sensor.roomNum} alertStatus={sensor.alertStatus}/> |
|  | }); |
|  |  |
|  | return ( |
|  | <div className="list-group card"> |
|  | {sensorList} |
|  | </div> |
|  | ); |
|  | } |
|  | } |
|  |  |
|  | export default SensorsList; |

## **5.7.3** [Web Client - ReactJS](https://github.com/SLIIT-DS/DS-Assignment-2_yr3_sem1/tree/Jayagoda/Web%20Client%20-%20ReactJS)/[src](https://github.com/SLIIT-DS/DS-Assignment-2_yr3_sem1/tree/Jayagoda/Web%20Client%20-%20ReactJS/src)/[components](https://github.com/SLIIT-DS/DS-Assignment-2_yr3_sem1/tree/Jayagoda/Web%20Client%20-%20ReactJS/src/components)/**sensor.jsx :**

|  |
| --- |
|  |
|  | import React, {Component} from 'react';  import {SensorContext} from "../contexts/sensorContext"; |
|  | import './sensor.css'; |
|  |  |
|  |  |
|  | /\*\* |
|  | \* |
|  | \* @author : Jayagoda N.M. |
|  | \* IT17184304 |
|  | \* |
|  | \*/ |
|  |  |
|  | class Sensor extends Component { |
|  | render() { |
|  | const { floorNum, roomNum, alertStatus } = this.props; |
|  | return ( |
|  | <SensorContext.Consumer>{( sensorContext ) => { |
|  | const { onClickSensor } = sensorContext; |
|  | let styleVal; |
|  | if (alertStatus == 'GREEN' ){ |
|  | styleVal = { |
|  | backgroundColor: "#008900" |
|  | } |
|  | } else if( alertStatus == 'RED') { |
|  | styleVal = { |
|  | backgroundColor: "#970000" |
|  | } |
|  | } else{ |
|  | styleVal = { |
|  | backgroundColor: "#b5b5b5", |
|  | color : "#000" |
|  | } |
|  | } |
|  | return( |
|  | <button type="button" className="list-group-item list-group-item-action card-body card-title" onClick={() => onClickSensor(floorNum, roomNum)} |
|  | style={styleVal}> |
|  | Sensor floor : {floorNum} &nbsp;&nbsp; Room : {roomNum} |
|  | </button> |
|  | ); |
|  | }}</SensorContext.Consumer> |
|  | ); |
|  | } |
|  | } |
|  |  |
|  | export default Sensor; |

## **5.7.4** [Web Client - ReactJS](https://github.com/SLIIT-DS/DS-Assignment-2_yr3_sem1/tree/Jayagoda/Web%20Client%20-%20ReactJS)/[src](https://github.com/SLIIT-DS/DS-Assignment-2_yr3_sem1/tree/Jayagoda/Web%20Client%20-%20ReactJS/src)/[components](https://github.com/SLIIT-DS/DS-Assignment-2_yr3_sem1/tree/Jayagoda/Web%20Client%20-%20ReactJS/src/components)/**sensorDataDisplay.jsx**

|  |
| --- |
|  |
|  | import React, {Component} from 'react';  import './componentCSS/sensorDataDisplay.css'; |
|  | import {SensorContext} from "../contexts/sensorContext"; |
|  |  |
|  | class SensorDataDisplay extends Component { |
|  |  |
|  | static contextType = SensorContext; |
|  | render() { |
|  | const { displayData } = this.context; |
|  | return ( |
|  | <div className='displayCard'> |
|  | <div className="sensorDetails card"> |
|  | <div className="card-header"> |
|  | Sensor Data |
|  | </div> |
|  | <div className="card-body sensorDetails"> |
|  | <p className="card-text"> |
|  | Floor Number : {displayData.floorNum} |
|  | </p> |
|  | <p className="card-text"> |
|  | Room Number : {displayData.roomNum} |
|  | </p> |
|  | <p className="card-text"> |
|  | Smoke Level : {displayData.smokeLevel} |
|  | </p> |
|  | <p className="card-text"> |
|  | CarbonDioxide Level : {displayData.carbondioxideLevel} |
|  | </p> |
|  | <p className="card-text"> |
|  | Sensor Status : {displayData.sensorStatus} |
|  | </p> |
|  | <p className="card-text"> |
|  | Alert Status : {displayData.alertStatus} |
|  | </p> |
|  | <p className="card-text"> |
|  | Last recorded Data and Time : {displayData.recordedDateAndTime} |
|  | </p> |
|  | </div> |
|  | </div> |
|  | <br/> |
|  | <br/> |
|  | <div className="card"> |
|  | <div className="card-header"> |
|  | Contact Details |
|  | </div> |
|  | <div className="card-body contactDetails"> |
|  | <p className="card-text"> |
|  | Contact Name : {displayData.ownerName} |
|  | </p> |
|  | <p className="card-text"> |
|  | Email : {displayData.email} |
|  | </p> |
|  | <p className="card-text"> |
|  | Phone Number : {displayData.phoneNumber} |
|  | </p> |
|  | </div> |
|  | </div> |
|  | </div> |
|  | ); |
|  | } |
|  | } |
|  |  |
|  | export default SensorDataDisplay; |

## 5.7.5 [Web Client - ReactJS](https://github.com/SLIIT-DS/DS-Assignment-2_yr3_sem1/tree/Jayagoda/Web%20Client%20-%20ReactJS)/[src](https://github.com/SLIIT-DS/DS-Assignment-2_yr3_sem1/tree/Jayagoda/Web%20Client%20-%20ReactJS/src)/[contexts](https://github.com/SLIIT-DS/DS-Assignment-2_yr3_sem1/tree/Jayagoda/Web%20Client%20-%20ReactJS/src/contexts)/**sensorContext.js**

|  |
| --- |
|  |
|  |  |
|  | import React, {Component , createContext} from 'react';  export const SensorContext = createContext(); |
|  |  |
|  | class SensorContextProvider extends Component { |
|  |  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |  |
|  | constructor(props) { |
|  | super(props); |
|  | this.state = { |
|  | sensorsBackend : [ |
|  |  |
|  | ], |
|  |  |
|  | sensors:[ |
|  |  |
|  | ], |
|  |  |
|  | currentSensor: { |
|  | floorNum: 0, |
|  | roomNum: 0, |
|  | smokeLevel: 0, |
|  | carbondioxideLevel: 0, |
|  | sensorStatus: '', |
|  | alertStatus: '', |
|  | ownerName: '', |
|  | email: '', |
|  | phoneNumber: '' |
|  | }, |
|  |  |
|  | displayData: { |
|  | floorNum: 0, |
|  | roomNum: 0, |
|  | smokeLevel: 0, |
|  | carbondioxideLevel: 0, |
|  | sensorStatus: '', |
|  | alertStatus: '', |
|  | ownerName: '', |
|  | email: '', |
|  | phoneNumber: '', |
|  | recordedDateAndTime: '' |
|  | } |
|  | }; |
|  | this.handleOnClick = this.handleOnClick.bind(this); |
|  | this.callAPI = this.callAPI.bind(this); |
|  | this.regularCallAPI = this.regularCallAPI.bind(this); |
|  |  |
|  | } |
|  |  |
|  | handleOnClick = (floor, room) => { |
|  | console.log("Sensors from backend : ", this.state.sensors ); |
|  |  |
|  | console.log("floor number", floor, ", room numbner : ", room); |
|  |  |
|  | let sensors = [...this.state.sensors]; |
|  | let sensorSelected = sensors.filter(function (sensor) { |
|  | return (sensor.roomNum == room && sensor.floorNum == floor); |
|  | }); |
|  | console.log("Sensor filtered : ", sensorSelected); |
|  | sensorSelected.length = 1; |
|  | sensorSelected = sensorSelected[0]; |
|  | console.log(sensorSelected); |
|  | this.setState({ |
|  | displayData: { |
|  | floorNum: sensorSelected.floorNum, |
|  | roomNum: sensorSelected.roomNum, |
|  | smokeLevel: sensorSelected.smokeLevel, |
|  | carbondioxideLevel: sensorSelected.carbondioxideLevel, |
|  | sensorStatus: sensorSelected.sensorStatus, |
|  | alertStatus: sensorSelected.alertStatus, |
|  | ownerName: sensorSelected.ownerName, |
|  | email: sensorSelected.email, |
|  | phoneNumber: sensorSelected.phoneNumber, |
|  | recordedDateAndTime : sensorSelected.recordedDateAndTime |
|  | } |
|  | }); |
|  | }; |
|  |  |
|  | async callAPI(){ |
|  | const url = "http://localhost:3000/sensorData"; |
|  | await fetch(url).then( res => res.json()).then( res => this.setState({ sensorsBackend : res})); |
|  | //console.log("Data from Backend : ", this.state.sensorsBackend); |
|  | let updatedSensors = this.state.sensorsBackend.map( sensor => { |
|  | return { |
|  | floorNum: sensor.floor\_no, |
|  | roomNum: sensor.room\_no, |
|  | smokeLevel: sensor.smoke\_level, |
|  | carbondioxideLevel: sensor.carbondioxide\_level, |
|  | sensorStatus: sensor.sensor\_status, |
|  | alertStatus: sensor.alert\_status, |
|  | ownerName: sensor.owner\_name, |
|  | email: sensor.email, |
|  | phoneNumber: sensor.phone\_number, |
|  | recordedDateAndTime: new Date(sensor.created\_date).toLocaleString() |
|  | } |
|  | }); |
|  | //console.log("Storing data to the state : ", updatedSensors); |
|  | this.setState({ sensors: updatedSensors }); |
|  | console.log("API call sent to the server"); |
|  |  |
|  | } |
|  |  |
|  | async regularCallAPI(){ |
|  | let result; |
|  | const url = "http://localhost:3000/sensorData"; |
|  | await fetch(url).then( res => res.json()).then( res => { result = res}); |
|  | console.log("Regular Data from Backend : ", result); |
|  | let updatedSensors = result.map( sensor => { |
|  | return { |
|  | floorNum: sensor.floor\_no, |
|  | roomNum: sensor.room\_no, |
|  | smokeLevel: sensor.smoke\_level, |
|  | carbondioxideLevel: sensor.carbondioxide\_level, |
|  | sensorStatus: sensor.sensor\_status, |
|  | alertStatus: sensor.alert\_status, |
|  | ownerName: sensor.owner\_name, |
|  | email: sensor.email, |
|  | phoneNumber: sensor.phone\_number, |
|  | recordedDateAndTime: new Date(sensor.created\_date).toLocaleString() |
|  | } |
|  | }); |
|  | await this.setState({ sensors: updatedSensors }); |
|  | console.log("sensor state updated ..."); |
|  | console.log("timerId :", this.timerId); |
|  | } |
|  |  |
|  | timerId = setInterval(() => { this.regularCallAPI() }, 15000); |
|  |  |
|  | componentWillMount() { |
|  | this.callAPI(); |
|  | } |
|  |  |
|  | componentWillUnmount() { |
|  | clearInterval(this.timerId); |
|  | } |
|  |  |
|  | render() { |
|  | return ( |
|  | <div> |
|  | <SensorContext.Provider value={{sensors: [...this.state.sensors], displayData: {...this.state.displayData}, onClickSensor: this.handleOnClick}}> |
|  | {this.props.children} |
|  | </SensorContext.Provider> |
|  | </div> |
|  | ); |
|  | } |
|  | } |
|  |  |
|  |  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |  |
|  | export default SensorContextProvider; |

## 5.7.6 [Web Client - ReactJS](https://github.com/SLIIT-DS/DS-Assignment-2_yr3_sem1/tree/Jayagoda/Web%20Client%20-%20ReactJS)/[src](https://github.com/SLIIT-DS/DS-Assignment-2_yr3_sem1/tree/Jayagoda/Web%20Client%20-%20ReactJS/src)/**App.js**

|  |
| --- |
|  |
|  | import React, {Component} from "react";  import './App.css'; |
|  | import NavBar from "./components/NavBar"; |
|  | import SensorsList from "./components/SensorsList"; |
|  | import SensorDataDisplay from "./components/sensorDataDisplay"; |
|  | import sensorDataConst from './sensorData' |
|  | import {SensorContext} from "./contexts/sensorContext"; |
|  |  |
|  | class App extends Component { |
|  | constructor(props) { |
|  | super(props); |
|  | } |
|  |  |
|  | render() { |
|  | return ( |
|  | <div> |
|  | <NavBar/> |
|  | <div className='contentBody'> |
|  | <div className='sideNav'> |
|  | <SensorsList /> |
|  | </div> |
|  | <div className='main'> |
|  | <div className="container card-body"> |
|  | <SensorDataDisplay/> |
|  | </div> |
|  | </div> |
|  | </div> |
|  | </div> |
|  | ); |
|  | } |
|  | } |
|  |  |
|  |  |
|  | export default App; |