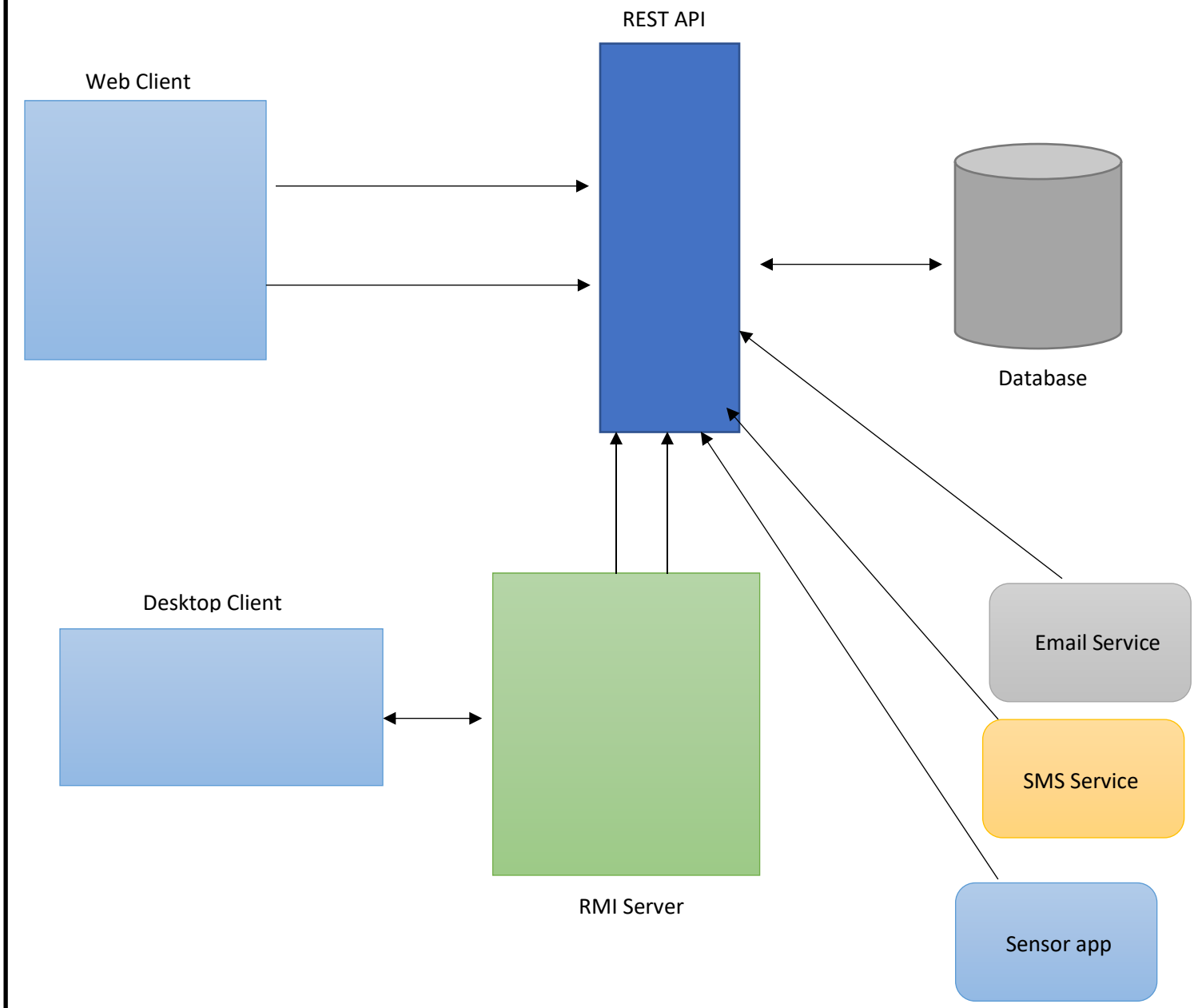
FIRE ALARM MONITORING SYSTEM

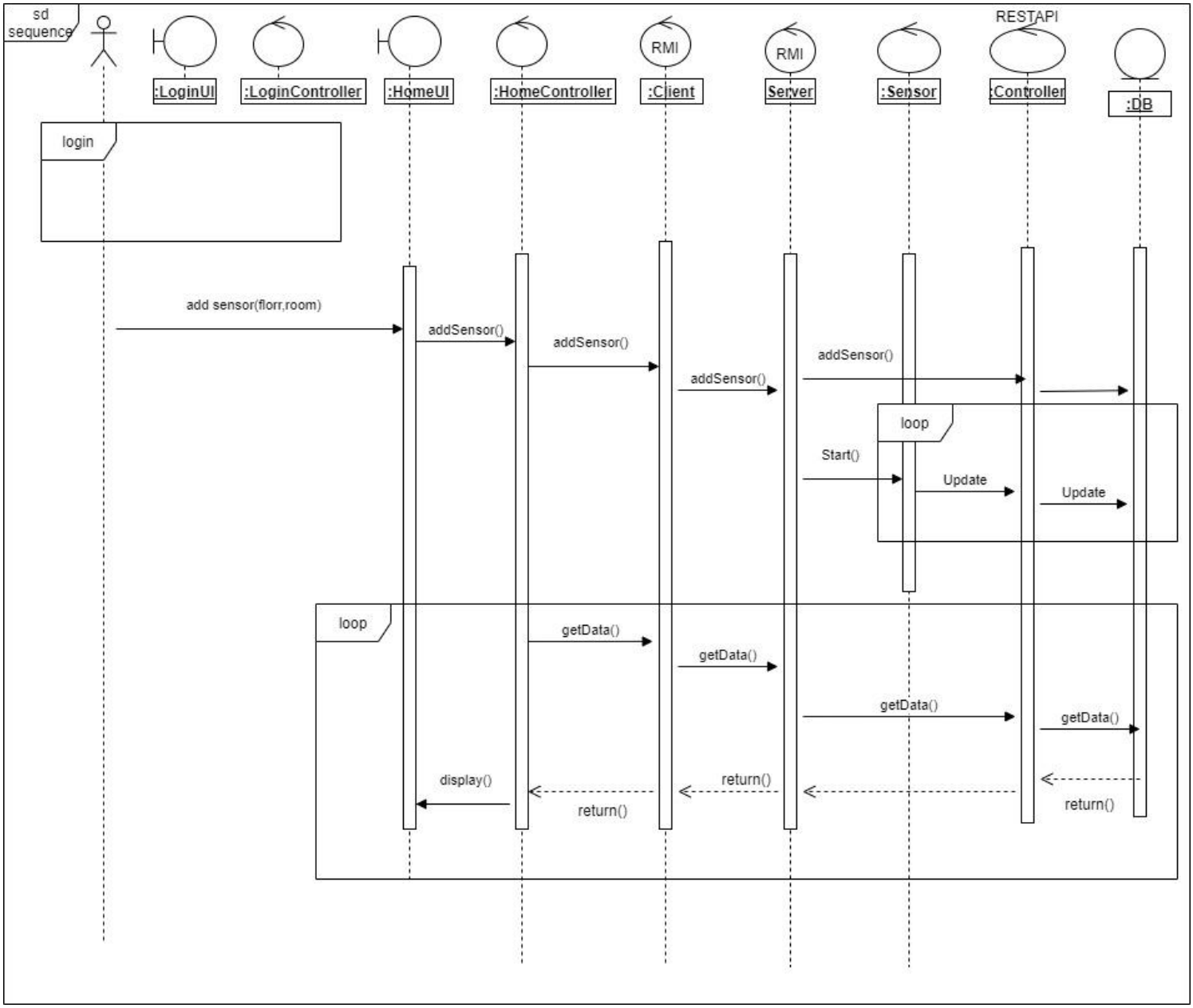
ASSIGNMENT 2-REST API

Table of Content.

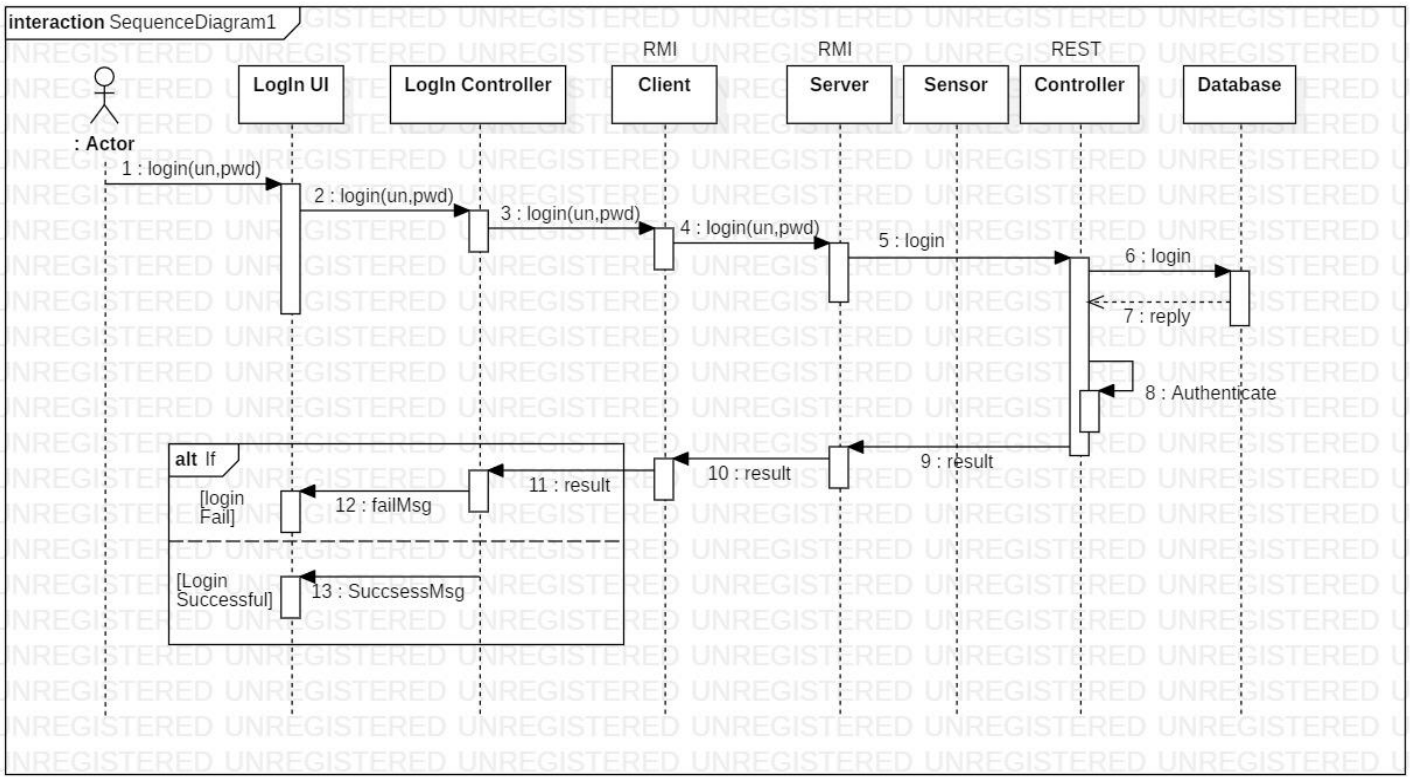
High level Architecture diagram for Fire alarm System



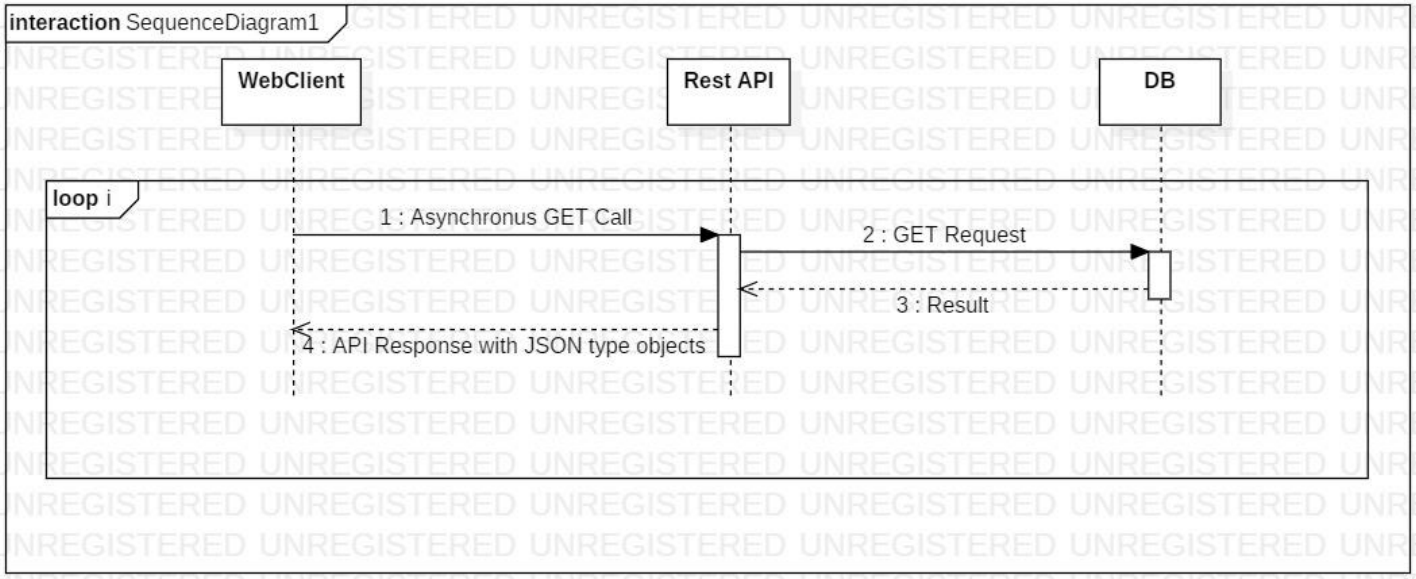
Sequence Diagram for Desktop Client Application



Login


interaction SequenceDiagram1
RMI
RMI
REST
LogIn UI
LogIn Controller
Client
Server
Sensor
Controller
Database
: Actor
1 : login(un,pwd)
2 : login(un,pwd)
3 : login(un,pwd)
4 : login(un,pwd)
5 : login
6 : login
7 : reply
8 : Authenticate
9 : result
10 : result
11 : result
12 : failMsg
13 : SuccsessMsg
alt If
[login Fail]
[Login Successful]

Sequence Diagram for Web Client Application



Web Client.

index.js

```
import React from 'react';
import ReactDOM from 'react-dom';
import './index.css';
import AlarmView from './AlarmView';
import * as serviceWorker from './serviceWorker';

ReactDOM.render(
  <React.StrictMode>
      //Render the Alarm View
    <AlarmView />
  </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();
```

AlarmView.js

```
import React, {Component} from 'react';
import Table from 'react-bootstrap/Table';
import './App.css';

class AlarmView extends Component {
    state = {
        isloading : true,
        alarms : []
    }

    //---Asynchronous call to REST API
    async componentDidMount(){
        const url = "/getAll";
        const response = await fetch(url);
        const data = await response.json();
        this.setState({alarms :data, isloading:false});
        //console.log(data);
    }


    render() {
```

```
        //---Check the connection
        const {alarms,isloading} = this.state;
        if(isloading)
            return (<div>Loading.....</div>);


        //---Display the fetch data through the API
        return (
            <div>
                <br/>
               <h2 className="heading">...Fire Alarm System...</h2>
                <br/><br/>
                <Table striped bordered hover className="alarm">
                    <thead>
                    <tr>
                        <th>Floor Number</th>
                        <th>Room Number</th>
                        <th>Smoke Level</th>
                        <th>CO2 Level</th>
                    </tr>
                    </thead>
                    <tbody>
                    {
                        //Dispaly all data and change the row color if smokelevel and co2level is
greater than 5
                        alarms.map(alarm =>
                            <tr key = {alarm.sid} style= {{backgroundColor: (alarm.smokeLevel > 5 ||
alarm.co2Level > 5) ? "#FA8072" : "#90EE90"}}>
                                <td>{alarm.floorNum}</td>
                                <td>{alarm.roomNum}</td>
                                <td>{alarm.smokeLevel}</td>
                                <td>{alarm.co2Level}</td>
                            </tr>

                        )
                    }

                    </tbody>
                </Table>

            </div>


        );
    }
}

export default AlarmView;
```

package.json

```
{
  "name": "fire-alarm-web",
  "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",
    "bootstrap": "^4.4.1",
    "react": "^16.13.1",
    "react-bootstrap": "^1.0.1",
    "react-dom": "^16.13.1",
    "react-scripts": "3.4.1"
  },
  "scripts": {
    "start": "react-scripts start",
    "build": "react-scripts build",
    "test": "react-scripts test",
    "eject": "react-scripts eject"
  },
  "proxy": "http://localhost:8080",
  "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"
    ]
  }
}
```

Rest API.

SensorController.java

```
@RestController
public class SensorController {

   @Autowired
   private SensorService sensorService;

   @Autowired
   private UserService userService;

   @RequestMapping("/create")
   public String create(@RequestParam int floorNum, @RequestParam int roomNum, @RequestParam int
sid,@RequestParam int smokeLevel,@RequestParam int co2Level) {
      Sensor p = sensorService.create(floorNum, roomNum, sid,smokeLevel,co2Level);
      return p.toString();
   }

   @RequestMapping("/get")
   public Sensor getSensor(@RequestParam int sid) {
      return sensorService.getBySid(sid);
   }

   @RequestMapping("/getAll")
   public List<Sensor> getAll(){
      return sensorService.getAll();
   }

   @RequestMapping("/update")
   public String update(@RequestParam int floorNum, @RequestParam int roomNum, @RequestParam int sid) {
      Sensor p = sensorService.update(floorNum, roomNum, sid);
      return p.toString();
   }

   @RequestMapping("/updateData")
   public String updateData(@RequestParam int sid,@RequestParam int smokeLevel,@RequestParam int
co2Level) {
      Sensor p = sensorService.updateData(sid,smokeLevel,co2Level);
      return p.toString();
   }

   @RequestMapping("/delete")
   public String delete(@RequestParam int sid) {
      sensorService.delete(sid);
      return "Deleted "+sid;
   }

   @RequestMapping ("/deleteAll")
   public String deleteAll() {
```

```
      sensorService.deleteAll();
      return "Deleted all records";
   }

   @RequestMapping("/createUser")
   public String createUser(@RequestParam String username, @RequestParam String password) {
      User p = userService.createUser(username,password);
      return p.toString();
   }

   @RequestMapping("/login")
public boolean login(@RequestParam String username, String password) {
      return userService.getByUsernameAndPassword(username,password);
   }
```

SensorRepository.java

```
@Repository
public interface SensorRepository extends MongoRepository<Sensor, String>{
   Sensor findBySid(int sid);
}
```

Sensor.java

```
@Document
public class Sensor {
   @Id
   String id;
   int floorNum;
   int roomNum;
   int    sid;
   int smokeLevel;
   int co2Level;

   public Sensor(int floorNum, int roomNum, int sid,int smokeLevel,int co2Level) {
      this.floorNum = floorNum;
      this.roomNum  = roomNum;
      this.sid       = sid;
      this.smokeLevel = smokeLevel;
      this.co2Level = co2Level;

   }

   //setters getters

   public String toString() {
      return "Sensor Floor Number:"+floorNum+" Room Number:"+roomNum+" SensorID:"+sid;
   }
}
```

Server

Server.java

```
public class Server extends UnicastRemoteObject implements Service {
```

```
    private final OkHttpClient httpClient = new OkHttpClient();
    private int sensorCount = 0;

    public Server() throws RemoteException {
        super();
    }

    public synchronized int getCount() throws RemoteException{
        return sensorCount;
    }

    @Override
    public void add(String sid,String roomNum,String floorNum) throws RemoteException {
        try {
            addSensor(sid,roomNum,floorNum);
            System.out.println("Sensor added");
            sensorCount++;
            Timer time = new Timer();
            SensorData st = new SensorData(sid);// start sensor dummy
            time.schedule(st, 0, 10000);// update sensor data every 10 seconds
        } catch (Exception e) {
            e.printStackTrace();
        }
    }

    @Override
    public void update(String sid, String roomNum, String floorNum) throws RemoteException {
        try {
            updateSensor(sid,roomNum,floorNum);
        } catch (Exception e) {
            e.printStackTrace();
        }
    }

    @Override
    public void delete(String sid) throws RemoteException {
        try {
            removeSensor(sid);
            System.out.println("Sensor deleted");
            if (sensorCount != 0)
                sensorCount--;
        } catch (Exception e) {
            e.printStackTrace();
        }
    }

    @Override
    public List<Sensor> getAll() throws RemoteException {
        List<Sensor> list = null;
        try {
            list = getSensors();
        } catch (IOException e) {
            e.printStackTrace();
        }
        return list;
    }
    //-------------------------- main method -----------------------------------------------------
    public static void main(String[] args){

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

        try{
```

```
            Server svr = new Server();

            Registry registry = LocateRegistry.getRegistry();
            registry.bind("SensorService", svr);

            System.out.println ("Service started....");

        }
        catch(RemoteException re){
            System.err.println(re.getMessage());
        }
        catch(AlreadyBoundException abe){
            System.err.println(abe.getMessage());
        }
    }
    //------------------------------ add sensor ----------------------------------------------------
    public void addSensor(String sid,String roomNum,String floorNum) throws Exception{
        RequestBody formBody = new FormBody.Builder()
                .add("sid",sid)
                .add("roomNum",roomNum)
                .add("floorNum",floorNum)
                .add("smokeLevel", String.valueOf(0))
                .add("co2Level", String.valueOf(0))
                .build();

        Request request = new Request.Builder()
                .url("http://localhost:8080/create")
                .addHeader("Content-Type", "application/json")
                .post(formBody)
                .build();

        try (Response response = httpClient.newCall(request).execute()) {
            if (!response.isSuccessful()) throw new IOException("Unexpected code " + response);
            // Get response body
            //System.out.println(response.body().string());
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
    //--------------------- delete sensor ------------------------------------------------------------
    public void removeSensor(String sid) throws Exception {

        RequestBody formBody = new FormBody.Builder()
                .add("sid", sid)
                .build();

        Request request = new Request.Builder()
                .url("http://localhost:8080/delete")
                .addHeader("Content-Type", "application/json")
                .post(formBody)
                .build();

        try (Response response = httpClient.newCall(request).execute()) {
            if (!response.isSuccessful()) throw new IOException("Unexpected code " + response);
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
    //--------------------------- update sensor ------------------------------------------------------
    public void updateSensor(String sid,String fno,String rno) throws Exception {

        RequestBody formBody = new FormBody.Builder()
                .add("sid", sid)
```

```
                .add("floorNum", fno)
                .add("roomNum", rno)
                .build();

        Request request = new Request.Builder()
                .url("http://localhost:8080/update")
                .addHeader("Content-Type", "application/json")
                .post(formBody)
                .build();

        try (Response response = httpClient.newCall(request).execute()) {
            if (!response.isSuccessful()) throw new IOException("Unexpected code " + response);
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
    //---------------------------------- get all ----------------------------------------------------
    public List<Sensor> getSensors() throws IOException {
        Request request = new Request.Builder()
                .url("http://localhost:8080/getAll")
                .build();

        ObjectMapper mapper = new ObjectMapper();

        try (Response response = httpClient.newCall(request).execute()) {
            String lst = response.body().string();
            List<Sensor> list = mapper.readValue(lst, new TypeReference<List<Sensor>>() {});

            return list;
        }
    }

}
```

Service.java

```
public interface Service extends Remote {
    void add(String sid,String roomNum,String floorNum) throws RemoteException;
    void update(String sid,String roomNum,String floorNum) throws RemoteException;
    void delete(String sid) throws RemoteException;
    List<Sensor> getAll() throws RemoteException;
    int getCount() throws RemoteException;
}
```

SensorData.java

```
public class SensorData extends TimerTask {

    private final OkHttpClient httpClient = new OkHttpClient();
    int smokeLevel =0;
    int co2Level =0;
    Random r1 = new Random();
    Random r2 = new Random();
    private String sid;

    public SensorData(String sid){
        this.sid = sid;
    }
```

```
    //generate random number between 1-10 and update
    @Override
    public void run() {

        smokeLevel = r1.nextInt(10);
        co2Level = r2.nextInt(10);

        try {
            updateSensor(getSid(),smokeLevel,co2Level);
        } catch (Exception e) {
            e.printStackTrace();
        }

    }
    //---------------- update sensor data smoke level and co2 level-------------------------------------
    public void updateSensor(String sid,int smoke,int co2) throws Exception {

        RequestBody formBody = new FormBody.Builder()
                .add("sid", sid)
                .add("smokeLevel", String.valueOf(smoke))
                .add("co2Level", String.valueOf(co2))
                .build();

        Request request = new Request.Builder()
                .url("http://localhost:8080/updateData")
                .addHeader("Content-Type", "application/json")
                .post(formBody)
                .build();
        try (Response response = httpClient.newCall(request).execute()) {
            if (!response.isSuccessful()) throw new IOException("Unexpected code " + response);
            // Get response body
            //System.out.println(response.body().string());
        } catch (IOException e) {
            e.printStackTrace();
        }

    }
    public String getSid(){return this.sid;}
}
```

DesktopClient

Client.java

```
public class Client {

    Service service;

    public Client(){
        System.setProperty("java.security.policy", "file:allowall.policy");
        try {
            service =  (Service) Naming.lookup("//localhost/SensorService");
        } catch (NotBoundException e) {
            e.printStackTrace();
        } catch (MalformedURLException e) {
            e.printStackTrace();
        } catch (RemoteException e) {
            e.printStackTrace();
        }
    }
```

```
    //adding new sensor
    public void add(String sid,String roomNum,String floorNum){
        try {
            service.add(sid,roomNum,floorNum);
        } catch (RemoteException ex) {
            System.err.println(ex.getMessage());
        }
    }
    //update existing sensor details
    public void update(String sid,String roomNum,String floorNum){
        try {
            service.update(sid,roomNum,floorNum);
        } catch (RemoteException ex) {
            System.err.println(ex.getMessage());
        }
    }
    //delete existing sensors
    public void delete(String sid){
        try {
            service.delete(sid);
        } catch (RemoteException ex) {
            System.err.println(ex.getMessage());
        }
    }
    //get all sensors list
    public List<Sensor> getAll(){
        List<Sensor> list = new ArrayList();
        try {
            list = service.getAll();
        } catch (RemoteException ex) {
            System.err.println(ex.getMessage());
        }
        return list;
    }
    //get number of connected sensors count
    public int getCount(){
        int count = 0;
        try {
            count = service.getCount();
        } catch (RemoteException e) {
            e.printStackTrace();
        }
        return count;
    }
}
```

HomeController.java

```
public class HomeController implements Initializable {

    //FXML attributes defining

    Client client;

    @Override
    public void initialize(URL location, ResourceBundle resources) {
        col_sid.setCellValueFactory(cellData -> cellData.getValue().getSidProperty().asString());
        col_floorNum.setCellValueFactory(cellData ->
cellData.getValue().getFloorNumProperty().asString());
        col_roomNum.setCellValueFactory(cellData ->
cellData.getValue().getRoomNumProperty().asString());
```

```
        col_smokeLevel.setCellValueFactory(cellData ->
cellData.getValue().getSmokeLevelProperty().asString());
        col_co2Level.setCellValueFactory(cellData ->
cellData.getValue().getCo2LevelProperty().asString());

        client = new Client();

        Timer timer = new Timer();
        timer.schedule(new TimerTask() {
            @Override
            public void run() {
                populateTable();
            }
        }, 0, 15000);//update table every 15 seconds
        table.setRowFactory(tv -> {
            TableRow<Sensor> row = new TableRow<>();
            row.setOnMouseClicked(event -> {
                if (! row.isEmpty() && event.getButton()== MouseButton.PRIMARY
                        && event.getClickCount() == 1) {

                    Sensor s = row.getItem();
                    txFloorNum.setText(String.valueOf(s.getFloorNum()));
                    txRoomNum.setText(String.valueOf(s.getRoomNum()));
                    txSid.setText(String.valueOf(s.getSid()));
                }
            });
            return row ;
        });
        //change row color if smoke level or co2 level exceed level 5
        table.setRowFactory(tv -> new TableRow<Sensor>() {
            @Override
            protected void updateItem(Sensor s, boolean empty) {
                super.updateItem(s, empty);
                if (s == null)
                    setStyle("");
                else if (s.getCo2Level() > 5 || s.getSmokeLevel()>5)
                    setStyle("-fx-background-color: #FF4658;");
                else
                    setStyle("");
            }
        });
    }
    //load sensor details into table
    private void populateTable(){
//        List<Sensor> sList = client.getAll();
//        ObservableList<Sensor> observableArrayList = FXCollections.observableArrayList(sList);
//        table.setItems(observableArrayList);

        RestManager restManager = new RestManager();
        List<Sensor> slist = null;
        try {
            slist = restManager.getAll();
        } catch (IOException e) {
            e.printStackTrace();
        }
        ObservableList<Sensor> observableArrayList = FXCollections.observableArrayList(slist);
        table.setItems(observableArrayList);
        txCount.setText(String.valueOf(client.getCount()));
    }
    //remove sensor button
    @FXML
    public void removeSensor(javafx.event.ActionEvent actionEvent) throws Exception {
        Sensor s = (Sensor) table.getSelectionModel().getSelectedItem();
```

```
        client.delete(String.valueOf(s.getSid()));
        clear();
        populateTable();
    }
    //update sensor details button
    @FXML
    public void updateSensor(javafx.event.ActionEvent actionEvent) throws Exception {
        Sensor s = (Sensor) table.getSelectionModel().getSelectedItem();
        client.update(String.valueOf(s.getSid()),txFloorNum.getText(),txRoomNum.getText());
        //populateTable();
        clear();
    }
    //add sensor button
    @FXML
    public void addSensor(javafx.event.ActionEvent actionEvent) throws Exception {
        client.add(txSid.getText(),txRoomNum.getText(),txFloorNum.getText());
        clear();
        populateTable();
    }
```