**Fire Alarm Monitoring System Report**

****

**Group Members:**

* **IT18154504** Diwantha K.S.
* **IT18145526** V.T. Konthasinghe
* **IT18143614** Kavindi Gunasinghe U.L.D
* **IT18137910** Wasana H.M.A.H

**High Level Architectural Diagram of the system**

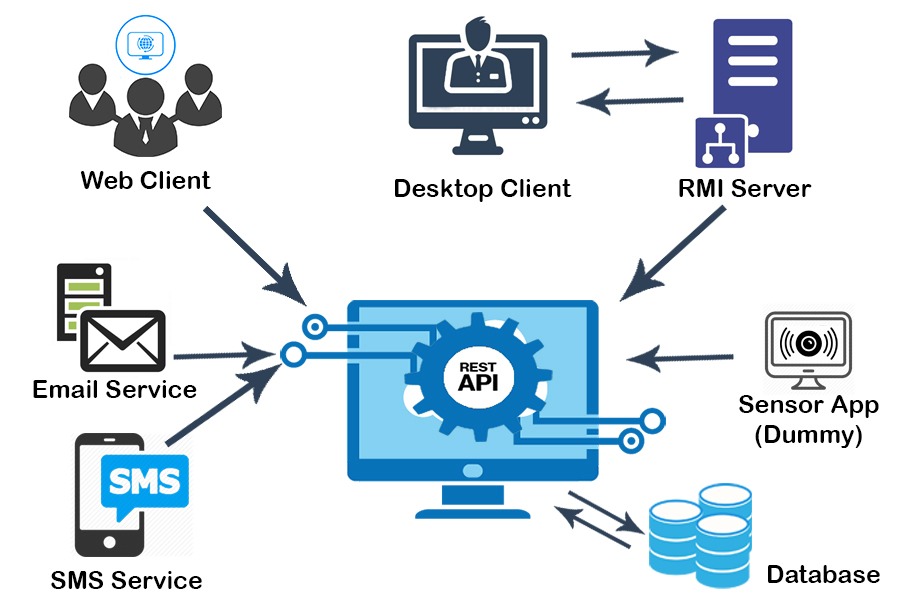
****

Fig 1.0

**Web Application**

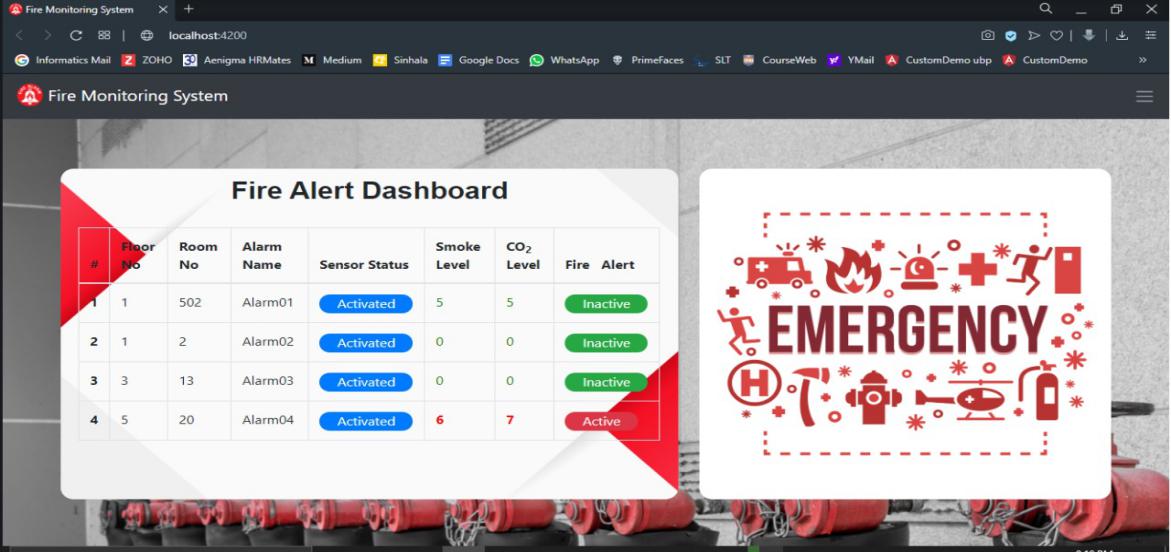


Fig 1.1

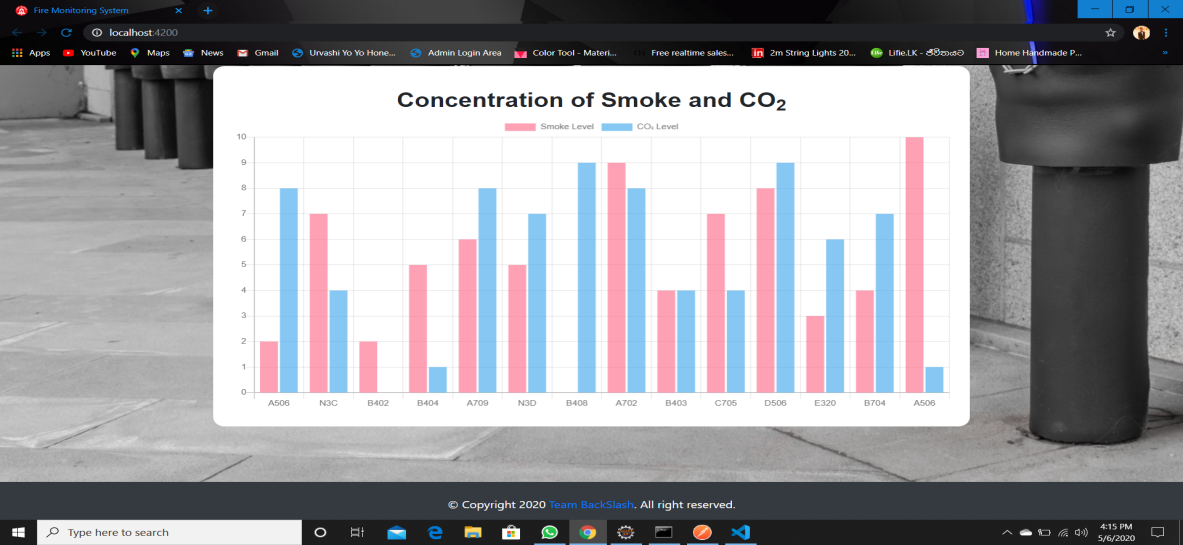
****

Fig 1.2

Figures 1.1 and 1.2 depict the interfaces a user gets once the user lands on the web application. Figure 1.1 depicts the active sensors while figure 1.2 shows the concentrations of smoke and C02 levels.

**Desktop Application**

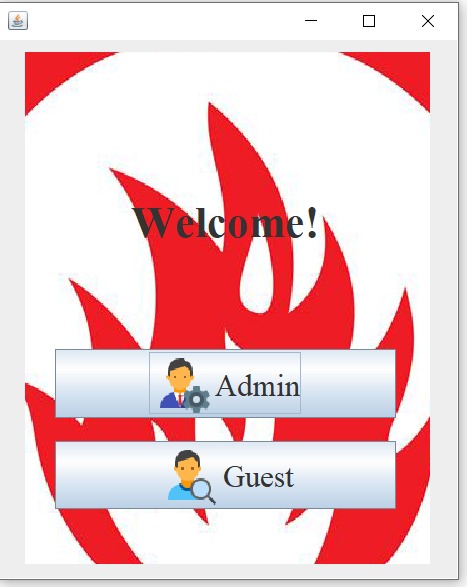


Fig 2.1

Fig 2.1 depicts foremost interface a user gets when the user starts using the desktop application.

***Admin***

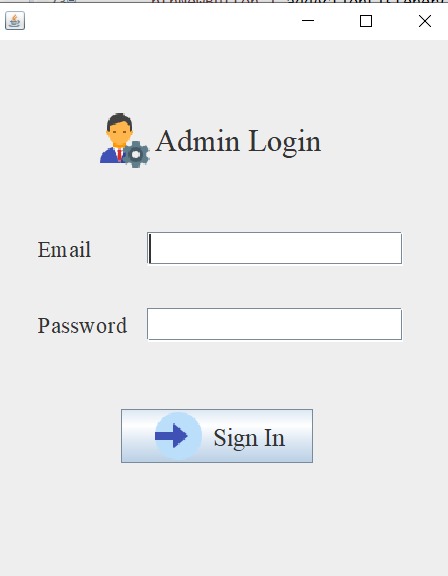
******

Fig 2.2

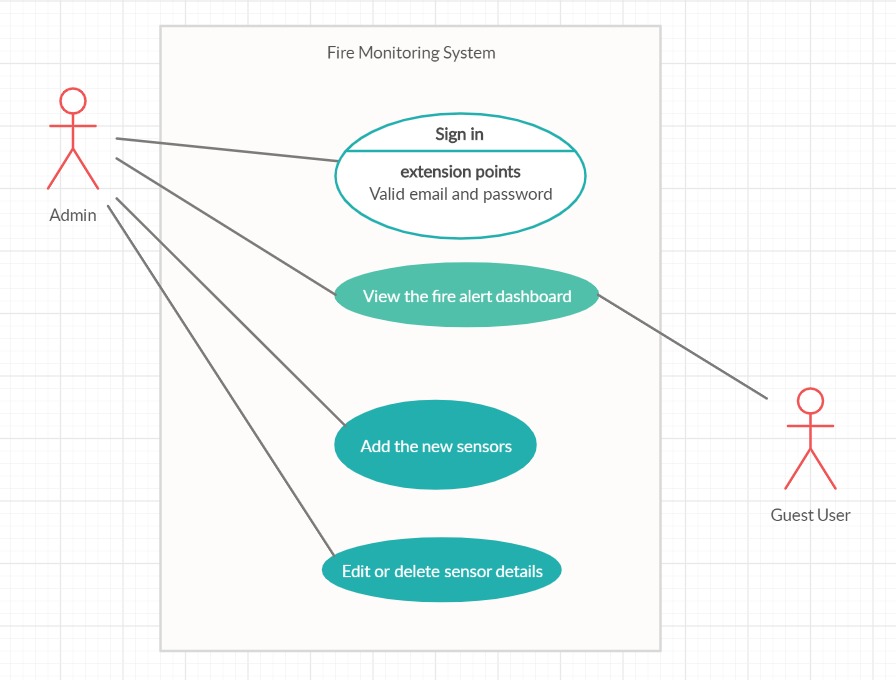


Fig 2.3

Figure 2.3 depicts the usecase diagram describing the Admin and Guest logins directing to the functionalities they can access after a successful login.

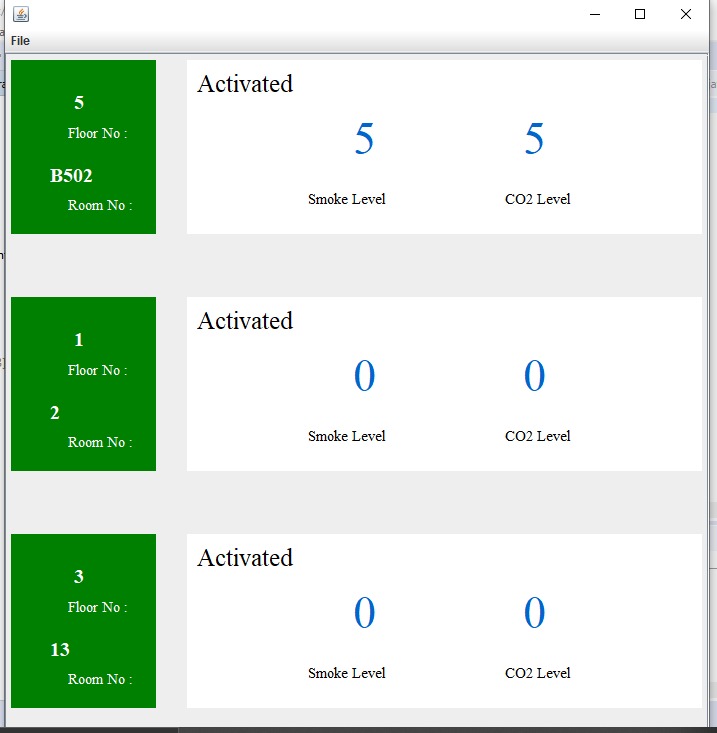


Fig 2.4

Figure 2.4 depicts the dashboard showing the active sensors once the admin logs in. (Note that this dashboard is the only interface the user gets when logged in as a guest as well)

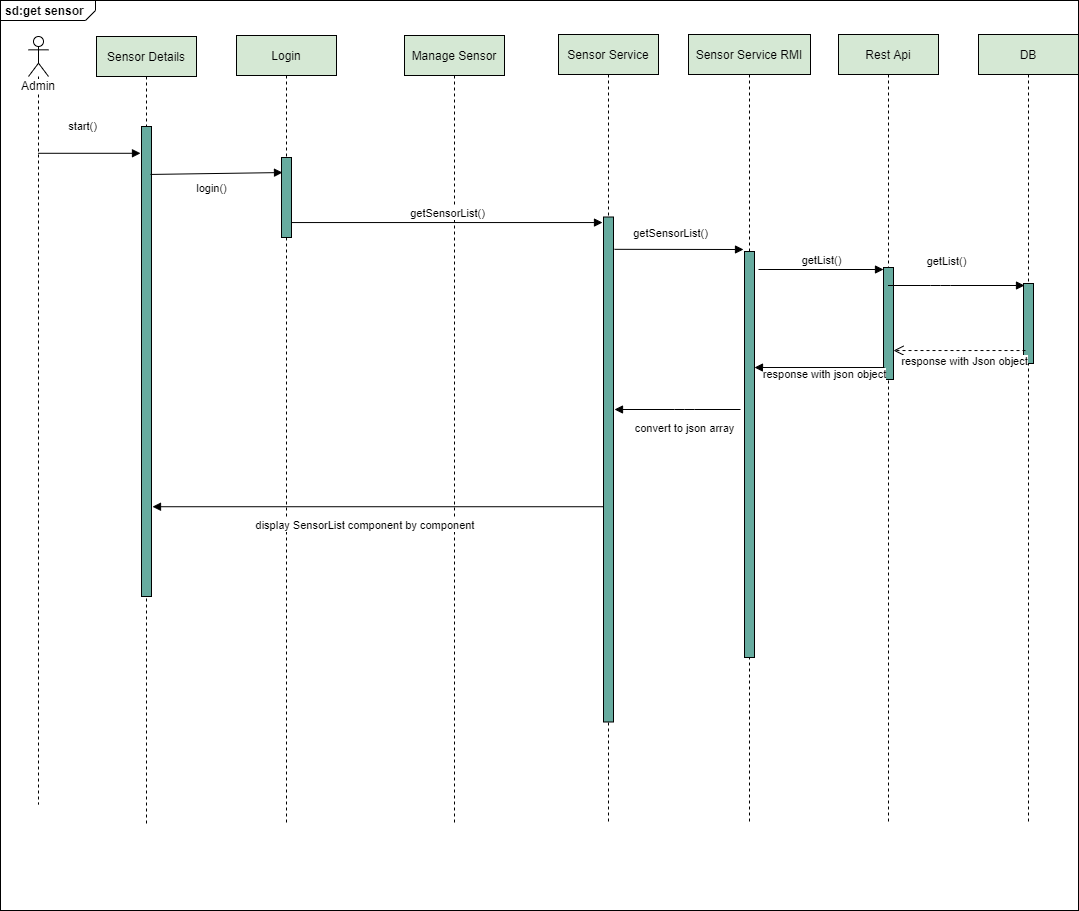


Fig 2.5

Figure 2.5 shows the sequence diagram describing the services rendered when displaying the list of active sensors.

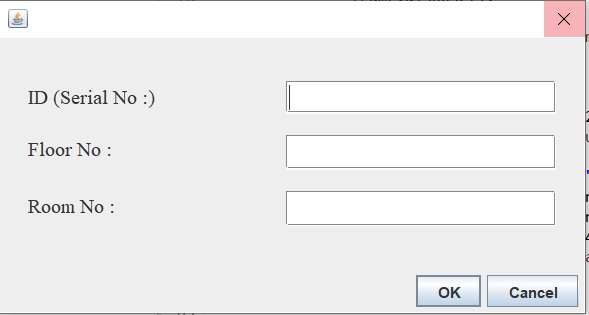


Fig 2.6

Figure 2.6 shows the interface the admin gets when adding a new sensor.

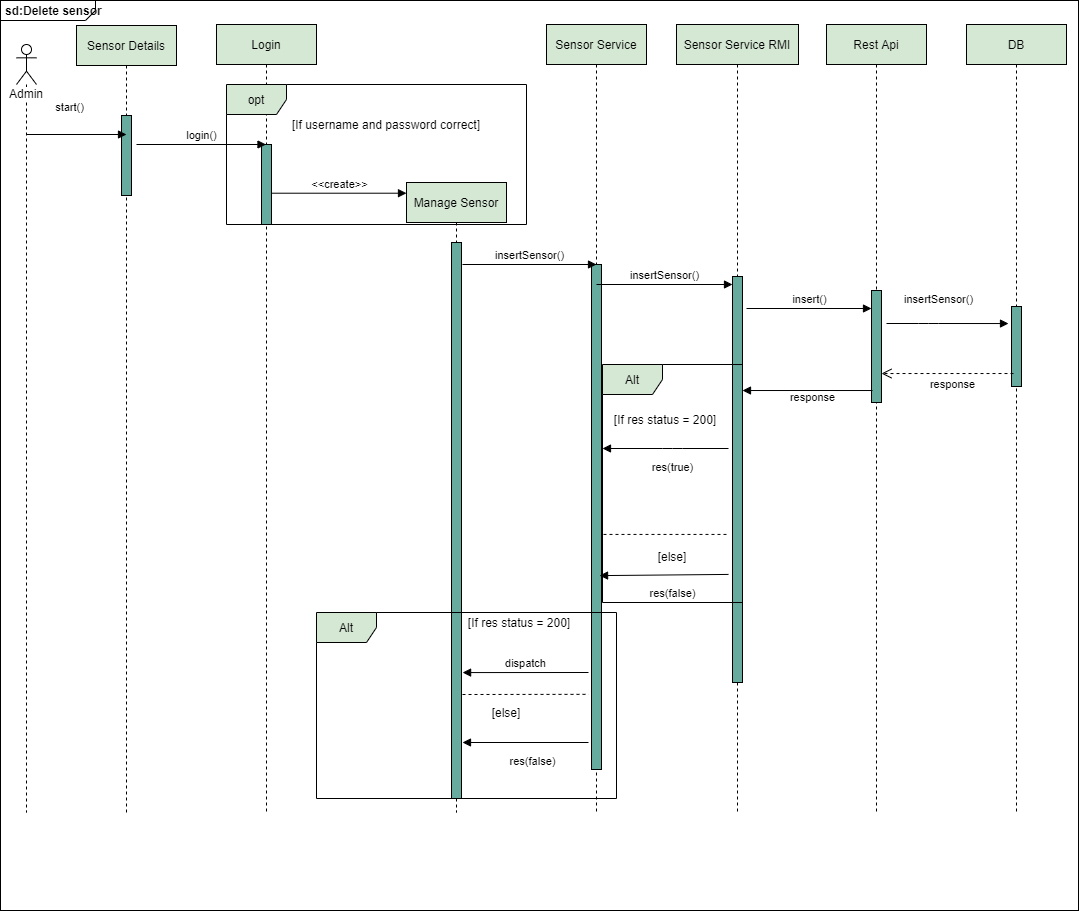


Fig 2.7

Figure 2.7 shows the sequence diagram describing the services rendered when adding a new sensor.

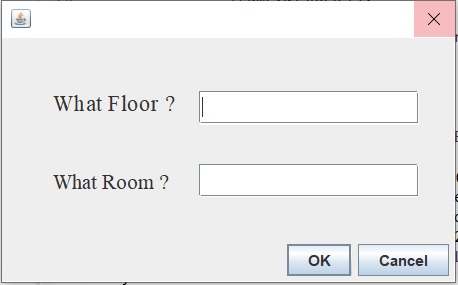


Fig 2.8

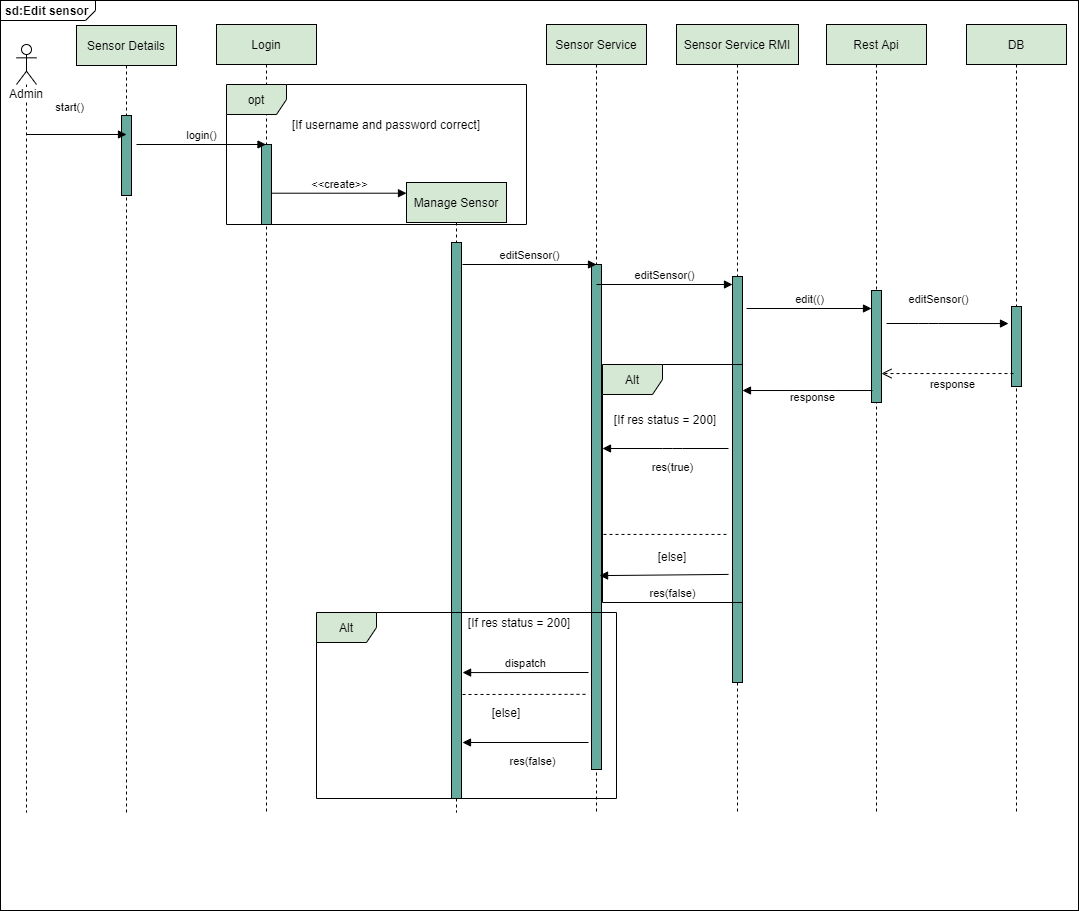
Figure 2.8 indicates the interface user gets when editing a particular sensor.

Fig 2.9

Figure 2.9 indicates the sequence diagram describing the services rendered when editing a particular sensor.

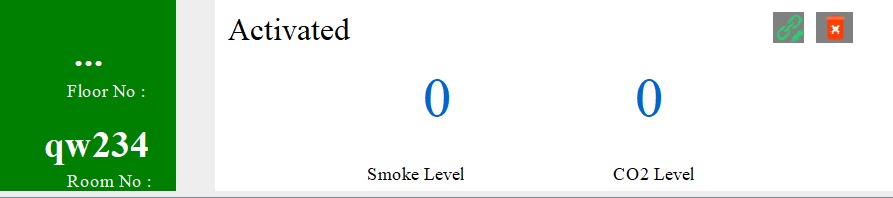


Fig 2.10

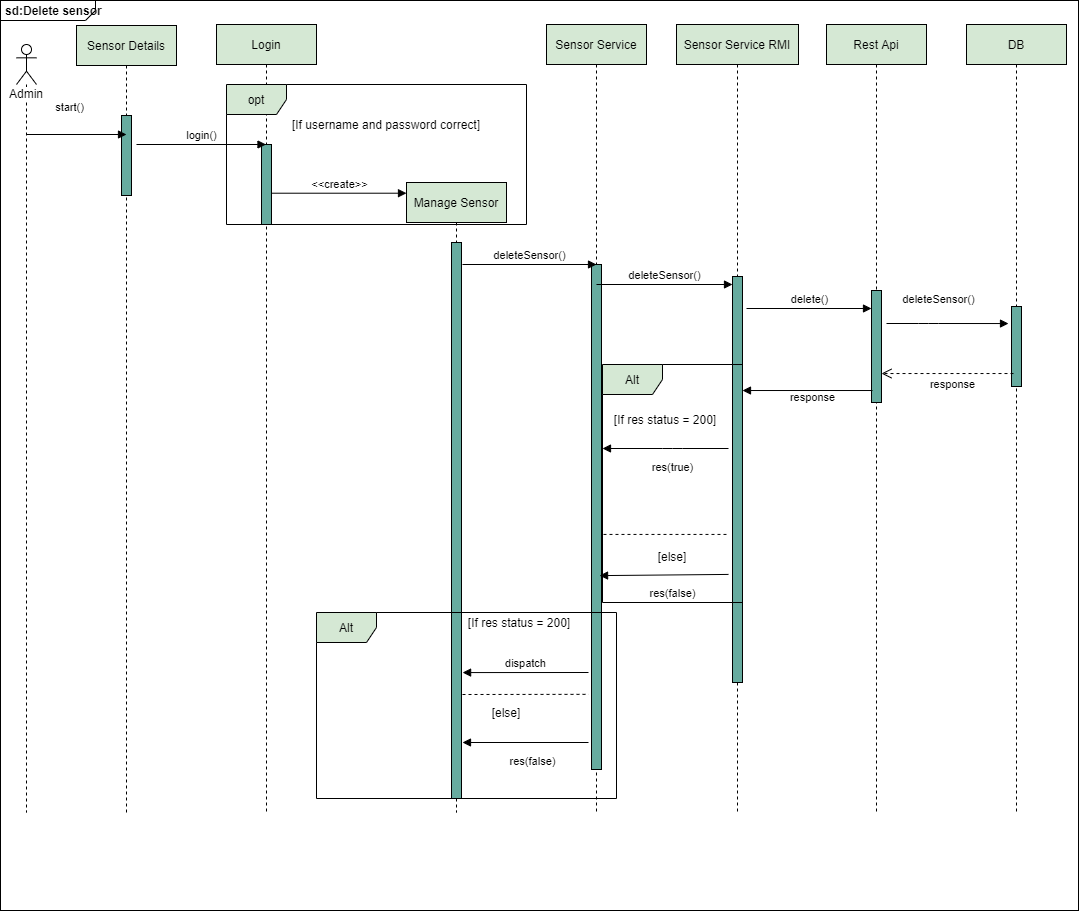
Figure 2.10 shows the interface to delete a particular sensor.

Fig 2.11

Figure 2.11 is the sequence diagram to delete a particular sensor.

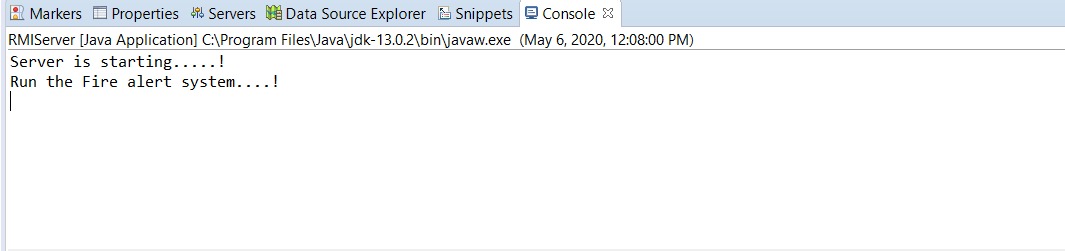
***Other diagrams***

Fig 3.1(running server)

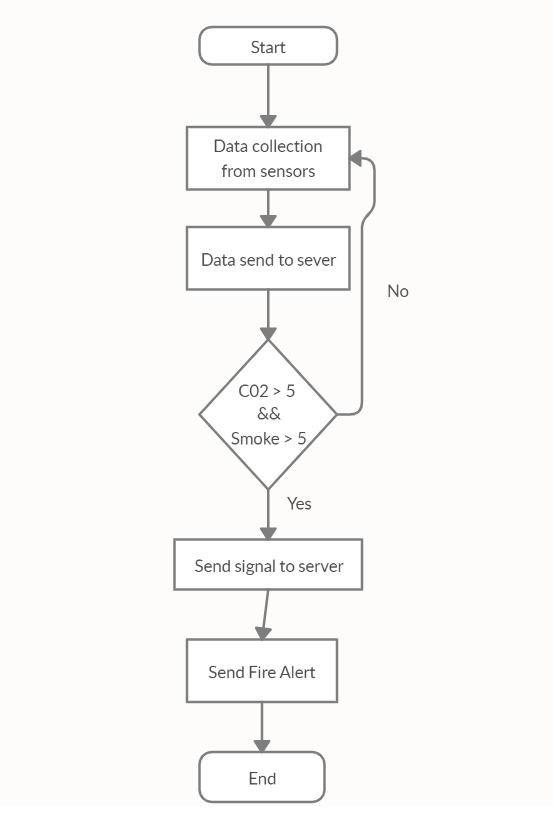


Fig 3.2

Figure 3.2 depicts flow chart to represent determine the smoke and CO2 levels of the environment.

**Appendix**

**Web Application(Angular)**

*Sensor chart –*

*Sensor-chart.component.html*

<div>  
 <div style="display: block">  
 <canvas baseChart  
 [datasets]="barChartData"  
 [labels]="barChartLabels"  
 [options]="barChartOptions"  
 [legend]="barChartLegend"  
 [chartType]="barChartType"></canvas>  
 </div>  
 </div>

Sensor-chart.component.ts

import { Component, OnInit } from '@angular/core';  
import { Chart } from 'chart.js';  
  
@Component({  
 selector: 'app-sensor-chart',  
 templateUrl: './sensor-chart.component.html',  
 styleUrls: ['./sensor-chart.component.css']  
})  
export class SensorChartComponent implements OnInit {  
  
 public barChartOptions = {  
 scaleShowVerticalLines: false,  
 responsive: true  
 };  
  
 public barChartLabels = ['A506', 'N3C', 'B402', 'B404', 'A709', 'N3D', 'B408', 'A702', 'B403', 'C705', 'D506', 'E320', 'B704', 'A506'];  
 public barChartType = 'bar';  
 public barChartLegend = true;  
  
 public barChartData = [  
 {data: [2, 7, 2, 5, 6, 5, 0, 9, 4, 7, 8, 3, 4, 10], label: 'Smoke Level'},  
 {data: [8, 4, 0, 1, 8, 7, 9, 8, 4, 4, 9, 6, 7, 1], label: 'CO₂ Level'}  
 ];  
  
 constructor() { }  
  
 ngOnInit(): void {  
  
 }}

*sensor –chart-data.service.ts*

import { Injectable } from '@angular/core';  
  
  
@Injectable({  
 providedIn: 'root'  
})  
export class SensorChartDataService {  
  
 constructor() { }  
  
}

*app.component.css*

h2{  
 padding-top: 10px;  
 text-align: center;  
 font-weight: 700;  
 font-family: sans-serif;  
}  
  
  
  
.sensor-body{  
 display: flex;  
 margin: 5%;  
 height: 70vh;  
 width: auto;  
   
}  
  
.sensor-details-table{  
 width: 60%;  
 height: 70vh;  
 border-radius: 15px;   
 background-image: url(../assets/images/comp.jpg);  
 background-size: cover;  
   
  
}  
.sensor-details-img{  
 width: 40%;  
 height: 70vh;  
 margin-left: 2%;  
 border-radius: 15px;  
 background-image: url(../assets/images/emergency.jpg);  
 background-position: center;  
 background-size: cover;  
   
   
}  
  
.sensor-details-graph{  
   
 width: 80%;  
 height: 70vh;  
 margin-left: auto;  
 margin-right: auto;  
   
}  
  
.sensor-details-graph-inner{  
  
width: inherit;  
height: inherit;  
display: table;  
margin: 0 auto;  
border-radius: 15px;  
background-color: white;  
padding: 2%;  
  
  
}  
  
.table-body{  
 margin: 20px;  
 padding-top: 10px;  
 height: 80%;  
 overflow: auto;  
  
   
}  
  
.status-p {  
 color: #fff;  
 padding: 0px 20px 1px;  
 border-radius: 20px;  
 display: inline-block;  
 text-transform: capitalize;  
 vertical-align: middle;  
   
}  
  
.footer-area {  
 margin-top: 10%;  
 text-align: center;  
 padding: 23px 0 19px;  
   
}  
  
.footer-area p {  
 color: #fff;  
 margin-bottom: 0;  
}  
  
::-webkit-scrollbar{  
 width: 10px;  
 }  
 ::-webkit-scrollbar-thumb{  
 background: #f7283d;  
 border-radius: 5px;  
 }  
   
 ::-webkit-scrollbar-thumb:hover{  
 background: #f41226;  
   
 }

*app.component.html*

<nav class="navbar navbar-dark bg-dark">  
 <a class="navbar-brand mb-1" href="#">  
 <img src="../assets/images/ic\_firealarm.png" width="30" height="30" class="d-inline-block align-top" alt="">  
 Fire Monitoring System  
 </a>  
 <span class="navbar-toggler-icon"></span>  
</nav>  
  
 <div class="sensor-body">  
 <div class="sensor-details-table">  
  
 <h2>Fire Alert Dashboard</h2>  
 <div class="table-body">  
 <table class="table table-bordered">  
 <thead>  
 <tr>  
 <th scope="col">#</th>  
 <th scope="col">Floor No</th>  
 <th scope="col">Room No</th>  
 <th scope="col">Alarm Name</th>  
 <th scope="col">Sensor Status</th>  
 <th scope="col">Smoke Level</th>  
 <th scope="col">CO<sub>2</sub> Level</th>  
 <th scope="col">Fire &nbsp; Alert</th>  
 </tr>  
 </thead>  
 <tbody>  
 <tr *\*ngFor*='let ***sensorDetails*** of sensors$'>  
 <th scope="row">{{***sensorDetails***.id}}</th>  
 <td>{{***sensorDetails***.floorNo}}</td>  
 <td>{{***sensorDetails***.roomNo}}</td>  
 <td>{{***sensorDetails***.name}}</td>  
 <td><span class="status-p bg-primary">Activated</span></td>  
 <td *\*ngIf*="***sensorDetails***.smokeLevel > 5; else ***smokeLevelLessThan***" style="color:red; font-weight: bold;">{{***sensorDetails***.smokeLevel}}</td>  
 <ng-template #***smokeLevelLessThan***>  
 <td style="color:green;">{{***sensorDetails***.smokeLevel}}</td>  
 </ng-template>  
 <td *\*ngIf*="***sensorDetails***.co2Level > 5; else ***co2LevelLessThan***" style="color:red; font-weight: bold;">{{***sensorDetails***.co2Level}}</td>  
 <ng-template #***co2LevelLessThan***>  
 <td style="color:green;">{{***sensorDetails***.co2Level}}</td>  
 </ng-template>  
 <td *\*ngIf*="***sensorDetails***.smokeLevel > 5 || ***sensorDetails***.co2Level > 5; else ***lessThan***"><span class="status-p bg-danger">Active</span></td>  
 <ng-template #***lessThan***>  
 <td><span class="status-p bg-success">Inactive</span></td>  
 </ng-template>  
 </tr>   
 </tbody>  
 </table>  
 </div>  
 </div>  
 <div class="sensor-details-img"></div>  
 </div>  
  
 <div class="sensor-details-graph">  
   
 <div class="sensor-details-graph-inner">  
 <h2>Concentration of Smoke and CO<sub>2</sub></h2>  
 <app-sensor-chart>  
  
 </app-sensor-chart>  
 </div>  
 </div>  
  
   
 <footer>  
 <div class="footer-area bg-dark">  
 <p>© Copyright 2020 <a href="">Team BackSlash</a>. All right reserved.</p>  
   
 </div>  
</footer>

*app.component.ts*

import { Component, OnInit } from '@angular/core';  
import { Sensor } from './sensor.model';  
import { SensorDataService } from './sensor-data.service';  
  
  
  
@Component({  
 selector: 'app-root',  
 templateUrl: './app.component.html',  
 styleUrls: ['./app.component.css']  
})  
export class AppComponent implements OnInit {  
   
   
 sensors$: Sensor[];  
   
 constructor(private sensorService: SensorDataService) {}  
  
 ngOnInit() {  
  
 setInterval(()=>{  
 return this.sensorService.getSensorDetails()  
 .subscribe(data => this.sensors$ = data);  
 },10000);  
 }  
  
}

*app.module.ts*

import { BrowserModule } from '@angular/platform-browser';  
import { NgModule } from '@angular/core';  
  
import { AppRoutingModule } from './app-routing.module';  
import { AppComponent } from './app.component';  
import { HttpClientModule } from '@angular/common/http';  
import { ChartsModule } from 'ng2-charts';  
import { SensorChartComponent } from './sensor-chart/sensor-chart.component';  
  
  
@NgModule({  
 declarations: [  
 AppComponent,  
 SensorChartComponent  
 ],  
 imports: [  
 BrowserModule,  
 AppRoutingModule,  
 HttpClientModule,  
 ChartsModule  
 ],  
 providers: [],  
 bootstrap: [AppComponent]  
})  
export class AppModule { }

*sensor.model.ts*

export class Sensor {  
  
 id: number;  
 floorNo: string;  
 roomNo: string;  
 name: string;  
 status: string;  
 smokeLevel: number;  
 co2Level: number;  
  
  
}

*sensor-data.service.ts*

import { Injectable } from '@angular/core';  
import { HttpClient } from '@angular/common/http';  
import { Sensor } from './sensor.model';  
  
  
@Injectable({  
 providedIn: 'root'  
})  
export class SensorDataService {  
  
 apiUrl = "http://localhost:8080/alarms/findAll";  
  
 constructor(private http:HttpClient) { }  
  
 getSensorDetails() {  
   
 return this.http.get<Sensor[]>(this.apiUrl);  
   
 }  
}

*API s*

*AlarmController.java*

package com.example.controllers;  
  
import com.example.entities.Alarm;  
import com.example.service.AlarmService;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.http.MediaType;  
import org.springframework.http.ResponseEntity;  
import org.springframework.web.bind.annotation.\*;  
  
import java.util.ArrayList;  
  
@RestController  
@RequestMapping(*value* = "/alarms")  
@CrossOrigin  
public class AlarmController {  
  
 @Autowired  
 private AlarmService alarmService;  
  
 @PostMapping(*value* = "/create",*consumes* = MediaType.APPLICATION\_JSON\_VALUE)  
 public String createAlarm(@RequestBody Alarm a){  
  
 try {  
 alarmService.createAlarm(a);  
 } catch (Exception e) {  
 e.printStackTrace();  
 }  
  
 String successMessage = "Alarm Created !";  
  
 return successMessage;  
  
 }  
  
 @GetMapping(*value* = "/findById/{id}")  
 public ResponseEntity<?> findAlarmById(@PathVariable("id")int id){  
  
 Alarm alarm = *null*;  
 try {  
 alarm = alarmService.findByAlarmId(id);  
 } catch (Exception e) {  
 e.printStackTrace();  
 }  
  
 return ResponseEntity.ok(alarm);  
  
 }  
  
 @GetMapping(*value* = "/findAll")  
 public ResponseEntity<?> findAll(){  
 ArrayList<Alarm> array = new ArrayList<>();  
 try {  
 array = alarmService.findAllAlarms();  
 } catch (Exception e) {  
 e.printStackTrace();  
 }  
  
 return ResponseEntity.ok(array);  
 }  
  
}

*Alarm.java*

package com.example.entities;  
  
import javax.persistence.\*;  
  
@Entity  
@Table(*name* = "Alarms")  
public class Alarm {  
 @Id  
 @GeneratedValue(*strategy* = GenerationType.IDENTITY) // Auto generate ID  
 private int id;  
 private String name;  
 private int roomNo;  
 private int floorNo;  
 private int smokeLevel;  
 private int co2Level;  
 private String status;  
  
 public Alarm() {  
 }  
  
 public Alarm(String name, int roomNo, int floorNo, int smokeLevel, int co2Level, String status) {  
 this.name = name;  
 this.roomNo = roomNo;  
 this.floorNo = floorNo;  
 this.smokeLevel = smokeLevel;  
 this.co2Level = co2Level;  
 this.status = status;  
 }  
  
 @Override  
 public String toString() {  
 return "Alarm{" +  
 "id=" + id +  
 ", name='" + name + '*\'*' +  
 ", roomNo=" + roomNo +  
 ", floorNo=" + floorNo +  
 ", smokeLevel=" + smokeLevel +  
 ", co2Level=" + co2Level +  
 ", status='" + status + '*\'*' +  
 '}';  
 }  
  
 public int getId() {  
 return id;  
 }  
  
 public void setId(int id) {  
 this.id = id;  
 }  
  
 public String getName() {  
 return name;  
 }  
  
 public void setName(String name) {  
 this.name = name;  
 }  
  
 public int getRoomNo() {  
 return roomNo;  
 }  
  
 public void setRoomNo(int roomNo) {  
 this.roomNo = roomNo;  
 }  
  
 public int getFloorNo() {  
 return floorNo;  
 }  
  
 public void setFloorNo(int floorNo) {  
 this.floorNo = floorNo;  
 }  
  
 public int getSmokeLevel() {  
 return smokeLevel;  
 }  
  
 public void setSmokeLevel(int smokeLevel) {  
 this.smokeLevel = smokeLevel;  
 }  
  
 public int getCo2Level() {  
 return co2Level;  
 }  
  
 public void setCo2Level(int co2Level) {  
 this.co2Level = co2Level;  
 }  
  
 public String getStatus() {  
 return status;  
 }  
  
 public void setStatus(String status) {  
 this.status = status;  
 }  
}

*AlarmRepository.java*

package com.example.repository;  
  
  
import com.example.entities.Alarm;  
import org.springframework.data.jpa.repository.JpaRepository;  
  
import java.util.ArrayList;  
  
public interface AlarmRepository extends JpaRepository<Alarm,Integer> {  
  
 Alarm findById(int id);  
 ArrayList<Alarm> findAll();  
  
}

*Application.java*

package com.example;  
  
import org.springframework.boot.SpringApplication;  
import org.springframework.boot.autoconfigure.SpringBootApplication;  
  
@SpringBootApplication  
public class Application {  
  
 public static void main(String[] args) {  
 SpringApplication.run(Application.class, args);  
 }  
  
}

*application.properties*

spring.jpa.hibernate.ddl-auto=update  
spring.datasource.url=jdbc:mysql://localhost:3306/example?createDatabaseIfNotExist=true  
spring.datasource.username=root  
spring.datasource.password=1234  
spring.jpa.show-sql=true  
spring.jpa.hibernate.dialec=org.hibernate.dialect.MySQL55Dialect

*RMI Service*

package rmi\_server\_codes;  
  
import java.rmi.Remote;  
import java.rmi.RemoteException;  
  
public interface RMIService extends Remote {  
  
 /\*  
 \* methods that are exposed to desktop client  
 \* \*/  
   
 public String getAllSensorDetails() throws RemoteException;  
  
 public String loginValidator(String email, String password) throws RemoteException;  
  
 public boolean addSensor(String id, int floor, String room) throws RemoteException;  
  
 public boolean editSensor(String id, int floor, String room) throws RemoteException;  
  
 public boolean deleteSensor(String id) throws RemoteException;  
  
}

*RMI Server*

package rmi\_server\_codes;  
  
import java.awt.event.ActionEvent;  
import java.awt.event.ActionListener;  
import java.io.IOException;  
import java.net.URI;  
import java.net.http.HttpClient;  
import java.net.http.HttpRequest;  
import java.net.http.HttpResponse;  
import java.rmi.AlreadyBoundException;  
import java.rmi.RemoteException;  
import java.rmi.registry.LocateRegistry;  
import java.rmi.registry.Registry;  
import java.rmi.server.UnicastRemoteObject;  
  
import javax.swing.Timer;  
  
import org.apache.http.client.methods.HttpDelete;  
import org.apache.http.client.methods.HttpPatch;  
import org.apache.http.client.methods.HttpPost;  
import org.apache.http.entity.StringEntity;  
import org.apache.http.impl.client.CloseableHttpClient;  
import org.apache.http.impl.client.HttpClientBuilder;  
import org.json.JSONArray;  
import org.json.JSONObject;  
  
import desktop\_forms.SensorDetailComponent;  
  
public class RMIServer extends UnicastRemoteObject implements RMIService {  
  
 public static void main(String[] args) throws RemoteException, AlreadyBoundException, IOException {  
  
 Registry registry = LocateRegistry.createRegistry(5099);  
 registry.bind("AirSensorService", new RMIServer());  
  
 System.out.println("Server is starting.....!");  
 System.out.println("Run the Fire monitoring system....!");  
   
 Timer time = new Timer(0, null);  
  
 time.addActionListener(new ActionListener() {  
  
 //periodically check the state  
 @Override  
 public void actionPerformed(ActionEvent e) {  
 try {  
 } catch (Exception e1) {  
  
 e1.printStackTrace();  
 }  
 }  
 });  
  
 time.setRepeats(true);  
 time.setDelay(15000); // periodic interval  
 time.start();  
  
 }  
  
 protected RMIServer() throws RemoteException {  
 super();  
 }  
  
   
 //Retrieve the sensor details  
   
 @Override // http request  
 public String getAllSensorDetails() throws RemoteException {  
 HttpClient client = HttpClient.newHttpClient();  
 HttpRequest request = HttpRequest  
 .newBuilder(URI.create("https://fire-alert-solution.herokuapp.com/api/v1/sensors/")).build();  
 return client.sendAsync(request, HttpResponse.BodyHandlers.ofString()).thenApply(HttpResponse::body)  
 .thenApply((responseBody) -> parse(responseBody)).join();  
 }  
  
 public static String parse(String responseBody) {  
 return responseBody;  
 }  
  
 //validate the admin login details (email and password)  
 @Override  
 public String loginValidator(String email, String password) throws RemoteException {  
  
 JSONObject json = new JSONObject();  
 json.put("email", email);  
 json.put("password", password);  
   
 String adminUN = "admin";  
 String adminPW = "admin321";  
  
 String res = null;  
  
 CloseableHttpClient httpClient = HttpClientBuilder.create().build();  
  
 try {  
 if (email.equals(adminUN) && password.equals(adminPW)){  
 res = "success";  
 }else {  
 res = "failed";  
 }  
  
 } catch (Exception ex) {  
 ex.printStackTrace();  
 } finally {  
 try {  
 httpClient.close();  
 } catch (IOException ex) {  
 ex.printStackTrace();  
 }  
 }  
 return res;  
 }  
  
 //co2 and smoke levels are checked  
 public static void checkStateRepeatedly() {  
   
 HttpClient client = HttpClient.newHttpClient();  
 //http request  
 //its implemented in front end  
   
 }  
   
  
 private static String checkCo2andSmokeLevel(String responseBody) {  
  
 JSONObject res = new JSONObject(responseBody);  
   
 JSONObject data = res.getJSONObject("data");  
   
 JSONArray sensors = data.getJSONArray("sensors");  
  
 for (int i = 0; i < sensors.length(); i++) {  
  
 JSONObject obj = sensors.getJSONObject(i);  
  
 JSONObject lastReading = obj.getJSONObject("lastReading");  
  
 int co2Level = lastReading.getInt("co2Level");  
   
 int smokeLevel = lastReading.getInt("smokeLevel");  
   
 String \_id = obj.getString("\_id");  
  
 if (co2Level > 5 || smokeLevel > 5) {  
  
 //JSON object to send with Email API  
 JSONObject jsonReadingEmail = new JSONObject();  
 jsonReadingEmail.put("smokeLevel", smokeLevel);  
 jsonReadingEmail.put("co2Level", co2Level);  
  
 JSONObject jsonEmail = new JSONObject();  
 jsonEmail.put("to", "uldkavindigunasinghe@gmail.com");  
 jsonEmail.put("sensor", \_id);  
 jsonEmail.put("reading", jsonReadingEmail);  
  
 //JSON object to send with SMS API  
 JSONObject jsonReadingSms = new JSONObject();  
 jsonReadingSms.put("smokeLevel", smokeLevel);  
 jsonReadingSms.put("co2Level", co2Level);  
  
 JSONObject jsonSms = new JSONObject();  
 jsonSms.put("to", "+w");  
 jsonSms.put("sensor", \_id);  
 jsonSms.put("reading", jsonReadingSms);  
  
 CloseableHttpClient httpClient = HttpClientBuilder.create().build();  
  
 try {  
 //http request  
 HttpPost requestEmail = new HttpPost("https://fire-alert-solution.herokuapp.com/api/v1/email");  
 StringEntity paramsEmail = new StringEntity(jsonEmail.toString());  
 // add headers   
 requestEmail.addHeader("content-type", "application/json");  
 requestEmail.addHeader("Authorization", "agfYjhdioJK5ghiH46dHr8gfg857yfrJYuit57vf");  
 requestEmail.setEntity(paramsEmail);  
 org.apache.http.HttpResponse responseEmail = httpClient.execute(requestEmail);  
  
 //http request  
 HttpPost requestSms = new HttpPost("https://fire-alert-solution.herokuapp.com/api/v1/sms");  
 StringEntity paramsSms = new StringEntity(jsonSms.toString());  
 // add headers to the request  
 requestSms.addHeader("content-type", "application/json");  
 requestSms.addHeader("Authorization", "agfYjhdioJK5ghiH46dHr8gfg857yfrJYuit57vf");  
 requestSms.setEntity(paramsSms);  
 org.apache.http.HttpResponse responseSms = httpClient.execute(requestSms);  
  
 //responses are checked  
 System.out.println(responseEmail.getStatusLine().toString().equalsIgnoreCase("HTTP/1.1 201 Created")  
 ? "Email has Sent"  
 : "Error occured!!");  
 System.out.println(responseSms.getStatusLine().toString().equalsIgnoreCase("HTTP/1.1 201 Created")  
 ? "Sms has Sent"  
 : "Error occured!!");  
  
 } catch (Exception ex) {  
 ex.printStackTrace();  
 } finally {  
 try {  
 httpClient.close();  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
 }  
 }  
 }  
  
 return null;  
 }  
  
 //to add sensors  
 @Override  
 public boolean addSensor(String id, int floor, String room) throws RemoteException {  
  
 boolean res = false;  
  
 JSONObject json = new JSONObject();  
 json.put("id", id);  
 json.put("floorNo", floor);  
 json.put("roomNo", room);  
  
 CloseableHttpClient httpClient = HttpClientBuilder.create().build();  
  
 try {  
 // prepare a HTTP request to send to API  
 HttpPost request = new HttpPost("http://localhost:8080/alarms/create");  
 StringEntity params = new StringEntity(json.toString());  
 // add headers to the request  
 request.addHeader("content-type", "application/json");  
 request.setEntity(params);  
 org.apache.http.HttpResponse response = httpClient.execute(request);  
  
 System.out.println(response.getStatusLine().toString().equalsIgnoreCase("Alarm Created"));  
  
 // check the response  
 res = response.getStatusLine().toString().equalsIgnoreCase("Alarm Created");  
  
 } catch (Exception ex) {  
 ex.printStackTrace();  
 } finally {  
 try {  
 httpClient.close();  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
 }  
  
 return res;  
 }  
  
 //edit sensor details  
 @Override  
 public boolean editSensor(String id, int floor, String room) throws RemoteException {  
  
 boolean res = false;  
  
 JSONObject json = new JSONObject();  
 json.put("id", id);  
 json.put("floor", floor);  
 json.put("room", room);  
  
 CloseableHttpClient httpClient = HttpClientBuilder.create().build();  
  
 try {  
 // http request  
 HttpPatch request = new HttpPatch("http://localhost:8080/alarms/edit" + id);  
 StringEntity params = new StringEntity(json.toString());  
   
 request.addHeader("content-type", "application/json");  
   
 request.setEntity(params);  
 org.apache.http.HttpResponse response = httpClient.execute(request);  
  
 System.out.println(response.getStatusLine().toString().equalsIgnoreCase("Alarm Updated"));  
  
 //responses are checked  
 res = response.getStatusLine().toString().equalsIgnoreCase("Alarm Updated");  
  
 } catch (Exception ex) {  
 System.out.println(ex);  
 } finally {  
 try {  
 httpClient.close();  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
 }  
 return res;  
 }  
  
 //delete sensors  
 @Override  
 public boolean deleteSensor(String id) throws RemoteException {  
  
 boolean res = false;  
  
 CloseableHttpClient httpClient = HttpClientBuilder.create().build();  
  
 try {  
 // http request  
 HttpDelete request = new HttpDelete("http://localhost:8080/alarms/delete" + id);  
   
 request.addHeader("content-type", "application/json");  
   
 org.apache.http.HttpResponse response = httpClient.execute(request);  
  
 // responses are checked  
 res = response.getStatusLine().toString().equalsIgnoreCase("Alarm Deleted");  
  
 } catch (Exception ex) {  
 System.out.println(ex);  
 } finally {  
 try {  
 httpClient.close();  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
 }  
 return res;  
 }  
}