A picture containing drawing

Description automatically generated

**SRI LANKA INSTITUTE OF INFORMATION TECHNOLOGY**

DISTRIBUTED SYSTEM

(SE3020)

YEAR 3 SEMESTER 1 | G9\_S1

FIRE ALARM SENSOR MONITORING SYSTEM

REPORT

|  |  |
| --- | --- |
| **STUDENT ID** | **STUDENT NAME** |
| IT18002034 | D.S.R.C.V. Perera |
| IT18028324 | Pabasara D.V.H |
| IT18002102 | N.A. Yapa |

# **Content**

**Page number**

1. Introduction………………………………………………………………………………3
2. High level architectural diagram……………………………………………….4
3. Workflow diagram…………………………………………………………………….5
4. System workflow scenario execution………………………………………..6
5. Appendix – source codes and binaries
   1. web client
   2. sensor application
   3. REST API
   4. RMI sever and RMI client

# **Introduction**

This fire alarm monitoring system is implemented using REST API, web client, dummy sensor application, RMI server and client. the technologies which are used to implement the system as follows;

* REST API - spring boot
* web client - react, bootstrap
* sensor application - react
* RMI server and client - java
* PhpMyAdmin is used for store sensor and user details.

The system administrator registers the sensors, and when the level of co2 and smoke rises, the relevant sensors will be activated.

The fire alarm monitoring system has a web client and a desktop client that can display sensor data such as room number, floor number, co2 level, smoke level and active / inactive status for both users and admin.

A dummy alarm sensor application is implemented to manually control the level of co2 and smoke. The Admin will receive an email from the system when the sensor is activated.

The structure and the process of each component of the project is explained below;

A picture containing object, clock

Description automatically generated

# **High level architectural diagram for fire alarm monitoring system**

A screenshot of a cell phone

Description automatically generated

# **A screenshot of a cell phone Description automatically generatedWorkflow diagram**

# **System workflow scenario execution**

First, it displays a welcome interface to select user the login option as Admin or User (figure 1).

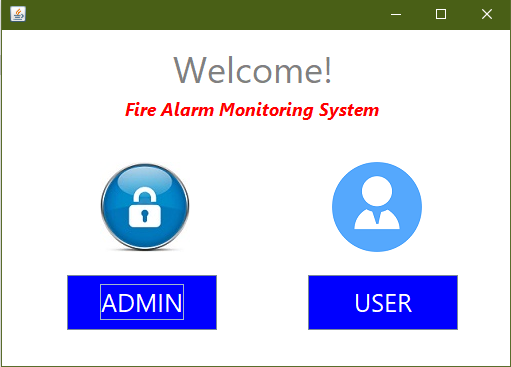


Figure 1

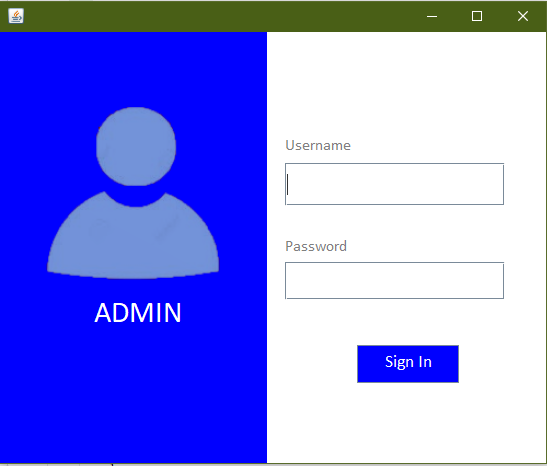
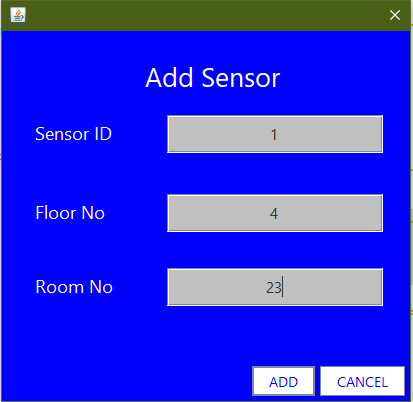
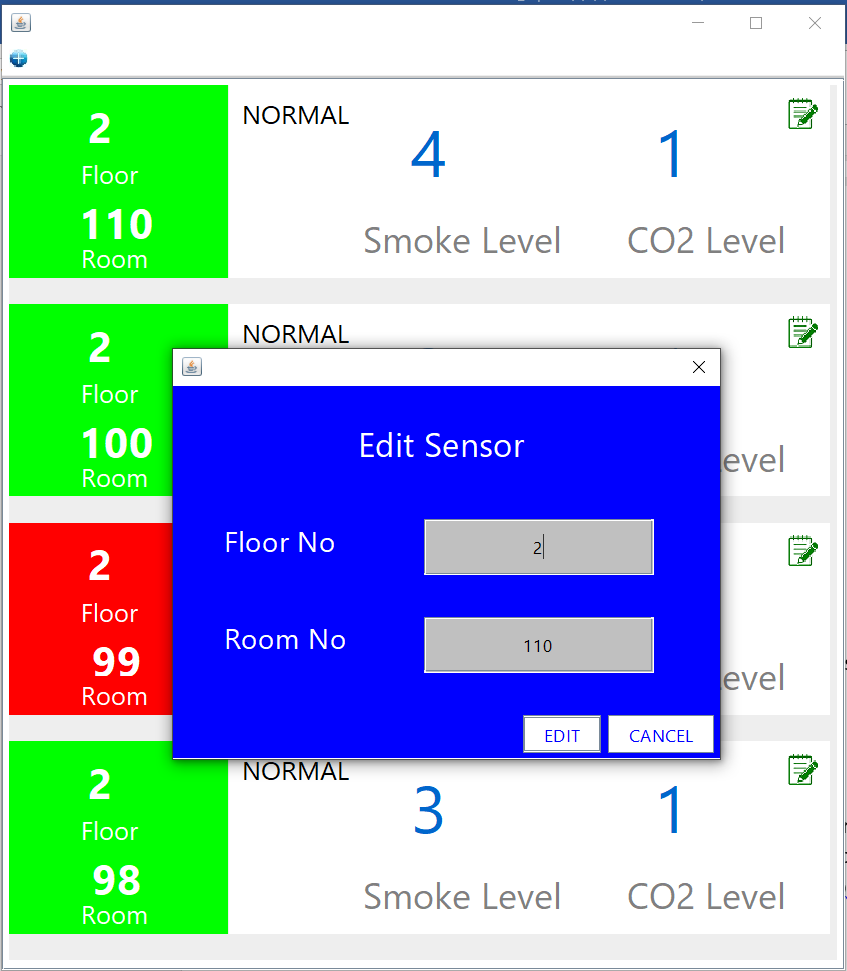
Admin should log in to the system using login credentials (figure 2) and he can add a new sensor for the building entering the relevant details (figure 3).

Figure 3

Figure 2

Admin can view and edit all the registered sensor details (figure 4).



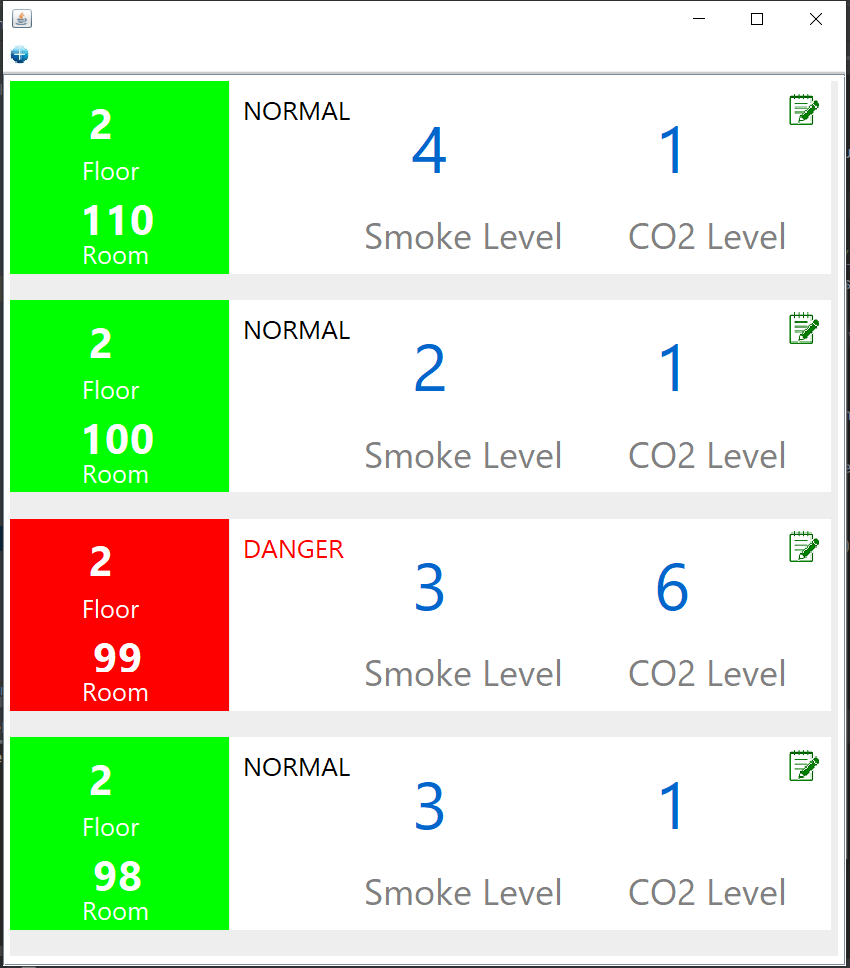


Figure 4

If CO2 level or smoke level is greater than five, it displays a danger alert in red on the dashboard of admin and user.

Parallelly, the CO2 level and smoke level can be updated from the sensor application (Figure 5). Three dummy sensors are used for the demonstration. By using the third sensor, any of the sensor details can be updated providing the necessary inputs.

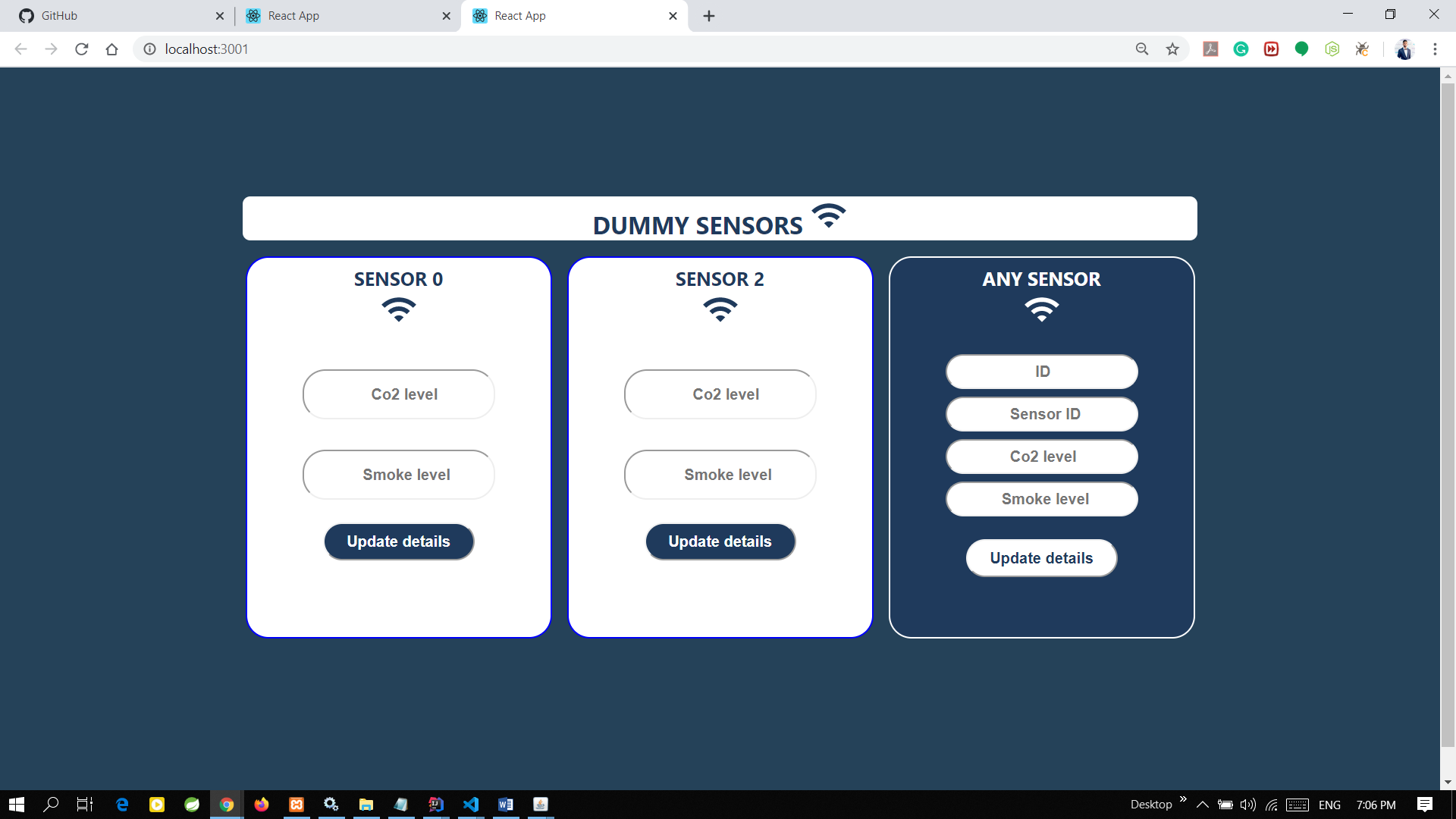


Figure 5

If the CO2 level or smoke level is less than five, sensors are inactive. However, if those values are greater than five are, sensors get active and on the web client (Figure 6.0) and the desktop application (figure 7) the relevant row turned into red. Inactive sensor are displaying in green.

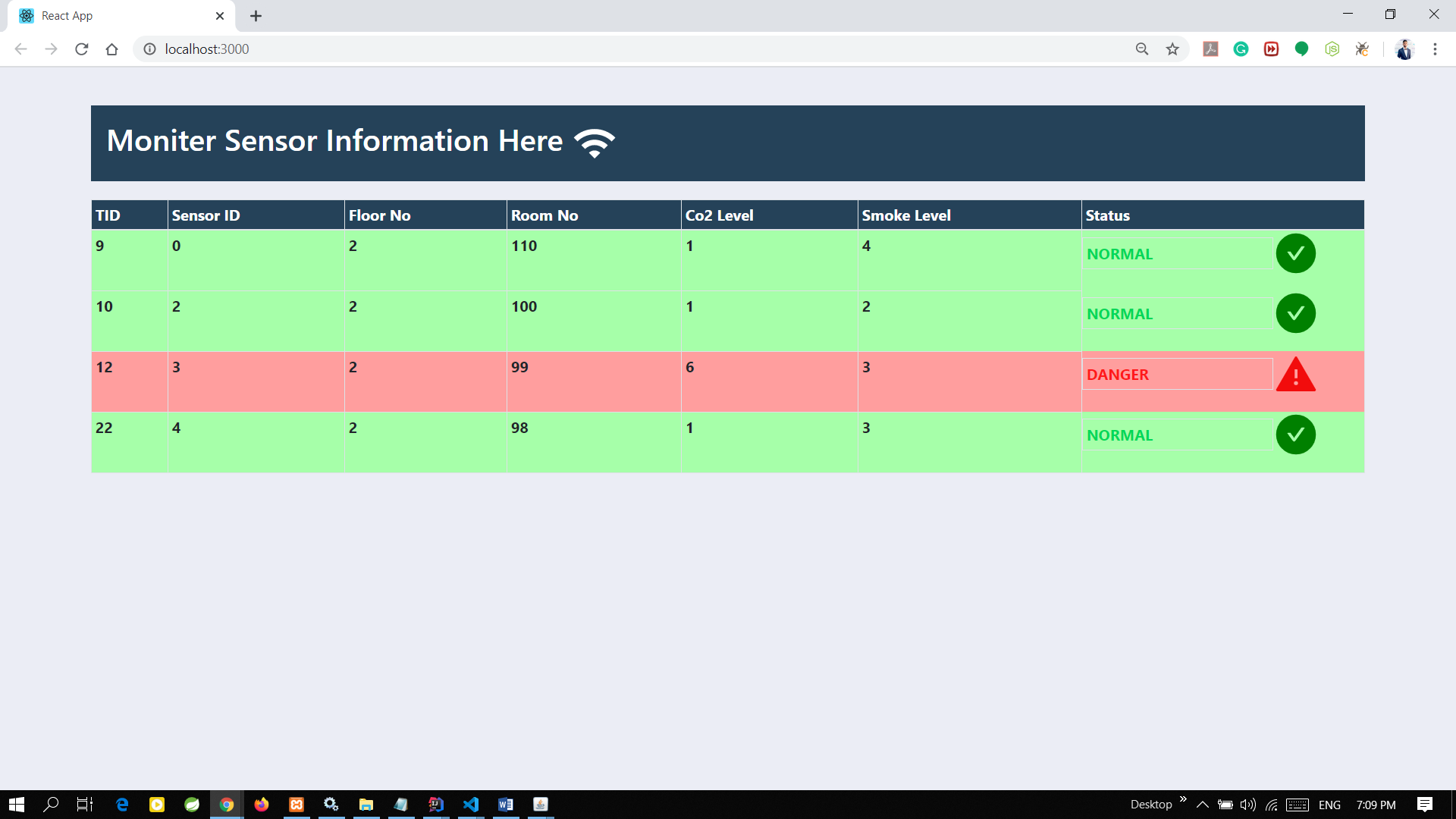


Figure 6

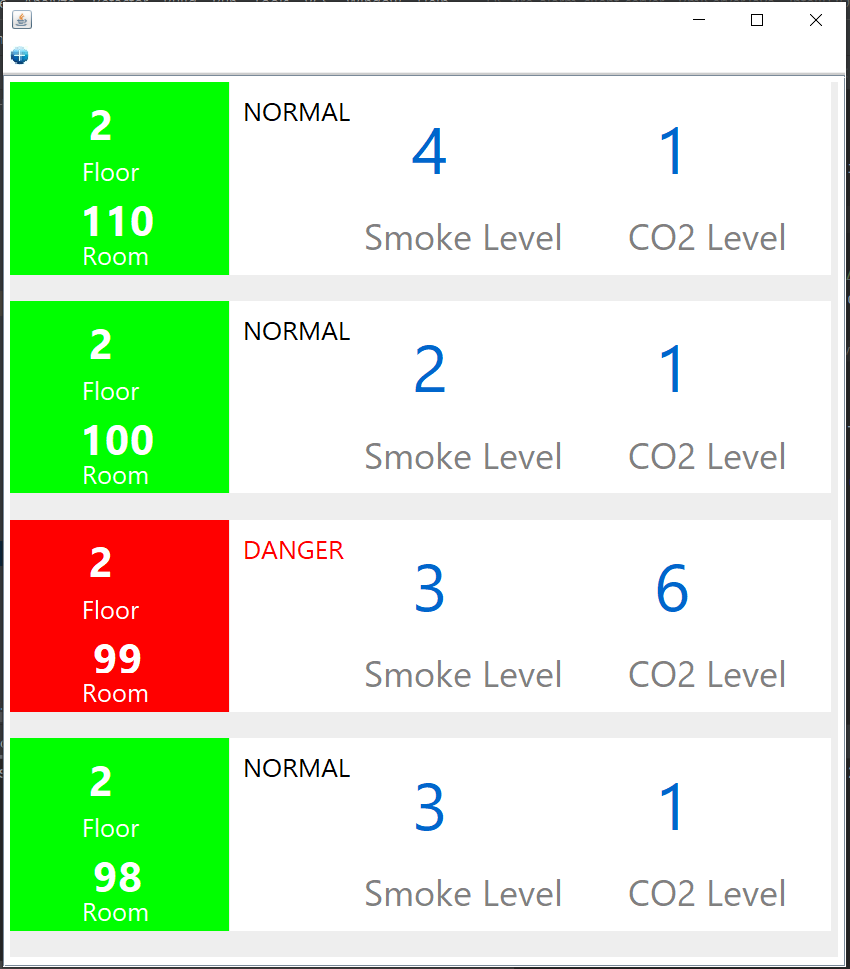


Figure 7

When the sensors are activated (co2 level or smoke level is greater than 5), the admin will get an email of the sensor activation. (Figure 8)

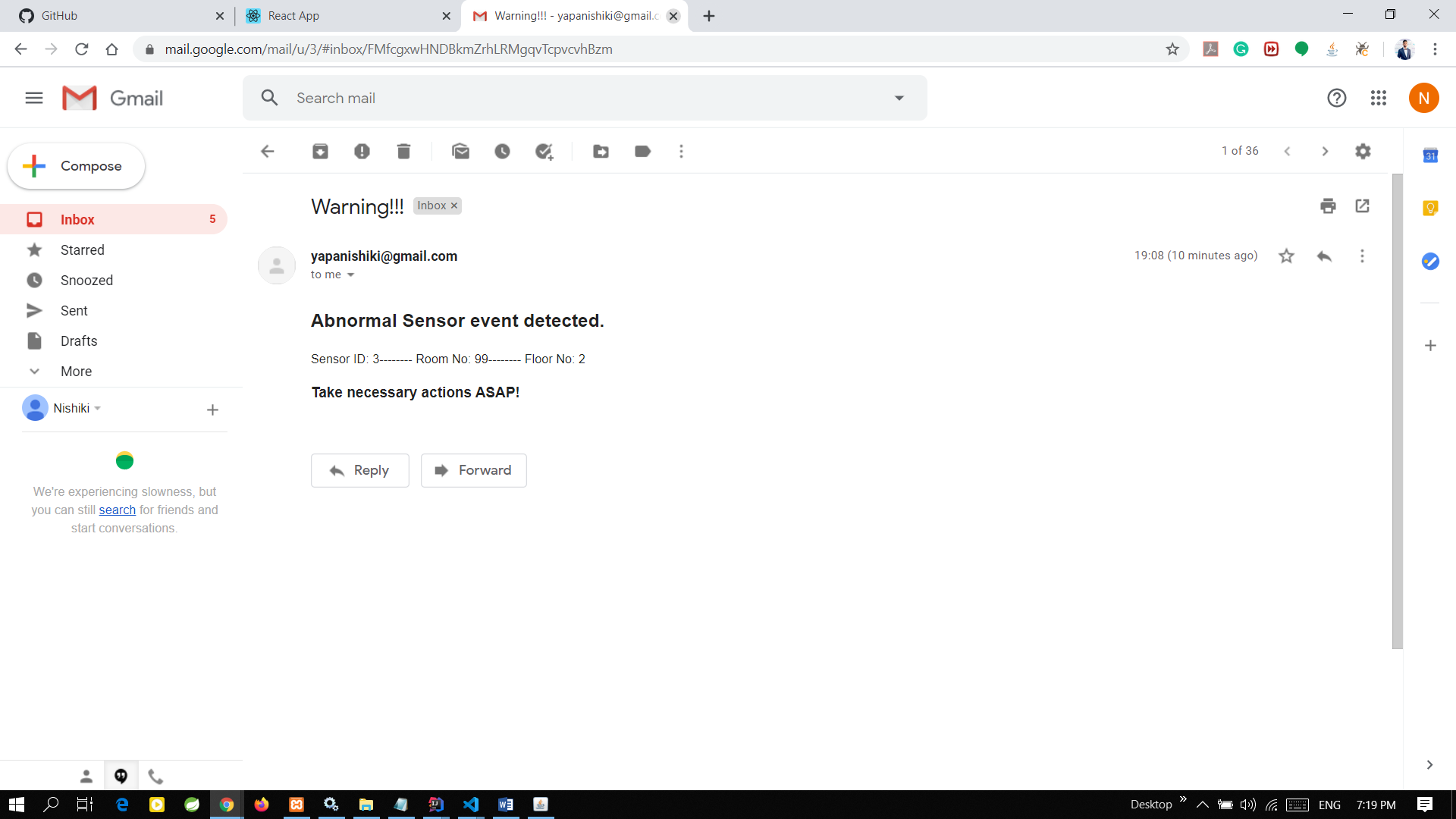


Figure 8

# Appendix

### 5.1 web client

import React, { Component } from 'react';

import { AiFillCheckCircle } from "react-icons/ai";

import { AiFillWarning } from "react-icons/ai";

import { Table } from 'react-bootstrap';

import { MdWifi } from "react-icons/md";

import './App.css';

import axios from 'axios';

export default class App extends Component {

  constructor(props) {

    super(props);

    this.state = { sensoeDetails: [] }

  }

  componentDidMount() {

    this.loadData();

  }

  loadData = () => {

    let baseURL = "http://localhost:8089";

    axios({

      method: 'GET',

      url: baseURL + '/sensors',

    })

      .then(response => {

        this.setState({

          sensoeDetails: response.data

        })

        console.log("success")

        console.log(this.state.sensoeDetails)

        var len = this.state.sensoeDetails.length;

        console.log(len + "this is the lenth")

      })

    //setting the auto refresh time as every 40 secs

    .then(setInterval(this.loadData, 40000))

  }

  showStatus(sensor) {

    if (sensor.smokeLvl >= 5 || sensor.co2Lvl >= 5) {

      return (

        <div style={{ flexDirection: 'row ', fontWeight: 'bold', color: '#ff1919  ', borderColor: 'black' }}>DANGER</div>

      )

    }

    else {

      return (

        <div style={{ flexDirection: 'row ', fontWeight: 'bold', color: '#00D857  ', }}>NORMAL</div>

      )

    }

  }

  showIcon(sensor) {

    if (sensor.smokeLvl >= 5 || sensor.co2Lvl >= 5) {

      return (

        <AiFillWarning size={60} style={{ color: '#f20c0c' }} />

      )

    }

    else {

      return (

        <AiFillCheckCircle size={60} style={{ color: 'green' }} />

      )

    }

  }

  render() {

    return (

      /\*PREVIOUS INTERFACE\*/

      // <div style={{ textAlign: 'center', backgroundColor: 'grren' }}>

      //   <header style={{ minHeight: '100vh', display: 'flex', flexDirection: 'column', justifyContent: 'center', alignItems: 'center' }}>

      //     <h1 style={{}}>SENSOR INFORMATION</h1>

      //     {this.state.sensoeDetails.map(sensor => (

      //       <div style={{

      //         width: 620, height: 160, borderRadius: 30,

      //         marginLeft: 10, marginRight: 10, marginTop: 10, backgroundColor: '#2d2d2d'

      //       }}>

      //         <div style={{

      //           width: 600, height: 140, borderRadius: 30, justifyContent: 'center', alignItems: 'center', display: 'flex',

      //           marginLeft: 10, marginRight: 10, backgroundColor: sensor.co2Lvl >= 5 || sensor.smokeLvl >= 5 ? '#ff1919' : '#00D857'

      //         }}>

      //           <div style={{ padding: 10, flex: 2 }}>

      //             <div style={{ fontWeight: 'bold' }}>Sensor ID  :{sensor.sensorId}</div>

      //             <div style={{ fontWeight: 'bold' }}>Co2 Level  :{sensor.co2Lvl}</div>

      //             <div style={{ fontWeight: 'bold' }}>Smoke Level:{sensor.smokeLvl}</div>

      //             <div style={{ backgroundColor: "#ffffff", height: 40, width: 120, padding: 5, borderRadius: 30, marginTop: 5 }}>

      //               <div style={{ marginLeft: 10 }}>Status   :{this.showStatus(sensor)}</div>

      //             </div>

      //           </div>

      //           <div style={{ flex: 1 }}>

      //             {this.showIcon(sensor)}

      //           </div>

      //         </div>

      //       </div>

      //     ))}

      //   </header>

      // </div>

      /\*NEW INTERFACE\*/

      <div style={{ minHeight: '100vh', backgroundColor: '#ebedf5', justifyContent: 'center', alignItems: 'center' }}>

        <div style={{ paddingTop: 50, marginLeft: 120, marginRight: 120 }}>

          <h1 style={{ backgroundColor: '#254259', padding: 20, color: '#ffffff' }}>Moniter Sensor Information Here        <MdWifi size={60} style={{ color: '#ffffff' }} /></h1>

          <Table className="mt-4" striped bordered hover size="sm" >

            <thead>

              <tr style={{ fontWeight: "bold", color: '#ffffff', fontSize: 20, backgroundColor: '#254259' }}>

                <td>TID</td>

                <td>Sensor ID</td>

                <td>Floor No</td>

                <td>Room No</td>

                <td>Co2 Level</td>

                <td>Smoke Level</td>

                <td>Status</td>

              </tr>

            </thead>

            <tbody>

              {this.state.sensoeDetails.map(sensor =>

                <tr style={{ backgroundColor: sensor.co2Lvl >= 5 || sensor.smokeLvl >= 5 ? '#ff9e9e' : '#a6ffa9', fontSize: 20, padding: 20, height: 80, fontWeight: 'bold', color: '#25425' }}>

                  <td>{sensor.id}</td>

                  <td>{sensor.sensorId}</td>

                  <td>{sensor.floorNo}</td>

                  <td>{sensor.roomNo}</td>

                  <td>{sensor.co2Lvl}</td>

                  <td>{sensor.smokeLvl}</td>

                  <div style={{ display: 'flex', flexDirection: 'row', justifyContent: 'center', alignItems: 'center' }}>

                    <td style={{ flex: 4, justifyContent: 'center', alignItems: 'center' }} >{this.showStatus(sensor)}

                    </td>

                    <div style={{ flex: 2, justifyContent: 'center', alignItems: 'center' }}> {this.showIcon(sensor)}</div>

                  </div>

                </tr>

              )}

            </tbody>

          </Table>

        </div>

      </div>

    );

  }

}

### 5.2 sensor application

import React, { Component } from 'react';

import { MdWifi } from "react-icons/md";

import axios from 'axios';

import './App.css';

export default class App extends Component {

  constructor(props) {

    super(props);

    this.state = {

      co2level0: '',

      smokelevel0: '',

      response0: '',

      co2level1: '',

      smokelevel1: '',

      response1: '',

      co2level2: '',

      smokelevel2: '',

      id: '',

      senid: '',

      response2: '',

      responseSensor0: '',

      responseSensor1: '',

      responseSensor2: '',

    };

  }

  //sensor 0---------------------------------------------------------------------------

  updateco2ValueSensor0 = event => {

    event.preventDefault()

    console.log(event.target.value)

    var val = event.target.value

    console.log(val + "this is the value retrived")

    this.setState({

      co2level0: parseInt(event.target.value)

    }, () => {

      console.log("New state in ASYNC callback:", this.state.co2level0);

    });

    console.log("New state DIRECTLY after setState:", this.state.co2level0);

    console.log("New state DIRECTLY after setState:", this.state.co2level0);

  }

  updatesmokeValueSensor0 = event => {

    event.preventDefault()

    console.log(event.target.value)

    var val = event.target.value

    console.log(val + "this is the value retrived")

    this.setState({

      smokelevel0: parseInt(event.target.value)

    }, () => {

      console.log("New state in ASYNC callback:", this.state.smokelevel0);

    });

    console.log("New state DIRECTLY after setState:", this.state.smokelevel0);

  }

  //sensor 1---------------------------------------------------------------------------

  updateco2ValueSensor1 = event => {

    event.preventDefault()

    console.log(event.target.value)

    var val = event.target.value

    console.log(val + "this is the value retrived")

    this.setState({

      co2level1: parseInt(event.target.value)

    }, () => {

      console.log("New state in ASYNC callback:", this.state.co2level1);

    });

    console.log("New state DIRECTLY after setState:", this.state.co2level1);

    console.log("New state DIRECTLY after setState:", this.state.co2level1);

  }

  updatesmokeValueSensor1 = event => {

    event.preventDefault()

    console.log(event.target.value)

    var val = event.target.value

    console.log(val + "this is the value retrived")

    this.setState({

      smokelevel1: parseInt(event.target.value)

    }, () => {

      console.log("New state in ASYNC callback:", this.state.smokelevel1);

    });

    console.log("New state DIRECTLY after setState:", this.state.smokelevel1);

  }

  //sensor 2---------------------------------------------------------------------------

  updateco2ValueSensor2 = event => {

    event.preventDefault()

    console.log(event.target.value)

    var val = event.target.value

    console.log(val + "this is the value retrived")

    this.setState({

      co2level2: parseInt(event.target.value)

    }, () => {

      console.log("New state in ASYNC callback:", this.state.co2level2);

    });

    console.log("New state DIRECTLY after setState:", this.state.co2level2);

    console.log("New state DIRECTLY after setState:", this.state.co2level2);

  }

  updatesmokeValueSensor2 = event => {

    event.preventDefault()

    console.log(event.target.value)

    var val = event.target.value

    console.log(val + "this is the value retrived")

    this.setState({

      smokelevel2: parseInt(event.target.value)

    }, () => {

      console.log("New state in ASYNC callback:", this.state.smokelevel2);

    });

    console.log("New state DIRECTLY after setState:", this.state.smokelevel2);

  }

  updateid = event => {

    event.preventDefault()

    console.log(event.target.value)

    var val = event.target.value

    console.log(val + "this is the value retrived")

    this.setState({

      id: parseInt(event.target.value)

    }, () => {

      console.log("New state in ASYNC callback:", this.state.id);

    });

    console.log("New state DIRECTLY after setState:", this.state.id);

  }

  updatesenid = event => {

    event.preventDefault()

    console.log(event.target.value)

    var val = event.target.value

    console.log(val + "this is the value retrived")

    this.setState({

      senid: parseInt(event.target.value)

    }, () => {

      console.log("New state in ASYNC callback:", this.state.senid);

    });

    console.log("New state DIRECTLY after setState:", this.state.senid);

  }

  //sensor0 api call---------------------------------------------------------------------------

  updateapi0 = (event) => {

    event.preventDefault();

    // alert("successfully added")

    let sensor0body = JSON.stringify(

      {

        "co2Lvl": this.state.co2level0,

        "smokeLvl": this.state.smokelevel0,

        "id": 9,

        "sensorId": 0,

      }

    );

    axios({

      headers: {

        // 'Content-Type ': 'application/x-www-form-urlencoded;charset=UTF-8'

        'Content-Type': 'application/json;charset=UTF-8',

      },

      method: 'PUT',

      url: 'http://localhost:8089/updateSensorApp',

      data: sensor0body,

    })

      .then(response => {

        console.log("Arrived to send request")

        this.setState({

          responseSensor0: response.data

        })

      })

      .then(() => {

        alert('Successfully updated !')

      })

      .catch((console.log("ISSUES !")))

  }

  //sensor1 api call---------------------------------------------------------------------------

  updateapi1 = (event) => {

    event.preventDefault();

    // alert("successfully added")

    let sensor0body1 = JSON.stringify(

      {

        "co2Lvl": this.state.co2level1,

        "smokeLvl": this.state.smokelevel1,

        "id": 10,

        "sensorId": 2,

      }

    );

    axios({

      headers: {

        // 'Content-Type ': 'application/x-www-form-urlencoded;charset=UTF-8'

        'Content-Type': 'application/json;charset=UTF-8',

      },

      method: 'PUT',

      url: 'http://localhost:8089/updateSensorApp',

      data: sensor0body1,

    })

      .then(response => {

        console.log("Arrived to send request")

        this.setState({

          responseSensor1: response.data

        })

      })

      .then(() => {

        alert('Successfully updated !')

      })

      .catch((console.log("ISSUES !")))

  }

  //sensor2 api call---------------------------------------------------------------------------

  updateapi2 = (event) => {

    event.preventDefault();

    // alert("successfully added")

    let sensor0body2 = JSON.stringify(

      {

        "co2Lvl": this.state.co2level2,

        "smokeLvl": this.state.smokelevel2,

        "id": this.state.id,

        "sensorId": this.state.senid,

      }

    );

    axios({

      headers: {

        // 'Content-Type ': 'application/x-www-form-urlencoded;charset=UTF-8'

        'Content-Type': 'application/json;charset=UTF-8',

      },

      method: 'PUT',

      url: 'http://localhost:8089/updateSensorApp',

      data: sensor0body2,

    })

      .then(response => {

        console.log("Arrived to send request")

        this.setState({

          responseSensor2: response.data

        })

      })

      .then(() => {

        alert('Successfully updated !')

      })

      .catch((console.log("ISSUES !")))

  }

  render() {

    return (

      <div style={{ minHeight: '100vh', textAlign: 'center', backgroundColor: '#254259', padding: 20, justifyContent: 'center', alignItems: 'center' }}>

        <div style={{ marginTop: 150, backgroundColor: '#ffffff', marginLeft: 300, marginRight: 300, borderRadius: 10 }}>

          <h1 style={{ marginTop: 50, color: '#1f3a5c' }}>DUMMY SENSORS <MdWifi size={50} style={{ color: '#1f3a5c' }} /></h1>

        </div>

        <header style={{ display: 'flex', flexDirection: 'row', justifyContent: 'center', alignItems: 'center' }}>

          <div style={{ backgroundColor: '#ffffff', height: 500, width: 400, borderRadius: 30, border: '2px solid blue' }} >

            <div style={{ fontSize: 25, marginTop: 10, color: '#1f3a5c', fontWeight: 'bold', }}>

              SENSOR 0

          </div>

            <MdWifi size={50} style={{ color: '#1f3a5c' }} />

            <div style={{ flexDirection: 'column', marginTop: 50 }}>

              <form >

                <input type="text" placeholder="                Co2 level" name="co2level0" onChange={this.updateco2ValueSensor0}

                  style={{ height: 60, width: 250, borderRadius: 30, fontSize: 20, alignItems: 'center', fontWeight: 'bold', }}

                />

                <br />

                <input type="text" name="smokelevel0" onChange={this.updatesmokeValueSensor0}

                  style={{ height: 60, width: 250, borderRadius: 30, marginTop: 40, fontSize: 20, alignItems: 'center', fontWeight: 'bold', }}

                  placeholder="              Smoke level"

                />

                <br />

                <button onClick={this.updateapi0} style={{

                  height: 50, width: 200, borderRadius: 30, backgroundColor: '#1f3a5c', color: '#ffffff', fontSize: 20, fontWeight: 'bold', marginTop: 30

                }}>

                  Update details

                  </button>

              </form>

            </div>

          </div>

          <div style={{ backgroundColor: '#ffffff', height: 500, width: 400, borderRadius: 30, border: '2px solid blue', marginLeft: 20 }} >

            <div style={{ fontSize: 25, fontWeight: 'bold', marginTop: 10, color: '#1f3a5c' }}>

              SENSOR 2

          </div>

            <MdWifi size={50} style={{ color: '#1f3a5c' }} />

            <div style={{ flexDirection: 'column', marginTop: 50 }}>

              <form>

                <input type="text" style={{ height: 60, width: 250, borderRadius: 30, fontSize: 20, alignItems: 'center', fontWeight: 'bold', }} onChange={this.updateco2ValueSensor1}

                  placeholder="                Co2 level"

                />

                <br />

                <input type="text" style={{ height: 60, width: 250, borderRadius: 30, marginTop: 40, fontSize: 20, alignItems: 'center', fontWeight: 'bold', }} onChange={this.updatesmokeValueSensor1}

                  placeholder="              Smoke level"

                />

                <button onClick={this.updateapi1} style={{

                  height: 50, width: 200, borderRadius: 30, backgroundColor: '#1f3a5c', color: '#ffffff', fontSize: 20, fontWeight: 'bold', marginTop: 30

                }}>

                  Update details

                  </button>

              </form>

            </div>

          </div>

          <div style={{ backgroundColor: '#1f3a5c', height: 500, width: 400, borderRadius: 30, border: '2px solid white', marginLeft: 20 }} >

            <div style={{ fontSize: 25, fontWeight: 'bold', marginTop: 10, color: '#ffffff' }}>

              ANY SENSOR

          </div>

            <MdWifi size={50} style={{ color: '#ffffff' }} />

            <div style={{ flexDirection: 'column', marginTop: 30 }}>

              <form>

                <input type="text" style={{ height: 40, width: 250, borderRadius: 30, fontSize: 20, alignItems: 'center', fontWeight: 'bold', }} onChange={this.updateid}

                  placeholder="                     ID"

                />

                <input type="text" style={{ height: 40, width: 250, borderRadius: 30, fontSize: 20, alignItems: 'center', fontWeight: 'bold', marginTop: 10 }} onChange={this.updatesenid}

                  placeholder="               Sensor ID"

                />

                <input type="text" style={{ height: 40, width: 250, borderRadius: 30, fontSize: 20, alignItems: 'center', fontWeight: 'bold', marginTop: 10 }} onChange={this.updateco2ValueSensor2}

                  placeholder="               Co2 level"

                />

                <br />

                <input type="text" style={{ height: 40, width: 250, borderRadius: 30, marginTop: 10, fontSize: 20, alignItems: 'center', fontWeight: 'bold', }} onChange={this.updatesmokeValueSensor2}

                  placeholder="             Smoke level"

                />

                <button onClick={this.updateapi2} style={{

                  height: 50, width: 200, borderRadius: 30, backgroundColor: '#ffffff', color: '#1f3a5c', fontSize: 20, fontWeight: 'bold', marginTop: 30

                }}>

                  Update details

                  </button>

              </form>

            </div>

          </div>

        </header>

      </div>

    );

  }

}

### 5.3 REST API

**Entity – sensorApp**

@Data  
@AllArgsConstructor  
@NoArgsConstructor  
@Entity  
public class SensorApp {  
  
 @Id  
 @GeneratedValue  
 private int Id;  
 private int sensorId;  
 private int floorNo;  
 private int roomNo;  
 private int co2Lvl;  
 private int smokeLvl;  
 private String status;  
  
 public int getId() {  
 return Id;  
 }  
  
 public void setId(int id) {  
 Id = id;  
 }  
  
 public int getCo2Lvl() {  
 return co2Lvl;  
 }  
  
 public void setCo2Lvl(int co2Lvl) {  
 this.co2Lvl = co2Lvl;  
 }  
  
 public int getSmokeLvl() {  
 return smokeLvl;  
 }  
  
 public void setSmokeLvl(int smokeLvl) {  
 this.smokeLvl = smokeLvl;  
 }  
  
 public String getStatus() {  
 return status;  
 }  
  
 public void setStatus(String status) {  
 this.status = status;  
 }  
  
 public int getSensorId() {  
 return sensorId;  
 }  
  
 public void setSensorId(int sensorId) {  
 this.sensorId = sensorId;  
 }  
  
 public int getFloorNo() {  
 return floorNo;  
 }  
  
 public void setFloorNo(int floorNo) {  
 this.floorNo = floorNo;  
 }  
  
 public int getRoomNo() {  
 return roomNo;  
 }  
  
 public void setRoomNo(int roomNo) {  
 this.roomNo = roomNo;  
 }  
}

**Controller – sensorApp**

@CrossOrigin(origin=” \*”)  
  
@RestController  
class SensorController {  
  
 @Autowired  
 private SensorService service;  
  
 *//add sensor details from desktop app* @PostMapping("/addSensor")  
 public SensorApp addSensorDetails(@RequestBody SensorApp sensor) {  
 return service.saveSensor(sensor);  
 }  
  
 *//edit sensor details from the desktop app* @PutMapping("/editSensor")  
 public SensorApp editSensorDetails(@RequestBody SensorApp desktopapp){  
 return service.updateDesktopApp(desktopapp);  
 }  
  
 *//retrieve data to the web client and desktop app* @GetMapping("/sensors")  
 public List<SensorApp> getSensor() {  
 return service.getAllSensorDetails();  
 }  
  
 *//update sensor details from sensor app* @PutMapping("/updateSensorApp")  
 public SensorApp updateSensorApp(@RequestBody SensorApp sensorApp) {  
 return service.updateSensorApp(sensorApp);  
 }  
  
 *//delete - web client* @DeleteMapping("/delete/{Id}")  
 public String deleteSensorDetails(@PathVariable int Id) {  
 return service.deleteSensorDetails(Id);  
 }  
  
  
}

**Services – sensorService**

@Service  
public class SensorService {  
  
 @Autowired  
 private SensorRepository srepo;  
  
 @Autowired  
 private JavaMailSender mailSender;  
  
 *//add sensor id, room no and floor no from the desktop app* public SensorApp saveSensor(SensorApp sensorApp){  
 return srepo.save(sensorApp);  
 }  
  
 *//read sensor details - webclient and desktop client* public List<SensorApp> getAllSensorDetails() {  
 return srepo.findAll();  
 }  
  
 *//update co2 lvl, smoke lvl from the sensor app* public SensorApp updateSensorApp(SensorApp sensorApp) {  
  
 SensorApp exisitingSensor = srepo.findById(sensorApp.getId()).orElse(null);  
 exisitingSensor.setSensorId(sensorApp.getSensorId());  
 exisitingSensor.setCo2Lvl(sensorApp.getCo2Lvl());  
 exisitingSensor.setSmokeLvl(sensorApp.getSmokeLvl());  
  
 *// check for the sensor value* if (sensorApp.getCo2Lvl() > 5 || sensorApp.getSmokeLvl() > 5) {  
 *// if the values are greater than than 5 send the email* sendMail(exisitingSensor.getSensorId(), exisitingSensor.getFloorNo(), exisitingSensor.getRoomNo());  
 }  
 return srepo.save(exisitingSensor);  
 }  
  
 *//edit room no, floor no from desktop app* public SensorApp updateDesktopApp(SensorApp desktopApp){  
 SensorApp ex = srepo.findBySensorId(desktopApp.getSensorId());  
 if (ex != null) {  
 ex.setSensorId(desktopApp.getSensorId());  
 ex.setRoomNo(desktopApp.getRoomNo());  
 ex.setFloorNo(desktopApp.getFloorNo());  
 } else {  
 return null;  
 }  
 return srepo.save(ex);  
 }  
  
 private void sendMail(int sensorId, int floorNo, int roomNo) {  
 *// send an email to the admin* MimeMessage msg = mailSender.createMimeMessage();  
  
 MimeMessageHelper helper;  
 try {  
 helper = new MimeMessageHelper(msg, true);  
 helper.setFrom("no-reply@gigara.info");  
 helper.setTo("yapanishiki@gmail.com");  
 helper.setSubject("Warning!!!");  
 helper.setText("<h2>Abnormal Sensor event detected.</h2>" +  
 "Sensor ID: " + sensorId +  
 "-------- Room No: </br>" + roomNo +  
 "-------- Floor No: " + floorNo +  
 "<h3>Take necessary actions ASAP!</h3>", true);  
 mailSender.send(msg);  
 } catch (Exception e) {  
 e.printStackTrace();  
 }  
 }  
 *//delete - web client* public String deleteSensorDetails(int Id) {  
 srepo.deleteById(Id);  
 return "Sensor removed!";  
 }  
}

**Repository – sensorRepository**

public interface SensorRepository extends JpaRepository<SensorApp, Integer> {  
 SensorApp findBySensorId(int id);  
}

**FireAlarmSensorApplication – Main**

@SpringBootApplication  
public class FirAlarmSensorApplication {  
  
 public static void main(String[] args)  
 {  
 SpringApplication.*run*(FirAlarmSensorApplication.class, args);  
 }  
  
}

**Application properties**

spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver  
spring.datasource.url=jdbc:mysql://localhost:3306/sensor\_db?serverTimezone=UTC  
spring.datasource.username=root  
spring.datasource.password=  
spring.jpa.show-sql=true  
spring.jpa.properties.hibernate.dialect = org.hibernate.dialect.MySQL5Dialect  
spring.jpa.hibernate.ddl-auto=update  
server.port=8088

### 5.4 RMI server and RMI client

**RMI Server**

public class RmiServer extends UnicastRemoteObject implements RmiService {  
  
 protected RmiServer() throws RemoteException {  
 super();  
 }  
  
 public static void main(String[] args) throws RemoteException, AlreadyBoundException, IOException {  
  
 Registry registry = LocateRegistry.*createRegistry*(1099);  
 registry.bind("FireAlarmSensor", new RmiServer());  
  
 System.*out*.println("Server is ready...");  
   
 Timer t = new Timer(0, null);  
  
 t.addActionListener(new ActionListener() {  
  
 @Override  
 public void actionPerformed(ActionEvent e) {  
 try {  
 *//checkStateRepeatedly();* } catch (Exception e1) {  
 e1.printStackTrace();  
 }  
 }  
 });  
  
 t.setRepeats(true);  
 t.setDelay(15000); *// repeat interval* t.start();  
  
 }

*\* Check Admin login credentials* public boolean ValidateLogin(String username, String password) throws RemoteException{  
   
 boolean found = false;  
 try {  
 if(username.equals("admin") && password.equals("admin"))  
 {  
 return found = true;  
 }else {  
 return found=false;  
 }  
   
 }catch(Exception ex) {  
 ex.printStackTrace();  
 }  
 return found;  
 }  
   
 */\*  
 \* Retrieve all the sensor details  
 \*/* public String getSensorDetails() throws RemoteException {  
 HttpClient client = HttpClient.*newHttpClient*();  
 *// prepare a HTTP request to send to API* HttpRequest request = HttpRequest  
 .*newBuilder*(URI.*create*("http://localhost:8089/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;  
 }  
  
  
 */\*  
 \* Register a new sensor to the system  
 \*/* @Override  
 public boolean addSensor(String id, int floor, String room) throws RemoteException {  
  
 boolean res = false;  
  
 JSONObject json = new JSONObject();  
 json.put("sensorId", 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:8089/addSensor");  
 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("HTTP/1.1 201 Created"));  
  
 *// check the response* res = response.getStatusLine().toString().equalsIgnoreCase("HTTP/1.1 201 Created");  
  
 } catch (Exception ex) {  
 ex.printStackTrace();  
 } finally {  
 try {  
 httpClient.close();  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
 }  
  
 return res;  
 }  
  
  *//Edit registered sensor details* @Override  
 public boolean editSensor(String id, int floor, String room) throws RemoteException {  
  
 boolean res = false;  
  
 JSONObject json = new JSONObject();  
 json.put("sensorId", id);  
 json.put("floorNo", floor);  
 json.put("roomNo", room);  
  
 CloseableHttpClient httpClient = HttpClientBuilder.*create*().build();  
  
 try {  
 *// prepare a HTTP request to send to API* HttpPut request = new HttpPut("http://localhost:8089/editSensor/");  
 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("HTTP/1.1 200 OK"));  
  
 *// check the response* res = response.getStatusLine().toString().equalsIgnoreCase("HTTP/1.1 200 OK");  
  
 } catch (Exception ex) {  
 System.*out*.println(ex);  
 } finally {  
 try {  
 httpClient.close();  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
 }  
 return res;  
 }  
  
}

**RMI Service**

public interface RmiService extends Remote {  
 public String getSensorDetails() throws RemoteException;  
  
 public boolean addSensor(String id, int floor, String room) throws RemoteException;  
  
 public boolean ValidateLogin(String username, String password) throws RemoteException;  
  
 public boolean editSensor(String id, int floor, String room) throws RemoteException;  
  
}

**Login**

btnLogin.addActionListener(new ActionListener() {  
  
 public void actionPerformed(ActionEvent e) {  
  
 boolean f = false;  
 String username = txtusername.getText();  
 String password = txtpassword.getText();  
  
 RmiService service;  
 *//String result = null;* try {  
 *// find the remote service* service = (RmiService) Naming.*lookup*("rmi://localhost:1099/FireAlarmSensor");  
 *// invoke the remote method* f = service.ValidateLogin(username, password);  
   
  
 } catch (MalformedURLException | RemoteException | NotBoundException ex) {  
 ex.printStackTrace();  
 }  
 System.*out*.println(f); *// write the result on the console* if (f == true) {  
  
 Main main = new Main(true); *//open the main interface* main.*main*(null);  
 } else {  
 errormsg.setVisible(true);  
 }   
 }  
});  
contentPane.add(btnLogin);

**ADD Sensor**

okButton.addActionListener(new ActionListener() {  
 public void actionPerformed(ActionEvent e) {  
  
 String id = txtsensorid.getText();  
 int floor = Integer.*parseInt*(txtfloorno.getText());  
 String room = txtroomno.getText();  
  
 RmiService service;  
  
 try {  
 *//find the service* service = (RmiService) Naming.*lookup*("rmi://localhost:1099/FireAlarmSensor");  
  
 try {   
 *//invoke remote method and assign to a variable* res = service.addSensor(id, floor, room);  
 JOptionPane.*showMessageDialog*(null, "Added Successfully");  
 dispose();  
  
 } catch (RemoteException e1) {  
 e1.printStackTrace();  
 }  
 } catch (MalformedURLException | RemoteException | NotBoundException ex) {  
 ex.printStackTrace();  
 }  
  
 if (res) {  
  
 Main main = new Main(true);  
 main.*main*(null);  
 } else {  
 lblerromsg.setVisible(true);  
 }  
  
 }  
});  
buttonPane.add(okButton);

**Edit Sensor**

okButton.addActionListener(new ActionListener() {  
 public void actionPerformed(ActionEvent e) {  
  
 RmiService service;  
  
 try {  
 *//find the service* service = (RmiService) Naming.*lookup*("rmi://localhost:1099/FireAlarmSensor");  
  
 try {  
 *//invoke remote method and assign to a variable* res = service.editSensor(lblsensorid, Integer.*parseInt*(txtFloorNo.getText()),  
 txtRoomNo.getText());  
 JOptionPane.*showMessageDialog*(null, "Edited Successfully");  
 dispose();  
  
 } catch (RemoteException e1) {  
 e1.printStackTrace();  
 }  
 } catch (MalformedURLException | RemoteException | NotBoundException ex) {  
 ex.printStackTrace();  
 }  
  
 if (res) {  
 frame.dispose();  
 Main main = new Main(true);  
 main.*main*(null);  
  
 } else {  
 System.*out*.println("Error");  
 }  
 }  
 });  
 okButton.setActionCommand("OK");  
 buttonPane.add(okButton);  
 getRootPane().setDefaultButton(okButton);  
}

**Dashboard**

public class Main extends JFrame {  
  
 private static JPanel *contentPane*;  
 private static String *responseBody*;  
 private static Main *frame*;  
 private static boolean *isAdmin* = true;  
  
  
 */\*\*  
 \* Launch the application.  
 \*/* public static void main(String[] args) {  
 EventQueue.*invokeLater*(new Runnable() {  
 public void run() {  
 try {  
 *frame* = new Main(*isAdmin*);  
 *frame*.setVisible(true);  
 } catch (Exception e) {  
 e.printStackTrace();  
 }  
   
 RmiService service;  
  
 try {  
  
 service = (RmiService) Naming.*lookup*("rmi://localhost:1099/FireAlarmSensor");  
 Timer t = new Timer(0, null);  
  
 t.addActionListener(new ActionListener() {  
 @Override  
 public void actionPerformed(ActionEvent e) {  
 try {  
 *responseBody* = service.getSensorDetails();  
 } catch (RemoteException e1) {  
 e1.printStackTrace();  
 }  
 *showSensorDetails*(*responseBody*);  
 }  
 });  
  
 t.setRepeats(true);  
 t.setDelay(5000); *// repeat interval is t.setDelay(30000);* t.start();  
  
 } catch (MalformedURLException | RemoteException | NotBoundException e) {  
 e.printStackTrace();  
 }  
 }  
 });  
 }

*// Create the frame.* public Main(boolean isAdmin) {  
 Main.*isAdmin* = isAdmin;  
 setDefaultCloseOperation(JFrame.*EXIT\_ON\_CLOSE*);  
 setBounds(210, 0, 689, 780);  
  
  
 JMenuBar menuBar = new JMenuBar();  
 menuBar.setBackground(Color.*WHITE*);  
 setJMenuBar(menuBar);  
  
 JMenu mnNewMenu = new JMenu("File");  
 mnNewMenu.setIcon(new ImageIcon(Main.class.getResource("/img/plus.jpg")));  
 mnNewMenu.setFont(new Font("Segoe UI", Font.*PLAIN*, 16));  
 mnNewMenu.setForeground(Color.*WHITE*);  
 menuBar.add(mnNewMenu);  
  
 JMenuItem menuItem1 = new JMenuItem("Add new Sensor");  
 menuItem1.setFont(new Font("Segoe UI", Font.*PLAIN*, 14));  
 menuItem1.setForeground(Color.*WHITE*);  
 menuItem1.setBackground(Color.*BLUE*);  
  
 menuItem1.addActionListener(new ActionListener() {  
 public void actionPerformed(ActionEvent e) {  
 AddSensor addSensor = new AddSensor();  
 addSensor.setVisible(true);  
 }  
 });  
 mnNewMenu.add(menuItem1);  
 menuItem1.setVisible(isAdmin);  
  
 JSeparator separator = new JSeparator();  
 mnNewMenu.add(separator);  
  
 JMenuItem menuItem2 = new JMenuItem("Exit");  
 menuItem2.setFont(new Font("Segoe UI", Font.*PLAIN*, 14));  
 menuItem2.setForeground(Color.*WHITE*);  
 menuItem2.setBackground(Color.*RED*);  
  
 menuItem2.addActionListener(new ActionListener() {  
 public void actionPerformed(ActionEvent e) {  
 System.*exit*(*EXIT\_ON\_CLOSE*);  
 }  
 });  
 mnNewMenu.add(menuItem2);  
  
 *contentPane* = new JPanel();  
 *contentPane*.setBackground(Color.*WHITE*);  
 *contentPane*.setBorder(new EmptyBorder(5, 5, 5, 5));  
  
 GridLayout gridLayout = new GridLayout();  
 gridLayout.setColumns(1);  
 gridLayout.setRows(0);  
 *contentPane*.setLayout(gridLayout);  
  
 JScrollPane scrollPane = new JScrollPane(*contentPane*, JScrollPane.*VERTICAL\_SCROLLBAR\_AS\_NEEDED*,  
 JScrollPane.*HORIZONTAL\_SCROLLBAR\_NEVER*);  
 setContentPane(scrollPane);  
  
 }  
  
  *// used to display sensor details* public static void showSensorDetails(String responseBody) {  
  
 *contentPane*.removeAll();  
  
 JSONArray resArray = new JSONArray(responseBody);  
  
 *//display sensor details in the user interface* for (int i = 0; i < resArray.length(); i++) { *//loop through the responseBody* JSONObject obj = resArray.getJSONObject(i);  
  
  
  
 int co2Level = obj.getInt("co2Lvl");  
 int smokeLevel = obj.getInt("smokeLvl");  
  
 String id = String.*valueOf*(obj.getInt("sensorId"));  
 int floor = obj.getInt("floorNo");  
 String room = String.*valueOf*(obj.getInt("roomNo"));  
  
 SensorDetails sensorDetails = new SensorDetails(id, floor, room, co2Level < 5 && smokeLevel <5,  
 co2Level, smokeLevel, *isAdmin*, *frame*);  
 sensorDetails.setVisible(true);  
 *contentPane*.add(sensorDetails);  
 }  
  
 *contentPane*.validate();  
 *contentPane*.repaint();  
 }  
  
}