**Distributed Systems**

**Assignment 2**

**REST API**

**Group Members**

|  |  |
| --- | --- |
| **Student ID** | **Student Name** |
| IT18146752 | Lakshan P.K.A. |
| IT18149654 | Rajapaksha T.N. |
| IT18142938 | Perera H.H.N. |
| IT18148046 | Pathirana K.P.A.K. |

**High Level Diagram of the System**

SMS Service

**Web Client**

**Sensor App**

**REST API**

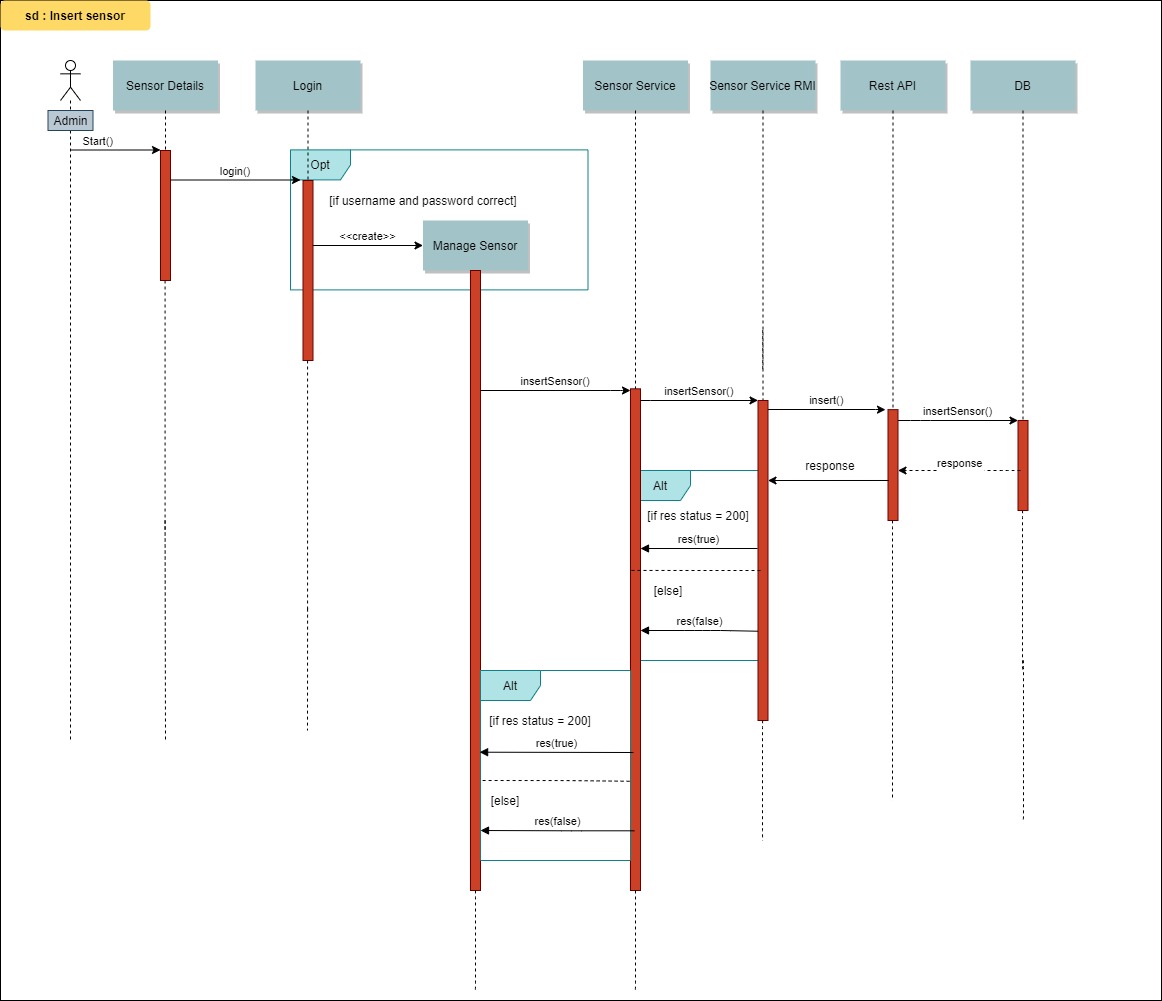
**Desktop App**

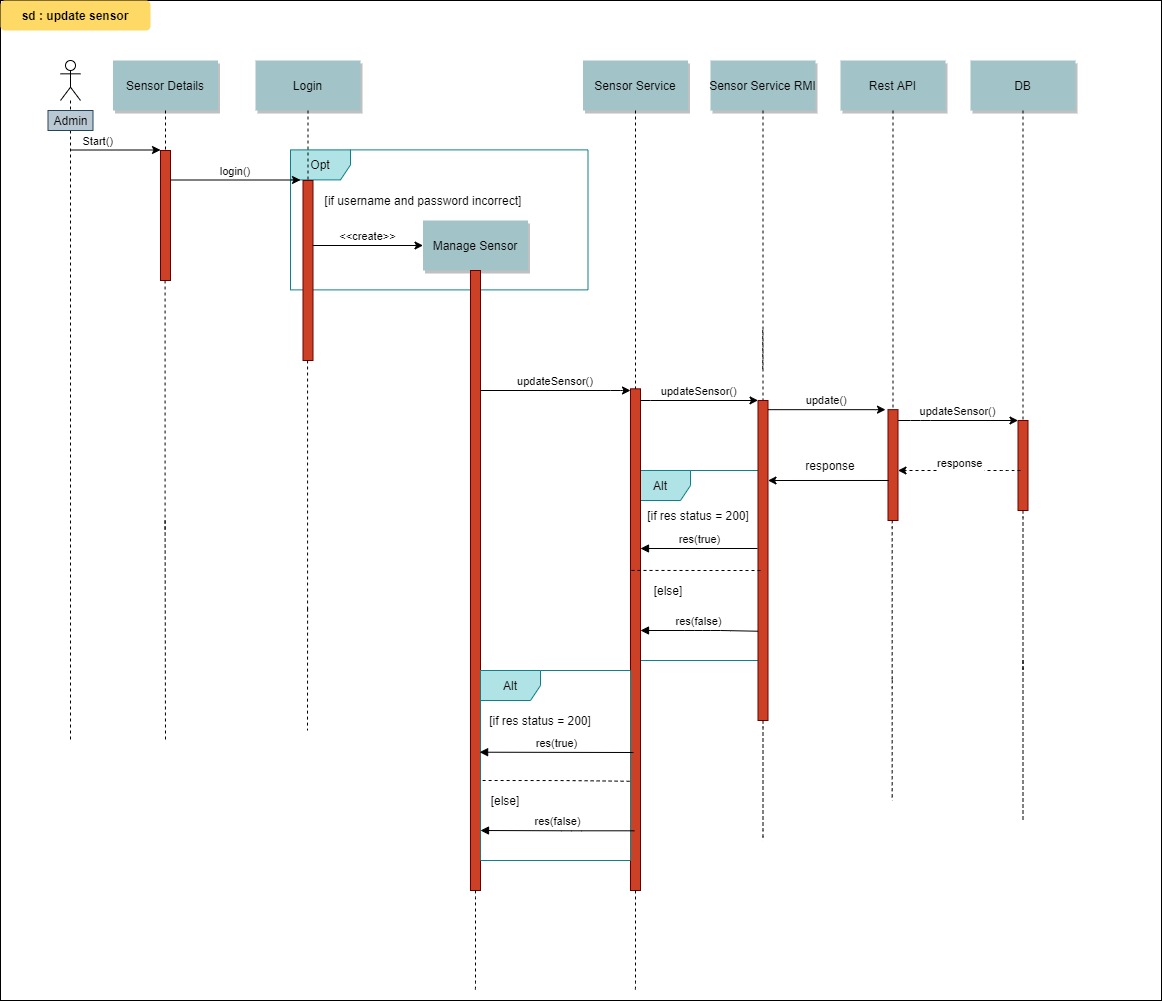
Email Service

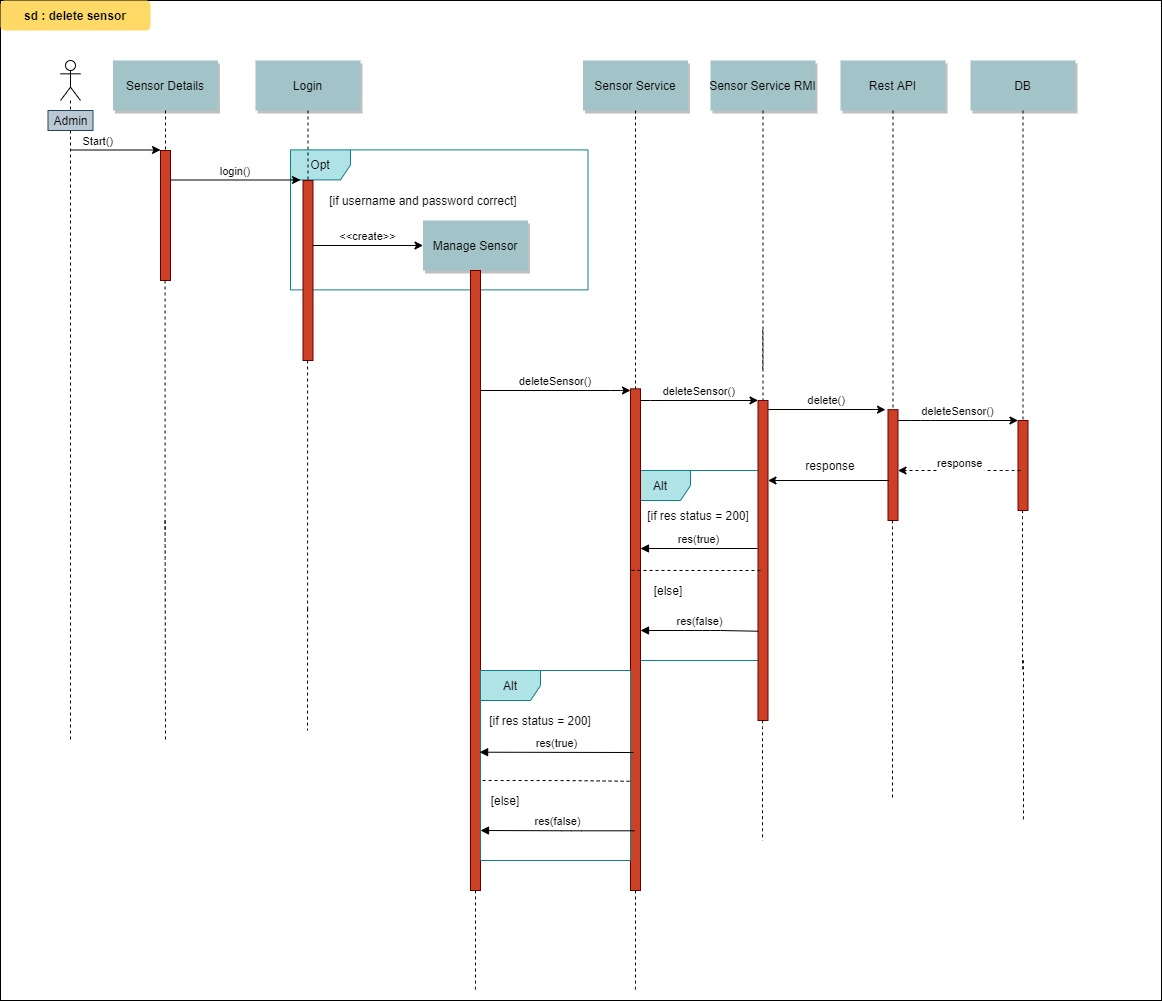
**MongoDB**

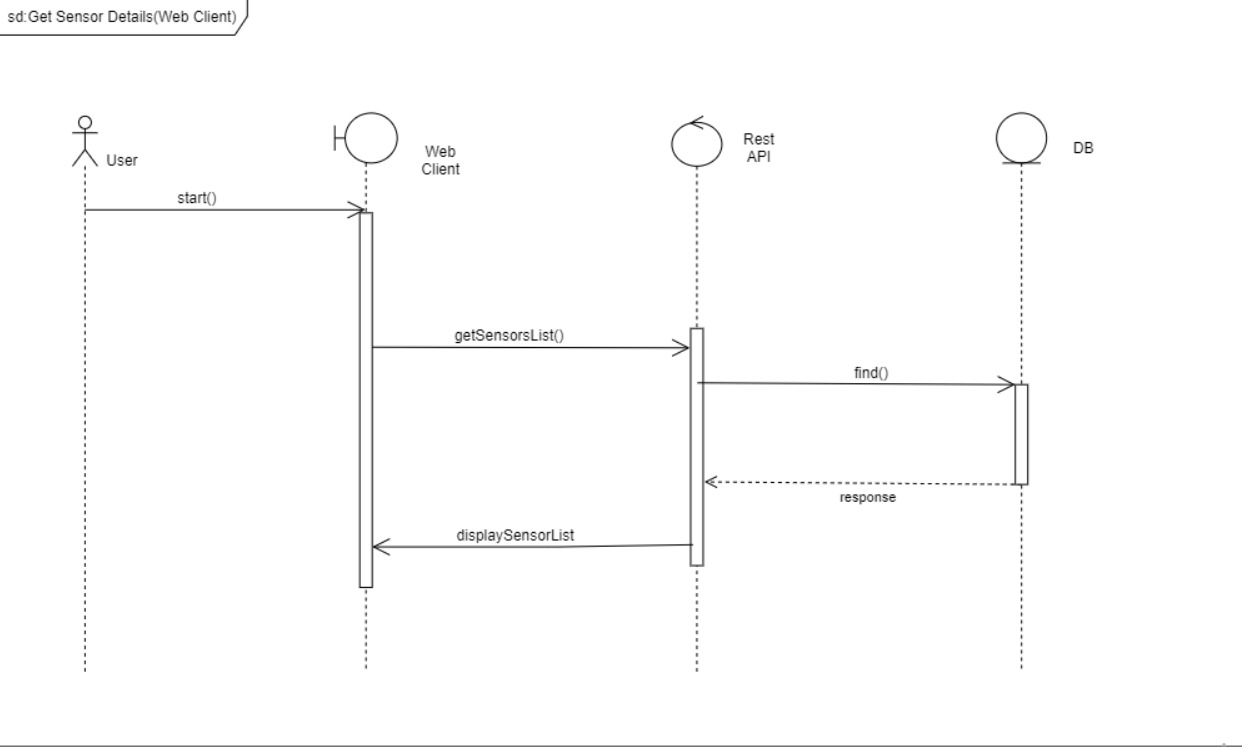
**RMI Server**

**Service Interfaces Exposed by Services and the Workflows used in the System**

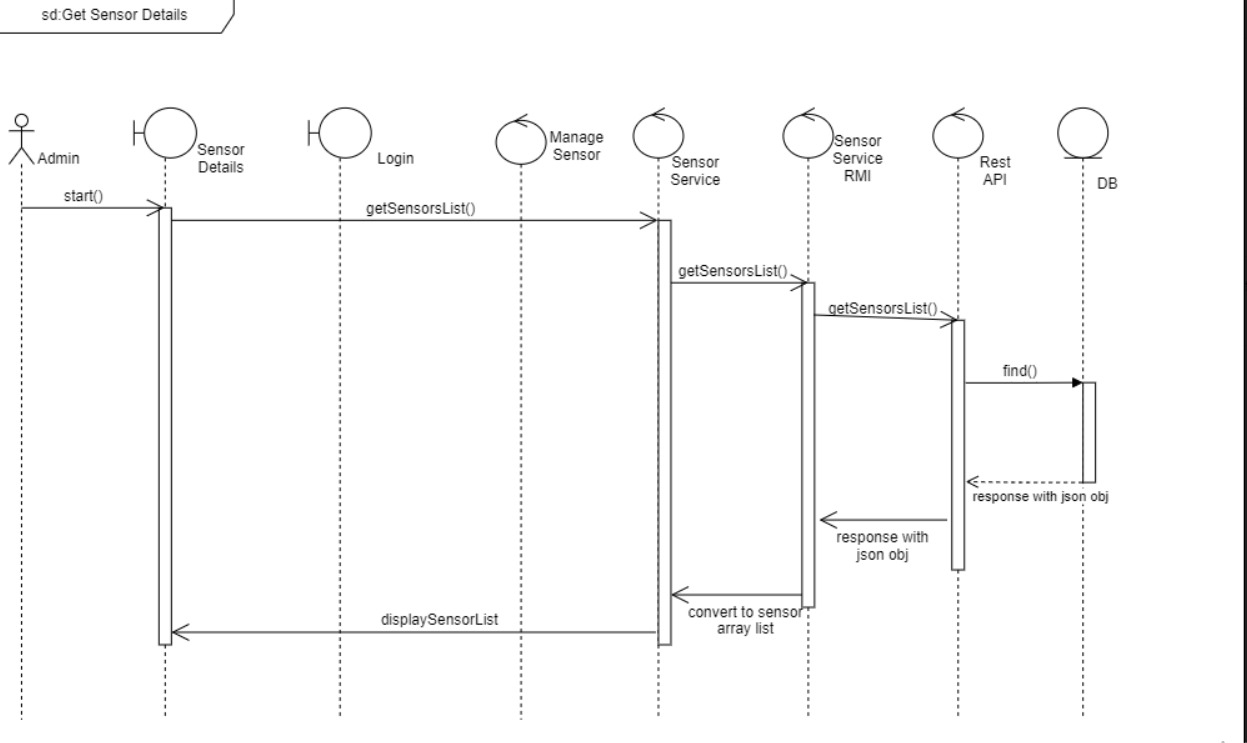
**Sequence diagram for inserting a new sensor**

**Sequence diagram for updating the details of an existing sensor**

**Sequence diagram for deleting a sensor**

**Sequence diagram for getting all the sensor details to the web client**

**Sequence diagram for getting all the sensor details to the desktop client**



**Service Interfaces exposed by each service**

**Sensor Service**

**boolean** addSensor(com.rmi.Sensor sensor) **throws** RemoteException, IOException;

**void** updateSensor(String sensorId, com.rmi.Sensor sensor) **throws** RemoteException, IOException;

**void** removeSensor(String sensorId) **throws** RemoteException, IOException;

com.rmi.Sensor getSensor(String sensorId) **throws** RemoteException, IOException;

ArrayList<com.rmi.Sensor> getSensorsList() **throws** RemoteException, IOException;

**Sensor Service RMI**

**boolean** addSensor(Sensor sensor) **throws** RemoteException, IOException;

**void** updateSensor(String sensorId, Sensor sensor) **throws** RemoteException, IOException;

**void** removeSensor(String sensorId) **throws** RemoteException, IOException;

Sensor getSensor(String sensorId) **throws** RemoteException, IOException;

**int** increment() **throws** RemoteException;

**Workflows**

**Insert Sensor Workflow**

Admin must login to the system giving valid login details and then the system navigates to ManageSensorUI. Then the admin can insert a sensor by entering sensor details. After “Add” button is clicked, addSensor(Sensor sensor) method in the SensorService class is called. Then the addSensor(Sensor sensor) method in the SensorServerRMI class is called by the addSensor() method in the SensorService class. Within the addSensor() method in the SensorServerRMI class, the sensor object is converted into a JSON object and then it will pass to the REST API using POST request. Then the new sensor details will be inserted to the MongoDB. If the data insertion is successful, API will send the response code as 200 to the addSensor() method in the SensorServerRMI. According to the response, a boolean value will be returned to the addSensor() method in the SensorService class. Finally, a success or error message will be shown in the ManageSensorUI based on the boolean return value.

**Update Sensor Workflow**

Admin must login to the system giving valid login details and then the system navigates to ManageSensorUI. Then the admin can update an existing sensor by selecting a sensor in the list and doing changes to sensor details. After “Update” button is clicked, updateSensor(Sensor sensor) method in the SensorService class is called. Then the updateSensor(Sensor sensor) method in the SensorServerRMI class is called by the updateSensor() method in the SensorService class. Within the updateSensor() method in the SensorServerRMI class, the sensor object is converted into a JSON object and then it will pass to the REST API using PUT request. Then the sensor details will be updated in the MongoDB. If the data modification is successful, API will send the response code as 200 to the updateSensor() method in the SensorServerRMI. According to the response, a boolean value will be returned to the updateSensor() method in the SensorService class. Finally, a success or error message will be shown in the ManageSensorUI based on the boolean return value.

**Delete Sensor Workflow**

Admin must login to the system giving valid login details and then the system navigates to ManageSensorUI. Then the admin can delete a sensor by selecting a sensor in the list. After “Remove” button is clicked, removeSensor(Sensor sensor) method in the SensorService class is called. Then the removeSensor (Sensor sensor) method in the SensorServerRMI class is called by the removeSensor() method in the SensorService class. Within the removeSensor() method in the SensorServerRMI class, the sensor will be attached as a parameter to the URL and passed to the REST API using DELETE request. Then the sensor will be deleted from the MongoDB. If the data deletion is successful, API will send the response code as 200 to the removeSensor() method in the SensorServerRMI. According to the response, a boolean value will be returned to the removeSensor() method in the SensorService class. Finally, a success or error message will be shown in the ManageSensorUI based on the boolean return value.

**Get Sensor List Workflow**

In SensorDetailsUI, after refreshing in every 30 seconds and in ManageSensorUI, after performing CRUD operations, the getSensorsList() method in the SensorService class is called. Then getSensorsList() method in the SensorServerRMI class is called by the getSensorsList() method in the SensorService class. Within the getSensorsList() method in the SensorServerRMI class, after sending a GET request to the REST API, a JSON array with all the sensor details will be passed to the getSensorsList() method in SensorServerRMI class as a response and then we convert JSON objects in the JSON array one by one to sensor objects using Java Sensor model and we add them to an ArrayList of Sensor type. This getSensorsList() method in SensorServerRMI class will be called every 15 seconds. The ArrayList will be returned to the getSensorsList() method in the SensorService class. That getSensorsList() method in the SensorService class will be called every 30 seconds by the SensorDetailsUI and refresh the sensor details in the JTable.

In the web client, a GET request will be sent to the REST API and then a JSON array with all the sensor objects will be sent to the web client as the response and then we map the JSON objects to a table. This process will happen every 40 seconds.

**Sensor Simulators Workflow**

Each sensor simulator will generate two random integers from 1 to 10 for CO2 level and smoke level every 10 seconds. Then it will pass to the REST API using PUT request. Then the sensor details will be updated in the MongoDB.

**The Authentication and Security Mechanisms Adopted in the System**

Only a verified administrator can access the system where he or she can add new sensors to the system, edit the details such as sensor id, sensor name, active or inactive status, floor number and room number of already added sensors in the system and delete any existing sensor. In order to perform all these operations in the ManageSensorUI window, the administrator must first login to the system providing the valid username and password. After the administrator enter the username and password in the LoginUI window, those credentials will be checked and validated. Only if the admin account is verified by the system, it will allow the administrator to navigate to the admin dashboard window of the system which is the ManageSensorUI window. If the entered login credentials are invalid or incorrect (does not match with the valid admin credentials), the system will display an error message and prompt to enter the login details again.

Once administrator logout from the system from ManageSensorUI window, the system will navigate to the SensorDetailsUI window. If the administrator wants to navigate back to the admin dashboard (ManageSensorUI window) now, the administrator must follow the login process once again properly in order to navigate there. There is no way to navigate to admin dashboard window directly without the following through the login process first.

All the input fields in this desktop application are properly validated. Only a valid set of characters are allowed to be entered by the user. Otherwise, the system will give an error message from the client side and will not allow to continue the process. Therefore, if the administrator added any invalid data types or characters that are not allowed in specific fields when he was editing sensor details or adding new sensors, the system will throw an error massage and the administrator should re-enter proper and valid data to the fields.

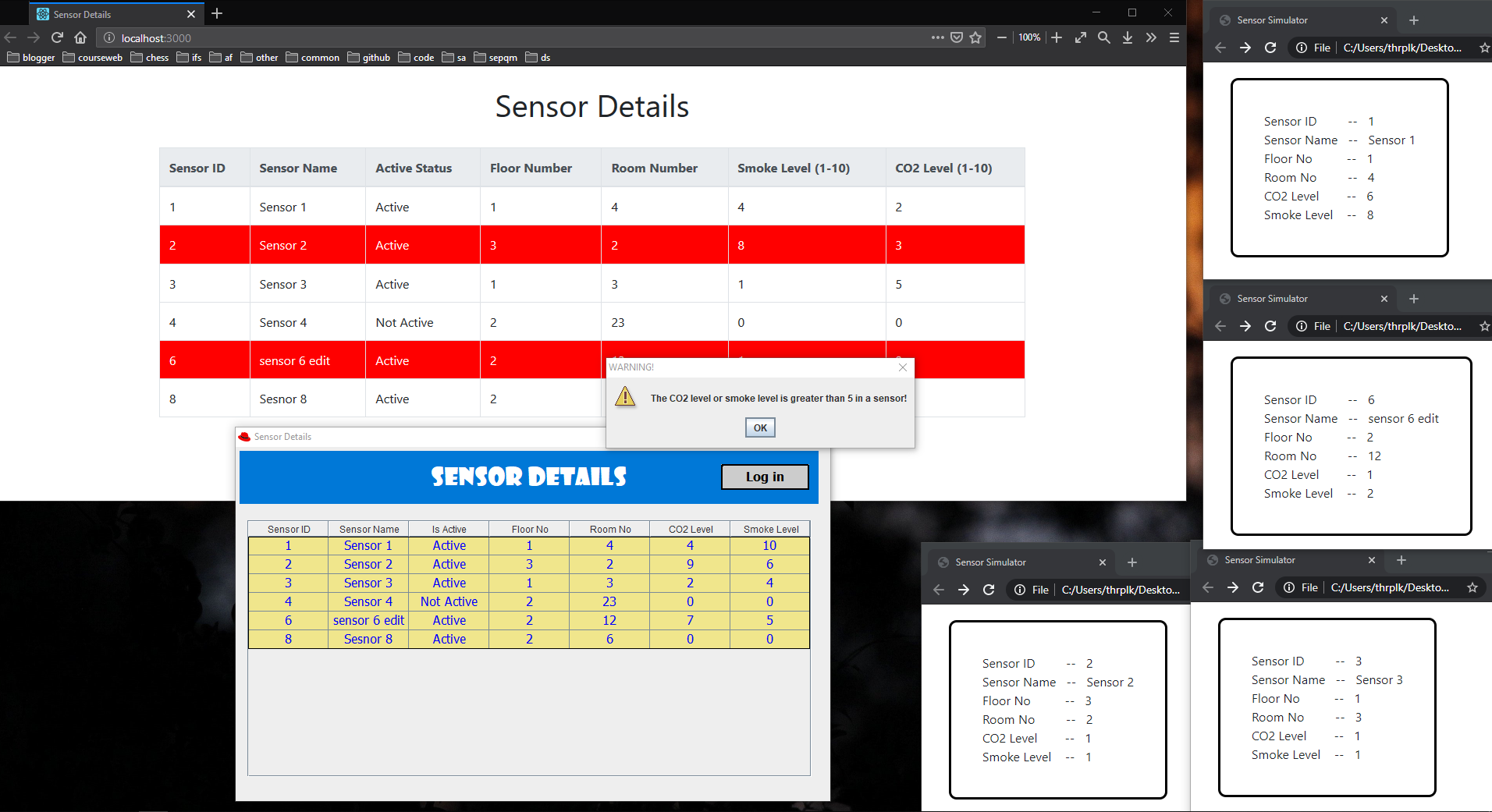
MongoDB database also provides some specific key security features such as,

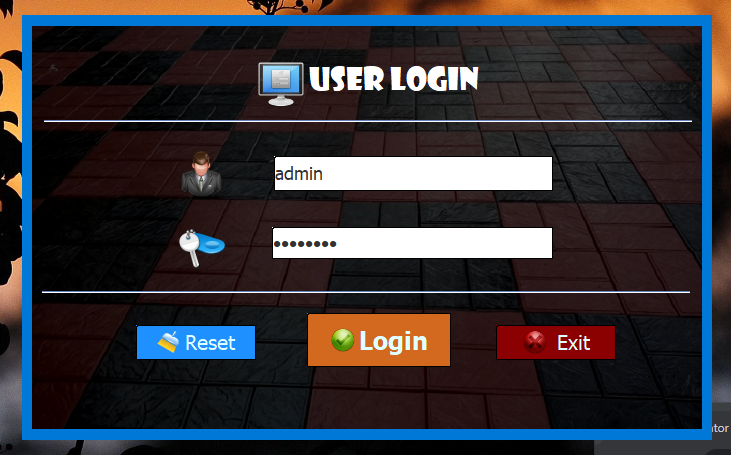
* Role based access control
* Enable access control
* TLS / SSL
* Encryption
* Authentication

These key features help the proper data transmission in the system and upgrade the security level of the system.

**Screenshots of the System Interfaces**

**User interfaces of web client, desktop client and sensor simulators for showing sensor details**

****

**Login user interface**

**Manage sensor details user interface**

**Appendix**

**SensorDetailsUI.java**

**package** com.interfaces;

**import** java.awt.BorderLayout;

**import** java.awt.Color;

**import** java.awt.Dimension;

**import** java.awt.EventQueue;

**import** java.awt.Font;

**import** java.awt.SystemColor;

**import** java.awt.Toolkit;

**import** java.awt.event.ActionEvent;

**import** java.awt.event.ActionListener;

**import** java.io.IOException;

**import** java.net.ConnectException;

**import** java.util.ArrayList;

**import** javax.swing.JButton;

**import** javax.swing.JFrame;

**import** javax.swing.JLabel;

**import** javax.swing.JOptionPane;

**import** javax.swing.JPanel;

**import** javax.swing.JScrollPane;

**import** javax.swing.JTable;

**import** javax.swing.ListSelectionModel;

**import** javax.swing.SwingConstants;

**import** javax.swing.Timer;

**import** javax.swing.UIManager;

**import** javax.swing.border.EmptyBorder;

**import** javax.swing.border.LineBorder;

**import** javax.swing.table.DefaultTableCellRenderer;

**import** javax.swing.table.DefaultTableModel;

**import** org.json.JSONException;

**import** com.rmi.SensorClientRMI;

**import** com.services.ISensorService;

**import** com.services.SensorService;

// sensor details user interface jframe

**public** **class** SensorDetailsUI **extends** JFrame {

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

**private** JPanel contentPane;

**private** JScrollPane scrollPane;

**public** **static** **int** *status* = 0;

**public** **static** **int** *statusExit* = 0;

**private** Timer timer;

**private** **final** **static** **int** ***INTERVAL*** = 30000;

// defining properties of all the design elements of the display sensor details

// user interface

**public** SensorDetailsUI() **throws** IOException {

setTitle("Sensor Details");

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

setBounds(100, 100, 768, 483);

setResizable(**false**);

*statusExit* = 0;

contentPane = **new** JPanel();

contentPane.setBorder(**new** EmptyBorder(5, 5, 5, 5));

contentPane.setLayout(**new** BorderLayout(0, 0));

setContentPane(contentPane);

JPanel panel\_1 = **new** JPanel();

panel\_1.setBounds(0, 73, 740, 343);

panel\_1.setLayout(**null**);

getContentPane().add(panel\_1);

scrollPane = **new** JScrollPane();

scrollPane.setBounds(10, 89, 723, 329);

scrollPane.setBackground(**new** Color(255, 255, 255));

scrollPane.setFont(**new** Font("Tahoma", Font.***BOLD***, 25));

panel\_1.add(scrollPane);

JPanel panel = **new** JPanel();

panel.setForeground(UIManager.*getColor*("CheckBoxMenuItem.selectionForeground"));

panel.setBackground(SystemColor.***textHighlight***);

panel.setBounds(0, 0, 742, 68);

panel.setLayout(**null**);

panel\_1.add(panel);

// login button

JButton btnLogin = **new** JButton("Log in");

btnLogin.setBounds(617, 17, 113, 33);

btnLogin.setBorder(**new** LineBorder(**new** Color(0, 0, 0), 2, **true**));

btnLogin.setForeground(**new** Color(0, 0, 0));

btnLogin.setBackground(**new** Color(204, 204, 204));

btnLogin.setFont(**new** Font("Tahoma", Font.***BOLD***, 16));

btnLogin.setFocusable(**false**);

btnLogin.addActionListener(**new** ActionListener() {

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

**try** {

**if** (JOptionPane.*showConfirmDialog*(**null**,

"Confirm if you really want to navigate to the login window.",

"Login window navigation confirmation",

JOptionPane.***YES\_NO\_OPTION***) == JOptionPane.***YES\_NO\_OPTION***) {

LoginUI loginUI = **new** LoginUI();

loginUI.displayFrame();

disposeFrame();

*statusExit* = 1;

timer.stop();

}

} **catch** (Exception e1) {

e1.printStackTrace();

}

}

});

panel.add(btnLogin);

JLabel lblTopic = **new** JLabel("Sensor Details");

lblTopic.setBounds(0, 11, 740, 57);

lblTopic.setBackground(UIManager.*getColor*("menu"));

lblTopic.setForeground(SystemColor.***textHighlightText***);

lblTopic.setHorizontalAlignment(SwingConstants.***CENTER***);

lblTopic.setFont(**new** Font("Showcard Gothic", Font.***BOLD***, 30));

panel.add(lblTopic);

displayTable();

// refresh the table every 30 seconds

timer = **new** Timer(***INTERVAL***, **new** ActionListener() {

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

**try** {

displayTable();

**if** (*status* == 1 && *statusExit* == 0) {

JOptionPane.*showMessageDialog*(**null**,

"The CO2 level or smoke level is greater than 5 in a sensor!", "WARNING!",

JOptionPane.***WARNING\_MESSAGE***);

}

**if** (*statusExit* == 1) {

timer.stop();

}

} **catch** (IOException e1) {

e1.printStackTrace();

} **catch** (Exception e1) {

e1.printStackTrace();

}

}

});

timer.start();

}

// jtable design

**public** **void** displayTable() **throws** IOException {

String col[] = { "Sensor ID", "Sensor Name", "Is Active", "Floor No", "Room No", "CO2 Level", "Smoke Level" };

DefaultTableModel tableModel = **new** DefaultTableModel(col, 0);

JTable table = **new** JTable(tableModel);

table.setBorder(**new** LineBorder(**new** Color(0, 0, 0)));

table.setRowSelectionAllowed(**false**);

table.setForeground(Color.***blue***);

table.setRowHeight(24);

table.setFont(**new** Font("Arial", Font.***BOLD***, 12));

table.setSelectionMode(ListSelectionModel.***SINGLE\_SELECTION***);

table.setEnabled(**false**);

table.setSurrendersFocusOnKeystroke(**true**);

table.setFont(**new** Font("Tahoma", Font.***PLAIN***, 16));

table.setBackground(**new** Color(240, 230, 140));

DefaultTableCellRenderer centerRenderer = **new** DefaultTableCellRenderer();

centerRenderer.setHorizontalAlignment(JLabel.***CENTER***);

table.getColumnModel().getColumn(0).setCellRenderer(centerRenderer);

table.getColumnModel().getColumn(1).setCellRenderer(centerRenderer);

table.getColumnModel().getColumn(2).setCellRenderer(centerRenderer);

table.getColumnModel().getColumn(3).setCellRenderer(centerRenderer);

table.getColumnModel().getColumn(4).setCellRenderer(centerRenderer);

table.getColumnModel().getColumn(5).setCellRenderer(centerRenderer);

table.getColumnModel().getColumn(6).setCellRenderer(centerRenderer);

*status* = 0;

// retrieving sensor details from the arraylist and display in the jtable

ArrayList<com.rmi.Sensor> sensorsList = **new** ArrayList<com.rmi.Sensor>();

sensorsList = refreshTable();

**for** (com.rmi.Sensor sensor1 : sensorsList) {

String sensorId = sensor1.getSensorId();

String sensorName = sensor1.getSensorName();

**int** roomNo = sensor1.getRoomNo();

**int** floorNo = sensor1.getFloorNo();

**int** CO2Level = sensor1.getCO2Level();

**int** smokeLevel = sensor1.getSmokeLevel();

String isActive;

**if** (sensor1.isActive()) {

isActive = "Active";

} **else** {

isActive = "Not Active";

}

Object[] objs = { sensorId, sensorName, isActive, floorNo, roomNo, CO2Level, smokeLevel };

tableModel.addRow(objs);

// checking the co2 level and smoke level of sonsors to show the warning message

**if** (CO2Level > 5 || smokeLevel > 5) {

*status* = 1;

}

}

scrollPane.setViewportView(table);

}

// getting the arraylist of all the sensors from sensor service

**public** ArrayList<com.rmi.Sensor> refreshTable() **throws** IOException {

ISensorService iSensorService = (ISensorService) **new** SensorService();

ArrayList<com.rmi.Sensor> sensorsList = **new** ArrayList<com.rmi.Sensor>();

sensorsList = iSensorService.getSensorsList();

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

**return** sensorsList;

}

// disposing the jframe

**public** **void** disposeFrame() {

**super**.dispose();

}

// main method implementation

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

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

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

**try** {

// display the client no from rmi client

SensorClientRMI sensorClientRMI = **new** SensorClientRMI();

sensorClientRMI.displayClientNo();

// creating an instance of the jframe

SensorDetailsUI frame = **new** SensorDetailsUI();

// centering the jframe in the screen

Dimension dim = Toolkit.*getDefaultToolkit*().getScreenSize();

frame.setLocation(dim.width / 2 - frame.getSize().width / 2,

dim.height / 2 - frame.getSize().height / 2);

// displaying the jframe

frame.setVisible(**true**);

// display warning message if a sensor co2 level or smoke level is greater than

// 5

**if** (*status* == 1) {

JOptionPane.*showMessageDialog*(**null**,

"The CO2 level or smoke level is greater than 5 in a sensor!", "WARNING!",

JOptionPane.***WARNING\_MESSAGE***);

}

} **catch** (ConnectException e) {

// catch a connection exception due to not starting the rest api

JOptionPane.*showMessageDialog*(**null**, "Connection failed! Connect to REST API and try again!",

"WARNING!", JOptionPane.***WARNING\_MESSAGE***);

e.printStackTrace();

} **catch** (JSONException e) {

// catch a json exception due to corrupted data in mongodb

JOptionPane.*showMessageDialog*(**null**,

"JSON object isuue! Check for corrupted data in the database and try again!", "WARNING!",

JOptionPane.***WARNING\_MESSAGE***);

e.printStackTrace();

} **catch** (Exception e) {

// catch any other runtime exceptions

e.printStackTrace();

}

}

});

}

}

**LoginUI.java**

**package** com.interfaces;

**import** java.awt.Color;

**import** java.awt.Dimension;

**import** java.awt.EventQueue;

**import** java.awt.Font;

**import** java.awt.Image;

**import** java.awt.SystemColor;

**import** java.awt.Toolkit;

**import** java.awt.event.ActionEvent;

**import** java.awt.event.ActionListener;

**import** java.awt.event.KeyAdapter;

**import** java.awt.event.KeyEvent;

**import** java.net.ConnectException;

**import** javax.swing.ImageIcon;

**import** javax.swing.JButton;

**import** javax.swing.JFrame;

**import** javax.swing.JLabel;

**import** javax.swing.JOptionPane;

**import** javax.swing.JPanel;

**import** javax.swing.JPasswordField;

**import** javax.swing.JSeparator;

**import** javax.swing.JTextField;

**import** javax.swing.SwingConstants;

**import** javax.swing.border.LineBorder;

**import** org.json.JSONException;

**import** com.rmi.SensorClientRMI;

// login user interface jframe

**public** **class** LoginUI **extends** JFrame {

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

**private** JFrame frmLoginSystem;

**private** JTextField txtUsername;

**private** JPasswordField txtPassword;

**private** JLabel lblUserNotify;

// defining properties of all the design elements of the login user interface

**public** LoginUI() {

// images

Image img1 = **new** ImageIcon(**this**.getClass().getResource("/04.png")).getImage();

Image img2 = **new** ImageIcon(**this**.getClass().getResource("/07.png")).getImage();

Image img3 = **new** ImageIcon(**this**.getClass().getResource("/05.png")).getImage();

Image img4 = **new** ImageIcon(**this**.getClass().getResource("/06.png")).getImage();

Image img5 = **new** ImageIcon(**this**.getClass().getResource("/02.png")).getImage();

Image img6 = **new** ImageIcon(**this**.getClass().getResource("/03.png")).getImage();

Image img7 = **new** ImageIcon(**this**.getClass().getResource("/01.png")).getImage();

Image img8 = **new** ImageIcon(**this**.getClass().getResource("/08.png")).getImage();

frmLoginSystem = **new** JFrame();

frmLoginSystem.setResizable(**false**);

frmLoginSystem.setIconImage(img7);

frmLoginSystem.setTitle("User Login");

frmLoginSystem.getContentPane().setBackground(SystemColor.***textHighlight***);

frmLoginSystem.setBounds(200, 200, 690, 425);

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

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

frmLoginSystem.setLocationRelativeTo(**null**);

JLabel lblUserLogin = **new** JLabel("User Login");

lblUserLogin.setHorizontalAlignment(SwingConstants.***CENTER***);

lblUserLogin.setForeground(SystemColor.***text***);

lblUserLogin.setFont(**new** Font("Showcard Gothic", Font.***PLAIN***, 30));

lblUserLogin.setBounds(22, 37, 648, 67);

lblUserLogin.setIcon(**new** ImageIcon(img5));

frmLoginSystem.getContentPane().add(lblUserLogin);

// login button

JButton btnLogin = **new** JButton("Login");

btnLogin.setIcon(**new** ImageIcon(img3));

btnLogin.setForeground(**new** Color(224, 255, 255));

btnLogin.setBackground(**new** Color(210, 105, 30));

btnLogin.setBorder(**new** LineBorder(**new** Color(0, 0, 0), 1, **true**));

btnLogin.setFont(**new** Font("Tahoma", Font.***BOLD***, 25));

btnLogin.setFocusable(**false**);

btnLogin.setBounds(285, 298, 144, 54);

btnLogin.addActionListener(**new** ActionListener() {

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

String username = "admin";

String password = "admin123";

**try** {

// all the necessary validations related to administrator login

**if** (String.*valueOf*(txtPassword.getPassword()).isEmpty() | txtUsername.getText().isEmpty()) {

JOptionPane.*showMessageDialog*(**null**, "Please fill login details!", "Login Error!",

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

txtUsername.setText(**null**);

txtPassword.setText(**null**);

} **else** **if** (txtUsername.getText().equals(username)

&& String.*valueOf*(txtPassword.getPassword()).equals(password)) {

txtUsername.setText(**null**);

txtPassword.setText(**null**);

JOptionPane.*showMessageDialog*(**null**, "Logged in sucessfully!");

frmLoginSystem.dispose();

disposeFrame();

// if login credentials are correct and pass all validations, navigate the user

// to manage sensor details user interface

ManageSensorUI manageSensorUI = **new** ManageSensorUI();

manageSensorUI.setVisible(**true**);

} **else** {

JOptionPane.*showMessageDialog*(**null**, "Invalid login details. Please try again!", "Login Error!",

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

txtUsername.setText(**null**);

txtPassword.setText(**null**);

}

} **catch** (Exception e1) {

e1.printStackTrace();

}

}

});

frmLoginSystem.getContentPane().add(btnLogin);

// reset button

JButton btnReset = **new** JButton("Reset");

btnReset.setIcon(**new** ImageIcon(img4));

btnReset.setForeground(**new** Color(224, 255, 255));

btnReset.setBackground(**new** Color(30, 144, 255));

btnReset.setBorder(**new** LineBorder(**new** Color(0, 0, 0), 1, **true**));

btnReset.setFont(**new** Font("Tahoma", Font.***PLAIN***, 20));

btnReset.setBounds(114, 310, 120, 35);

btnReset.setFocusable(**false**);

btnReset.addActionListener(**new** ActionListener() {

@SuppressWarnings("deprecation")

**public** **void** actionPerformed(ActionEvent arg0) {

// necessary validations

**if** (!txtUsername.getText().isEmpty() || !txtPassword.~~getText~~().isEmpty()) {

**int** action = JOptionPane.*showConfirmDialog*(**null**, "Do you really want to reset data?", "Reset Data",

JOptionPane.***YES\_NO\_OPTION***);

**if** (action == 0) {

txtUsername.setText(**null**);

txtPassword.setText(**null**);

}

}

}

});

frmLoginSystem.getContentPane().add(btnReset);

// exit button

JButton btnExit = **new** JButton(" Exit");

btnExit.setIcon(**new** ImageIcon(img2));

btnExit.setForeground(**new** Color(224, 255, 255));

btnExit.setBackground(**new** Color(139, 0, 0));

btnExit.setBorder(**new** LineBorder(**new** Color(0, 0, 0), 1, **true**));

btnExit.setFont(**new** Font("Tahoma", Font.***PLAIN***, 20));

btnExit.setFocusable(**false**);

btnExit.setBounds(474, 310, 120, 35);

btnExit.addActionListener(**new** ActionListener() {

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

frmLoginSystem = **new** JFrame("Exit");

**if** (JOptionPane.*showConfirmDialog*(frmLoginSystem, "Confirm if you really want to exit.",

"Login Exit Confirmation", JOptionPane.***YES\_NO\_OPTION***) == JOptionPane.***YES\_NO\_OPTION***) {

System.*exit*(JFrame.***EXIT\_ON\_CLOSE***);

}

}

});

frmLoginSystem.getContentPane().add(btnExit);

JSeparator separator1 = **new** JSeparator();

separator1.setBounds(22, 105, 648, 2);

frmLoginSystem.getContentPane().add(separator1);

JPanel panel = **new** JPanel();

panel.setBackground(Color.***BLACK***);

panel.setBounds(10, 11, 670, 403);

panel.setLayout(**null**);

frmLoginSystem.getContentPane().add(panel);

txtUsername = **new** JTextField();

txtUsername.setBounds(242, 130, 279, 35);

txtUsername.setBorder(**new** LineBorder(**new** Color(0, 0, 0), 1, **true**));

txtUsername.setFont(**new** Font("Tahoma", Font.***PLAIN***, 18));

txtUsername.setColumns(10);

txtUsername.addKeyListener(**new** KeyAdapter() {

@Override

**public** **void** keyReleased(KeyEvent arg0) {

// checking whether username is valid

String userName = txtUsername.getText();

**if** (!userName.matches("[a-zA-Z0-9 ,]+")) {

lblUserNotify.setText("Please enter a valid username.");

txtUsername.setText(**null**);

} **else** {

lblUserNotify.setText(**null**);

}

}

});

panel.add(txtUsername);

JLabel lblUsername = **new** JLabel("");

lblUsername.setBounds(118, 115, 104, 67);

lblUsername.setHorizontalAlignment(SwingConstants.***CENTER***);

lblUsername.setIcon(**new** ImageIcon(img6));

lblUsername.setForeground(SystemColor.***menu***);

lblUsername.setFont(**new** Font("Tw Cen MT", Font.***PLAIN***, 25));

panel.add(lblUsername);

JLabel lblPassword = **new** JLabel("");

lblPassword.setBounds(118, 193, 104, 54);

lblPassword.setForeground(SystemColor.***menu***);

lblPassword.setFont(**new** Font("Tw Cen MT", Font.***PLAIN***, 25));

lblPassword.setHorizontalAlignment(SwingConstants.***CENTER***);

lblPassword.setIcon(**new** ImageIcon(img1));

panel.add(lblPassword);

txtPassword = **new** JPasswordField();

txtPassword.setBounds(240, 201, 281, 32);

txtPassword.setBorder(**new** LineBorder(**new** Color(0, 0, 0), 1, **true**));

txtPassword.setFont(**new** Font("Tahoma", Font.***PLAIN***, 18));

panel.add(txtPassword);

JSeparator separator = **new** JSeparator();

separator.setBounds(10, 265, 648, 2);

panel.add(separator);

JLabel lblNewLabel = **new** JLabel("");

lblNewLabel.setHorizontalAlignment(SwingConstants.***CENTER***);

lblNewLabel.setIcon(**new** ImageIcon(img8));

lblNewLabel.setForeground(SystemColor.***text***);

lblNewLabel.setBounds(0, 0, 670, 403);

panel.add(lblNewLabel);

lblUserNotify = **new** JLabel("");

lblUserNotify.setForeground(SystemColor.***text***);

lblUserNotify.setBounds(242, 166, 279, 16);

panel.add(lblUserNotify);

frmLoginSystem.setUndecorated(**true**);

}

// displaying the jframe

**public** **void** displayFrame() {

LoginUI window = **new** LoginUI();

window.frmLoginSystem.setVisible(**true**);

}

// disposing the jframe

**public** **void** disposeFrame() {

**super**.dispose();

}

// main method implementation

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

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

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

**try** {

// display the client no from rmi client

SensorClientRMI sensorClientRMI = **new** SensorClientRMI();

sensorClientRMI.displayClientNo();

// creating an instance of the sensor details jframe

SensorDetailsUI frame = **new** SensorDetailsUI();

// centering the jframe in the screen

Dimension dim = Toolkit.*getDefaultToolkit*().getScreenSize();

frame.setLocation(dim.width / 2 - frame.getSize().width / 2,

dim.height / 2 - frame.getSize().height / 2);

// displaying the jframe

frame.setVisible(**true**);

// display warning message if a sensor co2 level or smoke level is greater than

// 5

**if** (SensorDetailsUI.*status* == 1) {

JOptionPane.*showMessageDialog*(**null**,

"The CO2 level or smoke level is greater than 5 in a sensor!", "WARNING!",

JOptionPane.***WARNING\_MESSAGE***);

}

} **catch** (ConnectException e) {

// catch a connection exception due to not starting the rest api

JOptionPane.*showMessageDialog*(**null**, "Connection failed! Connect to REST API and try again!",

"WARNING!", JOptionPane.***WARNING\_MESSAGE***);

e.printStackTrace();

} **catch** (JSONException e) {

// catch a json exception due to corrupted data in mongodb

JOptionPane.*showMessageDialog*(**null**,

"JSON object isuue! Check for corrupted data in the database and try again!", "WARNING!",

JOptionPane.***WARNING\_MESSAGE***);

e.printStackTrace();

} **catch** (Exception e) {

// catch any other runtime exceptions

e.printStackTrace();

}

}

});

}

}

**ManageSensorUI.java**

**package** com.interfaces;

**import** java.awt.Color;

**import** java.awt.Dimension;

**import** java.awt.EventQueue;

**import** java.awt.Font;

**import** java.awt.Image;

**import** java.awt.SystemColor;

**import** java.awt.Toolkit;

**import** java.awt.event.ActionEvent;

**import** java.awt.event.ActionListener;

**import** java.awt.event.FocusAdapter;

**import** java.awt.event.FocusEvent;

**import** java.awt.event.MouseAdapter;

**import** java.awt.event.MouseEvent;

**import** java.io.IOException;

**import** java.net.ConnectException;

**import** java.util.ArrayList;

**import** javax.swing.ImageIcon;

**import** javax.swing.JButton;

**import** javax.swing.JCheckBox;

**import** javax.swing.JFrame;

**import** javax.swing.JLabel;

**import** javax.swing.JOptionPane;

**import** javax.swing.JPanel;

**import** javax.swing.JScrollPane;

**import** javax.swing.JTable;

**import** javax.swing.JTextField;

**import** javax.swing.ListSelectionModel;

**import** javax.swing.SwingConstants;

**import** javax.swing.Timer;

**import** javax.swing.border.BevelBorder;

**import** javax.swing.border.EmptyBorder;

**import** javax.swing.border.LineBorder;

**import** javax.swing.table.DefaultTableCellRenderer;

**import** javax.swing.table.DefaultTableModel;

**import** org.json.JSONException;

**import** com.rmi.SensorClientRMI;

**import** com.services.ISensorService;

**import** com.services.SensorService;

// manage sensor details user interface jframe

**public** **class** ManageSensorUI **extends** JFrame {

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

**private** JPanel contentPane;

**private** JTextField txtSensorId;

**private** JTextField txtSensorName;

**private** JTextField txtRoomNo;

**private** JTextField txtFloorNo;

**private** JCheckBox checkboxIsActive;

**private** JPanel panel = **new** JPanel();

**private** com.rmi.Sensor sensor = **new** com.rmi.Sensor();

**private** ISensorService iSensorService = (ISensorService) **new** SensorService();

**private** ArrayList<com.rmi.Sensor> sensorsList = **new** ArrayList<com.rmi.Sensor>();

**private** Timer timer;

**private** **final** **static** **int** ***INTERVAL*** = 5000;

// defining properties of all the design elements of the manage senssor details

// user interface

**public** ManageSensorUI() **throws** IOException {

// images

Image img1 = **new** ImageIcon(**this**.getClass().getResource("/10.png")).getImage();

Image img2 = **new** ImageIcon(**this**.getClass().getResource("/11.png")).getImage();

Image img3 = **new** ImageIcon(**this**.getClass().getResource("/07.png")).getImage();

Image img4 = **new** ImageIcon(**this**.getClass().getResource("/06.png")).getImage();

Image img5 = **new** ImageIcon(**this**.getClass().getResource("/09.png")).getImage();

Image img6 = **new** ImageIcon(**this**.getClass().getResource("/13.png")).getImage();

setResizable(**false**);

setTitle("Manage Sensor");

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

setBounds(100, 100, 1028, 632);

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

contentPane = **new** JPanel();

contentPane.setBackground(**new** Color(255, 255, 255));

contentPane.setBorder(**new** EmptyBorder(5, 5, 5, 5));

contentPane.setLayout(**null**);

setContentPane(contentPane);

JPanel panel3 = **new** JPanel();

panel3.setBorder(**new** BevelBorder(BevelBorder.***LOWERED***, **null**, **null**, **null**, **null**));

panel3.setBackground(**new** Color(30, 144, 255));

panel3.setForeground(SystemColor.***text***);

panel3.setBounds(0, 0, 1022, 83);

panel3.setLayout(**null**);

contentPane.add(panel3);

// logout button

JButton btnLogout = **new** JButton("Log Out");

btnLogout.setBorder(**new** LineBorder(**new** Color(0, 0, 0), 2, **true**));

btnLogout.setBounds(885, 25, 113, 35);

btnLogout.setFont(**new** Font("Tahoma", Font.***BOLD***, 16));

btnLogout.setBackground(**new** Color(204, 204, 204));

btnLogout.setForeground(**new** Color(0, 0, 0));

btnLogout.setFocusable(**false**);

btnLogout.addMouseListener(**new** MouseAdapter() {

@Override

**public** **void** mouseClicked(MouseEvent e) {

**try** {

**if** (JOptionPane.*showConfirmDialog*(**null**, "Do you really want to logout?", "Logout Confirmation",

JOptionPane.***YES\_NO\_OPTION***) == JOptionPane.***YES\_NO\_OPTION***) {

// creating an instance of the sensor details jframe

SensorDetailsUI sensorDetailsUI = **new** SensorDetailsUI();

// centering the jframe in the screen

Dimension dim = Toolkit.*getDefaultToolkit*().getScreenSize();

sensorDetailsUI.setLocation(dim.width / 2 - sensorDetailsUI.getSize().width / 2,

dim.height / 2 - sensorDetailsUI.getSize().height / 2);

// displaying the jframe

sensorDetailsUI.setVisible(**true**);

// disposing the current jframe

disposeFrame();

// display warning message if a sensor co2 level or smoke level is greater than

// 5

**if** (SensorDetailsUI.*status* == 1) {

JOptionPane.*showMessageDialog*(**null**,

"The CO2 level or smoke level is greater than 5 in a sensor!", "WARNING!",

JOptionPane.***WARNING\_MESSAGE***);

}

}

} **catch** (Exception e1) {

e1.printStackTrace();

}

}

});

panel3.add(btnLogout);

JLabel lblTitle = **new** JLabel("Manage Sensors");

lblTitle.setIcon(**new** ImageIcon(img5));

lblTitle.setFont(**new** Font("Showcard Gothic", Font.***PLAIN***, 30));

lblTitle.setForeground(SystemColor.***text***);

lblTitle.setHorizontalAlignment(SwingConstants.***CENTER***);

lblTitle.setBounds(12, 11, 998, 65);

panel3.add(lblTitle);

JPanel panel1 = **new** JPanel();

panel1.setBackground(**new** Color(0, 0, 51));

panel1.setBounds(0, 81, 1022, 10);

contentPane.add(panel1);

JPanel panelManage = **new** JPanel();

panelManage.setBounds(0, 0, 1022, 603);

panelManage.setLayout(**null**);

contentPane.add(panelManage);

panel.setBounds(0, 88, 1022, 515);

panel.setBorder(**new** LineBorder(**new** Color(0, 0, 0), 1, **true**));

panel.setBackground(**new** Color(0, 51, 102));

panel.setLayout(**null**);

panelManage.add(panel);

// add button

JButton btnAdd = **new** JButton("Add");

btnAdd.setIcon(**new** ImageIcon(img6));

btnAdd.setBorder(**new** LineBorder(**new** Color(0, 0, 0), 1, **true**));

btnAdd.setForeground(**new** Color(255, 255, 255));

btnAdd.setBackground(**new** Color(210, 105, 30));

btnAdd.setFont(**new** Font("Tahoma", Font.***BOLD***, 16));

btnAdd.setBounds(265, 437, 113, 47);

btnAdd.setFocusable(**false**);

btnAdd.addActionListener(**new** ActionListener() {

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

**try** {

// validations related to insert sensor

**if** (!txtSensorId.getText().isEmpty() && !txtSensorName.getText().isEmpty()

&& !txtRoomNo.getText().isEmpty() && !txtFloorNo.getText().isEmpty()) {

**int** action = JOptionPane.*showConfirmDialog*(**null**, "Do you really want to add a new sensor?",

"Add Sensor", JOptionPane.***YES\_NO\_OPTION***);

**if** (action == 0) {

// getting the inserted values for the sensor

sensor.setSensorId(txtSensorId.getText());

sensor.setSensorName(txtSensorName.getText());

sensor.setFloorNo(Integer.*parseInt*(txtFloorNo.getText()));

sensor.setRoomNo(Integer.*parseInt*(txtRoomNo.getText()));

sensor.setActive(checkboxIsActive.isSelected());

// checking for any errors from mongodb such as duplicate sensor id

**boolean** status = iSensorService.addSensor(sensor);

**if** (!status) {

JOptionPane.*showMessageDialog*(**null**,

"Sensor ID already exists! Please try again with a unique Sensor ID.",

"Insert Error!", JOptionPane.***ERROR\_MESSAGE***);

txtSensorId.setText(**null**);

} **else** {

// if sensor added successfully

JOptionPane.*showMessageDialog*(**null**, "Sensor added sucessfully.");

resetFields();

}

}

} **else** {

JOptionPane.*showMessageDialog*(**null**, "Fill all the fields and try again!", "Insert Error!",

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

}

} **catch** (Exception e1) {

e1.printStackTrace();

} **finally** {

**try** {

// refresh jtable after adding a new sensor

displayTable();

} **catch** (IOException e1) {

e1.printStackTrace();

}

}

}

});

panel.add(btnAdd);

// update button

JButton btnUpdate = **new** JButton("Update");

btnUpdate.setIcon(**new** ImageIcon(img1));

btnUpdate.setBorder(**new** LineBorder(**new** Color(0, 0, 0), 1, **true**));

btnUpdate.setForeground(**new** Color(255, 255, 255));

btnUpdate.setBackground(**new** Color(210, 105, 30));

btnUpdate.setFont(**new** Font("Tahoma", Font.***BOLD***, 16));

btnUpdate.setFocusable(**false**);

btnUpdate.setBounds(454, 437, 113, 47);

btnUpdate.addActionListener(**new** ActionListener() {

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

**try** {

// validations related to update sensor

**if** (!txtSensorId.getText().isEmpty() && !txtSensorName.getText().isEmpty()

&& !txtRoomNo.getText().isEmpty() && !txtFloorNo.getText().isEmpty()) {

**int** action = JOptionPane.*showConfirmDialog*(**null**, "Do you really want to update the sensor?",

"Update Sensor", JOptionPane.***YES\_NO\_OPTION***);

**if** (action == 0) {

// getting the new values for the sensor

sensor.setSensorId(txtSensorId.getText());

sensor.setSensorName(txtSensorName.getText());

sensor.setFloorNo(Integer.*parseInt*(txtFloorNo.getText()));

sensor.setRoomNo(Integer.*parseInt*(txtRoomNo.getText()));

sensor.setActive(checkboxIsActive.isSelected());

iSensorService.updateSensor(sensor.getSensorId(), sensor);

JOptionPane.*showMessageDialog*(**null**, "Sensor updated sucessfully.");

resetFields();

}

} **else** {

JOptionPane.*showMessageDialog*(**null**, "Fill all the fields and try again!", "Update Error!",

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

}

} **catch** (Exception e1) {

JOptionPane.*showMessageDialog*(**null**, "Please select any sensor to update.", "WARNING!",

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

} **finally** {

**try** {

// refresh jtable after updating an existing sensor

displayTable();

} **catch** (IOException e1) {

e1.printStackTrace();

}

}

}

});

panel.add(btnUpdate);

// remove button

JButton btnRemove = **new** JButton("Remove");

btnRemove.setFont(**new** Font("Tahoma", Font.***BOLD***, 16));

btnRemove.setIcon(**new** ImageIcon(img2));

btnRemove.setBorder(**new** LineBorder(**new** Color(0, 0, 0), 1, **true**));

btnRemove.setForeground(**new** Color(255, 255, 255));

btnRemove.setBackground(**new** Color(210, 105, 30));

btnRemove.setFocusable(**false**);

btnRemove.setBounds(643, 437, 113, 47);

btnRemove.addActionListener(**new** ActionListener() {

**public** **void** actionPerformed(ActionEvent arg0) {

**try** {

// validations related to delete sensor

**if** (txtSensorId.getText().isEmpty() == **true**) {

JOptionPane.*showMessageDialog*(**null**, "Please select any sensor to remove." + "\n", "WARNING!",

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

} **else** {

**int** action = JOptionPane.*showConfirmDialog*(**null**, "Do you really want to remove?",

"Remove Sensor", JOptionPane.***YES\_NO\_OPTION***);

**if** (action == 0) {

sensor.setSensorId(txtSensorId.getText());

iSensorService.removeSensor(sensor.getSensorId());

JOptionPane.*showMessageDialog*(**null**, "Sensor deleted sucessfully.");

resetFields();

}

}

} **catch** (Exception e) {

JOptionPane.*showMessageDialog*(**null**, e.getMessage(), "WARNING!", JOptionPane.***ERROR\_MESSAGE***);

} **finally** {

**try** {

// refresh jtable after deleting a sensor

displayTable();

} **catch** (IOException e) {

e.printStackTrace();

}

}

}

});

panel.add(btnRemove);

// exit button

JButton btnExit = **new** JButton("Exit");

btnExit.setFont(**new** Font("Tahoma", Font.***BOLD***, 16));

btnExit.setIcon(**new** ImageIcon(img3));

btnExit.setForeground(**new** Color(255, 255, 255));

btnExit.setBackground(**new** Color(178, 34, 34));

btnExit.setFocusable(**false**);

btnExit.setBounds(832, 437, 113, 47);

btnExit.addActionListener(**new** ActionListener() {

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

**if** (JOptionPane.*showConfirmDialog*(**null**, "Do you really want to exit?", "Exit Confirmation",

JOptionPane.***YES\_NO\_OPTION***) == JOptionPane.***YES\_NO\_OPTION***) {

System.*exit*(JFrame.***EXIT\_ON\_CLOSE***);

}

}

});

panel.add(btnExit);

JPanel panel2 = **new** JPanel();

panel2.setBackground(**new** Color(102, 204, 255));

panel2.setBounds(38, 251, 947, 160);

panel2.setLayout(**null**);

panel.add(panel2);

JLabel lblSensorID = **new** JLabel("Sensor ID");

lblSensorID.setFont(**new** Font("Dialog", Font.***PLAIN***, 16));

lblSensorID.setBounds(27, 21, 118, 20);

panel2.add(lblSensorID);

txtSensorId = **new** JTextField();

txtSensorId.setFont(**new** Font("Dialog", Font.***PLAIN***, 13));

txtSensorId.setBounds(188, 21, 232, 20);

txtSensorId.setBackground(**new** Color(224, 255, 255));

txtSensorId.setEditable(**true**);

txtSensorId.setBorder(**new** LineBorder(Color.***BLACK***, 1, **true**));

txtSensorId.setColumns(10);

txtSensorId.addFocusListener(**new** FocusAdapter() {

@Override

**public** **void** focusLost(FocusEvent e) {

// validating the sensor id

String sensorId = txtSensorId.getText();

**if** (!sensorId.matches("[a-zA-Z0-9 ,]+") && !sensorId.isEmpty()) {

JOptionPane.*showMessageDialog*(**null**, "Please enter a valid sensor id with only letters and digits.");

txtSensorId.setText(**null**);

}

}

});

panel2.add(txtSensorId);

JLabel lblSensorName = **new** JLabel("Sensor Name");

lblSensorName.setFont(**new** Font("Dialog", Font.***PLAIN***, 16));

lblSensorName.setBounds(27, 65, 149, 20);

panel2.add(lblSensorName);

txtSensorName = **new** JTextField();

txtSensorName.setBounds(188, 65, 232, 20);

txtSensorName.setBorder(**new** LineBorder(**new** Color(0, 0, 51), 1, **true**));

txtSensorName.setFont(**new** Font("Dialog", Font.***PLAIN***, 13));

txtSensorName.setBackground(**new** Color(224, 255, 255));

txtSensorName.setColumns(10);

txtSensorName.addFocusListener(**new** FocusAdapter() {

@Override

**public** **void** focusLost(FocusEvent e) {

// validating the sensor name

String sensorName = txtSensorName.getText();

**if** (!sensorName.matches("[a-zA-Z0-9 ,]+") && !sensorName.isEmpty()) {

JOptionPane.*showMessageDialog*(**null**,

"Please enter a valid sensor name with only letters and digits.");

txtSensorName.setText(**null**);

}

}

});

panel2.add(txtSensorName);

JLabel lblRoomNo = **new** JLabel("Room Number");

lblRoomNo.setFont(**new** Font("Dialog", Font.***PLAIN***, 16));

lblRoomNo.setBounds(490, 21, 118, 20);

panel2.add(lblRoomNo);

JLabel lblFloorNo = **new** JLabel("Floor Number");

lblFloorNo.setFont(**new** Font("Dialog", Font.***PLAIN***, 16));

lblFloorNo.setBounds(490, 66, 131, 20);

panel2.add(lblFloorNo);

txtRoomNo = **new** JTextField();

txtRoomNo.setFont(**new** Font("Dialog", Font.***PLAIN***, 13));

txtRoomNo.setEditable(**true**);

txtRoomNo.setColumns(10);

txtRoomNo.setBorder(**new** LineBorder(Color.***BLACK***, 1, **true**));

txtRoomNo.setBackground(**new** Color(224, 255, 255));

txtRoomNo.setBounds(654, 21, 232, 20);

txtRoomNo.addFocusListener(**new** FocusAdapter() {

@Override

**public** **void** focusLost(FocusEvent e) {

// validating the room number

String roomNo = txtRoomNo.getText();

**if** (!roomNo.matches("[0-9 ,]+") && !roomNo.isEmpty()) {

JOptionPane.*showMessageDialog*(**null**, "Please enter a valid room number.");

txtRoomNo.setText(**null**);

}

}

});

panel2.add(txtRoomNo);

txtFloorNo = **new** JTextField();

txtFloorNo.setFont(**new** Font("Dialog", Font.***PLAIN***, 13));

txtFloorNo.setEditable(**true**);

txtFloorNo.setColumns(10);

txtFloorNo.setBorder(**new** LineBorder(Color.***BLACK***, 1, **true**));

txtFloorNo.setBackground(**new** Color(224, 255, 255));

txtFloorNo.setBounds(654, 65, 232, 20);

txtFloorNo.addFocusListener(**new** FocusAdapter() {

@Override

**public** **void** focusLost(FocusEvent e) {

// validating the floor number

String floorNo = txtFloorNo.getText();

**if** (!floorNo.matches("[0-9 ,]+") && !floorNo.isEmpty()) {

JOptionPane.*showMessageDialog*(**null**, "Please enter a valid floor number.");

txtFloorNo.setText(**null**);

}

}

});

panel2.add(txtFloorNo);

// is active checkbox

checkboxIsActive = **new** JCheckBox("Is Active");

checkboxIsActive.setSelected(**true**);

checkboxIsActive.setFont(**new** Font("Dialog", Font.***PLAIN***, 16));

checkboxIsActive.setFocusable(**false**);

checkboxIsActive.setBackground(**new** Color(102, 204, 255));

checkboxIsActive.setBounds(414, 115, 97, 23);

panel2.add(checkboxIsActive);

// reset button

JButton btnReset = **new** JButton("Reset");

btnReset.setFont(**new** Font("Tahoma", Font.***BOLD***, 16));

btnReset.setBackground(SystemColor.***controlText***);

btnReset.setForeground(SystemColor.***text***);

btnReset.setBounds(76, 437, 113, 47);

btnReset.setIcon(**new** ImageIcon(img4));

btnReset.setFocusable(**false**);

btnReset.addActionListener(**new** ActionListener() {

**public** **void** actionPerformed(ActionEvent arg0) {

**try** {

// all necessary validations related to resetting the input fields

**if** (!txtSensorId.getText().isEmpty() || !txtSensorName.getText().isEmpty()

|| !txtFloorNo.getText().isEmpty() || !txtRoomNo.getText().isEmpty()) {

**int** action = JOptionPane.*showConfirmDialog*(**null**, "Do you really want to reset data?",

"Reset Data", JOptionPane.***YES\_NO\_OPTION***);

**if** (action == 0) {

resetFields();

}

}

} **catch** (Exception e) {

JOptionPane.*showMessageDialog*(**null**, e);

}

}

});

panel.add(btnReset);

displayTable();

// refresh the table

timer = **new** Timer(***INTERVAL***, **new** ActionListener() {

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

**try** {

displayTable();

} **catch** (IOException e1) {

e1.printStackTrace();

} **catch** (Exception e1) {

e1.printStackTrace();

}

}

});

timer.start();

}

// method implementation to get the sensor arraylist from sensor service

**public** ArrayList<com.rmi.Sensor> refreshTable() **throws** IOException {

sensorsList = iSensorService.getSensorsList();

**return** sensorsList;

}

// method implementation to dispose the jframe

**public** **void** disposeFrame() {

**super**.dispose();

}

// method implementation to display the jtable

**public** **void** displayTable() **throws** IOException {

JScrollPane scrollPane = **new** JScrollPane();

scrollPane.setBackground(**new** Color(255, 255, 255));

scrollPane.setFont(**new** Font("Tahoma", Font.***BOLD***, 25));

scrollPane.setBounds(38, 34, 947, 195);

panel.add(scrollPane);

String col[] = { "Sensor ID", "Sensor Name", "Is Active", "Floor No", "Room No" };

DefaultTableModel tableModel = **new** DefaultTableModel(col, 0) {

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

@Override

**public** **boolean** isCellEditable(**int** row, **int** column) {

**return** **false**;

}

};

JTable table = **new** JTable(tableModel);

table.setColumnSelectionAllowed(**true**);

table.setCellSelectionEnabled(**false**);

table.setBorder(**new** LineBorder(**new** Color(0, 0, 0)));

table.setSelectionMode(ListSelectionModel.***SINGLE\_SELECTION***);

table.setForeground(Color.***blue***);

table.setRowHeight(24);

table.setFont(**new** Font("Arial", Font.***BOLD***, 12));

table.setEnabled(**true**);

table.setSurrendersFocusOnKeystroke(**true**);

table.setShowVerticalLines(**true**);

table.setFont(**new** Font("Tahoma", Font.***PLAIN***, 16));

table.setBackground(**new** Color(240, 230, 140));

table.setSurrendersFocusOnKeystroke(**true**);

table.setFocusable(**false**);

table.setRowSelectionAllowed(**true**);

DefaultTableCellRenderer centerRenderer = **new** DefaultTableCellRenderer();

centerRenderer.setHorizontalAlignment(JLabel.***CENTER***);

table.getColumnModel().getColumn(0).setCellRenderer(centerRenderer);

table.getColumnModel().getColumn(1).setCellRenderer(centerRenderer);

table.getColumnModel().getColumn(2).setCellRenderer(centerRenderer);

table.getColumnModel().getColumn(3).setCellRenderer(centerRenderer);

table.getColumnModel().getColumn(4).setCellRenderer(centerRenderer);

// retrieving sensor details from the arraylist and display in the jtable

sensorsList = refreshTable();

**for** (com.rmi.Sensor sensor1 : sensorsList) {

String sensorId = sensor1.getSensorId();

String sensorName = sensor1.getSensorName();

**int** roomNo = sensor1.getRoomNo();

**int** floorNo = sensor1.getFloorNo();

String isActive;

**if** (sensor1.isActive()) {

isActive = "Active";

} **else** {

isActive = "Not Active";

}

Object[] objs = { sensorId, sensorName, isActive, floorNo, roomNo };

tableModel.addRow(objs);

}

// triggers when click on a specific row of the jtable

table.addMouseListener(**new** MouseAdapter() {

@Override

**public** **void** mouseClicked(MouseEvent arg0) {

**try** {

// get the sensor id of the selected row

**int** row = table.getSelectedRow();

String sensorId = (table.getModel().getValueAt(row, 0)).toString();

// retrieve the details of the sensor with that sensor id

com.rmi.Sensor sensor1 = iSensorService.getSensor(sensorId);

// set the sensor details in the input fields

txtSensorId.setText(sensor1.getSensorId());

txtSensorName.setText(sensor1.getSensorName());

txtFloorNo.setText(Integer.*toString*(sensor1.getFloorNo()));

txtRoomNo.setText(Integer.*toString*(sensor1.getRoomNo()));

checkboxIsActive.setSelected(sensor1.isActive());

} **catch** (Exception e) {

JOptionPane.*showMessageDialog*(**null**, e);

}

}

});

scrollPane.setViewportView(table);

}

// reset input fields to empty

**public** **void** resetFields() {

txtSensorId.setText(**null**);

txtSensorName.setText(**null**);

txtFloorNo.setText(**null**);

txtRoomNo.setText(**null**);

}

// main method implementation

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

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

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

**try** {

// display the client no from rmi client

SensorClientRMI sensorClientRMI = **new** SensorClientRMI();

sensorClientRMI.displayClientNo();

// creating an instance of the sensor details jframe

SensorDetailsUI frame = **new** SensorDetailsUI();

// centering the jframe in the screen

Dimension dim = Toolkit.*getDefaultToolkit*().getScreenSize();

frame.setLocation(dim.width / 2 - frame.getSize().width / 2,

dim.height / 2 - frame.getSize().height / 2);

// displaying the jframe

frame.setVisible(**true**);

// display warning message if a sensor co2 level or smoke level is greater than

// 5

**if** (SensorDetailsUI.*status* == 1) {

JOptionPane.*showMessageDialog*(**null**,

"The CO2 level or smoke level is greater than 5 in a sensor!", "WARNING!",

JOptionPane.***WARNING\_MESSAGE***);

}

} **catch** (ConnectException e) {

// catch a connection exception due to not starting the rest api

JOptionPane.*showMessageDialog*(**null**, "Connection failed! Connect to REST API and try again!",

"WARNING!", JOptionPane.***WARNING\_MESSAGE***);

e.printStackTrace();

} **catch** (JSONException e) {

// catch a json exception due to corrupted data in mongodb

JOptionPane.*showMessageDialog*(**null**,

"JSON object isuue! Check for corrupted data in the database and try again!", "WARNING!",

JOptionPane.***WARNING\_MESSAGE***);

e.printStackTrace();

} **catch** (Exception e) {

// catch any other runtime exceptions

e.printStackTrace();

}

}

});

}

}

**ISensorService.java**

**package** com.services;

**import** java.io.IOException;

**import** java.rmi.RemoteException;

**import** java.util.ArrayList;

// interface for sensor service with method declarations

**public** **interface** ISensorService {

**public** **boolean** addSensor(com.rmi.Sensor sensor) **throws** RemoteException, IOException;

**public** **void** updateSensor(String sensorId, com.rmi.Sensor sensor) **throws** RemoteException, IOException;

**public** **void** removeSensor(String sensorId) **throws** RemoteException, IOException;

**public** com.rmi.Sensor getSensor(String sensorId) **throws** RemoteException, IOException;

**public** ArrayList<com.rmi.Sensor> getSensorsList() **throws** RemoteException, IOException;

}

**SensorService.java**

**package** com.services;

**import** java.io.IOException;

**import** java.rmi.RemoteException;

**import** java.util.ArrayList;

**import** com.rmi.SensorServerRMI;

// sensor service class

**public** **class** SensorService **implements** ISensorService {

**private** com.rmi.SensorServerRMI sensorServerRMI;

// default constructor to create an object of the rmi server class

**public** SensorService() **throws** RemoteException {

**super**();

sensorServerRMI = **new** com.rmi.SensorServerRMI();

}

// method implementation to call rmi server method for inserting a sensor

@Override

**public** **boolean** addSensor(com.rmi.Sensor sensor) **throws** RemoteException, IOException {

**return** sensorServerRMI.addSensor(sensor);

}

// method implementation to call rmi server method for updating an existing

// sensor

@Override

**public** **void** updateSensor(String sensorId, com.rmi.Sensor sensor) **throws** RemoteException, IOException {

sensorServerRMI.updateSensor(sensorId, sensor);

}

// method implementation to call rmi server method for deleting a sensor

@Override

**public** **void** removeSensor(String sensorId) **throws** RemoteException, IOException {

sensorServerRMI.removeSensor(sensorId);

}

// method implementation to call rmi server method for retrieving a sensor by

// sensor id

@Override

**public** com.rmi.Sensor getSensor(String sensorId) **throws** RemoteException, IOException {

**return** sensorServerRMI.getSensor(sensorId);

}

// method implementation to call rmi server method for retrieving an arraylist of

// all sensors in mongodb

@Override

**public** ArrayList<com.rmi.Sensor> getSensorsList() **throws** RemoteException, IOException {

System.***out***.println(SensorServerRMI.*getSensorsList*());

**return** SensorServerRMI.*getSensorsList*();

}

}

**ISensorServerRMI.java**

**package** com.rmi;

**import** java.io.IOException;

**import** java.rmi.Remote;

**import** java.rmi.RemoteException;

// interface for the rmi server with method declarations

**public** **interface** ISensorServerRMI **extends** Remote {

**public** **boolean** addSensor(Sensor sensor) **throws** RemoteException, IOException;

**public** **void** updateSensor(String sensorId, Sensor sensor) **throws** RemoteException, IOException;

**public** **void** removeSensor(String sensorId) **throws** RemoteException, IOException;

**public** Sensor getSensor(String sensorId) **throws** RemoteException, IOException;

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

}

**SensorServerRMI.java**

**package** com.rmi;

**import** java.io.BufferedReader;

**import** java.io.IOException;

**import** java.io.InputStreamReader;

**import** java.io.OutputStreamWriter;

**import** java.net.HttpURLConnection;

**import** java.net.URL;

**import** java.nio.charset.Charset;

**import** java.rmi.RemoteException;

**import** java.rmi.registry.LocateRegistry;

**import** java.rmi.registry.Registry;

**import** java.rmi.server.UnicastRemoteObject;

**import** java.util.ArrayList;

**import** org.json.JSONArray;

**import** org.json.JSONObject;

// rmi server class

**public** **class** SensorServerRMI **extends** UnicastRemoteObject **implements** ISensorServerRMI {

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

**private** **int** count = 0;

**public** **static** ArrayList<Sensor> *sensors*;

**public** SensorServerRMI() **throws** RemoteException {

**super**();

}

// method implementation to insert a sensor to mongodb

@Override

**public** **boolean** addSensor(Sensor sensor) **throws** RemoteException, IOException {

String url = "http://localhost:5000/api/sensor/";

URL object = **new** URL(url);

// creating http post request to send to rest api

HttpURLConnection con = (HttpURLConnection) object.openConnection();

con.setDoOutput(**true**);

con.setDoInput(**true**);

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

con.setRequestProperty("Accept", "application/json");

con.setRequestMethod("POST");

// adding attributes of a sensor and creating a json object

JSONObject obj = **new** JSONObject();

obj.put("smokeLevel", sensor.getSmokeLevel());

obj.put("co2Level", sensor.getCO2Level());

obj.put("id", sensor.getSensorId());

obj.put("floorNo", sensor.getFloorNo());

obj.put("name", sensor.getSensorName());

obj.put("roomNo", sensor.getRoomNo());

obj.put("active", sensor.isActive());

OutputStreamWriter wr = **new** OutputStreamWriter(con.getOutputStream());

wr.write(obj.toString());

wr.flush();

StringBuilder sb = **new** StringBuilder();

**int** HttpResult = con.getResponseCode();

**if** (HttpResult == HttpURLConnection.***HTTP\_OK***) {

// if http response code for OK (200) returns from the rest api

// if the inserting operation is successful

BufferedReader br = **new** BufferedReader(**new** InputStreamReader(con.getInputStream(), "utf-8"));

String line = **null**;

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

sb.append(line + "\n");

}

br.close();

// return true if the inserting operation is successful

**return** **true**;

} **else** {

// if http response code for OK (200) does not return from the rest api

// return false if the inserting operation is not successful

**return** **false**;

}

}

// method implementation to update a sensor in mongodb

@Override

**public** **void** updateSensor(String sensorId, Sensor sensor) **throws** RemoteException, IOException {

String url = "http://localhost:5000/api/sensor/" + sensorId;

URL object = **new** URL(url);

// creating http put request to send to rest api

HttpURLConnection con = (HttpURLConnection) object.openConnection();

con.setDoOutput(**true**);

con.setDoInput(**true**);

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

con.setRequestProperty("Accept", "application/json");

con.setRequestMethod("PUT");

// adding attributes of a sensor and creating a json object

JSONObject obj = **new** JSONObject();

obj.put("smokeLevel", sensor.getSmokeLevel());

obj.put("co2Level", sensor.getCO2Level());

obj.put("id", sensor.getSensorId());

obj.put("floorNo", sensor.getFloorNo());

obj.put("name", sensor.getSensorName());

obj.put("roomNo", sensor.getRoomNo());

obj.put("active", sensor.isActive());

OutputStreamWriter wr = **new** OutputStreamWriter(con.getOutputStream());

wr.write(obj.toString());

wr.flush();

StringBuilder sb = **new** StringBuilder();

**int** HttpResult = con.getResponseCode();

**if** (HttpResult == HttpURLConnection.***HTTP\_OK***) {

// if http response code for OK (200) returns from the rest api

// if the update operation is successful

BufferedReader br = **new** BufferedReader(**new** InputStreamReader(con.getInputStream(), "utf-8"));

String line = **null**;

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

sb.append(line + "\n");

}

br.close();

}

}

// method implementation to delete a sensor from mongodb

@Override

**public** **void** removeSensor(String sensorId) **throws** RemoteException, IOException {

String url = "http://localhost:5000/api/sensor/" + sensorId;

URL object = **new** URL(url);

// creating http delete request to send to rest api

HttpURLConnection con = (HttpURLConnection) object.openConnection();

con.setDoOutput(**true**);

con.setDoInput(**true**);

con.setRequestProperty("Accept", "application/json");

con.setRequestMethod("DELETE");

OutputStreamWriter wr = **new** OutputStreamWriter(con.getOutputStream());

wr.flush();

StringBuilder sb = **new** StringBuilder();

**int** HttpResult = con.getResponseCode();

**if** (HttpResult == HttpURLConnection.***HTTP\_OK***) {

// if http response code for OK (200) returns from the rest api

// if the delete operation is successful

BufferedReader br = **new** BufferedReader(**new** InputStreamReader(con.getInputStream(), "utf-8"));

String line = **null**;

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

sb.append(line + "\n");

}

br.close();

}

}

// method implementation to retrieve a sensor from mongodb

@Override

**public** Sensor getSensor(String sensorId) **throws** RemoteException, IOException {

String url = "http://localhost:5000/api/sensor/" + sensorId;

URL seatURL = **new** URL(url);

BufferedReader br = **new** BufferedReader(**new** InputStreamReader(seatURL.openStream(), Charset.*forName*("UTF-8")));

String readAPIResponse = " ";

StringBuilder jsonString = **new** StringBuilder();

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

jsonString.append(readAPIResponse);

}

// filtering the correct sensor instance from sensor id

// converting the retrieved json object data to correct format to get sensor

// details

JSONObject jsonObj = **new** JSONObject(jsonString.toString());

String obj = jsonObj.get(sensorId).toString();

JSONObject jsonObj2 = **new** JSONObject(obj);

// assigning the sensor details to a sensor object from the json object

Sensor s1 = **new** Sensor();

s1.setActive(Boolean.*parseBoolean*(jsonObj2.get("active").toString()));

s1.setCO2Level(Integer.*parseInt*(jsonObj2.get("co2Level").toString()));

s1.setFloorNo(Integer.*parseInt*(jsonObj2.get("floorNo").toString()));

s1.setRoomNo(Integer.*parseInt*(jsonObj2.get("roomNo").toString()));

s1.setSensorId(jsonObj2.get("id").toString());

s1.setSensorName(jsonObj2.get("name").toString());

s1.setSmokeLevel(Integer.*parseInt*(jsonObj2.get("smokeLevel").toString()));

// return the sensor object

**return** s1;

}

// method implementation to retrieve all sensors from mongodb

**public** **static** ArrayList<Sensor> getSensorsList() **throws** RemoteException, IOException {

String url = "http://localhost:5000/api/sensor/";

URL seatURL = **new** URL(url);

BufferedReader br = **new** BufferedReader(**new** InputStreamReader(seatURL.openStream(), Charset.*forName*("UTF-8")));

String readAPIResponse = " ";

StringBuilder jsonString = **new** StringBuilder();

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

jsonString.append(readAPIResponse);

}

// converting the the rest api response to a json array

JSONArray jsonObj = **new** JSONArray(jsonString.toString());

*sensors* = **new** ArrayList<Sensor>();

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

// retrieving sensors to json objects from json array

JSONObject jsonObj2 = (JSONObject) jsonObj.get(i);

// assigning sensor details to a sensor object from the json object

Sensor s1 = **new** Sensor();

s1.setActive(Boolean.*parseBoolean*(jsonObj2.get("active").toString()));

s1.setCO2Level(Integer.*parseInt*(jsonObj2.get("co2Level").toString()));

s1.setFloorNo(Integer.*parseInt*(jsonObj2.get("floorNo").toString()));

s1.setRoomNo(Integer.*parseInt*(jsonObj2.get("roomNo").toString()));

s1.setSensorId(jsonObj2.get("id").toString());

s1.setSensorName(jsonObj2.get("name").toString());

s1.setSmokeLevel(Integer.*parseInt*(jsonObj2.get("smokeLevel").toString()));

// adding each sensor object to an arraylist

*sensors*.add(s1);

}

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

// return all retrieved sensors in an arraylist

**return** *sensors*;

}

// method implementation to increment the number of rmi clients

@Override

**public** **synchronized** **int** increment() **throws** RemoteException {

count++;

**return** count;

}

// implementation of the main method

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

// including the allowall.policy file giving all permissions

System.*setProperty*("java.security.policy", "file:allowall.policy");

**try** {

// creating and starting the 1099 registry port and binding the created registry

Registry registry = LocateRegistry.*createRegistry*(Registry.***REGISTRY\_PORT***);

SensorServerRMI sensorServerRMI = **new** SensorServerRMI();

registry.rebind("rmi://localhost/server", sensorServerRMI);

System.***out***.println("Sensor server started...");

} **catch** (RemoteException remoteException) {

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

remoteException.printStackTrace();

} **catch** (Exception exception) {

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

exception.printStackTrace();

}

// creating a thread using Thread class and Runnable interface

Thread t = **new** Thread(**new** Runnable() {

@Override

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

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

**try** {

// checking the sensor status every 15 seconds to get the up to date readings

// from the rest api

Thread.*sleep*(15000);

*getSensorsList*();

} **catch** (RemoteException remoteException) {

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

remoteException.printStackTrace();

} **catch** (IOException iOException) {

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

iOException.printStackTrace();

} **catch** (InterruptedException interruptedException) {

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

interruptedException.printStackTrace();

} **catch** (Exception exception) {

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

exception.printStackTrace();

}

}

}

});

// starting the thread

t.start();

}

}

**SensorClientRMI.java**

**package** com.rmi;

**import** java.rmi.NotBoundException;

**import** java.rmi.RemoteException;

**import** java.rmi.registry.LocateRegistry;

**import** java.rmi.registry.Registry;

// rmi client class

**public** **class** SensorClientRMI {

**private** **int** clients = 0;

**public** SensorClientRMI() {

**super**();

**try** {

// accessing the 1099 registry port

Registry registry = LocateRegistry.*getRegistry*("localhost", Registry.***REGISTRY\_PORT***);

ISensorServerRMI iSensorServerRMI = (ISensorServerRMI) registry.lookup("rmi://localhost/server");

// incrementing the number of clients

clients = iSensorServerRMI.increment();

} **catch** (RemoteException remoteException) {

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

remoteException.printStackTrace();

} **catch** (NotBoundException notBoundException) {

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

notBoundException.printStackTrace();

} **catch** (Exception exception) {

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

exception.printStackTrace();

}

}

// method implementation to print the client number

**public** **void** displayClientNo() {

System.***out***.println("Clients: " + clients);

}

}

**Sensor.java**

**package** com.rmi;

// sensor model class

// includes getters, setters, constructors and attributes of a sensor

**public** **class** Sensor {

**private** String sensorId;

**private** String sensorName;

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

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

**private** **int** CO2Level;

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

**private** **boolean** isActive;

**public** Sensor() {

sensorId = "";

sensorName = "";

roomNo = 0;

floorNo = 0;

CO2Level = 0;

smokeLevel = 0;

setActive(**true**);

}

**public** Sensor(String sensorId, String sensorName, **int** roomNo, **int** floorNo, **int** CO2Level, **int** smokeLevel,

**boolean** isActive) {

**this**.sensorId = sensorId;

**this**.sensorName = sensorName;

**this**.roomNo = roomNo;

**this**.floorNo = floorNo;

**this**.CO2Level = CO2Level;

**this**.smokeLevel = smokeLevel;

**this**.setActive(isActive);

}

**public** Sensor(String sensorId, String sensorName, **int** roomNo, **int** floorNo, **boolean** isActive) {

**this**.sensorId = sensorId;

**this**.sensorName = sensorName;

**this**.roomNo = roomNo;

**this**.floorNo = floorNo;

**this**.setActive(isActive);

}

**public** String getSensorId() {

**return** sensorId;

}

**public** **void** setSensorId(String sensorId) {

**this**.sensorId = sensorId;

}

**public** String getSensorName() {

**return** sensorName;

}

**public** **void** setSensorName(String sensorName) {

**this**.sensorName = sensorName;

}

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

**return** roomNo;

}

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

**this**.roomNo = roomNo;

}

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

**return** floorNo;

}

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

**this**.floorNo = floorNo;

}

**public** **int** getCO2Level() {

**return** CO2Level;

}

**public** **void** setCO2Level(**int** cO2Level) {

CO2Level = cO2Level;

}

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

**return** smokeLevel;

}

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

**this**.smokeLevel = smokeLevel;

}

**public** **boolean** isActive() {

**return** isActive;

}

**public** **void** setActive(**boolean** isActive) {

**this**.isActive = isActive;

}

}

**allowall.policy**

grant {

permission java.security.AllPermission;

}

**Sensor Simulators (Sample JavaScript file)**

// dummy sensor simulator

// method implementation for generating a random number

generateRandomNumber = (min, max) => {

return Math.floor(Math.random() \* (max - min + 1) + min);

};

// update details method implementation

updateDetails = () => {

let co2Level;

let smokeLevel;

// generate random values from 1 to 10 for co2 level and smoke level

co2Level = generateRandomNumber(1, 10);

smokeLevel = generateRandomNumber(1, 10);

const sensor = {

co2Level,

smokeLevel,

};

// sending a put http request to update the values of co2 level and smoke level

$.ajax({

url: "http://localhost:5000/api/sensor/1",

type: "PUT",

contentType: "application/json",

dataType: "json",

data: JSON.stringify(sensor),

success: function (data) {

if (data) {

$(".sensorId").text(data.id);

$(".name").text(data.name);

$(".co2Level").text(data.co2Level);

$(".smokeLevel").text(data.smokeLevel);

$(".roomNo").text(data.roomNo);

$(".floorNo").text(data.floorNo);

}

},

});

};

// calling update details method

updateDetails();

// call update details method every 10 seconds

setInterval(updateDetails, 10000);

**Sensor Simulators (Sample HTML5 file)**

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8"/>

<meta content="width=device-width, initial-scale=1.0" name="viewport"/>

<title>Sensor Simulator</title>

<link crossorigin="anonymous"

href="https://stackpath.bootstrapcdn.com/bootstrap/4.4.1/css/bootstrap.min.css"

integrity="sha384-Vkoo8x4CGsO3+Hhxv8T/Q5PaXtkKtu6ug5TOeNV6gBiFeWPGFN9MuhOf23Q9Ifjh"

rel="stylesheet"

/>

<script crossorigin="anonymous"

integrity="sha256-r/AaFHrszJtwpe+tHyNi/XCfMxYpbsRg2Uqn0x3s2zc="

src="https://code.jquery.com/jquery-3.5.0.js"

></script>

</head>

<body>

<div class="container">

<div class="card-body align-content-center"

style="

border-color: black;

padding: 40px;

border-radius: 10px;

border-style: solid;

margin: 20px;

width: fit-content;

"

>

<span>Sensor ID &nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp; -- &nbsp; </span>

<span class="sensorId"></span><br/>

<span>Sensor Name &nbsp; -- &nbsp; </span>

<span class="name"></span><br/>

<span>Floor No &nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp; -- &nbsp; </span>

<span class="floorNo"></span><br/>

<span>Room No &nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp; -- &nbsp; </span>

<span class="roomNo"></span><br/>

<span>CO2 Level &nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp; -- &nbsp; </span>

<span class="co2Level"></span><br/>

<span>Smoke Level &nbsp;&nbsp; -- &nbsp; </span>

<span class="smokeLevel"></span>

</div>

</div>

<script src="2.js"></script>

</body>

</html>

**Web Client – index.html**

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="utf-8"/>

<link href="%PUBLIC\_URL%/favicon.ico" rel="icon"/>

<meta content="width=device-width, initial-scale=1" name="viewport"/>

<meta content="#000000" name="theme-color"/>

<meta

content="Web site created using create-react-app"

name="description"

/>

<link href="%PUBLIC\_URL%/logo192.png" rel="apple-touch-icon"/>

<!--

manifest.json provides metadata used when your web app is installed on a

user's mobile device or desktop. See https://developers.google.com/web/fundamentals/web-app-manifest/

-->

<link href="%PUBLIC\_URL%/manifest.json" rel="manifest"/>

<!--

Notice the use of %PUBLIC\_URL% in the tags above.

It will be replaced with the URL of the `public` folder during the build.

Only files inside the `public` folder can be referenced from the HTML.

Unlike "/favicon.ico" or "favicon.ico", "%PUBLIC\_URL%/favicon.ico" will

work correctly both with client-side routing and a non-root public URL.

Learn how to configure a non-root public URL by running `npm run build`.

-->

<title>Sensor Details</title>

</head>

<body>

<noscript>You need to enable JavaScript to run this app.</noscript>

<br>

<h1 style="text-align: center;">Sensor Details</h1>

<br>

<div id="root"></div>

<!--

This HTML file is a template.

If you open it directly in the browser, you will see an empty page.

You can add webfonts, meta tags, or analytics to this file.

The build step will place the bundled scripts into the <body> tag.

To begin the development, run `npm start` or `yarn start`.

To create a production bundle, use `npm run build` or `yarn build`.

-->

</body>

</html>

**Web Client – index.js**

import React from 'react';

import ReactDOM from 'react-dom';

import './index.css';

import App from './App';

import \* as serviceWorker from './serviceWorker';

ReactDOM.render(

<React.StrictMode>

<App/>

</React.StrictMode>,

document.getElementById('root')

);

// If you want your app to work offline and load faster, you can change

// unregister() to register() below. Note this comes with some pitfalls.

// Learn more about service workers: https://bit.ly/CRA-PWA

serviceWorker.unregister();

**Web Client – index.css**

body {

margin: 0;

font-family: -apple-system, BlinkMacSystemFont, 'Segoe UI', 'Roboto', 'Oxygen',

'Ubuntu', 'Cantarell', 'Fira Sans', 'Droid Sans', 'Helvetica Neue',

sans-serif;

-webkit-font-smoothing: antialiased;

-moz-osx-font-smoothing: grayscale;

}

code {

font-family: source-code-pro, Menlo, Monaco, Consolas, 'Courier New',

monospace;

}

**Web Client – App.css**

.App {

text-align: center;

}

.App-logo {

height: 40vmin;

pointer-events: none;

}

@media (prefers-reduced-motion: no-preference) {

.App-logo {

animation: App-logo-spin infinite 20s linear;

}

}

.App-header {

background-color: #282c34;

min-height: 100vh;

display: flex;

flex-direction: column;

align-items: center;

justify-content: center;

font-size: calc(10px + 2vmin);

color: white;

}

.App-link {

color: #61dafb;

}

@keyframes App-logo-spin {

from {

transform: rotate(0deg);

}

to {

transform: rotate(360deg);

}

}

**Web Client – App.js**

import React from "react";

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

import {BrowserRouter as Router} from "react-router-dom";

import SensorList from "./components/sensor-list-component";

function App() {

return (

<Router>

<div className="container">

<SensorList/>

</div>

</Router>

);

}

export default App;

**Web Client – package.json**

{

"name": "web-client",

"version": "0.1.0",

"private": true,

"dependencies": {

"@testing-library/jest-dom": "^4.2.4",

"@testing-library/react": "^9.5.0",

"@testing-library/user-event": "^7.2.1",

"axios": "^0.18.1",

"bootstrap": "^4.3.1",

"bootstrap-4-react": "0.0.59",

"cors": "^2.8.5",

"react": "^16.13.1",

"react-dom": "^16.13.1",

"react-router-dom": "^5.0.0",

"react-scripts": "3.4.1"

},

"scripts": {

"start": "react-scripts start",

"build": "react-scripts build",

"test": "react-scripts test",

"eject": "react-scripts eject"

},

"eslintConfig": {

"extends": "react-app"

},

"browserslist": {

"production": [

">0.2%",

"not dead",

"not op\_mini all"

],

"development": [

"last 1 chrome version",

"last 1 firefox version",

"last 1 safari version"

]

}

}

**Web Client – sensor-list-component.js**

import React, { Component } from "react";

import axios from "axios";

// sensor list component

class SensorList extends Component {

constructor(props) {

super(props);

this.state = {

sensors: [

{

id: "2",

name: "abc",

active: true,

floorNo: 3,

roomNo: 1,

smokeLevel: 4,

co2Level: 3,

},

{

id: "4",

name: "xyz",

active: false,

floorNo: 1,

roomNo: 2,

smokeLevel: 3,

co2Level: 2,

},

],

};

}

// get sensor list from mongodb via the express api

getSensorList() {

axios

.get("http://localhost:5000/api/sensor")

.then((response) => {

this.setState({ sensors: [...response.data] });

console.log("response");

})

.catch((err) => {

console.log(`error is ${err}`);

});

}

// refresh table every 40 seconds

componentDidMount() {

this.interval = setInterval(() => this.getSensorList(), 40000);

}

componentWillUnmount() {

clearInterval(this.interval);

}

// display the table and values of each sensor

// if co2 level or smoke level of a sensor is greater than 5, color that row in red

render() {

return (

<div className="container" style={{ marginTop: "50px" }}>

<h1 style={{ textAlign: "center", padding: "15px", color: "#343a40" }}>

Sensor Details

</h1>

<table className="table table-bordered">

<thead className="thead-dark">

<tr style={{ textAlign: "center" }}>

<th>Sensor ID</th>

<th>Sensor Name</th>

<th>Active Status</th>

<th>Floor Number</th>

<th>Room Number</th>

<th>Smoke Level (1-10)</th>

<th>CO2 Level (1-10)</th>

</tr>

</thead>

<tbody>

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

return sensor.smokeLevel > 5 || sensor.co2Level > 5 ? (

<tr

bgcolor="#FF0000"

style={{ color: "white" }}

key={sensor.id}

>

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

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

<td>{sensor.active ? "Active" : "Not Active"}</td>

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

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

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

<td>{sensor.co2Level}</td>

</tr>

) : (

<tr bgcolor="White" key={sensor.id}>

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

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

<td>{sensor.active ? "Active" : "Not Active"}</td>

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

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

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

<td>{sensor.co2Level}</td>

</tr>

);

})}

</tbody>

</table>

</div>

);

}

}

export default SensorList;

**REST API – app.js**

const express = require("express");

const mongoose = require("mongoose");

const bodyParser = require("body-parser");

const cors = require("cors");

require("dotenv").config();

const app = express();

app.use(bodyParser.urlencoded({extended: true}));

app.use(bodyParser.json());

app.use(cors());

// defining the route

app.use("/api", require("./routes/sensor-routes"));

app.use(function (err, req, res, next) {

console.log(err);

res.status(422).send({error: err.message});

});

// getting the mongodb atlas uri from the .env file

const uri = process.env.MONGO\_ATLAS;

const options = {

useNewUrlParser: true,

useUnifiedTopology: true,

useCreateIndex: true

};

// connecting to mongodb with mongoose

mongoose

.connect(uri, options)

.then(() => {

app.listen(5000);

})

.catch((error) => {

console.log(error);

});

**REST API – package.json**

{

"name": "sliit-y3s1-ds-rest-api",

"version": "1.0.0",

"description": "sliit-y3s1-ds-rest-api",

"main": "index.js",

"scripts": {

"test": "echo \"Error: no test specified\" && exit 1",

"start": "node app.js"

},

"repository": {

"type": "git",

"url": "git://github.com/TharindaNimnajith/SLIIT\_Y3S1\_DS\_SensorApp.git"

},

"author": "Ayesh\_Lakshan",

"license": "ISC",

"dependencies": {

"body-parser": "^1.19.0",

"cors": "^2.8.5",

"dotenv": "^8.2.0",

"express": "^4.17.1",

"mongoose": "^5.9.11",

"node": "^14.0.0",

"nodejs": "0.0.0",

"nodemailer": "^6.4.6"

}

}

**REST API – sensor-model.js**

const mongoose = require("mongoose");

const Schema = mongoose.Schema;

// defining a mongoose schema for a sensor model with necessary attributes

const sensorSchema = new Schema({

id: {type: String, required: true, unique: true},

name: {type: String, required: true},

floorNo: {type: Number, required: true},

roomNo: {type: Number, required: true},

active: {type: Boolean, required: false, default: true},

smokeLevel: {type: Number, required: true, default: 0},

co2Level: {type: Number, required: true, default: 0}

});

module.exports = mongoose.model("Sensors", sensorSchema);

**REST API – sensor-routes.js**

const express = require("express");

const router = express.Router();

const Sensor = require("../models/sensor-model");

const nodemailer = require("nodemailer");

require("dotenv").config();

// sending emails with nodemailer package

let transporter = nodemailer.createTransport({

service: "gmail",

auth: {

user: "ayeshlak1998@gmail.com",

pass: process.env.PASSWORD,

},

});

// adding a new sensor

router.post("/sensor", (req, res, next) => {

Sensor.create(req.body)

.then((sensor) => {

res.send(sensor);

})

.catch(next);

});

// updating a sensor by sensor id

router.put("/sensor/:id", (req, res, next) => {

Sensor.findOneAndUpdate(

{id: req.params.id},

{$set: req.body},

{new: true},

(error, doc) => {

// sending emails and sms messages according to the smoke level and co2 level of each sensor

if (doc.co2Level > 5 && doc.smokeLevel > 5) {

let info = {

from: '"Sensor Alerts 💥🔥" ayeshlak1998@gmail.com',

to: "aruniprashani@gmail.com",

subject: "SmokeLevel and CO2 level Increased",

text: `SmokeLevel of the sensor ${doc.id} increased to ${doc.smokeLevel} and CO2 Level of the sensor ${doc.id} increased to ${doc.co2Level}`,

};

transporter.sendMail(info, (err, data) => {

if (err) {

console.log(err);

} else {

console.log("sent");

}

});

console.log("message has been sent to 077465521");

console.log(doc.co2Level);

} else if (doc.co2Level > 5) {

let info = {

from: '"Sensor Alerts 💥🔥" ayeshlak1998@gmail.com',

to: "aruniprashani@gmail.com",

subject: "CO2 Level Increased",

text: `CO2 Level of the sensor ${doc.id} increased to ${doc.co2Level}`,

};

transporter.sendMail(info, (err, data) => {

if (err) {

console.log(err);

} else {

console.log("sent");

}

});

console.log("message has been sent to 077465521");

console.log(doc.co2Level);

} else if (doc.smokeLevel > 5) {

let info = {

from: '"Sensor Alerts 💥🔥" ayeshlak1998@gmail.com',

to: "aruniprashani@gmail.com",

subject: "SmokeLevel Increased",

text: `SmokeLevel of the sensor ${doc.id} increased to ${doc.smokeLevel}`,

};

transporter.sendMail(info, (err, data) => {

if (err) {

console.log(err);

} else {

console.log("sent");

}

});

console.log("message has been sent to 077465521");

console.log(doc.smokeLevel);

}

res.send(doc);

}

);

});

// retrieving all sensors

router.get("/sensor", (req, res, next) => {

Sensor.find({}, (err, sensors) => {

res.send(sensors);

}).catch(next);

});

// retrieving all sensors

router.get("/sensors", (req, res, next) => {

Sensor.find({}, (err, sensors) => {

var sensorMap = {};

sensors.forEach((sensor) => {

sensorMap[sensor.id] = sensor;

});

res.send(sensorMap);

}).catch(next);

});

// retrieving a sensor by sensor id

router.get("/sensor/:id", (req, res, next) => {

Sensor.find({id: req.params.id}, (err, sensors) => {

var sensorMap = {};

sensors.forEach((sensor) => {

sensorMap[sensor.id] = sensor;

});

res.send(sensorMap);

}).catch(next);

});

// deleting a sensor by sensor id

router.delete("/sensor/:id", (req, res, next) => {

Sensor.deleteOne({id: req.params.id}, (err, result) => {

if (result.deletedCount) {

res.json({

message: `deleted ${req.params.id}`,

});

} else {

res.json({

message: `deleted failed ${req.params.id}`,

});

}

}).catch(next);

});

module.exports = router;