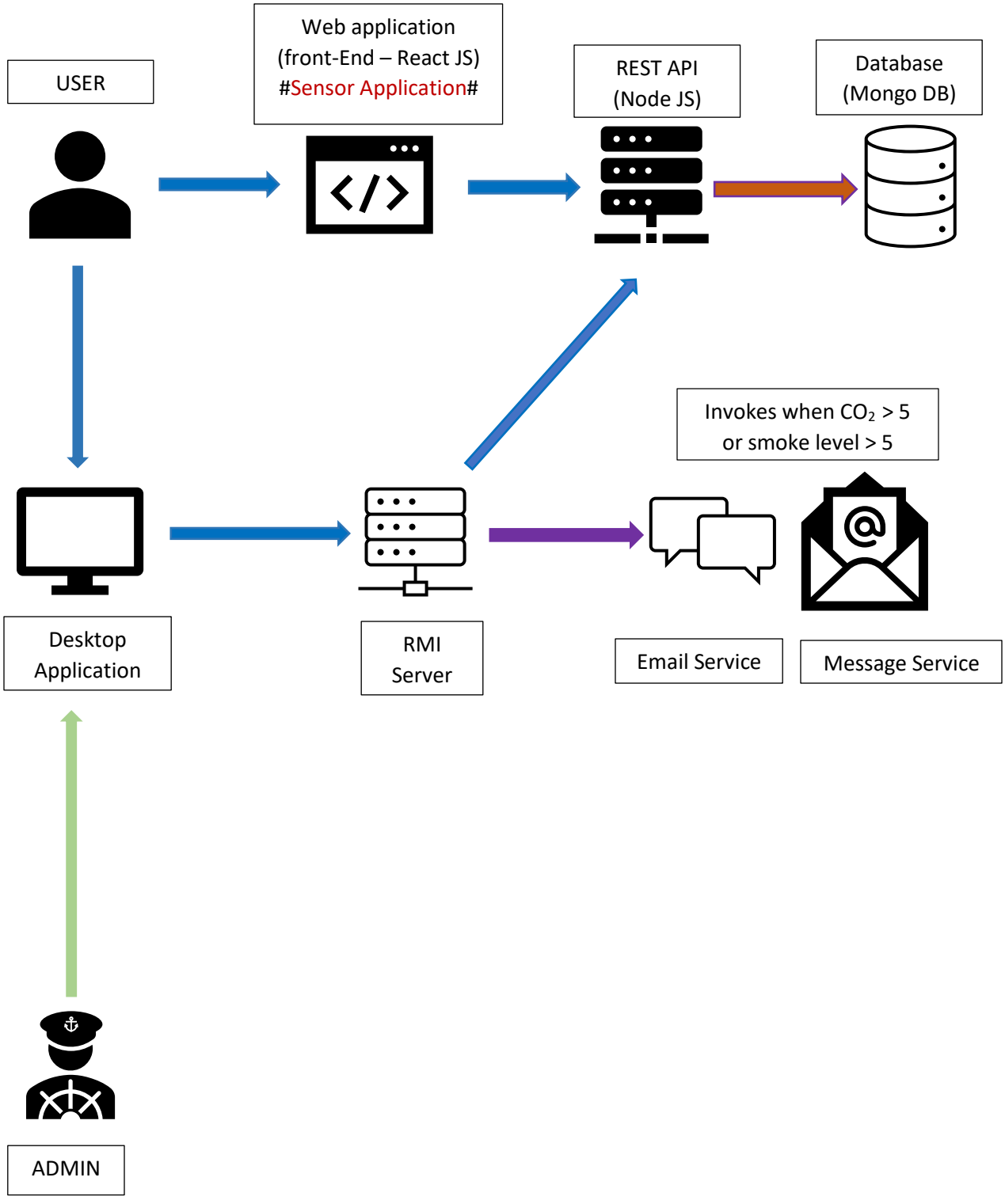
Fire Alarm Monitoring System

DS ASSIGNMENT 02

REPORT

IT NUMBER	NAME
IT18110012	Subasinghe C.
IT18112238	Dharmarathne R.S.C.K
IT18098242	Medagedara K.A
IT18111866	Nikapotha A.M.B.N

Architectural Diagram



Sequence Diagram for Web application of the fire alarm monitoring system

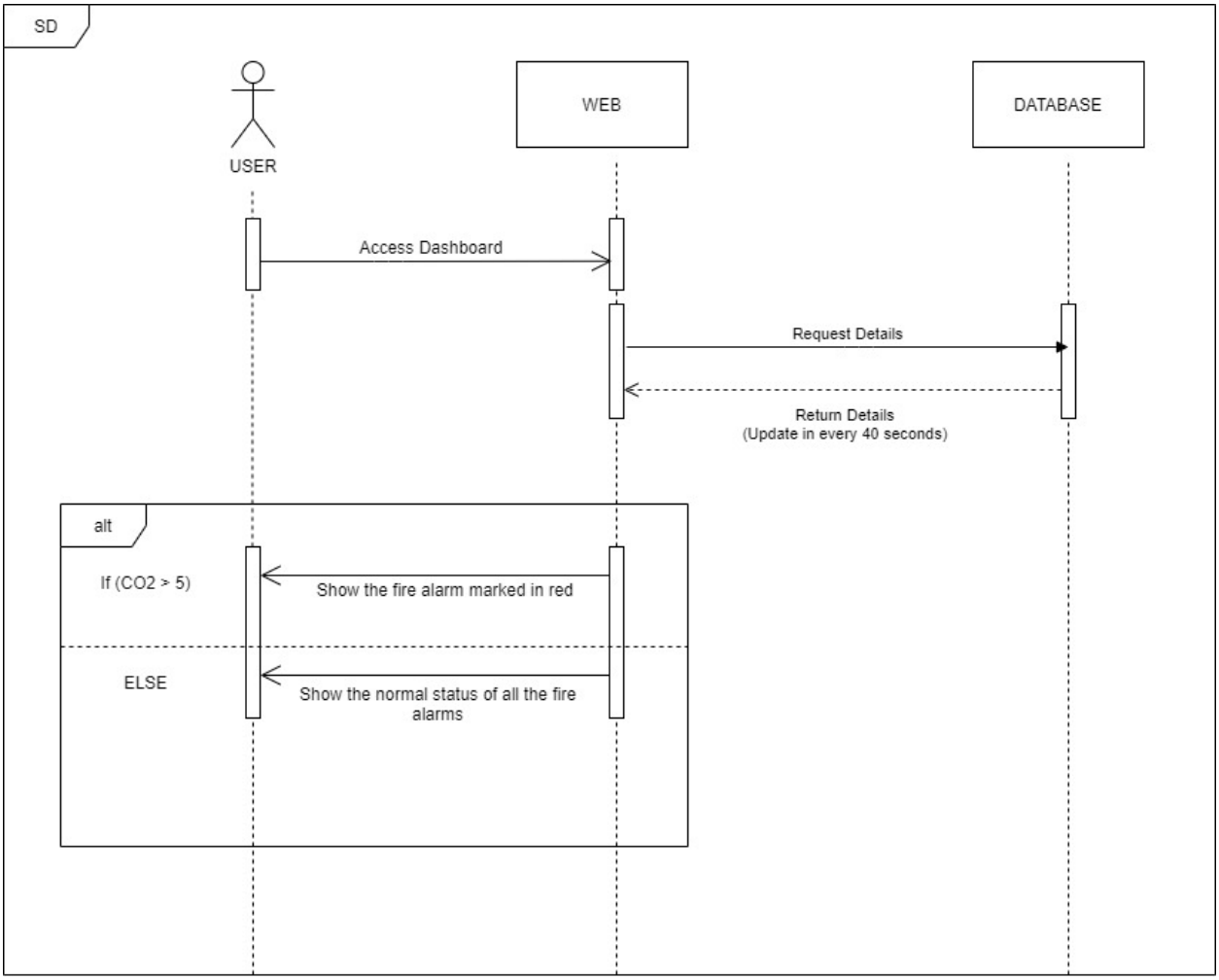


Fig 1

User can access the web application dashboard and request details. Web application dashboard show the Floor no, Room no, current Co2 level and the smoke level to the user. Web application shows the activated fire alarms in red. Web application is refreshing in every 40 seconds.

Sequence Diagram for Desktop application of the fire alarm monitoring system

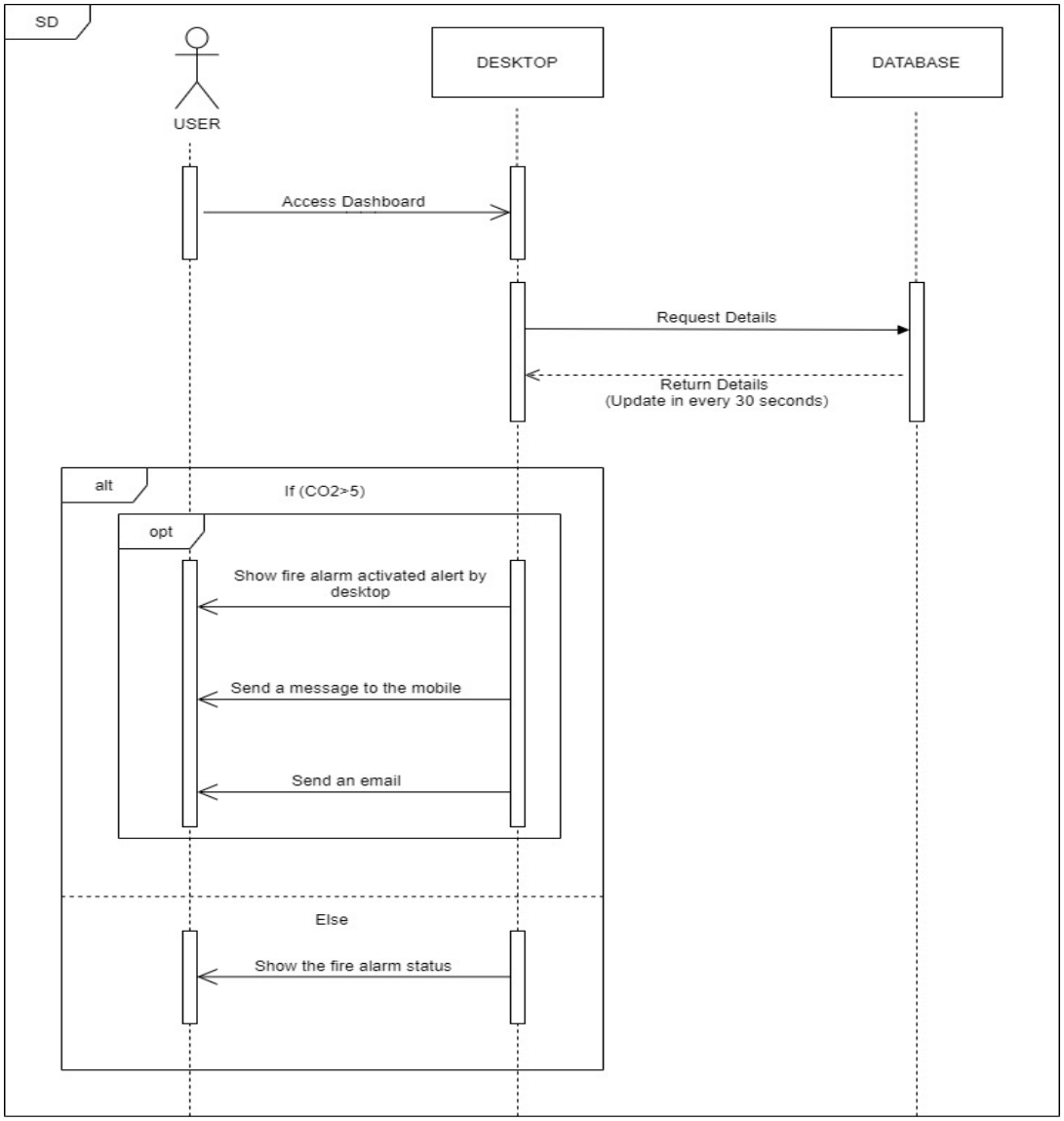


Fig 2

User can obtain fire alarm details through desktop also. Desktop also show the relevant floor no, room no, co2 level and current smoke level. Desktop application user can get the relevant state of the fire alarm active or not. If the fire alarm is active user get an email and a SMS. Fire alarm is activated when Co2 level above 5. Desktop application is refreshing in every 30 seconds.

Sequence diagram for Email service and Message Service

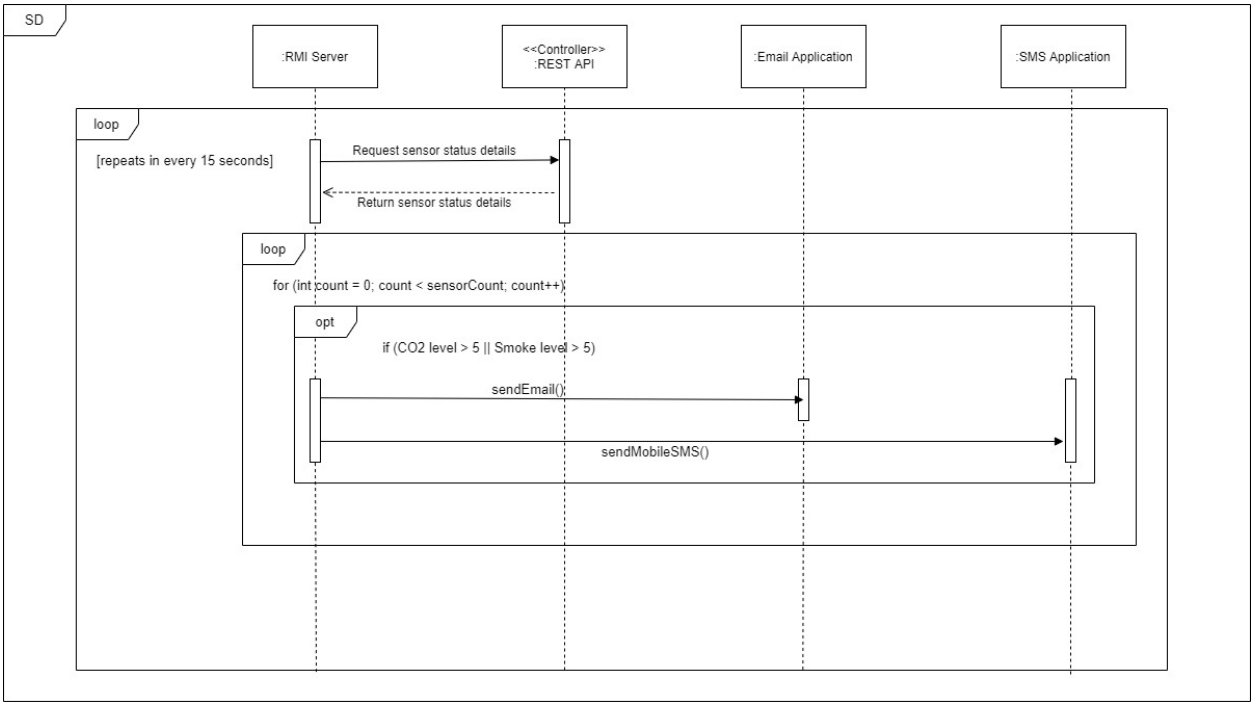


Fig 3

Email service and the message service is getting through the RMI server and these services are refreshing in every 15 seconds. This email and message service active when the co2 level and the smoke level is increased than 5.

Sequence diagram for Admin login and Admin tasks

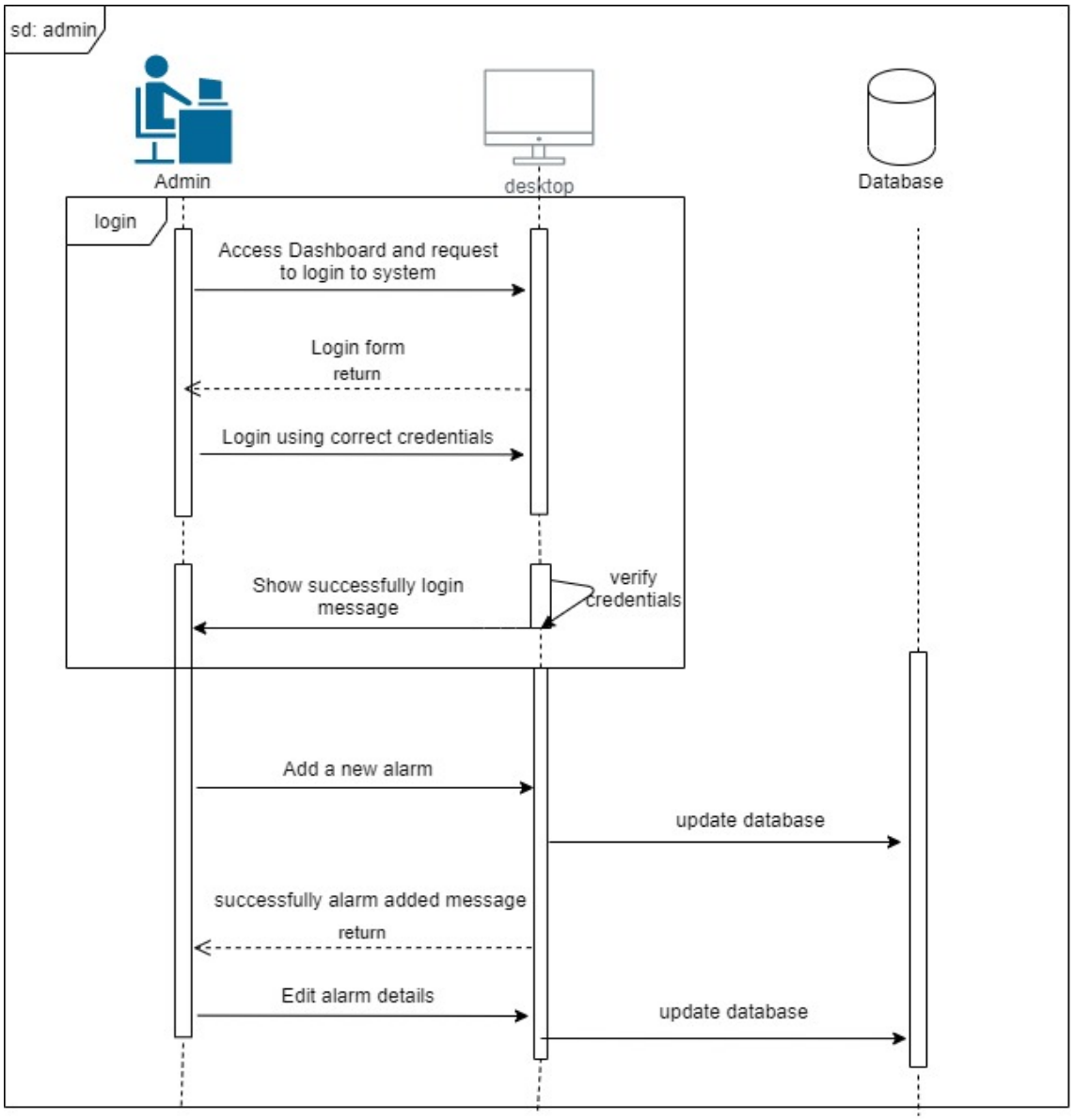


Fig 4

Admin can login to the system by providing valid credentials and can see the current states of the fire alarms. If admin provide incorrect credentials, then the desktop gives him a message that showing he enter incorrect details. Then admin login to the system by giving correct details he can edit fire alarm details and delete them.

Sequence Diagram for sensor application

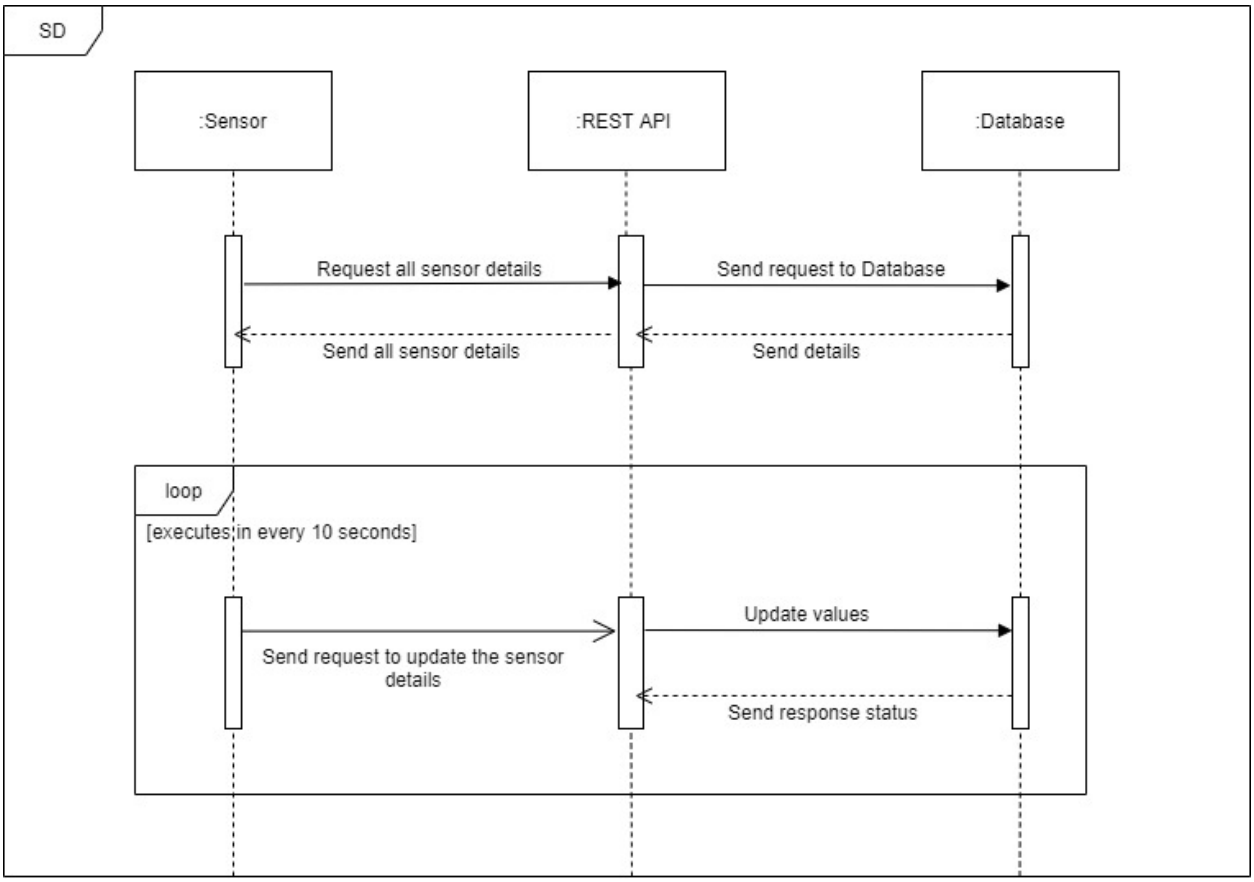


Fig 5

In sensor application first take all the sensor details for a one variable and pass it through a loop. Inside the loop we create random values. Then when looping occurs, sensor application gets various sensor values randomly.

APPENDIX

- Web Application (Front-end)

1. App.js

```
// importing dependencies
import React from 'react';
import {BrowserRouter as Router} from 'react-router-dom'
import axios from 'axios'

// importing components
import NavBar from './components/NavBar'
import IndexBody from './components/indexBody'

// class definition
class App extends React.Component{

// things should execute before load the components
componentDidMount(){


// SENSOR APPLICATIONS

// set interval function using asynchronous - every 10 seconds
setInterval(async () => {

// get response with API using await
const response = await axios.get('/api/sensors');

// assigning data using destructuring ES6 feature
const {data} = response

// for loop
for (let i = 0; i < data.length; i++) {

// create random values for carbon dioxide and smoke levels
const randomCarbonDioxideLevel = Math.floor(Math.random() * 10) + 1
const randomSmokeLevel = Math.floor(Math.random() * 10) + 1

// create new object using spread operation
const updatedSensor = {...data[i]}
```

```
// update properties
updatedSensor['carbonDioxideLevel'] = randomCarbonDioxideLevel;
updatedSensor['smokeLevel'] = randomSmokeLevel;

// send request to server to update every sensor

axios.put(`/api/sensors/${data[i]._id}`, updatedSensor)
}
},10000) // this will fire after every 10 seconds
}

// render function
render() {
return(
<div>
<Router>
<NavBar/>
<IndexBody />
</Router>
</div>
)
}
}

// export App
export default App;
```

2. NavBar.js

```
// importing libraries

import React from 'react'

import {Link} from 'react-router-dom'


// importing css

import css from '../public/css/navbar.css'
```

```
// export class
export default class NavBar extends React.Component {

   // render function
   render(){
      return(
         <nav class="navbar navbar-expand-lg navbar-dark">
            <Link class="navbar-brand" href="/">Fire Alarm Monitoring System</Link>
         </nav>
      )
   }
}
```

3. indexBody.js

```
// importing libraries
import React from 'react'
import axios from 'axios';

// importing css
import css from '../public/css/index.css'

// importing images
import helper from '../public/images/helper.jpg'

// functional components to fetch table rows
const Sensor = props => (
   <tr>
```

```
        <td>{props.item.name}</td>
        <td>{props.item.floorNumber}</td>
        <td>{props.item.roomNumber}</td>
        <td>{props.item.carbonDioxideLevel}</td>
        <td>{props.item.smokeLevel}</td>
        <td>
          <button className=
            {props.item.carbonDioxideLevel > 5 || props.item.smokeLevel > 5
            ? "btn btn-danger"
            : "btn btn-success"}
            >
              {props.item.carbonDioxideLevel > 5 || props.item.smokeLevel > 5
              ? "Activated"
              : "Normal"}
          </button>
        </td>
    </tr>
)

// class definition
export default class NavBar extends React.Component {

    // constructor
    constructor(props){
        super(props)

        // declaring this state
        this.state = {
            // all sensor objects goes here
            sensors: []
```

```
    }
}

// things should fire before render
componentDidMount (){

   // asynchronous function to fetch sensor details
   const getSensorInfo = async () => {
      // assigning response
      const response = await axios.get('/api/sensors')

      // get data using destructuring
      const {data} = response;

      // update this state
      this.setState({sensors: data})

      //getting boolean value using every callback function
      const isAny = data.every((sensor) => {
         return sensor.carbonDioxideLevel < 6 && sensor.smokeLevel < 6
      })

      if (isAny) {
         // assigning fire alert message
         const message = document.querySelector('#fireMessage')
         // its hidden property set to false then user can see the alert

         // set value to empty value
         const h4 = message.querySelector('h4');
         h4.innerText = '';
```

```
            // set value to empty value
            const p = message.querySelector('p');
            p.innerText = '';
        }else {
            // invoke the fireAlarmActivated
            this.fireAlarmActivated();
        }
    }

    // when page loaded first time this is going to execute
    getSensorInfo()

    // set interval for update the application
    setInterval(() => {
        // message
        console.log('this will execute after every 40 seconds!')
        // call the getSensorInfo function in every 40 seconds
        getSensorInfo();
    }, 40000);
}

// a function for rotate client image when emergency situation
fireAlarmActivated() {
    // assigning fire alert message
    const message = document.querySelector('#fireMessage')
    // its hidden property set to false then user can see the alert

    const h4 = message.querySelector('h4');
    h4.innerText = 'The fire alert has been activated!!';
```

```
      const p = message.querySelector('p');
      p.innerText = 'please leave the building by the nearest exit and go to the nearest assembly point immediately!!';

      // assigning image using javascript Document Object Model
      const image = document.querySelector('img');

      // rotate 180 degrees left
      image.style.transform = 'scale(-1,1)'

      // declare a setTimeOut function for rotate image again to 180 degrees right
      setTimeout(() => {
        // rotate 180 degrees right
        image.style.transform = 'scale(1,1)'
      },20000)
    }

    // fetch table rows from the Sensor functional component
    SensorsList(){
      return this.state.sensors.map(currentItem => {
        return <Sensor item={currentItem} key={currentItem._id}/>
      })
    }

    // render function
    render(){
      return(
        <div className="main">
          {/* alert message */}
```

```
            <div id="fireMessage" className="fireActivated">
              <h4 className="text-center"></h4>
              <br />
              <p className="text-center"></p>
            </div>
            <div className="container my-5">
                {/* client image */}
                <img src={helper} class="img-fluid rounded mx-auto d-block" alt="Responsive
image"></img>

                {/* all sensor details goes here */}
                <table class="table table-striped main">
                  <thead>
                    <tr>
                    <th scope="col">Sensor Name</th>
                    <th scope="col">Floor Number</th>
                    <th scope="col">Room Number</th>
                    <th scope="col">Carbon Dioxide Level</th>
                    <th scope="col">Smoke Level</th>
                    <th scope="col">Status</th>
                    </tr>
                  </thead>
                  <tbody>
                    {/* invoke sensorList function to fetch data */}
                    {this.SensorsList()}
                  </tbody>
                </table>
            </div>
          </div>
        )
```

```
  }
}
```

4. package.json

```
{
 "name": "fire-alarm-monitoring-system",
 "version": "0.1.0",
 "private": true,
 "dependencies": {
  "@testing-library/jest-dom": "^4.2.4",
  "@testing-library/react": "^9.5.0",
  "@testing-library/user-event": "^7.2.1",
  "axios": "^0.19.2",
  "react": "^16.13.1",
  "react-dom": "^16.13.1",
  "react-router-dom": "^5.1.2",
  "react-scripts": "3.4.1"
 },
 "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"
  ]
 },
 "proxy": "http://localhost:5000"
}
```

❖ Web Application (Back-end)

1. Index.js

```
// condition to require env file for developer purpose
if (process.env.NODE_ENV !== 'production') {
   require('dotenv').config();
}

// importing dependencies
const  express = require ('express')
const  mongoose  = require('mongoose')

// create app
const app = express();

// declaring port - default port = 5000
const PORT = process.env.PORT || 5000;

// using json
app.use(express.json());

// assigning database url from .env file
```

```
const databaseURL = process.env.MONGODB_URI;

// mongoose connection
mongoose.connect(databaseURL, {
   useNewUrlParser: true,
   useCreateIndex: true,
   useUnifiedTopology: true,
   useFindAndModify: false
});

// create connection
const connection = mongoose.connection;

// once connected to database, then console logging simple message
connection.once('open', () => {
   console.log('database connected!')
});

// requiring routes
const sensorRoute = require('./routes/sensor');
const userRoute = require('./routes/user')


// using routes
app.use(sensorRoute)
app.use(userRoute)

// port listener
app.listen(PORT,() => console.log(`Server running on PORT ${PORT}`))
```

2. Sensor.js – model

```
// importing dependencies

const mongoose = require('mongoose')


// create new schema

const Schema = mongoose.Schema;


// create new sensor schema
```

```
const sensorSchema = new Schema({
   name: {
      type: String,
      required: true,
      trim:true,
      lowercase: true
   },
   floorNumber: {
      type: Number,
      required: true
   },
   roomNumber: {
      type: Number,
      required:true,
      unique: true
   },
   carbonDioxideLevel: {
      type: Number,
      default: 0
   },

   smokeLevel: {
      type: Number,
      default: 0
   }
})
// compiling schema to model
const Sensor = mongoose.model('Sensor', sensorSchema);
// exporting sensor
module.exports = Sensor;
```

3. User.js - model

```
// importing dependencies
const mongoose = require('mongoose');

// create new schema
const Schema = mongoose.Schema;

// create new user schema
const userSchema = new Schema({
    username: {
        type: String,
        lowercase: true,
        trim: true,
        required: true
    },
    password: {
        type: String,
        trim: true,
        lowercase: true,
        required: true
    }
})

// compiling schema to model
const User = mongoose.model('User',userSchema)

// export user
module.exports = User
```

4. sensor.js – route

```
// importing dependencies
const express = require('express');

// create router
const router = new express.Router()

// importing Sensor model
const Sensor = require('../models/sensor')

// create route
router.post('/api/sensors', async (req, res) => {

    try {
        // create new sensor
        const sensor = new Sensor(req.body);

        // save it in db
        await sensor.save()

        // send response
        res.status(201).send(true)
    } catch (e) {
        // send response
        res.status(400).send(false)
    }
})

// read all route
```

```
router.get('/api/sensors', async (req, res) => {
    try {

        // assigning all sensors in the database
        const sensors = await Sensor.find({});

        // send response
        res.status(200).send(sensors)
    } catch (error) {
        // send response
        res.status(400).send(error.message)
    }
})

// read one sensor
router.get('/api/sensors/:id', async (req, res) => {

    // assigning provided id
    const _id = req.params.id

    try {

        // find specific sensor
        const sensor = await Sensor.findOne({_id})

        // condition
        if (!sensor) {
            // throw new error
            return res.status(404).send('not found')
        }
```

```
        // send response
        res.status(200).send(sensor)
    } catch (e) {
        // send response
        res.status(400).send(e.message)
    }
})

// edit route
router.put('/api/sensors/:id', async (req, res) => {

    // assigning provided id
    const _id = req.params.id

    try {

        // find specific sensor
        const sensor = await Sensor.findOneAndUpdate({_id},req.body)

        // condition
        if (!sensor) {
            // send response
            return res.status(404).send('not found')
        }

        // save back to the database
        await sensor.save()

        // send response
```

```
        res.status(200).send(true)
    } catch (e) {
        // send response
        res.status(400).send(false)
    }
})

// delete route
router.delete('/api/sensors/:id', async (req, res) => {
    // assigning provided id
    const _id =  req.params.id

    try {

        // delete sensor
        const sensor = await Sensor.findOneAndRemove({_id})

        // condition
        if (!sensor) {
            // return and send response
            return res.status(404).send('not found')
        }  // send response
        res.status(200).send(true)
 } catch (e) {
        // send response
        res.status(400).send(false)
    }
})
// exporting router
module.exports = router
```

5. User.js – route

```
// importing dependencies
const  express = require('express')

// create router
const router = new express.Router()
// importing User model
const User = require('../models/user')

// create login route
router.post('/api/users/login', async (req, res) => {
    try {
        // find user credentials
        const user = await User.findOne({username: req.body.username,password: req.body.password})
        // condition
        if (!user) {
            // send response
            return res.status(404).send(false);
        }
        // send response
        res.status(200).send(true)
    } catch (e) {
        // send response
        res.status(400).send(false)
    }
})

// export module
module.exports = router
```

6. env

```
MONGODB_URI="mongodb+srv://rmiApp:rmi_13@rmi-ugfst.mongodb.net/RMI?retryWrites=true"
```

7. package.json

```
{
  "name": "backend",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "keywords": [],
  "author": "",
  "license": "ISC",
  "dependencies": {
    "dotenv": "^8.2.0",
    "express": "^4.17.1",
    "mongoose": "^5.9.10"
  }
}
```

❖ Desktop Application – RMI and All the JAVA files

1. AddSensor.java

```
// importing dependencies
import java.awt.EventQueue;

import javax.swing.JFrame;
import javax.swing.JPanel;
import javax.swing.border.EmptyBorder;
import java.awt.Color;
import java.awt.Button;
import javax.swing.JTextField;
import javax.swing.JLabel;
import javax.swing.JOptionPane;

import java.awt.Font;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.rmi.NotBoundException;
import java.rmi.RemoteException;
import java.rmi.registry.LocateRegistry;
import java.rmi.registry.Registry;

@SuppressWarnings("serial")
// class definition
public class AddSensor extends JFrame implements ActionListener{
```

```
// declaring variables
private JPanel contentPane;
private JTextField textField;
private JTextField textField_1;
private JTextField textField_2;
public String SensorName;
public String FloorNumber;
public String RoomNumber;



/**
 * Launch the application.
 */

// main function
public static void main(String[] args) {
	EventQueue.invokeLater(new Runnable() {
		public void run() {
			try {
				// create new AddSensor object
				AddSensor frame = new AddSensor();

				// start visible
				frame.setVisible(true);
			} catch (Exception e) {
				// print error
				e.printStackTrace();
			}
		}
	});
```

```
}

/**
 * Create the frame.
 */
// Add sensor function
public AddSensor() {
	// set values
	setBackground(Color.WHITE);
	setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);
	setBounds(100, 100, 729, 476);
	contentPane = new JPanel();
	contentPane.setBackground(Color.LIGHT_GRAY);
	contentPane.setBorder(new EmptyBorder(5, 5, 5, 5));
	setContentPane(contentPane);
	contentPane.setLayout(null);

	Button button = new Button("Register New Alarm");
	button.addActionListener(this);
	button.setFont(new Font("Dialog", Font.BOLD, 16));
	button.setForeground(Color.BLACK);
	button.setBackground(Color.LIGHT_GRAY);
	button.setBounds(246, 342, 211, 48);
	contentPane.add(button);


	textField = new JTextField();
	textField.setBackground(new Color(250, 250, 210));
	textField.setBounds(319, 176, 283, 36);
	contentPane.add(textField);
```

```
textField.setColumns(10);

JLabel lblFloorNo = new JLabel("Floor NO");
lblFloorNo.setForeground(Color.BLACK);
lblFloorNo.setFont(new Font("Tahoma", Font.BOLD, 17));
lblFloorNo.setBounds(169, 174, 114, 36);
contentPane.add(lblFloorNo);

JLabel lblRoomNo = new JLabel("Room No");
lblRoomNo.setForeground(Color.BLACK);
lblRoomNo.setFont(new Font("Tahoma", Font.BOLD, 17));
lblRoomNo.setBounds(172, 247, 111, 21);
contentPane.add(lblRoomNo);

textField_1 = new JTextField();
textField_1.setColumns(10);
textField_1.setBounds(319, 241, 283, 36);
contentPane.add(textField_1);

JLabel lblNewLabel_1 = new JLabel("Add New Alarm");
lblNewLabel_1.setForeground(Color.BLACK);
lblNewLabel_1.setFont(new Font("Berlin Sans FB Demi", Font.BOLD, 19));
lblNewLabel_1.setBounds(275, 13, 171, 57);
contentPane.add(lblNewLabel_1);

JLabel lblSensorName = new JLabel("Sensor Name");
lblSensorName.setForeground(Color.BLACK);
lblSensorName.setFont(new Font("Tahoma", Font.BOLD, 17));
lblSensorName.setBounds(169, 109, 114, 36);
contentPane.add(lblSensorName);
```

```
                textField_2 = new JTextField();
                textField_2.setColumns(10);
                textField_2.setBackground(new Color(250, 250, 210));
                textField_2.setBounds(319, 111, 283, 36);
                contentPane.add(textField_2);
        }


        // when administrator check register new sensor button, then this function will execute
        @Override
        public void actionPerformed(ActionEvent arg0) {

                // try catch block
                try {
                        // create a register and looking for running RMI server
                        Registry reg = LocateRegistry.getRegistry("localhost", 1099);

                        // create a sensor interface object
                        final SensorInterface server =
(SensorInterface)reg.lookup("rmi://localhost/service");

                        // fetching values from the inputs
                        SensorName = textField_2.getText();
                        FloorNumber = textField.getText();
                        RoomNumber = textField_1.getText();

                        // send request to sever with values
                        String response = server.createSensor(SensorName,
Integer.parseInt(FloorNumber), Integer.parseInt(RoomNumber));
```

```
                    // condition
                    if(response.equals("true")) {
                        // print success message
                        System.out.println("created successfully!");
                        JOptionPane.showMessageDialog(null, "Created successfully.");

                        // after message frame will disappear
                        dispose();
                    }else {
                        // print error message
                        JOptionPane.showMessageDialog(null, "Room number already exist!!");
                    }

                } catch (RemoteException e) {
                    // print error
                    e.printStackTrace();
                } catch (NotBoundException e) {
                    // print errot
                    e.printStackTrace();
                }
            }
        }
```

2. AdminDashboard.java

```
// importing dependencies
import java.awt.Component;
import java.awt.EventQueue;
import java.awt.Font;
import javax.swing.JFrame;
import javax.swing.JPanel;
import javax.swing.border.EmptyBorder;
import javax.swing.JTable;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
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.Timer;
import java.util.TimerTask;

import javax.swing.JLabel;
import javax.swing.DefaultCellEditor;
import javax.swing.JButton;
import javax.swing.JCheckBox;
import javax.swing.UIManager;
import javax.swing.table.DefaultTableModel;
import javax.swing.table.TableCellRenderer;
import javax.swing.table.TableModel;
```

```
import org.json.JSONArray;

import javax.swing.JScrollPane;

@SuppressWarnings("serial")
// class definition
public class AdminDashboard extends JFrame {

	// declaring variables
	private JPanel contentPane;
	private JTable table;
	private static ArrayList<String> ids = new ArrayList<String>();

	/**
	 * Launch the application.
	 */

	// main function
	public static void main(String[] args) {
		EventQueue.invokeLater(new Runnable() {
			public void run() {
				try {
					// create new Administrator dashBoard object
					AdminDashboard frame = new AdminDashboard();

					// start to visible
					frame.setVisible(true);
				} catch (Exception e) {
					// print error
					e.printStackTrace();
```

```
                        }
                    }
                });
            }

            /**
             * Create the frame.
             * @throws RemoteException
             * @throws NotBoundException
             */
            // AdminDashboard function
            public AdminDashboard() throws RemoteException, NotBoundException {
                // set values
                setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
                setBounds(100, 100, 897, 522);
                contentPane = new JPanel();
                contentPane.setBorder(new EmptyBorder(5, 5, 5, 5));
                setContentPane(contentPane);
                contentPane.setLayout(null);

                JButton btnNewButton = new JButton("Add");
                btnNewButton.addActionListener(new ActionListener() {
                    public void actionPerformed(ActionEvent arg0) {
                        AddSensor ds = new AddSensor();
                        ds.setVisible(true);

                    }
                });
                btnNewButton.setFont(new Font("Arial Black", Font.BOLD, 14));
                btnNewButton.setBackground(UIManager.getColor("Button.light"));
```

```
btnNewButton.setBounds(39, 56, 161, 41);
contentPane.add(btnNewButton);

JScrollPane scrollPane = new JScrollPane();
scrollPane.setBounds(89, 137, 680, 335);
contentPane.add(scrollPane);

DefaultTableModel dm = new DefaultTableModel();
table = new JTable(dm);
table.setShowHorizontalLines(false);
table.setEnabled(true);
table.setRowSelectionAllowed(false);
table.setFont(new Font("Lucida Sans", Font.PLAIN, 12));

// declare the timer
Timer t = new Timer();

// create a register and looking for running RMI server
Registry reg = LocateRegistry.getRegistry("localhost", 1099);

// create a sensor interface object
final SensorInterface server = (SensorInterface)reg.lookup("rmi://localhost/service");

// getting all sensor details
String jsonString = server.readSensors();

// JSON response convert to JAVA object
JSONArray arr = new JSONArray(jsonString);
```

```
// declaring headers in the table
final Object[][] rowData = {};
final Object[] columnNames = {"Sensor Name", "Floor Number", "Room Number", "CO2 Level", "Smoke Level", "Status", "Edit"};


// creating table object
final DefaultTableModel listTableModel;

// parse declared row data and columns
table.setModel(listTableModel = new DefaultTableModel(
        // data
        rowData,
        columnNames
    ) {
        boolean[] columnEditables = new boolean[] {
            false, false, false, false, false, false, true
        };
        public boolean isCellEditable(int row, int column) {
            return columnEditables[column];
        }
    });

//Set the schedule function and rate
t.scheduleAtFixedRate(new TimerTask() {

   @Override
   public void run() {

       // when this loop start to run firstly, remove all row data
```

```
                    if (listTableModel.getRowCount() > 0) {
                            // removing every rows
                       for (int i = listTableModel.getRowCount() - 1; i > -1; i--) {
                            listTableModel.removeRow(i);
                       }
                    }

                    // try catch block
try {
// getting all sensor details
String jsonString = server.readSensors();

// JSON response convert to JAVA object
final JSONArray arr = new JSONArray(jsonString);

// for loop
for(int i = 0; i < arr.length(); i++) {
// declaring sensor's details to variables
ids.add(arr.getJSONObject(i).getString("_id"));
String status;
String name = arr.getJSONObject(i).getString("name");
int floorNumber = arr.getJSONObject(i).getInt("floorNumber");
int roomNumber = arr.getJSONObject(i).getInt("roomNumber");
int co2 = arr.getJSONObject(i).getInt("carbonDioxideLevel");
int smokeLevel = arr.getJSONObject(i).getInt("smokeLevel");

// condition
if(co2 > 5 || smokeLevel > 5) {
// if one of sensor's value greater than 5, the value of the status will activated
status = "Activated";
```

```
}else {
// otherwise value of the status will normal
status = "Normal";
}

// insert a row with data
listTableModel.addRow(new Object[]{name, floorNumber, roomNumber, co2, smokeLevel,
status,"Edit"});
}
} catch (RemoteException e) {
// print error
e.printStackTrace();
}
}
},0,30000); // this will run after every 30 seconds


                    // set values
                    table = new JTable(listTableModel);
                    table.getColumnModel().getColumn(0).setPreferredWidth(83);
                    table.getColumnModel().getColumn(4).setPreferredWidth(85);
                table.getColumn("Edit").setCellRenderer(new ButtonRenderer());
                table.getColumn("Edit").setCellEditor(
                new ButtonEditor(new JCheckBox()));
                    scrollPane.setViewportView(table);


                    // set values
                    JLabel lblWelcome = new JLabel("Administrator");
                    lblWelcome.setFont(new Font("Tahoma", Font.BOLD, 21));
```

```
        lblWelcome.setBounds(381, 13, 151, 41);
        contentPane.add(lblWelcome);
}
// class definition
class ButtonRenderer extends JButton implements TableCellRenderer {

         // constructor
         public ButtonRenderer() {
          setOpaque(true);
         }

         public Component getTableCellRendererComponent(JTable table, Object value,
            boolean isSelected, boolean hasFocus, int row, int column) {
          if (isSelected) {
            setForeground(table.getSelectionForeground());
            setBackground(table.getSelectionBackground());
          } else {
            setForeground(table.getForeground());
            setBackground(UIManager.getColor("Button.background"));
          }
          setText((value == null) ? "" : value.toString());
          return this;
         }
        }

        // class definition
        class ButtonEditor extends DefaultCellEditor {

         // declaring variables
         protected JButton button;
```

```
                private String label;

                // function
                public ButtonEditor(JCheckBox checkBox) {
                 super(checkBox);
                 button = new JButton();
                 button.setOpaque(true);
                 button.addActionListener(new ActionListener() {


                 // action performed    function
                        @Override
                        public void actionPerformed(ActionEvent arg0) {
                                // TODO Auto-generated method stub
                                int index = table.getSelectedRow();

                                // create new table model
                   TableModel model = table.getModel();

                   // getting values from row
                   String SensorName = model.getValueAt(index, 0).toString();
                   String FloorNumber = model.getValueAt(index, 1).toString();
                   String RoomNumber = model.getValueAt(index, 2).toString();

                   // call the edit sensor class and its constructor
                                EditSensor regFace =new
EditSensor(ids.get(index),SensorName,FloorNumber, RoomNumber);
                                regFace.setVisible(true);
                        }
                 });
```

```
                }

                // get table details function
                public Component getTableCellEditorComponent(JTable table, Object value,
                        // declaring variables
                                boolean isSelected, int row, int column) {

                        // condition
                        if (isSelected) {
                          button.setForeground(table.getSelectionForeground());
                          button.setBackground(table.getSelectionBackground());
                        } else {
                          button.setForeground(table.getForeground());
                          button.setBackground(table.getBackground());
                        }

                        //  condition
                        label = (value == null) ? "" : value.toString();
                        button.setText(label);
                        return button;
                       }
        }
}
```

3. Client Dashboard.java

```
// importing dependencies
import java.awt.BorderLayout;
import java.awt.EventQueue;

import javax.swing.JFrame;
import javax.swing.JPanel;
import javax.swing.border.EmptyBorder;
import javax.swing.JTable;
import javax.swing.border.LineBorder;
import java.awt.Color;
import java.awt.GridLayout;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.io.IOException;
import java.rmi.NotBoundException;
import java.rmi.RemoteException;
import java.rmi.registry.LocateRegistry;
import java.rmi.registry.Registry;

import javax.swing.JLabel;
import java.awt.Font;
import javax.swing.JButton;
import javax.swing.UIManager;
import javax.swing.table.DefaultTableModel;
import javax.swing.JScrollPane;
import javax.swing.border.BevelBorder;
```

```
import com.fasterxml.jackson.core.JsonParseException;
import com.fasterxml.jackson.databind.JsonMappingException;
import com.fasterxml.jackson.databind.ObjectMapper;
import org.json.*;

import java.util.Timer;
import java.util.TimerTask;

// class definition
public class ClientDashboard extends JFrame {

	// declaring private variables
	private JPanel contentPane;
	private JTable table;
	private JTable table_1;

	/**
	 * Launch the application.
	 */

	// main function
	public static void main(String[] args) {
		//init function
		EventQueue.invokeLater(new Runnable() {
			public void run() {
				// try catch block
				try {
					// create clientDashboard object
					ClientDashboard frame = new ClientDashboard();
					// start to visible
```

```
                                        frame.setVisible(true);
                                } catch (Exception e) {
                                        // pring error
                                        e.printStackTrace();
                                }
                        }
                });
        }

        /**
         * Create the frame.
         * @throws NotBoundException
         * @throws IOException
         * @throws JsonMappingException
         * @throws JsonParseException
         */
        // declaring client dashBoard function
        public ClientDashboard() throws NotBoundException, JsonParseException,
JsonMappingException, IOException {
                // set values
                setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
                setBounds(100, 100, 897, 397);
                contentPane = new JPanel();
                contentPane.setBorder(new EmptyBorder(5, 5, 5, 5));
                setContentPane(contentPane);
                contentPane.setLayout(null);

                // declaring a new button - login button
                JButton btnNewButton = new JButton("Login");
```

```
// listener for when user click the login button this will execute
btnNewButton.addActionListener(new ActionListener() {
	public void actionPerformed(ActionEvent arg0) {
		// create a loginFrame object
		LoginFrame ds = new LoginFrame();
		// start to visible
		ds.setVisible(true);
		dispose();

	}
});
//set values
btnNewButton.setFont(new Font("Arial Black", Font.BOLD, 14));
btnNewButton.setBackground(UIManager.getColor("Button.light"));
btnNewButton.setBounds(685, 51, 161, 41);
contentPane.add(btnNewButton);

// creating scrollPane for table
JScrollPane scrollPane = new JScrollPane();
// set values
scrollPane.setBounds(89, 137, 680, 199);
contentPane.add(scrollPane);

// creating a new table object and set values
table = new JTable();
table.setEnabled(false);
table.setRowSelectionAllowed(false);
table.setFont(new Font("Bodoni MT Black", Font.PLAIN, 17));

// declare the timer
```

```
Timer t = new Timer();

// create a register and looking for running RMI server
Registry reg = LocateRegistry.getRegistry("localhost", 1099);
// create a sensor interface object
final SensorInterface server = (SensorInterface)reg.lookup("rmi://localhost/service");


// getting all sensor details
String jsonString = server.readSensors();

// JSON response convert to JAVA object
JSONArray arr = new JSONArray(jsonString);

// declaring headers in the table
final Object[][] rowData = {};
final Object[] columnNames = {"Sensor Name", "Floor Number","Room Number","CO2
level","Smoke Level", "status"};

// creating table object
final DefaultTableModel listTableModel;

// parse declared row data and columns
listTableModel = new DefaultTableModel(rowData, columnNames);


//Set the schedule function and rate
t.scheduleAtFixedRate(new TimerTask() {

   @Override
   public void run() {
```

```
                    // when this loop start to run firstly, remove all row data
                    if (listTableModel.getRowCount() > 0) {
                            // removing every rows
                       for (int i = listTableModel.getRowCount() - 1; i > -1; i--) {
                            listTableModel.removeRow(i);
                       }
                    }

                    // try catch block
                            try {
                                    // getting all sensor details
                                    String jsonString = server.readSensors();

                                    // JSON response convert to JAVA object
                                    final JSONArray arr = new JSONArray(jsonString);

                                    // for loop
                                    for(int i = 0; i < arr.length(); i++) {
                                            // declaring sensor's details to variables
                                            String status;
                                            String name = arr.getJSONObject(i).getString("name");
                                            int floorNumber =
arr.getJSONObject(i).getInt("floorNumber");
                                            int roomNumber =
arr.getJSONObject(i).getInt("roomNumber");
                                            int co2 =
arr.getJSONObject(i).getInt("carbonDioxideLevel");
                                            int smokeLevel =
arr.getJSONObject(i).getInt("smokeLevel");

                                            // condition
```

```
                                        if(co2 > 5 || smokeLevel > 5) {
                                                // if one of sensor's value greater than 5, the
value of the status will activated
                                                status = "Activated";
                                        }else {
                                                // otherwise value of the status will normal
                                                status = "Normal";
                                        }

                                        // insert a row with data
                                        listTableModel.addRow(new Object[]{name,
floorNumber, roomNumber, co2, smokeLevel, status,});
                                }
                        } catch (RemoteException e) {
                                // print error
                                e.printStackTrace();
                        }                       }
                },0,30000); // this will run after every 30 seconds
                // set values
                table_1 = new JTable(listTableModel);
                table_1.setEnabled(false);
                table_1.getColumnModel().getColumn(0).setPreferredWidth(83);
                table_1.getColumnModel().getColumn(4).setPreferredWidth(85);
                scrollPane.setViewportView(table_1);
                // create label object and set values
                JLabel lblWelcome = new JLabel("Welcome");
                lblWelcome.setFont(new Font("Tahoma", Font.BOLD, 21));
                lblWelcome.setBounds(365, 13, 101, 41);
                contentPane.add(lblWelcome);
        }
}
```

4. EditSensor.java

```
// importing dependencies
import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JPanel;
import javax.swing.border.EmptyBorder;
import java.awt.Color;
import java.awt.Button;
import javax.swing.JTextField;
import javax.swing.JLabel;
import java.awt.Font;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.rmi.NotBoundException;
import java.rmi.RemoteException;
import java.rmi.registry.LocateRegistry;
import java.rmi.registry.Registry;

@SuppressWarnings("serial")
// class definition
public class EditSensor extends JFrame implements ActionListener {

	// declaring variables
	private JPanel contentPane;
	public static JTextField textField;
	public static JTextField textField_1;
	public static JTextField textField_2;
```

```
public String SensorName;
public String FloorNumber;
public String RoomNumber;

public static String _id;

JButton btnDelete = new JButton("Delete");

// edit sensor function
public EditSensor(String id,String sensorName, String floorNumber, String roomNumber) {
        // set values
        setBackground(Color.GRAY);
        setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);
        setBounds(100, 100, 729, 476);
        contentPane = new JPanel();
        contentPane.setBackground(Color.LIGHT_GRAY);
        contentPane.setBorder(new EmptyBorder(5, 5, 5, 5));
        setContentPane(contentPane);
        contentPane.setLayout(null);

        Button button = new Button("Edit & Exit");
        button.addActionListener(this);
        button.setFont(new Font("Dialog", Font.BOLD, 16));
        button.setForeground(Color.BLACK);
        button.setBackground(Color.LIGHT_GRAY);
        button.setBounds(246, 342, 211, 48);
        contentPane.add(button);

        JLabel lblFloorNo = new JLabel("Floor NO");
```

```
lblFloorNo.setForeground(Color.BLACK);
lblFloorNo.setFont(new Font("Tahoma", Font.BOLD, 17));
lblFloorNo.setBounds(169, 174, 114, 36);
contentPane.add(lblFloorNo);

JLabel lblRoomNo = new JLabel("Room No");
lblRoomNo.setForeground(Color.BLACK);
lblRoomNo.setFont(new Font("Tahoma", Font.BOLD, 17));
lblRoomNo.setBounds(172, 247, 111, 21);
contentPane.add(lblRoomNo);

JLabel lblNewLabel_1 = new JLabel("Edit a Alarm");
lblNewLabel_1.setForeground(Color.BLACK);
lblNewLabel_1.setFont(new Font("Berlin Sans FB Demi", Font.BOLD, 19));
lblNewLabel_1.setBounds(308, 13, 114, 57);
contentPane.add(lblNewLabel_1);

JLabel lblSensorName = new JLabel("Sensor Name");
lblSensorName.setForeground(Color.BLACK);
lblSensorName.setFont(new Font("Tahoma", Font.BOLD, 17));
lblSensorName.setBounds(169, 109, 114, 36);
contentPane.add(lblSensorName);

textField = new JTextField();
textField.setBackground(new Color(250, 250, 210));
textField.setBounds(319, 176, 283, 36);
contentPane.add(textField);
textField.setColumns(10);

textField_1 = new JTextField();
```

```
		textField_1.setColumns(10);
		textField_1.setBounds(319, 241, 283, 36);
		contentPane.add(textField_1);

		textField_2 = new JTextField();
		textField_2.setColumns(10);
		textField_2.setBackground(new Color(250, 250, 210));
		textField_2.setBounds(319, 111, 283, 36);
		contentPane.add(textField_2);

		_id = id;
		textField.setText(floorNumber);
		textField_1.setText(roomNumber);
		textField_2.setText(sensorName);

		btnDelete.setBounds(54, 65, 105, 36);
		btnDelete.addActionListener(this);
		contentPane.add(btnDelete);

	}

	// action performed function
	@Override
	public void actionPerformed(ActionEvent arg0) {
		// try catch block
		try {
			// create source object to fetch current button details
			Object source = arg0.getSource();

			// create a register and looking for running RMI server
```

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

                    // condition
            if(source == btnDelete) {

                // create a sensor interface object
                final SensorInterface server = (SensorInterface)reg.lookup("rmi://localhost/service");

                // send request to server and get response
                String response = server.deleteSensor(_id);

                // condition
                        if(response.equals("true")) {
                                // success message
                                System.out.println("deleted successfully!");
                        }else {
                                // error message
                                System.out.println("Something went wrong!");
                        }

        }else {
          // create a sensor interface object
          final SensorInterface server = (SensorInterface)reg.lookup("rmi://localhost/service");

          // getting values
                        SensorName = textField_2.getText();
                        FloorNumber = textField.getText();
                        RoomNumber = textField_1.getText();

                        // convert strings to integer
```

```
                        int floorNumber = Integer.parseInt(FloorNumber);
                        int roomNumber = Integer.parseInt(RoomNumber);

                        // send request to server and get response
                        String response = server.updateSensor(_id, SensorName, floorNumber,
roomNumber);

                        // condition
                        if(response.equals("true")) {
                                // success message
                                System.out.println("updated successfully!");
                        }else {
                                // error message
                                System.out.println("Something went wrong!");
                        }
        }

                        // end of this function frame will disappear
                        dispose();
                } catch (RemoteException e) {
                        // print error
                        e.printStackTrace();
                } catch (NotBoundException e) {
                        // print error
                        e.printStackTrace();
                }
         }
}
```

5. LoginFrame.java

```
// importing dependencies
import java.awt.BorderLayout;
import java.awt.GridLayout;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.WindowAdapter;
import java.awt.event.WindowEvent;
import java.io.IOException;
import java.rmi.NotBoundException;
import java.rmi.RemoteException;
import java.rmi.registry.LocateRegistry;
import java.rmi.registry.Registry;
import java.util.Scanner;

import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JOptionPane;
import javax.swing.JPanel;
import javax.swing.JPasswordField;
import javax.swing.JTextField;
import java.awt.Font;
import javax.swing.SwingConstants;
import java.awt.Color;

// class definition
public class LoginFrame extends JFrame implements ActionListener {
```

```
        // creating necessary objects and set values
JPanel panel;
JLabel user_label, password_label, message;
static JTextField userName_text;
static JTextField password_text;
JButton submit, cancel;

// constructor
LoginFrame() {
        // se values
        setBackground(new Color(204, 51, 0));

   // User Label and set values
   user_label = new JLabel();
   user_label.setFont(new Font("Tahoma", Font.PLAIN, 18));
   user_label.setText("        User Name :");
   userName_text = new JTextField();

   // Password and set values
   password_label = new JLabel();
   password_label.setFont(new Font("Tahoma", Font.PLAIN, 18));
   password_label.setText("          Password :");
   password_text = new JPasswordField();

   // Submit and set values
   submit = new JButton("Login");
   submit.setForeground(new Color(255, 255, 240));
   submit.setBackground(new Color(51, 102, 255));
   submit.setFont(new Font("Tahoma", Font.BOLD | Font.ITALIC, 18));
```

```
// create new panel object
panel = new JPanel(new GridLayout(3, 1));

// set values
panel.add(user_label);
panel.add(userName_text);
panel.add(password_label);
panel.add(password_text);

// set values
message = new JLabel();
message.setForeground(new Color(255, 102, 0));
message.setBackground(new Color(255, 102, 51));
panel.add(message);
panel.add(submit);

// Adding the listeners to components..
submit.addActionListener(this);
getContentPane().add(panel, BorderLayout.CENTER);
setTitle("Please Login Here !");
setSize(409, 170);
setVisible(true);
setLocation(600,290);

addWindowListener(new java.awt.event.WindowAdapter() {
   public void windowClosing(java.awt.event.WindowEvent e) {
     ClientDashboard ds;
                                try {
                                        ds = new ClientDashboard();
                                        ds.setVisible(true);
```

```
                                        } catch (NotBoundException | IOException e1) {
                                                // TODO Auto-generated catch block
                                                e1.printStackTrace();
                                        }
                }
            });
        }

        // main function
        public static void main(String[] args) {new LoginFrame();}

        // when user click submit button this function will start to execute
                public void actionPerformed(ActionEvent e) {
                        // try-catch block
                        try {
                                        // create a registry and looking for RMI server
                                        Registry reg = LocateRegistry.getRegistry("localhost", 1099);

                                        // create an object of login interface
                                        LoginInterface server = (LoginInterface)
reg.lookup("rmi://localhost/service");

                                        // get user input
                                        String username = userName_text.getText();
                                        String password = password_text.getText();

                                        // call the login function and get response
                                        String response = server.login(username, password);

                                        // condition
```

```
                                        if(response.equals("false")) {
                                                // if invalid credentials, then prompt a message to user
                                                JOptionPane.showMessageDialog(null, "username or password
incorrect!");
                                        }
                                        else {
                                                // if valid credentials prompt a message
                                                JOptionPane.showMessageDialog(null, "successfully logged
in.");

                                                // show administrator dashBaord to user
                                                AdminDashboard regFace =new AdminDashboard();
                                                // start to visible
                                                regFace.setVisible(true);
                                                dispose();
                                        }
                                } catch (RemoteException | NotBoundException ex) {
                                        // print error
                                        System.out.println(ex.getMessage());
                                }
                        }
                }
```

6. LoginInterface.java

```
// importing dependencies
import java.rmi.Remote;
import java.rmi.RemoteException;

// class definition
public interface LoginInterface extends Remote{
        // login function
   public String login(String username, String password) throws RemoteException;
}
```

7. SendEmail.java

```
// importing dependencies
import com.sun.mail.smtp.SMTPTransport;

import com.sun.mail.smtp.SMTPTransport;

import javax.mail.Message;
import javax.mail.MessagingException;
import javax.mail.Session;
import javax.mail.internet.InternetAddress;
import javax.mail.internet.MimeMessage;
import java.util.Date;
import java.util.Properties;

// class definition
public class SendEmail {

        // declaring global variables
   private static final String SMTP_SERVER = "smtp.gmail.com";
   private static final String USERNAME = "gotukolamalluma@gmail.com";
   private static final String PASSWORD = " gotukolamalluma_13";

   private static final String EMAIL_FROM = "gotukolamalluma@gmail.com";
   private static final String EMAIL_TO = "gotukolamalluma@gmail.com";
   private static final String EMAIL_TO_CC = "";

   private static final String EMAIL_SUBJECT = "The fire alert has been activated!!";
```

```
    private static final String EMAIL_TEXT = "please leave the building by the nearest exit and go to the
nearest assembly point immediately!!";


    // defining constructor
    SendEmail() {
            // get system properties and include necessary data
            Properties prop = System.getProperties();
        prop.put("mail.smtp.starttls.enable", "true");
        prop.put("mail.smtp.host", SMTP_SERVER); //optional, defined in SMTPTransport
        prop.put("mail.smtp.auth", "true");
        prop.put("mail.smtp.port", "25"); // default port 25

        // declaring session
        Session session = Session.getInstance(prop, null);
        Message msg = new MimeMessage(session);

        // try catch block
        try {

                                // from
            msg.setFrom(new InternetAddress(EMAIL_FROM));

                                // to
            msg.setRecipients(Message.RecipientType.TO,
                    InternetAddress.parse(EMAIL_TO, false));

                                // cc
            msg.setRecipients(Message.RecipientType.CC,
                    InternetAddress.parse(EMAIL_TO_CC, false));
```

```
                        // subject
            msg.setSubject(EMAIL_SUBJECT);

                        // content
            msg.setText(EMAIL_TEXT);

            msg.setSentDate(new Date());

                        // Get SMTPTransport
            SMTPTransport t = (SMTPTransport) session.getTransport("smtp");

                        // connect
            t.connect(SMTP_SERVER, USERNAME, PASSWORD);

                        // send
            t.sendMessage(msg, msg.getAllRecipients());

            // user message when email sent successfully.
            System.out.println("Email sent successfully.");

            // closing SMTPTransport
            t.close();
     } catch (MessagingException e) {
               // print error
            e.printStackTrace();
          }
       }
    }
```

8. SendSMS.java

```
// importing dependencies
import com.twilio.Twilio;
import com.twilio.rest.api.v2010.account.Message;
import com.twilio.type.PhoneNumber;

// class definition
public class SendSMS {

 // declaring global variables
 public static final String ACCOUNT_SID = "AC8c941f54f9b3450f5d1d2fd069a55db6";

 //  -- you have to create an Twilio account to get auth_token --
 public static final String AUTH_TOKEN = "0e2f7d304ad04c65e2b3c1721c0f5663";

 // declaring constructor
 SendSMS() {
        // start configurations
        Twilio.init(ACCOUNT_SID, AUTH_TOKEN);

        // send message
        Message message = Message.creator(new PhoneNumber("+94757924827"), // to
           new PhoneNumber("+15183179461"), // from -- this number is generated by Twilio.com
           "The fire alert has been activated!!").create(); // message we want to send

        // message if message sent successfully.
        System.out.println("Message sent Successfully.");
 }
}
```

9. SensorInterface.java

```
// importing dependencies
import java.rmi.Remote;
import java.rmi.RemoteException;

// class definition
public interface SensorInterface extends Remote{
	// function for create sensor
	public String createSensor(String name,int floorNumber,int roomNumber) throws RemoteException;

	// function for read all sensors
	public String readSensors() throws RemoteException;

	// function for read only one sensor
	public String readSensor(String id) throws RemoteException;

	// function for update sensor using _id
	public String updateSensor(String id,String name,int floorNumber,int roomNumber) throws RemoteException;

	// function for delete sensor
	public String deleteSensor(String id) throws RemoteException;
}
```

10. Server.java

```
// importing dependencies
import java.rmi.RemoteException;
import java.rmi.registry.LocateRegistry;
import java.rmi.registry.Registry;
import java.rmi.server.UnicastRemoteObject;
import java.util.Iterator;
import java.util.Map;
import java.util.Set;
import java.util.Timer;
import java.util.TimerTask;
import java.util.TreeMap;
import java.util.logging.Level;
import java.util.logging.Logger;

import org.json.JSONArray;

import com.sun.jersey.api.client.Client;
import com.sun.jersey.api.client.ClientResponse;
import com.sun.jersey.api.client.WebResource;

// class definition with extends UnicastRemoteObject and implements LoginInterface, SensorInterface
public class Server extends UnicastRemoteObject implements LoginInterface,SensorInterface{

   // constructor
   public Server() throws RemoteException{
      super();
   }
```

```
// main method
public static void main(String[] args) {
   try {
        // creating registry
      Registry reg = LocateRegistry.createRegistry(Registry.REGISTRY_PORT);
      // creating server object
      Server obj = new Server();

      // bind the rmi server with server object
      reg.rebind("rmi://localhost/service", obj);

      // message that indicate when server running
      System.out.println("Server is running");

      // declare the timer
                Timer t = new Timer();

                // set the schedule function and rate
                t.scheduleAtFixedRate(new TimerTask() {

                   @Override
                   public void run() {

                              try {
                                        // create server object
                                        Server obj = new Server();
```

```
                    // call the REST API and get the all sensor details and assign it to
a variable
                    String jsonString = obj.readSensors();

                    // convert to java JSON Array object
                JSONArray arr = new JSONArray(jsonString);

                    // loop
                        for(int i = 0; i < arr.length(); i++) {
                            // initializing carbon dioxide level
                            int co2 =
arr.getJSONObject(i).getInt("carbonDioxideLevel");

                            // initializing smoke level
                            int smokeLevel =
arr.getJSONObject(i).getInt("smokeLevel");

                            // condition
                            if(co2 > 5 || smokeLevel > 5) {
                                // if any sensor's carbon dioxide level or
                                // smoke level is grater than 5 these
objects will fire.
                                new SendEmail();
                                new SendSMS();
                            }
                        }
                    } catch (RemoteException e) {
                        // print error
                        e.printStackTrace();
                    }
```

```
                    }
                },0,15000); // after every 15 seconds this will execute
        } catch (RemoteException ex) {
            // print error
            System.out.println(ex.getMessage());
        }
    }

    // login function
    public String login(String username, String password) throws RemoteException {
            // create client object
                Client client = Client.create();

                // create we resource object
                WebResource webResource;

                // declaring API URL
                webResource = client.resource("http://localhost:5000/api/users/login");

                // setting JSON type string before send it to the API
                String input = "{\"username\":\"" + username + "\",\"password\":\"" + password + "\"}";

                // getting response
                ClientResponse response =
webResource.type("application/json").post(ClientResponse.class, input);

                // assigning that response to a string variable
                String output = response.getEntity(String.class);
```

```
                // return response
                return output;
    }

        @Override
        public String createSensor(String name, int floorNumber, int roomNumber)
                        throws RemoteException {
                // create client object
                Client client = Client.create();

                // create we resource object
                WebResource webResource;

                // declaring API URL
                webResource = client.resource("http://localhost:5000/api/sensors");

                // setting JSON type string before send it to the API
                String input = "{\"name\":\"" + name + "\",\"floorNumber\":" + floorNumber +
",\"roomNumber\":" + roomNumber + "}";

                // getting response
                ClientResponse response =
webResource.type("application/json").post(ClientResponse.class, input);

                // assigning that response to a string variable
                String output = response.getEntity(String.class);

                // return status
                return output;
```

```
	}

	@Override
	public String readSensors() throws RemoteException {
		// create client object
		Client client = Client.create();
		// create we resource object
		WebResource webResource;

		// declaring API URL
		webResource = client.resource("http://localhost:5000/api/sensors");

		// getting response
		ClientResponse response =
webResource.accept("application/json").get(ClientResponse.class);

		// condition
		if (response.getStatus() != 200) {
			// if response status not equal to 200 then throw new error
			throw new RuntimeException("Failed : HTTP error code : "+
response.getStatus());
		}

		// assigning that response to a string variable
		String output = response.getEntity(String.class);

		// return response
		return output;
	}
```

```
	@Override
	public String readSensor(String id) throws RemoteException {
		// create client object
		Client client = Client.create();

		// create we resource object
		WebResource webResource;

		// declaring API URL
		webResource = client.resource("http://localhost:5000/api/sensors/" + id + "");

		// getting response
		ClientResponse response =
webResource.accept("application/json").get(ClientResponse.class);

		// condition
		if (response.getStatus() != 200) {
			// if response status not equal to 200 then throw new error
			throw new RuntimeException("Failed : HTTP error code : "+
response.getStatus());
		}

		// assigning that response to a string variable
		String output = response.getEntity(String.class);

		// return response
		return output;
	}
```

```
@Override
public String updateSensor(String id, String name, int floorNumber, int roomNumber) throws RemoteException {

        // create client object
        Client client = Client.create();

        // create we resource object
        WebResource webResource;

        // declaring API URL
        webResource = client.resource("http://localhost:5000/api/sensors/" + id + "");

        // setting JSON type string before send it to the API
        String input = "{\"name\":\"" + name + "\",\"floorNumber\":" + floorNumber + ",\"roomNumber\":" + roomNumber + "}";

        // getting response
        ClientResponse response = webResource.type("application/json").put(ClientResponse.class, input);

        // condition
        if (response.getStatus() != 200) {
                // if response status not equal to 200 then throw new error
                throw new RuntimeException("Failed : HTTP error code : " + response.getStatus());
        }

        // assigning that response to a string variable
```

```
		String output = response.getEntity(String.class);

		// return response
		return output;
	}

	@Override
	public String deleteSensor(String id) throws RemoteException {
		// create client object
		Client client = Client.create();

		// create we resource object
		WebResource webResource;

		// declaring API URL
		webResource = client.resource("http://localhost:5000/api/sensors/" + id + "");

		// getting response
		ClientResponse response =
webResource.type("application/json").delete(ClientResponse.class);

		// condition
		if (response.getStatus() != 200) {
			// if response status not equal to 200 then throw new error
			throw new RuntimeException("Failed : HTTP error code : " +
response.getStatus());
		}

		// assigning that response to a string variable
```

```
                String output = response.getEntity(String.class);


                // return response
                return output;
        }
}
```

11. Pom.xml

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-
4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>FireAlarm</groupId>
  <artifactId>FireAlarm</artifactId>
  <version>0.0.1-SNAPSHOT</version>
    <dependencies>
        <dependency>
                <groupId>com.sun.jersey</groupId>
                <artifactId>jersey-client</artifactId>
                <version>1.8</version>
        </dependency>
        <dependency>
        <groupId>com.fasterxml.jackson.core</groupId>
        <artifactId>jackson-databind</artifactId>
        <version>2.9.8</version>
</dependency>
<dependency>
```

```
    <groupId>org.json</groupId>
    <artifactId>json</artifactId>
    <version>20190722</version>
</dependency>
<dependency>
        <groupId>com.sun.mail</groupId>
        <artifactId>javax.mail</artifactId>
        <version>1.6.2</version>
</dependency>
<dependency>
  <groupId>com.twilio.sdk</groupId>
  <artifactId>twilio</artifactId>
  <version>7.49.1</version>
</dependency>
  </dependencies>
</project>
```