****

**Sri Lanka Institute of Information Technology**

**DISTRIBUTED sYSTEMS**

**aSSIGNMENT 2**

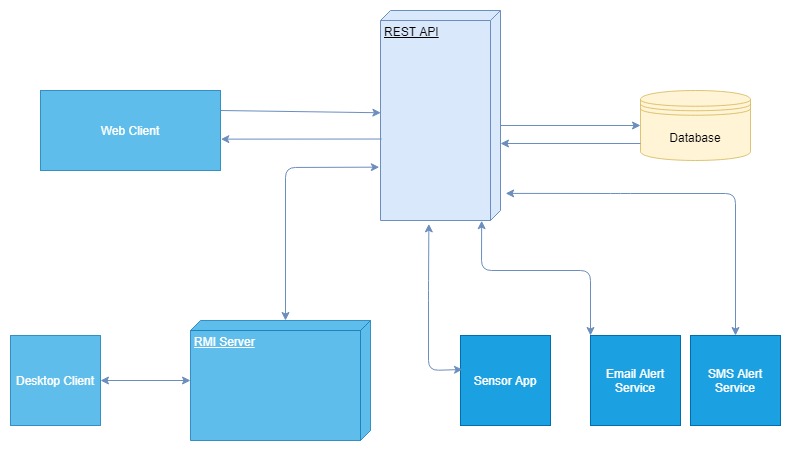
Group Members :

|  |  |
| --- | --- |
| IT18111552 | Weerakeshara T.N.N.D |
| IT18070668 | Wijesinghe B.T.C. |
| IT18100280 | Kumarage K.I.B. |
| IT18098556 | Rathnapala D.G.H |

**Content**

* High Level Software Architecture Diagram
* Brief explanation about the System Architecture
* Sequence Diagram - Monitoring sensors through desktop application
* Sequence Diagram – Monitoring sensors through web application
* Class Diagram with REST API & Database
* Deployment diagram
* Implementation of Communication between RMIServer and Desktop Client
* Implementation of Communication between RMIServer and Web API
* Appendix I - Screenshots of UI
* Appendix II – Code

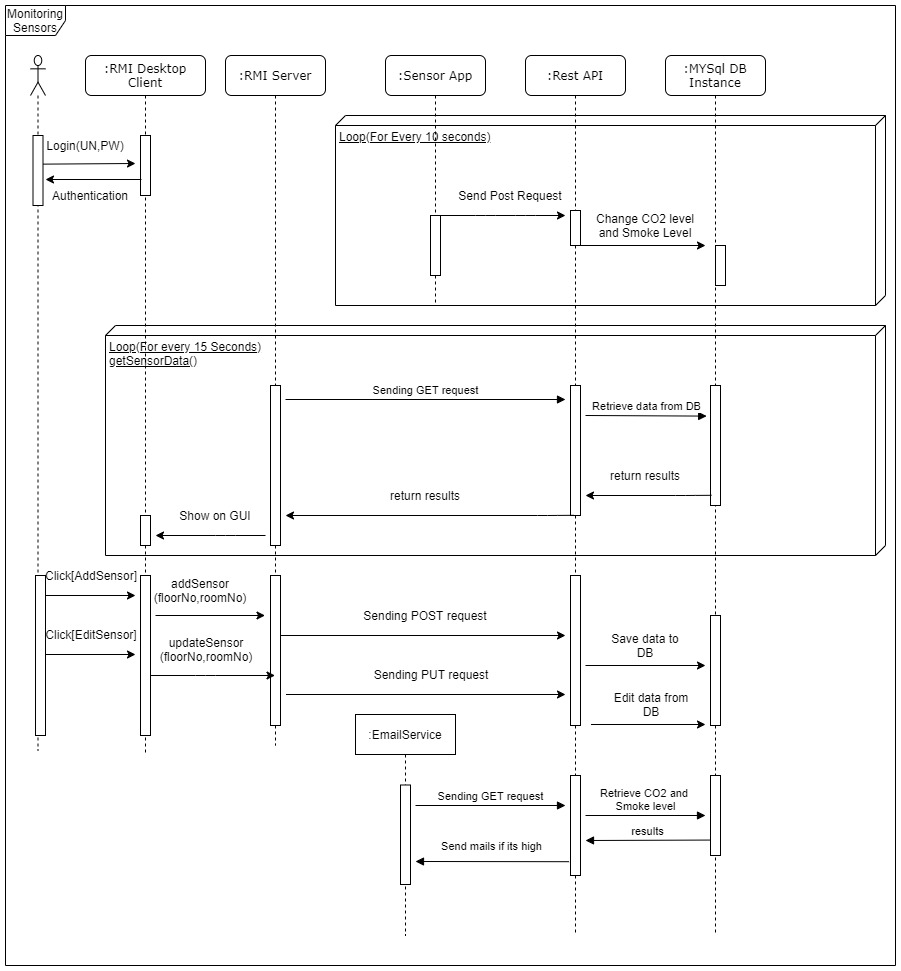
**High Level Software Architecture Diagram**

****

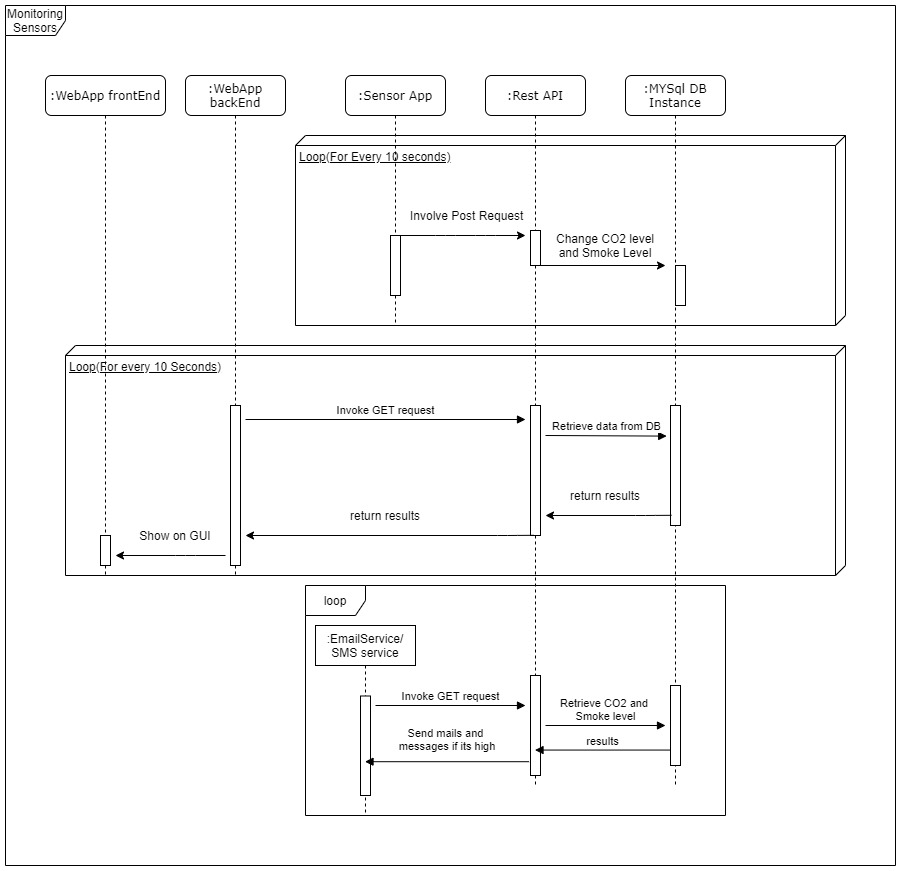
**Brief explanation about the System Architecture**

* This fire sensor monitoring system is implemented using the concepts of RMI and Restful web services.
* Restful web service is implemented using NodeJS and the web client is implemented using React.
* Java is used for the development of RMI client and the server.
* RMI is used to invoke the methods implemented in the server by the desktop client.
* Web client and RMI server will communicate with the database through Restful web service.
* There is asynchronous communication between Restful web service and the RMI sever. Desktop client can perform tasks without waiting till the responses come.
* There is a sensor application to change the sensor’s co2 level and smoke level for every 10 seconds.
* Email Service app and SMS Service app will trigger events if the co2 level or smoke level is high.
* MySQL database is used to store data because data are well structured.

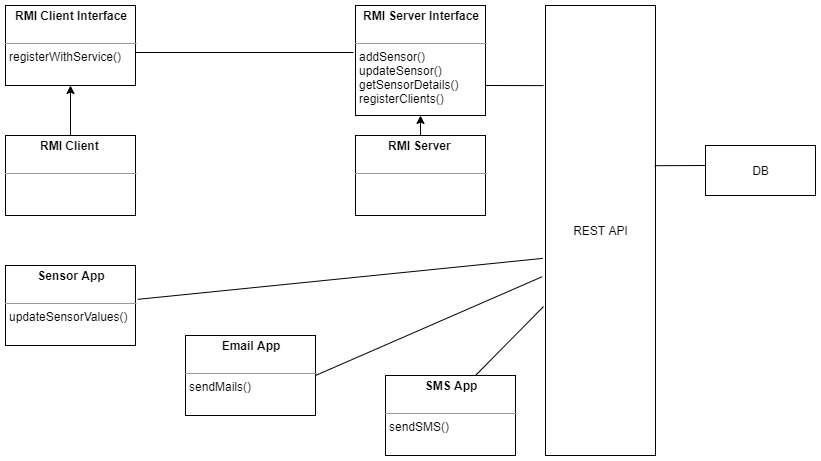
**Sequence Diagram - Monitoring sensors through desktop application**

****

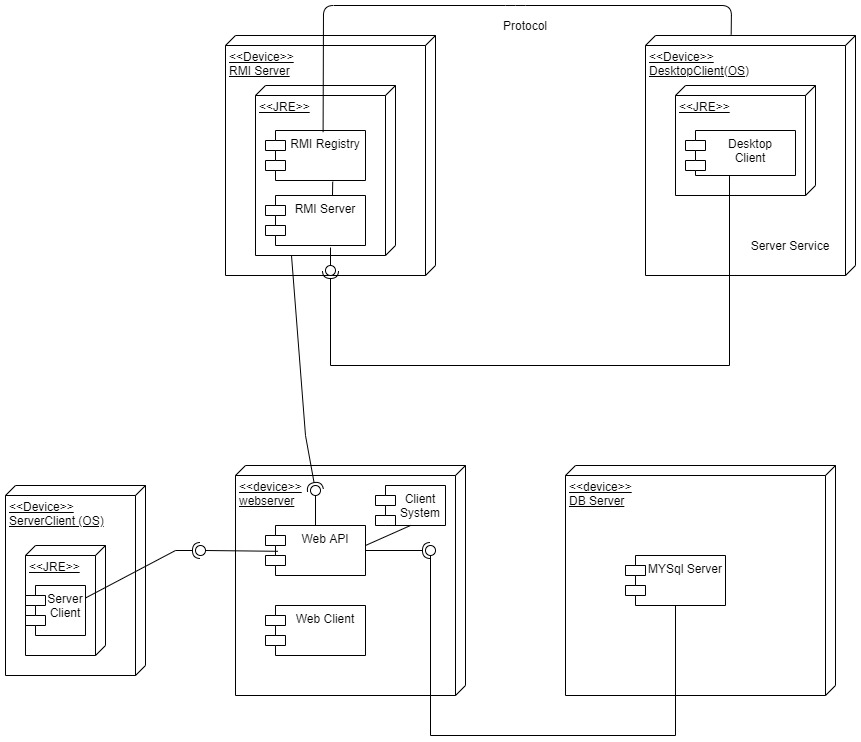
**Sequence Diagram – Monitoring sensors through web application**

****

**Class Diagram with REST API & Database**

****

**Deployment Diagram**

****

**Implementation of Communication between RMIServer and Desktop Client**

Java RMI concept is used here to establish communication between RMIServer and Desktop Client.

Registry reg = LocateRegistry.createRegistry(1099);

Here port 1099 has been used to create the RMI connection.

Desktop Client invokes methods in RMIServer through ServerService Interface (Remote interface).

**Code Segment from Desktop Client:**

**try {**

**Registry reg = LocateRegistry.getRegistry("localhost", 1099);**

**ServerService ss = (ServerService) reg.lookup("SensorService");**

**status = ss.delSensor(id); /\*delete method\*/**

**} catch (Exception e) {**

**e.printStackTrace();**

**}**

**Code Segment from ServerService (Remote interface):**

**public interface ServerService extends Remote {**

**public ArrayList sensorlist() throws RemoteException;**

**public int addSensor(String id, String fno, String rno) throws RemoteException;**

**public int updateSensor(String id,String active, String fno, String rno, String slevel, String co2level) throws RemoteException;**

**public int delSensor(String id) throws RemoteException;**

**}**

**Code Segment from RMIServer:**

**System.setProperty("java.security.policy", "file:allowall.policy");**

**try {**

**Registry reg = LocateRegistry.createRegistry(1099);**

**System.out.print(reg);**

**ServerService cl = new RMIServer();**

**reg.rebind("SensorService", cl);**

**} catch (Exception e){**

**e.printStackTrace();**

**}**

**Implementation of Communication between RMIServer and Web API**

Here RMIServer connects with the REST API which is in the URL “**http://localhost:3000**”.

**Code Segment from RMI Server (Sending a GET request):**

**try {**

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

**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())));**

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

**while ((jsonString = br.readLine()) != null) {**

**//System.out.println(jsonString);**

**finalJsonString = jsonString;**

**}**

**conn.disconnect();**

**} catch (MalformedURLException e) {**

**e.printStackTrace();**

**} catch (IOException e) {**

**e.printStackTrace();**

**}**

Code segment from REST API

//GET route --> http://localhost:3000  
app.get("/", function (req, res) {  
 //console.log("GET request received...");  
  
 connection.query("select \* from sensor\_list", function (  
 error,  
 results,  
 fields  
 ) {  
 if (error) throw error;

**A screenshot of a cell phone

Description automatically generatedAppendix I - Screenshots of UI**

Fig 1: Login UI of Desktop Client

A screenshot of a computer

Description automatically generated

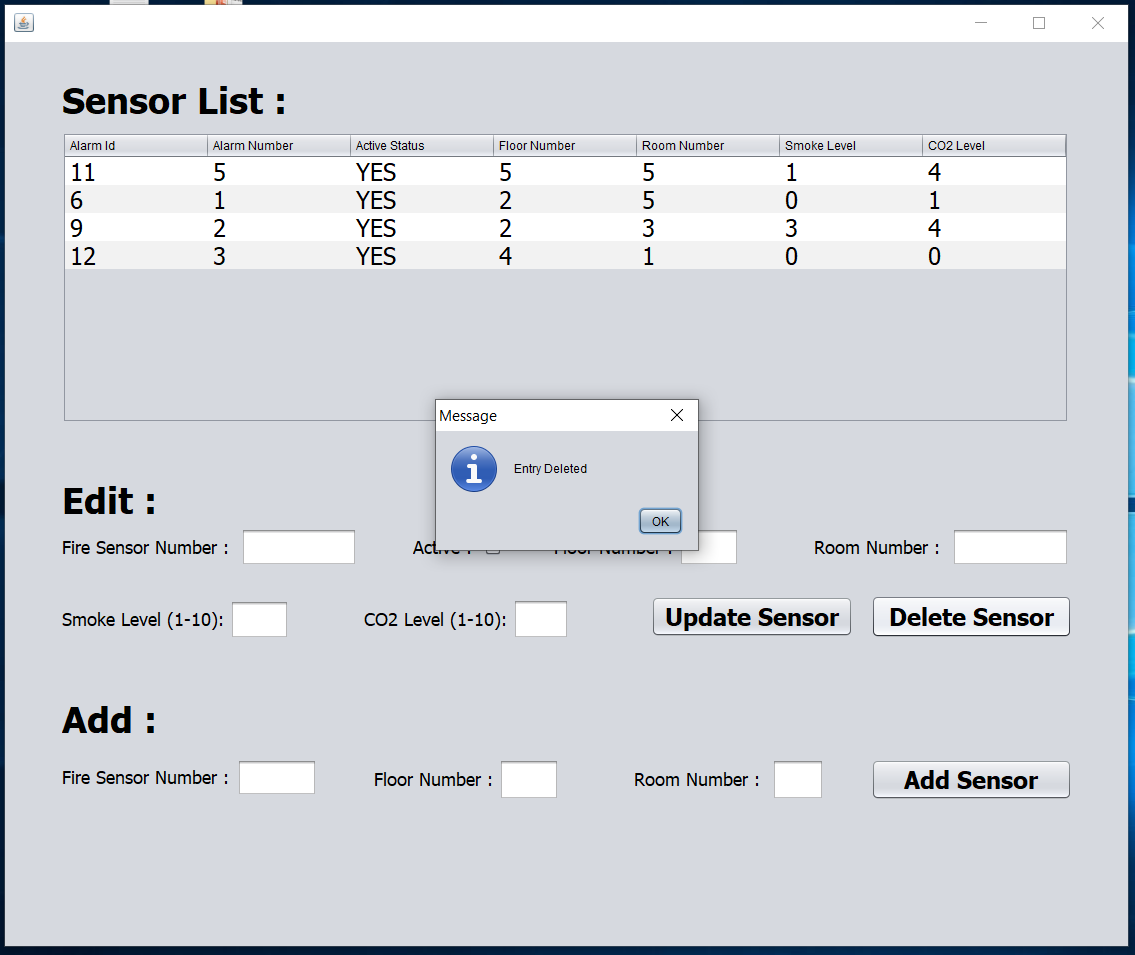
Fig 2: Main UI of Desktop Client

**A screenshot of a social media post

Description automatically generatedA screenshot of a computer

Description automatically generated** Fig 3: Main UI of Desktop Client (New Sensor Added)

Fig 4: Main UI of Desktop Client (Sensor Updated)

**** Fig 5: Main UI of Desktop Client (Sensor Deleted)

**A screenshot of a social media post

Description automatically generated**

Fig 6: Main UI of Desktop Client (High CO2 Alert)

**A screenshot of a social media post

Description automatically generated** Fig 7: Main UI of Desktop Client (High Smoke Alert)

**A screenshot of a social media post

Description automatically generated**

Fig 8: Main UI of Desktop Client (High CO2 and Smoke Alert)

**A screenshot of a computer

Description automatically generated** Fig 3: Main UI of Web Client

**A screenshot of a cell phone

Description automatically generated**

Fig 3: Alert Email

**A screenshot of a cell phone

Description automatically generated**

Fig 3: Alert Email Details

**Appendix II – Source Code**

**Remote interface of RMI server**

**ServerService.java:**

package rmiserver;

import java.rmi.Remote;

import java.rmi.RemoteException;

import java.sql.ResultSet;

import java.util.ArrayList;

public interface ServerService extends Remote {

public ArrayList sensorlist() throws RemoteException;

public int addSensor(String id, String fno, String rno) throws RemoteException;

public int updateSensor(String id,String active, String fno, String rno, String slevel, String co2level) throws RemoteException;

public int delSensor(String id) throws RemoteException;

}

**Implementation of RMI server**

**RMIServer.java:**

public class RMIServer extends UnicastRemoteObject implements ServerService{

public RMIServer() throws RemoteException{

super();

}

private static Connection con;

private PreparedStatement ps;

**/\*Method to send a GET request to REST API and fetch data from DB\*/**

public ArrayList sensorlist() throws RemoteException {

/\*Array List for each coloumn in table\*/

ArrayList<String> pk = new ArrayList<String>();

ArrayList<String> id = new ArrayList<String>();

ArrayList<String> active = new ArrayList<String>();

ArrayList<String> fnum = new ArrayList<String>();

ArrayList<String> rnum = new ArrayList<String>();

ArrayList<String> slevel = new ArrayList<String>();

ArrayList<String> co2level = new ArrayList<String>();

/\*Array list to add the above arrayliist. Here 7 arraylist will be added to this array list. Reason for this

is you cant pass a ResultSet directly using RMI. But you can pass arraylists using RMI\*/

ArrayList<ArrayList> list = new ArrayList<ArrayList>();

//calling rest api to fetch data from database

String jsonString;

String finalJsonString = null;

try {

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

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())));

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

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

//System.out.println(jsonString);

finalJsonString = jsonString;

}

conn.disconnect();

} catch (MalformedURLException e) {

e.printStackTrace();

} catch (IOException e) {

e.printStackTrace();

}

//System.out.println(finalJsonString);

//manipulating json object

try{

JSONArray jsonArray = new JSONArray(finalJsonString);

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

JSONObject jsonObject1 = jsonArray.getJSONObject(i);

pk.add(jsonObject1.optString("sensorId"));

id.add(jsonObject1.optString("sensorNo"));

active.add(jsonObject1.optString("active"));

fnum.add(jsonObject1.optString("floorNo"));

rnum.add(jsonObject1.optString("roomNo"));

slevel.add(jsonObject1.optString("smokeLevel"));

co2level.add(jsonObject1.optString("co2Level"));

}

}

catch (JSONException e){

e.printStackTrace();

}

/\*7 array lists will be added to the list arraylist\*/

list.add(pk);

list.add(id);

list.add(active);

list.add(fnum);

list.add(rnum);

list.add(slevel);

list.add(co2level);

return list;

}

**/\*Method to send a POST request to REST API and store data to DB\*/**

public int addSensor(String id, String fno, String rno) throws RemoteException {

int responseCode = 0;

//invoking rest api to add data

try {

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

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

conn.setDoOutput(true);

conn.setRequestMethod("POST");

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

//setting json object

String input = "{\"sensorNo\":"+id

+ ",\"active\" :1,\"floorNo\":"+fno

+ ", \"roomNo\":" +rno

+ ", \"smokeLevel\": 5, \"co2Level\": 5}";

OutputStream os = conn.getOutputStream();

os.write(input.getBytes());

os.flush();

//saving response code to a variable

responseCode = conn.getResponseCode();

conn.disconnect();

} catch (MalformedURLException e) {

e.printStackTrace();

} catch (IOException e) {

e.printStackTrace();

}

if(responseCode == 200){

return 1;

}

else

return 0;

}

**/\*Method to send a PUT request to REST API and edit data from DB\*/**

public int updateSensor(String id,String active, String fno, String rno, String slevel, String co2level) throws RemoteException {

int responseCode = 0;

//invoking rest api to update data

try {

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

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

conn.setDoOutput(true);

conn.setRequestMethod("PUT");

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

//setting json object

String input = "{\"sensorNo\":"+id

+ ",\"active\" :" +active

+ ",\"floorNo\":"+fno

+ ", \"roomNo\":" +rno

+ ", \"smokeLevel\":" +slevel

+ ", \"co2Level\": " +co2level

+ "}";

OutputStream os = conn.getOutputStream();

os.write(input.getBytes());

os.flush();

//saving response code to a variable

responseCode = conn.getResponseCode();

conn.disconnect();

} catch (MalformedURLException e) {

e.printStackTrace();

} catch (IOException e) {

e.printStackTrace();

}

if(responseCode == 200){

return 1;

}

else

return 0;

}

**/\*Method to send a DELETE request to REST API and delete data from DB\*/**

public int delSensor(String id) throws RemoteException {

int responseCode = 0;

//invoking rest api to delete data

try {

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

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

conn.setDoOutput(true);

conn.setRequestMethod("DELETE");

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

//setting json object

String input = "{\"sensorNo\":"+id

+ "}";

OutputStream os = conn.getOutputStream();

os.write(input.getBytes());

os.flush();

//saving response code to a variable

responseCode = conn.getResponseCode();

conn.disconnect();

} catch (MalformedURLException e) {

e.printStackTrace();

} catch (IOException e) {

e.printStackTrace();

}

if(responseCode == 200){

return 1;

}

else

return 0;

}

**/\*Registering RMI Clients\*/**

public static void main(String[] args){

/\*To establish RMI connection\*/

System.setProperty("java.security.policy", "file:allowall.policy");

try {

Registry reg = LocateRegistry.createRegistry(1099);

System.out.print(reg);

ServerService cl = new RMIServer();

reg.rebind("SensorService", cl);

} catch (Exception e){

e.printStackTrace();

}

}

}

**Callback interface of RMI client**

public interface DesktopListner extends java.rmi.Remote{

registerToServer();

}

**Implementation of RMI client (GUI)**

**DesktopFrame.java:**

**(Auto generated codes are removed that relates to produce the UI)**

public class DesktopFrame extends javax.swing.JFrame {

/\*\*

\* Creates new form DesktopFrame

\*/

String uninputi;

public DesktopFrame() {

initComponents();

}

void viewPanel(String cname) {

CardLayout cL = (CardLayout) jPanel1.getLayout();

cL.show(jPanel1, cname);

}

**//login method**

private void jButton1ActionPerformed(java.awt.event.ActionEvent evt) {

/\*username and password is checked\*/

if (uninput.getText().toString().equals("admin") && pwinput.getText().toString().equals("admin")) {

System.out.println("success login");

/\*To establish RMI connection\*/

System.setProperty("java.security.policy", "file:allowall.policy");

try {

Registry reg = LocateRegistry.getRegistry("localhost", 1099);

ServerService ss = (ServerService) reg.lookup("SensorService");

ArrayList list = ss.sensorlist();

ArrayList pk = (ArrayList) list.get(0);

ArrayList id = (ArrayList) list.get(1);

ArrayList active = (ArrayList) list.get(2);

ArrayList fnum = (ArrayList) list.get(3);

ArrayList rnum = (ArrayList) list.get(4);

ArrayList slevel = (ArrayList) list.get(5);

ArrayList rlevel = (ArrayList) list.get(6);

DefaultTableModel dm = (DefaultTableModel) sensortable.getModel();

while (dm.getRowCount() > 0) {

dm.removeRow(0);

}

/\*data from arraylists are added to table\*/

DefaultTableModel model = (DefaultTableModel) sensortable.getModel();

Object rawdata[] = new Object[7];

for (int i = 0; i < pk.size(); i++) {

rawdata[0] = pk.get(i);

rawdata[1] = id.get(i);

if (active.get(i).equals("1")) {

rawdata[2] = "YES";

} else {

rawdata[2] = "NO";

}

rawdata[3] = fnum.get(i);

rawdata[4] = rnum.get(i);

rawdata[5] = slevel.get(i);

rawdata[6] = rlevel.get(i);

model.addRow(rawdata);

}

} catch (Exception e) {

e.printStackTrace();

}

viewPanel("card3");

}

}

**//data table clicked event**

private void sensortableMouseClicked(java.awt.event.MouseEvent evt) {

int rowselected = sensortable.getSelectedRow();

String fireSensorNo = sensortable.getValueAt(rowselected, 1).toString();

String active = sensortable.getValueAt(rowselected, 2).toString();

String fNo = sensortable.getValueAt(rowselected, 3).toString();

String rNo = sensortable.getValueAt(rowselected, 4).toString();

String sLevel = sensortable.getValueAt(rowselected, 5).toString();

String co2Level = sensortable.getValueAt(rowselected, 6).toString();

/\*Through this data will be loaded to input fields allowing the user to edit them\*/

alarmnumberinput.setText(fireSensorNo);

floorinput.setText(fNo);

roominput.setText(rNo);

if (active.equals("YES")) {

activecheck.setSelected(true);

} else {

activecheck.setSelected(false);

}

smokeinput.setText(sLevel);

co2input.setText(co2Level);

}

**//add button**

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

String id = addalarmnumberinput.getText();

String fno = addfloorinput.getText();

String rno = addroominput.getText();

/\*To establish RMI connection\*/

System.setProperty("java.security.policy", "file:allowall.policy");

int status = 0;

try {

Registry reg = LocateRegistry.getRegistry("localhost", 1099);

ServerService ss = (ServerService) reg.lookup("SensorService");

status = ss.addSensor(id, fno, rno); /\*add method\*/

} catch (Exception e) {

e.printStackTrace();

}

addalarmnumberinput.setText("");

addfloorinput.setText("");

addroominput.setText("");

if (status == 1) {

JOptionPane.showMessageDialog(null, "Entry Added");

} else {

JOptionPane.showMessageDialog(null, "Entry Failed");

}

}

**//update button**

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

String id = alarmnumberinput.getText();

String active = "0";

if (activecheck.isSelected()) {

active = "1";

} else {

active = "0";

}

String fno = floorinput.getText();

String rno = roominput.getText();

String slevel = smokeinput.getText();

String co2level = co2input.getText();

/\*To establish RMI connection\*/

System.setProperty("java.security.policy", "file:allowall.policy");

int status = 0;

try {

Registry reg = LocateRegistry.getRegistry("localhost", 1099);

ServerService ss = (ServerService) reg.lookup("SensorService");

status = ss.updateSensor(id, active, fno, rno, slevel, co2level); /\*update method\*/

} catch (Exception e) {

e.printStackTrace();

}

alarmnumberinput.setText("");

floorinput.setText("");

roominput.setText("");

activecheck.setSelected(false);

smokeinput.setText("");

co2input.setText("");

if (status == 1) {

JOptionPane.showMessageDialog(null, "Entry Updated");

} else {

JOptionPane.showMessageDialog(null, "Failed");

}

}

**//delete button**

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

String id = alarmnumberinput.getText();

/\*To establish RMI connection\*/

System.setProperty("java.security.policy", "file:allowall.policy");

int status = 0;

try {

Registry reg = LocateRegistry.getRegistry("localhost", 1099);

ServerService ss = (ServerService) reg.lookup("SensorService");

status = ss.delSensor(id); /\*delete method\*/

} catch (Exception e) {

e.printStackTrace();

}

alarmnumberinput.setText("");

floorinput.setText("");

roominput.setText("");

activecheck.setSelected(false);

smokeinput.setText("");

co2input.setText("");

if (status == 1) {

JOptionPane.showMessageDialog(null, "Entry Deleted");

} else {

JOptionPane.showMessageDialog(null, "Failed");

}

}

public static void main(String args[]) {

/\* Set the Nimbus look and feel \*/

//<editor-fold defaultstate="collapsed" desc=" Look and feel setting code (optional) ">

/\* If Nimbus (introduced in Java SE 6) is not available, stay with the default look and feel.

\* For details see http://download.oracle.com/javase/tutorial/uiswing/lookandfeel/plaf.html

\*/

try {

for (javax.swing.UIManager.LookAndFeelInfo info : javax.swing.UIManager.getInstalledLookAndFeels()) {

if ("Nimbus".equals(info.getName())) {

javax.swing.UIManager.setLookAndFeel(info.getClassName());

break;

}

}

} catch (ClassNotFoundException ex) {

java.util.logging.Logger.getLogger(DesktopFrame.class.getName()).log(java.util.logging.Level.SEVERE, null, ex);

} catch (InstantiationException ex) {

java.util.logging.Logger.getLogger(DesktopFrame.class.getName()).log(java.util.logging.Level.SEVERE, null, ex);

} catch (IllegalAccessException ex) {

java.util.logging.Logger.getLogger(DesktopFrame.class.getName()).log(java.util.logging.Level.SEVERE, null, ex);

} catch (javax.swing.UnsupportedLookAndFeelException ex) {

java.util.logging.Logger.getLogger(DesktopFrame.class.getName()).log(java.util.logging.Level.SEVERE, null, ex);

}

//</editor-fold>

/\* Create and display the form \*/

java.awt.EventQueue.invokeLater(new Runnable() {

public void run() {

new DesktopFrame().setVisible(true);

}

});

**//Refresh desktop GUI for every 15 seconds**

//creating timer task, timer

Timer t = new Timer();

TimerTask tt = new TimerTask() {

int n = 0;

int p = 0;

public void run() {

/\*To establish RMI connection\*/

System.setProperty("java.security.policy", "file:allowall.policy");

try {

Registry reg = LocateRegistry.getRegistry("localhost", 1099);

ServerService ss = (ServerService) reg.lookup("SensorService");

ArrayList list = ss.sensorlist();

ArrayList pk = (ArrayList) list.get(0);

ArrayList id = (ArrayList) list.get(1);

ArrayList active = (ArrayList) list.get(2);

ArrayList fnum = (ArrayList) list.get(3);

ArrayList rnum = (ArrayList) list.get(4);

ArrayList slevel = (ArrayList) list.get(5);

ArrayList rlevel = (ArrayList) list.get(6);

DefaultTableModel dm = (DefaultTableModel) sensortable.getModel();

while (dm.getRowCount() > 0) {

dm.removeRow(0);

}

DefaultTableModel model = (DefaultTableModel) sensortable.getModel();

Object rawdata[] = new Object[7];

for (int i = 0; i < pk.size(); i++) {

rawdata[0] = pk.get(i);

rawdata[1] = id.get(i);

if (active.get(i).equals("1")) {

rawdata[2] = "YES";

} else {

rawdata[2] = "NO";

}

rawdata[3] = fnum.get(i);

rawdata[4] = rnum.get(i);

rawdata[5] = slevel.get(i);

rawdata[6] = rlevel.get(i);

if(Integer.parseInt(slevel.get(i).toString()) > 5){

n++;

}if (Integer.parseInt(rlevel.get(i).toString()) > 5){

p++;

}

model.addRow(rawdata);

}

if((n > 0) && (p > 0)){

JOptionPane frame = new JOptionPane();

JOptionPane.showMessageDialog(frame,

"Smoke and CO2 are High",

"Warning",

JOptionPane.WARNING\_MESSAGE);

n=0; p=0;

}

else if(n > 0){

JOptionPane frame = new JOptionPane();

JOptionPane.showMessageDialog(frame,

"Smoke is High",

"Warning",

JOptionPane.WARNING\_MESSAGE);

n=0;p=0;

}else if(p > 0){

JOptionPane frame = new JOptionPane();

JOptionPane.showMessageDialog(frame,

"CO2 is High",

"Warning",

JOptionPane.WARNING\_MESSAGE);

n=0;p=0;

}

} catch (Exception e) {

e.printStackTrace();

}

}

;

};

t.scheduleAtFixedRate(tt, 0, 15000);

}

}

**Implementation of REST API**

**Server.js:**

const http = require("http");  
const express = require("express");  
const mysql = require("mysql");  
const bodyParser = require("body-parser");  
const cors = require("cors");  
  
  
const app = express();  
app.use(cors());  
const connection = mysql.createConnection({  
 host: "localhost",  
 user: "root",  
 password: "",  
 database: "sensor\_db",  
 port: "3308"  
});  
process.env["NODE\_TLS\_REJECT\_UNAUTHORIZED"] = 0;  
connection.connect(function (err) {  
 if (err) throw err;  
 ***console***.log("You are now connected with mysql database...");  
});  
  
app.use(bodyParser.json()); // to support JSON-encoded bodies  
app.use(  
 bodyParser.urlencoded({  
 // to support URL-encoded bodies  
 extended: true,  
 })  
);  
//Start server  
var server = app.listen(3000, "localhost", function () {  
 var host = server.address().address;  
 var port = server.address().port;  
  
 ***console***.log("Example app listening at http://%s:%s", host, port);  
});  
  
//GET route --> http://localhost:3000  
app.get("/", function (req, res) {  
 //console.log("GET request received...");  
  
 connection.query("select \* from sensor\_list", function (  
 error,  
 results,  
 fields  
 ) {  
 if (error) throw error;   
 res.send(***JSON***.stringify(results));  
 });  
});  
  
//POST  
app.post("/add", (req, res) => {  
 const { sensorNo, active, roomNo, floorNo, smokeLevel, co2Level } = req.body;  
  
 res.send({  
 sensorNo,  
 active,  
 roomNo,  
 floorNo,  
 smokeLevel,  
 co2Level,  
 });  
 connection.query(  
 `INSERT INTO sensor\_list (sensorNo, active, floorNo, roomNo, smokeLevel, co2Level) VALUES ('${sensorNo}', '${active}', '${roomNo}', '${floorNo}', '${smokeLevel}', '${co2Level}') `,  
 function (err, result) {  
 if (err) throw err;  
 }  
 );  
});  
  
//PUT  
app.put("/update", (req, res) => {  
 const { sensorNo, active, roomNo, floorNo, smokeLevel, co2Level } = req.body;  
  
 connection.query(  
 `UPDATE sensor\_list SET active = '${active}', floorNo = '${floorNo}', roomNo = '${roomNo}', smokeLevel = '${smokeLevel}', co2Level = '${co2Level}' WHERE sensorNo = '${sensorNo}' `,  
 function (err, result) {  
 if (err) throw err;  
  
 return res.send("Received a PUT HTTP request");  
 }  
 );  
});  
  
//DELETE  
app.delete("/delete", (req, res) => {  
 const { sensorNo } = req.body;  
  
 connection.query(  
 `DELETE FROM sensor\_list WHERE sensorNo = '${sensorNo}' `,  
 function (err, result) {  
 if (err) throw err;  
  
 return res.send("Received a DELETE HTTP request");  
 }  
 );  
});  
  
//this will be used for timely update sensors co2 and smoke levels  
app.put("/updatesensor", (req, res) => {  
 const { sensorNo, smokeLevel, co2Level } = req.body;  
  
 connection.query(  
 `UPDATE sensor\_list SET smokeLevel = '${smokeLevel}', co2Level = '${co2Level}' WHERE sensorNo = '${sensorNo}' `,  
 function (err, result) {  
 if (err) throw err;  
  
 return res.send("Received a PUT HTTP request");  
 ***console***.log("Received a PUT HTTP request");  
 }  
 );  
});

**Implementation of Web client (GUI)**

**App.js:**

import ***React*** from "react";  
import "./App.css";  
import { Component } from "react";  
import ***axios*** from "axios";  
import Header from "./components/Header";  
import FireAlarm from "./components/FireAlarm";  
import { Container, Row, Col } from "reactstrap";  
  
class App extends Component {  
 constructor(props) {  
 super(props);  
 this.state = {  
 sensors: [],  
 };  
 }  
  
 componentDidMount() {  
 this.interval = setInterval(() => {  
 ***axios***.get("http://localhost:3000").then((res) => {  
 const sensors = res.data;  
 this.setState({ sensors });  
 {  
 ***console***.log(this.state.sensors);  
 }  
 });  
 }, 5000);  
 }  
  
 render() {  
 return (  
 <div className="App">  
 <Header></Header>  
 <Container>  
 <Row>  
 {" "}  
 {this.state.sensors.map((sensor) => {  
 return (  
 <FireAlarm  
 key={sensor.sensorId}  
 id={sensor.sensorNo}  
 active={sensor.active}  
 floor={sensor.floorNo}  
 room={sensor.roomNo}  
 smokelevel={sensor.smokeLevel}  
 co2level={sensor.co2Level}  
 ></FireAlarm>  
 );  
 })}  
 </Row>  
 </Container>  
 </div>  
 );  
 }  
}  
  
export default App;

**FireAlarm.js:**

import ***React*** from "react";  
import {  
 Card,  
 Badge,  
 CardHeader,  
 CardFooter,  
 CardBody,  
 CardTitle,  
 CardText,  
 Progress,  
} from "reactstrap";  
  
export default function FireAlarm(props) {  
 const { id, active, floor, room, smokeLevel, co2Level } = props;  
  
 return (  
 <div>  
 <Card  
 style={{ margin: "10px" }}  
 body  
 outline  
 color={  
 props.smokelevel > 5 || props.co2level > 5 ? "danger" : "success"  
 }  
 >  
 <CardHeader className="text-center" tag="h3">  
 Alarm {id}  
 </CardHeader>  
 <CardBody>  
 <CardTitle>  
 Status:{" "}  
 <Badge color={active === 1 ? "success" : "danger"}>  
 {active === 1 ? "Active" : "Disabled"}  
 </Badge>  
 </CardTitle>  
 <CardText>Floor Number: {floor}</CardText>  
 <CardText>Room Number: {room}</CardText>  
  
 <Progress  
 style={{ width: "70%" }}  
 value={(props.smokelevel / 10) \* 100}  
 color={props.smokelevel > 5 ? "danger" : "success"}  
 />  
 <CardText>Smoke Level: {props.smokelevel}</CardText>  
 <Progress  
 style={{ width: "70%" }}  
 value={(props.co2level / 10) \* 100}  
 color={props.co2level > 5 ? "danger" : "success"}  
 />  
 <CardText>CO2 Level: {props.co2level}</CardText>  
 </CardBody>  
 <CardFooter className="text-muted"></CardFooter>  
 </Card>  
 </div>  
 );  
}

**Header.js:**

import ***React*** from "react";  
import { Navbar, NavbarBrand } from "reactstrap";  
  
export default function Header() {  
 return (  
 <div>  
 <Navbar color="dark" dark expand="md">  
 <NavbarBrand href="/">Fire Alarm Dashboard</NavbarBrand>  
 </Navbar>  
 </div>  
 );  
}

**Implementation of Sensor App (To change sensor’s co2 level and smoke level for every 10 seconds)**

**SensorClient**.**java:**

public class SensorClient {

private static Connection conn = DBConnect.DBConn();

private static PreparedStatement pst;

private static ResultSet rs;

public static void main(String args[]) {

//creating timer

Timer t = new Timer();

//creating timertask

TimerTask tt = new TimerTask() {

@Override

public void run(){

Random rand = new Random();

try {

String query = "SELECT \* FROM sensor\_list";

pst = conn.prepareStatement(query);

rs = pst.executeQuery();

while (rs.next()) {

//System.out.println(rs.getInt(1));

pst = conn.prepareStatement("UPDATE sensor\_list SET smokeLevel = ?, co2Level = ? WHERE sensorId = ? ");

pst.setString(1, Integer.toString(rand.nextInt(10)));

pst.setString(2, Integer.toString(rand.nextInt(10)));

pst.setString(3, Integer.toString(rs.getInt(1)));

pst.execute();

}

} catch (SQLException e) {

// TODO Auto-generated catch block

e.printStackTrace();

}

};

};

//executing the timer

t.scheduleAtFixedRate(tt,0,10000);

}

}

**Implementation of Email Service App (To send emails when co2 level and smoke level are high)**

const http = require("http");  
const express = require("express");  
const mysql = require("mysql");  
const bodyParser = require("body-parser");  
const cors = require("cors");  
var nodemailer = require("nodemailer");  
  
  
const app = express();  
app.use(cors());  
const connection = mysql.createConnection({  
 host: "localhost",  
 user: "root",  
 password: "",  
 database: "sensor\_db",  
 port: "3308"  
 });  
  
process.env["NODE\_TLS\_REJECT\_UNAUTHORIZED"] = 0;  
connection.connect(function (err) {  
 if (err) throw err;  
 ***console***.log("Connected to MySQL Database...");  
});  
  
app.use(bodyParser.json()); // to support JSON-encoded bodies  
app.use(  
 bodyParser.urlencoded({  
 // to support URL-encoded bodies  
 extended: true,  
 })  
);  
//Start server  
var server = app.listen(3005, "localhost", function () {  
 var host = server.address().address;  
 var port = server.address().port;  
  
 ***console***.log("Connected to server : port ",port);  
});  
  
setInterval(()=>{  
 connection.query("select \* from sensor\_list", function (  
 error,  
 results,  
 fields  
 ) {  
 if (error) throw error;  
 var sensors = ***JSON***.stringify(results);  
 // console.log(results[0].co2Level);  
 // console.log(results[0].smokeLevel);  
   
 // console.log(JSON.stringify(results,["smokeLevel"]));  
 // console.log(results[].["co2Level"]);  
 if (results[0].co2Level > 5 && results[0].smokeLevel > 5) {  
 var transporter = nodemailer.createTransport({  
 service: "gmail",  
 auth: {  
 user: "sliitgroup19@gmail.com",  
 pass: "sliit123",  
 },  
 });  
   
 var mailOption = {  
 from: "sliitgroup19@gmail.com",  
 to: "sliitgroup19@gmail.com",  
 subject: "Fire Alarm Alert",  
 text: `Both CO2 and Smoke level is high`,  
 };  
   
 transporter.sendMail(mailOption, function (error, info) {  
 if (error) {  
 ***console***.log(error);  
 } else {  
 ***console***.log("Email sent: " + info.response);  
 ***console***.log("Both CO2 and Smoke level is high!!");  
 }  
 });  
 } else if (results[0].co2Level > 5) {  
 var transporter = nodemailer.createTransport({  
 service: "gmail",  
 auth: {  
 user: "sliitgroup19@gmail.com",  
 pass: "sliit123",  
 },  
 });  
   
 var mailOption = {  
 from: "sliitgroup19@gmail.com",  
 to: "sliitgroup19@gmail.com",  
 subject: "Fire Alarm Alert",  
 text: `CO2 level is high`,  
 };  
   
 transporter.sendMail(mailOption, function (error, info) {  
 if (error) {  
 ***console***.log(error);  
 } else {  
 ***console***.log("Email sent: " + info.response);  
 ***console***.log("CO2 level is high!!");  
 }  
 });  
 } else if (results[0].smokeLevel > 5) {  
 var transporter = nodemailer.createTransport({  
 service: "gmail",  
 auth: {  
 user: "sliitgroup19@gmail.com",  
 pass: "sliit123",  
 },  
 });  
   
 var mailOption = {  
 from: "sliitgroup19@gmail.com",  
 to: "sliitgroup19@gmail.com",  
 subject: "Fire Alarm Alert",  
 text: `Smoke Level level is high`,  
 };  
   
 transporter.sendMail(mailOption, function (error, info) {  
 if (error) {  
 ***console***.log(error);  
 } else {  
 ***console***.log("Email sent: " + info.response);  
 ***console***.log("Smoke level is high!!");  
 }  
 });  
 } else{  
 ***console***.log("Status is good!!");  
 }   
   
   
 });  
},10000);