Distributed System
(SE3020)

Fire Alarm Monitoring System

Report

Group Details

Kulendran.P	IT17184854
Attanayake T.H.M.D.R.M	IT16006058
Amuthini.K	IT17173100
Jeyasumangala.R	IT17160162

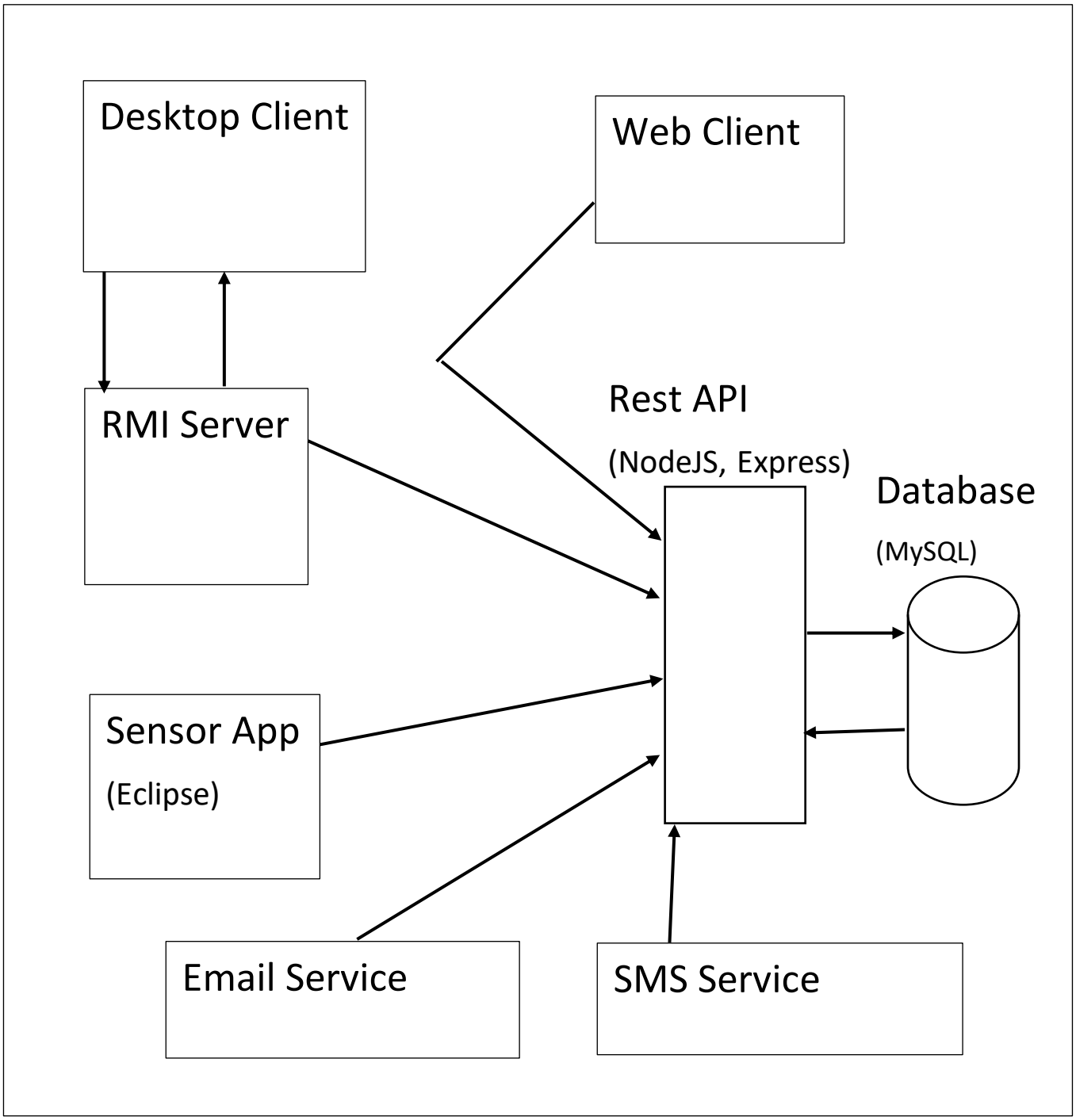
Table of Contents

Introduction

The fire Alarm Monitoring System is implemented to monitor the fire sensors in a building. The System has a Web Client Application to monitor the sensor remotely, desktop Client Application to view the user and sensor details by desktop client, Administrator Dashboard to view, register and edit the sensor details, REST API to provide sensor services, Email and SMS Services to send the Email and SMS updates when the when any problem occurs (when the CO2 or smoke level increases), Dummy Sensor app to periodically send sensor details to database, And a database to Store and manage the sensor and user details.

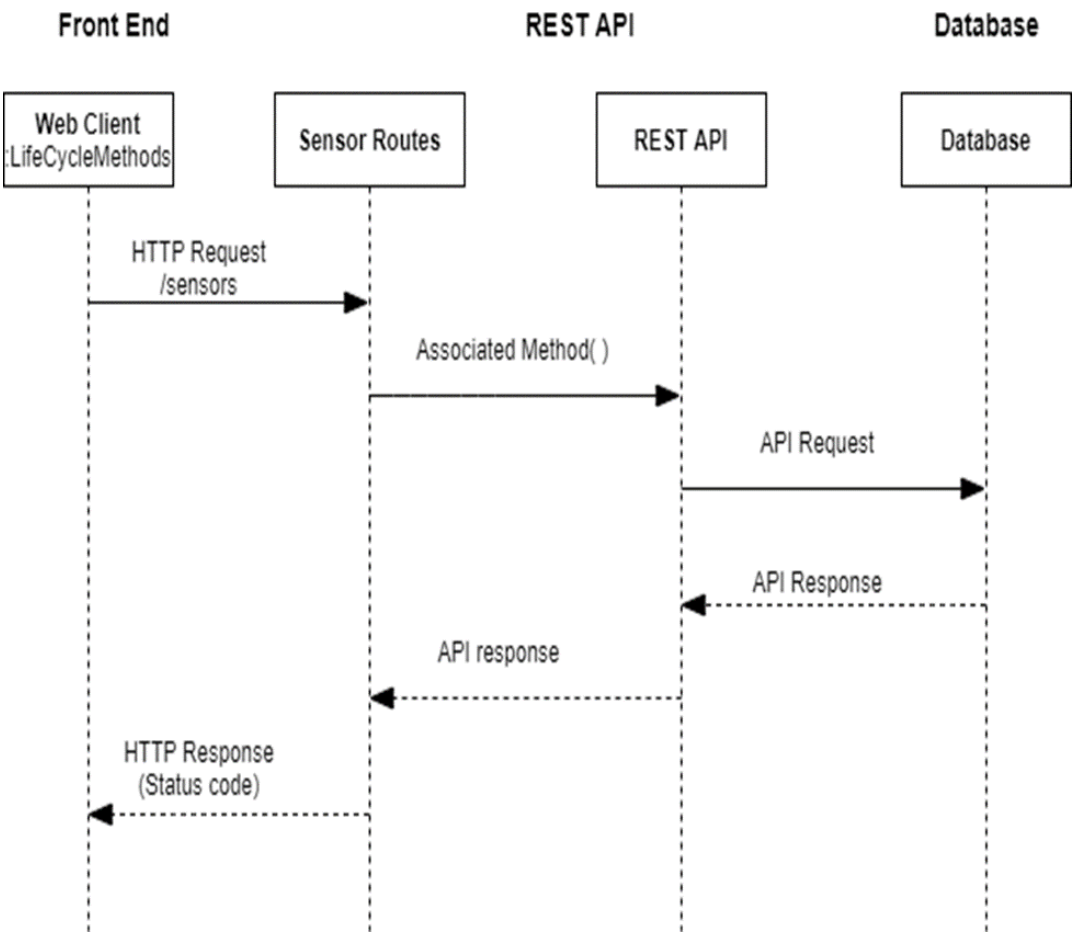
The Web Client Application is implemented using react framework, RESTful web services API is implemented using NodeJS and express server, RMI Server and an RMI desktop client is implemented using NetBeans , Dummy Sensor App implemented using Eclipse , and Java Email Service MySQL database are used by the system.

System architecture diagram

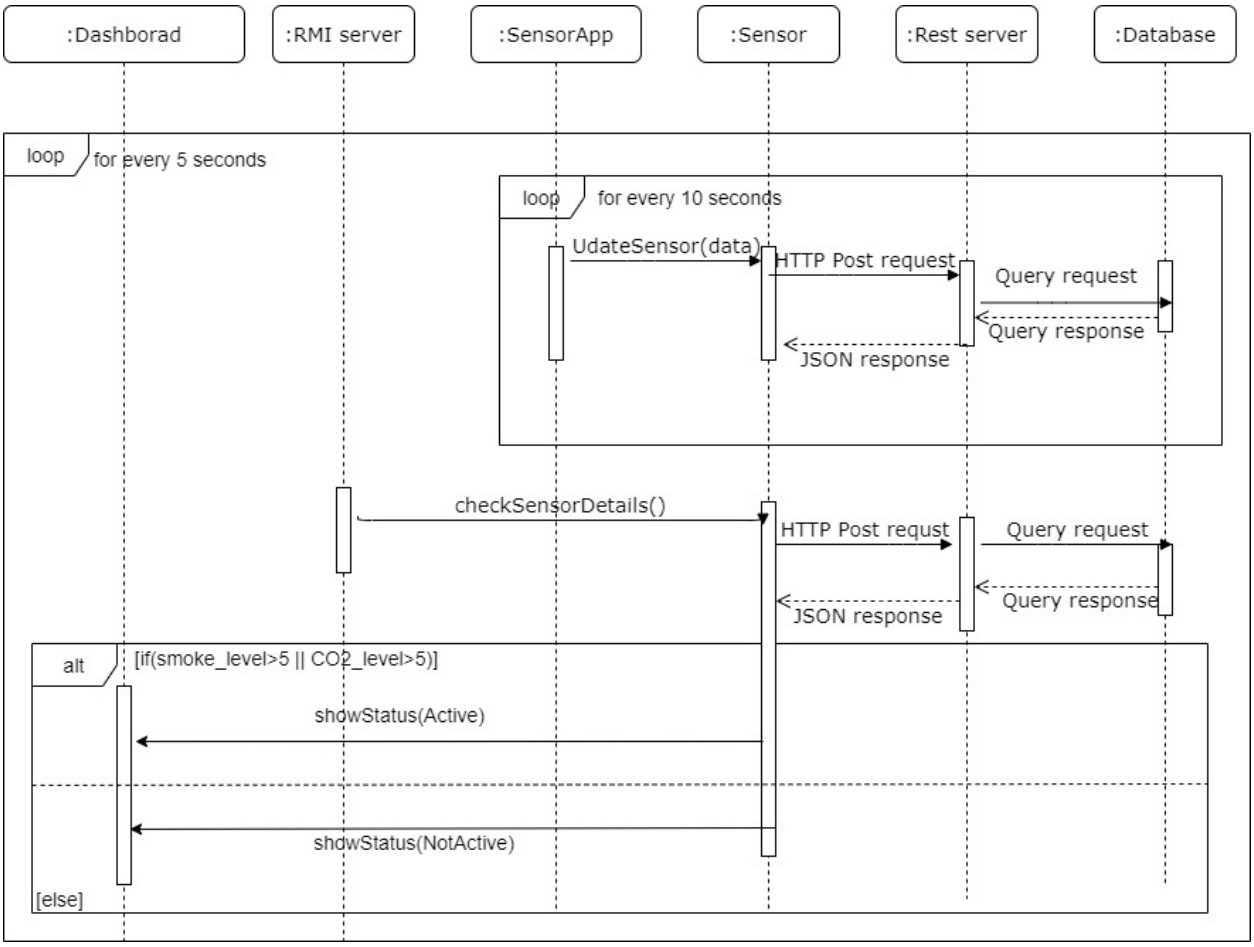


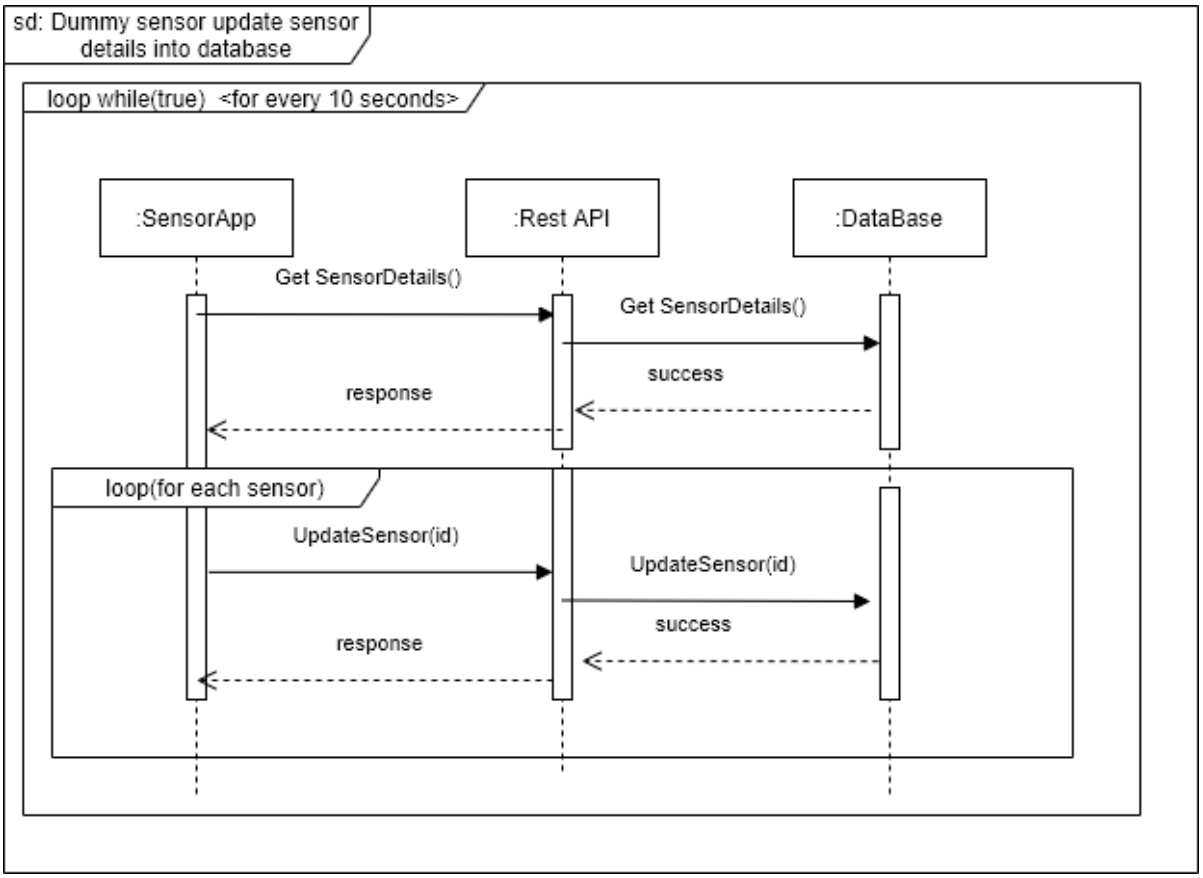
Sequence diagrams

1.Display details of sensors in web client

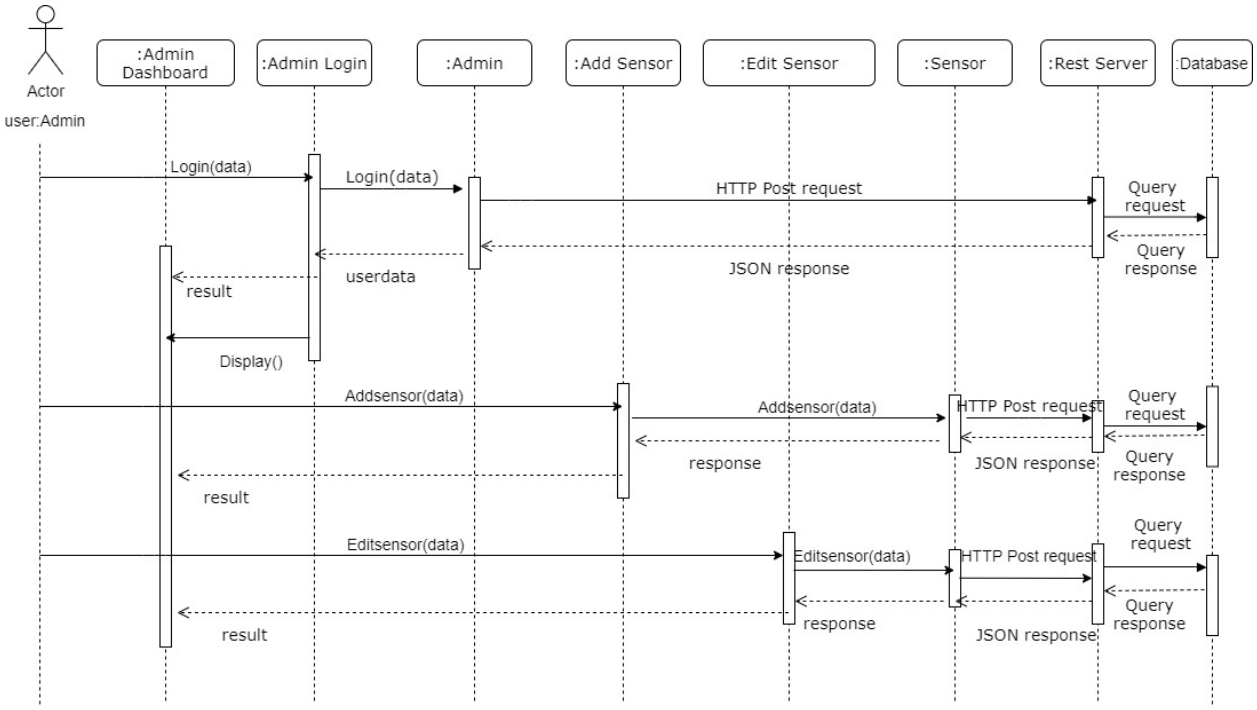


2. Update the sensor details in Database




sd: Dummy sensor update sensor details into database
loop while(true) <for every 10 seconds>
:SensorApp
:Rest API
:DataBase
Get SensorDetails()
Get SensorDetails()
success
response
loop(for each sensor)
UpdateSensor(id)
UpdateSensor(id)
success
response

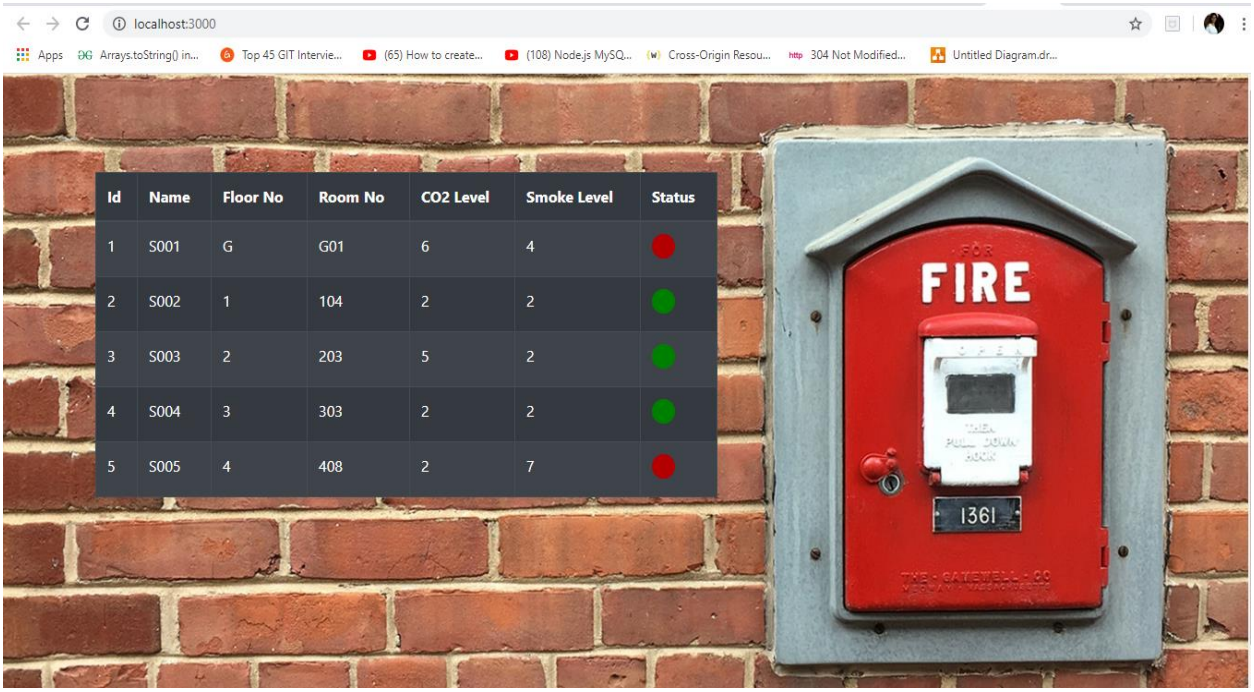
3. Managing Sensor Details



System Interfaces

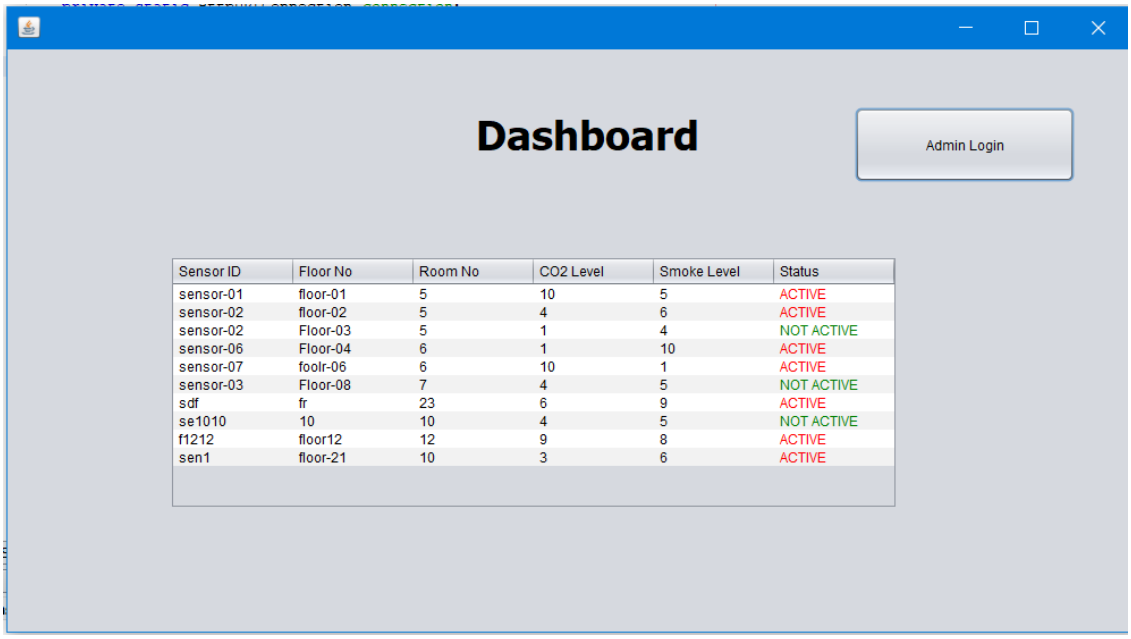
Web Client

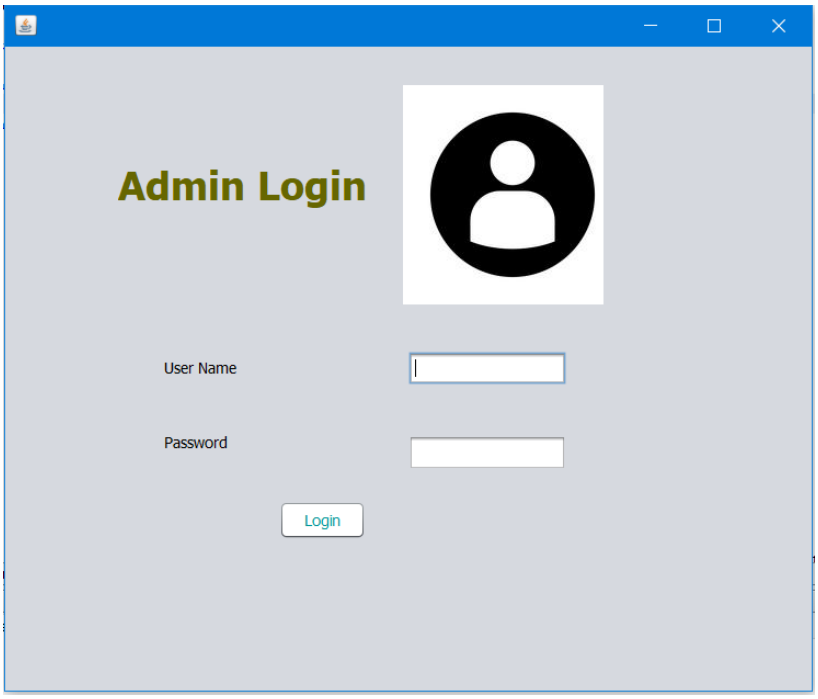
Web client is a JavaScript application which is implemented using React JS, CSS and React-bootstrap libraries. The main functionality of the web client is displaying sensor details including the status of the sensor. Using React Life cycle methods front end sends API calls to rest API in every 40 seconds and updating the front end application with constantly changing sensor details. If the smoke level or CO2 level is being checked for every 40 seconds and in any case, if the value of smoke level or CO2 level is more than 5 the status is shown in red color. Unless the status is in green color. This web client is running on port 3000.

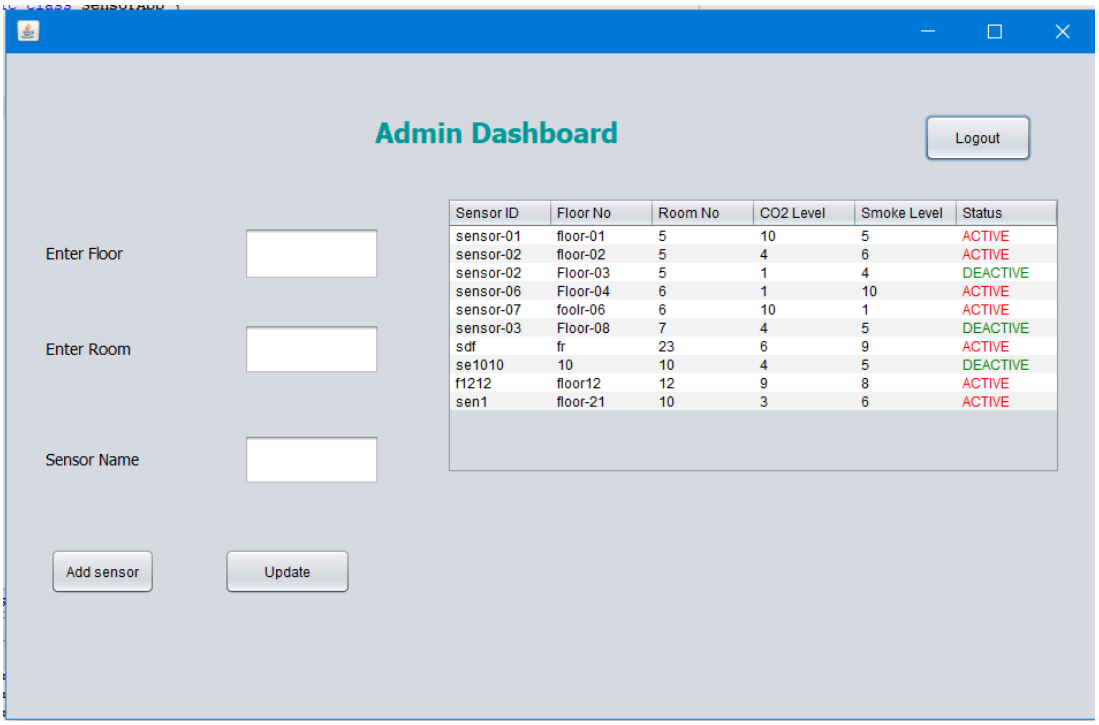


Id	Name	Floor No	Room No	CO2 Level	Smoke Level	Status
1	S001	G	G01	6	4	
2	S002	1	104	2	2	
3	S003	2	203	5	2	
4	S004	3	303	2	2	
5	S005	4	408	2	7	

Admin Dashboard







Email Message

Fire Alarm Activated Inbox ×

dsfirealarmsystem123@gmail.com
to me

Warning!!! The Floor Number: floor12, The Room Number: 12 alarm has been activated. Please call 101 for Fire Emergency Services

80

dsfirealarmsystem123@gmail.com
Warning!!! The Floor Number: 10, The Room Number: 10 alarm has been activated. Please call 101 for Fire Emergency Services

dsfirealarmsystem123@gmail.com
to me

Warning!!! The Floor Number: fr, The Room Number: 23 alarm has been activated. Please call 101 for Fire Emergency Services

Security Measures

Dashboard login:

Username : admin

Password : admin123

Appendix

RMI Server – Server.java

```
/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package service;
import Controller.httpJson;
import Models.getAllData;
import java.rmi.RemoteException;
import java.rmi.registry.LocateRegistry;
import java.rmi.registry.Registry;
import java.rmi.server.UnicastRemoteObject;
import java.util.ArrayList;
import java.util.List;
import org.json.simple.parser.JSONParser;

import org.json.simple.JSONArray;
import org.json.simple.JSONObject;



public class Server extends UnicastRemoteObject implements ServiceInterface {

        private int count=0;
        private List<getAllData> AllDataList;

         public Server() throws RemoteException {
        super();
        GetAllData();

       }
```

```
 public synchronized int increment(){

                                        count++;
                                        return count;
                        }



@Override
public List<getAllData> allData() throws RemoteException {
    return AllDataList;
}

private void GetAllData() {


    AllDataList = new ArrayList<>();

    serverUpdate();

}
private void serverUpdate() {
    System.out.println("Server Update Function Execute----------------------");
    Thread thread = new Thread(new Runnable() {

        @Override
        public void run() {
            try {
                System.out.println("Server Thread Start----------------------");
                apiRequests();
                Thread.sleep(5000);
                serverUpdate();
            } catch (Exception e) {
```

```
                System.out.println("Thread Exception : " + e);
            }

        }
    }
    );

    thread.start();
}

    private void apiRequests() {
    System.out.println("Send API Requests");

    String fetchAllData_response = new httpJson().jsonRequest("http://localhost:5000/sensors/");

   // System.out.println(fetchAllData_response);
    if (fetchAllData_response != null) {
        try {
            JSONParser jsonParser = new JSONParser();
            JSONArray responseObj = (JSONArray) jsonParser.parse(fetchAllData_response);
            AllDataList.clear();
            for (Object obj : responseObj) {
                JSONObject jSONObject = (JSONObject) obj;
                AllDataList.add(
                        new getAllData(
                                jSONObject.get("id").toString(),
                                jSONObject.get("floor_no").toString(),
                                jSONObject.get("sensor_name").toString(),
                                jSONObject.get("smoke_level").toString(),
                                jSONObject.get("co2_level").toString(),
                                jSONObject.get("room_no").toString())
```

```
                );
            }
            System.out.println("Response from all getData: complete---------");
        } catch (Exception e) {
            System.out.println("Server Exception: " + e);
        }
    }
  }


  public static void main(String[] args) {
    try {
        Registry reg = LocateRegistry.createRegistry(9090);
        reg.rebind("Sensor_server", new Server());
        System.err.println("Server ready");

    } catch (Exception e) {
        System.err.println("Server exception: " + e.getMessage());
    }
  }
}
```

SensorApp.java

```
package service;
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.OutputStreamWriter;
import java.net.HttpURLConnection;
import java.net.MalformedURLException;
import java.net.URL;
import java.util.Random;
import org.json.JSONArray;
import org.json.JSONObject;

public class SensorApp {

	private static HttpURLConnection connection;

	public static void main(String[] args) {

		while (true) {

			BufferedReader reader;
			String line = "";
			StringBuffer responseContent = new StringBuffer();

			try {

				URL url = new URL("http://localhost:5000/sensors");
				connection = (HttpURLConnection) url.openConnection();
				connection.setRequestMethod("GET");
				// conn.setRequestProperty("Accept", "application/json");
```

```
                                        // Open a connection stream to the corresponding API.
                                        connection.connect();
                                        int responseCode = connection.getResponseCode();

                                        if (responseCode != 200)
                                                throw new RuntimeException("Http response code :" + responseCode);
                                        // from http response we get a inputstream

                                        else {
                                                reader = new BufferedReader(new
InputStreamReader(connection.getInputStream()));
                                                while ((line = reader.readLine()) != null) {
                                                        responseContent.append(line);
                                                        // System.out.println(responseContent.toString());
                                                }

                                                reader.close();

                                        }
                                        parse(responseContent.toString());

                                        SensorApp.ExpireTimer();

                                } catch (MalformedURLException e) {
                                        e.printStackTrace();

                                } catch (IOException e) {
                                        e.printStackTrace();
                                } finally {
                                        connection.disconnect();
                                }
                        } // end of while

                }
```

```
// method: to parse the http response data
public static String parse(String responseBody) {
        int id, room_no, co2_level, smoke_level;
        String floor_no,sensor_name;
        JSONArray array = new JSONArray(responseBody);

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

                int co2Level = 0, smokeLevel = 0, status = 0;
                Random co2Random = new Random();
                Random smokeRandom = new Random();

                smokeLevel = 1 + smokeRandom.nextInt(10);
                co2Level = 1 + co2Random.nextInt(10);

                JSONObject object = array.getJSONObject(i);
                id = object.getInt("id");
                floor_no = object.getString("floor_no");
                room_no = object.getInt("room_no");
                co2_level = object.getInt("co2_level");
                smoke_level = object.getInt("smoke_level");
      sensor_name=object.getString("sensor_name");
                try {
                        URL url = new URL("http://localhost:5000/sensors/" + id);
                        connection = (HttpURLConnection) url.openConnection();

                        connection.setDoOutput(true);
                        connection.setDoInput(true);
                        connection.setRequestMethod("PUT");
                        // connection.setFixedLengthStreamingMode(length);
                        connection.setRequestProperty("Content-Type", "application/json");
                        connection.setRequestProperty("Accept", "application/json");
```

```
                connection.connect();

                JSONObject jsonObject = new JSONObject();
                jsonObject.put("floor_no", floor_no);
                jsonObject.put("room_no", room_no);
                jsonObject.put("co2_level", co2Level);
                jsonObject.put("smoke_level", smokeLevel);
        jsonObject.put("sensor_name",sensor_name );

                //jsonObject.put("status", status);

                OutputStreamWriter wr = new
OutputStreamWriter(connection.getOutputStream());
                wr.write(jsonObject.toString());
                wr.flush();

                StringBuilder sb = new StringBuilder();
                int HttpResult = connection.getResponseCode();
                if (HttpResult == HttpURLConnection.HTTP_OK) {
                        BufferedReader br = new BufferedReader(new
InputStreamReader(connection.getInputStream(), "utf-8"));
                        String line1 = null;
                        while ((line1 = br.readLine()) != null) {
                                sb.append(line1 + "\n");
                        }
                        br.close();
                        System.out.println("" + sb.toString());
                } else {
                        System.out.println(connection.getResponseMessage());
                }

        } catch (MalformedURLException e) {
                // TODO Auto-generated catch block
```

```
                    e.printStackTrace();
                } catch (IOException e) {
                    // TODO Auto-generated catch block
                    e.printStackTrace();
                } finally {
                    connection.disconnect();
                }

            }

            return null;

        }

        public static void ExpireTimer() {

            try {
                Thread.sleep(10000);

            } catch (Exception e) {

            }

        }

}
```

ServiceInterface.java

```
/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
```

```
 * and open the template in the editor.
 */
package service;

import Models.getAllData;
import java.rmi.Remote;
import java.rmi.RemoteException;
import java.util.List;



public interface ServiceInterface extends Remote{

    List<getAllData> allData() throws RemoteException;
    public int increment() throws RemoteException;
}
```

Client.java

```
/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package service;

import java.net.MalformedURLException;
import java.rmi.NotBoundException;
import java.rmi.RemoteException;
import java.rmi.registry.LocateRegistry;
import java.rmi.registry.Registry;
```

```
public class client {

   private static ServiceInterface s1;

    public static void main(String[] args) throws MalformedURLException, RemoteException, NotBoundException
{
       client Client = new client();
       Client.connectRemote();
       System.out.println("User Count: "+s1.increment());

   }

 private void connectRemote() throws RemoteException, NotBoundException {

       Registry reg = LocateRegistry.getRegistry("localhost", 9090);
       s1 =(ServiceInterface) reg.lookup("Sensor_server");

   }

}
```

Models

Floor.java

```
/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package Models;

import java.io.Serializable;
```

```
public class Floor implements Serializable{

    public int Id;
    public String No;
    public int User;

    public int getId() {
        return Id;
    }

    public void setId(int Id) {
        this.Id = Id;
    }

    public String getNo() {
        return No;
    }

    public void setNo(String No) {
        this.No = No;
    }

    public int getUser() {
        return User;
    }

    public void setUser(int User) {
        this.User = User;
    }
```

```
}
```

Room.java

```
package Models;

import java.io.Serializable;



public class Room implements Serializable{

    public int roomId;
    public String roomNo;
    public int userID;
    public int floorID;

    public int getRoomId() {
        return roomId;
    }

    public void setRoomId(int roomId) {
        this.roomId = roomId;
    }

    public String getRoomNo() {
        return roomNo;
    }

    public void setRoomNo(String roomNo) {
        this.roomNo = roomNo;
    }
```

```
    public int getUserID() {
        return userID;
    }

    public void setUserID(int userID) {
        this.userID = userID;
    }

    public int getFloorID() {
        return floorID;
    }

    public void setFloorID(int floorID) {
        this.floorID = floorID;
    }


}
```

getAllData.java

```
/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package Models;

import java.io.Serializable;


public class getAllData implements Serializable{
```

```
private String id;
private String floor;
private String sensor;
private String smoke_level;
private String co2_level;
private String room;
private String status_val;
private String status_color;


 public getAllData( String id,String floor, String sensor, String smoke_level, String co2_level, String room ) {

    this.id=id;
    this.floor = floor;
    this.sensor = sensor;
    this.smoke_level = smoke_level;
    this.co2_level = co2_level;
    this.room = room;



 }

public String getId() {
    return id;
}

public void setId(String id) {
    this.id = id;
}
```

```
public String getFloor() {
    return floor;
}

public void setFloor(String floor) {
    this.floor = floor;
}

public String getSensor() {
    return sensor;
}

public void setSensor(String sensor) {
    this.sensor = sensor;
}

public String getSmoke_level() {
    return smoke_level;
}

public void setSmoke_level(String smoke_level) {
    this.smoke_level = smoke_level;
}

public String getCo2_level() {
    return co2_level;
}

public void setCo2_level(String co2_level) {
    this.co2_level = co2_level;
}

public String getRoom() {
```

```
        return room;
    }

    public void setRoom(String room) {
        this.room = room;
    }

    public String getStatus_val() {
        return status_val;
    }

    public void setStatus_val(String status_val) {
        this.status_val = status_val;
    }

    public String getStatus_color() {
        return status_color;
    }

    public void setStatus_color(String status_color) {
        this.status_color = status_color;
    }

}
```

Sensor.java

```
/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package Models;
```

```
import java.io.Serializable;


public class sensor implements Serializable{

    public int sensorID;
    public String sensorName;
    public String sensorDescription;
    public int smoke_value;
    public int co2_value;


   public sensor(int sensorID, String sensorName, String sensorDescription,  int smoke_value, int co2_value) {
        this.sensorID = sensorID;
        this.sensorName = sensorName;
        this.sensorDescription = sensorDescription;
        this.smoke_value = smoke_value;
        this.co2_value = co2_value;
    }

    public int getSensorID() {
        return sensorID;
    }

    public void setSensorID(int sensorID) {
        this.sensorID = sensorID;
    }

    public String getSensorName() {
        return sensorName;
    }

    public void setSensorName(String sensorName) {
```

```
        this.sensorName = sensorName;
    }

    public String getSensorDescription() {
        return sensorDescription;
    }

    public void setSensorDescription(String sensorDescription) {
        this.sensorDescription = sensorDescription;
    }

    public int getSmoke_value() {
        return smoke_value;
    }

    public void setSmoke_value(int smoke_value) {
        this.smoke_value = smoke_value;
    }

    public int getCo2_value() {
        return co2_value;
    }

    public void setCo2_value(int co2_value) {
        this.co2_value = co2_value;
    }

}
```

Admin Dashboard – Dashboard.java

```
/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package views;

import Common.Email_Send;
import Models.getAllData;
import Models.sensor;
import java.awt.event.ActionEvent;
import java.rmi.NotBoundException;
import java.rmi.RemoteException;
import java.rmi.registry.LocateRegistry;
import java.rmi.registry.Registry;
import java.util.ArrayList;
import java.util.List;
import java.util.Vector;
import java.util.logging.Level;
import java.util.logging.Logger;
import javax.swing.JTable;
import javax.swing.table.DefaultTableModel;
import service.ServiceInterface;



public class Dashboard extends javax.swing.JFrame {


    private static ServiceInterface s1;


    public Dashboard() {
```

```
        initComponents();

        try {
            GetDataToTable();
        } catch (RemoteException ex) {
            Logger.getLogger(Dashboard.class.getName()).log(Level.SEVERE, null, ex);
        } catch (NotBoundException ex) {
            Logger.getLogger(Dashboard.class.getName()).log(Level.SEVERE, null, ex);
        }
    }

    /**
     * This method is called from within the constructor to initialize the form.
     * WARNING: Do NOT modify this code. The content of this method is always
     * regenerated by the Form Editor.
     */
    @SuppressWarnings("unchecked")
    // <editor-fold defaultstate="collapsed" desc="Generated Code">
    private void initComponents() {

        jScrollPane1 = new javax.swing.JScrollPane();
        jTable1 = new javax.swing.JTable();
        jButton1 = new javax.swing.JButton();
        jLabel1 = new javax.swing.JLabel();

        setDefaultCloseOperation(javax.swing.WindowConstants.EXIT_ON_CLOSE);

        jTable1.setModel(new javax.swing.table.DefaultTableModel(
            new Object [][] {

            },
            new String [] {
                "Sensor ID", "Floor No", "Room No", "CO2 Level", "Smoke Level", "Status"
```

```
        }
    ));
    jScrollPane1.setViewportView(jTable1);

    jButton1.setText("Admin Login");
    jButton1.addActionListener(new java.awt.event.ActionListener() {
        public void actionPerformed(java.awt.event.ActionEvent evt) {

            jButton1ActionPerformed(evt);
        }
    });

    jLabel1.setFont(new java.awt.Font("Tahoma", 1, 36)); // NOI18N
    jLabel1.setText("Dashboard");

    javax.swing.GroupLayout layout = new javax.swing.GroupLayout(getContentPane());
    getContentPane().setLayout(layout);
    layout.setHorizontalGroup(
        layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
        .addGroup(javax.swing.GroupLayout.Alignment.TRAILING, layout.createSequentialGroup()
            .addContainerGap(javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)
            .addComponent(jLabel1, javax.swing.GroupLayout.PREFERRED_SIZE, 316,
javax.swing.GroupLayout.PREFERRED_SIZE)
            .addGap(18, 18, 18)
            .addComponent(jButton1, javax.swing.GroupLayout.PREFERRED_SIZE, 194,
javax.swing.GroupLayout.PREFERRED_SIZE)
            .addGap(51, 51, 51))
        .addGroup(layout.createSequentialGroup()
            .addGap(143, 143, 143)
            .addComponent(jScrollPane1, javax.swing.GroupLayout.PREFERRED_SIZE, 642,
javax.swing.GroupLayout.PREFERRED_SIZE)
            .addContainerGap(209, Short.MAX_VALUE))
    );
    layout.setVerticalGroup(
        layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
```

```
                .addGroup(layout.createSequentialGroup()
                    .addContainerGap(51, Short.MAX_VALUE)
                    .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)
                        .addComponent(jButton1, javax.swing.GroupLayout.PREFERRED_SIZE, 66,
javax.swing.GroupLayout.PREFERRED_SIZE)
                        .addComponent(jLabel1, javax.swing.GroupLayout.PREFERRED_SIZE, 62,
javax.swing.GroupLayout.PREFERRED_SIZE))
                    .addGap(65, 65, 65)
                    .addComponent(jScrollPane1, javax.swing.GroupLayout.PREFERRED_SIZE, 223,
javax.swing.GroupLayout.PREFERRED_SIZE)
                    .addGap(108, 108, 108))
            );

        setSize(new java.awt.Dimension(1010, 560));
        setLocationRelativeTo(null);
    }// </editor-fold>

    /**
     * @param args the command line arguments
     */
     private void jButton1ActionPerformed(java.awt.event.ActionEvent evt) {
        AdminLogin login = new AdminLogin();
        this.setVisible(false);
        login.setVisible(true);
    }




    public static void main(String args[]) {
        /* Set the Nimbus look and feel */
        //<editor-fold defaultstate="collapsed" desc=" Look and feel setting code (optional) ">
        /* If Nimbus (introduced in Java SE 6) is not available, stay with the default look and feel.
         * For details see http://download.oracle.com/javase/tutorial/uiswing/lookandfeel/plaf.html
         */
        try {
```

```
            for (javax.swing.UIManager.LookAndFeelInfo info : javax.swing.UIManager.getInstalledLookAndFeels()) {
                if ("Nimbus".equals(info.getName())) {
                    javax.swing.UIManager.setLookAndFeel(info.getClassName());
                    break;
                }
            }
        } catch (ClassNotFoundException ex) {
            java.util.logging.Logger.getLogger(Dashboard.class.getName()).log(java.util.logging.Level.SEVERE, null,
ex);
        } catch (InstantiationException ex) {
            java.util.logging.Logger.getLogger(Dashboard.class.getName()).log(java.util.logging.Level.SEVERE, null,
ex);
        } catch (IllegalAccessException ex) {
            java.util.logging.Logger.getLogger(Dashboard.class.getName()).log(java.util.logging.Level.SEVERE, null,
ex);
        } catch (javax.swing.UnsupportedLookAndFeelException ex) {
            java.util.logging.Logger.getLogger(Dashboard.class.getName()).log(java.util.logging.Level.SEVERE, null,
ex);
        }
        //</editor-fold>

        /* Create and display the form */
        java.awt.EventQueue.invokeLater(new Runnable() {
            public void run() {
                new Dashboard().setVisible(true);
            }
        });
    }

    // Variables declaration - do not modify
    private javax.swing.JButton jButton1;
    private javax.swing.JLabel jLabel1;
    private javax.swing.JScrollPane jScrollPane1;
    private javax.swing.JTable jTable1;
    // End of variables declaration
```

```
    List<getAllData> allDatasExists;
private void GetDataToTable() throws RemoteException, NotBoundException {

        allDatasExists = new ArrayList<>();

        new Thread(new Runnable() {

            @Override
            public void run() {
                try {
                    Registry reg = LocateRegistry.getRegistry("localhost", 9090);
                    s1 = (ServiceInterface) reg.lookup("Sensor_server");
                    while (true) {

                        List<getAllData> allDatas = s1.allData();
                        DefaultTableModel dtm = (DefaultTableModel) jTable1.getModel();
                        dtm.setRowCount(0);

                        int i = 0;

                        for (getAllData data : allDatas) {

                            Vector v = new Vector();
                            v.add(data.getSensor());
                            v.add(data.getFloor());
                            v.add(data.getRoom());
                             v.add(data.getCo2_level());
                            v.add(data.getSmoke_level());

                           // v.add(sensor.getStatus_val());
                           // v.add((sensor.getStatus_val().equals("Active")) ? "<html><p
style='color:red'>ACTIVE</p></html>" : "<html><p style='color:green'>DEACTIVE</p></html>");
```

```
                    int n=Integer.parseInt(data.getCo2_level());
                    int m=Integer.parseInt(data.getSmoke_level());
                   if(n >5  || m>5){
                      String user_email="vinothvino1478@gmail.com";
                    v.add("<html><p style='color:red'>ACTIVE</p></html>");
                     new Email_Send().sendEmail("Fire Alarm Activated ", user_email, "Warning!!! The Floor Number: "+data.getFloor()+",  The Room Number: " + data.getRoom() + " alarm has been activated. Please call 101 for Fire Emergency Services");
                          System.out.println("Email sent");
                          System.out.println("SMS sent");

                   }
                   else{
                     v.add("<html><p style='color:green'>NOT ACTIVE</p></html>");
                   }

                    dtm.addRow(v);



                    }

                            Thread.sleep(15000);


                } } catch (Exception e) {
                System.out.println(e);
              }
            }
          }).start();

      }}
```

Web Client

App.js

```
import React, { Component } from "react";
import './App.css';
import { Table } from "react-bootstrap";
import 'bootstrap/dist/css/bootstrap.min.css';

class App extends Component {
  constructor(props) {
    super(props);

     /**
      * Initial State
      */
    this.state = {
      sensors: [],
      isLoaded: false,
      status:false
    }
  }

  /**
   * Fetch the data after initial render by HTTP GRT request
   */
  componentDidMount() {
      fetch('http://localhost:5000/sensors')
      .then(res => res.json())
      .then(json => {
        this.setState({
          isLoaded: true,
          sensors: json,
        })
        console.log(this.state.sensors);
      });
   }

   /**
    * Fetch the data in 40 second time intervals by HTTP GET request
    */
   componentWillMount(){
     setInterval(() => {
      fetch('http://localhost:5000/sensors')
      .then(res => res.json())
```

```
            .then(json => {
              this.setState({
                isLoaded: true,
                sensors: json,
              })
              console.log(this.state.sensors);
            });
          }, 40000);
        }

        render() {
          var { sensors, isLoaded } = this.state;

          if (!isLoaded) {
            return <div>Loading....</div>
          }
          else
            return (<div className="main-layout">
              <div className="tableView">
                {/* The data retreiving from HTTP response are mapped into
                 the React Bootstrap table */}
                <Table responsive="sm" striped bordered hover variant="dark">
                  <thead>
                    <tr>
                      <th>Id</th>
                      <th>Name</th>
                      <th>Floor No</th>
                      <th>Room No</th>
                      <th>CO2 Level</th>
                      <th>Smoke Level</th>
                      <th>Status</th>
                    </tr>
                  </thead>
                  <tbody>
                    {sensors.map(sensor => (
                      <tr>
                        <td>{sensor.id}</td>
                        <td>{sensor.name}</td>
                        <td>{sensor.floor_no}</td>
                        <td>{sensor.room_no}</td>
                        <td>{sensor.co2_level}</td>
                        <td>{sensor.smoke_level}</td>
                        <td>{((sensor.co2_level>5) || (sensor.smoke_level>5) )? <span className=
"reddot"></span> : <span className="greendot"></span>}</td>
                      </tr>
```

```
                ))}
              </tbody>
            </Table>
          </div>
        </div>
        )
    }
}

export default App;
```

App.css

```
/* CSS file for adding styles */

@media All and (min-width: 360px) {
    .main-layout {
        background: url("background.jpg");
        width: 100%;
        height: 655px;
        background-position: center;
        background-repeat: no-repeat;
        background-size: cover;
        -moz-background-size: cover;
        -webkit-background-size: cover;
    }
    .tableView {
        width: 50%;
        position: absolute;
        left: 100px;
        top: 100px;
    }
    .reddot {
        height: 25px;
        width: 25px;
        background-color: #b30000;
        border-radius: 50%;
        display: inline-block;
    }
    .greendot {
        height: 25px;
        width: 25px;
```

```
            background-color: green;
            border-radius: 50%;
            display: inline-block;
        }
    }
```

Package.json

```
{
  "name": "web-client",
  "version": "0.1.0",
  "private": true,
  "dependencies": {
    "@material-ui/core": "^4.9.12",
    "@testing-library/jest-dom": "^4.2.4",
    "@testing-library/react": "^9.5.0",
    "@testing-library/user-event": "^7.2.1",
    "axios": "^0.19.2",
    "bootstrap": "^4.4.1",
    "react": "^16.13.1",
    "react-bootstrap": "^1.0.1",
    "react-dom": "^16.13.1",
    "react-scripts": "3.4.1",
    "react-table-6": "^6.11.0"
  },
  "scripts": {
    "start": "react-scripts start",
    "build": "react-scripts build",
    "test": "react-scripts test",
    "eject": "react-scripts eject"
  },
  "eslintConfig": {
    "extends": "react-app"
  },
  "browserslist": {
    "production": [
      ">0.2%",
      "not dead",
      "not op_mini all"
    ],
    "development": [
      "last 1 chrome version",
      "last 1 firefox version",
```

```
      "last 1 safari version"
    ]
  }
}
```

Rest API

1. Server.js

```
const express = require('express');
const bodyParser = require('body-parser');
const mysqlConnection = require('./connection.js');
const SeonsorRoutes = require('./routes/sensorRoutes.js');


const app = express();
app.use(bodyParser.json());

//all the requests comes to '/sensors' path will use the SeonsorRoutes and
    there handle all the requests and send the response  back
app.use('/sensors', SeonsorRoutes);


app.listen(5000);
```

2. connection.js

```
const mysql = require('mysql');

//configuration  for mysql db
var mysqlConnection = mysql.createConnection({
    host : "localhost",
    user : "root",
    password : "admin",
    database : "sensordb",
    multipleStatements : true

});

//connect to db
mysqlConnection.connect((err) =>{
    if(!err){
        console.log("DB Connected");

    }
```

```
    else{
        console.log("Connection  Failed");
    }
});

module.exports = mysqlConnection;
```

3. sensorRoutes.js

```
// this file have the routes for all sensor
//connection to the database
require('dotenv').config();
const express = require('express');
const mysqlConnection = require('../connection.js');


//create  router object for the app on which we are going to create all the route
s
const Router = express.Router();


//To Retrieve all Sensors details
Router.get('/', (req, res) => {
    // mysqlConnection.query("SELECT * from sensors", (err, rows, fields) => {
    mysqlConnection.query("SELECT * from sensors", (err, results) => {
        if (!err) {
            // res.send(rows);
            res.send(JSON.stringify(results));
        }
        else {
            console.log(err);
        }
    })
});


//To Retrieve a sensor by using id
Router.get('/:id', (req, res) => {
    mysqlConnection.query("select * from sensors where id=" + req.params.id + " "
, (error, results) => {

        if (error) throw error;
        console.log(results);
        res.send(JSON.stringify(results));
    });
});
```

```
//To delete a sensor by using id
Router.delete('/:id', (req, res) => {
    mysqlConnection.query("DELETE FROM  sensors  WHERE id =" + req.params.id + "
", (error, results) => {
        if (error) throw error;
        console.log(results);
        res.send(JSON.stringify(results));
    });
});


//To Insert A New Sensor
Router.post('/', (req, res) => {
    mysqlConnection.query('INSERT INTO sensors SET ?', req.body, (err, results, f
ields) => {
        if (err) throw err;
        res.end(JSON.stringify(results));
    });
});

//To Update a sensor by using id
Router.put('/:id', (req, res) => {
    mysqlConnection.query("update  sensors  SET name = ?, floor_no =?, room_no =
?,  co2_level = ?, smoke_level = ?  where id= ?", [req.body.name, req.body.floor_
no, req.body.room_no, req.body.co2_level, req.body.smoke_level, req.params.id], (
error, results) => {
        if (error) throw error;
        console.log(results);
        res.send(JSON.stringify(results));
    });
});


module.exports = Router;
```

Dummy Sensor Application

```
//Dummy Sensor App
/**
 * this class is used send 'GET' request to REST API and receive sensor details,
 * then change the co2 level and smoke level,
 * and send 'PUT' request to API to update the details.
 */
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.OutputStreamWriter;
import java.net.HttpURLConnection;
import java.net.MalformedURLException;
import java.net.URL;
import java.util.Random;
import org.json.JSONArray;
import org.json.JSONObject;

public class SensorApp {

	private static HttpURLConnection connection;

	public static void main(String[] args) {

		while (true) {

			BufferedReader reader;
			String line = "";
			StringBuffer responseContent = new StringBuffer();

			try {

				URL url = new URL("http://localhost:5000/sensors");
				connection = (HttpURLConnection) url.openConnection();
				connection.setRequestMethod("GET");

				connection.connect();
				int responseCode = connection.getResponseCode();

				if (responseCode != 200)
					throw new RuntimeException("Http response code :" +
responseCode);


				else {
					reader = new BufferedReader(new
InputStreamReader(connection.getInputStream()));
					while ((line = reader.readLine()) != null) {
						responseContent.append(line);
						//
System.out.println(responseContent.toString());
					}
```

```
                        reader.close();
                    }
                    parse(responseContent.toString());
                    SensorApp.ExpireTimer();

                } catch (MalformedURLException e) {
                    e.printStackTrace();

                } catch (IOException e) {
                    e.printStackTrace();
                } finally {
                    connection.disconnect();
                }
            }

        }


        public static String parse(String responseBody) {
            int id, room_no, co2_level, smoke_level;
            String floor_no, name;

            JSONArray array = new JSONArray(responseBody);

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

                int co2Level = 0, smokeLevel = 0, status = 0;
                Random co2Random = new Random();
                Random smokeRandom = new Random();

                smokeLevel = 1 + smokeRandom.nextInt(10);
                co2Level = 1 + co2Random.nextInt(10);

                JSONObject object = array.getJSONObject(i);
                id = object.getInt("id");
                name = object.getString("name");
                floor_no = object.getString("floor_no");
                room_no = object.getInt("room_no");
                co2_level = object.getInt("co2_level");
                smoke_level = object.getInt("smoke_level");


                try {
                    URL url = new URL("http://localhost:5000/sensors/" + id);
                    connection = (HttpURLConnection) url.openConnection();

                    connection.setDoOutput(true);
                    connection.setDoInput(true);
                    connection.setRequestMethod("PUT");
                    connection.setRequestProperty("Content-Type",
"application/json");
                    connection.setRequestProperty("Accept",
"application/json");
                    connection.connect();
```

```
                        JSONObject jsonObject = new JSONObject();
                        jsonObject.put("name", name);
                        jsonObject.put("floor_no", floor_no);
                        jsonObject.put("room_no", room_no);
                        jsonObject.put("co2_level", co2Level);
                        jsonObject.put("smoke_level", smokeLevel);


                        OutputStreamWriter wr = new
OutputStreamWriter(connection.getOutputStream());
                        wr.write(jsonObject.toString());
                        wr.flush();

                        StringBuilder sb = new StringBuilder();
                        int HttpResult = connection.getResponseCode();
                        if (HttpResult == HttpURLConnection.HTTP_OK) {
                              BufferedReader br = new BufferedReader(new
InputStreamReader(connection.getInputStream(), "utf-8"));
                              String line1 = null;
                              while ((line1 = br.readLine()) != null) {
                                    sb.append(line1 + "\n");
                              }
                              br.close();
                              System.out.println("" + sb.toString());
                        } else {
                              System.out.println(connection.getResponseMessage());
                        }

                  } catch (MalformedURLException e) {
                        // TODO Auto-generated catch block
                        e.printStackTrace();
                  } catch (IOException e) {
                        // TODO Auto-generated catch block
                        e.printStackTrace();
                  } finally {
                        connection.disconnect();
                  }
            }

            return null;
      }

      public static void ExpireTimer() {
            try {
                  Thread.sleep(10000);

            } catch (Exception e) {

            }}
}
}
```