**Fire Alarm Monitoring System**

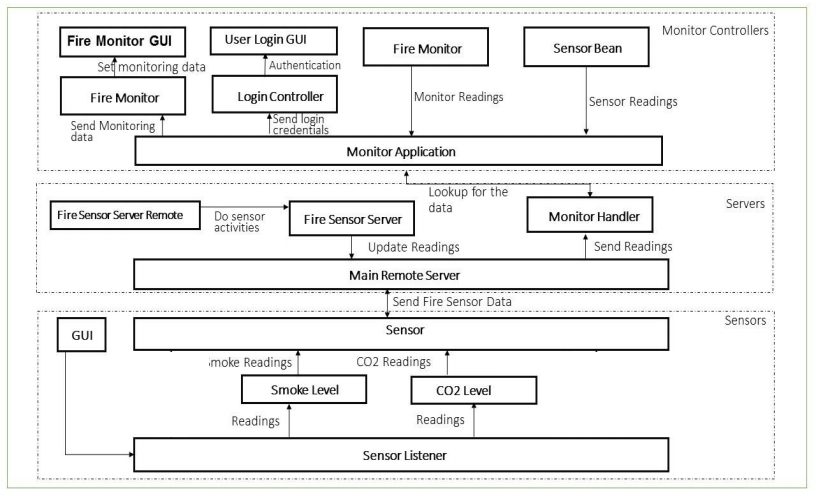
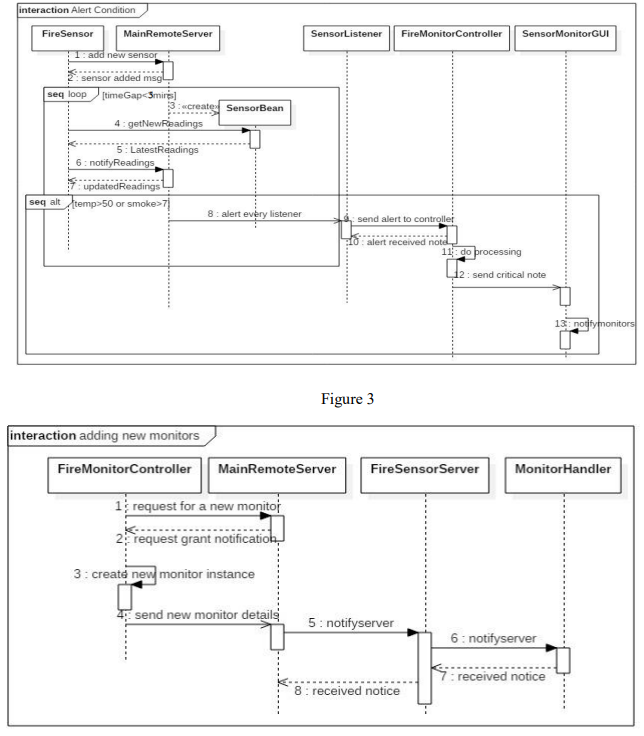
**Report**

**DS Assignment 2**

**Done by:**

**IT18001594 – S.M. Adheeb**

**High Level Architectural Diagram**

******Sequence Diagrams**

}

Appendix

RMI Client & Server

Java File Name: FireSensorServer

/\*\*

\* This is the server class which takes care of the activities regarding the sensor & monitor

\*/

**public** **class** FireSensorServer **extends** UnicastRemoteObject **implements** FireSensorServerRemote{

**private** **static** **final** **long** ***serialVersionUID*** = 1L;

**private** List<Sensor> fSensor= **new** ArrayList<>();

//FireSensorMonitors Connected with the RemoteServer

**private** List<SensorListener> fMonitor = **new** ArrayList<>();

**public** FireSensorServer() **throws** RemoteException{}

@Override

**public** **int** getConnectedScreensCount() **throws** RemoteException{

**return** fMonitor.size();

}

@Override

**public** **int** getFSensorCount() **throws** RemoteException{

**return** fSensor.size();

}

@Override

**public** List<Sensor> getTheSensorType() **throws** RemoteException {

**return** fSensor;

}

@Override

**public** **void** getNewReading(SensorListener sensorListener) **throws** RemoteException {

sensorListener.ModifyTheReading(fSensor, getConnectedScreensCount(), getFSensorCount());

}

@Override

**public** **void** addFSensorMonitor(SensorListener sensorListener) **throws** RemoteException {

System.***out***.println(sensorListener + " has been added");

**synchronized**(fMonitor){

fMonitor.add(sensorListener);

}

**new** MonitorHandler(**this**,fSensor).start();

}

@Override

**public** Sensor getNewReadingsByFloor(SensorListener sensorListener, **int** floorNo, **int** roomNo) **throws** Exception {

**for**(Sensor sensor : fSensor){

**if** (sensor.getFloorNo() == floorNo && sensor.getRoomNo() == roomNo){

**return** sensor;

}

}

**throw** **new** Exception();

}

**public** **void** addNewSensor(Sensor sensor) {

System.***out***.println(sensor + " has been added");

**synchronized**(fSensor){

**this**.fSensor.add(sensor);

}

}

**public** **void** removeExistingSensor(Sensor sensor) {

System.***out***.println(sensor + " has been removed");

**synchronized**(fSensor){

**this**.fSensor.remove(sensor);

}

}

**public** **void** makeAlertToUser(Sensor sensor){

**if**(!fMonitor.isEmpty()){

// Notify all the listeners

fMonitor.forEach((listener) -> {

**try** {

System.***out***.println(sensor + " seems to be in danger.");

// alerting the monitor

listener.notifyAlert(sensor);

}

**catch** (ConnectException e){

// remove listeners if failure occurs

fMonitor.remove(listener);

}**catch** (RemoteException r) {

System.***err***.println(r.getMessage());

}

});

}

}

**public** **void** notifyListeners(List<Sensor> sensors) {

**if**(!fMonitor.isEmpty()){

// Notify all the listeners

fMonitor.forEach((listener) -> {

**try** {

//readings get updated

listener.ModifyTheReading(sensors, getConnectedScreensCount(), getFSensorCount());

} **catch** (ConnectException e){

fMonitor.remove(listener);

System.***err***.println(e);

} **catch** (RemoteException e) {

System.***err***.println(e);

}

});

}

}

**public** **void** makeAlertsOnFailure(Sensor sensor){

**if**(!fMonitor.isEmpty()){

fMonitor.forEach((listener) -> {

**try** {

System.***out***.println(sensor + " is failing");

listener.makeAlertsOnFailures(sensor);

}

**catch** (ConnectException e){

fMonitor.remove(listener);

}**catch** (RemoteException e) {

System.***err***.println(e.getMessage());

}

});

}

}

}

Java File Name: FireSensorServerRemote

//RMI Call back methods contained in this class

**public** **interface** FireSensorServerRemote **extends** Remote{

**public** **int** getFSensorCount() **throws** RemoteException;

**public** **int** getConnectedScreensCount() **throws** RemoteException;

**public** **void** addFSensorMonitor(SensorListener sensorListener) **throws** RemoteException;

**public** **void** getNewReading(SensorListener sensorListener) **throws** RemoteException ;

**public** Sensor getNewReadingsByFloor(SensorListener sensorListener, **int** floorNo, **int** roomNo) **throws** Exception ;

**public** List<Sensor> getTheSensorType() **throws** RemoteException;

}

Java File Name: MainRemoteServer

// The class where the rmi registry is created and the service is looked up

**public** **class** MainRemoteServer {

**private** **static** **final** **int** ***PORT\_OF\_SERVER*** = 9001;

**private** **static** FireSensorServer *fSensorServer*;

**public** **static** **void** main(String[] args) **throws** Exception {

System.***out***.println("Getting the server......");

**try**{

*fSensorServer* = **new** FireSensorServer();

FireAlarmSensor fSensor = **new** FireAlarmSensor();

Registry reg = LocateRegistry.*createRegistry*(1099);

Naming.*rebind*("//127.0.0.1/fireSensor", *fSensorServer*);

Thread.*sleep*(700);

System.***out***.println("Server has been Connected.");

}

**catch**(RemoteException e){

System.***err***.println(e);

}

**try** (ServerSocket ss = **new** ServerSocket(***PORT\_OF\_SERVER***)) {

**while** (**true**) {

SensorController sensorController = **new** SensorController(ss.accept(),*fSensorServer*);

sensorController.start();

}

}

**catch**(RemoteException e){

System.***err***.println(e);

}

}

}

Java File Name: MonitorHandler

// Class to handle clients which are connected to the server

**public** **class** MonitorHandler **extends** Thread **implements** Runnable {

**private** FireSensorServer fSS;

**private** List<Sensor> allSensors;

**public** MonitorHandler(FireSensorServer fireSensorServer, List<Sensor> sensors) {

**this**.fSS = fireSensorServer;

**this**.allSensors = sensors;

}

@Override

**public** **void** run(){

**while**(**true**){

**try** {

Thread.*sleep*(56250);

**synchronized**( allSensors ){

fSS.notifyListeners(allSensors);

}

System.***out***.println("Notification is done Successfully!");

} **catch** (InterruptedException e) {

System.***err***.println(e.getMessage());

}

}

}

}

Java File Name: FireMonitorController

// Class where all callback methods are implemented and is also the middle-tier between server & client to transfer readings.

**public** **class** FireMonitorController **extends** UnicastRemoteObject **implements** SensorListener {

**private** MonitorApplication mAMonitorController;

**private** FireSensorServerRemote fSRemoteServer;

**private** FireMonitorController fMonitor;

**private** Sensor sensor;

**private** FireMonitorGUI fMonitorGUI;

**public** FireMonitorController(FireSensorServerRemote aSensor) **throws** RemoteException{

**this**.fSRemoteServer = aSensor;

**this**.mAMonitorController = **new** MonitorApplication();

}

**public** **void** showMonitor(FireMonitorController fireMon) **throws** RemoteException{

fMonitor = fireMon;

fMonitorGUI = **new** FireMonitorGUI(fireMon);

fMonitorGUI.setDefaultCloseOperation(JFrame.***EXIT\_ON\_CLOSE***);

fMonitorGUI.setVisible(**true**);

fMonitorGUI.setTheMonitorCount(fSRemoteServer.getConnectedScreensCount());

fMonitorGUI.setTheSensorCount(fSRemoteServer.getFSensorCount());

fireMon.mAMonitorController.addTheMonitor(fMonitorGUI, fireMon.fSRemoteServer.getTheSensorType());

}

**public** **void** updateCurrentReadings(){

**try** {

fSRemoteServer.getNewReading(fMonitor);

} **catch** (RemoteException x) {

System.***out***.println(x);

}

}

@Override

**public** **void** makeAlertsOnFailures(Sensor sensor) **throws** RemoteException {

JOptionPane.*showMessageDialog*(fMonitorGUI,"Floor "+sensor.getFloorNo()+" Sensors seem to be failing","Warning",JOptionPane.***ERROR\_MESSAGE***);

}

@Override

**public** **void** ModifyTheReading(List<Sensor> sensors, **int** mCount, **int** sCount) **throws** RemoteException {

fMonitorGUI.setTheMonitorCount(mCount);

fMonitorGUI.setTheSensorCount(sCount);

**if**( !sensors.isEmpty() || sensors != **null** ){

**this**.mAMonitorController.addTheMonitor(fMonitorGUI, sensors);

}

**else**{

JOptionPane.*showMessageDialog*(**null**,"Message" ,"Restart Server",JOptionPane.***WARNING\_MESSAGE***);

}

}

**public** **void** getNewestReadingsByFloorNo(**int** floorNo, **int** roomNo){

**try** {

Sensor sensor = fSRemoteServer.getNewReadingsByFloor(fMonitor, floorNo, roomNo);

mAMonitorController.updateTheMonitor(fMonitorGUI, sensor);

} **catch** (Exception e) {

System.***err***.println(e.getMessage());

}

}

@Override

**public** **void** notifyAlert(Sensor sensor) **throws** RemoteException {

updateCurrentReadings();

JOptionPane.*showMessageDialog*(fMonitorGUI,"Floor "+sensor.getFloorNo()+ " Room " + sensor.getRoomNo() + " might be in danger. Message sent to email!","Warning",JOptionPane.***WARNING\_MESSAGE***);

}

}

Java File Name: LoginController

//Class to validate client

**public** **class** LoginController {

**public** **static** **boolean** validateUser(String uName, String pwd) **throws** IOException{

String[] logins;

String username, password, identity;

**try** ( BufferedReader bufferedReader = **new** BufferedReader(**new** InputStreamReader

(LoginController.**class**.getResourceAsStream("UserLoginDetails.txt")))) {

**while** ((identity = bufferedReader.readLine()) != **null**) {

logins = identity.split("-");

username = logins[0];

password = logins[1];

**if**(username.equals(uName) && password.equals(pwd)){

**return** **true**;

}

}

} **catch** (IOException e) {

System.***err***.println(e.getMessage());

}

**return** **false**;

}

}

Java File Name: MonitorApplication

// Class where components to execute fire monitors are initialized.

**public** **class** MonitorApplication {

**private** Map<Integer,SensorMonitorGUI> sensMonitors = **new** HashMap<>();

**private** List<Integer> sensLocations = **new** ArrayList<>();

**public** **void** intializeTheMonitor(){

**try** {

FireSensorServerRemote fSensServerRemote = (FireSensorServerRemote)

Naming.*lookup*("rmi://localhost/fireSensor");

FireMonitorController monitor = **new** FireMonitorController(fSensServerRemote);

fSensServerRemote.addFSensorMonitor(monitor);

monitor.showMonitor(monitor);

} **catch** (RemoteException | MalformedURLException | NotBoundException e) {

System.***err***.println(e.getMessage());

}

}

**public** **void** addTheMonitor(FireMonitorGUI aMonitorGUI,List<Sensor> allSensors){

allSensors.forEach(sensor ->{

SensorMonitorGUI monitor;

**int** floorNo = sensor.getFloorNo();

**int** roomNo = sensor.getRoomNo();

**if**(!sensMonitors.containsKey(floorNo)){

monitor = **new** SensorMonitorGUI(floorNo, roomNo);

sensMonitors.put(floorNo, monitor);

aMonitorGUI.addNewMonitor(monitor);

sensLocations.add(floorNo);

}

**else**{

monitor = sensMonitors.get(sensor.getFloorNo());

}

monitor.setSmokeLevel(sensor.getNewReading().getSmokeLevel());

monitor.setCO2Level(sensor.getNewReading().getCO2Level());

});

}

**public** **void** updateTheMonitor(FireMonitorGUI amonitorGUI, Sensor sens){

**if**(sensMonitors.containsKey(sens.getFloorNo())){

SensorMonitorGUI monitor = sensMonitors.get(sens.getFloorNo());

monitor.setSmokeLevel(sens.getNewReading().getSmokeLevel());

monitor.setCO2Level(sens.getNewReading().getCO2Level());

}

}

}

Java File Name: SensorController

//Sensors related activities implementation

**public** **class** SensorController **extends** Thread **implements** Runnable {

**private** SensorObject newReading;

**private** String inputtedSensor;

**private** **final** Socket socket;

**private** **final** Sensor sens;

**private** **static** HashSet<String> *allSensorTypes* = **new** HashSet<String>();

**private** **static** Map<String,Sensor> *theMappedSensors* = **new** HashMap<>();

**private** FireSensorServer fSensorServer;

**private** String password, input;

**private** **int** floorNo, roomNo;

**public** SensorController(Socket socket, FireSensorServer fSensorServer) {

**this**.socket = socket;

**this**.sens = **new** Sensor();

**this**.fSensorServer = fSensorServer;

**this**.newReading = **new** SensorObject();

}

@Override

**public** **void** run() {

BufferedReader br;

PrintWriter printWriter;

**try** {

br = **new** BufferedReader(**new** InputStreamReader(socket.getInputStream()));

printWriter = **new** PrintWriter(socket.getOutputStream(), **true**);

**while** (**true**) {

printWriter.println("GETNAME");

inputtedSensor = br.readLine();

**if** (inputtedSensor == **null**) {

System.***out***.println("Enter a valid name");

**return**;

}

**synchronized** (*allSensorTypes*) {

**if** (!*allSensorTypes*.contains(inputtedSensor)) {

*allSensorTypes*.add(inputtedSensor);

**break**;

}

}

}

**while** (**true**){

printWriter.println("GETPASSWORD");

password = br.readLine();

**if** (password == **null**) {

System.***out***.println("Enter a valid password");

**return**;

}

**if**(*validateSensorUser*(inputtedSensor, password)){

**break**;

}

**else** {

System.***out***.println("Enter a valid password");

**return**;

}

}

**while** (**true**) {

printWriter.println("GETFLOOR");

floorNo = Integer.*parseInt*(br.readLine());

**if** (floorNo == 0) {

System.***out***.println("Enter a valid Floor no");

**return**;

}

sens.setFloorNo(floorNo);

**break**;

}

**while** (**true**) {

printWriter.println("GETROOM");

roomNo = Integer.*parseInt*(br.readLine());

**if** (roomNo == 0) {

System.***out***.println("Enter a valid Room no");

**return**;

}

sens.setRoomNo(roomNo);

**break**;

}

printWriter.println("REGISTEREDSENSOR");

fSensorServer.addNewSensor(sens);

*theMappedSensors*.put(inputtedSensor, sens);

**while** (**true**) {

input = br.readLine();

**if** (input == **null**) {

**return**;

}

**try**(ObjectInputStream objc = **new** ObjectInputStream

(**new** ByteArrayInputStream(Base64.*getDecoder*().decode(input)))){

newReading = (SensorObject) objc.readObject();

sens.setLatestReading(newReading);

*theMappedSensors*.entrySet().forEach(y->System.***out***.println(y.getValue()));

authenticateAlertConditions(sens);

} **catch** (IOException | ClassNotFoundException e) {

System.***err***.println(e.getMessage());

}

}

} **catch** (IOException e) {

System.***err***.println(e.getMessage());

} **finally** {

fSensorServer.removeExistingSensor(sens);

*allSensorTypes*.remove(inputtedSensor);

**try** {

socket.close();

} **catch** (IOException e) {

System.***err***.println(e.getMessage());

}

}

}

**public** **static** **boolean** validateSensorUser(String uName, String pwd){

String identity, username, password;

String[] dataArray;

**try** (BufferedReader br = **new** BufferedReader(**new** InputStreamReader

(SensorController.**class**.getResourceAsStream("SensorLoginsDetails.txt")))) {

**while** ( **null** != (identity = br.readLine()) ) {

dataArray = identity .split("-");

username = dataArray[0];

password = dataArray[1];

**if**(username.equals(uName) && password.equals(pwd)){

**return** **true**;

}

}

**return** **false**;

} **catch** (IOException e) {

System.***err***.println(e.getMessage());

}

**return** **false**;

}

**public** **void** authenticateAlertConditions(Sensor sensor){

**int** smke = sensor.getNewReading().getSmokeLevel();

**double** co2 = sensor.getNewReading().getCO2Level();

**if**( co2 > 5.0 || smke > 5){

System.***out***.println("In a critical Situation");

fSensorServer.makeAlertToUser(sensor);

}

}}

Java File Name: SensorObject

//Setters and Getters of Sensor Object

**public** **class** SensorObject **implements** Serializable {

**private** **static** **final** **long** ***serialVersionUID*** = 1L;

**private** **double** cO2Level;

**private** **int** smokeLevel;

**public** **double** getCO2Level() {

**return** cO2Level;

}

**public** **void** setCO2Level(**double** co2Level) {

**this**.cO2Level = co2Level;

}

**public** **int** getSmokeLevel() {

**return** smokeLevel;

}

**public** **void** setSmokeLevel(**int** smokeLevel) {

**this**.smokeLevel = smokeLevel;

}

@Override

**public** String toString() {

**return** "Sensor Readings as of now \n"

+ "Smoke Level is: " + smokeLevel + "\n"

+ "CO2 Level is: " + cO2Level;

}

}

Java File Name: FireMonitorGUI

//Class containing the gui components of the monitor

**public** **class** FireMonitorGUI **extends** JFrame {

**private** JButton btnUpdate, btnGetReading;

**private** JLabel labelMon, labelSen, labelImg, labelmonitorCount, labelsensorCount,

labelSensor;

**private** JPanel readingPanel;

**private** JScrollPane scrollPanel;

**private** **int** floorNo, roomNo;

**private** **static** FireMonitorController *fireMonitor*;

**public** FireMonitorGUI(FireMonitorController fireMonitor) {

initComponents();

**this**.*fireMonitor* = fireMonitor;

**this**.setLocationRelativeTo(**null**);

}

**public** **void** setTheMonitorCount(**int** monitorCount){

**this**.labelmonitorCount.setText(Integer.*toString*(monitorCount));

}

**public** **void** setTheSensorCount(**int** sensorCount){

**this**.labelsensorCount.setText(Integer.*toString*(sensorCount));

}

**public** **void** addNewMonitor(SensorMonitorGUI sensorMonitor){

readingPanel.add(sensorMonitor);

sensorMonitor.setVisible(**true**);

}

**private** **void** initComponents() {

labelMon = **new** JLabel();

labelSensor = **new** JLabel();

scrollPanel = **new** JScrollPane();

readingPanel = **new** JPanel();

labelImg = **new** JLabel();

btnUpdate = **new** JButton();

labelSen = **new** JLabel();

labelmonitorCount = **new** JLabel();

labelsensorCount = **new** JLabel();

btnGetReading = **new** JButton();

setMaximumSize(**new** Dimension(1100, 600));

setMinimumSize(**new** Dimension(1100, 600));

setPreferredSize(**new** Dimension(1100, 600));

setDefaultCloseOperation(WindowConstants.***EXIT\_ON\_CLOSE***);

setResizable(**false**);

setSize(**new** Dimension(1100, 600));

getContentPane().setLayout(**null**);

setTitle("Fire Alarm System Control Panel");

readingPanel.setLayout(**new** GridLayout(0, 14, 100, 150));

readingPanel.setBackground(**new** Color(19, 200, 255));

scrollPanel.setViewportView(readingPanel);

getContentPane().add(scrollPanel);

scrollPanel.setBounds(20, 110, 1020, 380);

scrollPanel.setBackground(**new** Color(59, 89, 182));

labelMon.setFont(**new** java.awt.Font("Arial", 0, 18));

labelMon.setForeground(**new** java.awt.Color(255, 255, 255));

labelMon.setText("Monitor Count");

getContentPane().add(labelMon);

labelMon.setBounds(20, 30, 170, 21);

labelSen.setFont(**new** java.awt.Font("Arial", 0, 18));

labelSen.setForeground(**new** java.awt.Color(255, 255, 255));

labelSen.setText("Sensor Count");

getContentPane().add(labelSen);

labelSen.setBounds(250, 30, 190, 21);

labelSensor.setFont(**new** java.awt.Font("Arial", 0, 18));

labelSensor.setForeground(**new** java.awt.Color(255, 255, 255));

labelSensor.setText("Sensor Controllers");

getContentPane().add(labelSensor);

labelSensor.setBounds(806, 30, 190, 21);

labelmonitorCount.setFont(**new** java.awt.Font("Arial", 0, 30));

labelmonitorCount.setForeground(**new** java.awt.Color(255, 255, 255));

labelmonitorCount.setText("0");

getContentPane().add(labelmonitorCount);

labelmonitorCount.setBounds(70, 30, 50, 75);

btnUpdate.setText("Latest Reading");

btnUpdate.setBackground(**new** Color(189, 28, 230));

btnUpdate.addActionListener(**new** java.awt.event.ActionListener() {

**public** **void** actionPerformed(java.awt.event.ActionEvent evt) {

btnUpdateActionPerformed(evt);

}

});

labelsensorCount.setFont(**new** java.awt.Font("Arial", 0, 30));

labelsensorCount.setForeground(**new** java.awt.Color(255, 255, 255));

labelsensorCount.setText("4");

getContentPane().add(labelsensorCount);

labelsensorCount.setBounds(300, 55, 20, 21);

getContentPane().add(btnUpdate);

btnUpdate.setBounds(700, 55, 164, 29);

btnGetReading.setBackground(**new** Color(189, 28, 230));

btnGetReading.setText("Get New Monitors");

btnGetReading.addActionListener(**new** java.awt.event.ActionListener() {

**public** **void** actionPerformed(java.awt.event.ActionEvent evt) {

btnGetReadingActionPerformed(evt);

}

});

getContentPane().add(btnGetReading);

btnGetReading.setBounds(880, 55, 164, 29);

labelImg.setIcon(**new** javax.swing.ImageIcon(getClass().getResource("/com/sliit/images/fire.jpg")));

getContentPane().add(labelImg);

labelImg.setBounds(0, 0, 1100, 590);

pack();

}

**private** **void** btnUpdateActionPerformed(java.awt.event.ActionEvent evt) {

*fireMonitor*.updateCurrentReadings();

}

**private** **void** btnGetReadingActionPerformed(java.awt.event.ActionEvent evt) {

*fireMonitor*.getNewestReadingsByFloorNo(floorNo, roomNo);

}

**public** **static** **void** main(String args[]) **throws** ClassNotFoundException, IllegalAccessException, UnsupportedLookAndFeelException {

**try** {

**for** (UIManager.LookAndFeelInfo data : UIManager.*getInstalledLookAndFeels*()) {

**if** ("Nimbus".equals(data.getName())) {

UIManager.*setLookAndFeel*(data.getClassName());

**break**;

}

}

} **catch** (Exception ex) {

System.***err***.println(ex.getMessage());

}

EventQueue.*invokeLater*(**new** Runnable() {

**public** **void** run() {

**new** FireMonitorGUI(*fireMonitor*).setVisible(**true**);

}

});

}}

Java File Name: UserLoginMonitorGUI

//Class holding GUI of login components

**public** **class** UserLoginMonitorGUI **extends** JFrame {

**private** JButton buttontLogin;

**private** JLabel userLabel, passLabel;

**private** JPasswordField password;

**private** JTextField username;

**private** MonitorApplication monApp;

**private** JPanel panel;

**public** UserLoginMonitorGUI() {

initComponents();

// Relative to center

**this**.setLocationRelativeTo(**null**);

//initialize the application

monApp = **new** MonitorApplication();

}

@SuppressWarnings("unchecked")

**private** **void** initComponents() {

userLabel = **new** JLabel();

userLabel.setText("User Name :");

username = **new** JTextField();

passLabel = **new** JLabel();

passLabel.setText("Password :");

password = **new** JPasswordField();

buttontLogin = **new** JButton();

panel = **new** JPanel(**new** GridLayout(3, 1));

panel.add(userLabel);

panel.add(username);

panel.add(passLabel);

panel.add(password);

panel.add(buttontLogin);

setDefaultCloseOperation(JFrame.***EXIT\_ON\_CLOSE***);

add(panel, BorderLayout.***CENTER***);

setTitle("Login To Fire Monitor System");

setSize(400, 120);

setVisible(**true**);

buttontLogin.setText("Login");

buttontLogin.addActionListener(**new** java.awt.event.ActionListener() {

**public** **void** actionPerformed(ActionEvent evt) {

**try** {

btnLoginActionPerformed(evt);

} **catch** (IOException | HeadlessException e) {

System.***out***.println(e.getMessage());

}

}

});

}

**private** **void** btnLoginActionPerformed(java.awt.event.ActionEvent evt) **throws** HeadlessException, IOException {

**if**(LoginController.*validateUser*(username.getText(),password.~~getText~~())){

monApp.intializeTheMonitor();

**this**.dispose();

}

**else**{

JOptionPane.*showMessageDialog*(

**null**,

"Authentication Failed! Please check your username and password",

"Login Failed",

JOptionPane.***ERROR\_MESSAGE***);

}

}

**public** **static** **void** main(String args[]) {

**try** {

**for** (javax.swing.UIManager.LookAndFeelInfo info : javax.swing.UIManager.*getInstalledLookAndFeels*()) {

**if** ("Windows".equals(info.getName())) {

javax.swing.UIManager.*setLookAndFeel*(info.getClassName());

**break**;

}

}

} **catch** (ClassNotFoundException ex) {

} **catch** (InstantiationException ex) {

} **catch** (IllegalAccessException ex) {

} **catch** (javax.swing.UnsupportedLookAndFeelException ex) {

}

java.awt.EventQueue.*invokeLater*(**new** Runnable() {

**public** **void** run() {

**new** UserLoginMonitorGUI().setVisible(**true**);

}

});

}}

Java File Name: CO2Level

//Class to have the reading of the CO2 Level

**public** **class** CO2Level {

**private** **double** cO2lvel;

**public** CO2Level() {

cO2lvel = 1;

}

**public** **double** generateNewReading(){

Random random = **new** Random();

**int** val = random.nextInt(10);

**if** (val < 0) {

**return** (cO2lvel = cO2lvel + 1);

}

**return** (cO2lvel+val);

}

}

Java File Name: SmokeLevel

//Class to have the reading of the smoke Level

**public** **class** SmokeLevel {

**private** **int** smkeLvel;

**public** SmokeLevel() {

smkeLvel = 1;

}

**public** **double** generateNewReading(){

Random random = **new** Random();

**int** val = random.nextInt(10);

**if** (val < 0) {

**return** (smkeLvel = smkeLvel + 1);

}

**return** (smkeLvel+val);

}

}

Java File Name: Sensor

**public** **class** Sensor **implements** Serializable{

**private** **int** floorNo;

**private** **int** roomNo;

**private** SensorObject newReading;

**public** Sensor() {}

**public** **int** getFloorNo() {

**return** floorNo;

}

**public** **void** setFloorNo(**int** floorNo) {

**this**.floorNo = floorNo;

}

**public** **int** getRoomNo() {

**return** roomNo;

}

**public** **void** setRoomNo(**int** roomNo) {

**this**.roomNo = roomNo;

}

**public** SensorObject getNewReading() {

**return** newReading;

}

**public** **void** setLatestReading(SensorObject newReading) {

**this**.newReading = newReading;

}

@Override

**public** String toString() {

**return** "Sensor Reading\n"

+ "Floor No is: " + floorNo + "Room No is: " + roomNo +

"\nCurrent New Reading is: " + newReading;

}

}

Java File Name: SensorListener

// This is the class which contains the interface to notify all monitors with

// the call back methods

**public** **interface** SensorListener **extends** Remote{

**public** **void** ModifyTheReading(List<Sensor> sensors, **int** moCount, **int** sCount) **throws** RemoteException;

**public** **void** notifyAlert(Sensor sensor) **throws** RemoteException;

**public** **void** makeAlertsOnFailures(Sensor sensor) **throws** RemoteException;

}

Java File Name: FireAlarmSensor

//Class where after validation is used to add the new sensors to the monitor

**public** **class** FireAlarmSensor {

**private** **static** FireAlarmSensor *fSensor*;

**private** SmokeLevel smkeLvel;

**private** CO2Level co2Lvel;

**private** SensorObject readin;

**private** BufferedReader br;

**private** PrintWriter pW;

**private** Socket socket;

**private** String uname, pwd, userIteractInput;

**private** Scanner userInput;

**private** **int** floorNo, roomNo;

**private** **long** breakTheReading;

**public** FireAlarmSensor() {

userInput = **new** Scanner(System.***in***);

co2Lvel = **new** CO2Level();

readin = **new** SensorObject();

smkeLvel = **new** SmokeLevel();

}

**public** **void** run() **throws** IOException, InterruptedException {

socket = **new** Socket("127.0.0.1", 9001);

br = **new** BufferedReader(**new** InputStreamReader(

socket.getInputStream()));

pW = **new** PrintWriter(socket.getOutputStream(), **true**);

**while** (**true**) {

userIteractInput = br.readLine();

**if** (userIteractInput.startsWith("GETNAME") && userIteractInput != **null** ) {

pW.println(getUsername());

} **else** **if** (userIteractInput.startsWith("GETPASSWORD") && userIteractInput != **null**){

pW.println(getPassword());

} **else** **if** (userIteractInput.startsWith("GETFLOOR") && userIteractInput != **null**){

pW.println(getFloor());

}

**else** **if**(userIteractInput.startsWith("GETROOM") && userIteractInput != **null**) {

pW.println(getRoom());

}**else** **if** (userIteractInput.startsWith("REGISTEREDSENSOR")&& userIteractInput != **null**){

generateSensorReading();

}

}

}

**private** String getUsername() {

**try** {

System.***out***.print("Enter your sensor username : ");

uname = userInput.nextLine();

**return** uname;

} **catch** (Exception e) {

**return** "Invalid Username ";

}

}

**private** String getPassword(){

**try** {

System.***out***.print("Enter your sensor password: ");

pwd = userInput.nextLine();

**return** pwd;

} **catch** (Exception e) {

**return** "Invalid Username ";

}

}

**private** **int** getFloor() {

System.***out***.print("Enter Floor number between(1-10): ");

floorNo = Integer.*parseInt*(userInput.nextLine());

**if**(floorNo < 1 || floorNo > 10) {

System.***out***.println("Please enter a valid floor number between 1 and 10");

System.***out***.print("Please Enter Floor Number: ");

floorNo = Integer.*parseInt*(userInput.nextLine());

}

**return** floorNo;

}

**private** **int** getRoom() {

System.***out***.print("Enter Room number between(1-20): ");

roomNo = Integer.*parseInt*(userInput.nextLine());

**if**(roomNo < 1 || roomNo > 20) {

System.***out***.println("Please enter a valid room number between 1 and 15");

System.***out***.print("Please Enter Room Number: ");

roomNo = Integer.*parseInt*(userInput.nextLine());

}

**return** roomNo;

}

**private** **void** generateSensorReading() **throws** InterruptedException, IOException {

**for** (; ;) {

readin.setSmokeLevel((**int**) smkeLvel.generateNewReading());

readin.setCO2Level(co2Lvel.generateNewReading());

System.***out***.println(readin);

ByteArrayOutputStream os = **new** ByteArrayOutputStream();

**try** (ObjectOutputStream obj = **new** ObjectOutputStream( os )) {

obj.writeObject( readin );

pW.println(Base64.*getEncoder*().encodeToString(os.toByteArray()));

} **catch** (IOException exe) {

System.***err***.println(exe.getMessage());

}

breakTheReading = 30000;

Thread.*sleep*(breakTheReading);

}

}

**public** **static** **void** main(String[] args) **throws** Exception {

*fSensor* = **new** FireAlarmSensor();

*fSensor*.run();

}

}

RESTful Web Service & Web Client(React JS)

App.js

import React, { Component } from 'react'

import axios from 'axios'

import {Table} from 'reactstrap'

class App extends Component {

  state = {

      sensors: [],

  }

  componentDidMount() {

    axios.get("http://localhost:3000/sensors")

      .then(response => {

        this.setState({

          sensors: response.data

        })

      })

      this.interval = setInterval(() => this.setState({ time: Date.now() }), 40000);

  }

  componentWillUnmount() {

    clearInterval(this.interval);

  }

  render() {

    let sensors = this.state.sensors.map(sensor => {

    let smokeNumber = Math.floor((Math.random() \* 10) + 1)

    let co2Number = Math.floor((Math.random() \* 10) + 1)

      return (

        <tr key={sensor.id}>

          <td>{sensor.id}</td>

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

          <td>{sensor.floorNo}</td>

          <td>{sensor.roomNo}</td>

          <td className={smokeNumber > 5 ? 'bg-danger' : ''}> {smokeNumber}  </td>

          <td className={co2Number > 5 ? 'bg-danger': ''}> {co2Number}  </td>

        </tr>

      )

    })

    return (

      <div className="App container">

      <div className="text-center">

      <h2 className="mt-4">Fire Alarm Monitoring System</h2>

      </div>

        <Table className="mt-4 center-block">

          <thead>

            <tr>

              <th>#</th>

              <th>Sensor Status</th>

              <th>Floor Number</th>

              <th>Room number</th>

              <th>Smoke Level</th>

              <th>CO2 Level</th>

            </tr>

          </thead>

        <tbody>{sensors}</tbody>

        </Table>

      </div>

    )

  }

}

export default App