Distributed Systems – SE3020

Assignment 02 – REST API

REPORT

1. IT17180108 – A.D. Kalpani Sandupaba Abeysinghe
2. IT17186766 - T.J.P.S Liyanage

CONTENT

1. Introduction about the system -- 3

2. Technologies Used--- 4

3. System Overview and Flow Diagrams---5

3. RMI---7

3.1. RMI Server--7

3.2. RMI API---8

3.3 RMI GUI Client---8

4.REST API--- 9

5.Appendix--10

1. Introduction about the system

The implemented fire alarm system is concurrent smoke and co2 level monitoring sensor management system which helps the user to monitor the co2 and smoke levels of the air in the real time.

Every fire alarm sensor has location which it has been implemented, their status whether it is in the activated mode or the deactivated mode and real time reding of the co2 levels and smoke levels of the air.

System has records of the existing fire alarms and system has been updated in every 10 seconds with the current reading from the fire sensors.

Those fire sensor values are viewed by the system administrator. If any of the senor detects the smoke level or the co2 level value which is greater than the 10, then the alert message is displayed in the system in order to notify the administrator.

The system contains desktop application which can be used by the system administrator. Only a user who has valid admin credentials can view sensor data and add (register) new fire sensor to the system.

From Administrator dashboard user can view the real time fire sensor status of the desktop application.

Web client can only used to view the fire alarm status only.

2. Technologies Used

Desktop Application - Java Fx and Java RMI (Remote Method Invocation)

Web Client – React JS

Middleware Technologies – Node JS, Express

Database – Mongo DB Altes

Development Environments and Tools Used

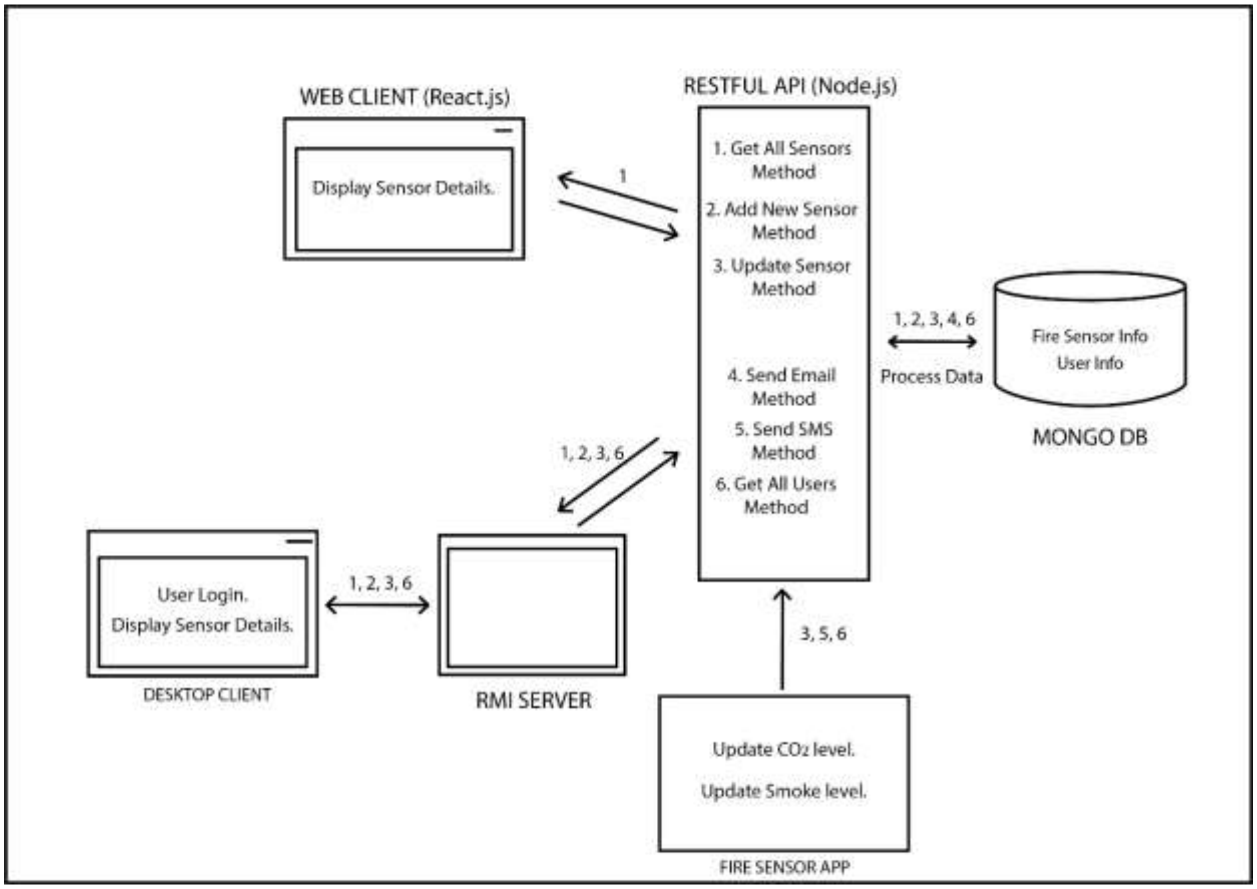
Desktop Application – Netbeans IDE 8.2 RC

Web Client – VS Code

Middleware Technologies – Node JS - v14.0.0

Database – Mongo DB Altes [cloud based mongo db solution]

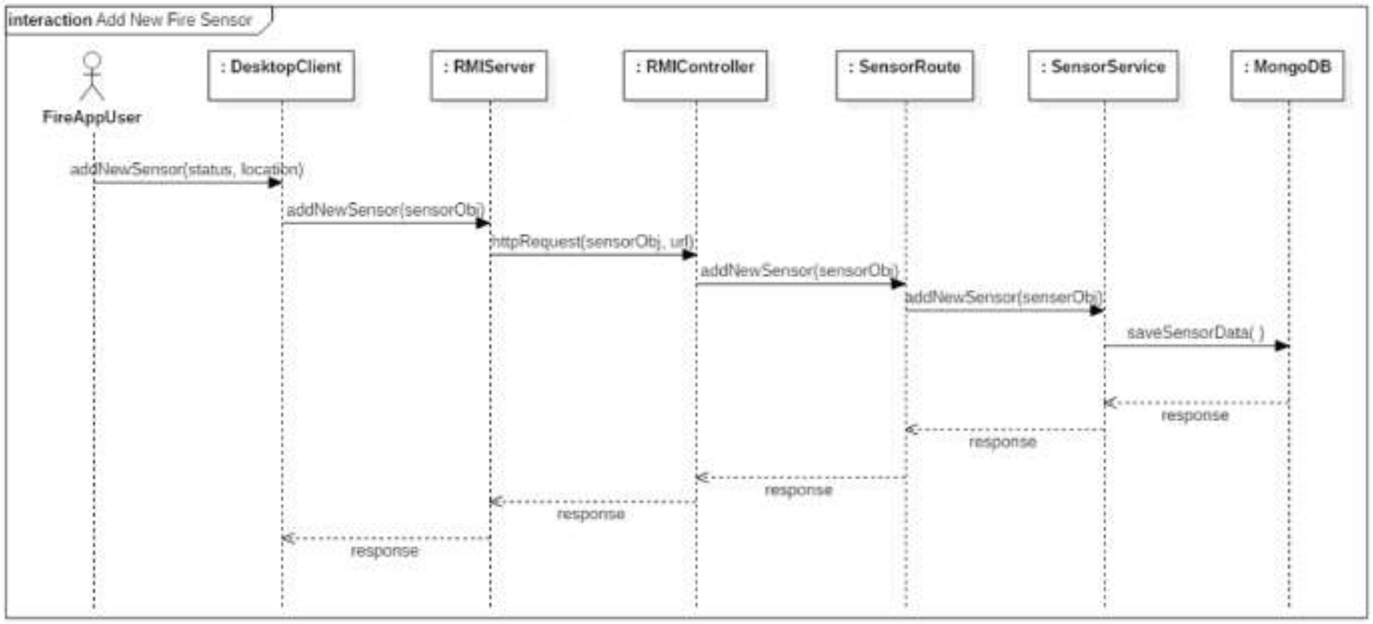
3. System Overview and Flow Diagrams



Appendix A.1 – System Overview Components

This diagram displays systems overview components and how they are intercommunicating.

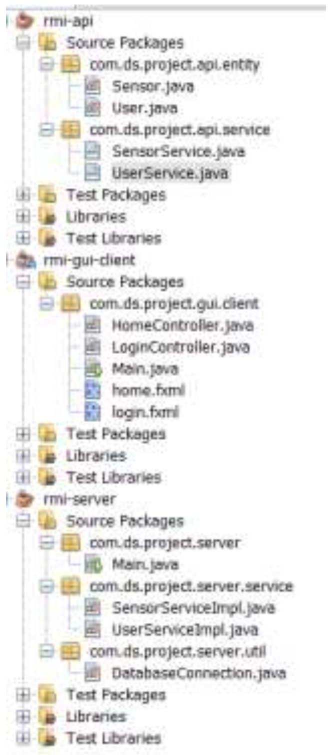
A brief explanation about one scenario of the intercommunication is, when we want to display the all the data related to the fire sensors of the system, the initial server call is done by the desktop client, invoking the getAllSensor method through the call back method. Then the RMI server send that RMI call back to the remote object which has been implemented using the remote interface. Through the remote interface call back method call REST API call has been made in order to get data from the database. Once the Data has been retrieved as the response, data is displayed in the desktop clients and web clients.



Appendix A.2 – Add new fire Sensor – Sequence Diagram

This sequence diagram shows the process of the adding new fire alarm sensor to the System.

4. RMI



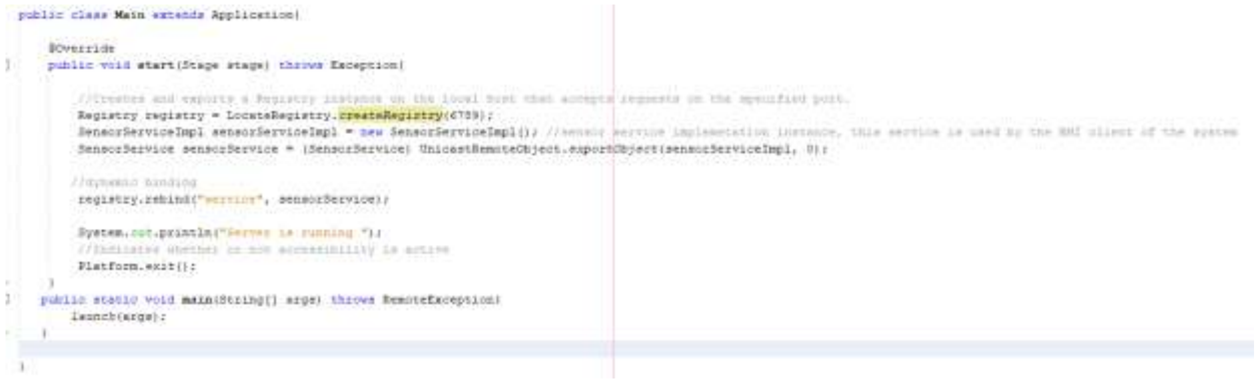
```
rmi-api
    Source Packages
        com.ds.project.api.entity
            Sensor.java
            User.java
        com.ds.project.api.service
            SensorService.java
            UserService.java
    Test Packages
    Libraries
    Test Libraries
rmi-gui-client
    Source Packages
        com.ds.project.gui.client
            HomeController.java
            LoginController.java
            Main.java
            home.fxml
            login.fxml
    Test Packages
    Libraries
    Test Libraries
rmi-server
    Source Packages
        com.ds.project.server
            Main.java
        com.ds.project.server.service
            SensorServiceImpl.java
            UserServiceImpl.java
        com.ds.project.server.util
            DatabaseConnection.java
    Test Packages
    Libraries
    Test Libraries
```

This is the folder structure Of the RMI project. RMI API contains the model classes and call back interfaces of the project.

RMI-GUI-Client contains all the UI files and controller classes related to that.

RMI-Server contains all the server, RMI registry implementation and call back remote object implementation of the project.

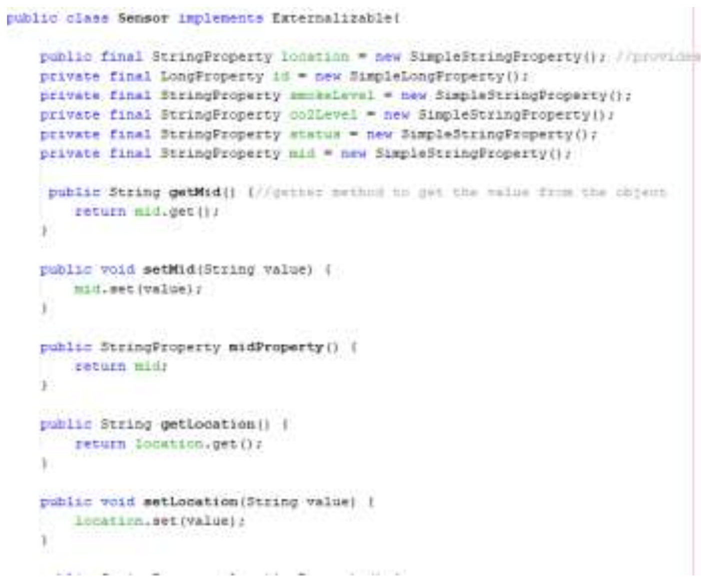
3.1. RMI Server



```java
public class Main extends Application{

    @Override
    public void start(Stage stage) throws Exception{

        //Creates and exports a Registry instance on the local host that accepts requests on the specified port.
        Registry registry = LocateRegistry.createRegistry(6789);
        SensorServiceImpl sensorServiceImpl = new SensorServiceImpl(); //sensor service implementation instance, this service is used by the RMI client of the system
        SensorService sensorService = (SensorService) UnicastRemoteObject.exportObject(sensorServiceImpl, 0);

       //dynamic binding
        registry.rebind("service", sensorService);

        System.out.println("Server is running ");
        //Indicates whether or not accessibility is active
        Platform.exit();
    }
   public static void main(String[] args) throws RemoteException{
       launch(args);
   }

}
```

This code snippet is used to create and exports new RMI registry on port 6789.

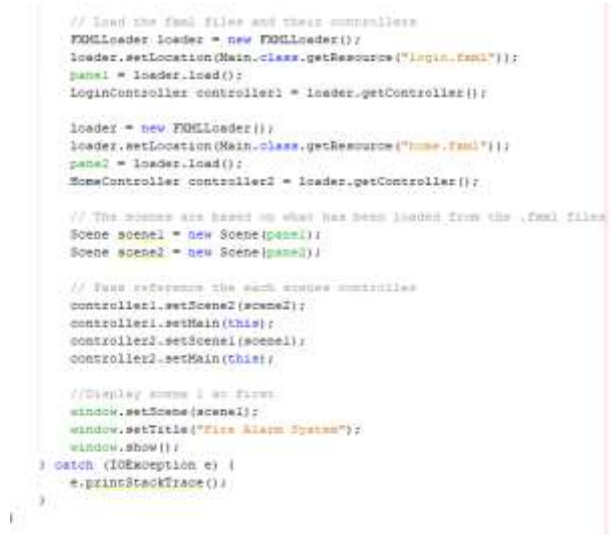
4.2. RMI API



```
public class Sensor implements Externalizable{

    public final StringProperty location = new SimpleStringProperty(); //provides a full implementation of a Property wrapping a String value
    private final LongProperty id = new SimpleLongProperty();
    private final StringProperty smokeLevel = new SimpleStringProperty();
    private final StringProperty co2Level = new SimpleStringProperty();
    private final StringProperty status = new SimpleStringProperty();
    private final StringProperty mid = new SimpleStringProperty();

     public String getMid() {//getter method to get the value from the object
        return mid.get();
    }

    public void setMid(String value) {
        mid.set(value);
    }

    public StringProperty midProperty() {
        return mid;
    }

    public String getLocation() {
        return location.get();
    }

    public void setLocation(String value) {
        location.set(value);
    }
```

This shows the model class for the sensor which is used to store the data in data processing.

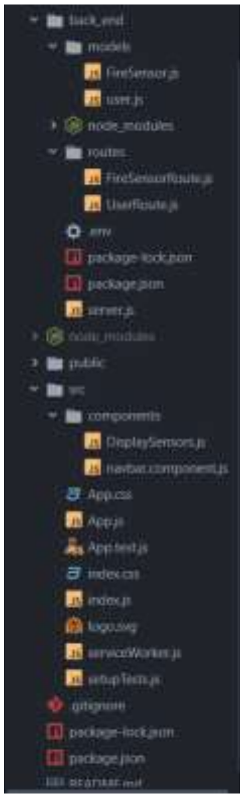
3.3 RMI GUI Client



```
        // Load the fxml files and their controllers
        FXMLLoader loader = new FXMLLoader();
        loader.setLocation(Main.class.getResource("login.fxml"));
        pane1 = loader.load();
        LoginController controller1 = loader.getController();

        loader = new FXMLLoader();
        loader.setLocation(Main.class.getResource("home.fxml"));
        pane2 = loader.load();
        HomeController controller2 = loader.getController();

        // The scenes are based on what has been loaded from the .fxml files
        Scene scene1 = new Scene(pane1);
        Scene scene2 = new Scene(pane2);

        // Pass reference the each scenes controller
        controller1.setScene2(scene2);
        controller1.setMain(this);
        controller2.setScene1(scene1);
        controller2.setMain(this);

        //Display scene 1 at first
        window.setScene(scene1);
        window.setTitle("Fire Alarm System");
        window.show();
    } catch (IOException e) {
        e.printStackTrace();
    }
}
```

This code shows the loading the FXML files into the stage using FXML Loader. And link the scene with each other.

4.REST API



```
back_end
  models
    FireSensor.js
    user.js
  node_modules
  routes
    FireSensorRoute.js
    UserRoute.js
  .env
  package-lock.json
  package.json
  server.js
node_modules
public
src
  components
    DisplaySensors.js
    navbar.component.js
  App.css
  App.js
  App.test.js
  index.css
  index.js
  logo.svg
  serviceWorker.js
  setupTests.js
.gitignore
package-lock.json
package.json
README.md
```

This is the folder structure for the web client(React.js) and the RESTFUL service. back_end folder includes server.js, routes folder and model folder.

Server.js includes required modules, require files, mongoDB connection method, file routes and codes need to run the RESTFUL service.

routes folder conatins all the methods for process RESTFUL services. First method is to get all sensor details from database. Second method is for add new sensor details to the databsae. Third metod can use to update details of a fire sensor. Fourth mehod can be used to send emails about critical fire sensors. Last method is for send sms about critical fire sensors. Since sms functions can't implement with out payments we used only for print the phone number and the message in the console.

Models folder includes models for users and fire sensors. These two models used for process sensors and user details with mongoDB through RESTFUL service.

React app components are in the components folder. DisplaySensors file is for display sensor details and navbar component is also there.

5.Appendix

Web App

FireSensor.js

```
const mongoose = require('mongoose');


const Schema = mongoose.Schema;


const fireSensorSchema = new Schema ({

  state: {

    // Validations

    type: String,

    required: true,

  },

  location: {

    type: String,

    required: true,

  },

  smoke: {

    type: Number,

    required: true,

  },

  co2: {

    type: Number,

    required: true,

  },
}, {
  timestamps: true,
});

const FireSensor = mongoose.model('FireSensor', fireSensorSchema);

module.exports = FireSensor;
```

User.js

```
const mongoose = require('mongoose');

const Schema = mongoose.Schema;

const userSchema = new Schema ({
  username: {
    type: String,
    required: true
  },
  password: {
    type: String,
    required: true
  },
}, {
  timestamps: true,
});

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

module.exports = User;
```

FireSensorRoute.js

```
const router = require('express').Router();
const nodemailer = require('nodemailer');

// REQUIRE MODEL CLASS
let FireSensor = require('../models/FireSensor');

// GET ALL FIRESENSOR
router.route('/').get((req, res) => {
  FireSensor.find()
```

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

// ADD NEW FIRESENSOR
router.route('/add').post((req, res) => {
  const state = req.body.state;
  const location = req.body.location;
  const smoke = Number(req.body.smoke);
  const co2 = Number(req.body.co2);

  // NEW FIRESENSOR INSTENCE
  const newSensor = new FireSensor({
    state,
    location,
    smoke,
    co2,
  });

  // SAVE NEW FIRESENSOR
  newSensor.save()
    .then(() => res.json('FireSensor added..'))
      .catch(err => res.status(400).json('Error: ' + err));
});

// GET A SINGLE FIRESENSOR
router.route('/:id').get((req, res) => {
  FireSensor.findById(req.params.id)
```

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

// UPDATE A FIRESENSOR
router.route('/update/:id').post((req, res) => {
  FireSensor.findById(req.params.id)
    .then(sensor => {
      sensor.state = req.body.state;
      sensor.location = req.body.location;
      sensor.smoke = Number(req.body.smoke);
      sensor.co2 = Number(req.body.co2);

      sensor.save()
        .then(() => res.json('FireSensor Updated..'))
          .catch(err => res.status(400).json('Error: ' + err));
    })
      .catch(err => res.status(400).json('Error: ' + err));
});

// DELETE A FIRESENSOR
router.route('/delete/:id').delete((req, res) => {
  FireSensor.findByIdAndDelete(req.params.id)
    .then(() => res.json('FireSensor Deleted..!'))
      .catch(err => res.status(400).json('Error: ' + err));
});

// SEND EMAILS
```

```
router.route('/send/:email/:body').post((req, res) => {
  let email = req.params.email;
  let body = req.params.body;

  let transporter = nodemailer.createTransport({
    service: 'gmail',
    port: 25,
    secure: false,
    auth: {
      user: 'prabathshalithads@gmail.com',
      pass: 'kjksz263ds'
    }
  });

  let mailOptions = {
    from: 'prabathshalithads@gmail.com',
    to: email,
    subject: 'Fire App Alert Email',
    text: body
  };

  transporter.sendMail(mailOptions, function(error, info){
    if (error) {
      console.log(error);
    } else {
      console.log('Email sent: ' + info.response);
    }
  });
});


// SEND SMS
router.route('/sms/:phone/:text').post((req, res) => {
  let phone = req.params.phone;
  let text = req.params.text;

  console.log("Phone : " + phone + " Text : " + text);
});

module.exports = router;
```

UserRoute.js

```
const router = require('express').Router();

let User = require('../models/user');

// GET ALL USERS
router.route('/').get((req, res) => {
  User.find()
    .then(users => res.json(users))
      .catch(err => res.status(400).json('Error: ' + err));
});
```

```
// ADD NEW USER
router.route('/add').post((req, res) => {
  const username = req.body.username;
  const password = req.body.password;

  // NEW USER INSTENCE
  const newUser = new User({
    username,
    password
  });

  // SAVE NEW USER
  newUser.save()
    .then(() => res.json('User added..'))
      .catch(err => res.status(400).json('Error: ' + err));
});

module.exports = router;
```

DisplaySensor.js

```
import React, {Component} from 'react';
import {Link} from 'react-router-dom';
import  axios from 'axios';
import '../App.css';

export default class DisplaySensors extends Component {
  constructor(props) {
      super(props);
      this.state = {
        sensors: []
      }
  }

  // GET ALL SENSOR DETAILS
  componentDidMount() {
    // UPDATE THE STATE AFTER EVERY 40 SECOND
    this.interval = setInterval(() => this.setState({ time: Date.now() }), 40000);

    axios.get('http://localhost:5000/firesensor/')
      .then(response => {
        this.setState({sensors: response.data})
      })
        .catch((error) => {
          console.log(error);
        })
  }

  componentWillUnmount() {
    clearInterval(this.interval);
  }


  // DISPLAY SENSOR DETAILS

  sensorList() {

    return this.state.sensors.map(fireSensor => {

      return  <tr>

                <td>{fireSensor.state}</td>

                <td>{fireSensor.location}</td>

                <td className={(fireSensor.smoke <= 5) ? "active" : "inactive"}>{fireSensor.sm
oke}</td>

                <td className={(fireSensor.co2 <= 5) ? "active" : "inactive"}>{fireSensor.co2}
</td>

            </tr>;

    })

  }


  render() {

    return (

      <div>

        <h3 className="text-center mt-3 mb-3"><i className="fas fa-bell mr-
3"></i>Fire Sensors</h3>

        <table className="table">

          <thead className="thead-light">

            <tr>

              <th>State</th>

              <th>Location</th>

              <th>Smoke Level</th>

              <th>CO2 Level</th>

            </tr>
          </thead>
          <tbody>
            {this.sensorList()}
          </tbody>
        </table>
      </div>
    );
  }
}
```

Navbar.componet.js

```
import React, {Component} from 'react';
import {Link} from 'react-router-dom';

export default class Navbar extends Component {
  render() {
    return (
      <nav className="navbar navbar-dark bg-dark navbar-expand-lg">
        <Link to="/" className="navbar-brand"><i className="fas fa-fire fa-lg mr-
3"></i>Fire App</Link>
        <div className="collpase navbar-collpase">
        </div>
      </nav>
    );
  }
}
```

App.js

```
import React from 'react';
import { BrowserRouter as Router, Route } from 'react-router-dom';
import 'bootstrap/dist/css/bootstrap.min.css';

import Navbar from './components/navbar.component';
import DisplaySensors from './components/DisplaySensors';

function App() {
  return (
    <Router>
      <Navbar/>
      <div className="container">
        <br/>
        <Route path="/" exact component={DisplaySensors}/>
      </div>
    </Router>
  );
}

export default App;
```

.env

ATLAS_URI = mongodb://shali12359:kjksz263@cluster0-shard-00-00-wsl7m.mongodb.net:27017,cluster0-shard-00-01-wsl7m.mongodb.net:27017,cluster0-shard-00-02-

wsl7m.mongodb.net:27017/test?ssl=true&replicaSet=Cluster0-shard-0&authSource=admin&retryWrites=true&w=majority

In order to shorten the document length, we have removed all the import statements, FXML variable declaration from the appendix.

Sensor.js

```
package com.ds.project.api.entity;

//model class for the sensor model
public class Sensor implements Externalizable{

   public final StringProperty location = new SimpleStringProperty(); //provides a full implementation of a Property wrapping a String value
   private final LongProperty id = new SimpleLongProperty();
   private final StringProperty smokeLevel = new SimpleStringProperty();
   private final StringProperty co2Level = new SimpleStringProperty();
   private final StringProperty status = new SimpleStringProperty();
   private final StringProperty mid = new SimpleStringProperty();

   //provide the logic for serialization i.e. writing the fields of class into bytes
   @Override
   public void writeExternal(ObjectOutput out) throws IOException {
      out.writeLong(getId());
      out.writeObject(getLocation());
```

```
        out.writeObject(getSmokeLevel());
        out.writeObject(getCo2Level());
        out.writeObject(getStatus());
        out.writeObject(getMid());
    }

    //this method must read the values in the same sequence and with the same types as were written by writeExternal() method
    @Override
    public void readExternal(ObjectInput in) throws IOException, ClassNotFoundException {
        setId(in.readLong());
        setLocation((String) in.readObject());
        setSmokeLevel((String) in.readObject());
        setCo2Level((String) in.readObject());
        setStatus((String) in.readObject());
        setMid((String) in.readObject());

         }

}
```

SensorService.js

```
package com.ds.project.api.service;
```

```
//callback interface for the sensor client

//client has to implement this call back interface

public interface SensorService extends Remote {


   Sensor insertSensor(Sensor sensor) throws RemoteException; // used to insert sensor object into the
database


   List<Sensor> updateSensor(ArrayList<Sensor> array) throws RemoteException; //used to update the
sensor state


   void deleteSensor(Long id) throws RemoteException; //delete the sensor from the database


   Sensor getSensorById(Long id) throws RemoteException; //get sensor detailes by providing the id


   List<Sensor> getAllSensor() throws RemoteException; //get the list of the all the sensors avalible in
the database


   List<User> getAllUser() throws RemoteException; //get all the users available inside the database


   void sendNotification(String msg) throws RemoteException; //sending the notification for the co2 or
smoke change


}
```

UserService.js

```
package com.ds.project.api.service;


//call back interface for the users client

//client has to implement this call back interface

public interface UserService extends Remote{


    List<User> getAllUser() throws RemoteException; //get the list of the all the users avalible in the
database

}
```

HomeController.js

```
package com.ds.project.gui.client;



//controller class for the login.fxml file

public class HomeController implements Initializable {


   private TextField txtid;


   private Main main; //main context inorder to store the main context of the application

   private SensorService sensorService; //service object to store the service

   private Scene scene1;
```

```
    @Override
    public void initialize(URL url, ResourceBundle rb) {
        collocation.setCellValueFactory(new PropertyValueFactory<>("location"));
        colsmokelevel.setCellValueFactory(new PropertyValueFactory<>("smokeLevel"));
        colco2level.setCellValueFactory(new PropertyValueFactory<>("co2Level"));
        colstatus.setCellValueFactory(new PropertyValueFactory<>("status"));
    }

    @FXML
    private void onInsert(ActionEvent event) { //method for the insert button click event
//        System.out.println("button clicked");

        if (isFieldValid()) { //checking if any filed in the add senor is empty


        }
        try {
            Sensor sensor = new Sensor(); //creating the new sensor object in order to store the data
            sensor.setLocation(txtlocation.getText()); //initializing the
            sensor.setSmokeLevel(txtsmokelevel.getText());
            sensor.setCo2Level(txtco2level.getText());
            sensor.setStatus(txtstatus.getText());
            sensor = sensorService.insertSensor(sensor);
            System.out.println(sensor);
            tableView.getItems().add(sensor);
        } catch (Exception ex) {
            ex.printStackTrace();
        }
    }

    @FXML
    private void onRefresh(ActionEvent event) {
        clearField();
    }

    public void setMain(Main main) {
        this.main = main;
        this.sensorService = main.getSensorService();
        String msg = "";

        try {
            List<Sensor> all = new ArrayList<>();

            all = sensorService.getAllSensor();
            tableView.getItems().setAll(all);

            for (Sensor temp : all) {

                final List<Sensor> up = all;
                final String msgNot = msg;
            Timeline fiveSecondsWonder = new Timeline(new KeyFrame(Duration.seconds(100), new
EventHandler<ActionEvent>() {

                @Override
                public void handle(ActionEvent event) {
                    try {
                        System.out.println("this is called");
                        List<Sensor> result = new ArrayList<>();
                        System.out.println(msgNot);
                        result = sensorService.updateSensor((ArrayList<Sensor>) up);
                        showNotification(result);

                    } catch (RemoteException ex) {
                        Logger.getLogger(HomeController.class.getName()).log(Level.SEVERE, null, ex);
                    }
                }
            }));

            fiveSecondsWonder.setCycleCount(Timeline.INDEFINITE);

            fiveSecondsWonder.play();
        }

    } catch (RemoteException ex) {
        ex.printStackTrace();
    }
}


public void showNotification(List<Sensor> up) throws RemoteException {
    System.out.println("show notifica");
    for (Sensor temp : up) {
        if (Long.parseLong(temp.getCo2Level()) > 5 || Long.parseLong(temp.getCo2Level()) > 5) {
            Alert alertlevel = new Alert(Alert.AlertType.ERROR);
            alertlevel.initModality(Modality.APPLICATION_MODAL);
            alertlevel.setTitle("Level Alert");
            alertlevel.setHeaderText("Smoke and CO2 level Alert");
            String msg = "CO2 Level " + temp.getCo2Level() + " Smoke Level " + temp.getSmokeLevel() + "
for " + temp.getLocation() + " is High";
            alertlevel.setContentText(msg);
//                alertlevel.setContentText("Notofication has been sent");
            sensorService.sendNotification(msg);
            alertlevel.show();
        }
    }
}


private void clearField() {
    txtid.setText("");
    txtlocation.setText("");
    txtsmokelevel.setText("");
    txtco2level.setText("");
    txtstatus.setText("");
}


private boolean isFieldValid() {
    String errorMessage = "";
    if (txtlocation.getText() == null || txtlocation.getText().isEmpty()) {
        errorMessage += "No valid location";
    }
    if (txtsmokelevel.getText() == null || txtsmokelevel.getText().isEmpty()) {
        errorMessage += "No valid smoke level";
    }
    if (txtco2level.getText() == null || txtco2level.getText().isEmpty()) {
        errorMessage += "No valid co2 level";
    }
    if (txtstatus.getText() == null || txtstatus.getText().isEmpty()) {
        errorMessage += "No valid status";
    }
    if (errorMessage.length() == 0) {
        return true;
    } else {
        Alert alert = new Alert(Alert.AlertType.ERROR);
        alert.initModality(Modality.APPLICATION_MODAL);
        alert.setTitle("Invalid Inputs");
        alert.setHeaderText("please enter valid fields");
        alert.setContentText(errorMessage);
        alert.show();
```

```
          return false;
        }


    }



}
```

LoginController.js

```
package com.ds.project.gui.client;


public class LoginController implements Initializable {
    private SensorService sensorService;
    private Main m;
    private boolean valid = false;
    List<User> all = new ArrayList<>();

    private Scene scene2;
    private Main main;
    @FXML
    private void onlogin(ActionEvent event) throws IOException {
        for (User temp : all) {


            if (temp.getUsername().equalsIgnoreCase(txtusername.getText()) &&
(temp.getPassword().equalsIgnoreCase(txtpassword.getText()))) {

                System.out.println("inside");

                valid = true;

            }

        }


        if (valid) {


            main.setScene(scene2);


        } else {

            Alert alertlevel = new Alert(Alert.AlertType.ERROR);

            alertlevel.initModality(Modality.APPLICATION_MODAL);

            alertlevel.setTitle("Wrong credential");

            alertlevel.setHeaderText("");

            alertlevel.setContentText("Please enter correct username or password");

            alertlevel.show();

        }


    }


    public void setMain(Main main) throws RemoteException {

        this.main = main;
        m = main;
        this.sensorService = main.getSensorService();


        System.out.println(txtusername.getText());
        all = sensorService.getAllUser();


        System.out.println(all.size());


    }


    Stage prevStage;


    void setPrevStage(Stage stage) {
        this.prevStage = stage;
    }


    void setScene2(Scene scene2) {
        this.scene2 = scene2;
    }


}
```

Main.js

```
package com.ds.project.server;


public class Main extends Application{


   @Override
   public void start(Stage stage) throws Exception{


      //Creates and exports a Registry instance on the local host that accepts requests on the specified port.
      Registry registry = LocateRegistry.createRegistry(6789);
      SensorServiceImpl sensorServiceImpl = new SensorServiceImpl(); //sensor service implemetation instance, this service is used by the RMI client of the system
      SensorService sensorService = (SensorService) UnicastRemoteObject.exportObject(sensorServiceImpl, 0);


      //dynamic binding
      registry.rebind("service", sensorService);


      System.out.println("Server is running ");
      //Indicates whether or not accessibility is active
      Platform.exit();
   }
}
```

SensorServiceImpl.js

```
package com.ds.project.server.service;

//call back interface implemenatation for the client
public class SensorServiceImpl implements SensorService {

  StringBuffer response;

  @Override
  public Sensor insertSensor(Sensor sensor) throws RemoteException {

    Sensor s = new Sensor();

    String POST_URL = "http://localhost:5000/firesensor/add"; //URL for the insert

    try {
      URL obj = new URL(POST_URL);// URL initialization
      HttpURLConnection cont = (HttpURLConnection) obj.openConnection(); //creaing the connection

      cont.setDoOutput(true);

      cont.setDoInput(true);
      cont.setRequestProperty("Content-Type", "application/json");
      cont.setRequestProperty("Accept", "application/json");
      cont.setRequestMethod("POST"); //setting the request method type to POST
```

```
            JSONObject obj1 = new JSONObject(); // Initializing the JSON object to pass the header values
            obj1.put("location", sensor.getLocation()); //Adding values to the JSON object
            obj1.put("smoke", sensor.getSmokeLevel());
            obj1.put("co2", sensor.getCo2Level());
            obj1.put("state", sensor.getStatus());

            OutputStreamWriter wr = new OutputStreamWriter(cont.getOutputStream());

            System.out.println(obj1.toString());
            wr.write(obj1.toString());
            wr.flush();

            try (BufferedReader br = new BufferedReader(new InputStreamReader(cont.getInputStream(),
"utf-8"))) { // sending the request
                StringBuilder response = new StringBuilder();
                String responseLine = null;
                while ((responseLine = br.readLine()) != null) {
                    response.append(responseLine.trim()); //storing the http response message into the string
builder
                }

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

        } catch (Exception ex) {
            ex.printStackTrace();
        }

        return sensor; // returing the inserted object
    }

    @Override
    public List<Sensor> getAllSensor() throws RemoteException {

        String url = "http://localhost:5000/firesensor/"; // The GET request URL.
        List<Sensor> list = new ArrayList<>();
        try {
            URL obj = new URL(url); // URL initialization.
            HttpURLConnection con = (HttpURLConnection) obj.openConnection(); // HttpURLConnection
opens a connection.

            con.setRequestMethod("GET"); // sets the Method
//          con.setRequestProperty("User-Agent", "Mozilla/5.0");

            int responseCode = con.getResponseCode(); // Retrieves the status CODE such as, 200 or 404.

            BufferedReader in = new BufferedReader(new InputStreamReader(con.getInputStream())); //
Initialized the
            // BufferReader.
            String inputLine;

            while ((inputLine = in.readLine()) != null) {
                this.response = null;

                this.response = new StringBuffer();

                this.response.append(inputLine); // Append the string to the response.

            }


            JSONArray jSONArray = new JSONArray(response.toString()); //adding the response object into
the JSON Array

//          System.out.println(jSONArray.length());

            for (int count = 0; count < jSONArray.length(); count++) { //Extracting the values from theJSON
Array


                Sensor s = new Sensor();//creating new sensor object to store the values from database

                JSONObject myResponse = jSONArray.getJSONObject(count);


                s.setMid(myResponse.getString("_id")); //storing JSON object information insidee the sensor
object

                s.setLocation(myResponse.getString("location")); //values are accsed through the

                s.setSmokeLevel(String.valueOf(myResponse.getLong("smoke")));

                s.setCo2Level(String.valueOf(myResponse.getLong("co2")));

                s.setStatus(myResponse.getString("state"));

                list.add(s);

            }


            in.close(); // Buffer is closed.


        } catch (Exception e) {

            e.printStackTrace();
            System.out.println("Make sure you've run the API Server!");
        }


        return list;
    }


    @Override
    public List<User> getAllUser() throws RemoteException {


        String url = "http://localhost:5000/users/"; // The GET request URL.


        List<User> list = new ArrayList<>();
        try {
            URL obj = new URL(url); // URL initialization.
            HttpURLConnection con = (HttpURLConnection) obj.openConnection(); // HttpURLConnection
opens a connection.


            con.setRequestMethod("GET"); // sets the Method


            int responseCode = con.getResponseCode(); // Retrieves the status CODE such as, 200 or 404.


            BufferedReader in = new BufferedReader(new InputStreamReader(con.getInputStream())); //
Initialized the BufferReader.
            String inputLine;


            while ((inputLine = in.readLine()) != null) {
                this.response = null;
            this.response = new StringBuffer();
            this.response.append(inputLine); // Append the string to the response.
          }

          JSONArray jSONArray = new JSONArray(response.toString());
          for (int count = 0; count < jSONArray.length(); count++) {

            User s = new User();
            JSONObject myResponse = jSONArray.getJSONObject(count);
            s.setUsername(myResponse.getString("username"));
            s.setPassword(myResponse.getString("password"));
            list.add(s);
          }
          in.close(); // Buffer is closed.

        } catch (Exception e) {
          e.printStackTrace();
        }

        return list;
      }

      @Override
      public List<Sensor> updateSensor(ArrayList<Sensor> array) throws RemoteException {
//        System.out.println("at update method");
        Random randomGenerator = new Random();//generate random no
```

```
int co2 = randomGenerator.nextInt(10) + 1; //generating random no between 1 to 10
System.out.println("co2"+co2);
int smoke = randomGenerator.nextInt(10) + 1;

int index = randomGenerator.nextInt(array.size());
Sensor sensor = array.get(index);
String mid = sensor.getMid();
sensor.setCo2Level(String.valueOf(co2));
sensor.setCo2Level(String.valueOf(smoke));

array.set(index, sensor);

String POST_URL = "http://localhost:5000/firesensor/update/"+mid;
System.out.println(POST_URL);

try {
  URL obj = new URL(POST_URL);
  HttpURLConnection cont = (HttpURLConnection) obj.openConnection();

  cont.setDoOutput(true);

  cont.setDoInput(true);
  cont.setRequestProperty("Content-Type", "application/json");
  cont.setRequestProperty("Accept", "application/json");
  cont.setRequestMethod("POST");
```

```
            JSONObject obj1 = new JSONObject();//Adding values to the JSON object
            obj1.put("location", sensor.getLocation());
            obj1.put("smoke", sensor.getSmokeLevel());
            obj1.put("co2", sensor.getCo2Level());
            obj1.put("state", sensor.getStatus());

            OutputStreamWriter wr = new OutputStreamWriter(cont.getOutputStream());

             System.out.println("--------------------------");
            System.out.println(obj1.toString());
            wr.write(obj1.toString());
            wr.flush();

            try (BufferedReader br = new BufferedReader(new InputStreamReader(cont.getInputStream(),
"utf-8"))) {
                StringBuilder response = new StringBuilder();
                String responseLine = null;
                while ((responseLine = br.readLine()) != null) {
                    response.append(responseLine.trim());
                }

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

        } catch (Exception ex) {
            ex.printStackTrace();
        }


        return array;
    }





    @Override
    public void sendNotification(String msg) throws RemoteException {


        String POST_URL =
"http://localhost:5000/firesensor/send/kalpanisandupaba9510@gmail.com/"+msg;
        System.out.println(POST_URL);
        System.out.println(msg);
        try {
            URL obj = new URL(POST_URL);
            HttpURLConnection cont = (HttpURLConnection) obj.openConnection();


            cont.setDoOutput(true);


            cont.setDoInput(true);
            cont.setRequestProperty("Content-Type", "application/json");
            cont.setRequestProperty("Accept", "application/json");
            cont.setRequestMethod("POST");
            JSONObject obj1 = new JSONObject();

            obj1.put("dummy", "msg");//Adding values to the JSON object


            OutputStreamWriter wr = new OutputStreamWriter(cont.getOutputStream());




            wr.write(obj1.toString());

            wr.flush();


            try (BufferedReader br = new BufferedReader(new InputStreamReader(cont.getInputStream(),
"utf-8"))) {

                StringBuilder response = new StringBuilder();

                String responseLine = null;

                while ((responseLine = br.readLine()) != null) {

                    response.append(responseLine.trim());

                }


                System.out.println("com.ds.project.server.service.SensorServiceImpl.sendNotification()");

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

            }


        } catch (Exception ex) {

            ex.printStackTrace();

        }
```

```
    }
}
```