**Sri Lanka Institute of Information**

**Technology**



**DISTRIBUTED SYSTEMS(SE3020)**

**Assignment 2**

**Assignment Report**

|  |  |
| --- | --- |
| IT number | Name |
| IT17182638 | Hewapathirana C.D.D |
| IT17182324 | Maheshani H.M.P |

CONTENT

Page Number

1. Introduction -------------------------------------------------------------- 3
2. High Level Architectural Diagram

2.1 Component Diagram -------------------------------------------------------------- 4

1. System services with workflows
   1. User Login Service -------------------------------------------------------------- 5
   2. Add Sensor Service ---------------------------------------------------------------- 6
   3. Update Sensor Service ------------------------------------------------------------ 7
   4. Get Sensor details Service -------------------------------------------------------- 8
   5. Email and SMS Service ------------------------------------------------------------ 9
2. Appendix -------------------------------------------------------------------- 10
3. End of the Report ---------------------------------------------------------- 54

INTRODUCTION

In here, ‘Fire alarm monitoring system’ is implemented using below mentioned technologies. ReactJS is used for the front-end(in the Web Client as the client side). As the backend, NodeJS and ExpressJS were used for the server side(REST API). In addition to that, as a service, sending automated emails is implemented using Nodemailer module. As well as in here, it is used to send dummy messages for sending messages service. Desktop client is developed using RMI server and an RMI desktop client. And MongoDB is used as the database to store the data. For communication between clients and the backend, JSON is used as the communication method.

Here, it is not used MongoDB local database but MongoDB Atlas cloud which is the global cloud database service which allows to host and manage the data in the cloud. MongoDB Atlas cloud is flexible and scalable document-based database and available as a fully managed service.

In web client, it is implemented using ReactJS and used Bootstrap themes. Axios library is used to send requests to the API. In the REST API is implemented considering the SOA principles. By using this, it can be achieved interoperability between different applications easily. The RMI client service is developed using Java language.

***Simply, the workflow of this ‘Fire Alarm Monitoring System’ as following*.**

As a user, admin can login through the desktop client can add sensors and edit the details of added sensors. When adding sensors, it includes the sensor number, floor number and room number. Then the added sensors are updated each 30 seconds by the Sensor app. Using the web client, it can be viewed the details about all added sensors with the status, smoke level and co2 level which mainly considered. And also, in web client it shows all sensor details. If smoke level or co2 level is above 5, then that field is shown in red which shows as a warning and below 5, then that field is shown in green which shows no harm. Each 40 seconds, web client calls the REST API and get the updates. RMI server is reading the sensor status details in every 5 seconds. When reading, if any added sensor has smoke level or co2 level above 5, then RMI server sends emails to the admins and show messages using dummy messages service as an alert in the server console. Also, in the desktop client, if the smoke level or co2 level is above 5, shows an alert.

In conclusion, this ‘Fire Alarm Monitoring System’ is combination of a website(frontend) and a REST API (backend) which is accompanied by the MongoDB database.

HIGH LEVEL ARCHITECTURAL DIAGRAM

Component Diagram

A close up of text on a white background

Description automatically generated

Architecture diagram

A screenshot of a cell phone

Description automatically generated

SERVICES WITH WORKFLOWS

1. User Login Service



* When a user, login to the system as admin, after adding the credentials, those credentials are passed through a HTTP POST request of URL (user / create) to the REST API. Then the REST API server will call user routes. After that user routes will call the user service requesting the User Login Service while passing the admin credentials. And then compare those credentials with the details in MongoDB. Then it will pass the response as HTTP response.

**Security Mechanism for Sign in and sign up**

* It is achieved using hashing the password. It is implemented using npm ‘bcrypt’ module.

                /\*

                 \* hash the password

                 \*/

                bcrypt.hash(req.body.Password,10,(err, hash) =>{

                    if(err){

                        return res.status(500).json({

                            error: err

                        })

                    }else{

                        /\*

                         \* Create the User Schema

                         \*/

                        const nwuser = new User({

                            \_id:mongoose.Types.ObjectId(),

                            Username: req.body.Username,

                            Password: hash

                        });

* When sign in password is compared with hashing password.

  /\*

         \* Compare the entered passwaord with exisiting password

         \*/

        bcrypt.compare(req.body.Password, user[0].Password,(err,result) => {

            if(err){

                return res.send({message :'Password does not match!!!!!'});

            }

1. Add Sensor Service



* When adding a sensor, admin has to enter the sensor details such as sensor id, floor number and room number. Then those details will pass to the Sensor Service requesting for creating a new server through a HTTP POST request or URL / sensor / create. Then new sensor is created, and sensor details are stored in the MongoDB. Then the response is passed through a response message by the server.

1. Update Sensor Service



* When an admin wants to update an existing sensor, the request is passed through a HTTP PUT request including a sensor object. Using the sensor object id, it is requested for updating the sensor from the Update Sensor Service in Sensor Service. Then the sensor object id will pass to the MongoDB. Then find the existing sensor which match with the passed sensor id. After that, save the update details in the database of the particular sensor object id. Then server will pass the response as a JSON response.

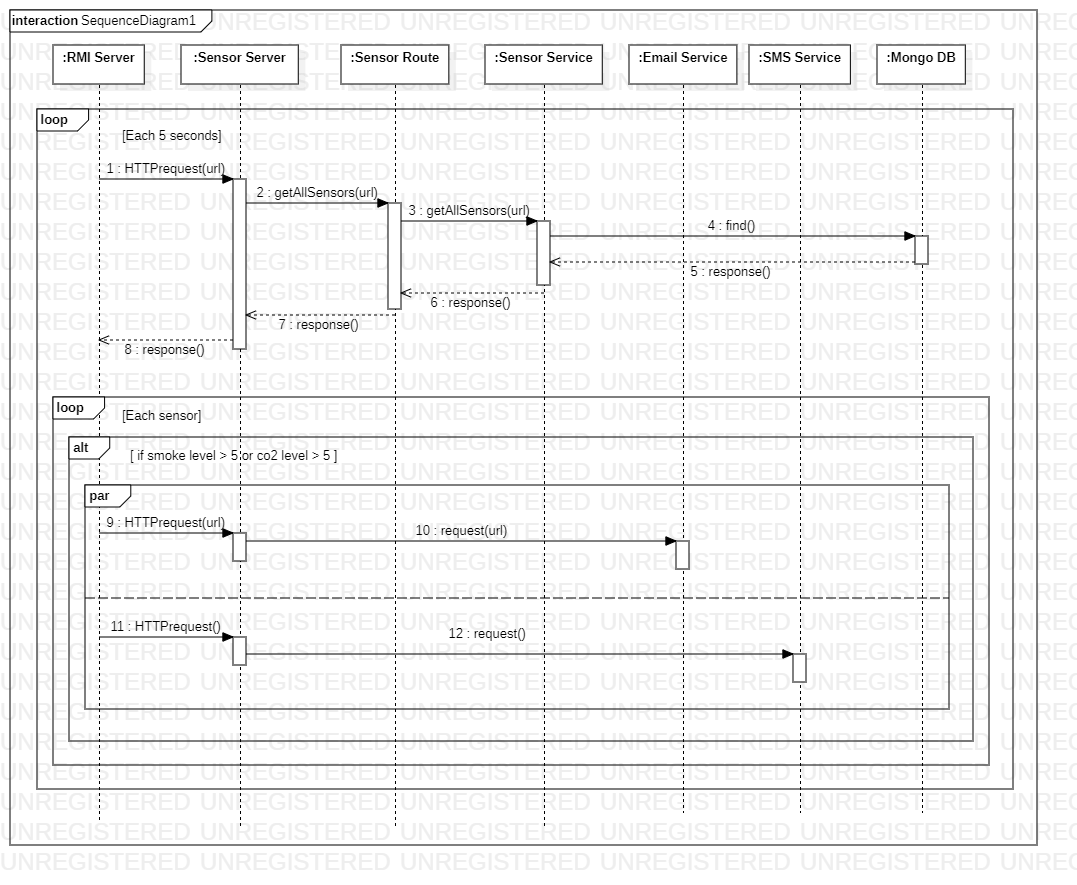
1. Get Sensor Details Service



If an admin wants to retrieve the existing sensor details, it is requested through the axios GET request with the URL ( axios.get('http://localhost:5000/sensor/') ). Then that request is sent to the Sensor Service, requesting Get Sensor Details Service using the URL. After that, the database retrieves the all data of sensors and fetch the details to the Get Sensor Details Service. Then server will pass the response as JSON response. Each 40 seconds, this process is done. Like this, desktop client also get updates through RMI server, each 30 seconds.

1. Email and SMS Service

APPENDIX



* If smoke level or co2 level is above 5 in any sensor, a warning message should be sent as an email and a message. First, it needs to be checked the details of all existing sensors. RMI server reads the sensor details each 5 seconds. If there is any details about smoke level or co2 level is above 5 of any sensor, it should send email or message. For each sensor, this should be checked. Then RMI server send a HTTP request with the URL to the Email Service and SMS (Message) Service. Then admin is informed about the danger. Though Email service and SMS service mentioned in here as a one service for the ease of demonstration, those services are considered as two services.

WebClient (front-end)

**Monitor.js**

import React, { Component } from 'react';

import axios from 'axios';

import { MDBTable, MDBTableBody, MDBTableHead } from 'mdbreact';

import Details from './details';

import {Card} from 'react-bootstrap';

class monitor extends Component {

    constructor(props) {

        super(props);

        this.state = {

            FireDetail : []

         }

    }

    componentDidMount(){

        this.getDetails();

        this.interval = setInterval(() => {

          this.getDetails();

        }, 40000);

    }

    getDetails() {

        axios.get('http://localhost:5000/sensor/')

        .then(response => {

            this.setState({ FireDetail: response.data });

        })

        .catch(function (error) {

            console.log(error);

        })

      }

    tableRow(){

        return this.state.FireDetail.map(function(object, i){

            return <Details obj={object} key={i} />;

        });

    }

    render() {

        return (

            <Card className="text-center">

            <Card.Header style={{color:"red"}}><h5>Fire Sensor Monitor</h5></Card.Header>

            <Card.Body>

                <center>

                <MDBTable small style={{ marginTop: 20, width:"1000px" }}>

                    <MDBTableHead>

                    <tr className="text-center">

                        <th>Sensor ID</th>

                        <th>Floor No</th>

                        <th>Room No</th>

                        <th>CO2 Level</th>

                        <th>Smoke Level</th>

                        <th>Status</th>

                    </tr>

                    </MDBTableHead>

                    <MDBTableBody>

                    { this.tableRow() }

                    </MDBTableBody>

                </MDBTable>

                </center>

            </Card.Body>

            <Card.Footer className="text-muted"></Card.Footer>

            </Card>

         );

    }

}

export default monitor;

**Details.js**

import React, { Component } from 'react';

import {Alert} from 'react-bootstrap';

class details extends Component {

    constructor(props) {

        super(props);

        this.state = {  }

    }

    render() {

        return (

            <tr className="text-center">

            <td> {this.props.obj.SensorID} </td>

            <td> {this.props.obj.FloorNo} </td>

            <td> {this.props.obj.roomNo}  </td>

            <td >

               {(this.props.obj.Co2Level > 5) ?

                    <Alert variant="danger">

                        {this.props.obj.Co2Level}

                    </Alert>

                    :

                    <Alert variant="success ">

                        {this.props.obj.Co2Level}

                    </Alert>

                }

            </td>

            <td>

                {(this.props.obj.smokeLevel > 5) ?

                    <Alert variant="danger">

                        {this.props.obj.smokeLevel}

                    </Alert>

                    :

                    <Alert variant="success ">

                        {this.props.obj.smokeLevel}

                    </Alert>

                }

            </td>

            <td> {this.props.obj.status} </td>

        </tr>

         );

    }

}

export default details;

**App.js**

import React, { Component } from 'react';

import 'bootstrap/dist/css/bootstrap.min.css';

import Monitor from './Component/monitor';

class App extends Component {

  render() {

    return (

      <div>

          <Monitor />

      </div>

     );

  }

}

export default App;

**Back-End (Server – REST API)**

**Server.js**

const express = require('express');

const cors = require('cors');

const mongoose = require('mongoose'); // help to connect mongodb database

const texts = require('./constants/texts');//take constant to prompt messages

const serverMessages = texts.server;

//create express server

const app = express();

const port = process.env.PORT || 5000;

//middlewares

app.use(cors());  // cors middaleware

app.use(express.json()); // allows to get JSON

//connect to the mongoDB Atlas

mongoose.connect(serverMessages.MONGODB\_URL,

{

    useNewUrlParser:true,useUnifiedTopology: true

})

.then(() =>{

    console.log(serverMessages.DB\_CONNECTED);

})

.catch(()=>{

    console.log(serverMessages.DB\_NOT\_CONNECTED);

});

/\*

 \* routes the request to the user route

 \*/

const userRoute = require('./routes/userRouter');

app.use('/user', userRoute);

/\*

 \* routes the request to the sensor route

 \*/

const sensorRoute = require('./routes/SensorRouters');

app.use('/sensor', sensorRoute);

/\*

 \* routes the request to the email Route

 \*/

const emailRoute = require('./routes/emailRouters');

app.use('/email', emailRoute);

/\*

 \* routes the request to the sms Route

 \*/

const smsRoute = require('./routes/smsRouters');

app.use('/sms', smsRoute);

/\*

 \* Backend server is lisenting to the port 5000

 \*/

app.listen(port, () => {

    console.log(serverMessages.SERVER + port);

});

**Routes**

**User routes**

const express = require("express");

const usersRouter = express.Router();

const UsersController = require('../controllers/UserController');// get the user controller

usersRouter.post("/sign-up", UsersController.user\_signup);//user sign up endpoint

usersRouter.post('/sign-in', UsersController.userSignin); //user sign in endpoint

module.exports = usersRouter;

**Sensor routes**

const express = require("express");

const SensorRoute = express.Router();

const SensorController = require("../controllers/SensorController");//get the sensor controller

SensorRoute.post("/create", SensorController.addSensor);     // sensor create endpoint

SensorRoute.get("/", SensorController.getAllSensor);         // get ALl sensor endpoint

SensorRoute.get("/:id", SensorController.getSensor);         // get Sensor by id endpoint

SensorRoute.put("/update/:id", SensorController.editSensor); // Update Sensor endpoint

SensorRoute.delete("/delete/:id", SensorController.deleteSensor);// delete Sensor endpoint

module.exports = SensorRoute;

**Email routes**

const express = require("express");

const EmailRoutes = express.Router();

const EmailControllers = require("../controllers/EmailController"); //get the email controller

EmailRoutes.post("/send", EmailControllers.sendEmail); // email send endpoint

module.exports = EmailRoutes;

**SMS routes**

const express = require("express");

const smsRoute = express.Router();

const smsController = require("../controllers/SmslController"); //get the sms controller

smsRoute.post("/send", smsController.sendSms); // sms send endpoint

module.exports = smsRoute;

**Model**

**Sensor Model**

const mongoose = require('mongoose');

const Schema = mongoose.Schema;

/\*

 \* Sensor schema

 \*/

let Sensor = new Schema({

    SensorID: {

        type: String,

        required: true

    },

    FloorNo: {

        type: Number,

        required: true

    },

    roomNo: {

        type: Number,

        required: true

    },

    smokeLevel: {

        type: Number,

        default:0

    },

    Co2Level: {

        type: Number,

        default:0

    },

    status: {

        type: String,

        default:null

    }

},{

    collection: 'Sensor'

});

module.exports = mongoose.model('Sensor',Sensor);

**User model**

const mongoose = require('mongoose');

const Schema = mongoose.Schema;

/\*

 \* User schema

 \*/

let User = new Schema({

    \_id: {

        type:String,

        required: true

    },

    Username: {

        type: String,

        required: true,

        unique: true,

        match: /[a-z0-9!#$%&'\*+/=?^\_`{|}~-]+(?:\.[a-z0-9!#$%&'\*+/=?^\_`{|}~-]+)\*@(?:[a-z0-9](?:[a-z0-9-]\*[a-z0-9])?\.)+[a-z0-9](?:[a-z0-9-]\*[a-z0-9])?/

    },

    Password: {

        type: String,

        required: true

    },

    LoginTime: {

        type: Date,

        default:null

    },

},{

    collection: 'User'

});

module.exports = mongoose.model('User', User);

**Controllers**

**UserController.js**

const mongoose = require("mongoose");

const bcrypt = require('bcrypt');

const User = require('../model/user');

/\*

 \* post request calls to the user\_signup in userController class to insert a new user

 \*/

exports.user\_signup = (req,res,next) => {

    const {Username} = req.body;

    //find whether User Already exist or not

    User.find({Username})

        .exec()

        .then(user => {

            if(user.length >= 1){

                return res.json({

                    message: 'Already exist'

                });

            }else{

                /\*

                 \* hash the password

                 \*/

                bcrypt.hash(req.body.Password,10,(err, hash) =>{

                    if(err){

                        return res.status(500).json({

                            error: err

                        })

                    }else{

                        /\*

                         \* Create the User Schema

                         \*/

                        const adduser = new User({

                            \_id:mongoose.Types.ObjectId(),

                            Username: req.body.Username,

                            Password: hash

                        });

                        adduser

                            .save()

                            .then(result => {

                                console.log('User is Created',result);

                                res.status(200).json({

                                    success:true,

                                    message: 'User is Created'

                                })

                            })

                            .catch( err =>{

                                console.log(err);

                                res.status(500).json({

                                    error: err

                                })

                            });

                    }

                });

            }

        })

}

/\*

 \* post request calls to the user Signin in userController class to login

 \*/

exports.userSignin =(req,res,next) => {

    User.find({Username:req.body.Username}).exec().then(user => {

        if(user.length < 1){

            return res.send({message:'not a User'});

        }

        /\*

         \* Compare the entered passwaord with exisiting password

         \*/

        bcrypt.compare(req.body.Password, user[0].Password,(err,result) => {

            if(err){

                return res.send({message :'Password not match'});

            }

            if(result){

                user[0].\_id = user[0].\_id;

                user[0].LoginTime = Date.now();

                user[0]

                    .save()

                    .then(result => {

                        console.log("User:"+user[0].\_id+"Login-Time"+Date.now());

                    })

                    .catch( err =>{

                        console.log(err);

                    });

                return res.status(200).json({

                    success:true,

                    message:'successful',

                });

            }else{

                return res.status(401).send('not a user');

            }

        })

    });

}

**SensorController.js**

const mongoose = require("mongoose");

const bcrypt = require('bcrypt');

const Sensor = require('../model/sensor');

/\*

 \* post request calls to the addSensor in SensorController class to insert a new Sensor

 \*/

exports.addSensor = (req, res, next) => {

    const {body} = req;

    const {

        SensorID,

        FloorNo,

        roomNo,

    } = body;

/\*

 \* check the sensor id already exist

 \*/

    Sensor.find({

        SensorID

    }).exec()

      .then(sensor => {

        if(sensor.length >= 1){

            return res.json({

                message : 'sensor already exist'

            });

        }else{

            /\*

             \* Create the sensor schema

             \*/

            const newsensor = new Sensor();

            newsensor.SensorID = SensorID;

            newsensor.FloorNo = FloorNo;

            newsensor.roomNo = roomNo;

            /\*

             \* Save the sensor schema

             \*/

            newsensor

                .save()

                .then(result => {

                    console.log(result);

                    res.status(200).json({

                        message: 'Sensor successfully created'

                    })

                })

                .catch(err => {

                    console.log(err);

                });

        }

    });

}

/\*

 \* Get the all sensors

 \*/

exports.getAllSensor = (req, res) => {

    Sensor.find((err, sensor) => {

        if(err){

            console.log(err);

        }

        else {

            res.json(sensor);

        }

    });

}

/\*

 \* Get the specific sensor

 \*/

exports.getSensor = (req, res) => {

    let sensorid = req.params.id;

    Sensor.findById(sensorid)

    .then(sensor => res.json(sensor))

    .catch(err => res.status(400).json('Error: ' + err));

}

/\*

 \* PUT request calls to the editSensor in SensorController class to Update a existing sensor

 \*/

exports.editSensor = (req, res) => {

    const {body} = req;

    const {

        SensorID,

        FloorNo,

        roomNo,

        smokeLevel,

        Co2Level,

        status

    } = body;

    /\*

     \* find the sensor

     \*/

    Sensor.findById(req.params.id, (err, sensor) => {

        if (!sensor)

            res.status(404).send({

                message:"sensor is not found"});

        else {

            sensor.SensorID = SensorID;

            sensor.FloorNo = FloorNo;

            sensor.roomNo = roomNo;

            sensor.smokeLevel = smokeLevel;

            sensor.Co2Level = Co2Level;

            sensor.status = status;

             /\*

              \* update the sensor

              \*/

            sensor

            .save().then(sensor => {

                res.json({

                    message:'Update sensor complete'});

            })

            .catch(err => {

                res.status(400).send({

                    message:"unable to update the database"});

            });

        }

    });

}

exports.deleteSensor = (req,res,next) => {

    Sensor.remove({\_id: req.params.id})

        .exec()

        .then(result => {

            res.status(200).json({

                message: "sensor deleted"

            });

        })

        .catch(err => {

            console.log(err);

            res.status(500).json({

                error:err

            });

        });

}

**SmsController.js**

const nodeMailer = require('nodemailer');

const texts = require('../constants/texts');

const emailConfig = texts.emailConfigure;

/\*

 \* POST request calls to the sendSms in SmsController class to send a sms

 \*/

exports.sendSms = (req, res, next) => {

    const {body} = req;

    const {

        FloorNb,

        roomNo,

        Co2Level,

        smokeLevel,

        email

    } = body;

    let message = "\*\*\*\*\* Warning \*\*\*\*\*\* \n"+ "Please pay Attention \n" +

                FloorNb + " th floor room nb "+ roomNo +

                '\n CO2 level : ' + Co2Level +

                '\n Smoke level : ' + smokeLevel + '\n'

    //print the sms in the console

    console.log(message);

    res.status(200).json({

        message: 'SMS successfully send'

    })

}

**EmailController.js**

// include the nodemailer module

const nodeMailer = require('nodemailer');

const texts = require('../constants/texts');

const emailConfig = texts.emailConfigure;

/\*

 \* POST request calls to the sendEmail in EmailController class to send a email

 \*/

exports.sendEmail = (req, res, next) => {

    //get the request details from the body

    const {body} = req;

    const {

        FloorNb,

        roomNo,

        Co2Level,

        smokeLevel,

        email

    } = body;

    //setup sender email

    const EmailSender = nodeMailer.createTransport({

        service: 'gmail',

        auth: {

            user: emailConfig.Email,

            pass: emailConfig.password

        }

    });

    //setup reciver email

    let mailOption = {

        from : emailConfig.Email,

        to : email,

        subject : 'Warning\n',

        text : 'Please pay Attention\n'+

        FloorNb +' th floor room nb '+ roomNo +

        '\n CO2 level : ' + Co2Level +

        '\n Smoke level : ' + smokeLevel + '\n'

    };

    //send email

    EmailSender.sendMail(mailOption, (err,info) => {

            if(err){

                console.log(err);

            }else{

                console.log('Email Sent : ' + info.response);

                res.status(200).json({

                    message: 'email successfully send'

                })

            }

    });

}

**Texts.js**

module.exports = {

    //constant values

    server: {

        //MongoDB application connection URL

        MONGODB\_URL : "mongodb+srv://chamil:Chamil123@firesensor-aurwy.gcp.mongodb.net/test?retryWrites=true&w=majority",

        DB\_CONNECTED : "Database is connected ",

        DB\_NOT\_CONNECTED : "Can not connect to the database ",

        SERVER : "Server is running on Port : "

    },

    //sender email and password

    emailConfigure : {

        Email : 'chamilpearson@gmail.com',

        password: 'Sliit#\*1996'

    }

}

**RMI SERVER**

**FireSensor.java (Server interface)**

package DekstopClient;

import java.io.IOException;

import org.json.JSONArray;

import org.json.JSONException;

/\*\*

 \*

 \* @author UDILUCH

 \*/

public interface FireSensor extends java.rmi.Remote{

        //get the initial temparature

        public String getSensor() throws java.rmi.RemoteException, IOException, JSONException;

        // set the update status

        public void updateStatus() throws java.rmi.RemoteException, IOException, JSONException;

        //get the updated status

        public String Getupdate() throws java.rmi.RemoteException, IOException, JSONException;

        //Expose remote method for registration of the server object's listner list

        public void addTemperatureListener(SensorListner listener )throws java.rmi.RemoteException;

        //remote method to unregister from the server object

        public void removeTemperatureListener(SensorListner listener )throws java.rmi.RemoteException;

        //add sensor details

        public String addSensorDetails(String id, int floorno, int roomNo)throws java.rmi.RemoteException, IOException, JSONException;

        //update the sensor details

        public String UpdateSensorDetails(String Sid, int floorno, int roomNo, String id,int co2, int smoke, String status)throws java.rmi.RemoteException, IOException, JSONException;

}

**FireSensorServer.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 DekstopClient;

import java.io.BufferedReader;

import java.io.IOException;

import java.io.InputStreamReader;

import java.io.OutputStream;

import java.net.HttpURLConnection;

import java.net.MalformedURLException;

import java.net.URL;

import java.rmi.Naming;

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.Scanner;

import java.util.Timer;

import java.util.TimerTask;

import java.util.logging.Level;

import java.util.logging.Logger;

import org.json.JSONArray;

import org.json.JSONException;

import org.json.JSONObject;

/\*\*

 \*

 \* @author UDILUCH

 \*/

public class FireSensorServer extends UnicastRemoteObject implements FireSensor{

    private ArrayList<SensorListner> list = new ArrayList<SensorListner>();

    String update = "";

    public FireSensorServer() throws java.rmi.RemoteException {

    super();

    }

    @Override

   public  String getSensor() throws RemoteException,IOException, JSONException {

//create a URL object with the target URI string that accepts the JSON data via HTTP GET method

        URL url= new URL("http://localhost:5000/sensor/");

        //invoke the openConnection method to get the HttpURLConnection object

     HttpURLConnection conn = (HttpURLConnection)url.openConnection();

     conn.setRequestMethod("GET");//send a GET request

     conn.connect(); //Open a connection stream to the corresponding API

     int responsecode = conn.getResponseCode(); //Get the corresponding response code

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

        String inputLine;

        StringBuffer response = new StringBuffer();

     //read the data line by line from the input stream using readLine method

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

            response.append(inputLine);

     } in .close();

     // print result

     System.out.println("\nJSON data in string format");

     System.out.println(response);

     return response.toString();

    }

//Implement the remote method to register listner in the listner list

    @Override

    public void addTemperatureListener(SensorListner listener) throws RemoteException {

        list.add(listener);

    }

    //Implement the remote method to unregister listner in the listner list

    @Override

    public void removeTemperatureListener(SensorListner listener) throws RemoteException {

       list.remove(listener);

    }

    @Override

    public String addSensorDetails(String id, int floorno, int roomNo) throws RemoteException, IOException, JSONException {

        String message = null;

        //creating a custom JSON String

        final String REQ\_PARAM = "{\n" + "\"FloorNo\": "+floorno+",\r\n" +

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

                        "    \"SensorID\": \""+id+"\"" + "\n}";

    System.out.println(REQ\_PARAM);

        //create a URL object with the target URI string that accepts the JSON data via HTTP POST method

        URL obj = new URL("http://localhost:5000/sensor/create/");

        //invoke the openConnection method to get the HttpURLConnection object

    HttpURLConnection postConnection = (HttpURLConnection) obj.openConnection();

    postConnection.setRequestMethod("POST"); //send a POST request

        //parameter has to be set to send the request body in JSON format

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

        // enable the URLConnection object's doOutput property to true

    postConnection.setDoOutput(true);

        //Open the DataOutputStream object

    OutputStream ost = postConnection.getOutputStream();

    ost.write(REQ\_PARAM.getBytes());

    ost.flush();

    ost.close();

    int responseCode = postConnection.getResponseCode(); //Check the response code

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

    System.out.println("POST Response Message : " + postConnection.getResponseMessage()); //print response message

    //response code is OK then create the input stream to read the returned data

    if (responseCode == HttpURLConnection.HTTP\_OK) { //success

            BufferedReader inBuf = new BufferedReader(new InputStreamReader(

            postConnection.getInputStream()));

            String input;

            StringBuffer response = new StringBuffer();

            //read the data line by line from the input stream using readLine method

            while ((input = inBuf .readLine()) != null) {

                response.append(input);

            } inBuf .close();

            // print result

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

            String inline2 = "["+ response+"]";

            //create json array

            JSONArray jsonar = new JSONArray(inline2);

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

                //read json array one by one using json object

                JSONObject album = jsonar.getJSONObject(i);

                message = album.getString("message");

            }

            return message; // return thr message

    } else {

        System.out.println("POST NOT WORKED");

    }

        return message;// return thr message

    }

 @Override

 public String UpdateSensorDetails(String Sid, int floorno, int roomNo, String id,int co2, int smoke, String status) throws RemoteException, IOException, JSONException {

        String message = null;

         //creating a custom JSON String

        final String REQ\_PARAM = "{\n" + "\"FloorNo\": "+floorno+",\r\n" +

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

                        "    \"Co2Level\": "+co2+",\r\n" +

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

                        "    \"status\": \""+status+"\",\r\n" +

                        "    \"SensorID\": \""+Sid+"\"" + "\n}";

        System.out.println(REQ\_PARAM);

        //create a URL object with the target URI string that accepts the JSON data via HTTP PUT method

        URL obj = new URL("http://localhost:5000/sensor/update/"+id);

        //invoke the openConnection method to get the HttpURLConnection object

        HttpURLConnection postConnection = (HttpURLConnection) obj.openConnection();

        postConnection.setRequestMethod("PUT"); //send a POST request

        //parameter has to be set to send the request body in JSON format

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

        // enable the URLConnection object's doOutput property to true

        postConnection.setDoOutput(true);

        //Open the DataOutputStream object

        OutputStream ost = postConnection.getOutputStream();

        ost.write(REQ\_PARAM.getBytes());

        ost.flush();

        ost.close();

        int responseCode = postConnection.getResponseCode(); //Check the response code

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

        System.out.println("POST Response Message : " + postConnection.getResponseMessage()); //print response message

        //response code is OK then create the input stream to read the returned data

        if (responseCode == HttpURLConnection.HTTP\_OK) { //success

            BufferedReader inBuf = new BufferedReader(new InputStreamReader(

            postConnection.getInputStream()));

            String input;

            StringBuffer response = new StringBuffer();

            //read the data line by line from the input stream using readLine method

            while ((input = inBuf .readLine()) != null) {

            response.append(input);

            } inBuf .close();

            // print result

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

            String inline2 = "["+ response+"]";

            //create json array

            JSONArray jsonar = new JSONArray(inline2);

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

                 //read json array one by one using json object

                JSONObject album = jsonar.getJSONObject(i);

                message = album.getString("message");

            }

            return message; // return thr message

    } else {

        System.out.println("POST NOT WORKED");

    }

        return message;// return thr message

    }

   // notify the listners

   private void notifyListeners(String result) throws IOException, JSONException{

        for(SensorListner Listner : list) {

            try {

                //call to the callback method

                Listner.SensorChanged(result);

            } catch (RemoteException e) {

                e.printStackTrace();

            }

        }

    }

    // update the sensor details every 15 seconds

    public void updateStatus(){

          // start timer object

          TimerTask task = new TimerTask() {

          @Override

          public void run() {

             try {

                   update = getSensor(); //get the update

                   sentEmailMsg(update); // call the sentEmailMsg method to send email and messages

                   SendAlertCo2Smoke(update); // notify the client

             } catch (IOException ex) {

               Logger.getLogger(FireSensorServer.class.getName()).log(Level.SEVERE, null, ex);

             } catch (JSONException ex) {

               Logger.getLogger(FireSensorServer.class.getName()).log(Level.SEVERE, null, ex);

             }

          }

        };

        Timer timer = new Timer();

        long delay = 0;

        long intevalPeriod = 1 \* 15000; // RMI sever get upto date every 15 seconds

        // schedules the task to be run in an interval

        timer.scheduleAtFixedRate(task, delay,intevalPeriod);

    }

// send notification to client using callback method

 public void SendAlertCo2Smoke(String result) throws IOException, JSONException {

        notifyListeners(result); // notify listner when co2 or smoke level > 5

 }

// send email, sms when co2 level or smoke level move to greater than 15

public void sentEmailMsg(String result) throws JSONException, MalformedURLException, IOException{

        JSONArray jsonar = new JSONArray(result.toString());

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

            JSONObject album = jsonar.getJSONObject(i);

            String email = "chamildilu@gmail.com";

            int co2   =  album.getInt("Co2Level");

            int smoke =  album.getInt("smokeLevel");

            int floor =  album.getInt("FloorNo");

            int room  =  album.getInt("roomNo");

            final String REQ\_PARAMS = "{\n" + "\"FloorNb\": "+floor+",\r\n" +

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

                                       "    \"Co2Level\": "+co2+",\r\n" +

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

                                       "    \"email\": \""+email+"\"" + "\n}";

            if(co2 > 5 || smoke > 5 ){

                System.out.println(REQ\_PARAMS);

                //create a URL object with the target URI string that accepts the JSON data via HTTP POST method

                URL obj = new URL("http://localhost:5000/email/send/");

                //invoke the openConnection method to get the HttpURLConnection object

                HttpURLConnection postConnection = (HttpURLConnection) obj.openConnection();

                postConnection.setRequestMethod("POST");//send a POST request

                //parameter has to be set to send the request body in JSON format

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

                // enable the URLConnection object's doOutput property to true

                postConnection.setDoOutput(true);

                //Open the DataOutputStream object

                OutputStream ost = postConnection.getOutputStream();

                ost.write(REQ\_PARAMS.getBytes());

                ost.flush();

                ost.close();

                int responseCode = postConnection.getResponseCode();

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

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

                //response code is OK then create the input stream to read the returned data

                if (responseCode == HttpURLConnection.HTTP\_OK) { //success

                    BufferedReader inBuf = new BufferedReader(new InputStreamReader(postConnection.getInputStream()));

                    String input;

                    StringBuffer response = new StringBuffer();

                    while ((input = inBuf .readLine()) != null) {

                response.append(input);

                    } inBuf .close();

                    // print result

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

                } else {

                    System.out.println("POST NOT WORKED");

                }

            }

            if(co2 > 5 || smoke > 5 ){

                URL obj = new URL("http://localhost:5000/sms/send/");

                HttpURLConnection postConnection = (HttpURLConnection) obj.openConnection();

                postConnection.setRequestMethod("POST");

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

                postConnection.setDoOutput(true);

                OutputStream ost = postConnection.getOutputStream();

                ost.write(REQ\_PARAMS.getBytes());

                ost.flush();

                ost.close();

                int responseCode = postConnection.getResponseCode();

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

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

                if (responseCode == HttpURLConnection.HTTP\_OK) { //success

                    BufferedReader inBuf = new BufferedReader(new InputStreamReader(postConnection.getInputStream()));

                    String input;

                    StringBuffer response = new StringBuffer();

                    while ((input = inBuf .readLine()) != null) {

                    response.append(input);

                    } inBuf .close();

                    // print result

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

                } else {

                    System.out.println("POST NOT WORKED");

                }

            }

        }

    }

    // return the updated sensor details

     public String Getupdate(){

         return update;

     }

    public static void main(String[] args)  {

    System.out.println("Loading temperature service");

    try {

            FireSensorServer sensor = new FireSensorServer();

            //register the sensor sever

            Registry reg = LocateRegistry.createRegistry(1099);

            reg.rebind("FireSensor", sensor);

            sensor.updateStatus();

            } catch (RemoteException re) {

        System.err.println("Remote Error - " + re);

            } catch (Exception e) {

        System.err.println("Error - " + e);

            }

    }

}

**RMI CLIENT**

**SensorListner.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 DekstopClient;

//this is callback interface

//this is also remote interface of the client

public interface SensorListner extends java.rmi.Remote{

    //This will expose the remote method which will give the changed Sensor deytails

    public void SensorChanged(String object) throws     java.rmi.RemoteException;

}

**SensorMonitor.java**

package DekstopClient;

import java.io.IOException;

import java.net.MalformedURLException;

import java.rmi.Naming;

import java.rmi.NotBoundException;

import java.rmi.Remote;

import java.rmi.RemoteException;

import java.rmi.server.UnicastRemoteObject;

import java.util.logging.Level;

import java.util.logging.Logger;

import org.json.JSONArray;

import org.json.JSONException;

/\*\*

 \*

 \* @author UDILUCH

 \*/

public class SensorMonitor extends UnicastRemoteObject implements SensorListner{

    static String reading, changed = null;

    static FireSensor sensor;

    public SensorMonitor() throws RemoteException {

    }

    public static void main(String[] args) throws RemoteException, IOException, JSONException, NotBoundException {

        try {

            String registration = "rmi://localhost:1099/FireSensor";

            Remote remoteService = Naming.lookup(registration);

            //getting the reference of remote server interface

             sensor = (FireSensor) remoteService;

            //This is blocking call

            reading = sensor.getSensor();

            SensorMonitor monitor = new SensorMonitor();

            //client object to register to the server

            sensor.addTemperatureListener(monitor);

         } catch (MalformedURLException mue) {

        }

    }

    // call RMI server to add details of the sensor

    public static String addDetails(String id, int floor, int room) throws IOException, RemoteException, JSONException, NotBoundException{

        String registration = "rmi://localhost:1099/FireSensor";

        Remote remoteService = Naming.lookup(registration);

        sensor = (FireSensor) remoteService;

        String result =  sensor.addSensorDetails(id, floor, room);

        return  result;

    }

    // call RMI server to Update details of the sensor

     public static String UpdateDetails(String Sensorid, int floor, int room, String id, int co2, int smoke, String status) throws IOException, RemoteException, JSONException, NotBoundException{

        String registration = "rmi://localhost:1099/FireSensor";

        Remote remoteService = Naming.lookup(registration);

        sensor = (FireSensor) remoteService;

        String result =  sensor.UpdateSensorDetails(Sensorid, floor, room, id, co2,smoke, status);

        return  result;

    }

   //implement the callback method at the client side

    public void SensorChanged(String object) throws RemoteException {

      changed = object;

      MonitorApp app = new MonitorApp();

        try {

            app.showAlert(object);// call showAtlert method to show alerts

        } catch (JSONException ex) {

            Logger.getLogger(SensorMonitor.class.getName()).log(Level.SEVERE, null, ex);

        }

    }

    //get the update of the sensor deatils

    public String getUpdate() throws NotBoundException, MalformedURLException, RemoteException, IOException, JSONException{

        String registration = "rmi://localhost:1099/FireSensor";

        Remote remoteService = Naming.lookup(registration);

        sensor = (FireSensor) remoteService;

        return sensor.Getupdate();

    }

}

**Admin Login / Register GUI(UILogin.java)**

**Sign up**

  if (!RegUserNAmeText.getText().trim().isEmpty() & !RegPaswrdText.getText().trim().isEmpty() ) {

            String username = RegUserNAmeText.getText();

            String paswrd = RegPaswrdText.getText();

               if(!username.equals("Enter your Username") & !paswrd.equals("mmmmmmmm"))

                {

                     boolean value = false;

                try {

                   value = login.SignUpPOST(username, paswrd);

                } catch (ProtocolException ex) {

                    Logger.getLogger(UILogin.class.getName()).log(Level.SEVERE, null, ex);

                } catch (IOException ex) {

                    Logger.getLogger(UILogin.class.getName()).log(Level.SEVERE, null, ex);

                } catch (JSONException ex) {

                    Logger.getLogger(UILogin.class.getName()).log(Level.SEVERE, null, ex);

                }

                    if(value)

                    {

                       JOptionPane.showMessageDialog(null, "Registration Successfull");

                } else {

                     JOptionPane.showMessageDialog(null, "Registration Faild");

                }

            }else{

                  JOptionPane.showMessageDialog(null, "Please fill all feilds");

            }

        } else if (!RegUserNAmeText.getText().trim().isEmpty() & !RegPaswrdText.getText().trim().isEmpty() ) {

            JOptionPane.showMessageDialog(null, "Please enter Confirmation password!");

        } else if (!RegUserNAmeText.getText().trim().isEmpty() & RegPaswrdText.getText().trim().isEmpty() ) {

            JOptionPane.showMessageDialog(null, "Please enter Password!");

        } else if (RegUserNAmeText.getText().trim().isEmpty() & !RegPaswrdText.getText().trim().isEmpty() ) {

            JOptionPane.showMessageDialog(null, "Please enter username!");

        }

        else {

            JOptionPane.showMessageDialog(null, "Please fill all feilds");

        }

    }//GEN-LAST:event\_SignUpREGMouseClicked

**Sign In**

 if (!LoginUserNAme.getText().trim().isEmpty() && !LoginPaswrd.getText().trim().isEmpty()) {

            String userN = LoginUserNAme.getText().trim();

            String pass = LoginPaswrd.getText().trim();

             if(!userN.equals("Enter your Username") & !pass.equals("mmmmmmmm") )

             {

              boolean value = false;

                try {

                   value = login.LoginPOST(userN, pass);

                } catch (ProtocolException ex) {

                    Logger.getLogger(UILogin.class.getName()).log(Level.SEVERE, null, ex);

                } catch (IOException ex) {

                    Logger.getLogger(UILogin.class.getName()).log(Level.SEVERE, null, ex);

                } catch (JSONException ex) {

                    Logger.getLogger(UILogin.class.getName()).log(Level.SEVERE, null, ex);

                }

                if (value) {

                   MonitorApp ctool = new MonitorApp();

                   ctool.check = true;

                    ctool.setLogout();

                   ctool.setVisible(true);

                   this.dispose();

            } else {

                JOptionPane.showMessageDialog(null, "Login does not exist!");

            }

            }

             else if (!LoginPaswrd.getText().trim().isEmpty() & LoginUserNAme.getText().trim().isEmpty() ) {

                JOptionPane.showMessageDialog(null, "Please fill all feilds");

            }

             else if (!LoginUserNAme.getText().trim().isEmpty()  & LoginPaswrd.getText().trim().isEmpty() ) {

            JOptionPane.showMessageDialog(null, "Please enter  password!");

        } else if (!LoginPaswrd.getText().trim().isEmpty() & LoginUserNAme.getText().trim().isEmpty() ) {

            JOptionPane.showMessageDialog(null, "Please enter Username!");

        }else {

            JOptionPane.showMessageDialog(null, "Please fill all feilds");

        }

     }

  }//GEN-LAST:event\_jLabel10MouseClicked

**Login.java**

package DekstopClient;

import java.io.BufferedReader;

import java.io.IOException;

import java.io.InputStreamReader;

import java.io.OutputStream;

import java.net.HttpURLConnection;

import java.net.MalformedURLException;

import java.net.ProtocolException;

import java.net.URL;

import org.json.JSONArray;

import org.json.JSONException;

import org.json.JSONObject;

/\*\*

 \*

 \* @author UDILUCH

 \*/

public class login {

    public static boolean SignUpPOST(String username,String password) throws MalformedURLException, ProtocolException, IOException, JSONException{

            boolean sucess = false;

            //creating a custom JSON String

        final String REQ\_PARAMS = "{\n" +

                "    \"Type\": \"User\",\r\n" +

                "    \"Username\": \""+username+"\",\r\n" +

                "    \"Password\": \""+password+"\"" + "\n}";

        System.out.println(REQ\_PARAMS);

            //create a URL object with the target URI string that accepts the JSON data via HTTP POST method

        URL obj = new URL("http://localhost:5000/user/sign-up");

            //invoke the openConnection method to get the HttpURLConnection object

        HttpURLConnection postConnection = (HttpURLConnection) obj.openConnection();

        postConnection.setRequestMethod("POST");//send a POST request

            //parameter has to be set to send the request body in JSON format

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

            // enable the URLConnection object's doOutput property to true

        postConnection.setDoOutput(true);

            //Open the DataOutputStream object

        OutputStream ost = postConnection.getOutputStream();

        ost.write(REQ\_PARAMS.getBytes());

        ost.flush();

        ost.close();

        int responseCode = postConnection.getResponseCode();//Check the response code

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

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

            //response code is OK then create the input stream to read the returned data

        if (responseCode == HttpURLConnection.HTTP\_OK) { //success

            BufferedReader inBuf = new BufferedReader(new InputStreamReader(

                postConnection.getInputStream()));

            String input;

            StringBuffer response = new StringBuffer();

                //read the data line by line from the input stream using readLine method

            while ((input = inBuf .readLine()) != null) {

                response.append(input);

            } inBuf .close();

            // print result

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

                String inline2 = "["+ response+"]";

        //create json array

        JSONArray jsonar = new JSONArray(inline2);

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

                    //read json array one by one using json object

                    JSONObject album = jsonar.getJSONObject(i);

                    sucess = album.getBoolean("success");

        }

                return sucess;

        } else {

            System.out.println("POST NOT WORKED");

        }

            return sucess;

    }

    public static boolean LoginPOST(String username,String password) throws MalformedURLException, ProtocolException, IOException, JSONException{

            boolean sucess = false;

            //creating a custom JSON String

        final String REQ\_PARAMS = "{\n" +

                "    \"Username\": \""+username+"\",\r\n" +

                "    \"Password\": \""+password+"\"" + "\n}";

        System.out.println(REQ\_PARAMS);

            //create a URL object with the target URI string that accepts the JSON data via HTTP POST method

        URL obj = new URL("http://localhost:5000/user/sign-in");

            //invoke the openConnection method to get the HttpURLConnection object

        HttpURLConnection postConnection = (HttpURLConnection) obj.openConnection();

        postConnection.setRequestMethod("POST");//send a POST request

            //parameter has to be set to send the request body in JSON format

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

            // enable the URLConnection object's doOutput property to true

        postConnection.setDoOutput(true);

            //Open the DataOutputStream object

        OutputStream ost = postConnection.getOutputStream();

        ost.write(REQ\_PARAMS.getBytes());

        ost.flush();

        ost.close();

        int responseCode = postConnection.getResponseCode();

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

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

            //response code is OK then create the input stream to read the returned data

        if (responseCode == HttpURLConnection.HTTP\_OK) { //success

            BufferedReader inBuf = new BufferedReader(new InputStreamReader(

                postConnection.getInputStream()));

            String input;

            StringBuffer response = new StringBuffer();

                //read the data line by line from the input stream using readLine method

            while ((input = inBuf .readLine()) != null) {

                response.append(input);

            } inBuf .close();

            // print result

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

                String inline2 = "["+ response+"]";

               //create json array

         JSONArray jsonar = new JSONArray(inline2);

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

                    //read json array one by one using json object

                    JSONObject album = jsonar.getJSONObject(i);

                    sucess = album.getBoolean("success");

         }

return sucess;

        } else {

            System.out.println("POST NOT WORKED");

        }

            return sucess;

    }

}

**Fire Sensor Monitor GUI(MonitorApp.java)**

**Sensor add**

  String Sid = SensorID.getText();

        int floor = Integer.parseInt(FloorNo.getText());

        int room = Integer.parseInt(roomNo.getText());

          try {

              SensorMonitor sensor1 = new SensorMonitor();

              String result = sensor1.addDetails(Sid, floor, room);

              JOptionPane.showMessageDialog(null, result);

          } catch (IOException ex) {

              Logger.getLogger(MonitorApp.class.getName()).log(Level.SEVERE, null, ex);

          } catch (JSONException ex) {

              Logger.getLogger(MonitorApp.class.getName()).log(Level.SEVERE, null, ex);

          } catch (NotBoundException ex) {

              Logger.getLogger(MonitorApp.class.getName()).log(Level.SEVERE, null, ex);

          }

       id.setText("");

       SensorID.setText("");

       FloorNo.setText("");

**Get the table record to the form(by clicking table row)**

  saveBtn.setVisible(false);

       updateBtn.setVisible(true);

       int i = SensorTable.getSelectedRow();

       TableModel model = SensorTable.getModel();

       //get the selected row details

       String Sensorid = model.getValueAt(i, 0).toString();

       String floor =  model.getValueAt(i, 1).toString() ;

       String room =  model.getValueAt(i, 2).toString() ;

       String co2 =   model.getValueAt(i, 3).toString() ;

       String smoke =  model.getValueAt(i, 4).toString() ;

       String status =  model.getValueAt(i, 5).toString() ;

       String Iid =  model.getValueAt(i, 6).toString() ;

       id.setText(Iid);

       SensorID.setText(Sensorid);

       FloorNo.setText(floor);

       roomNo.setText(room);

       Co2text.setText(co2);

       SmokeText.setText(smoke);

       StatusText.setText(status);

**Update selected sensor details**

  String Sid = SensorID.getText();

        String status = StatusText.getText();

        int floor = Integer.parseInt(FloorNo.getText());

        int room = Integer.parseInt(roomNo.getText());

        int co2 = Integer.parseInt(Co2text.getText());

        int smoke = Integer.parseInt(SmokeText.getText());

        String Iid = id.getText();

          try {

              SensorMonitor sensor1 = new SensorMonitor();

              String result = sensor1.UpdateDetails(Sid, floor, room, Iid,co2,smoke,status); // update the sensor details

              JOptionPane.showMessageDialog(null, result);

          } catch (IOException ex) {

              Logger.getLogger(MonitorApp.class.getName()).log(Level.SEVERE, null, ex);

          } catch (JSONException ex) {

              Logger.getLogger(MonitorApp.class.getName()).log(Level.SEVERE, null, ex);

          } catch (NotBoundException ex) {

              Logger.getLogger(MonitorApp.class.getName()).log(Level.SEVERE, null, ex);

          }

       saveBtn.setVisible(true);

       updateBtn.setVisible(false);

       id.setText("");

       SensorID.setText("");

       FloorNo.setText("");

       roomNo.setText("");

       Co2text.setText("");

       SmokeText.setText("");

       StatusText.setText("");

**Table details show and get the update every 30 seconds**

   public void showStatmentDetails()

    {

        // each 30 seconds update the Dekstop client object

         TimerTask task = new TimerTask() {

         @Override

         public void run() {

         try {

            DefaultTableModel dft = (DefaultTableModel)SensorTable.getModel();

            dft.setRowCount(0);

             SensorMonitor sensor1 = new SensorMonitor();

             String result = sensor1.getUpdate();

             System.out.println(result);

             JSONArray jsonar = new JSONArray(result.toString());

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

        JSONObject album = jsonar.getJSONObject(i);

                // set the value in to the table

                Vector vector = new Vector();

                vector.add( album.getString("SensorID") );

                vector.add( album.getInt("FloorNo") );

                vector.add( album.getInt("roomNo") );

                vector.add( album.getInt("Co2Level") );

                vector.add( album.getInt("smokeLevel") );

                vector.add( album.getString("status") );

                vector.add( album.getString("\_id") );

                dft.addRow(vector);

            }

        } catch (Exception e) {

      }

    }

  };

    Timer timer = new Timer();

    long delay = 0;

    long intevalPeriod = 1 \* 30000;

    // schedules the task to be run in an interval

    timer.scheduleAtFixedRate(task, delay,intevalPeriod);

}

**Callback Alert**

 //show alert when co2 or smoke level > 5

    public void showAlert(String result) throws JSONException{

           JSONArray jsonar = new JSONArray(result.toString());

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

         JSONObject album = jsonar.getJSONObject(i);

                 String sensorid = album.getString("SensorID");

                  if(album.getInt("Co2Level") > 5 ){

                    String co2 = sensorid + " : " + "CO2 level increase - " + album.getInt("Co2Level");

                    JOptionPane.showMessageDialog(null,co2);

                }

                if(album.getInt("smokeLevel") > 5){

                    String smoke = sensorid + " : " + "Smoke level increase - " + album.getInt("smokeLevel");

                    JOptionPane.showMessageDialog(null, smoke);

                }

           }

    }

**Sensor APP (update REST API every 10 seconds)**

public class timer {

          public static void main(String[] args) {

            TimerTask task = new TimerTask() {

            @Override

            public void run() {

                // task to run goes here

                Random randomGenerator=new Random();

                try {

                    //get the sensor details as JSON every 10 seconds

                     JSONArray jsonar = (JSONArray) getFireDetails();

                     String id = null;

                     String Sensorid = null;

                     int floor, room;

                     String[] arr={"active", "deactive"};

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

                         //get the one JSON object

                         JSONObject album = jsonar.getJSONObject(i);

                         id = album.getString("\_id");

                         Sensorid = album.getString("SensorID");

                         floor = album.getInt("FloorNo");

                         room = album.getInt("roomNo");

                         int co2 = randomGenerator.nextInt(10) + 1; // set the random number between 0-10

                         int smoke = randomGenerator.nextInt(10) + 1;// set the random number between 0-10

                         int idx = randomGenerator.nextInt(arr.length);

                         String status = (arr[idx]);                   // set the status as random

                         //call update sensor to update the details  every 10 seconds.

                         updateSensor(id, co2, smoke,status,Sensorid,floor,room);

                     }

                } catch (JSONException | IOException e) {

                    // TODO Auto-generated catch block

                    e.printStackTrace();

                }

              }

            };

            Timer timer = new Timer();

            long delay = 0;

            long intevalPeriod = 1 \* 10000; // set the time interval as 10 seconds

            // schedules the task to be run in an interval

            timer.scheduleAtFixedRate(task, delay,intevalPeriod);

          } // end of main

          public static Object getFireDetails() throws JSONException, IOException {

                 //create a URL object with the target URI string that accepts the JSON data via HTTP GET method

                 URL url= new URL("http://localhost:5000/sensor/");

                 //invoke the openConnection method to get the HttpURLConnection object

                 HttpURLConnection conn = (HttpURLConnection)url.openConnection();

                 conn.setRequestMethod("GET");//send a GET request

                 conn.connect();//Open a connection stream to the corresponding API

                 int responsecode = conn.getResponseCode();

                 BufferedReader inBuf = new BufferedReader(new InputStreamReader(conn.getInputStream()));

                 String input;

                 StringBuffer response = new StringBuffer();

                 while ((input = inBuf .readLine()) != null) {

                      response.append(input);

                 } inBuf .close();

                 // print result

                 System.out.println("\nJSON data in string format");

                 System.out.println(response);

                 // get the JSON response

                 JSONArray jsonar = new JSONArray(response.toString());

                 return jsonar; // return the response as JSON array

          }

          // this method update the rest API every 10 seconds

          private static void updateSensor(String id, int co2, int smoke, String status,String sensorId, int floor, int room) throws IOException {

              //creating a custom JSON String

                final String REQ\_PARAMS = "{\n" + "\"FloorNo\": "+floor+",\r\n" +

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

                        "    \"Co2Level\": "+co2+",\r\n" +

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

                        "    \"status\": \""+status+"\",\r\n" +

                        "    \"SensorID\": \""+sensorId+"\"" + "\n}";

                System.out.println(REQ\_PARAMS);

                //create a URL object with the target URI string that accepts the JSON data via HTTP PUT method

                URL obj = new URL("http://localhost:5000/sensor/update/"+id);

                HttpURLConnection postConnection = (HttpURLConnection) obj.openConnection();

                postConnection.setRequestMethod("PUT"); //send a POST request

                //parameter has to be set to send the request body in JSON format

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

                // enable the URLConnection object's doOutput property to true

                postConnection.setDoOutput(true);

                //Open the DataOutputStream object

                OutputStream ost = postConnection.getOutputStream();

                ost.write(REQ\_PARAMS.getBytes());

                ost.flush();

                ost.close();

                int responseCode = postConnection.getResponseCode();

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

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

                //response code is OK then create the input stream to read the returned data

                if (responseCode == HttpURLConnection.HTTP\_OK) { //success

                    BufferedReader inBuf = new BufferedReader(new InputStreamReader(postConnection.getInputStream()));

                    String input;

                    StringBuffer response = new StringBuffer();

                    while ((input = inBuf .readLine()) != null) {

                        response.append(input);

                    } inBuf .close();

                    // print result

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

                } else {

                    System.out.println("POST NOT WORKED");

                }

            }

}

END OF THE REPORT

THANK YOU