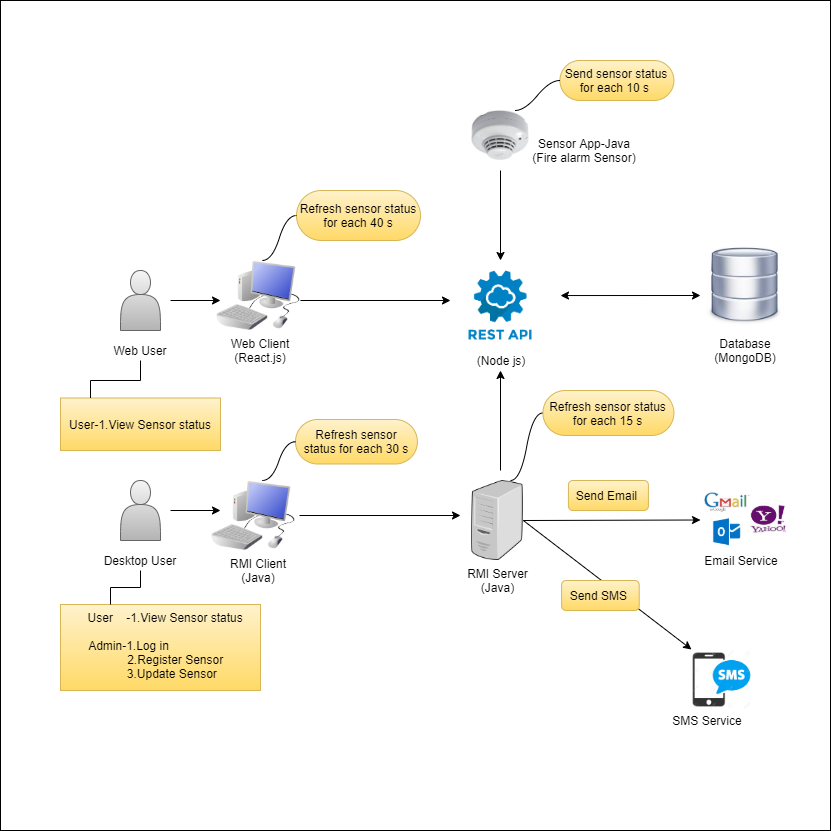
Introduction

This is a fire alarm monitoring system, which is implemented using REST API, RMI Client, RMI Server, Web Client and Sensor application. Java was used to implement RMI Client, RMI Server and Sensor application. React js was used to implement the web application. Node js was used to implement REST API. MongoDB was used as the database in the implemented system.

High Level Architecture Diagram



Services in the system

Desktop Application

* Users can view status of all fire alarm sensors. They are, floor no, room no, active state, smoke level, CO2 level. The sensors in which CO2 level > 5 or smoke level > 5, are marked in red.
* In desktop application, the status of all fire alarm sensors is refreshed in every 30 seconds. Then, if CO2 level > 5 or smoke level > 5 in a sensor, an alert message will be displayed.
* Log in by Admin – Admin can login giving ‘Admin’ as both the username and the password.
* Register new sensor by Admin – Admin can register a new sensor entering a floor no and a room no.
* Edit and Update sensor details by Admin – Admin can edit and update the sensor details like floor no and room no.

RMI Server

* The RMI server is refreshed on sensor status of all fire alarm sensors, checking from the API in every 15 seconds. Then, if the CO2 level > 5 or smoke level > 5 in a sensor, an email and an SMS will be sent regarding that.

Web Application

* Web users can view status of all fire alarm sensors. They are, floor no, room no, active state, smoke level, co2 level. The sensors in which CO2 level > 5 or smoke level > 5, are marked in red
* In web application, the status of all fire alarm sensors is refreshed in every 40 seconds

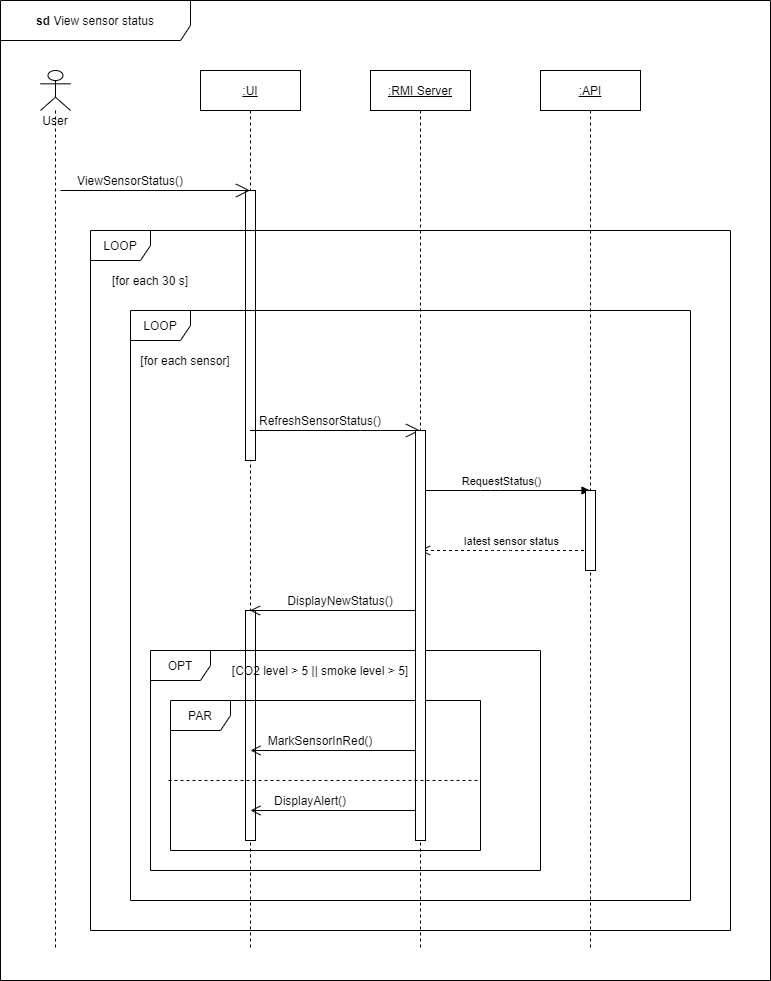
Sensor Application

* Sensor application behave as a sensor. By running multiple applications, they send their newest sensor status to the API, as sensors do. This is done in every 10 seconds

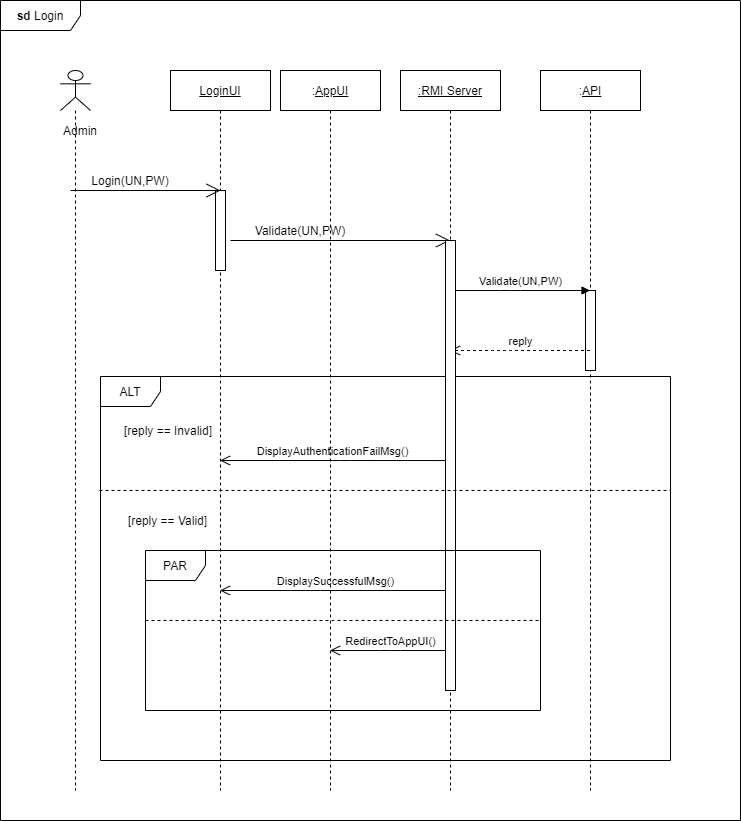
These services are described using sequence diagrams in this report.

Desktop Application

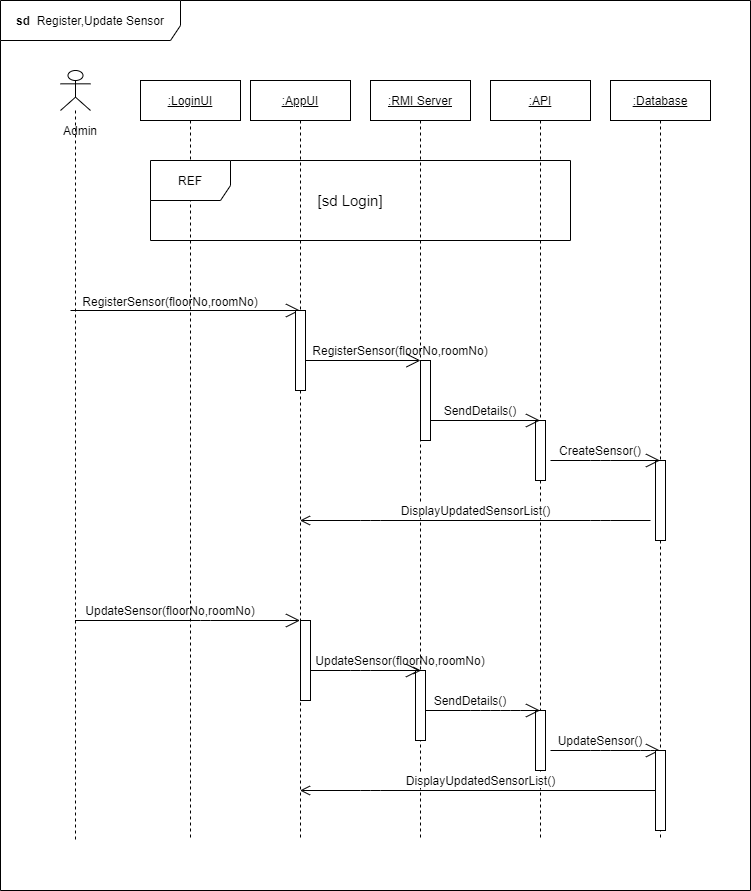
**View sensor status which is refreshed every 30 seconds**



**Log in by Admin**

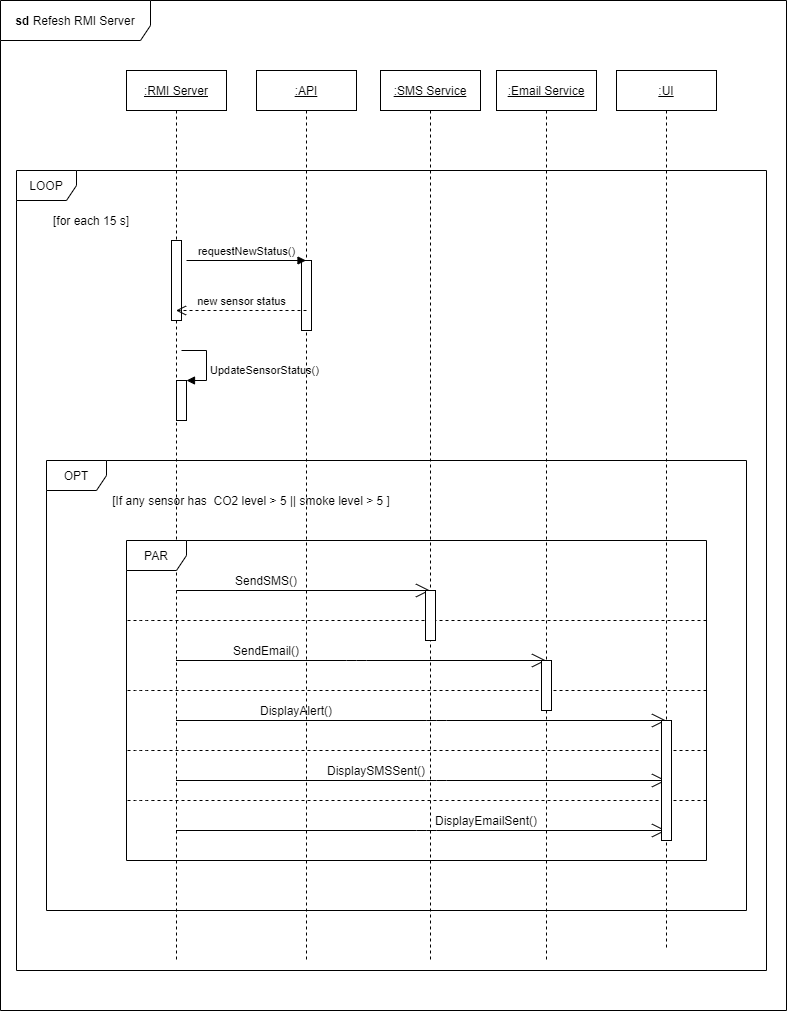


**Register new sensor, edit & update sensor details by Admin**



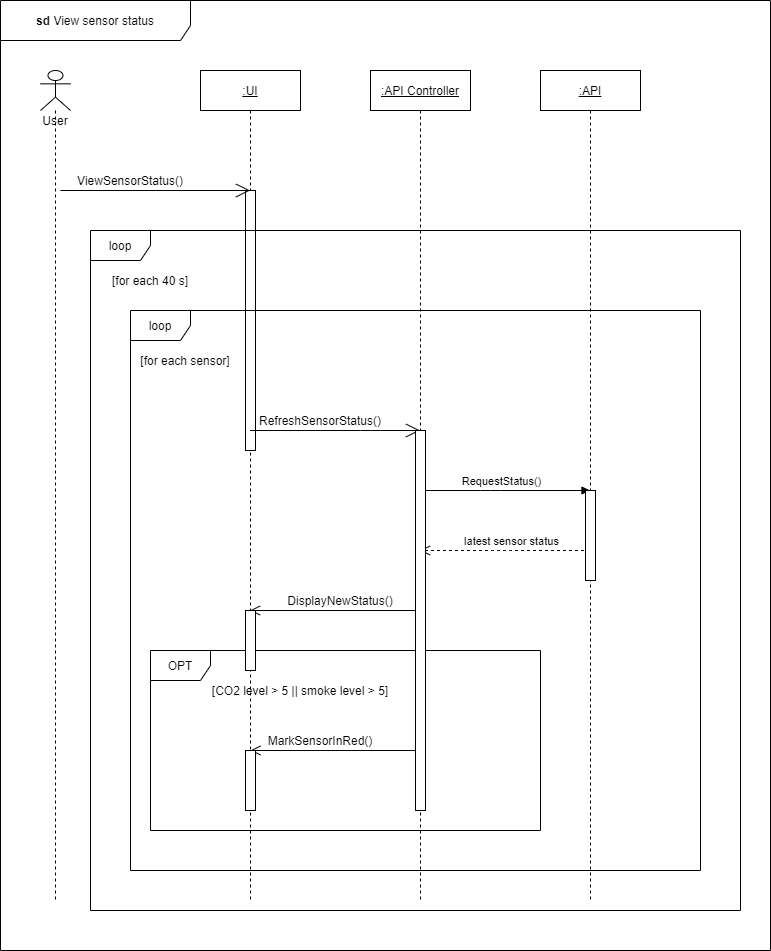
RMI Server

**Auto update sensor status in every 15 seconds**



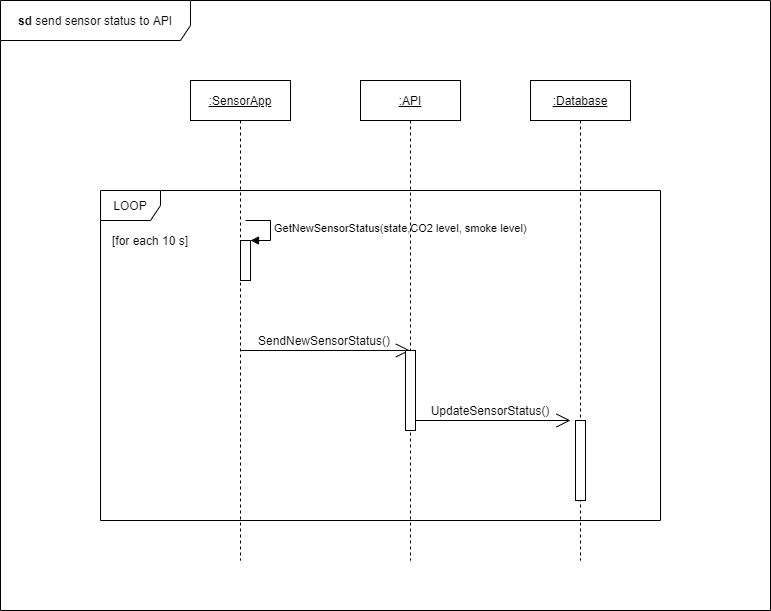
Web Application

**View details which are refreshed every 40 seconds**

****

Sensor Application

**Send sensor status to the API every 10 seconds**



Appendix

Desktop Application

**SensorModel.java**

package models;

import java.io.Serializable;

public class SensorModel implements Serializable{

//this class is to store a sensor which is taken by the API

//sensor attributes

private Object \_id;

private boolean isActive;

private String floorNo;

private String roomNo;

private int smokeLevel;

private int CO2Level;

//SensorModel getters and setters

public Object get\_id() {

return \_id;

}

public void set\_id(Object \_id) {

this.\_id = \_id;

}

public boolean isActive() {

return isActive;

}

public void setActive(boolean isActive) {

this.isActive = isActive;

}

public String getFloorNo() {

return floorNo;

}

public void setFloorNo(String floorNo) {

this.floorNo = floorNo;

}

public String getRoomNo() {

return roomNo;

}

public void setRoomNo(String roomNo) {

this.roomNo = roomNo;

}

public int getSmokeLevel() {

return smokeLevel;

}

public void setSmokeLevel(int smokeLevel) {

this.smokeLevel = smokeLevel;

}

public int getCO2Level() {

return CO2Level;

}

public void setCO2Level(int cO2Level) {

CO2Level = cO2Level;

}

}

**NotificationListner.java**

package models;

import java.util.ArrayList;

//this interaface is responsible to send the notification by the client and RMI server

public interface NotificationListner

{

//inimplemented method with the list of SensorModel

public void notifyWarning(ArrayList<SensorModel> item);

}

**APIService.java**

package rmi.api;

import java.io.BufferedReader;

import java.io.InputStreamReader;

import java.net.HttpURLConnection;

import java.net.MalformedURLException;

import java.net.URL;

import java.rmi.RemoteException;

import java.util.ArrayList;

import org.json.JSONArray;

import org.json.JSONObject;

import models.SensorModel;

//ApiService class implements the ApiServiceInterface

public class ApiService implements ApiServiceInterface {

//overriden methods of the ApiServiceInterface

//this method is to get all the sensors from the api by sending a request

//and returning a list of sensors

@Override

public ArrayList<SensorModel> getAllSensors() throws Exception

{

//url to send the request

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

//creating an URL object with url we created

URL obj = new URL(url);

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

//setting the reuest type to GET

con.setRequestMethod("GET");

//catching the response code

int respondCode = con.getResponseCode();

//reading the response

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

String inputLine;

StringBuffer response = new StringBuffer();

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

{

response.append(inputLine);

}

in.close();

//the response string converts to an JSONArray

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

//create a list of SensorModel

ArrayList<SensorModel> list = new ArrayList<SensorModel>();

//running a for loop to create to SensorModel list with JSONArray

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

{

//getting the sensors from the JSONArray and created new SensorModel object

JSONObject jObject = array.getJSONObject(i);

SensorModel model = new SensorModel();

model.set\_id(jObject.get("\_id"));

model.setActive(jObject.getBoolean("isActive"));

model.setFloorNo(jObject.getString("floorNo"));

model.setRoomNo(jObject.getString("roomNo"));

model.setSmokeLevel(jObject.getInt("smokeLevel"));

model.setCO2Level(jObject.getInt("CO2Level"));

//adding it to the list

list.add(model);

}

//returning finally added list

return list;

}

//this is an overriden method from the ApiServiceInterface

//this method will be called to create a new sensor

@Override

public boolean createNewSensor(boolean isActive, String floorNo, String roomNo, int smokeLevel, int co2Level)

throws Exception {

//setting up the url to send to HTTP request

String burl = "http://localhost:5000/api/sensors/createNewSensor/?isActive="+isActive+"&floorNo="+floorNo+"&roomNo="+roomNo+"&smokeLevel="+smokeLevel+"&CO2Level="+ co2Level;

//removing spaces of the url by replacing "%20"

String url = burl.replaceAll(" ","%20");

//creating an URL object with the url we created

URL obj = new URL(url);

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

//setting up the request type to POST

con.setRequestMethod("POST");

//catching the response code

int respondCode = con.getResponseCode();

System.out.println("POST" + respondCode);

//checking the response code to turn a boolean to make sure the sensor is created or not

if(respondCode == 200)

{

return true;

}

return false;

}

//this method is overriden by the ApiServiceInterface

//this method will be called to check the login with usersname and password

@Override

public boolean checkLogin(String username, String password) throws Exception {

//setting up the url

String burl = "http://localhost:5000/api/users/login?username=" + username + "&password=" + password;

//replacing the spaces

String url = burl.replaceAll(" ","%20");

//creating the URL object with url we created

URL obj = new URL(url);

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

//setting the request type

con.setRequestMethod("GET");

//holding the response code

int respondCode = con.getResponseCode();

//reading the response

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

String inputLine;

StringBuffer response = new StringBuffer();

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

{

response.append(inputLine);

}

in.close();

//converting response to an JSONArray

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

//check whether the login is success or not to return

if(respondCode == 200 && array.length() > 0)

{

return true;

}

return false;

}

//this method is responsible to register an user

@Override

public boolean RegisterUser(String floorNo, String roomNo) throws Exception {

//setting up the url with parameters

String burl = "http://localhost:5000/api/users/register?floorNo=" + floorNo + "&roomNo=" + roomNo;

//replacing the spaces url conatins

String url = burl.replaceAll(" ","%20");

//creating an URL object

URL obj = new URL(url);

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

//setting the request type to POST

con.setRequestMethod("POST");

//getting the response code

int respondCode = con.getResponseCode();

//check whether the user successfully registered or not with response code

//if 200 retured its success

if(respondCode == 200)

{

return true;

}

//retuning

return false;

}

//this method will be called to update a sensor

@Override

public boolean updateSensor(String id, String floorNo, String roomNo) throws Exception

{

//setting up the url

String burl = "http://localhost:5000/api/sensors/updateSensor?id=" + id + "&floorNo=" + floorNo + "&roomNo=" + roomNo;

//removing spaces

String url = burl.replaceAll(" ","%20");

URL obj = new URL(url);

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

//setting the request type to POST

con.setRequestMethod("POST");

//get the response code

int respondCode = con.getResponseCode();

//checking whether the respondCode is 200 to return the update state

if(respondCode == 200)

{

return true;

}

return false;

}

}

**ApiServiceInterface.java**

package rmi.api;

import java.rmi.Remote;

import java.rmi.RemoteException;

import java.util.ArrayList;

import models.SensorModel;

public interface ApiServiceInterface extends Remote {

public ArrayList<SensorModel> getAllSensors() throws Exception;

public boolean createNewSensor(boolean isActive,String floorNo,String roomNo,int smokeLevel,int co2Level) throws Exception;

public boolean checkLogin(String username,String password) throws Exception;

public boolean RegisterUser(String floorNo,String roomNo) throws Exception;

public boolean updateSensor(String id,String floorNo,String roomNo) throws Exception;

}

**RmiServer.java**

package rmi.server;

import java.rmi.RemoteException;

import java.rmi.registry.LocateRegistry;

import java.rmi.registry.Registry;

import java.rmi.server.UnicastRemoteObject;

import java.util.ArrayList;

import java.util.Timer;

import java.util.TimerTask;

import javafx.application.Platform;

import models.NotificationListner;

import models.SensorModel;

import rmi.api.ApiService;

import rmi.api.ApiServiceInterface;

public class RmiServer

{

//notification listner interface

private static NotificationListner listner;

//server running port number

private static int port;

//constructor with the NotificationListner and port

public RmiServer(int port,NotificationListner listner)

{

//storing the paramaters

this.listner = listner;

this.port = port;

}

public static void main(String[] args) throws RemoteException

{

//creating a registry

Registry registry = LocateRegistry.createRegistry(port);

//creating ApiService class object

ApiService apiService = new ApiService();

ApiServiceInterface apiServiceInterface = (ApiServiceInterface) UnicastRemoteObject.exportObject(apiService,0);

//binding the object

registry.rebind("RmiServer", apiServiceInterface);

System.out.println("Rmi Server is running!");

//this timer will call its run method once in every 15 seconds

int timePeriod = 15000;

Platform.runLater(() ->

{

Timer timer = new Timer();

timer.schedule(new TimerTask() {

@Override

public void run()

{

try

{

//get sensors from the api

ArrayList<SensorModel> list = apiService.getAllSensors();

//waring list holds the imformation of the sensors which is either co2 level or smoke level is

//greater than to 5

ArrayList<SensorModel> warningList = new ArrayList<>();

for(int i = 0; i < list.size(); i++)

{

//adding to the warning list

if(list.get(i).getCO2Level() > 5 || list.get(i).getSmokeLevel() > 5)

{

warningList.add(list.get(i));

}

if(i == list.size() - 1)

{

//calling the notifyWarning with warningList to show the notification

if(warningList.size() > 0)

{

listner.notifyWarning(warningList);

}

}

}

}

catch (Exception e)

{

e.printStackTrace();

}

}

}, 0, timePeriod);

});

}

}

**Client.java**

package rmi.client;

import java.net.ServerSocket;

import java.net.URL;

import java.rmi.RemoteException;

import java.rmi.registry.LocateRegistry;

import java.rmi.registry.Registry;

import java.util.ArrayList;

import java.util.ResourceBundle;

import java.util.Timer;

import java.util.TimerTask;

import org.controlsfx.control.Notifications;

import javafx.application.Application;

import javafx.application.Platform;

import javafx.collections.FXCollections;

import javafx.collections.ObservableList;

import javafx.event.ActionEvent;

import javafx.event.EventHandler;

import javafx.stage.Stage;

import models.NotificationListner;

import models.SensorModel;

import rmi.api.ApiServiceInterface;

import rmi.server.RmiServer;

import javafx.scene.Parent;

import javafx.scene.Scene;

import javafx.scene.control.Alert;

import javafx.scene.control.Alert.AlertType;

import javafx.scene.control.Button;

import javafx.scene.control.Label;

import javafx.scene.control.ListView;

import javafx.scene.control.TextField;

import javafx.scene.input.MouseEvent;

import javafx.scene.layout.Pane;

import javafx.fxml.FXML;

import javafx.fxml.FXMLLoader;

import javafx.fxml.Initializable;

//this class implements NotificationListner

public class Client extends Application implements Initializable,NotificationListner

{

//all the UI elements

@FXML

private TextField username;

@FXML

private TextField password;

@FXML

private Label response;

@FXML

private Pane loginPane;

@FXML

private Pane loggedOnPane;

@FXML

private ListView sensorListView;

@FXML

private TextField floorNo;

@FXML

private TextField roomNo;

@FXML

private Label addNewOrUpdateSensorLabel;

@FXML

private Pane addNewSensorPane;

@FXML

private Label createSensorResponse;

@FXML

private Button createOrUpdateSensorButton;

@FXML

private Pane sensorFullDetailsPane;

@FXML

private Label sensorID;

@FXML

private Label sensorStatus;

@FXML

private Label sensorFloorNumber;

@FXML

private Label sensorRoomNumber;

@FXML

private Label sensorCo2Level;

@FXML

private Label sensorSmokeLevel;

@FXML

private Button addNewSensorButton;

@FXML

private Button logoutButton;

@FXML

private Button updateSensorButton;

//attribues

//ApiServiceInterface object

private ApiServiceInterface apiServiceInterface;

//listview ObservableList

private ObservableList<String> list = FXCollections.observableArrayList();

//sensor list

private ArrayList<SensorModel> sensList;

//updating sensor id

private String updatingSensorId = null;

//updating sensor floor number

private String updatingSensorFloorNumber = null;

//updating sensor room number

private String updatingSensorRoomNumber = null;

private boolean isCreating;

//last clicked position of the list

private int lastListClickedPosition = -99;

//is the admin is logged in or not

private boolean isLoggedIn = false;

@Override

public void initialize(URL url,ResourceBundle rb)

{

try {

//getting a free port to run the RMI server

ServerSocket serverSocket = new ServerSocket(0);

int port = serverSocket.getLocalPort();

serverSocket.close();

//create server object with the port number and the notification lister

RmiServer server = new RmiServer(port,this);

//run the main method

server.main(null);

//accessing the rmi server

Registry registry;

registry = LocateRegistry.getRegistry("localhost",port);

//get the apiServiceInterface object

apiServiceInterface = (ApiServiceInterface) registry.lookup("RmiServer");

}

catch (Exception e1)

{

e1.printStackTrace();

}

//making the login screen visible at the very first time

loginPane.setVisible(true);

sensorListView.setItems(list);

refreshingList();

//on item click listner of sensor listview

sensorListView.setOnMouseClicked(new EventHandler<MouseEvent>() {

@Override

public void handle(MouseEvent event)

{

//storing the last clicked position to show the full detailed view of the sensor

lastListClickedPosition = sensorListView.getSelectionModel().getSelectedIndex();

//setting the full detailed view of the currently clicked position

setupItemUI(lastListClickedPosition);

}

});

}

@Override

public void start(Stage primaryStage) {

try {

//showing the login UI

setLoginUI(primaryStage);

setLoggedOnUI(true);

} catch(Exception e) {

e.printStackTrace();

}

}

public void setSensorList()

{

//setting up the sensor list

Platform.runLater(() ->

{

//clearing the list to eleminate duplicates

list.clear();

try {

//getting all the sensors with the apiServiceInterface object

sensList = apiServiceInterface.getAllSensors();

//running a for each loop to create the listview items

for(SensorModel model : sensList)

{

//adding to the observableArrayList

list.add("Floor : " + model.getFloorNo() + " Room : " + model.getRoomNo());

}

//check whether the last position item is not out of bound of the array to handle runtime exceptions

if(lastListClickedPosition >=0 && lastListClickedPosition < sensList.size())

{

//setting up full detailed view of a sensor

setupItemUI(lastListClickedPosition);

}

} catch (Exception e) {

// TODO Auto-generated catch block

e.printStackTrace();

}

});

}

//setting the login UI

private void setLoginUI(Stage primaryStage) throws Exception

{

Parent root = FXMLLoader.load(getClass().getResource("UI.fxml"));

Scene scene = new Scene(root,500,500);

primaryStage.setScene(scene);

primaryStage.show();

}

//setting the UI after the admin logged into the system

private void setLoggedOnUI(boolean isLogged)

{

try

{

if(isLogged)

{

loginPane.setVisible(false);

addNewSensorPane.setVisible(false);

loggedOnPane.setVisible(true);

isLoggedIn = true;

addNewSensorButton.setVisible(true);

logoutButton.setText("Logout");

updateSensorButton.setVisible(true);

}

else

{

loginPane.setVisible(false);

addNewSensorPane.setVisible(false);

loggedOnPane.setVisible(true);

isLoggedIn = false;

addNewSensorButton.setVisible(false);

logoutButton.setText("Login");

updateSensorButton.setVisible(false);

}

}

catch(Exception e)

{

}

}

public static void main(String[] args) {

launch(args);

}

//this method will be called when the login button is clicked

public void Login(ActionEvent event)

{

try {

//check whether the username and password is valid

if((username.getText().equals("") || username.getText() == null) || password.getText().equals("") || password.getText() == null)

{

//if not valid show the error

response.setText("Enter Something Valid!");

return;

}

//if there's something entered as the username and password.password

//check the username and password is valid or not

boolean isLoginSuccess = apiServiceInterface.checkLogin(username.getText(),password.getText());

//if isLoginSuccess is true. which means the entered username and password is valid

if(isLoginSuccess)

{

//setting the response in the UI

response.setText("Login Success!");

setLoggedOnUI(true);

}

else

{

//setting the response in the UI

response.setText("Authentication failed!");

}

} catch (Exception e) {

e.printStackTrace();

}

}

//setting the full detailed view of a sensor

public void setupItemUI(int position)

{

Platform.runLater(() ->

{

//showing the sensorFullDetailsPane

sensorFullDetailsPane.setVisible(true);

//setting up css for the pane and textfields based on the co2 level and smoke level

//if the smoke level or co2 level is greater than 5

//the background will be shown as red

//else as white

if(sensList.get(position).getCO2Level() > 5 || sensList.get(position).getSmokeLevel() > 5)

{

sensorFullDetailsPane.setStyle("-fx-background-color: #FF0000");

sensorFloorNumber.setStyle("-fx-text-fill: white");

sensorRoomNumber.setStyle("-fx-text-fill: white");

sensorCo2Level.setStyle("-fx-text-fill: white");

sensorSmokeLevel.setStyle("-fx-text-fill: white");

sensorStatus.setStyle("-fx-text-fill: white");

sensorID.setStyle("-fx-text-fill: white");

}

else

{

//setting up css for the pane and textfields based on the co2 level and smoke level

sensorFullDetailsPane.setStyle("-fx-background-color: #FFFFFF");

sensorFloorNumber.setStyle("-fx-text-fill: black");

sensorRoomNumber.setStyle("-fx-text-fill: black");

sensorCo2Level.setStyle("-fx-text-fill: black");

sensorSmokeLevel.setStyle("-fx-text-fill: black");

sensorStatus.setStyle("-fx-text-fill: black");

sensorID.setStyle("-fx-text-fill: black");

}

//checking the active state of the sensor

if(sensList.get(position).isActive())

{

sensorStatus.setText("Sensor Status : Active");

}

else

{

sensorStatus.setText("Sensor Status : Inactive");

}

//setting the UI with the currently clicked sensor details

sensorID.setText("Sensor ID : " + sensList.get(position).get\_id());

sensorFloorNumber.setText("Floor Number : " + sensList.get(position).getFloorNo());

sensorRoomNumber.setText("Room Number : " + sensList.get(position).getRoomNo());

sensorCo2Level.setText("CO2 Level : " + sensList.get(position).getCO2Level());

sensorSmokeLevel.setText("Smoke Level : " + sensList.get(position).getSmokeLevel());

updatingSensorId = sensList.get(position).get\_id().toString();

updatingSensorFloorNumber = sensList.get(position).getFloorNo();

updatingSensorRoomNumber = sensList.get(position).getRoomNo();

});

}

//this method will be called when adding a new sensor or updating sensor

public void AddNewSensor(ActionEvent event)

{

//setting the UI for updating or creating

isCreating = true;

loggedOnPane.setVisible(false);

addNewSensorPane.setVisible(true);

addNewOrUpdateSensorLabel.setText("Add New Sensor");

createOrUpdateSensorButton.setText("Create Sensor");

floorNo.setText("");

roomNo.setText("");

}

public void skipLogin(ActionEvent event)

{

//skiping the login

setLoggedOnUI(false);

}

//this method will be called when an sensor is about to update

public void UpdateSensor(ActionEvent event) throws Exception

{

//setting up the UI

isCreating = false;

loggedOnPane.setVisible(false);

addNewSensorPane.setVisible(true);

addNewOrUpdateSensorLabel.setText("Update Existing Sensor");

createOrUpdateSensorButton.setText("Update Sensor");

floorNo.setText(updatingSensorFloorNumber);

roomNo.setText(updatingSensorRoomNumber);

}

//this method will be called when the CreateNewSensor button is clicked

public void CreateNewSensor(ActionEvent event) throws Exception

{

createOrUpdateSensor(isCreating);

}

//updating or creating a sensor method

public void createOrUpdateSensor(boolean isCreateNew) throws Exception

{

//check whether the everything is okay to proceed

if(floorNo.getText().equals("") || floorNo.getText() == null && roomNo.getText().equals("") || roomNo.getText() == null)

{

createSensorResponse.setText("Enter valid details");

return;

}

//if a new sensor is creating

if(isCreateNew)

{

//calling createNewSensor method to create a sensor with the floor number and room number

boolean isAdded = apiServiceInterface.createNewSensor(true, floorNo.getText(), roomNo.getText(), 0, 0);

//check whether the sensor is properly created or not to show the response in the UI

if(isAdded)

{

floorNo.setText("");

roomNo.setText("");

createSensorResponse.setText("Sensor was created!");

updateUIAfterAddingOrUpdatingASensor();

}

else

{

createSensorResponse.setText("Something went wrong!");

}

}

//or a sensor is updating

else

{

//calling the update sensor method

//with the id,floor number and room number

boolean isUpdated = apiServiceInterface.updateSensor(updatingSensorId, floorNo.getText(), roomNo.getText());

//check whether it is properly updated or not to show in the UI

if(isUpdated)

{

createSensorResponse.setText("Sensor was updated!");

updateUIAfterAddingOrUpdatingASensor();

}

else

{

createSensorResponse.setText("Something went wrong!");

}

}

}

//this method will be called when the go back button is clicked

public void goBackToLoggedOnScreen(ActionEvent event)

{

loggedOnPane.setVisible(true);

addNewSensorPane.setVisible(false);

updatingSensorId = null;

updatingSensorFloorNumber = null;

updatingSensorRoomNumber = null;

sensorFullDetailsPane.setVisible(false);

}

//this method will be called when the logout button is clicked

public void goBackToLoginScreen(ActionEvent event)

{

response.setText("");

username.setText("");

password.setText("");

loginPane.setVisible(true);

addNewSensorPane.setVisible(false);

loggedOnPane.setVisible(false);

isLoggedIn = false;

}

//updating ui after a sensor is created or updated

public void updateUIAfterAddingOrUpdatingASensor()

{

setSensorList();

loggedOnPane.setVisible(true);

addNewSensorPane.setVisible(false);

createSensorResponse.setText("");

updatingSensorId = null;

updatingSensorFloorNumber = null;

updatingSensorRoomNumber = null;

sensorFullDetailsPane.setVisible(false);

}

//updating the sensorList every 30 seconds

public void refreshingList()

{

//updating sensor list every 30 seconds

int timePeriod = 30000;

Platform.runLater(() ->

{

Timer timer = new Timer();

timer.schedule(new TimerTask() {

@Override

public void run()

{

//setting up the list

setSensorList();

if(isLoggedIn)

{

String body = "";

//showing the warning notification if any of sensor's smoke level or co2 level is graeter than 5

for(int i = 0; i < sensList.size(); i++)

{

//setting up the notification body

if(sensList.get(i).getCO2Level() > 5 || sensList.get(i).getSmokeLevel() > 5)

{

body = body + "Floor no " + sensList.get(i).getFloorNo() + ", Room no " + sensList.get(i).getRoomNo() + " sensor shows an unusual behaviour. Please check!\n";

}

if(i == sensList.size() - 1)

{

showNotification("Warning!",body);

}

}

}

}

}, 0, timePeriod);

});

}

//showing a notification method with the title and body

public void showNotification(String title,String text)

{

Platform.runLater(() ->

{

//show notification

Notifications.create()

.title(title)

.text(text)

.showWarning();

});

}

//this method will be called by the rmi server when it needs to send an email or sms

@Override

public void notifyWarning(ArrayList<SensorModel> items)

{

Platform.runLater(() ->

{

//the sending email or sms notification will be shown only if the admin has logged into the system

if(isLoggedIn)

{

//setting up the notification body

String body = "";

for(int i = 0; i < items.size(); i++)

{

body = body + "Floor no " + items.get(i).getFloorNo() + ", Room no " + items.get(i).getRoomNo() + " sensor shows an unusual behaviour. Please check!\n";

if(i == items.size() - 1)

{

showNotification("Sending an email & SMS",body);

}

}

}

});

}

}

**Application.css**

/\* JavaFX CSS - Leave this comment until you have at least create one rule which uses -fx-Property \*/

**UI.fxml**

<?xml version="1.0" encoding="UTF-8"?>

<?import javafx.scene.paint.\*?>

<?import javafx.scene.text.\*?>

<?import javafx.scene.control.\*?>

<?import java.lang.\*?>

<?import javafx.scene.layout.\*?>

<AnchorPane prefHeight="500.0" prefWidth="500.0" xmlns="http://javafx.com/javafx/8" xmlns:fx="http://javafx.com/fxml/1" fx:controller="rmi.client.Client">

<children>

<Pane fx:id="loginPane" layoutX="15.0" layoutY="14.0" prefHeight="472.0" prefWidth="469.0">

<children>

<Label layoutX="72.0" layoutY="79.0" text="Admin Login">

<font>

<Font name="System Bold" size="20.0" />

</font>

</Label>

<TextField fx:id="password" layoutX="72.0" layoutY="202.0" prefHeight="39.0" prefWidth="335.0" promptText="Password" />

<Button layoutX="72.0" layoutY="253.0" mnemonicParsing="false" onAction="#Login" prefHeight="31.0" prefWidth="73.0" text="Login" />

<Label fx:id="response" alignment="CENTER" layoutX="240.0" layoutY="252.0" prefHeight="31.0" prefWidth="168.0" textAlignment="CENTER" />

<TextField fx:id="username" layoutX="72.0" layoutY="157.0" prefHeight="39.0" prefWidth="335.0" promptText="Username" />

<Button layoutX="156.0" layoutY="253.0" mnemonicParsing="false" onAction="#skipLogin" prefHeight="30.0" prefWidth="74.0" text="Skip" />

</children>

</Pane>

<Pane fx:id="loggedOnPane" layoutX="13.0" layoutY="10.0" prefHeight="480.0" prefWidth="476.0" visible="false">

<children>

<Pane fx:id="sensorFullDetailsPane" cacheShape="false" layoutX="10.0" layoutY="299.0" prefHeight="175.0" prefWidth="457.0" visible="false">

<children>

<Label fx:id="sensorID" layoutX="16.0" layoutY="18.0" text="Sensor ID" />

<Label fx:id="sensorFloorNumber" layoutX="18.0" layoutY="71.0" text="Floor Number" />

<Label fx:id="sensorRoomNumber" layoutX="16.0" layoutY="92.0" text="Room Number" />

<Label fx:id="sensorStatus" layoutX="16.0" layoutY="45.0" text="Status" />

<Label fx:id="sensorCo2Level" layoutX="14.0" layoutY="116.0" text="CO2 Level" />

<Label fx:id="sensorSmokeLevel" layoutX="16.0" layoutY="141.0" text="Smoke L:evel" />

<Button fx:id="updateSensorButton" layoutX="349.0" layoutY="137.0" mnemonicParsing="false" onAction="#UpdateSensor" text="Update Sensor" />

</children>

</Pane>

<Button fx:id="logoutButton" layoutX="247.0" layoutY="14.0" mnemonicParsing="false" onAction="#goBackToLoginScreen" prefHeight="25.0" prefWidth="104.0" text="Logout" />

<Label layoutX="8.0" layoutY="6.0" prefHeight="17.0" prefWidth="113.0" text="Welcome" textFill="#ff1c1c">

<font>

<Font name="System Bold" size="21.0" />

</font>

</Label>

<ListView fx:id="sensorListView" layoutX="8.0" layoutY="58.0" prefHeight="234.0" prefWidth="457.0" />

<Label layoutX="8.0" layoutY="37.0" text="All Sensors">

<font>

<Font name="Arial" size="12.0" />

</font>

</Label>

<Button fx:id="addNewSensorButton" layoutX="357.0" layoutY="14.0" mnemonicParsing="false" onAction="#AddNewSensor" text="Add New Sensor" />

</children>

</Pane>

<Pane fx:id="addNewSensorPane" layoutX="23.0" layoutY="20.0" prefHeight="480.0" prefWidth="476.0" visible="false">

<children>

<Label fx:id="addNewOrUpdateSensorLabel" alignment="CENTER" layoutX="66.0" layoutY="89.0" prefHeight="31.0" prefWidth="343.0" text="Add New Sensor" textFill="#ff1c1c">

<font>

<Font name="System Bold" size="21.0" />

</font>

</Label>

<TextField fx:id="floorNo" layoutX="66.0" layoutY="159.0" prefHeight="31.0" prefWidth="343.0" promptText="Floor Number" />

<TextField fx:id="roomNo" layoutX="67.0" layoutY="196.0" prefHeight="31.0" prefWidth="343.0" promptText="Room Number" />

<Button fx:id="createOrUpdateSensorButton" layoutX="66.0" layoutY="240.0" mnemonicParsing="false" onAction="#CreateNewSensor" prefHeight="30.0" prefWidth="130.0" text="Create" />

<Label fx:id="createSensorResponse" alignment="CENTER" contentDisplay="CENTER" layoutX="167.0" layoutY="241.0" prefHeight="31.0" prefWidth="243.0" textAlignment="CENTER" />

<Button layoutY="2.0" mnemonicParsing="false" onAction="#goBackToLoggedOnScreen" prefHeight="25.0" prefWidth="78.0" text="Go Back" />

</children>

</Pane>

</children>

</AnchorPane>

Sensor Application

**API.java**

package application;

import java.io.BufferedReader;

import java.io.InputStreamReader;

import java.net.HttpURLConnection;

import java.net.URL;

import org.json.JSONArray;

public class API

{

//check whether a sensor is avaiable by the id which is passed as a parameter

//and returning a boolean.

public boolean checkSensorById(String id)

{

try

{

//this is the url to check whther a sensor is avaiable or not

String burl = "http://localhost:5000/api/sensors/findSensorById?id=" +id;

//and replacing it with spaces with "%20" to make sure no spaces are in the url

String url = burl.replaceAll(" ","%20");

//creating URL object with the url we created

URL obj = new URL(url);

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

//the request method is GET

con.setRequestMethod("GET");

//getting the response code

int respondCode = con.getResponseCode();

//reading the response

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

String inputLine;

StringBuffer response = new StringBuffer();

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

{

response.append(inputLine);

}

in.close();

//store the response in to a JSONArray

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

//check the response code

//if it is 200, there is no issue with the connection or anything

//and checking the JSONArray size to make sure there is a sensor by the specified id.

//if avaiable we are returning true

//else rturning false which means there are no sensors by the id

if(respondCode == 200 && array.length() > 0)

{

return true;

}

return false;

}

catch(Exception e)

{

//if an exception occured we send the boolean false

return false;

}

}

//update a sensor method

public boolean updateSensor(String id,boolean isActive,int co2Level,int smokeLevel)

{

//getting id,active state,co2Level,smokeLevel as parameters

try

{

//creating the url to send the request

String burl = "http://localhost:5000/api/sensors/updateSensorBySensorApp?id=" + id + "&co2Level=" + co2Level + "&smokeLevel=" + smokeLevel + "&isActive="+ isActive;

//replacing spaces with the "%20" to eleminate spaces in the url

String url = burl.replaceAll(" ","%20");

//creating a URl object with the url we created

URL obj = new URL(url);

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

//setting the request type (Here it is post)

con.setRequestMethod("POST");

int respondCode = con.getResponseCode();

//if the response code is true

//the updates has been made successfully

//so we return true

if(respondCode == 200)

{

return true;

}

//else return false

return false;

}

catch(Exception e)

{

//if an exception occured we return false as well

return false;

}

}

}

**Application.css**

/\* JavaFX CSS - Leave this comment until you have at least create one rule which uses -fx-Property \*/

**Main.java**

package application;

import java.util.ArrayList;

import java.util.Random;

import java.util.Timer;

import java.util.TimerTask;

import javafx.application.Application;

import javafx.application.Platform;

import javafx.event.ActionEvent;

import javafx.fxml.FXML;

import javafx.fxml.FXMLLoader;

import javafx.stage.Stage;

import javafx.scene.Parent;

import javafx.scene.Scene;

import javafx.scene.control.Button;

import javafx.scene.control.Label;

import javafx.scene.control.TextField;

import javafx.scene.layout.BorderPane;

public class Main extends Application {

//all the UI elements

@FXML

private Label status;

@FXML

private Label co2Level;

@FXML

private Label smokeLevel;

@FXML

private TextField sensorIdField;

@FXML

private Button registerButton;

@FXML

private Label response;

@FXML

private Label countLabel;

//keeping a count to send the state of a sensor

private static int count = 10;

//storing the valid sensor id

private String sensorId;

//create an object of API class

API api = new API();

@Override

public void start(Stage primaryStage) throws Exception {

//loading the UI

Parent root = FXMLLoader.load(getClass().getResource("UI.fxml"));

Scene scene = new Scene(root,500,500);

primaryStage.setScene(scene);

primaryStage.show();

}

public static void main(String[] args) {

launch(args);

}

public void registerSensor(ActionEvent event) throws Exception

{

//this method will be triggered when the register button is clicked

//check whether the sensorIdField is entered or not

if(sensorIdField.getText().equals("") || sensorIdField.getText() == null)

{

//if not showing an error

response.setText("Enter a valid sensor ID");

return;

}

//if the sensorIdField is successfully entered

//checking the entered id valid or not by calling a method in the API class

//if valid it will return true

boolean isValidId = api.checkSensorById(sensorIdField.getText().toString());

//checking the boolean value

if(isValidId)

{

//if valid the registration is success

//and it is ready to send its state

sensorIdField.setDisable(true);

registerButton.setText("Registered");

registerButton.setDisable(true);

sensorId = sensorIdField.getText().toString();

response.setText("App was registered as a sensor!");

updateSensorDetails();

}

else

{

//else showing the error response

response.setText("Entered ID is not mapped to a sensor!");

}

}

//this method is reposnsible to update the sensor every 10 seconds

public void updateSensorDetails()

{

//send sensor status every 10 seconds

Timer timer = new Timer();

timer.schedule(new TimerTask() {

@Override

public void run()

{

//decreasing the count variable

count--;

//if the count == 0

//it is the time to send to state of a sensor to thr API

if(count == 0)

{

//we make the count varibale 10 again to show the remaining time in the app

count = 10;

//and this method will generate the sensor state

sendUpdates();

}

Platform.runLater(() -> {

countLabel.setText("Sensor updates will be sent in " +count + " seconds");

});

}

}, 0, 1000);

}

public void sendUpdates()

{

//generating random values for active state,co2 level and smoke level

Random rand = new Random();

int smoke = rand.nextInt((10 - 1) + 1) + 1;

int co2 = rand.nextInt((10 - 1) + 1) + 1;

Platform.runLater(() ->

{

//setting the UI elements with the randomly generated values

boolean isActive = false;

if((smoke + co2) > 10)

{

isActive = true;

}

if(isActive)

{

status.setText("Status : Active");

}

else

{

status.setText("Status : Inactive");

}

co2Level.setText("CO2 Level : " + co2);

smokeLevel.setText("Smoke Level : " + smoke);

//sending the values to the API with the randomly generated values

sendToAPI(isActive,smoke,co2);

});

}

private void sendToAPI(boolean isActive, int smoke, int co2)

{

//calling the API class updateSensor method to update the sensor with the parameters

boolean isUpdated = api.updateSensor(sensorId, isActive, co2, smoke);

Platform.runLater(() ->

{

//check whether it is updated or not

if(isUpdated)

{

response.setText("Updates were successfully sent!");

}

else

{

response.setText("Something went wrong!");

}

});

}

}

**UI.fxml**

<?xml version="1.0" encoding="UTF-8"?>

<?import javafx.scene.control.\*?>

<?import java.lang.\*?>

<?import javafx.scene.layout.\*?>

<?import javafx.scene.layout.AnchorPane?>

<AnchorPane prefHeight="500.0" prefWidth="500.0" xmlns="http://javafx.com/javafx/8" xmlns:fx="http://javafx.com/fxml/1" fx:controller="application.Main">

<children>

<TextField fx:id="sensorIdField" layoutX="83.0" layoutY="243.0" prefHeight="32.0" prefWidth="346.0" promptText="Enter Sensor ID" />

<Button fx:id="registerButton" layoutX="186.0" layoutY="285.0" mnemonicParsing="false" onAction="#registerSensor" prefHeight="32.0" prefWidth="148.0" text="Register As Sensor" />

<Label fx:id="status" layoutX="83.0" layoutY="150.0" prefHeight="17.0" prefWidth="346.0" text="Status" />

<Label fx:id="smokeLevel" layoutX="83.0" layoutY="178.0" prefHeight="17.0" prefWidth="346.0" text="Smoke Level" />

<Label fx:id="co2Level" layoutX="83.0" layoutY="206.0" prefHeight="17.0" prefWidth="346.0" text="CO2 Level" />

<Label fx:id="response" alignment="CENTER" layoutX="83.0" layoutY="329.0" prefHeight="17.0" prefWidth="346.0" />

<Label fx:id="countLabel" layoutX="12.0" layoutY="14.0" prefHeight="23.0" prefWidth="478.0" text="Update Details In" />

</children>

</AnchorPane>

Web Client Application

**Sensor.model.js**

const mongoose = require('mongoose')

const Schema = mongoose.Schema;

//this is the sensor schema of a sensor which contains all the required attributes

const SensorSchema = new Schema({

isActive : Boolean,

floorNo : String,

roomNo : String,

smokeLevel : Number,

CO2Level : Number,

});

const Sensor = mongoose.model('Sensor',SensorSchema);

module.exports = Sensor;

**user.model.js**

const mongoose = require('mongoose')

const Schema = mongoose.Schema;

//this is the user schema of a user which contains all the required attributes

const UserSchema = new Schema({

username : String,

password : String,

});

const User = mongoose.model('User',UserSchema);

module.exports = User;

**sensorRoute.js**

const express = require('express');

const router = express.Router();

let SensorModel = require('../models/sensor.model');

//this route is to get all the sesnors in the database

router.route('/').get((req, res) => {

SensorModel.find()

.then(sensors => res.status(200).json(sensors))

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

});

//this route is to create a new sensor

router.route('/createNewSensor').post((req,res) => {

//all the required attributes is taking from the request variable

const isActive = req.query.isActive;

const floorNo = req.query.floorNo;

const roomNo = req.query.roomNo;

const smokeLevel = req.query.smokeLevel;

const CO2Level = req.query.CO2Level;

//checking whether any of required attributes are not null

if(isActive != null && floorNo != null && roomNo != null && smokeLevel != null && CO2Level != null)

{

const newSensor = new SensorModel();

newSensor.isActive = isActive;

newSensor.floorNo = floorNo;

newSensor.roomNo = roomNo;

newSensor.smokeLevel = smokeLevel;

newSensor.CO2Level = CO2Level;

//saving the sensor in the database

newSensor.save()

.then(()=> res.json('New sensor was successfully added!'))

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

}

else

{

//sending an error code of 400

res.status(400).json('Error: Parameters are empty!' );

}

});

//this route is to update a sensor

router.route('/updateSensor').post((req,res) => {

//these are the reuired paramaters that is needed to update a sensor

const id = req.query.id;

const floorNo = req.query.floorNo;

const roomNo = req.query.roomNo;

//checking whether none of the are not equal to null

if(id != null && floorNo != null && roomNo != null)

{

//updating the sensor with its id

SensorModel.update({\_id: id}, {

floorNo: floorNo,

roomNo: roomNo

}).then(()=> res.json('Sensor Updated!').status(200))

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

}

else

{

res.status(400);

}

});

//this route is to find a sensor

router.route('/findSensorById').get((req,res) => {

//getting the id as a paramater

const id = req.query.id;

if(id != null )

{

//finding in the database

SensorModel.find({"\_id" : id})

.then(sensor => res.json(sensor).status(200))

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

}

else

{

//sending error coe of 400

res.status(400);

}

});

//this route is to update a sensor by the sensor App

router.route('/updateSensorBySensorApp').post((req,res) => {

//gathering all the reuired paramaters

const id = req.query.id;

const co2Level = req.query.co2Level;

const smokeLevel = req.query.smokeLevel;

const isActive = req.query.isActive;

//cheking whether they are null or not

if(id != null && co2Level != null && smokeLevel != null && isActive != null)

{

//update the sensor with attributes

SensorModel.update({\_id: id}, {

isActive: isActive,

smokeLevel: smokeLevel,

CO2Level : co2Level

}).then(()=> res.json('Sensor Updated!').status(200))

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

}

else

{

//sennding an error code

res.status(400);

}

});

module.exports = router;

**userRoute.js**

const express = require('express');

const router = express.Router();

let UserModel = require('../models/user.model');

router.route('/login').get((req,res) => {

const username = req.query.username;

const password = req.query.password;

if(username != null && password != null)

{

UserModel.find(

{"username" : username,"password" : password}

)

.then(users => res.json(users).status(200))

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

}

else

{

res.status(400).json('Error: Parameters are null');

}

});

router.route('/register').get((req,res) => {

const username = req.query.username;

const password = req.query.password;

const user = new UserModel();

user.username = username;

user.password = password;

if(username != null && password != null)

{

user.save()

.then(()=> res.json('User registered!').status(200))

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

}

else

{

res.status(400).json('Error: Parameters are null')

}

});

module.exports = router;

**server.js**

const express = require('express');

const cors = require('cors');

const mongoose = require('mongoose');

require('dotenv').config();

const app = express();

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

app.use(cors());

app.use(express.json());

//making the mongodb conection

const uri = process.env.ATLAS\_URI;

mongoose.connect(uri, {useNewUrlParser : true, useCreateIndex : true});

const connection = mongoose.connection;

connection.once('open', () => {

console.log("MongoDb database connection established sucessfully!");

});

//redirecting to the sensor route

const sensorRouter = require('./routes/SensorRoute');

app.use('/api/sensors',sensorRouter);

//redirecting to the user route

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

app.use('/api/users',userRouter);

//server is running on port

app.listen(port, () => {

console.log("server is running on port " + port);

});

**App.js**

import React from 'react';

import './App.css';

//importing the sensorView component

import SensorView from './components/sensorView';

//importing the css

import css from './css/app.css';

function App() {

return (

//setting the sensorView

<div className="container">

<SensorView/>

</div>

);

}

export default App;

**list.js**

import React from 'react'

import css from '../css/list.css';

import warningImage from '../images/warning.png';

function App(props)

{

//getting the sensor list from props

var sensors = [];

sensors = props.sensorList;

//run a for each loop to create a list

const sensorList = sensors.map(item => {

return (

<div className="parentSensorRow" key={item.\_id}>

<div>

{

//check whether the sensor is active or not

(item.isActive == true) &&

<p>

<p>{"Status : Active"}</p>

<p>{"Floor Number : "+ item.floorNo }</p>

<p>{"Room Number : "+ item.roomNo }</p>

<p>{"Smoke Level : "+ item.smokeLevel }</p>

<p>{"CO2 Level : "+ item.CO2Level }</p>

</p>

}

{

//checking whether the active status of the sensor is false

(item.isActive == false) &&

<p>

<p>{"Status : Inactive"}</p>

<p>{"Floor Number : "+ item.floorNo }</p>

<p>{"Room Number : "+ item.roomNo }</p>

<p>{"Smoke Level : "+ item.smokeLevel }</p>

<p>{"CO2 Level : "+ item.CO2Level }</p>

</p>

}

{(item.smokeLevel > 5 || item.CO2Level > 5) &&

<p className="warningText">Warning!</p>

}

{

//checking the smoke level whether its is less than or equal to 5 or not

(item.smokeLevel <= 5 && item.CO2Level <= 5) &&

<h2>

No warning!

</h2>

}

</div>

</div>

)

})

return(

//showing the final sensor list

<div className="sensorList">

{sensorList}

</div>

)

}

export default App;

**sensorView.js**

import React from 'react'

import axios from 'axios';

//importing the sensorList component

import SensorList from './list';

class App extends React.Component

{

//constructor of the class

constructor(props)

{

super(props);

//keeping a list in the states

this.state = {

list : []

}

//getting sensors from the api

this.getCurrentSensors();

}

//getting all the sensors by sending a request to the api using axios

getCurrentSensors()

{

//for first time loading sensors

axios.get("http://localhost:5000/api/sensors/")

.then(response => {

if(response.status == 200)

{

//setting the sensor list with the response we got

var li = response.data;

//reverse it to get the newly created sensors to the front

li.reverse();

//setting the list in states

this.setState({list :li});

}

})

.catch(error => alert(error));

//for update the sensors every 40 seconds

const timePeriod = 40000;

setInterval(() =>{

axios.get("http://localhost:5000/api/sensors/")

.then(response => {

if(response.status === 200)

{

//this method will be called every 40 seconds and update the list

var li = response.data;

li.reverse();

this.setState({list :li});

}

})

.catch(error => alert(error));

}, timePeriod);

}

render() {

return (

//stting the sensor list by passing the sensors we got by the API

<div>

<SensorList sensorList ={this.state.list}></SensorList>

</div>

)

}

}

export default App;

**app.css**

.container

{

background-color: darkorchid;

width: 100%;

min-height: 635px;

padding-bottom: 5%;

padding-top:5%;

height: auto; }

**List.css**

.sensorList

{

width: 55%;

height: auto;

margin-left: auto;

margin-right: auto;

background-color: white;

border-radius: 10px;

border: 10px solid #F1F1F1;

}

.parentSensorRow

{

background-color: white;

padding: 10px;

border-radius: 10px;

border: 5px solid darkorchid;

margin-bottom: 10px;

margin-top: 10px;

}

.warningImage

{

width: 5%;

height: 5%;

}

.activeStatus

{

background-color: red;

color: white;

border-radius: 10px;

padding-top: 10px;

padding-bottom: 10px;

text-align: center;

}

.warningText

{

background-color: red;

color: white;

border-radius: 10px;

padding-top: 10px;

padding-bottom: 10px;

text-align: center;

}