**Sri Lanka Institute of Information**

**Technology**



**Distributed Systems**

**Assignment 2**

Group Members

|  |  |
| --- | --- |
| Name | Registration number |
| Liyanarachchi L. L. R. K | IT18113150 |
| Samarawickrama H. U | IT18123050 |
| Premathilaka R.S.M | IT18124668 |
| M.N.D.S. Thilakasiri | IT18113082 |

Table of Contents

[1. High Level Architectural diagram 3](#_Toc39065521)

[2. Service interfaces and the sequence diagrams 4](#_Toc39065522)

[1. Web Client 4](#_Toc39065523)

[2. Desktop Client 5](#_Toc39065524)

[3. Sensor Client 7](#_Toc39065525)

[4. Email service 7](#_Toc39065526)

[Conclusion 8](#_Toc39065527)

[Appendix 9](#_Toc39065528)

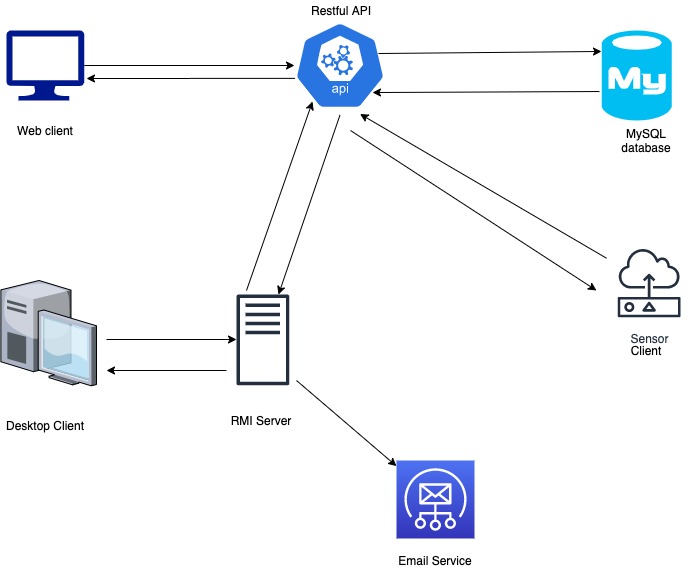
[1. Restful API – NodeJS 9](#_Toc39065529)

[2. Web Client - ReactJS 11](#_Toc39065530)

[3.Sensor Client – Java 14](#_Toc39065531)

[4.Desktop Client 16](#_Toc39065532)

# High Level Architectural diagram



# Service interfaces and the sequence diagrams

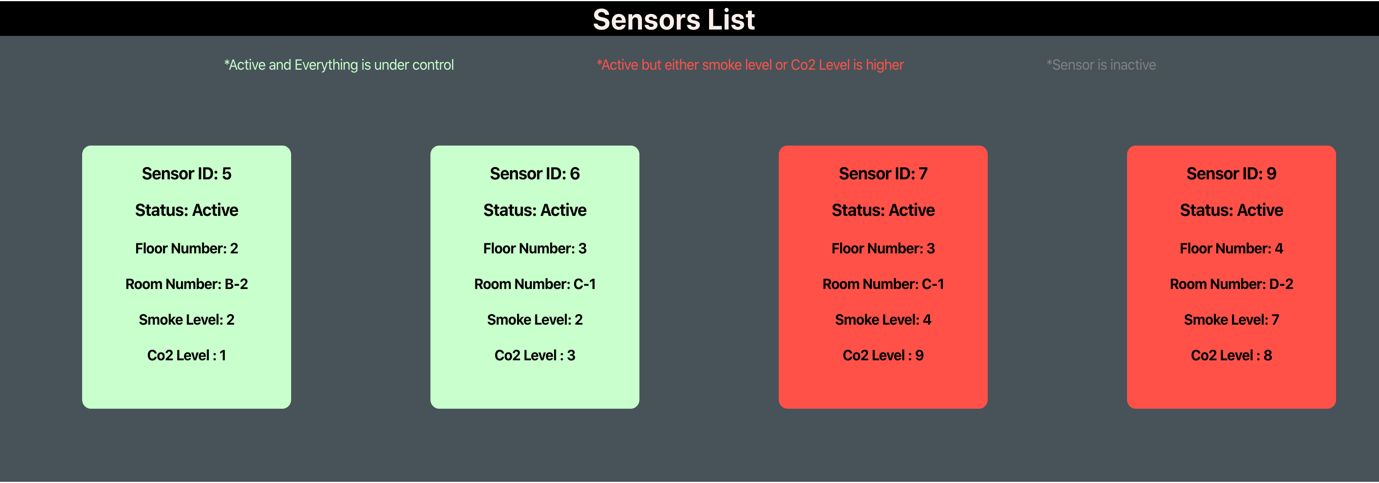
## Web Client

Description

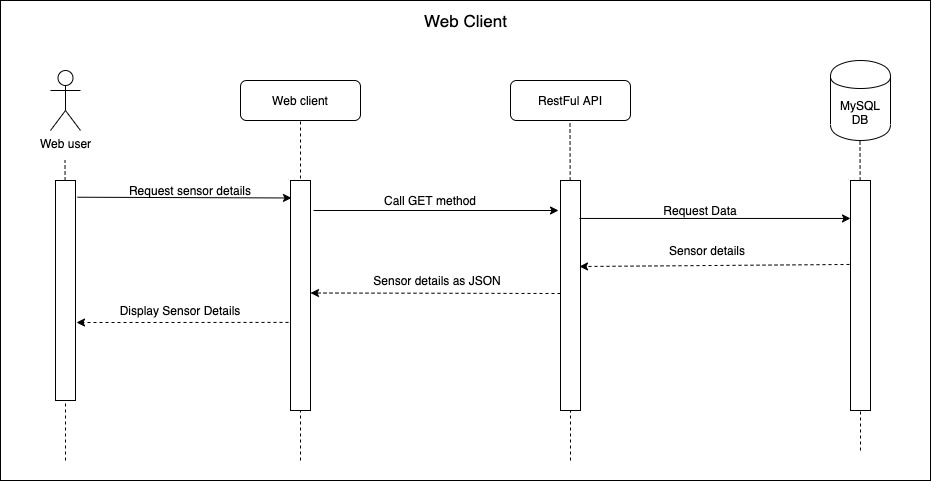
The web client directly accesses the Restful API through it’s GET method to retrieve the details of the sensors. The web client application retrieves data from the API every 40s and updates the page every 40s. The background colour of each sensor details card changes according to the following factors:

1. Green: If the sensor is active and both smoke level and Co2 level is below 5
2. Red: If the sensor is active and either smoke level or Co2 level is equal or greater than to 5 or if both smoke level and co2 level is equal or greater than to 5.
3. Grey: If the sensor is inactive.

Service Interface



Sequence Diagram



## Desktop Client

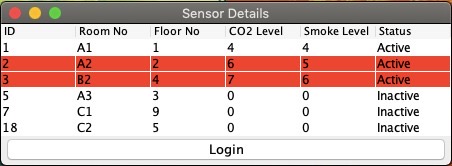
Description

There are two types of users. Both users access the data through the RMI server which communicated with the Restful API.

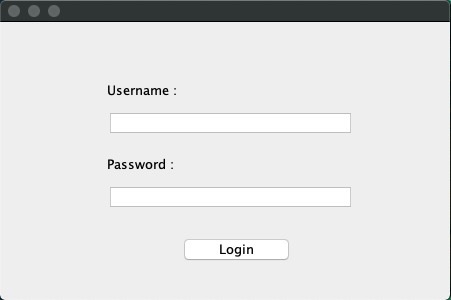
1. Admin
   1. Has to log in to the system
   2. Add a new Sensor
   3. Edit a sensor
   4. Delete a sensor
2. Desktop user

View Sensor details

Whenever the smoke level or the CO2 level is equal or higher than 5, an email service is called through the RMI service

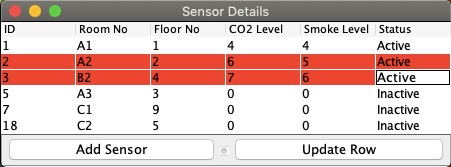


Desktop User – View Sensor Details

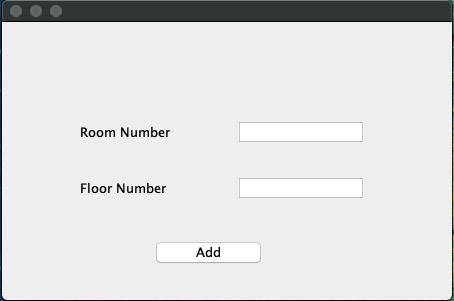


Admin Login



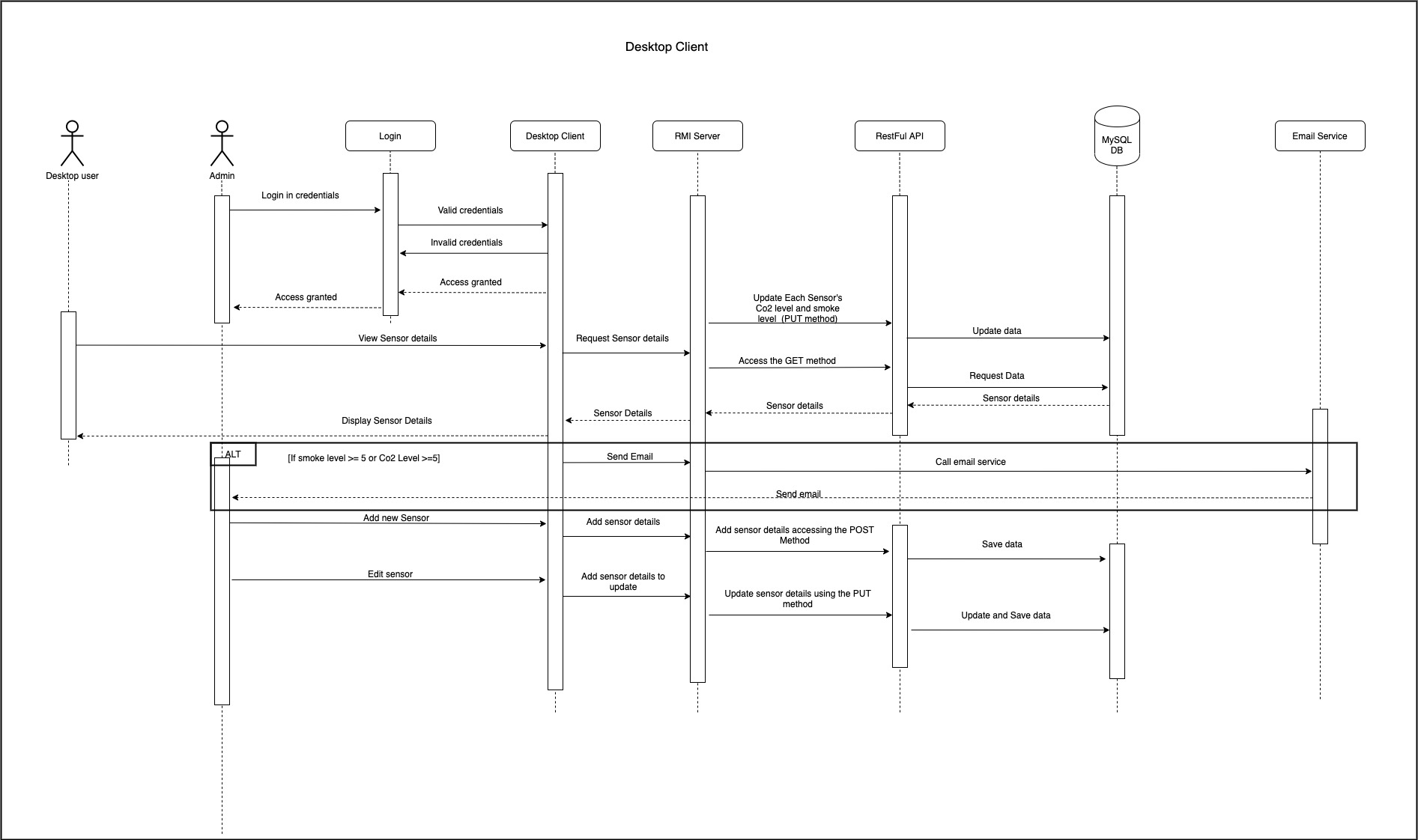


Edit/ Update Sensor Details



Add Sensor

Sequence Diagram



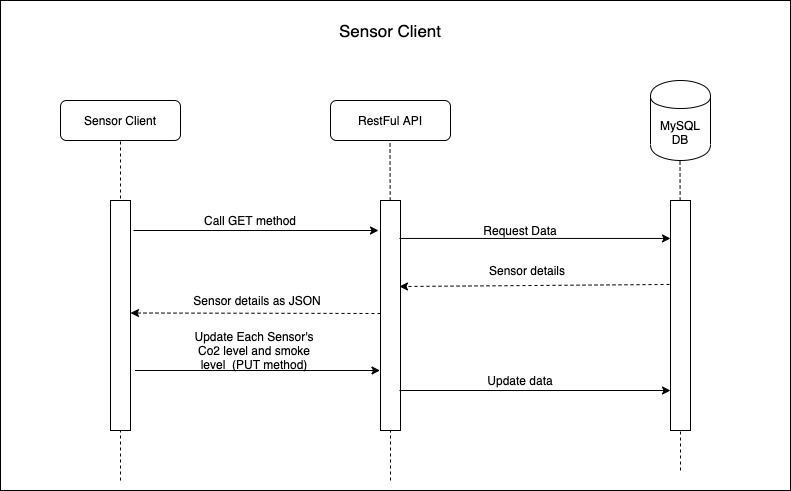
AALTAKTALTALT

## Sensor Client

Description

This is a client that acts as a sensor. This sensor client retrieves data from the restful API and the values of smoke level and co2 level of each sensor is changed randomly. Each sensor is accessed by is ID and each sensor is updated with different values. Each sensor is updated every 10 seconds.

Sequence diagram



## Email service

Called by the RMI Server which gets the request from the Desktop client whenever either Co2 level or smoke level is equal or higher than 5 in any of the sensors.

# Conclusion

This project contains a web client and a desktop client along with an RMI server and a RestAPI which is used to access data from the Database.

In this, the RestAPI is coded with NodeJS and the database is MySQL.

The Web client, which can view the sensor details is coded with ReactJS and is directly connected to the RestAPI.

The Desktop client have two GUI in which only an admin can login to the admin panel by providing the username and password and the other one can be accessed by anyone to view the sensor details.

The desktop client through RMI server also connects to an Email service which sends an email whenever the smoke level and co2 level is higher than normal.

A dummy sensor client is also available in this which is used to update the data through the restAPI every 10s.

# Appendix

## 1. Restful API – NodeJS

1. database.js

'use strict'  
  
var mysql = require('mysql');  
  
var ***dbCon*** = mysql.createConnection({  
 host: 'localhost',  
 user: 'root',  
 password: '',  
 database: 'ds\_api'  
});  
// connect the database  
***dbCon***.connect();  
  
module.exports = ***dbConn***;

1. sensorRoutes.js

var express = require('express');

var bodyParser = require('body-parser');

var mysql = require("mysql");  
var app = express();  
  
  
  
var ***dbConn*** = require('../models/database');  
  
module.exports = function(app) {  
  
 app.use(function(req, res, next) {  
 res.header("Access-Control-Allow-Origin", "\*");  
 res.header("Access-Control-Allow-Headers", "Origin, X-Requested-With, Content-Type, Accept");  
 next();  
 });

//Get all sensors  
 app.get('/api/sensors', function (req, res) {  
  
 ***dbConn***.query('SELECT \* FROM Sensors', function (error, resultOutput, fields) {  
 if (error) throw error;  
 return res.send({error: false, data: resultOutput, message: 'Sensor List.'});  
 });  
 });

//Add sensors  
 app.post('/api/sensors', function (request, res) {  
 let floorNo = request.body.floorNo;  
 let roomNo = request.body.roomNo;  
 let smokeLevel = request.body.smokeLevel;  
 let Co2Level = request.body.Co2Level;  
 let status = request.body.status;  
  
 if (!floorNo || !roomNo || !smokeLevel || !Co2Level || !status) {  
 return res.status(400).send({error: true, message: ‘Enter sensor details Please'});  
 }  
  
 ***dbConn***.query("INSERT INTO Sensors SET ?",{floorNo:floorNo, roomNo:roomNo,smokeLevel:smokeLevel,Co2Level:Co2Level,status:status},function (error, results, fields) {  
 if (error) throw error;  
 return res.send({  
 error: false,  
 data: results,  
 message: 'New sensor has been added to the system successfully.'  
 });  
 });  
 });

//Edit/update sensors – Sensor client  
 app.put('/api/sensors', function (reqst, res) {  
 let sensor\_id = reqst.body.Id;  
 let smokeLevel = reqst.body.smokeLevel;  
 let Co2Level = reqst.body.Co2Level;  
 // let status = reqst.body.status;  
  
 if (!sensor\_id||!smokeLevel || !Co2Level) {  
 return res.status(400).send({ message: 'Please provide sensor details and sensor id' });  
 }  
 ***dbConn***.query("UPDATE Sensors SET smokeLevel = ?,Co2Level = ? WHERE id = ?", [smokeLevel,Co2Level, sensor\_id], function (error, resultOutput, fields) {  
 if (error) throw error;  
 return res.send({ error: false, data: resultOutput, message: 'sensor has been updated successfully.' });  
 });  
 });

//Edit Sensors – Desktop Admin  
app.put('/api/sensors/admin', function (reqst, res) {  
 let sensor\_id = reqst.body.Id;  
 let floorNo = reqst.body.floorNo;  
 let roomNo = reqst.body.roomNo;  
 let status = reqst.body.status;  
  
 if (!sensor\_id || !floorNo || !roomNo || !status) {  
 return res.status(400).send({ message: 'Provide sensor details and sensor id please' });  
 }  
 ***dbConn***.query("UPDATE Sensors SET floorNo = ? ,roomNo = ? ,status =? WHERE id = ?", [floorNo,roomNo,status, sensor\_id], function (error, resultOutput, fields) {  
 if (error) throw error;  
 return res.send({ error: false, data: resultOutput, message: 'sensor has been updated successfully.' });  
 });  
});

//Delete Sensors  
 app.delete('/api/sensors', function (reqst, res) {  
 let sensor\_id = reqst.body.Id;  
  
 if (!sensor\_id) {  
 return res.status(400).send({ error: true, message: 'Select a sensor please' });  
 }  
 ***dbConn***.query('DELETE FROM Sensors WHERE Id = ?', [sensor\_id], function (error, results, fields) {  
 if (error) throw error;  
 return res.send({ error: false, data: results, message: 'Sensor has been deleted successfully.' });  
 });  
 });  
  
}

1. server.js

var express = require('express');

var bodyParser = require('body-parser');

var mysql = require("mysql");  
var ***application*** = express();  
  
  
  
***application***.use(bodyParser.json());  
***application***.use(bodyParser.urlencoded({  
 extended: true  
}));  
// default route  
***application***.get('/', function (req, res) {  
 return res.send({ error: true, message: 'hello' })  
});  
// set port  
***application***.listen(8000, function () {  
 ***console***.log('Node app is running on port 8000');  
});  
var routes = require('./app/routes/sensorRoutes');  
routes(***application***);  
module.exports = ***application***;

## 2. Web Client - ReactJS

App.js

import ***React***,{Component} from 'react';  
import './App.css';  
import ***axios*** from 'axios';  
import {Row,Col, Container, Jumbotron} from "react-bootstrap";  
  
class App extends Component{  
constructor(props) {  
 super(props);  
  
 this.state = {  
 sensors: [],  
 interval : null,  
  
 }  
}

getSensorData() {  
  
 ***axios***.get(`http://localhost:8000/api/sensors`, {  
 headers: {  
 'Content-Type': 'application/json; charset=utf-8'  
 }  
 })  
 .then(res => {  
 const data = res.data;  
 const sensors = data.data;  
  
 this.setState({  
 sensors  
 })  
  
 })  
 .catch((error) => {  
 ***console***.log(error)  
 })  
  
 }  
  
componentDidMount() {  
 this.getSensorData();  
 setInterval(this.getSensorData,40000);  
  
}  
  
componentWillUnmount() {  
 clearInterval(this.interval);  
}

SmokeLevelAndCo2LevelExceeds(){  
 this.state.sensors.map(sensor =>{  
 ***console***.log(sensor.smokeLevel);  
 if(sensor.smokeLevel >= 5 && sensor.Co2Level >=5) {  
 return true;  
 ***console***.log('lol')  
 }  
 else {  
 return false;  
 ***console***.log('aawe naaa');  
 }  
 })  
}  
  
 render() {  
  
 return (  
 <div className="App" style={{backgroundColor:"#465358",

height:"100%",maxWidth:"100%" }}>

//Refresh page every 40s  
 <meta http-equiv="refresh" content="40"/>

<Jumbotron fluid style ={{  
 alignItems: "center",  
 justifyContent: "center",  
 backgroundColor: 'black',  
 flex: 1}} className = "col-50 mx-auto col-mid-30 col-sm-20" >  
 <Container>  
 <center>  
 <h1 style={{color:'#f9eeeb'}}>Sensors List</h1>  
 </center>  
 </Container>  
 </Jumbotron>  
 <div>  
 <ul style={{display:"inline"}}>  
 <li style={{color:' #c9ffcb',

display:"inline", margin:"5rem"}}>\*Active and Everything is under control </li>

<li style={{color:' #ff5148',display:"inline",margin:"5rem"}}> \*Active but either smoke level or Co2 Level is higher </li>  
 <li style={{color:' grey',display:"inline",margin:"5rem"}}>\*Sensor is inactive </li>  
 </ul>  
 <br/>  
 </div>  
 <div style={{overflow:"auto",whiteSpace:"nowrap"}}>  
 <Row>  
 {this.state.sensors.map(sensor =>{  
 if((sensor.smokeLevel>= 5 || sensor.Co2Level >= 5) && sensor.status == "Active" ){  
 return(

<div className='Container text-dark' style={{backgroundColor:'#ff5148',borderRadius:'10px',alignItems:"left",height: "18rem",width:"15rem",display:"inline-block",margin:"5rem"}} >  
 <h3>Sensor ID: {sensor.Id}</h3>  
 <h3>Status: {sensor.status}</h3>  
 <h4>Floor Number: {sensor.floorNo}</h4>  
 <h4>Room Number: {sensor.roomNo}</h4>  
 <h4>Smoke Level: {sensor.smokeLevel}</h4>  
 <h4>Co2 Level : {sensor.Co2Level}</h4>  
  
 </div>  
 )  
 }  
 else if(sensor.status == "Active" ){  
 return(

<div className='Container text-dark' style={{backgroundColor:’#c9ffcb',

borderRadius:'10px',alignItems:"left",height:"18rem",width:"15rem",

display:"inline-block",margin:"5rem"}} >

<h3>Sensor ID: {sensor.Id}</h3>  
 <h3>Status: {sensor.status}</h3>  
 <h4>Floor Number: {sensor.floorNo}</h4>  
 <h4>Room Number: {sensor.roomNo}</h4>  
 <h4>Smoke Level: {sensor.smokeLevel}</h4>  
 <h4>Co2 Level : {sensor.Co2Level}</h4>  
  
 </div>  
 )  
 }  
else{  
 return( <div className='Container text-dark' style={{backgroundColor:'grey',borderRadius:'10px',alignItems:"left",height: "18rem",width:"15rem",display:"inline-block",margin:"5rem"}} >  
 <h3>Sensor ID: {sensor.Id}</h3>  
 <h3>Status: {sensor.status}</h3>  
 <h4>Floor Number: {sensor.floorNo}</h4>  
 <h4>Room Number: {sensor.roomNo}</h4>  
 <h4>Smoke Level: {sensor.smokeLevel}</h4>  
 <h4>Co2 Level : {sensor.Co2Level}</h4>  
 </div>  
 )}  
})}  
</Row>  
</div>  
</div>  
  
 );  
 }  
  
}  
  
export default App;

## 3.Sensor Client – Java

package com.ds.assignment2.SensorClient;

import org.json.\*;

import java.io.BufferedReader;

import java.util.TimerTask;

import java.io.IOException;

import java.util.Timer;

import java.io.InputStreamReader;

import java.util.Random;

import java.io.OutputStream;

import java.net.URL;

import java.net.HttpURLConnection;

import com.google.gson.\*;

public class App{

**public static void main( String[] args ){**

getJsondata();

}

**public static String readRESTAPI()** {

String outputString = null;

try {

URL url = new URL("http://localhost:8000/api/sensors");//your url i.e fetch data from .

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());

}

InputStreamReader in = new InputStreamReader(conn.getInputStream());

BufferedReader br = new BufferedReader(in);

outputString = br.readLine();

System.out.println(outputString);

conn.disconnect();

} catch (Exception e) {

System.out.println(e);

}

return outputString;

}

**public static void getJsondata()** {

String output = readRESTAPI();

JSONObject jsonObject = new JSONObject(output);

JSONArray Jarray = jsonObject.getJSONArray("data");

for (int t = 0; t < Jarray.length(); t++) {

int id = Jarray.getJSONObject(t).getInt("Id");

int floor = Jarray.getJSONObject(t).getInt("floorNo");

String room = Jarray.getJSONObject(t).getString("roomNo");

int smoke = Jarray.getJSONObject(t).getInt("smokeLevel");

int co2 = Jarray.getJSONObject(t).getInt("Co2Level");

Random random = new Random();

smoke = random.nextInt(10 - 1 + 1) + 1;

co2 = random.nextInt(10 - 1 + 1) + 1;

UpdateSensor(id, smoke, co2);

}

Timer Responsetimer = new Timer();

Responsetimer.schedule(new TimerTask() {

*@Override*

**public void run()** {

//call the getJondata method

getJsondata();

}

}, 10000, 10000);

}

**public static void UpdateSensor(int id,int smokeLevel, int Co2Level)** {

String UPDATE\_PARAMS = "{\n" + "\"Id\":"+id+",\r\n" +

" \"smokeLevel\":"+smokeLevel+",\r\n" +

" \"Co2Level\":"+Co2Level+"\r\n}";

try {

URL APIurl = new URL("http://localhost:8000/api/sensors");

HttpURLConnection UrlConn = (HttpURLConnection) APIurl.openConnection();

UrlConn setRequestProperty("Accept", "application/json");

UrlConn.setRequestMethod("PUT");

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

UrlConn.setDoOutput(true);

UrlConn.setDoInput(true);

OutputStream outputStream = UrlConn.getOutputStream();

outputStream.write(UPDATE\_PARAMS.getBytes());

outputStream.flush();

outputStream.close();

int responseCode = UrlConn.getResponseCode();

System.out.println("Update Response Code : " + responseCode);

System.out.println("Update Response Message : " + UrlConn.getResponseMessage());

if (responseCode == HttpURLConnection.HTTP\_CREATED) {

//success

BufferedReader in = new BufferedReader(new InputStreamReader(

UrlConn.getInputStream()));

String inputLine;

StringBuffer response = new StringBuffer();

while ((inputLine = in .readLine()) != null) {

response.append(inputLine);

} in .close();

// print result

System.out.println(response.toString());

}

} catch (IOException e) {

e.printStackTrace();

}

}

}

## 4.Desktop Client

1. RMI Server

import java.io.BufferedReader;

import com.twilio.type.PhoneNumber;

import java.io.IOException;

import com.twilio.Twilio;

import java.io.InputStreamReader;

import javax.mail.internet.MimeMessage;

import java.io.OutputStream;

import javax.mail.internet.InternetAddress;

import java.net.HttpURLConnection;

import javax.mail.Transport;

import java.net.URL;

import javax.mail.Session;

import java.rmi.RemoteException;

import javax.mail.PasswordAuthentication;

import java.rmi.server.UnicastRemoteObject;

import javax.mail.MessagingException;

import java.util.ArrayList;

import javax.mail.Message;

import java.util.HashMap;

import javax.mail.Authenticator;

import java.util.Properties;

public class ReadRESTAPI extends UnicastRemoteObject implements ServerService{

public static String *reciever* = "[manujayapremathilaka@gmail.com](mailto:manujayapremathilaka@gmail.com)";

public static String *PHONE\_NUMBER* = "+940717795556";

public static String *mymail* = "[test.purpose.lanka@gmail.com](mailto:test.purpose.lanka@gmail.com)";

public static String *host* = "smtp.gmail.com";

public static String *ACCOUNT\_SID* = "ACe3f5d5a6da41f1668e6293d885be39da";

public static String *AUTH\_TOKEN* = "af97f2aebcf1268dfad57b918ef2ac9d";

protected ReadRESTAPI() throws RemoteException {

super();

}

//getting all the details from the API as a JSON object

public String readRESTAPI() {

String output = null;

try {

URL APIurl = new URL("http://localhost:8000/api/sensors");

HttpURLConnection UrlConn = (HttpURLConnection) APIurl.openConnection();

UrlConn.setRequestMethod("GET");

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

//handling the error

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

throw new RuntimeException("Failed : HTTP Error code : "+ UrlConn.getResponseCode());

}

//reading the response

InputStreamReader inStream = new InputStreamReader(UrlConn.getInputStream());

BufferedReader bufferR = new BufferedReader(inStream);

output = bufferR.readLine();

UrlConn.disconnect();

} catch (Exception e) {

System.***out***.println("Exception in NetClientGet:- " + e);

}

return output;

}

public void addSensor(int floorNo, String roomNo) {

//adding the values to a JSON object

String POST\_PARAMS = "{\n" + "\"floorNo\":"+"\""+floorNo+"\",\r\n" +

" \"roomNo\":"+"\""+roomNo+"\",\r\n" +

" \"smokeLevel\":"+"\""+0+"\",\r\n" +

" \"Co2Level\":"+"\""+0+"\",\r\n" +

" \"status\": \"Inactive\"" + "\n}";

try {

URL APIurl = new URL("http://localhost:8000/api/sensors"); //url to post the values

HttpURLConnection Urlconn = (HttpURLConnection) APIurl.openConnection();

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

Urlconn setRequestMethod("POST");

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

Urlconn.setDoOutput(true);

//sending the values as a JSON object

OutputStream outputStream = Urlconn.getOutputStream();

outputStream.write(POST\_PARAMS.getBytes());

outputStream.flush();

outputStream.close();

//printing the response code and response message

int responseCode = Urlconn.getResponseCode();

System.***out***.println("POST Response Code : " + responseCode);

System.***out***.println("POST Response Message : " + Urlconn.getResponseMessage());

} catch (IOException e) {

e.printStackTrace();

}

}

*@Override*

public void editSensor(HashMap<String, String> row) throws RemoteException {

//adding the values to a JSON object

String PUT\_PARAMS = "{\n" + "\"Id\":"+"\""+row.get("id")+"\",\r\n" +

" \"roomNo\":"+"\""+row.get("roomNo")+"\",\r\n" +

" \"floorNo\":"+"\""+row.get("floorNo")+"\",\r\n" +

" \"status\":"+"\""+ row.get("status") +"\"\n}";

try {

URL APIurl = new URL("http://localhost:8000/api/sensors/desktopPut"); //url to put the values

HttpURLConnection Urlconn = (HttpURLConnection) APIurl.openConnection();

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

Urlconn.setRequestMethod("PUT");

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

Urlconn.setDoOutput(true);

//sending the value as a JSON object

OutputStream outputStream = Urlconn.getOutputStream();

outputStream.write(PUT\_PARAMS.getBytes());

outputStream.flush();

outputStream.close();

//printing the response code and the message

int responseCode = Urlconn.getResponseCode();

System.***out***.println("POST Response Code : " + responseCode);

System.***out***.println("POST Response Message : " + Urlconn.getResponseMessage());

} catch (IOException e) {

e.printStackTrace();

}

}

public void sendAlert(String messageBody) {

System.***out***.println(messageBody);

sendMail(*reciever*, messageBody);

}

//sending alert email

public void sendMail(String reciever, String messageBody) {

//system props

Properties props = System.*getProperties*();

// Set mail-server

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

props.put("mail.smtp.port", "465");

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

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

// creating session’s object to get properties

Session sesion = Session.*getDefaultInstance*(properties, new Authenticator() {

protected PasswordAuthentication getPasswordAuthentication() {

return new [PasswordAuthentication("test.purpose.lanka@gmail.com](mailto:PasswordAuthentication(%22test.purpose.lanka@gmail.com)", "Test\_purpose123");

}

});

sesion.setDebug(true);

try {

// Message object.

MimeMessage Mmessage = new MimeMessage(sesion);

//Adding sender email.

Mmessage.setFrom(new InternetAddress(*mymail*));

//Adding reciever's email.

Mmessage.addRecipient(Message.RecipientType.***TO***, new InternetAddress(reciever));

//Subject of the email

Mmessage.setSubject("Sensor Alert");

//Body of the email.

Mmessage.setText(messageBody);

//COnfirmation MEssage.

Transport.*send*(Mmessage);

System.***out***.println("Mail successfully sent");

}

catch (MessagingException mex) {

mex.printStackTrace();

}

}

}

1. Desktop Client

import java.net.MalformedURLException;

import org.json.JSONObject;

import java.rmi.Naming;

import java.rmi.RemoteException;

import java.util.ArrayList;

import org.json.JSONArray;

import java.rmi.NotBoundException;

import java.util.HashMap;

import com.model.FloorDetails;

public class ClientMain {

ServerService serverService;

public static ArrayList<FloorDetails> *initialList*;

/\*\*

\* get all the sensor details

\* **@return** floorDetails

\*/

public ArrayList<FloorDetails> getSensors(){

ArrayList<FloorDetails> floorDetails = new ArrayList<FloorDetails>();

try {

serverService = (ServerService) Naming.*lookup*("rmi://localhost/RMIServer");

String output = serverService.readRESTAPI();

JSONObject jsonObject = new JSONObject(output);

JSONArray Jarray= jsonObject.getJSONArray("data");

for(int t = 0; t < Jarray.length(); t++) {

FloorDetails details = new FloorDetails();

details.setId(array.getJSONObject(t).getInt("Id"));

details.setStatus(array.getJSONObject(t).getString("status"));

details.setFloorNo(array.getJSONObject(t).getInt("floorNo"));

details.setRoomNo(array.getJSONObject(t).getString("roomNo"));

details.setSmokeLevel(array.getJSONObject(t).getInt("smokeLevel"));

details.setCo2Level(array.getJSONObject(t).getInt("Co2Level"));

floorDetails.add(details);

}

} catch (Exception e) {

e.printStackTrace();

}

return floorDetails;

}

//sending room number & floor number to rmi server to add a new sensor

public void AddSensor(String roomNo, String floorNumber) {

try {

serverService = (ServerService) Naming.*lookup*("rmi://localhost/RMIServer");

serverService.addSensor(Integer.*parseInt*(floorNumber), roomNo);

} catch (MalformedURLException | RemoteException | NotBoundException e) {

e.printStackTrace();

}

}

//sending new values to edit sensor details

public void EditSensor(HashMap<String, String> row) {

try {

serverService = (ServerService) Naming.*lookup*("rmi://localhost/RMIServer");

serverService.editSensor(row);

} catch (MalformedURLException | RemoteException | NotBoundException e) {

e.printStackTrace();

}

}

//sending initial alert message body to rmi server

public void sendInitialAlert(ArrayList<FloorDetails> alertList) {

String alertMessage = "";

*initialList* = alertList;

try {

serverService = (ServerService) Naming.*lookup*("rmi://localhost/RMIServer");

if(*initialList*.size() > 0) {

for(int t = 0; t < *initialList*.size(); t++) {

alertMessage = alertMessage + "FloorNo: " + *initialList*.get(t).getFloorNo() + "\nRoomNo: " + *initialList*.get(t).getRoomNo() +

"\nCO2Level: " + *initialList*.get(t).getCo2Level() + "\nSmokeLevel: " + *initialList*.get(t).getSmokeLevel() + "\n\n";

}

serverService.sendAlert(alertMessage);

}

} catch (MalformedURLException | RemoteException | NotBoundException e) {

e.printStackTrace();

}

}

/\*\*

\* Sending alert message if new sensor fired

\* maintain the list of rooms with higher CO2 and smoke levels

\*/

public void sendNewAlert() {

//getting new sensor details

ArrayList<FloorDetails> list = getSensors();

String alertMessage = "";

for(int t = 0; t < list.size(); t++) {

int id = list.get(t).getId();

//finding the rooms with higher CO2 and Smoke levels

if((list.get(t).getCo2Level() >= 5) || (list.get(t).getSmokeLevel() >= 5)) {

//if not found in the previous list, creating the allert message body and adding to the list of higher smoke and CO2 level rooms

if(!*initialList*.stream().filter(o -> o.getId() == id).findFirst().isPresent()) {

alertMessage = alertMessage + "FloorNo: " + list.get(t).getFloorNo() + "\nRoomNo: " + list.get(t).getRoomNo() + "\nCO2Level: " + list.get(t).getCo2Level() + "\nSmokeLevel: " + list.get(t).getSmokeLevel() + "\n\n";

*initialList*.add(list.get(t));

}

}

else {

//after fixing the room, removing from the list of containing higher gas level rooms

if(*initialList*.stream().filter(o -> o.getId() == id).findFirst().isPresent()) {

int j = 0;

while(*initialList*.get(j).getId() != id) {

j++;

}

*initialList*.remove(j);

}

}

}

if(alertMessage.length() > 0) {

try {

serverService = (ServerService) Naming.*lookup*("rmi://localhost/RMIServer");

serverService.sendAlert(alertMessage);

} catch (MalformedURLException | RemoteException | NotBoundException e) {

e.printStackTrace();

}

}

}

}

1. ClientGUI

import javax.swing.JFrame;

import java.awt.BorderLayout;

import javax.swing.table.TableModel;

import java.awt.Color;

import javax.swing.table.TableCellRenderer;

import java.awt.Component;

import javax.swing.table.DefaultTableModel;

import java.awt.EventQueue;

import javax.swing.table.DefaultTableCellRenderer;

import java.awt.event.ActionEvent;

import javax.swing.WindowConstants;

import java.awt.event.ActionListener;

import javax.swing.Timer;

import java.util.ArrayList;

import javax.swing.JTable;

import javax.swing.JButton;

import com.model.FloorDetails;

public class ClientGUI {

public static JFrame *frameJ*;

private ClientMain clientMain = new ClientMain();

private JTable table;

private Timer timer;

ArrayList<FloorDetails> initialAlertList;

/\*\*

\* Launch the application.

\*/

public static void main(String[] args) {

EventQueue.*invokeLater*(new Runnable() {

public void run() {

try {

ClientGUI window = new ClientGUI();

window.*frameJ*.setVisible(true);

} catch (Exception e) {

e.printStackTrace();

}

}

});

}

/\*\*

\* Create the application.

\*/

public ClientGUI() {

initialize();

}

/\*\*

\* Initialize the contents of the frame.

\*/

private void initialize() {

*frameJ* = new JFrame();

*frameJ*.setDefaultCloseOperation(JFrame.***EXIT\_ON\_CLOSE***);

//setting the table

table = executeTable();

*frameJ*.getContentPane().add(table.getTableHeader(),

BorderLayout.***PAGE\_START***);

*frameJ*.getContentPane().add(table, BorderLayout.***CENTER***);

//action on login button push

JButton btnLogin = new JButton("Login");

btnLogin.addActionListener(new ActionListener() {

public void actionPerformed(ActionEvent e) {

timer.stop();

LoginGUI.*Login*();

}

});

*frameJ*.getContentPane().add(btnLogin, BorderLayout.***SOUTH***);

*frameJ*.setDefaultCloseOperation(WindowConstants.***EXIT\_ON\_CLOSE***);

*frameJ*.pack();

*frameJ*.setLocationRelativeTo(null);

*frameJ*.setTitle("Sensor Details");

refreshTimer();

}

//creating the table

public JTable executeTable( ) {

initialAlertList = new ArrayList<FloorDetails>();

//table columns

Object[] columns = new String[] {

"ID", "Room No", "Floor No", "CO2 Level", "Smoke Level", "Status"

};

//getting table values by calling getSensors() method

ArrayList<FloorDetails> arrayList = clientMain.getSensors();

Object[][] dataRow = new Object[arrayList.size()][6];

//setting the values to 2D array

for(int t = 0; t < arrayList.size(); t++) {

dataRow [t][0] = arrayList.get(t).getId();

dataRow [t][1] = arrayList.get(t).getRoomNo();

dataRow [t][2] = arrayList.get(t).getFloorNo();

dataRow [t][3] = arrayList.get(t).getCo2Level();

dataRow [t][4] = arrayList.get(t).getSmokeLevel();

dataRow [t][5] = arrayList.get(t).getStatus();

if((arrayList.get(t).getCo2Level() >= 5) || (arrayList.get(t).getSmokeLevel() >= 5)) {

initialAlertList.add(arrayList.get(t));

}

}

//sending alert message body

clientMain.sendInitialAlert(initialAlertList);

//creating the table

table = new JTable(dataRow,columns) {

*@Override*

public boolean isCellEditable(int row, int column) {

return false;

}

*@Override*

public Component prepareRenderer(TableCellRenderer renderer, int row, int col) {

Component component = super.prepareRenderer(renderer, raw, col);

int co2Level = (int) getValueAt(raw, 3);

int smokeLevel = (int) getValueAt(raw, 4);

/\*\*

\* checking the co2 and smoke level

\* if greater than 5, mark the row in red colour

\*/

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

component.setBackground(Color.***RED***);

component.setForeground(Color.***BLACK***);

} else {

component.setBackground(super.getBackground());

component.setForeground(super.getForeground());

}

return component;

}

};

*frameJ*.setTitle("Sensor Details");

return table;

}

//timer function to refresh in every 5 seconds

public void refreshTimer() {

timer = new Timer(0, new ActionListener() {

*@Override*

public void actionPerformed(ActionEvent e) {

ArrayList<FloorDetails> newAlertList = new ArrayList<FloorDetails>();

Object[] columns = new String[] {

"ID", "Room No", "Floor No", "CO2 Level", "Smoke Level", "Status"

};

ArrayList<FloorDetails> arrayList = clientMain.getSensors();

Object[][] dataRow = new Object[arrayList.size()][6];

for(int t = 0; t < arrayList.size(); t++) {

dataRow [t][0] = arrayList.get(t).getId();

dataRow [t][1] = arrayList.get(t).getRoomNo();

dataRow [t][2] = arrayList.get(t).getFloorNo();

dataRow [t][3] = arrayList.get(t).getCo2Level();

dataRow [t][4] = arrayList.get(t).getSmokeLevel();

dataRow [t][5] = arrayList.get(t).getStatus();

}

clientMain.sendNewAlert();

TableModel tableModel = new DefaultTableModel(dataRow, columns);

table.setDefaultRenderer(Object.class, (TableCellRenderer) new DefaultTableCellRenderer(){

*@Override*

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

super.getTableCellRendererComponent(table, value, isSelected, hasFocus, row, col);

return this;

}

});

table.removeAll();

table.setModel(tableModel);

System.***out***.println("REFRESHED");

}});

timer.setDelay(15000); // delay for 15 seconds

timer.start();

}

}

1. AdminGUI

import java.awt.BorderLayout;

import javax.swing.JSplitPane;

import java.awt.Color;

import javax.swing.JButton;

import java.awt.event.ActionListener;

import javax.swing.table.TableModel;

import javax.swing.table.DefaultTableCellRenderer;

import java.util.ArrayList;

import javax.swing.WindowConstants;

import java.util.HashMap;

import com.model.FloorDetails;

import javax.swing.JFrame;

import javax.swing.Timer;

import javax.swing.JTable;

import java.awt.event.ActionEvent;

import javax.swing.table.TableCellRenderer;

import java.awt.Component;

import javax.swing.table.DefaultTableModel;

import java.awt.EventQueue;

public class AdminGUI extends JFrame{

private JFrame frameJ;

private ClientMain clientMain = new ClientMain();

private JTable table;

/\*\*

\* Launch the application.

\*/

public static void AdminGUI() {

EventQueue.*invokeLater*(new Runnable() {

public void run() {

try {

AdminGUI window = new AdminGUI();

window. frameJ.setVisible(true);

} catch (Exception e) {

e.printStackTrace();

}

}

});

}

/\*\*

\* Create the application.

\*/

public AdminGUI() {

initialize();

}

/\*\*

\* Initialize the contents of the frame.

\*/

private void initialize() {

frameJ = new JFrame();

frameJ.setDefaultCloseOperation(JFrame.***EXIT\_ON\_CLOSE***);

//executing the table

table = executeTable();

frameJ.getContentPane().add(table.getTableHeader(), BorderLayout.***PAGE\_START***);

frameJ.getContentPane().add(table, BorderLayout.***CENTER***);

JSplitPane splitPane = new JSplitPane();

splitPane.setResizeWeight(0.5);

frameJ.getContentPane().add(splitPane, BorderLayout.***SOUTH***);

//action on add sensor button pushed

JButton btnAddSensor = new JButton("Add Sensor");

btnAddSensor.addActionListener(new ActionListener() {

public void actionPerformed(ActionEvent e) {

AddSensorGUI.*AddSensor*();

}

});

splitPane.setLeftComponent(btnAddSensor);

//action on update button pushed

JButton btnUpdateRow = new JButton("Update Row");

btnUpdateRow.addActionListener(new ActionListener() {

public void actionPerformed(ActionEvent e) {

int selectedRowIndex = table.getSelectedRow();

HashMap<String, String> row = new HashMap<String, String>();

row.put("id", table.getModel().getValueAt(selectedRowIndex, 0).toString());

row.put("roomNo", table.getModel().getValueAt(selectedRowIndex, 1).toString());

row.put("floorNo", table.getModel().getValueAt(selectedRowIndex, 2).toString());

row.put("co2Level", table.getModel().getValueAt(selectedRowIndex, 3).toString());

row.put("smokeLevel", table.getModel().getValueAt(selectedRowIndex, 4).toString());

row.put("status", table.getModel().getValueAt(selectedRowIndex, 5).toString());

//passing the values

clientMain.EditSensor(row);

}

});

splitPane.setRightComponent(btnUpdateRow);

frameJ.setDefaultCloseOperation(WindowConstants.***EXIT\_ON\_CLOSE***);

frameJ.pack();

frameJ.setLocationRelativeTo(null);

frameJ.setTitle("Sensor Details");

//calling the refresh timer to refresh the table

refreshTimer();

}

public JTable executeTable( ) {

//table columns

Object[] columns = new String[] {

"ID", "Room No", "Floor No", "CO2 Level", "Smoke Level", "Status"

};

//getting values for the table

ArrayList<FloorDetails> arrayList = clientMain.getSensors();

Object[][] dataRow = new Object[arrayList.size()][6];

//assigning values to the 2D array

for(int t = 0; t < arrayList.size(); t++) {

dataRow [t][0] = arrayList.get(t).getId();

dataRow [t][1] = arrayList.get(t).getRoomNo();

dataRow [t][2] = arrayList.get(t).getFloorNo();

dataRow [t][3] = arrayList.get(t).getCo2Level();

dataRow [t][4] = arrayList.get(t).getSmokeLevel();

dataRow [t][5] = arrayList.get(t).getStatus();

}

table = new JTable(dataRow,columns) {

private static final int ***CO2\_COL*** = 3;

///defining the editable cells

*@Override*

public boolean isCellEditable(int row, int colmn) {

return colmn == 0 || colmn == 1 || colmn == 2 || colmn == 5 ? true : false;

}

/\*\*

\* checking the co2 and smoke level

\* if greater than 5, mark the row in red colour

\*/

*@Override*

public Component prepareRenderer(TableCellRenderer renderer, int row, int col) {

Component compnent = super.prepareRenderer(renderer, row, col);

int co2Level = (int) getValueAt(row, ***CO2\_COL***);

int smokeLevel = (int) getValueAt(row, 4);

/\*\*

\* checking the co2 and smoke level

\* if greater than 5, mark the row in red colour

\*/

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

compnent.setBackground(Color.***RED***);

compnent.setForeground(Color.***BLACK***);

} else {

compnent.setBackground(super.getBackground());

compnent.setForeground(super.getForeground());

}

return compnent;

}

};

frameJ.setTitle("Sensor Details");

return table;

}

//timer function to refresh in every 5 seconds

public void refreshTimer() {

Timer timer = new Timer(0, new ActionListener() {

*@Override*

public void actionPerformed(ActionEvent e) {

Object[] columns = new String[] {

"ID", "Room No", "Floor No", "CO2 Level", "Smoke Level", "Status"

};

ArrayList<FloorDetails> arrayList = clientMain.getSensors();

Object[][] dataRow = new Object[arrayList.size()][6];

for(int t = 0; t < arrayList.size(); t++) {

dataRow [t][0] = arrayList.get(t).getId();

dataRow [t][1] = arrayList.get(t).getRoomNo();

dataRow [t][2] = arrayList.get(t).getFloorNo();

dataRow [t][3] = arrayList.get(t).getCo2Level();

dataRow [t][4] = arrayList.get(t).getSmokeLevel();

dataRow [t][5] = arrayList.get(t).getStatus();

}

clientMain.sendNewAlert();

TableModel tableModel = new DefaultTableModel(dataRow, columns);

table.setDefaultRenderer(Object.class, (TableCellRenderer) new DefaultTableCellRenderer(){

*@Override*

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

super.getTableCellRendererComponent(table, value, isSelected, hasFocus, row, col);

return this;

}

});

table.removeAll();

table.setModel(tableModel);

System.***out***.println("REFRESHED");

}

});

timer.setDelay(15000); // delay for 15 seconds

timer.start();

}

}

1. LoginGUI

import java.awt.EventQueue;

import java.awt.event.ActionEvent;

import javax.swing.JFrame;

import javax.swing.JOptionPane;

import java.awt.event.ActionListener;

import javax.swing.JTextField;

import java.awt.BorderLayout;

import javax.swing.JButton;

import javax.swing.JLabel;

public class LoginGUI {

private JFrame frameJ;

private JTextField textUserName;

private JTextField textPassword;

/\*\*

\* Start the app.

\*/

public static void Login() {

EventQueue.*invokeLater*(new Runnable() {

public void run() {

try {

LoginGUI LogWindow = new LoginGUI();

LogWindow.frameJ.setVisible(true);

} catch (Exception e) {

e.printStackTrace();

}

}

});

}

/\*\*

\* Create the application.

\*/

public LoginGUI() {

initialize();

}

/\*\*

\* Initialize the contents of the frame.

\*/

private void initialize() {

frameJ = new JFrame();

frameJ.setBounds(100, 100, 450, 300);

frameJ.setDefaultCloseOperation(JFrame.***DISPOSE\_ON\_CLOSE***);

frameJ.getContentPane().setLayout(null);

textUserName = new JTextField();

textUserName.setBounds(107, 88, 247, 26);

frameJ.getContentPane().add(textUserName);

textUserName.setColumns(10);

textPassword = new JTextField();

textPassword.setColumns(10);

textPassword.setBounds(107, 162, 247, 26);

frameJ.getContentPane().add(textPassword);

JLabel lblNew = new JLabel("Username :");

lblNew.setBounds(107, 60, 75, 16);

frameJ.getContentPane().add(lblNew);

JLabel lblNew \_1 = new JLabel("Password :");

lblNew \_1.setBounds(107, 134, 75, 16);

frameJ.getContentPane().add(lblNew \_1);

//action on login button pushed

JButton btnLogin = new JButton("Login");

btnLogin.addActionListener(new ActionListener() {

public void actionPerformed(ActionEvent e) {

String txtusername = textUserName.getText();

String txtpassword = textPassword.getText();

if(txtusername.equals("admin") && txtpassword.equals("admin")) {

AdminGUI.*AdminGUI*();

frameJ.dispose();

ClientGUI.frameJ.dispose();

} else {

JOptionPane.*showMessageDialog*(null, "Invalid Credentials");

textPassword.setText("");

textUserName.setText("");

}

}

});

btnLogin.setBounds(178, 214, 117, 29);

frameJ.getContentPane().add(btnLogin);

}

}

1. AddSensorGUI

import java.awt.EventQueue;

import javax.swing.GroupLayout.Alignment;

import javax.swing.JFrame;

import javax.swing.JFormattedTextField;

import javax.swing.GroupLayout;

import java.awt.BorderLayout;

import javax.swing.JTextField;

import java.awt.event.ActionEvent;

import javax.swing.JLabel;

import javax.swing.BoxLayout;

import javax.swing.JButton;

import java.awt.event.ActionListener;

import java.awt.GridLayout;

public class AddSensorGUI {

private JFrame frameJ;

private JTextField roomNo;

private JTextField floorNo;

private ClientMain clientMain = new ClientMain();

/\*\*

\* Launch the application.

\*/

public static void AddSensor() {

EventQueue.*invokeLater*(new Runnable() {

public void run() {

try {

AddSensorGUI Addwindow = new AddSensorGUI();

Addwindow.frameJ.setVisible(true);

} catch (Exception e) {

e.printStackTrace();

}

}

});

}

/\*\*

\* Create the application.

\*/

public AddSensorGUI() {

initialize();

}

/\*\*

\* Initialize the contents of the frame.

\*/

private void initialize() {

frameJ = new JFrame();

frameJ.setBounds(100, 100, 450, 300);

frameJ.setDefaultCloseOperation(JFrame.***DISPOSE\_ON\_CLOSE***);

roomNo = new JTextField();

roomNo.setColumns(10);

floorNo = new JTextField();

floorNo.setColumns(10);

JLabel lblNew = new JLabel("Floor Number");

JLabel lblNew \_1 = new JLabel("Room Number");

//action on add sensor button pushed

JButton btnAddSensor = new JButton("Add");

btnAddSensor.addActionListener(new ActionListener() {

public void actionPerformed(ActionEvent e) {

clientMain.AddSensor(roomNo.getText(), floorNo.getText());

frameJ.dispose();

}

});

GroupLayout groupLayout = new GroupLayout(frameJ.getContentPane());

groupLayout.setHorizontalGroup(

groupLayout.createParallelGroup(*Alignment*.***LEADING***)

.addGroup(groupLayout.createSequentialGroup()

.addGap(78)

.addComponent(lblNew\_1, GroupLayout.***DEFAULT\_SIZE***, GroupLayout.***DEFAULT\_SIZE***, Short.***MAX\_VALUE***)

.addGap(66)

.addComponent(roomNo, GroupLayout.***DEFAULT\_SIZE***, 130, Short.***MAX\_VALUE***)

.addGap(86))

.addGroup(groupLayout.createSequentialGroup()

.addGap(78)

.addComponent(lblNew, GroupLayout.***DEFAULT\_SIZE***, 90, Short.***MAX\_VALUE***)

.addGap(66)

.addComponent(floorNo)

.addGap(86))

.addGroup(groupLayout.createSequentialGroup()

.addGap(148)

.addComponent(btnAddSensor, GroupLayout.***DEFAULT\_SIZE***, 117, Short.***MAX\_VALUE***)

.addGap(185))

);

groupLayout.setVerticalGroup(

groupLayout.createParallelGroup(*Alignment*.***LEADING***)

.addGroup(groupLayout.createSequentialGroup()

.addGap(97)

.addGroup(groupLayout.createParallelGroup(*Alignment*.***LEADING***)

.addGroup(groupLayout.createSequentialGroup()

.addGap(5)

.addComponent(lblNew\_1, GroupLayout.***DEFAULT\_SIZE***, GroupLayout.***DEFAULT\_SIZE***, Short.***MAX\_VALUE***)

.addGap(5))

.addComponent(roomNo))

.addGap(30)

.addGroup(groupLayout.createParallelGroup(*Alignment*.***LEADING***)

.addGroup(groupLayout.createSequentialGroup()

.addGap(5)

.addComponent(lblNew, GroupLayout.***DEFAULT\_SIZE***, GroupLayout.***DEFAULT\_SIZE***, Short.***MAX\_VALUE***)

.addGap(5))

.addComponent(floorNo))

.addGap(38)

.addComponent(btnAddSensor, GroupLayout.***PREFERRED\_SIZE***, 29, GroupLayout.***PREFERRED\_SIZE***)

.addGap(32))

);

frameJ.getContentPane().setLayout(groupLayout);

}

}