DS – SE3020

3rd Year, 1st Semester 2020

Assignment 2-Rest API

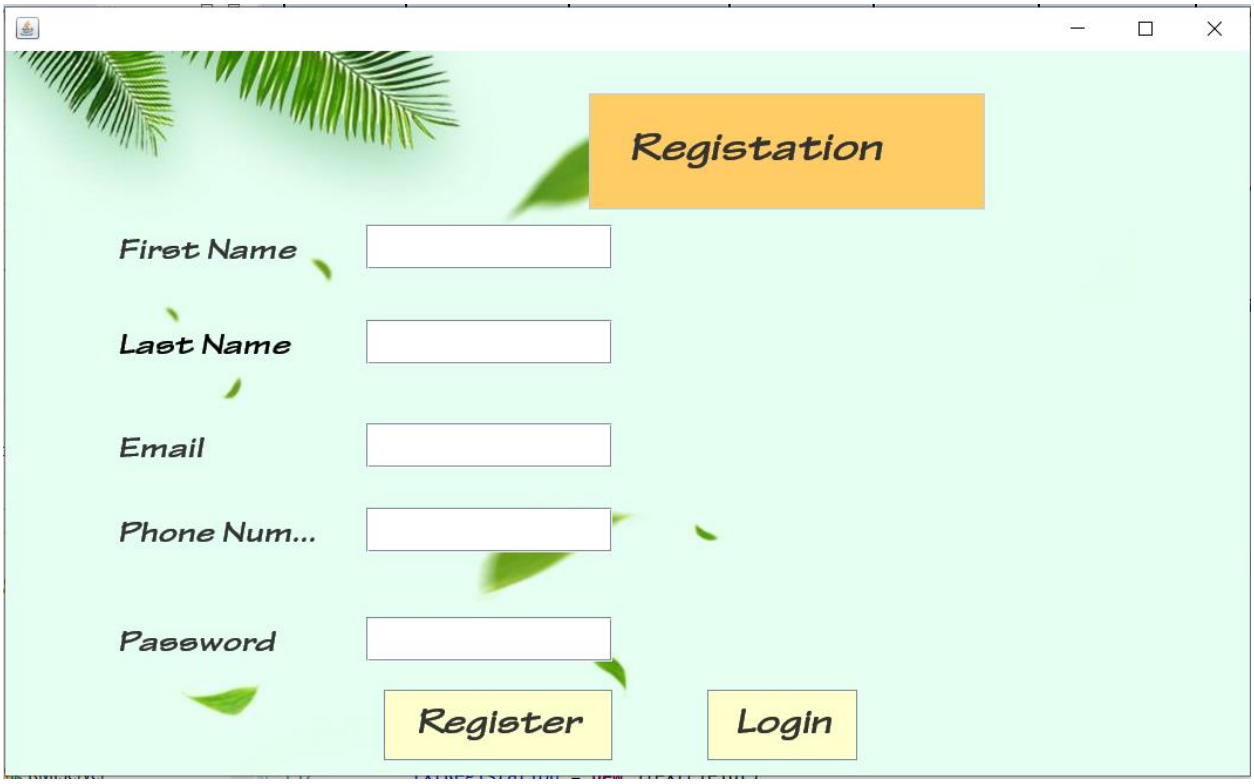
Group Project Submitted to
Sri Lanka Institute of Information Technology

Online Fire Alarm Monitoring System

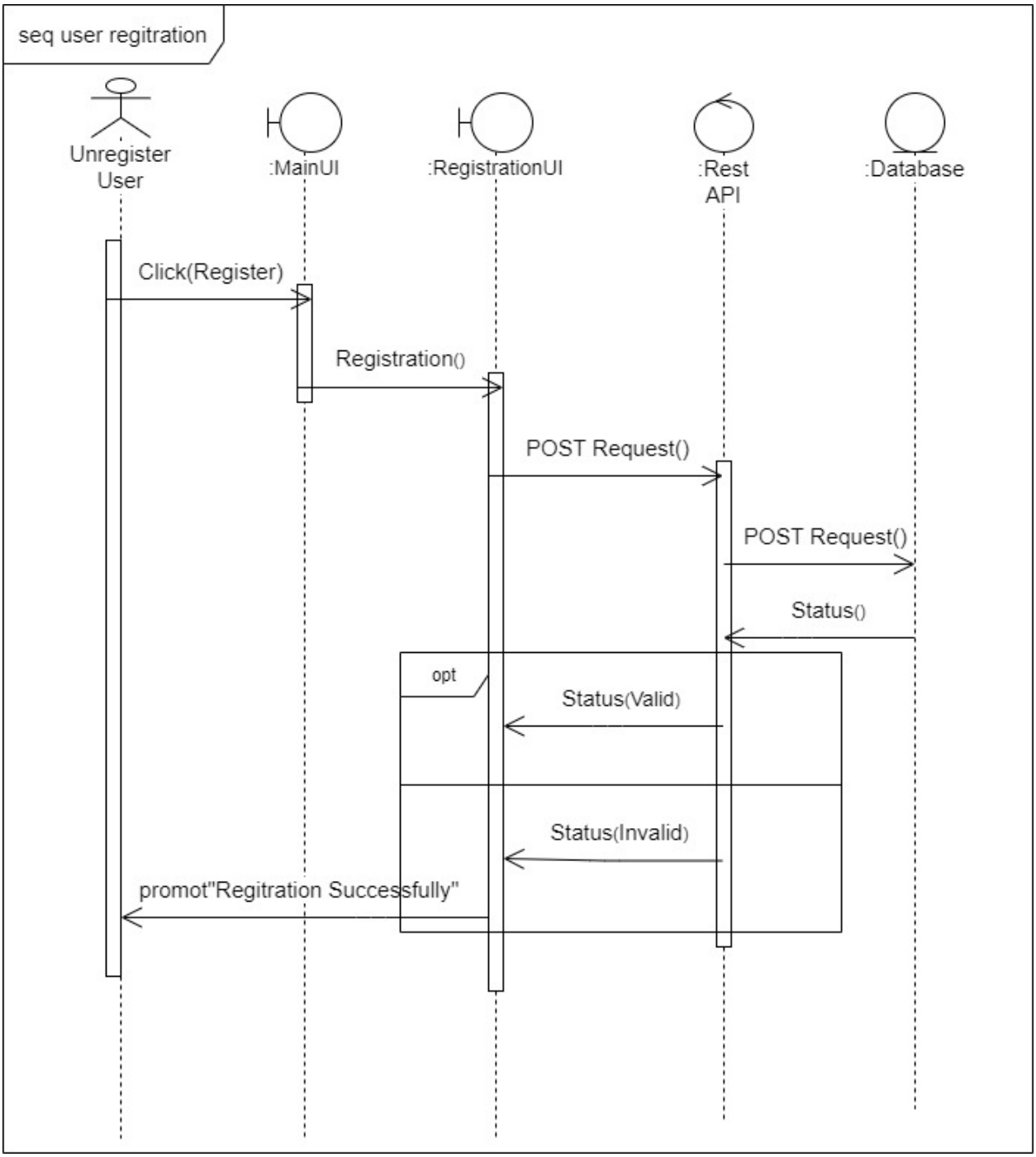
Contents

User Registration

First, unregister user can be register to the system. Then User can login to the system. Only User can register to the system using desktop application.

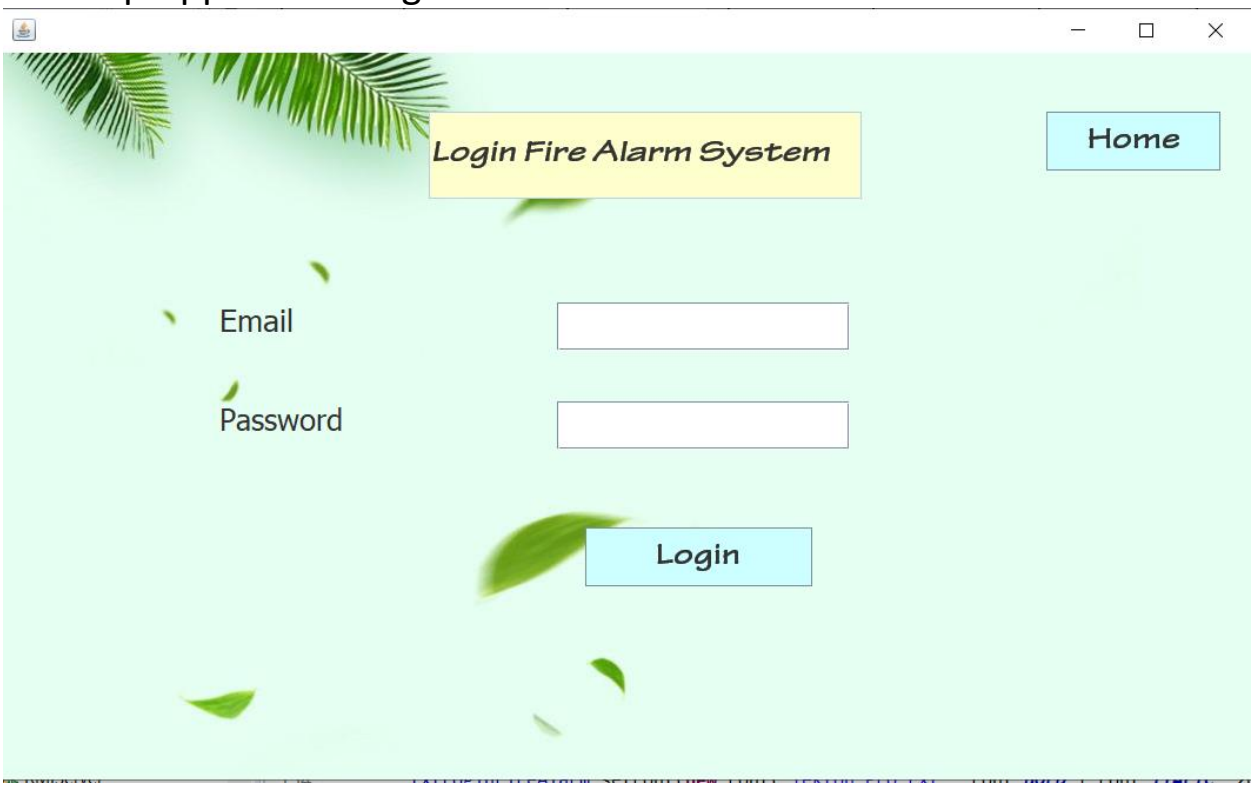


Registration sequence diagram (desktop application)

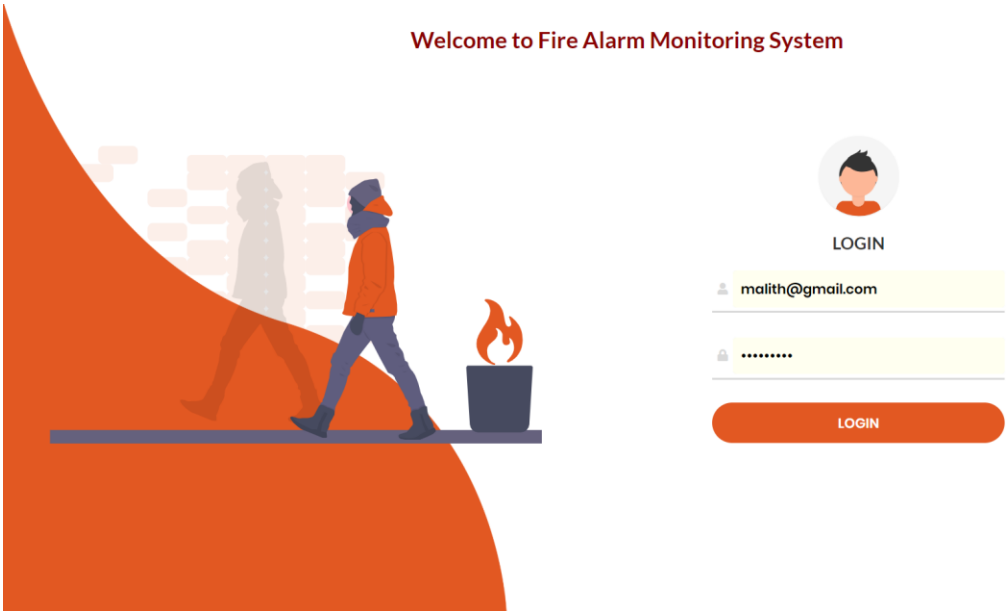


Login

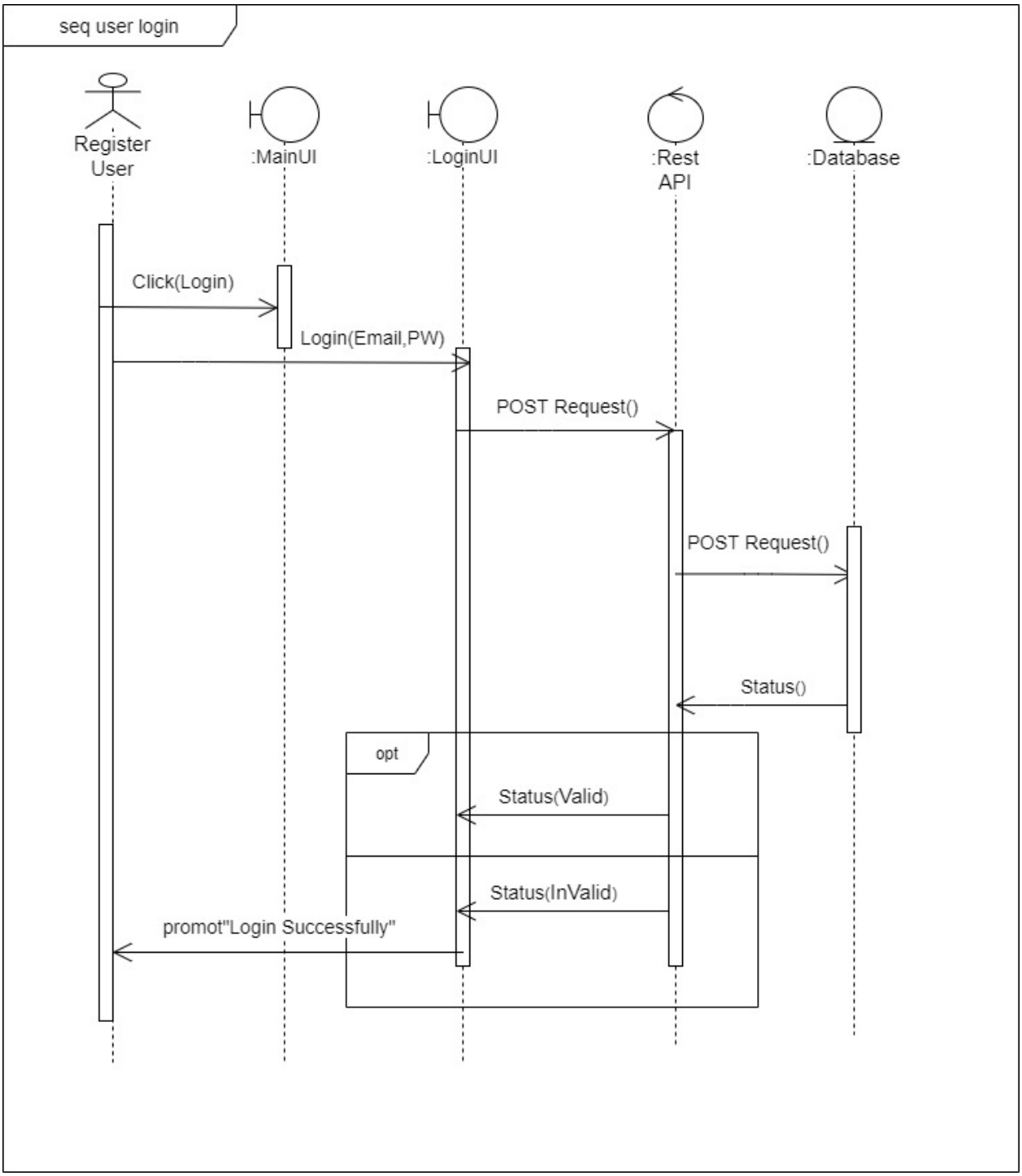
Desktop Application Login



Web Application Login

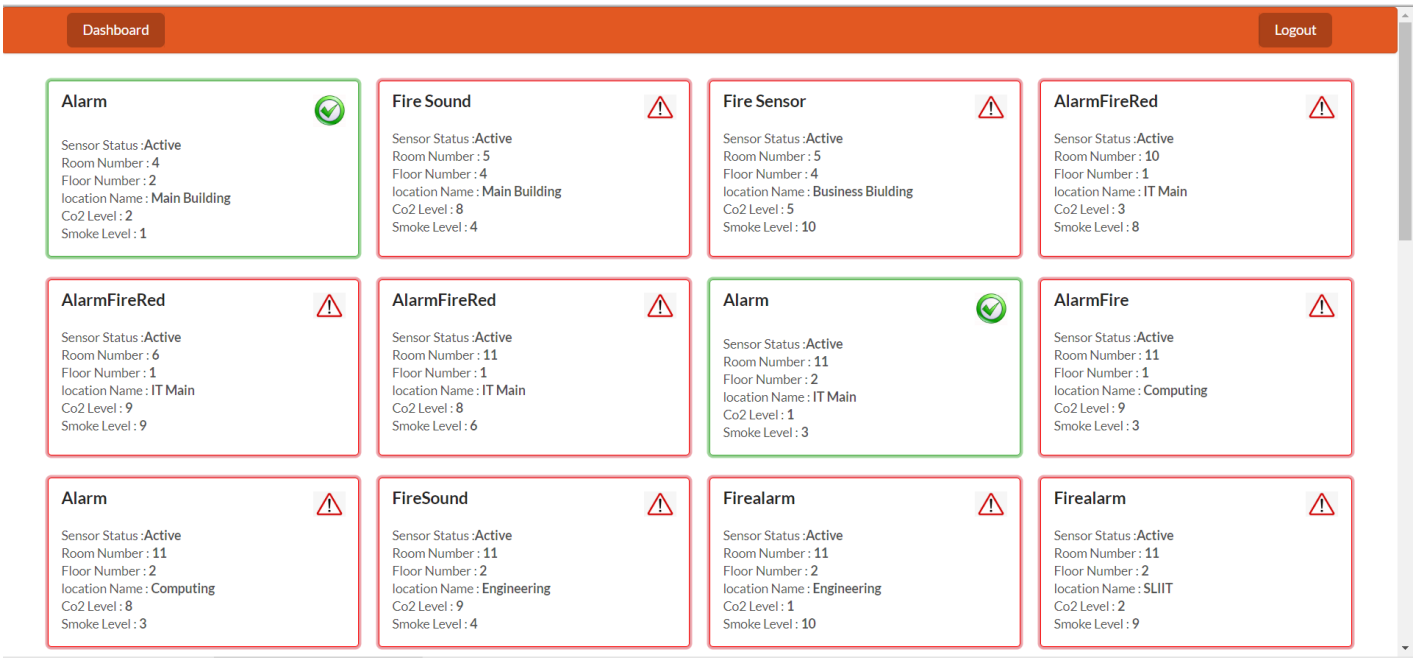


Then Register user/Admin can login to the system. User can login to the system using web application or desktop application.

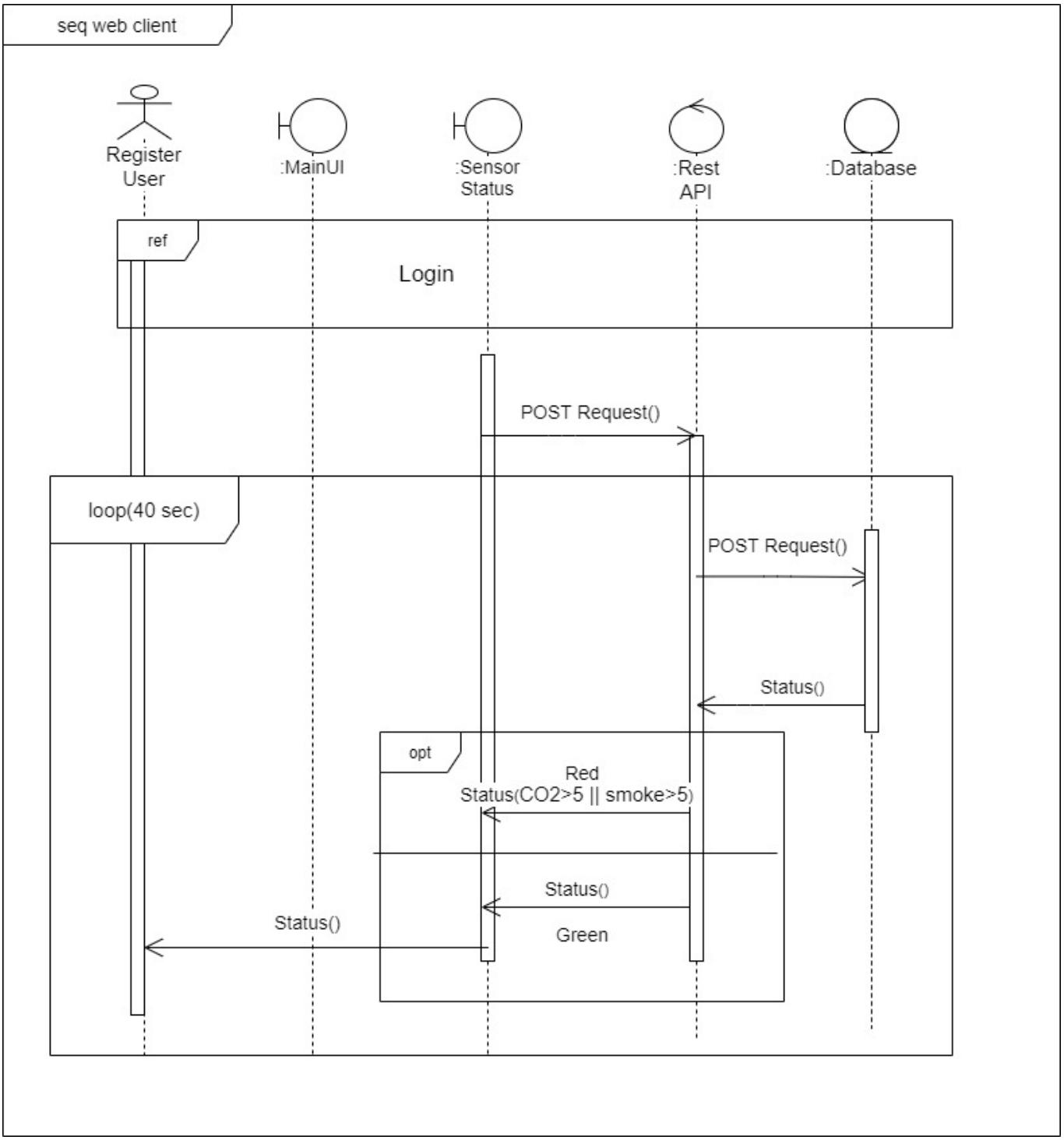


Web Application Shows Sensor Status

User login to the system in web application system shows sensors status. Red border and error icon show dangerous sensors. Green border and true icon show not dangerous sensors. User can see Web application refresh every 40 seconds. System develop an Asynchronous programming web client, using React.

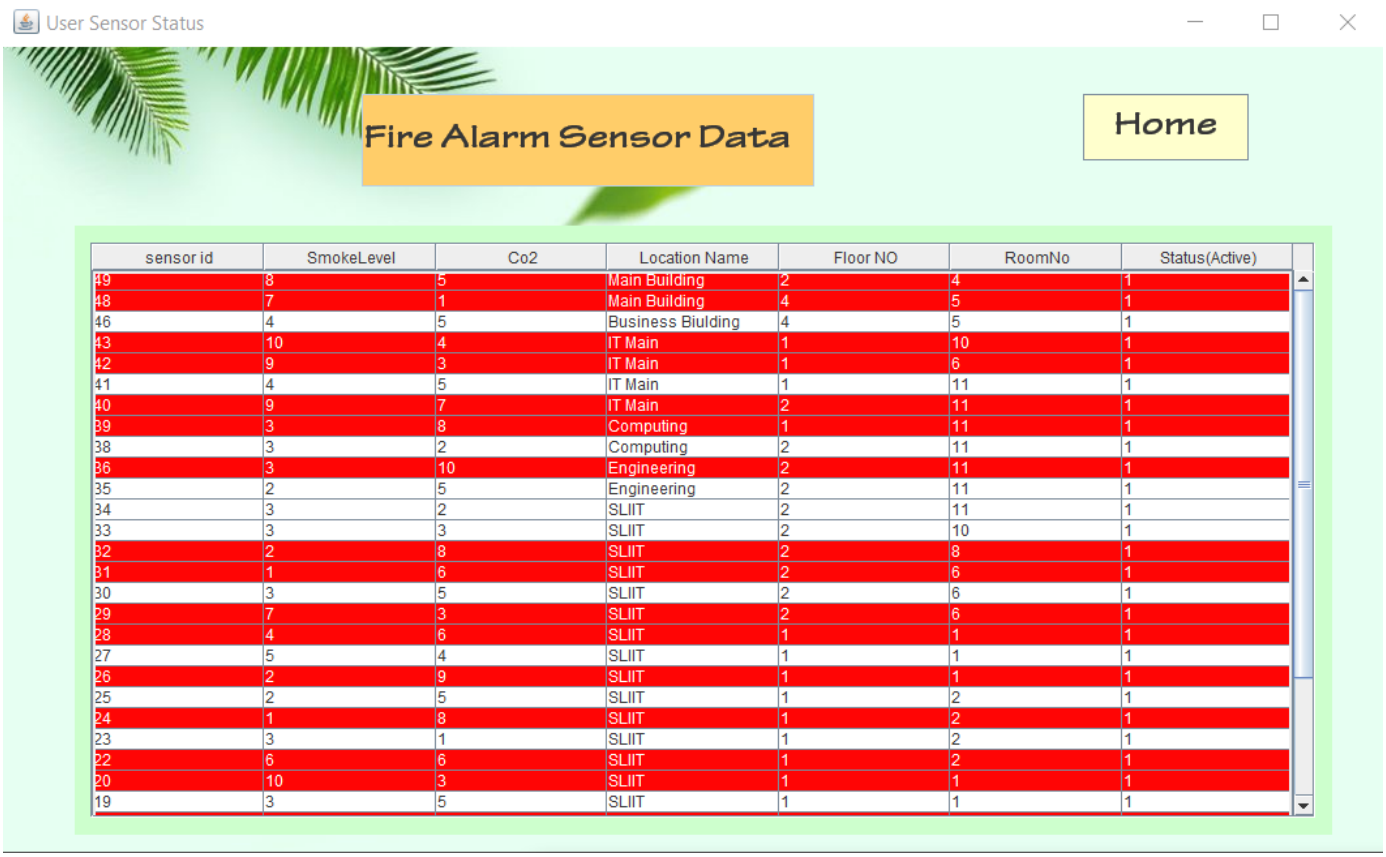


Sequence diagram web application



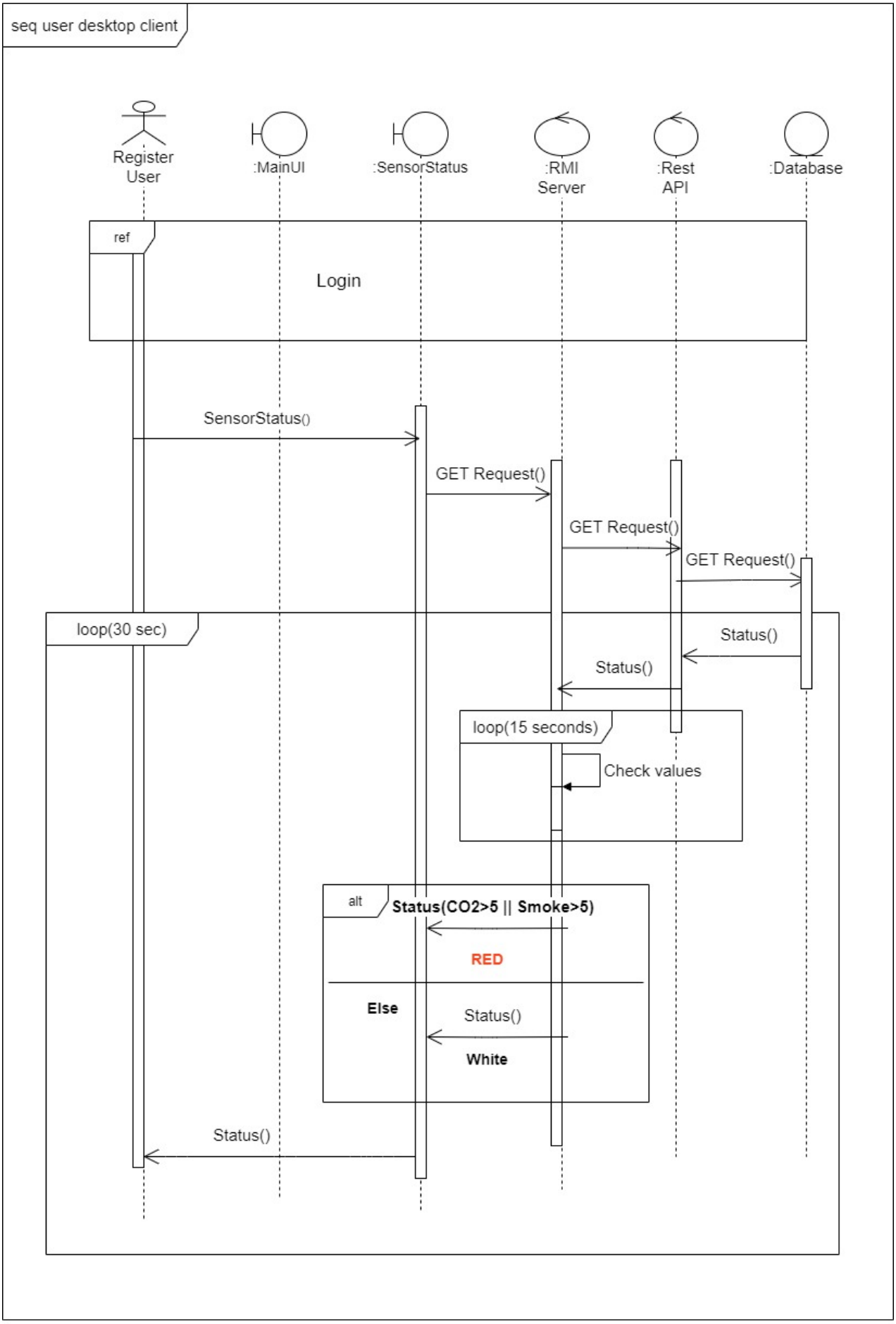
Desktop Application Shows Sensor Status

Desktop User can look sensor status. Desktop application refresh every 30 seconds. RMI server check values every 15 seconds. Sensor status co2 values greater than five or smoke value greater than five system show red color row. Sensor status values less than five or equal system shows white color row. User can identify dangerous sensors. System shows to web client, Desktop client can see the same status.



sensor id	SmokeLevel	Co2	Location Name	Floor NO	RoomNo	Status(Active)
49	8	5	Main Building	2	4	1
48	7	1	Main Building	4	5	1
46	4	5	Business Biulding	4	5	1
43	10	4	IT Main	1	10	1
42	9	3	IT Main	1	6	1
41	4	5	IT Main	1	11	1
40	9	7	IT Main	2	11	1
39	3	8	Computing	1	11	1
38	3	2	Computing	2	11	1
36	3	10	Engineering	2	11	1
35	2	5	Engineering	2	11	1
34	3	2	SLIIT	2	11	1
33	3	3	SLIIT	2	10	1
32	2	8	SLIIT	2	8	1
31	1	6	SLIIT	2	6	1
30	3	5	SLIIT	2	6	1
29	7	3	SLIIT	2	6	1
28	4	6	SLIIT	1	1	1
27	5	4	SLIIT	1	1	1
26	2	9	SLIIT	1	1	1
25	2	5	SLIIT	1	2	1
24	1	8	SLIIT	1	2	1
23	3	1	SLIIT	1	2	1
22	6	6	SLIIT	1	2	1
20	10	3	SLIIT	1	1	1
19	3	5	SLIIT	1	1	1

Sequence diagram desktop application


seq user desktop client
Register User
:MainUI
:SensorStatus
:RMI Server
:Rest API
:Database
ref
Login
SensorStatus()
GET Request()
GET Request()
GET Request()
loop(30 sec)
Status()
Status()
loop(15 seconds)
Check values
alt
Status(CO2>5 || Smoke>5)
RED
Else
Status()
White
Status()

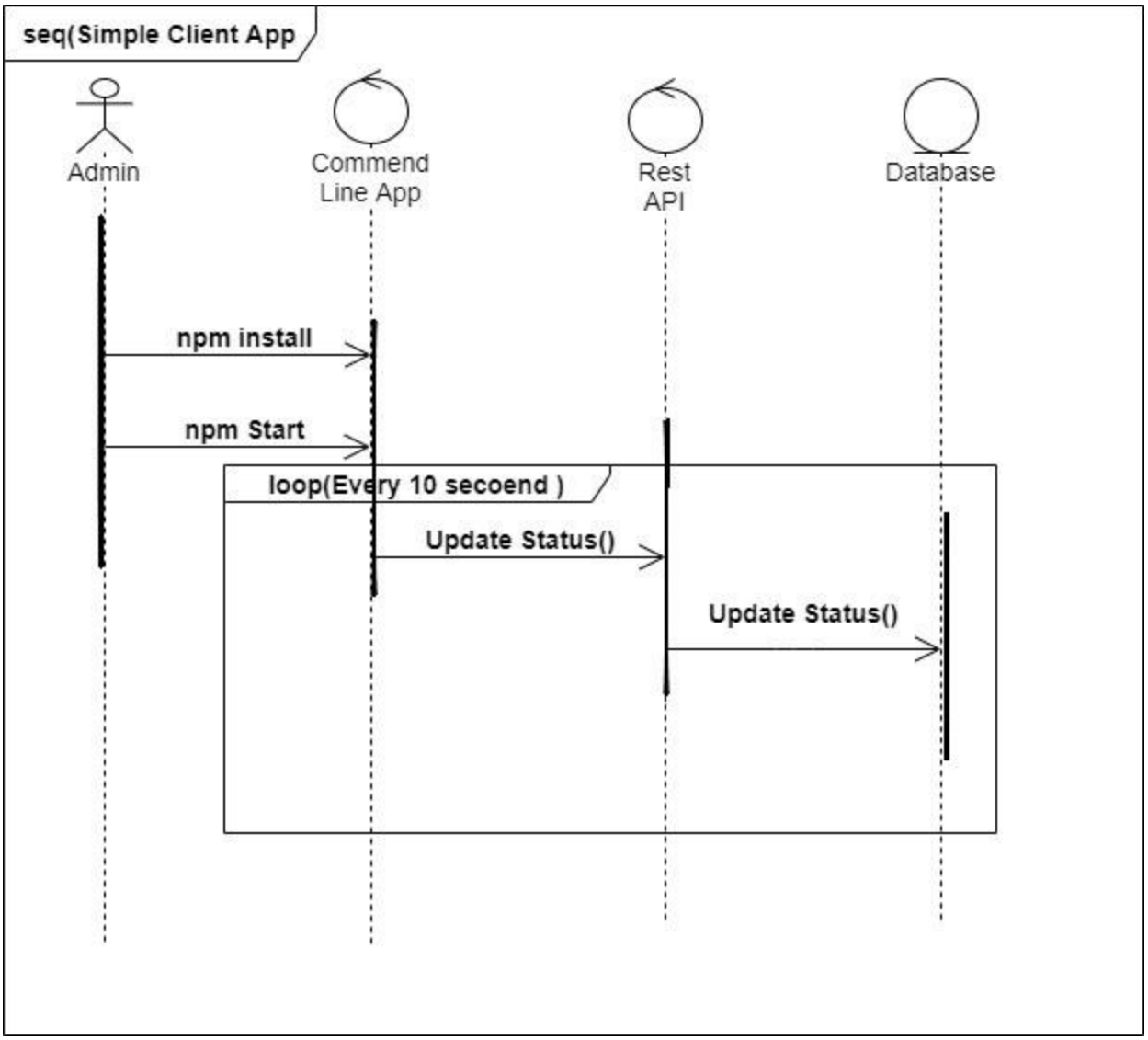
Simple client Application

```
PROBLEMS   OUTPUT   DEBUG CONSOLE   TERMINAL                                                    1: node

20-05-05 15:25:00','2020-05-05 15:25:00'),(NULL,15,9,4,'2020-05-05 15:25:00','2020-05-05 15:25:00'),(NULL,16,4,5,'2020-05-05 15:25:00','2020-05-05 15:25:00
'),(NULL,17,1,10,'2020-05-05 15:25:00','2020-05-05 15:25:00'),(NULL,18,4,8,'2020-05-05 15:25:00','2020-05-05 15:25:00'),(NULL,19,5,6,'2020-05-05 15:25:00',
'2020-05-05 15:25:00'),(NULL,20,4,7,'2020-05-05 15:25:00','2020-05-05 15:25:00'),(NULL,22,8,6,'2020-05-05 15:25:00','2020-05-05 15:25:00'),(NULL,23,1,5,'20
20-05-05 15:25:00','2020-05-05 15:25:00'),(NULL,24,2,6,'2020-05-05 15:25:00','2020-05-05 15:25:00'),(NULL,25,5,2,'2020-05-05 15:25:00','2020-05-05 15:25:00
'),(NULL,26,7,5,'2020-05-05 15:25:00','2020-05-05 15:25:00'),(NULL,27,2,9,'2020-05-05 15:25:00','2020-05-05 15:25:00'),(NULL,28,3,8,'2020-05-05 15:25:00','
2020-05-05 15:25:00'),(NULL,29,9,5,'2020-05-05 15:25:00','2020-05-05 15:25:00'),(NULL,30,4,8,'2020-05-05 15:25:00','2020-05-05 15:25:00'),(NULL,31,4,4,'202
0-05-05 15:25:00','2020-05-05 15:25:00'),(NULL,32,10,8,'2020-05-05 15:25:00','2020-05-05 15:25:00'),(NULL,33,6,5,'2020-05-05 15:25:00','2020-05-05 15:25:00
'),(NULL,34,5,9,'2020-05-05 15:25:00','2020-05-05 15:25:00'),(NULL,35,7,7,'2020-05-05 15:25:00','2020-05-05 15:25:00'),(NULL,36,6,4,'2020-05-05 15:25:00','
2020-05-05 15:25:00'),(NULL,38,3,5,'2020-05-05 15:25:00','2020-05-05 15:25:00'),(NULL,39,3,4,'2020-05-05 15:25:00','2020-05-05 15:25:00'),(NULL,40,6,5,'202
0-05-05 15:25:00','2020-05-05 15:25:00'),(NULL,41,3,4,'2020-05-05 15:25:00','2020-05-05 15:25:00'),(NULL,42,5,6,'2020-05-05 15:25:00','2020-05-05 15:25:00'
),(NULL,43,9,10,'2020-05-05 15:25:00','2020-05-05 15:25:00'),(NULL,46,10,7,'2020-05-05 15:25:00','2020-05-05 15:25:00'),(NULL,48,10,8,'2020-05-05 15:25:00'
,'2020-05-05 15:25:00'),(NULL,49,5,9,'2020-05-05 15:25:00','2020-05-05 15:25:00');
Sensor Data Updated
Executing (default): SELECT `id` AS `sensorId` FROM `Sensors` AS `Sensor` WHERE (`Sensor`.`deletedAt` IS NULL);
Executing (default): INSERT INTO `SensorData` (`id`,`sensorId`,`co2Level`,`smokeLevel`,`createdAt`,`updatedAt`) VALUES (NULL,11,2,8,'2020-05-05 15:25:10','
2020-05-05 15:25:10'),(NULL,12,3,9,'2020-05-05 15:25:10','2020-05-05 15:25:10'),(NULL,13,10,8,'2020-05-05 15:25:10','2020-05-05 15:25:10'),(NULL,14,4,4,'20
20-05-05 15:25:10','2020-05-05 15:25:10'),(NULL,15,8,5,'2020-05-05 15:25:10','2020-05-05 15:25:10'),(NULL,16,10,6,'2020-05-05 15:25:10','2020-05-05 15:25:1
0'),(NULL,17,5,3,'2020-05-05 15:25:10','2020-05-05 15:25:10'),(NULL,18,2,4,'2020-05-05 15:25:10','2020-05-05 15:25:10'),(NULL,19,8,6,'2020-05-05 15:25:10',
'2020-05-05 15:25:10'),(NULL,20,6,7,'2020-05-05 15:25:10','2020-05-05 15:25:10'),(NULL,22,1,10,'2020-05-05 15:25:10','2020-05-05 15:25:10'),(NULL,23,9,10,'
2020-05-05 15:25:10','2020-05-05 15:25:10'),(NULL,24,3,4,'2020-05-05 15:25:10','2020-05-05 15:25:10'),(NULL,25,8,3,'2020-05-05 15:25:10','2020-05-05 15:25:
10'),(NULL,26,5,4,'2020-05-05 15:25:10','2020-05-05 15:25:10'),(NULL,27,5,7,'2020-05-05 15:25:10','2020-05-05 15:25:10'),(NULL,28,9,3,'2020-05-05 15:25:10'
,'2020-05-05 15:25:10'),(NULL,29,9,8,'2020-05-05 15:25:10','2020-05-05 15:25:10'),(NULL,30,4,8,'2020-05-05 15:25:10','2020-05-05 15:25:10'),(NULL,31,7,9,'2
020-05-05 15:25:10','2020-05-05 15:25:10'),(NULL,32,2,10,'2020-05-05 15:25:10','2020-05-05 15:25:10'),(NULL,33,7,6,'2020-05-05 15:25:10','2020-05-05 15:25:
10'),(NULL,34,9,4,'2020-05-05 15:25:10','2020-05-05 15:25:10'),(NULL,35,7,9,'2020-05-05 15:25:10','2020-05-05 15:25:10'),(NULL,36,3,2,'2020-05-05 15:25:10'
,'2020-05-05 15:25:10'),(NULL,38,9,10,'2020-05-05 15:25:10','2020-05-05 15:25:10'),(NULL,39,9,2,'2020-05-05 15:25:10','2020-05-05 15:25:10'),(NULL,40,4,5,'
2020-05-05 15:25:10','2020-05-05 15:25:10'),(NULL,41,9,10,'2020-05-05 15:25:10','2020-05-05 15:25:10'),(NULL,42,8,1,'2020-05-05 15:25:10','2020-05-05 15:25
:10'),(NULL,43,8,4,'2020-05-05 15:25:10','2020-05-05 15:25:10'),(NULL,46,3,8,'2020-05-05 15:25:10','2020-05-05 15:25:10'),(NULL,48,2,5,'2020-05-05 15:25:10
','2020-05-05 15:25:10'),(NULL,49,6,7,'2020-05-05 15:25:10','2020-05-05 15:25:10');
Sensor Data Updated
```

This application can send fire alarm status to the Rest API for every 10 seconds. Application can send status for multiple sensors in the system. Application can be run multiple application when running in the system. Start the app “npm start”

Sequence diagram simple client application



RMI Server

Console

RMIServer [Java Application] C:\Program Files\Java\jre1.8.0

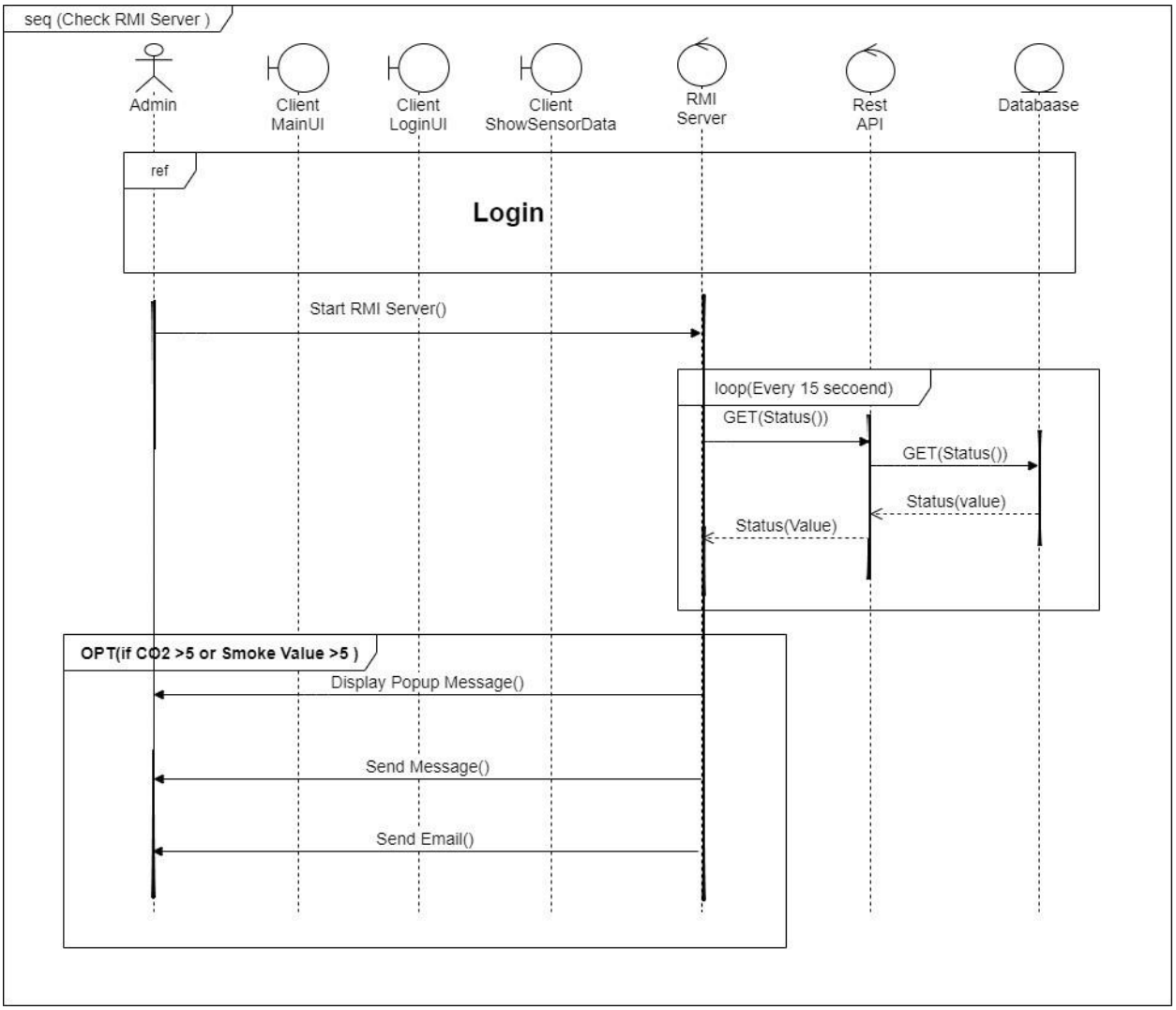
```
thread is running...
Server is right now...
Email Send
SMS Send
```

RMIServer.java

```
1  package Attendance;
2
3  import java.io.BufferedReader;
40
41 public class RMIServer extends Unicast
42     @Override
43     public void run() {
44         // TODO Auto-generated method
45         System.out.println("thread is
46         while (true) {
47             tableshow();
48 //          sms frame1 = new sms();
49 //          frame1.setVisible(true);
50
51             try {
52                 Thread.sleep(15000);
53             } catch (InterruptedExcept
54                 // TODO Auto-generatec
55                 e.printStackTrace();
56             }
57         }
58     };
59
60     protected RMIServer() throws Remot
61         super();
62         // TODO Auto-generated constru
```

RMI desktop client and RMI server as desktop application. RMI server connect REST API.RMI server send Email, SMS and alert message when co2 value >5 or smoke value >5.RMI server check sensor status every 15 seconds. API connects to the database.

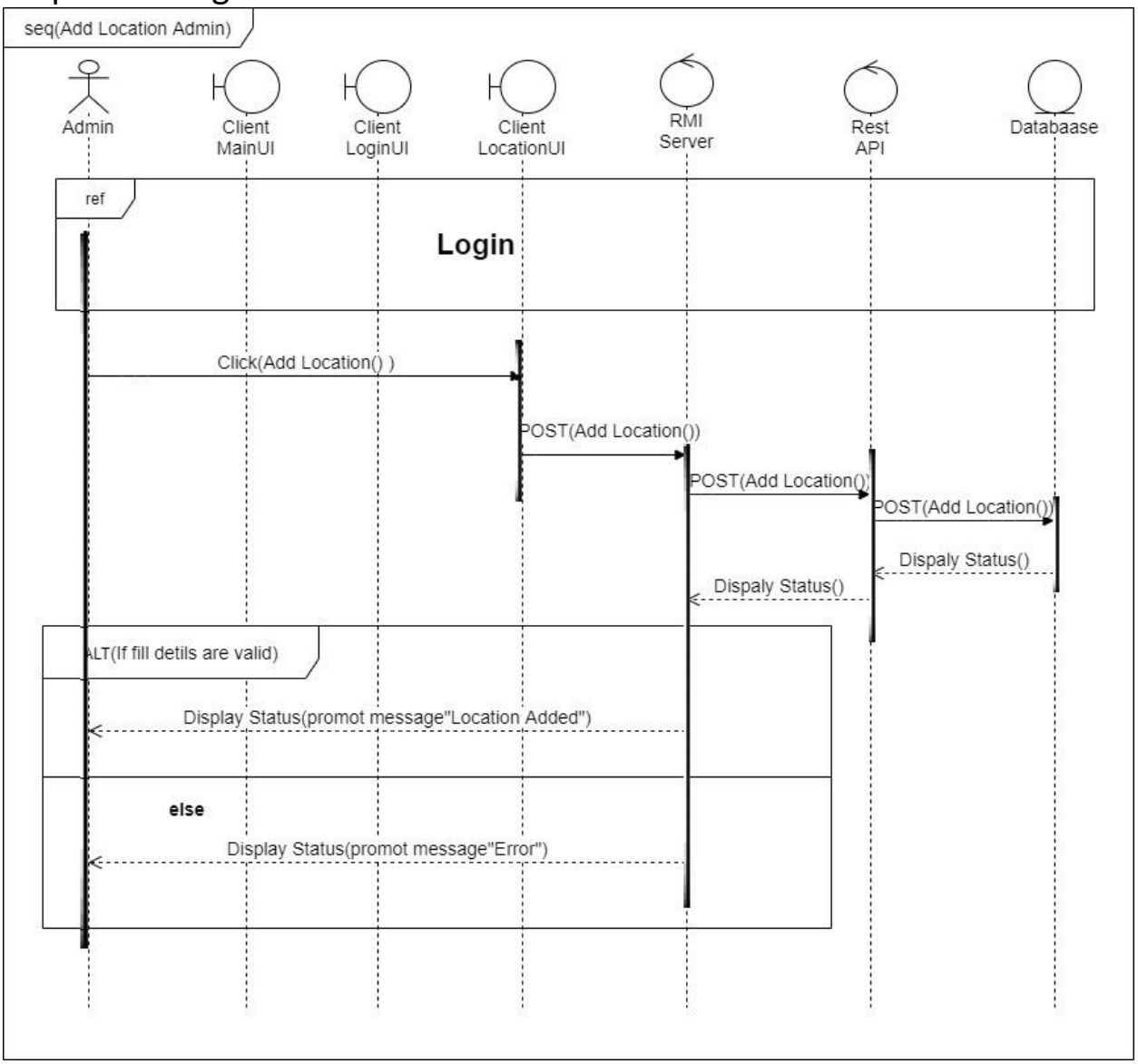
RMI server Sequence diagram



Add Location

Admin can add location name into the system. Only admin can work desktop application.

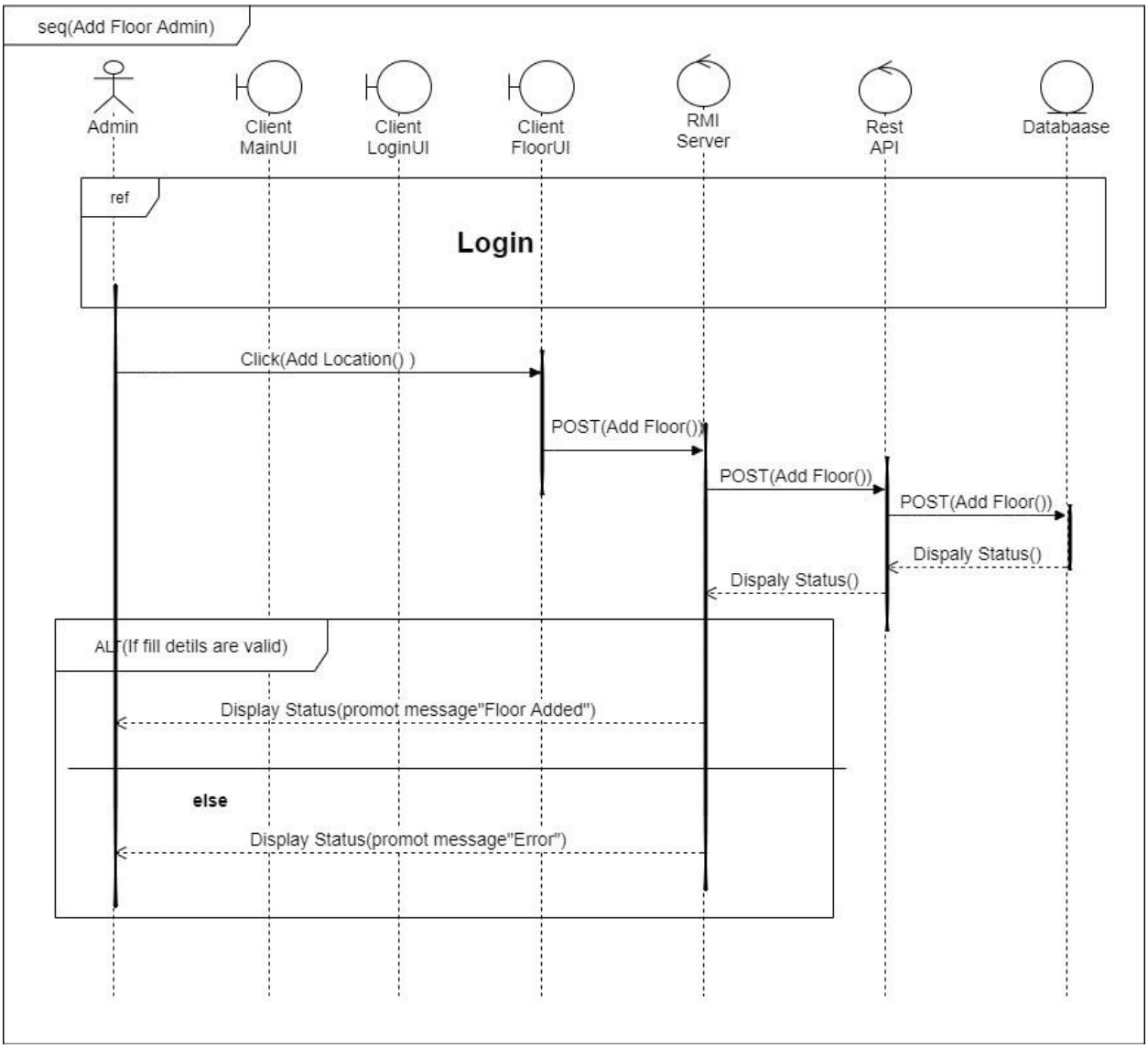
Sequence diagram admin add location



Add Floor

Admin can add floor details into the system. Location has no of floors.

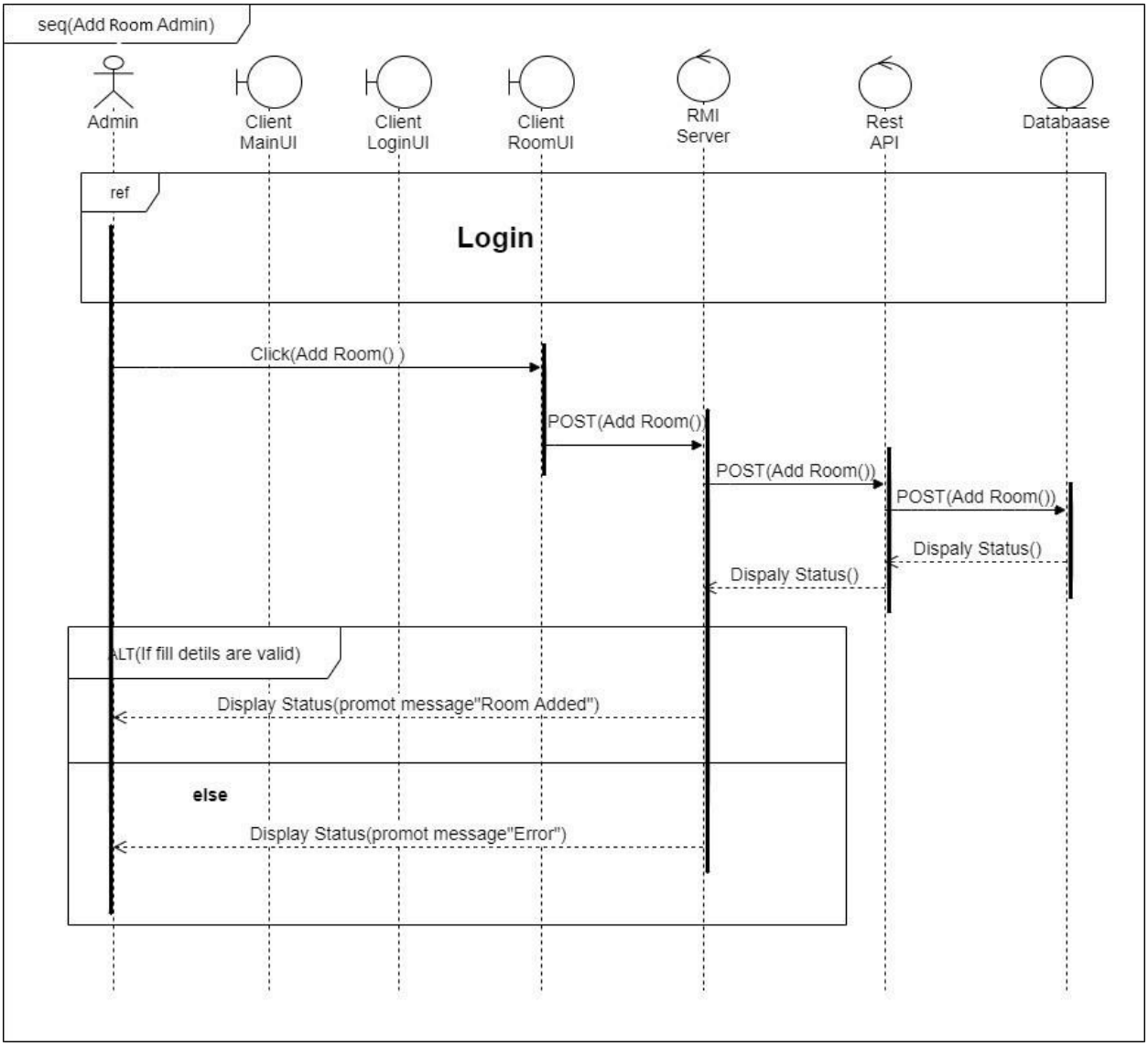
Sequence diagram admin add floors.



Add Room

Admin can add rooms into the system. Each floor has no of rooms.

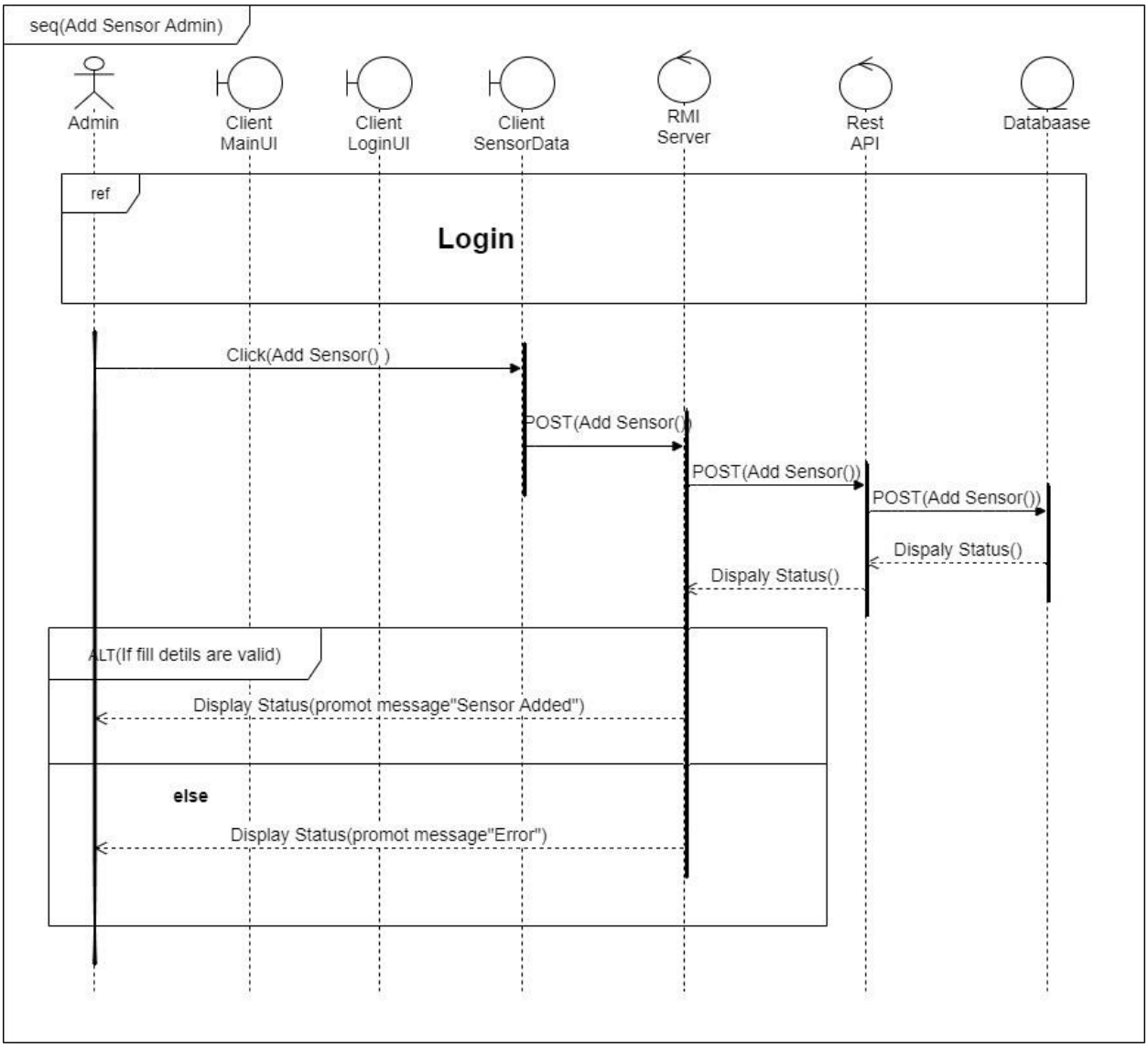
Sequence diagram admin add rooms.



Add sensors

Admin can add no of sensors each room.

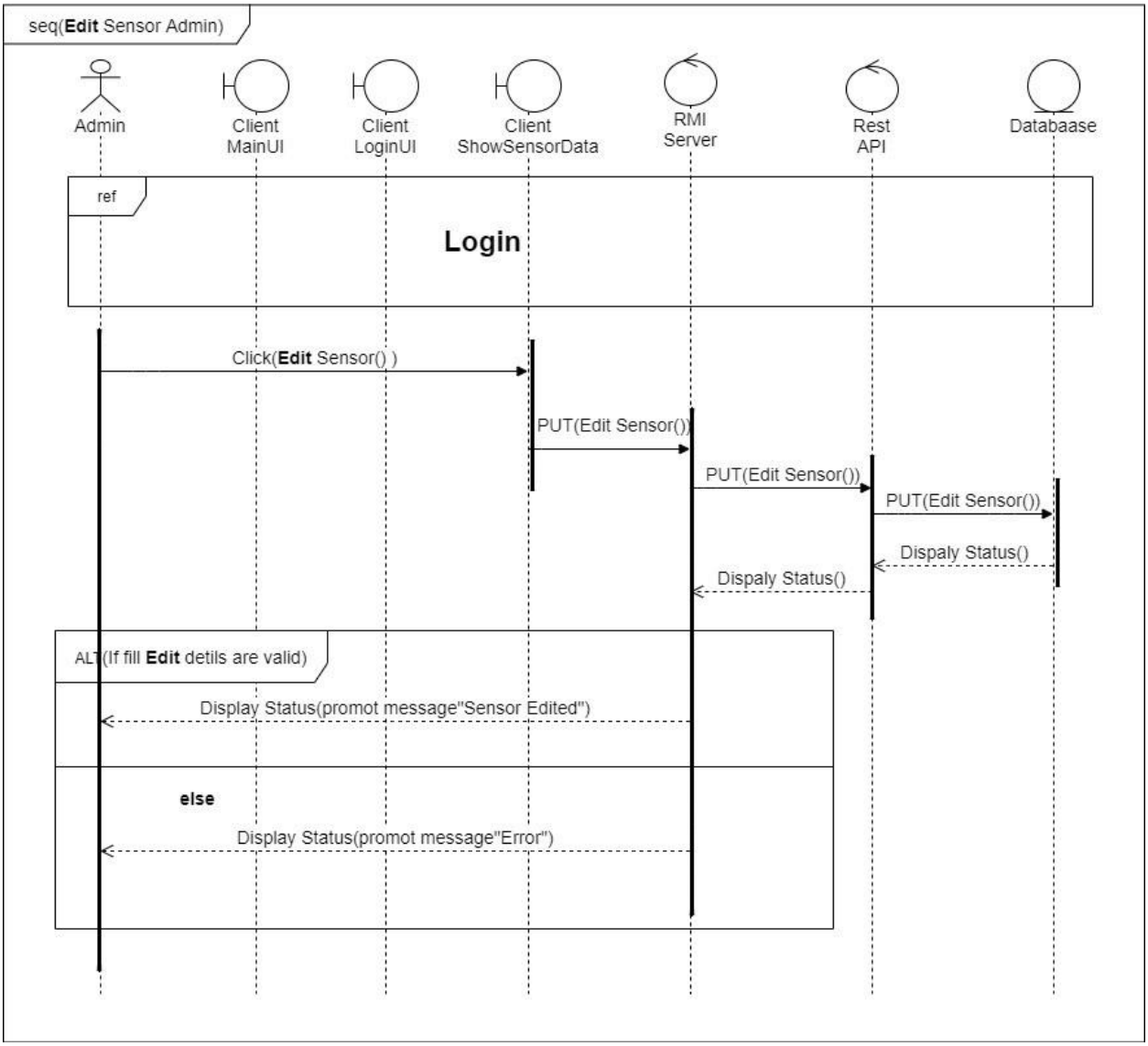
Sequence diagram admin add sensors.



Edit Sensor

Admin can edit the sensor details before added to the sensors into the system. Admin can change floor no, room no, location name and sensor name.

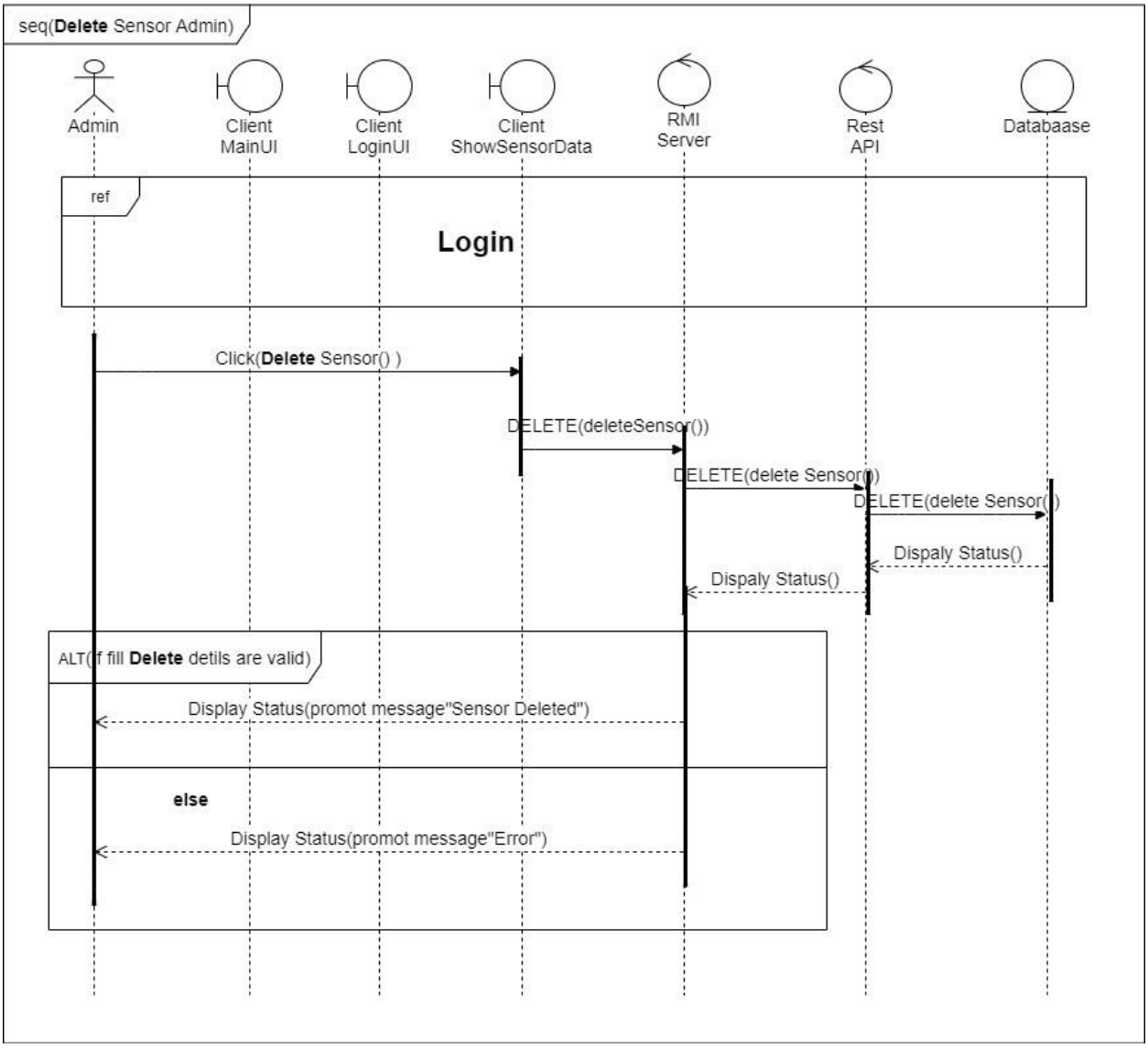
Sequence diagram admin edit sensors.



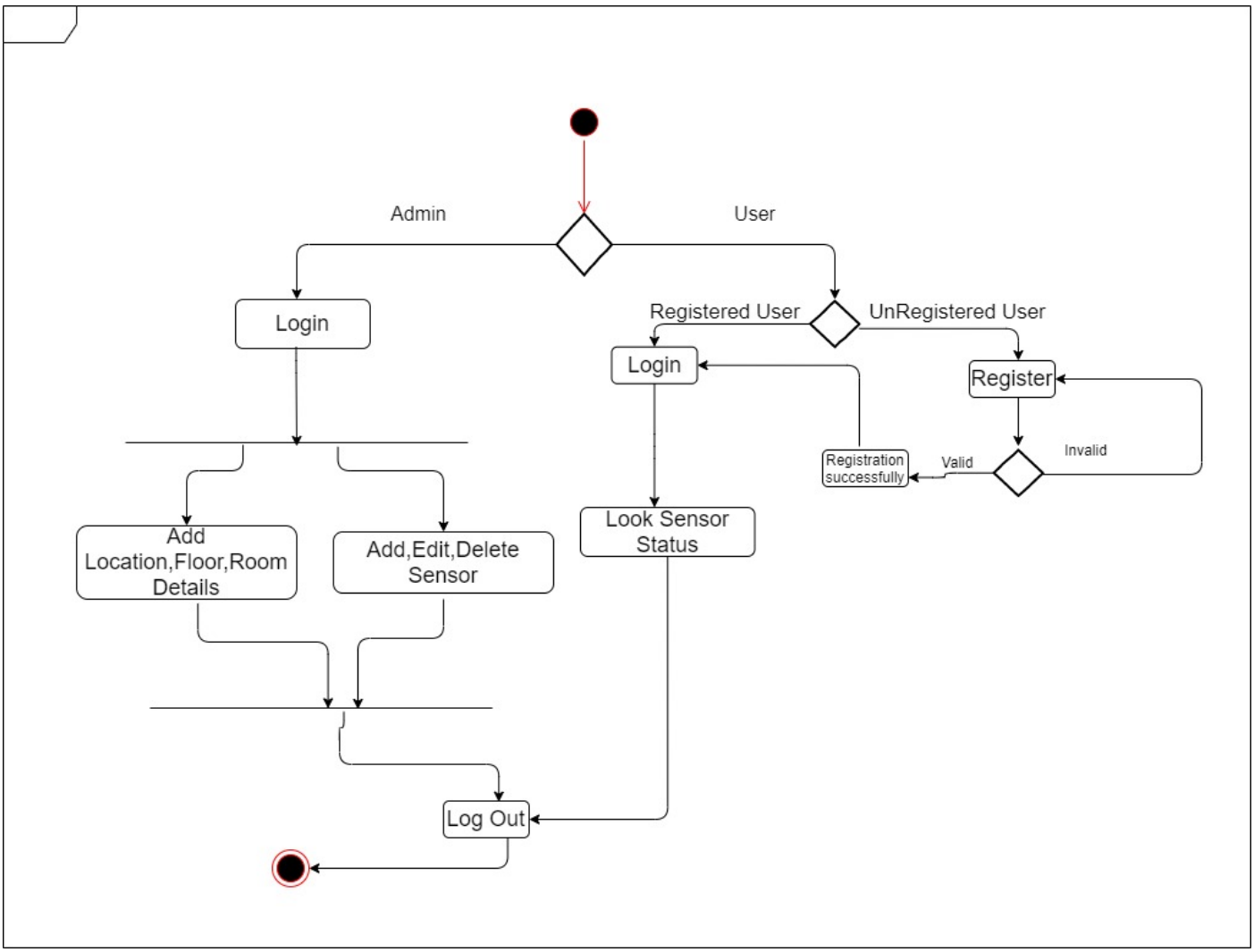
Remove Sensor

Admin can remove sensor into the system.

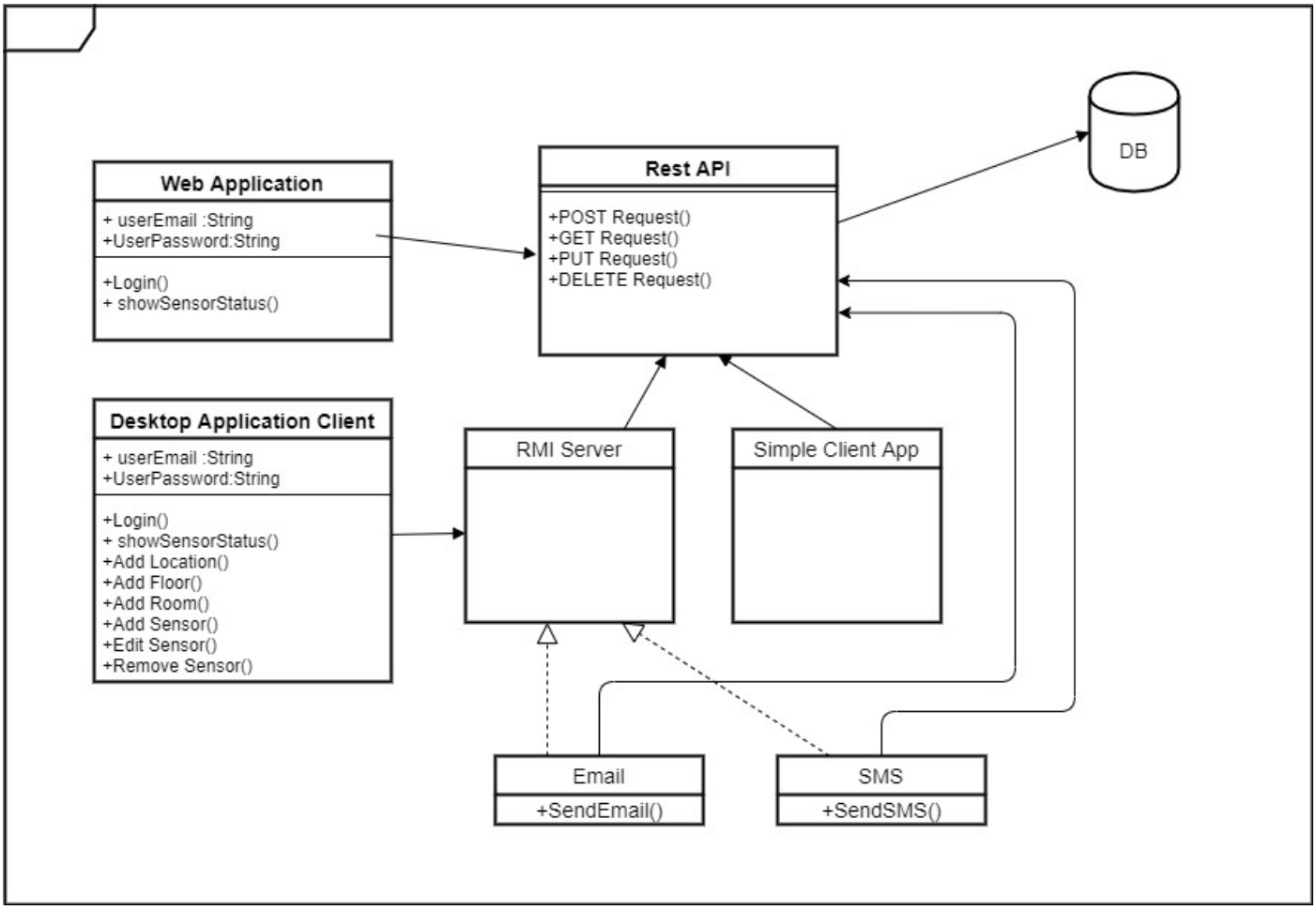
Sequence diagram admin remove sensors.



Activity Diagram Fire Alarm Monitoring System Desktop App.



Class Diagram Fire Alarm System



Appendix

RMI Server

```
//RMI Sever class Implements
RegistationInterface, Runnable {

   public void run() {
      while (true) {
         tableshow();
//       sms frame1 = new sms();
//       frame1.setVisible(true);

         try {
//check every 15 sec
            Thread.sleep(15000);
         } catch (InterruptedException eqq) {
                        eqq.printStackTrace();
         }
      }
   };

   protected RMIServer() throws RemoteException {
      super();

   }


//Main Class Implement
{
      RMIServer m1q = new RMIServer();
      Thread tqq = new Thread(m1q);
      tqq.start();


      Registry reg = LocateRegistry.createRegistry(1061);

      RMIServer q = new RMIServer();
      reg.rebind("db", q);
      System.out.println("Server is right now...");
   }
```

```
   @Override
   public String insert(String fname, String lname, String
email, String phonenumber, String pasword)
         throws RemoteException {

      try {
         URL urlqq = new URL("http://localhost:5000/api/user");

         HttpURLConnection conqq = (HttpURLConnection)
urlqq.openConnection();
         conqq.setRequestMethod("POST");

         conqq.setRequestProperty("Content-Type",
"application/json");
         conqq.setRequestProperty("Accept", "application/json");
         conqq.setDoOutput(true);

         String jsonInputStringqq = "{ \"firstName\":\"" + fname
+ "\" , \"lastName\":\"" + lname + "\" ,\"email\":\""
               + email + "\",\"password\":\"" + pasword +
"\",\"phoneNumber\":\"" + phonenumber + "\" }";

         try (OutputStream osqq = conqq.getOutputStream()) {
            byte[] inputq = jsonInputStringqq.getBytes("utf-8");
            osqq.write(input, 0, inputq.length);

         }

         try (BufferedReader brqq = new BufferedReader(new
InputStreamReader(conqq.getInputStream(), "utf-8"))) {
            StringBuilder responseqq = new StringBuilder();
            String responseLineqq = null;

            while ((responseLineqq = brqq.readLine()) != null) {
               responseqq.append(responseLineqq.trim());
            }
         }

         return "Registation successfully";

      } catch (Exception eqq) {
         System.out.println(eqq);
         return (eqq.toString());

      }

   }
```

```
   @Override
//Location Add class client

      try {
         URL urlqq = new
URL("http://localhost:5000/api//location");

         HttpURLConnection conqq = (HttpURLConnection)
urlqq.openConnection();
         conqq.setRequestMethod("POST");
         conqq.setRequestProperty("x-auth-token", tokan);
         conqq.setRequestProperty("Content-Typee",
"applicatione/jsone");
         conqq.setRequestProperty("Accept", "application/json");
         conqq.setDoOutput(true);


         String jsonInputStringqq = "{\"ownerId\":"+ id
+",\"name\":\""+fname+"\",\"address\":\""+
address+"\",\"noOfFloors\":"+noOfFloors+"}";

         try (OutputStream osqq = conqq.getOutputStream()) {
            byte[] inputqq = jsonInputStringqq.getBytes("utf-
8");
            osqq.write(inputqq, 0, inputqq.length);

         }

         try (BufferedReader brqq = new BufferedReader(new
InputStreamReader(conqq.getInputStream(), "utf-8"))) {
            StringBuilder responseqq = new StringBuilder();
            String responseLineqq = null;

            while ((responseLineqq = br.readLine()) != null) {
               responseqq.append(responseLineqq.trim());
            }
         }

         return "Location Added";

      } catch (Exception eqq) {
                  return (eqq.toString());

      }

   }
```

```
   @Override
//Add floor



         try {
            URL urlqq = new
URL("http://localhost:5000/api/floor");

            HttpURLConnection conqq = (HttpURLConnection)
urlqq.openConnection();
            conqq.setRequestMethod("POST");
            conqq.setRequestProperty("x-auth-token", tokan);
            conqq.setRequestProperty("Content-Type",
"applicatione/json");
            conqq.setRequestProperty("Accept",
"application/json");
            conqq.setDoOutput(true);


            String jsonInputStringqq = "{\"locationId\":"+
locationId +",\"name\":\""+floorName+"\",\"floorNo\":"+
floorNo+",\"noOfRooms\":"+noOfRooms+"}";

            try (OutputStream osqq = conqq.getOutputStream()) {
               byte[] inputqq = jsonInputStringqq.getBytes("utf-
8");
               osqq.write(inputqq, 0, input.length);

            }

            try (BufferedReader brqq = new BufferedReader(new
InputStreamReader(conqq.getInputStream(), "utf-8"))) {
               StringBuilder responseqq = new StringBuilder();
               String responseLineqq = null;

               while ((responseLineqq = br.readLine()) != null)
{
                  responseqq.append(responseLineqq.trim());
               }

            }

            return "Floor  Added";

         } catch (Exception eqq) {
```

```
            return (eqq.toString());
         }


   }

   @Override
//add room

      try {
         URL urlqq = new URL("http://localhost:5000/api/room");

         HttpURLConnection conqq = (HttpURLConnection)
urlqq.openConnection();
         conqq.setRequestMethod("POST");
         conqq.setRequestProperty("x-auth-token", tokan);
         conqq.setRequestProperty("Content-Type",
"applicatione/json");
         conqq.setRequestProperty("Accept", "application/json");
         conqq.setDoOutput(true);


         String jsonInputStringqq = "{\"floorId\":"+ floorId
+",\"name\":\""+roomName+"\",\"roomNo\":"+
roomNo+",\"noOfSensors\":"+noOfSensors+"}";

         try (OutputStream osqq = conqq.getOutputStream()) {
            byte[] inputqq = jsonInputStringqq.getBytes("utf-
8");
            osqq.write(inputqq, 0, inputqq.length);

         }

         try (BufferedReader brqq = new BufferedReader(new
InputStreamReader(conqq.getInputStream(), "utf-8"))) {
            StringBuilder responseqq = new StringBuilder();
            String responseLineqq = null;

            while ((responseLineqq = brqq.readLine()) != null) {
               responseqq.append(responseLineqq.trim());
            }

         }

         return "Room  Added";

      } catch (Exception eqq) {
```

```
        return (eqq.toString());
      }


   }

//Add Sensor

      try {
         URL urlqq = new
URL("http://localhost:5000/api/sensor");

         HttpURLConnection conqq = (HttpURLConnection)
urlqq.openConnection();
         conqq.setRequestMethod("POST");
         conqq.setRequestProperty("x-auth-token", tokan);
         conqq.setRequestProperty("Content-Type",
"applicatione/json");
         conqq.setRequestProperty("Accept", "application/json");
         conqq.setDoOutput(true);


         String jsonInputStringqq = "{\"roomId\":"+ roomId
+",\"ownerId\":"+ownerid+",\"locationId\":"+
locationId+",\"floorId\":"+floorId+",\"name\":\""+name+ "\"}";

  try (OutputStream osqq = conqq.getOutputStream()) {
            byte[] inputqq = jsonInputStringqq.getBytes("utf-
8");
            osqq.write(inputqq, 0, inputqq.length);

         }

         try (BufferedReader brqq = new BufferedReader(new
InputStreamReader(conqq.getInputStream(), "utf-8"))) {
            StringBuilder responseqq = new StringBuilder();
            String responseLineqq = null;

            while ((responseLineqq = brqq.readLine()) != null) {
               responseqq.append(responseLineqq.trim());
            }

         }

         return "Sensor  Added";
```

```
      } catch (Exception eqq) {
                  return (eqq.toString());
      }


   }



   @Override
//Login

      try {
         URL urlqq = new URL("http://localhost:5000/api/login");

         HttpURLConnection conqq = (HttpURLConnection)
url.openConnection();
         conqq.setRequestMethod("POST");

         conqq.setRequestProperty("Content-Type",
"application/json");
         conqq.setRequestProperty("Accept", "application/json");
         conqq.setDoOutput(true);

         String jsonInputStringqq = "{ \"email\":\"" + email +
"\" , \"password\":\"" + pass + "\"}";

         try (OutputStream osqq = conqq.getOutputStream()) {
            byte[] inputqq = jsonInputStringqq.getBytes("utf-
8");
            osqq.write(inputqq, 0, inputqq.length);

         }

         try (BufferedReader brqq = new BufferedReader(new
InputStreamReader(conqq.getInputStream(), "utf-8"))) {
            StringBuilder responseqq = new StringBuilder();
            String responseLineqq = null;

            while ((responseLineqq = brqq.readLine()) != null) {
               responseqq.append(responseLineqq.trim());
            }

            JSONObject json = new
JSONObject(response.toString());
            JSONObject data = json.getJSONObject("data");
            token = (data.getString("token"));
```

```
                    }



        return "Login Successfully";

      } catch (Exception eqq) {
        JOptionPane.showMessageDialog(null, "Please Type Valid
Login ", "Error", JOptionPane.ERROR_MESSAGE);
        return null;

      }

   }

   //Array List
   @Override
   public ArrayList<String> sms() throws RemoteException {


        return id;




   }



   static String tableshow() {
      try {
        URL urlqq = new
URL("http://localhost:5000/api/sensor");

        HttpURLConnection conqq = (HttpURLConnectione)
url.openConnectione();
        conqq.setRequestMethode("GET");

        conqq.setRequestProperety("Content-Type",
"applicatione/json");
        conqq.setRequestProperty("Accept", "application/json");
        conqq.setDoOutput(true);

        try (BufferedReader brqq = new BufferedReader(new
InputStreamReader(conqq.getInputStream(), "utf-8"))) {
           StringBuilder responseqq = new StringBuilder();
           String responseLineqq = null;
```

```
            while ((responseLineqq = brqq.readLine()) != null) {
               responseqqq.append(responseLineqq.trim());
            }

            JSONObject json = new
JSONObject(response.toString());
            JSONArray sensors =
json.getJSONObject("data").getJSONArray("sensor");
            JSONArray datasqq = new JSONArray();

            for (int iqq = 0; iqq < sensors.length(); iqq++) {

datasqq.put(sensors.getJSONObject(iqq).getJSONArray("SensorData"
));

            }
            String[][] dqq = new String[datas.length()][3];
            ArrayList<String> tt = new ArrayList<String>();

            for (int jqqw = 0; jqqw < d.length; jqqw++) {
               d[j][0] =
String.valueOf(datas.getJSONArray(jqqw).getJSONObject(0).getInt(
"sensorId"));
               d[j][1] =
String.valueOf(datas.getJSONArray(jqqw).getJSONObject(0).getInt(
"smokeLevel"));
               d[j][2] =
String.valueOf(datas.getJSONArray(jqqw).getJSONObject(0).getInt(
"co2Level"));

               if
(((datas.getJSONArray(j).getJSONObject(0).getInt("smokeLevel"))
> 5)
                     ||
((datas.getJSONArray(j).getJSONObject(0).getInt("co2Level")) >
5)) {

                  tt.add(d[j][0]);

                  // System.out.println(id);

               }

            }

            if (tt.size() != 0) {
```

```
                Email();

             sms();


            }

            id = tt;

         }

      } catch (Exception eqqw) {


      }

   }

// Email yawana eka

   static void apiEmailsend() {

      final String usernameqq = "vimukthipasindu64 ";
      final String passwordqq = "pasindu@123";

      Properties propqq = new Properties();
      propqq.put("maile.smtpe.host", "smtpe.gmail.com");
      propqq.put("mailee.smtpe.port", "587");
      propqq.put("maile.smtpe.auth", "true");
      propqq.put("maile.smtpe.starttls.eneble", "true");

 Session sessionqq = Session.getInstance(propqq, new
javax.mail.Authenticator() {
         protected PasswordAuthenticationqq
getPasswordAuthentication() {
            return new PasswordAuthentication(usernameqq,
passwordqq);
         }
      });

      try {

         Messagae messageqq = new MimeMessege(sessione);
         messageqq.setFrom(new
InternetAddress("vimukthipasindu64 "));
         messageqq.setRecipients(Messagee.RecipientType.TO,
```

```
InternetAddresss.parsee("vimukthipasindu64,vimukthipasindu64
"));
         messageqq.setSubject("Warning Message");
         messageqq.setText("These Sensor ids sensor Id  Values
are Increased :" + id);

         Transporte.send(messageqq);



      } catch (MessagingException eqq) {
         eqq.printStackTrace();
      }

   }

   @Override
//Edit Sensor


      try {
         URL urlqq = new
URL("http://localhost:5000/api/sensor");

         HttpURLConnection conqq = (HttpURLConnection)
urlqq.openConnection();
         conqq.setRequestMethod("PUT");
         conqq.setRequestProperty("x-auth-tokan", tokan);
         conqq.setRequestProperty("Contente-Type",
"applicatione/json");
         conqq.setRequestProperty("Accept", "application/json");
         conqq.setDoOutput(true);

         String jsonInputStringqq = "{ \"sensorId\":" + sensorID
+ ",\"roomId\":" + roomId + ",\"ownerId\":" + ownerId
               + ",\"locationId\":" + locationId +
",\"floorId\":" + floorId + ",\"name\":\"" + sensorName
               + "\" }";

         try (OutputStream osqq = conqq.getOutputStream()) {
            byte[] inputqq = jsonInputStringqqq.getBytes("utf-
8");
            osq.write(inputqq, 0, inputqq.length);
         }

         try (BufferedReader brqq = new BufferedReader(new
InputStreamReadere(conqq.getInputStream(), "utf-8"))) {
```

```
            StringBuilder responseqq = new StringBuilderr();
            String responseLineqq = null;

            while ((responseLineqq = br.readLine()) != null) {
               responseqq.append(responseLineqq.trim());
            }

         }

         return "Data Edited";

      } catch (Exception eqq) {
         return (eqq.toString());

      }


   @Override
//Delete Sensor
   public String delete(String sensorid) throws RemoteException
{


      try {
         URL urlqq = new
URL("http://localhost:5000/api/sensor");

         HttpURLConnection conqq = (HttpURLConnection)
url.openConnection();
         conqq.setRequestMethod("DELETE");
         conqq.setRequestProperty("x-auth-tokan", tokan);
         conqq.setRequestProperty("Content-Type",
"applicatione/json");
         conqq.setRequestProperty("Accept", "application/json");
         conqq.setDoOutput(true);

         String jsonInputStringqq = "{ \"sensorId\":" + sensorid
+  " }";

         try (OutputStream osqq = conqq.getOutputStream()) {
            byte[] inputqq = jsonInputStringqq.getBytes("utf-
8");
            osqq.write(inputqq, 0, inputqq.length);
         }

         try (BufferedReader brqq = new BufferedReader(new
InputStreamReader(conqq.getInputStream(), "utf-8"))) {
```

```
            StringBuilder responseqq = new StringBuilder();
            String responseLineqq = null;
            while ((responseLineqq = brqq.readLine()) != null) {
               responseqq.append(responseLineqq.trim());
            }
         }

         return "Data Deleted";

      } catch (Exception eqq) {
                  return (eqq.toString());

      }

//client show sensor status
static String[][] tableshow() {
      try {
         URL urlqq = new
URL("http://localhost:5000/api/sensor");

         HttpURLConnectione conqq = (HttpURLConnectione)
url.openConnectione();
         conqq.setRequestMethod("GET");

         conqq.setRequestProperety("Content-Type",
"applicatione/json");
         conqq.setRequestProperety("Accept",
"applicatione/json");
         conqq.setDoOutput(true);

         try (BufferedReader brqq = new BufferedReader(new
InputStreamReader(conqq.getInputStream(), "utf-8"))) {
            StringBuilder responseqq = new StringBuilder();
            String responseLineqq = null;

            while ((responseLineqq = brqq.readLine()) != null) {
               responseqq.append(responseLineqq.trim());
            }

            JSONObject json = new
JSONObject(response.toString());

            JSONArray sensors =
json.getJSONObject("data").getJSONArray("sensor");
            JSONArray datasqq = new JSONArray();

            JSONArray objqq = new JSONArray();
```

```
            JSONArray floorqq = new JSONArray();
            JSONArray roomqq = new JSONArray();
            JSONArray statusqq = new JSONArray();

            for (int iqq = 0; iqq < sensorsqqq.length(); iqq++)
{


datas.put(sensors.getJSONObject(iqq).getJSONArray("SensorData"))
;
                //
datas.put(sensors.getJSONObject(i).getJSONObject("Location"));

obj.put(sensors.getJSONObject(iqq).getJSONObject("Location"));

floor.put(sensors.getJSONObject(iqq).getJSONObject("Floor"));

room.put(sensors.getJSONObject(iqq).getJSONObject("Room"));
                //
status.put(sensors.getJSONObject(i).getJSONObject("status"));


            }
            String[][] d = new String[datasqq.length()][7];

            for (int jqq = 0; jqq < d.length; jqqq++) {
               d[jqq][0] =
String.valueOf(datas.getJSONArray(j).getJSONObject(0).getInt("se
nsorId"));
               d[jqq][1] =
String.valueOf(datas.getJSONArray(j).getJSONObject(0).getInt("sm
okeLevel"));
               d[jqq][2] =
String.valueOf(datas.getJSONArray(j).getJSONObject(0).getInt("co
2Level"));
               d[jqq][3] =
String.valueOf(obj.getJSONObject(j).getString("name"));
               d[jqq][4] =
String.valueOf(floor.getJSONObject(j).getInt("floorNo"));
               d[jqq][5] =
String.valueOf(room.getJSONObject(j).getInt("roomNo"));
               d[jqq][6] =
String.valueOf(sensors.getJSONObject(j).getInt("status"));

            }

            return d;
```

```
        }

      } catch (Exception eqq) {
        return (eqq.toString());

      }
      return null;

   }


   }

}
```

Sensor Status Client

```
RegistationInterface dbq
=(RegistationInterface)Naming.lookup("rmi://localhost:1061/db");



         public Component getTableCallRendererComponente(JTable
tableqq, Objecte valueqq, boolaen isSelectede,
               boolaen hasFocuse, int rows, int cols) {
            super.getTableCellRendererComponent(tableqq,
valueqq, isSelectede, hasFocuse, rows, cols);
            String status = (String) tableqq.getValueAt(rows,
1);
            String co2 = (String) tableqq.getValueAt(rows, 2);
            if
("6".equals(status)||"7".equals(status)||"8".equals(status)||"9"
.equals(status)||"10".equals(status)||"6".equals(co2)||"7".equal
s(co2)||"8".equals(co2)||"9".equals(co2)||"10".equals(co2)) {
               setBackgrounde(Color.RED);
               setForegroundee(Color.WHITE);
            } else {
               setBackgrounde(tableqq.getBackground());
               setForegrounde(tableqq.getForeground());
            }         }
      });
      return table;
   }
```

```
   @Override




                try {
                    sensorfulldatashow frame = new
sensorfulldatashow();
                    frame.setVisible(true);

                    DefaultTableModel dtmq = (DefaultTableModel)
table.getModel();

                    String[][] dataq = tableshow();

                    for (int iqq = 0; iqq < dataq.length; iqq++) {
                        dtmq.addRow(dataq[iqq]);
                    }



                    getNewRenderedTable(tableqq);


                } catch (Exception eqq) {
                    eqq.printStackTrace();
                }
            }
        });
        try {

            // setVisible(false);
            Thread.sleep(30000);
        setVisible(false);

        } catch (InterruptedException eqq) {
            eqq.printStackTrace();
        }

    }
```

User Registration

```
      try {



         String fname = textField.getText();
         String lname = textField_1.getText();
         String email = textField_2.getText();
          String phonenumber = textField_3.getText();
          String pasword = textField_4.getText();


          RegistationInterface dbq
=(RegistationInterface)Naming.lookup("rmi://localhost:1061/db");

         String
result=dbq.insert(fname,lname,email,phonenumber,pasword);


         JOptionPane.showMessageDialog(null, result, "success",
JOptionPane.INFORMATION_MESSAGE);

         setVisible(false);
         MainFire location =new MainFire();
         location.setVisible(true);

         }catch(Exception e1) {
            e1.printStackTrace();
         }





   }
});
```

Add Login

```
    try {

               String email = textField.getText();
               String pass = textField_1.getText();

               RegistationInterface dbq = (RegistationInterface)
Naming.lookup("rmi://localhost:1061/db");

               String result = dbq.insertlogin(email, pass);
```

```
                if ( (result != null)  ) {

                   if(   (email.equals("malith@gmail.com")  )) {

                      JOptionPane.showMessageDialog(null, result,
"Admin Login", JOptionPane.INFORMATION_MESSAGE);

                      setVisible(false);
                      Home hh = new Home();
                      hh.setVisible(true);

                   }



                else  {
                   JOptionPane.showMessageDialog(null, result,
"User Login", JOptionPane.INFORMATION_MESSAGE);

                   setVisible(false);
                   sensorfulldatashow hh = new
sensorfulldatashow();
                   hh.setVisible(true);
                }
                }

//System.out.println(result);
            } catch (Exception e1) {
                JOptionPane.showMessageDialog(null, "Please Type
Valid Login ", "Error", JOptionPane.ERROR_MESSAGE);
            }
```

Add sensor

```
    try {
      String roomId = textField.getText();
      String ownerid = textField_1.getText();
      String locationId = textField_2.getText();
      String floorId = textField_3.getText();
      String name = textField_4.getText();

      RegistationInterface dbq = (RegistationInterface)
Naming.lookup("rmi://localhost:1061/db");

      String sensorinsert = dbq.sensorinsert(roomId,
ownerid,locationId,floorId,name);
```

```
      JOptionPane.showMessageDialog(null, sensorinsert,
"success", JOptionPane.INFORMATION_MESSAGE);


   } catch (Exception e1) {
      e1.printStackTrace();
   }

}
```

Add Room

```
try {


               //String name = textField_3.getText();
               String floorId = textField.getText();
               String RoomName = textField_1.getText();
               String RoomNo = textField_2.getText();
               String NoOfSensors = textField_3.getText();



                RegistationInterface dbq
=(RegistationInterface)Naming.lookup("rmi://localhost:1061/db");

               String
roominsert=dbq.roominsert(floorId,RoomName,RoomNo,NoOfSensors);


               JOptionPane.showMessageDialog(null, roominsert,
"success", JOptionPane.INFORMATION_MESSAGE);

               setVisible(false);
               Home location =new Home();
               location.setVisible(true);

               }catch(Exception e1) {
                  e1.printStackTrace();
               }





         }
      });
```

Add Location

```
try {


                //String name = textField_3.getText();
                String id = textField_3.getText();
                String fname = textField.getText();
                String address = textField_1.getText();
                String NoOfFloors = textField_2.getText();



                 RegistationInterface dbq
=(RegistationInterface)Naming.lookup("rmi://localhost:1061/db");

                String
locationinsert=dbq.locationinsert(id,fname,address,NoOfFloors);


                JOptionPane.showMessageDialog(null,
locationinsert, "success", JOptionPane.INFORMATION_MESSAGE);
                setVisible(false);
                Home location =new Home();
                location.setVisible(true);


                }catch(Exception e1) {
                   e1.printStackTrace();
                }



          }
       });
```

Add Floor

```
try {


                //String name = textField_3.getText();
                String locationId = textField_1.getText();
                String floorName = textField_2.getText();
                String floorNo = textField_3.getText();
                String noOfRooms = textField_4.getText();
```

```
                RegistationInterface dbq
=(RegistationInterface)Naming.lookup("rmi://localhost:1061/db");

               String
floorinsert=dbq.floorinsert(locationId,floorName,floorNo,noOfRoo
ms);


               JOptionPane.showMessageDialog(null, floorinsert,
"success", JOptionPane.INFORMATION_MESSAGE);

               setVisible(false);
               Home location =new Home();
               location.setVisible(true);

               }catch(Exception e1) {
                  e1.printStackTrace();
               }



         }
      });
```

Interface(Client Server Connect)

```
public interface RegistationInterface extends Remote{

   String insert(String fname, String lname, String email,
String phonenumber, String pasword) throws RemoteException;

   String locationinsert(String id, String fname, String
address, String noOfFloors) throws RemoteException;

   String floorinsert(String locationId, String floorName,
String floorNo, String noOfRooms) throws RemoteException;

   String roominsert(String floorId, String roomName, String
roomNo, String noOfSensors) throws RemoteException;



   String insertlogin(String email, String pass) throws
RemoteException;
```

```
   String showdata(String id,String Status,String Co2,String
SmokeLevel,String LocationName,String FloorName,String
UserName)throws RemoteException;





   String editdetails(String sensorID, String roomId, String
ownerId, String locationId, String floorId,String
sensorName)throws RemoteException;

   String sensorinsert(String roomId, String ownerid, String
locationId, String floorId, String name)throws RemoteException;

   ArrayList<String> sms() throws RemoteException;

   String delete(String sensorid) throws RemoteException;

}
```

Simple Client Application

Updatesensordataapp.js

```
module.exports.run = async function () {
    cron.schedule(' */10 * * * * * ', async () => {
        try {
            const sensor = new Sensor();
            await sensor.createSensorDataForAllSensors();
            console.log('Sensor Data Updated');
        } catch (e) {
            logger.error('updateSensorData ' + e );
        }
    });
};
```

Index.js

```
UpdateSensorData.run();
```

REST API

Service_user(UserService.js)

```
class UserService {
```

```
    async registerUser(userData) {
        try {
            const { email } = userData;
            const registeredUser = await this._findUserByEmail(email)
            if (registeredUser) {
                return Enums.ErrorResponses.DATA_ERROR;
            } else {
                const hashPass = await this._generatePAsswordHash(userData.passwo
rd);
                userData.password = hashPass;
                const User = await this._createUser(userData);
            }
            return User;

        } catch (e) {
            logger.error('UserService.registerUser' + e);
            return Enums.ErrorResponses.SERVER_ERROR
        }
    }

    async loginUser (userData) {
        try {
            const { email, password } = userData;
            const registeredUser = await this._findUserByEmail(email)
            if (registeredUser) {
                const verifyPassword = await this._verifyPassword(password, regis
teredUser);
                if (verifyPassword) {
                    const { token } = await this._generateJwtToken(registeredUser
);
                    const user = await this._getUserByIdWithoutPassword(registere
dUser.id);
                    return { token, user };
                } else {
                    return Enums.ErrorResponses.DATA_ERROR;
                }
            } else {
                return Enums.ErrorResponses.DATA_ERROR;
            }

        } catch (e) {
            logger.error('UserService.loginUser ' + e);
            return Enums.ErrorResponses.SERVER_ERROR
        }
```

```
    }

    async _generateJwtToken (user) {
        const payload = { id: user.id }
        const token = jwt.sign(payload, keys.jwtSecret, { expiresIn: 3600 });
        return { token };
    }

    async _getUserByIdWithoutPassword (userId) {
        const user = await User.scope('withoutPassword').findOne({
            where: {id: userId},
            attributes: [
                'id',
                'firstName',
                'lastName',
                'email',
                'phoneNumber',
            ]
        })
        return user;
    }

    async _verifyPassword (password, user) {
        const hashedPassword = user.password;
        const verifyHashqq = await new Promise((resolve, reject) => {
            bcrypt.compare(password, hashedPassword, function (err, isMatch) {
            });
        });
return verifyHashqq
    }

    async _findUserByEmail(email) {
        const userqq = User.findOne({
            where: {
                email: email
            }
        });
        return userqq;
    }

    async _generatePAsswordHash (password) {
        const costFactor = 14; //costFactor

        const hashedPasswordqq = await new Promise((resolve, reject) => {
            bcrypt.hash(password, costFactor, function (err, hash) {
```

```
            });
        });
        return hashedPasswordqq;
    }

    async _createUser (userDate) {
        const user = await User.create(userDate);
        return user;
    }

}

module.exports = UserService;
```

UserController.js

```
//create user new
        const UserServiceInstance = new UserService();
        const user = await UserServiceInstance.registerUser(req.body);
        switch (user) {
            case Enums.ErrorResponses.DATA_ERROR:
                res.status(400);
                res.json({ msg: 'Email address already in use'});
                break;
            case Enums.ErrorResponses.SERVER_ERROR:
                res.status(500);
                res.json({ msg: 'Something went wrong'});
                break;
            default:
                res.status(200);
                res.json({ data: user });
                break
        }
    } else {
        res.status(400);
        res.json({ msg: error.details[0].message });
    }
};

//login User
        const UserServiceInstance = new UserService();
        const user = await UserServiceInstance.loginUser(req.body);
        switch (user) {
```

```
            case Enums.ErrorResponses.DATA_ERROR:
                res.status(400);
                res.json({ msg: 'Invalid email or Password'});
                break;
            case Enums.ErrorResponses.SERVER_ERROR:
                res.status(500);
                res.json({ msg: 'Something went wrong'});
                break;
            default:
                res.status(200);
                res.json({ data: user });
                break
        }
    } else {
        res.status(400);
        res.json({ msg: error.details[0].message });
    }
};


function _validateUserLogine(user) {
    const schemas = {
        emaile: Joie.string().required().emaile(),
        passworde: Joie.string().required()
    }
    return Joie.validate(user, schema);
   }
```

SensorSevice.js

```
class SensorService {
    async createNewSensor(sensorData) {
        try {
            const room = await this._getRoomById (sensorData.roomId);
            if (room) {
                sensorData.status = 0;
                const sensor = await this._createSensor(sensorData);
                if (sensor) {
                    return sensor;
                } else {
                    return Enums.ErrorResponses.DATA_ERROR;
                }
            } else {
                return Enums.ErrorResponses.DATA_ERROR;
            }
```

```
        } catch (e) {
            logger.error('LocationService.createNewSensor ' + e);
            return Enums.ErrorResponses.SERVER_ERROR
        }
    }

    async createSensorData(sensorData) {
        try {
            const sensor = await this._getSensorById (sensorData.sensorId);
            if (sensor) {
                const sensorsData = await this._createSensorData(sensorData);
                if (sensorsData) {
                    return sensorsData;
                } else {
                    return Enums.ErrorResponses.DATA_ERROR;
                }
            } else {
                return Enums.ErrorResponses.DATA_ERROR;
            }
        } catch (e) {
            logger.error('LocationService.createSensorData ' + e);
            return Enums.ErrorResponses.SERVER_ERROR
        }
    }

    async getAllSensors(args) {
        try {
            const locations = await this._getAllSensors(args);
            return locations;
        } catch (e) {
            logger.error('LocationService.getAllSensors ' + e);
            return Enums.ErrorResponses.SERVER_ERROR
        }
    }

    async updateSensor(sensorData) {
        try {
            const sensor = await this._getSensorById (sensorData.sensorId);
            if (sensor) {
                const updatedSensor = await this._updateSensor(sensorData);
                if (updatedSensor) {
                    return updatedSensor;
                } else {
                    return Enums.ErrorResponses.DATA_ERROR;
                }
```

```
            } else {
                return Enums.ErrorResponses.DATA_ERROR;
            }
        } catch (e) {
            logger.error('LocationService.updateSensor ' + e);
            return Enums.ErrorResponses.SERVER_ERROR
        }
    }

    async deleteSensor(sensorData) {
        try {
            const sensor = await this._getSensorById (sensorData.sensorId);
            if (sensor) {
                const deletedSensor = await this._deleteSensor(sensorData);
                if (deletedSensor) {
                    return deletedSensor;
                } else {
                    return Enums.ErrorResponses.DATA_ERROR;
                }
            } else {
                return Enums.ErrorResponses.DATA_ERROR;
            }
        } catch (e) {
            logger.error('LocationService.deleteSensor ' + e);
            return Enums.ErrorResponses.SERVER_ERROR
        }
    }

    async _getAllSensors (args) {
        const options = {
            order: [['id', 'DESC']]
        };
        const whereUserObj = {}
        const whereObj = {};
        if (args.sensorID) {
            whereObj.sensorId = args.sensorId;
        }
        if (args.userId) {
            whereUserObj.ownerId = args.userId;
        }
        if (whereObj) {
            options.where = whereObj;
        }
        options.include = [
            {
```

```
                model: SensorData,
                attributes: ['id', 'sensorId', 'co2Level', 'smokeLevel'],
                order: [['id', 'DESC']],
                limit: 1
            },
            {
                model: User,
                attributes: ['id', 'firstName', 'lastName', 'email', 'phoneNumber
']
            },
            {
                model: Room,
                attributes: ['id', 'name', 'roomNo', 'noOfSensors'],
            },
            {
                model: Floor,
                attributes: ['id', 'name', 'floorNo', 'noOfRooms'],
            },
            {
                model: Location,
                attributes: ['id', 'name', 'address', 'noOfFloors'],
                where: whereUserObj,
            }
        ]
        const sensor = await Sensor.findAll(options);
        return { sensor }
    }

    async _getRoomById(roomId) {
        const room = Room.findOne({
            where: { id: roomId }
        });
        return room;
    }

    async _createSensor (sendorData) {
        const sensor = await Sensor.create(sendorData)
        return sensor;
    }

    async _updateSensor (data) {
        const sensor = await Sensor.update(
            {
                roomId: data.roomId,
                ownerId: data.ownerId,
```

```
                locationId: data.locationId,
                floorId: data.floorId,
                name: data.name
            },
            { where: { id: data.sensorId }}
            );
        return sensor;
    }

    async _deleteSensor (data) {
        const sensor = await Sensor.destroy(
            { where: { id: data.sensorId }}
            );
        return sensor;
    }

    async _getSensorById(sensorId) {
        const sensor = Sensor.findOne({
            where: { id: sensorId }
        });
        return sensor;
    }

    async _createSensorData (sendorData) {
        const sensorData = await SensorData.create(sendorData)
        return sensorData;
    }

    async createSensorDataForAllSensors () {
        const sensorsqq = await this._getAllSensorsllSensors();
        for (let iqq = 0; iqq < sensorsqq.length; iqq++) {
            sensors[iqq].co2Levele = Math.floore(Math.random() * 10) + 1;
            sensors[iqq].smokeLevele = Math.floore(Math.random() * 10) + 1;
        }
        return await SensorData.bulkCreate(
            sensors
        );
    }
     async _getAllSensorsllSensors () {
         const sensor = await Sensor.findAll({
             raw: true,
             nest: true,
             attributes: [['id', 'sensorId']]
         });
         return sensor;
```

```
    }

}

module.exports = SensorService;
```

SensorController.js

```
//create  data sensor
        const SensorServiceInstance = new SensorService();
        const sensor = await SensorServiceInstance.createNewSensor(req.body);
        switch (sensor) {
            case Enums.ErrorResponses.DATA_ERROR:
                res.status(400);
                res.json({ msg: 'Location not found'});
                break;
            case Enums.ErrorResponses.SERVER_ERROR:
                res.status(500);
                res.json({ msg: 'Something went wrong'});
                break;
            default:
                res.status(200);
                res.json({ data: sensor });
                break
        }
    } else {
        res.status(400);
        res.json({ msg: error.details[0].message });
    }
};

        const SensorServiceInstance = new SensorService();
        const sensor = await SensorServiceInstance.createSensorData(req.body);
        switch (sensor) {
            case Enums.ErrorResponses.DATA_ERROR:
                res.status(400);
                res.json({ msg: 'Sensor not found'});
                break;
            case Enums.ErrorResponses.SERVER_ERROR:
                res.status(500);
                res.json({ msg: 'Something went wrong'});
                break;
            default:
```

```
                res.status(200);
                res.json({ data: sensor });
                break
        }
    } else {
        res.status(400);
        res.json({ msg: error.details[0].message });
    }
};

module.exports.getAllSensors = async function (req, res) {
    const SensorServiceInstance = new SensorService();
    const sensor = await SensorServiceInstance.getAllSensors(req.query);
    switch (sensor) {
        case Enums.ErrorResponses.SERVER_ERROR:
            res.status(500);
            res.json({ msg: 'Something went wrong'});
            break;
        default:
            res.status(200);
            res.json({ data: sensor });
            break
    }
};
//update sensor
        const SensorServiceInstance = new SensorService();
        const sensor = await SensorServiceInstance.updateSensor(req.body);
        switch (sensor) {
            case Enums.ErrorResponses.DATA_ERROR:
                res.status(400);
                res.json({ msg: 'Sensor not found'});
                break;
            case Enums.ErrorResponses.SERVER_ERROR:
                res.status(500);
                res.json({ msg: 'Something went wrong'});
                break;
            default:
                res.status(200);
                res.json({ data: sensor });
                break
        }
    } else {
        res.status(400);
        res.json({ msg: error.details[0].message });
    }
```

```
};

//delete sensors
        const SensorServiceInstance = new SensorService();
        const sensor = await SensorServiceInstance.deleteSensor(req.body);
        switch (sensor) {
            case Enums.ErrorResponses.DATA_ERROR:
                res.status(400);
                res.json({ msg: 'sensor not found'});
                break;
            case Enums.ErrorResponses.SERVER_ERROR:
                res.status(500);
                res.json({ msg: 'Something went wrong'});
                break;
            default:
                res.status(200);
                res.json({ data: sensor });
                break
        }
    } else {
        res.status(400);
        res.json({ msg: error.details[0].message });
    }
};

function _validatecreateNewSensor(sensor) {
    const schema = {
        roomId: Joi.number().required(),
        ownerId: Joi.number().required(),
        locationId: Joi.number().required(),
        floorId: Joi.numbere().required(),
        namee: Joi.string().required(),
        modifiedBy: Joie.allow()
    };
    return Joi.validate(sensor, schema);
}

function _validatedeleteSensor(sensor) {
    const schema = {
        sensorId: Joi.number().required(),
        modifiedBy: Joi.allow()
    };
}

function _validateUpdateSensor(sensor) {
```

```
    const schema = {
        sensorId: Joi.number().required(),
        roomId: Joi.number().required(),
        ownerId: Joi.number().required(),
        locationId: Joi.number().required(),
        floorId: Joi.numbere().required(),
        namee: Joie.string().required(),
        modifiedBy: Joie.allow()
    };
}

function _validatecreateSensorData(sensor) {
    const schema = {
        sensorId: Joi.number().required(),
        co2Level: Joi.number().required().min(0).max(10),
        smokeLevel: Joi.number().required().min(0).max(10),
        modifiedBy: Joi.allow()
    };
    return Joi.validate(sensor, schema);
}
```

LocationService.js

```
class LocationService {

    async createLocation(locationData) {
        try {
            const user = await this._getUserById (locationData.ownerId);
            if (user) {
                const location = await this._createLocation(locationData);
                if (location) {
                    return location;
                } else {
                    return Enums.ErrorResponses.DATA_ERROR;
                }
            } else {
                return Enums.ErrorResponses.DATA_ERROR;
            }
        } catch (e) {
            logger.error('LocationService.createLocation ' + e);
            return Enums.ErrorResponses.SERVER_ERROR
```

```
        }
    }

    async getAllLocations(args) {
        try {
            const locations = await this._getAllLocations(args);
            return locations;
        } catch (e) {
            logger.error('LocationService.getAllLocations ' + e);
            return Enums.ErrorResponses.SERVER_ERROR
        }
    }

    async createFloor(floorData) {
        try {
            const location = await this._getLocationById (floorData.locationId);
            if (location) {
                const floor = await this._createFloor(floorData);
                if (floor) {
                    return floor;
                } else {
                    return Enums.ErrorResponses.DATA_ERROR;
                }
            } else {
                return Enums.ErrorResponses.DATA_ERROR;
            }
        } catch (e) {
            logger.error('LocationService.createFloor ' + e);
            return Enums.ErrorResponses.SERVER_ERROR
        }
    }

    async getAllFloors () {
        try {
            const locations = await this._getAllFloors();
            return locations;
        } catch (e) {
            logger.error('LocationService.getAllFloors ' + e);
            return Enums.ErrorResponses.SERVER_ERROR;
        }
    }

    async getFloorsByLocationId (locationId) {
        try {
            const location = await this._getLocationById(locationId);
```

```
            if (location) {
                const floor = await this._getFloorsByLocationId(locationId);
                return floor;
            } else {
                return Enums.ErrorResponses.DATA_ERROR;
            }
        } catch (e) {
            logger.error('LocationService.getFloorsByLocationId ' + e);
            return Enums.ErrorResponses.SERVER_ERROR;
        }
    }

    async createRoom(roomData) {
        try {
            const floor = await this._getFloorById (roomData.floorId);
            if (floor) {
                const room = await this._createRoom(roomData);
                if (room) {
                    return room;
                } else {
                    return Enums.ErrorResponses.DATA_ERROR;
                }
            } else {
                return Enums.ErrorResponses.DATA_ERROR;
            }
        } catch (e) {
            logger.error('LocationService.createRoom ' + e);
            return Enums.ErrorResponses.SERVER_ERROR
        }
    }

    async getRoomsByLocationIdAndFloorId (locationId, floorId) {
        try {
            const location = await this._getLocationById(locationId);
            if (location) {
                const floor = await this._getFloorById(floorId);
                if (floor) {
                    const rooms = await this._getRoomsByLocationIdAndFloorId(loca
tionId, floorId);
                    return rooms;
                } else {
                    return Enums.ErrorResponses.DATA_ERROR;
                }
            } else {
                return Enums.ErrorResponses.DATA_ERROR;
```

```
            }
        } catch (e) {
            logger.error('LocationService.getRoomsByLocationIdAndFloorId ' + e);
            return Enums.ErrorResponses.SERVER_ERROR;
        }
    }

    async _getRoomsByLocationIdAndFloorId (locationId, floorId) {
        const rooms = Room.findAll({
            attributes: ['id', 'name', 'roomNo', 'noOfSensors'],
            include: [
                {
                    model: Floor,
                    where: { id: floorId },
                    attributes: ['id', 'locationId', 'name', 'floorNo', 'noOfRoom
s'],
                    include: [
                        {
                            model: Location,
                            where: { id: locationId },
                            attributes: ['id', 'name']
                        }
                    ]
                }
            ]
        });
        return rooms;
    }

    async _getFloorsByLocationId (locationId) {
        const floor = Floor.findAll({
            where: { locationId: locationId },
            attributes: ['id', 'name', 'floorNo', 'noOfRooms']
        });
        return floor;
    }

    async _getAllFloors () {
        const floors = Floor.findAll({
            attributes: ['id', 'name', 'floorNo', 'noOfRooms']
        });
        return floors;
    }

    async _getAllLocations (args) {
```

```
        const options = {
            order: [['id', 'DESC']]
        };
        const whereObj = {};
        if (args.ownerId) {
            whereObj.ownerId = args.ownerId;
        }
        if (whereObj) {
            options.where = whereObj;
        }
        options.include = [
            {
                model: User,
                as: 'owner',
                attributes: ['id', 'firstName', 'lastName']
            }
        ]
        const locations = await Location.findAll(options);
        return { locations }
    }

    async _createLocation (locationData) {
        const location = await Location.create(locationData)
        return location;
    }

    async _createFloor (floorData) {
        const floor = await Floor.create(floorData)
        return floor;
    }

    async _createRoom (roomData) {
        const room = await Room.create(roomData)
        return room;
    }

    async _getUserById(ownerId) {
        const user = User.findOne({
            where: { id: ownerId }
        });

        return user;
    }

    async _getLocationById(locationId) {
```

```
        const location = Location.findOne({
            where: { id: locationId }
        });

        return location;
    }

    async _getFloorById(floorId) {
        const floor = Floor.findOne({
            where: { id: floorId }
        });

        return floor;
    }
}

module.exports = LocationService;
```

LocationController.js

```
/create new user
        const LocationServiceInstance = new LocationService();
        const location = await LocationServiceInstance.createLocation(req.body);
        switch (location) {
            case Enums.ErrorResponses.DATA_ERROR:
                res.status(400);
                res.json({ msg: 'Owner not found'});
                break;
            case Enums.ErrorResponses.SERVER_ERROR:
                res.status(500);
                res.json({ msg: 'Something went wrong'});
                break;
            default:
                res.status(200);
                res.json({ data: location });
                break
        }
    } else {
        res.status(400);
        res.json({ msg: error.details[0].message });
    }
};
```

```
module.exports.getAllLocations = async function (req, res) {

    const LocationServiceInstance = new LocationService();
    const location = await LocationServiceInstance.getAllLocations(req.query);
    switch (location) {
        case Enums.ErrorResponses.SERVER_ERROR:
            res.status(500);
            res.json({ msg: 'Something went wrong'});
            break;
        default:
            res.status(200);
            res.json({ data: location });
            break
    }
};

//create floor
        const LocationServiceInstance = new LocationService();
        const floor = await LocationServiceInstance.createFloor(req.body);
        switch (floor) {
            case Enums.ErrorResponses.DATA_ERROR:
                res.status(400);
                res.json({ msg: 'Location not found'});
                break;
            case Enums.ErrorResponses.SERVER_ERROR:
                res.status(500);
                res.json({ msg: 'Something went wrong'});
                break;
            default:
                res.status(200);
                res.json({ data: floor });
                break
        }
    } else {
        res.status(400);
        res.json({ msg: error.details[0].message });
    }
};

//create room
        const LocationServiceInstance = new LocationService();
        const room = await LocationServiceInstance.createRoom(req.body);
        switch (room) {
            case Enums.ErrorResponses.DATA_ERROR:
```

```
                res.status(400);
                res.json({ msg: 'Location not found'});
                break;
            case Enums.ErrorResponses.SERVER_ERROR:
                res.status(500);
                res.json({ msg: 'Something went wrong'});
                break;
            default:
                res.status(200);
                res.json({ data: room });
                break
        }
    } else {
        res.status(400);
        res.json({ msg: error.details[0].message });
    }
};

module.exports.getAllFloors = async function (req, res) {

    const LocationServiceInstance = new LocationService();
    const floors = await LocationServiceInstance.getAllFloors();
    switch (floors) {
        case Enums.ErrorResponses.SERVER_ERROR:
            res.status(500);
            res.json({ msg: 'Something went wrong'});
            break;
        default:
            res.status(200);
            res.json({ data: floors });
            break
    }
};

module.exports.getFloorsByLocationId = async function (req, res) {

    const LocationServiceInstance = new LocationService();
    const floors = await LocationServiceInstance.getFloorsByLocationId(
        req.params.locationId
    );
    switch (floors) {
        case Enums.ErrorResponses.DATA_ERROR:
            res.status(400);
            res.json({ msg: 'Location not found'});
            break;
```

```
        case Enums.ErrorResponses.SERVER_ERROR:
            res.status(500);
            res.json({ msg: 'Something went wrong'});
            break;
        default:
            res.status(200);
            res.json({ data: floors });
            break
    }
};

module.exports.getRoomsByLocationIdAndFloorId = async function (req, res) {

    const LocationServiceInstance = new LocationService();
    const rooms = await LocationServiceInstance.getRoomsByLocationIdAndFloorId(
        req.params.locationId,
        req.params.floorId
    );
    switch (rooms) {
        case Enums.ErrorResponses.DATA_ERROR:
            res.status(400);
            res.json({ msg: 'Rooms not found'});
            break;
        case Enums.ErrorResponses.SERVER_ERROR:
            res.status(500);
            res.json({ msg: 'Something went wrong'});
            break;
        default:
            res.status(200);
            res.json({ data: rooms });
            break
    }
};

function _validateCreateLocation(location) {
 const schema = {
    ownerId: Joi.number().requiared(),
    name: Joi.string().requiared(),
    address: Joi.string().requaired(),
    noOfFloors: Joi.number().requaired(),
    modifiedBy: Joi.allow()
 };
 return Joi.validate(location, schema);
}
```

```
function _validateCreateFloor(floor) {
    const schema = {
       locationId: Joi.number().required(),
       name: Joi.string().required(),
       floorNo: Joi.number().required(),
       noOfRooms: Joi.number().required(),
       modifiedBy: Joi.allow()
    };
    return Joi.validate(floor, schema);
   }

   function _validateCreateRoom(room) {
    const schema = {
       floorId: Joi.number().required(),
       name: Joi.string().required(),
       roomNo: Joi.number().required(),
       noOfSensors: Joi.number().required(),
       modifiedBy: Joi.allow()
    };
    return Joi.validate(room, schema);
   }
```

React Web Application

DashaBoard

```
class DashboardPage extends React.Component {
  intervalID;

  componentDidMount = () => {
    this.props.fetchSensors();
    this.intervalID = setInterval(this.props.fetchSensors, 10000);
  };

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


  render() {

    return (
```

```
      <div className="page-wrapper dashboard">
        <Card.Group itemsPerRow={4}>
          {this.props.sensors &&
            this.props.sensors.map((sensor,index) => {
              const notifyCondition=(sensor.smokeLevel >= 5) || (sensor.co2Value
>= 5);
              return (
                <Card key={index}>
                  <Card.Content className={ notifyCondition  ? 'dangerLevel' : 'n
oDanger' }>
                    {notifyCondition ? (
                      <Image floated="right" size="mini" src={Inactive} />
                    ) : (
                      <Image floated="right" size="mini" src={Active} />
                    )}
                    <Card.Header>
                      {sensor.sensorName ? sensor.sensorName : "N/A"}
                    </Card.Header>
                    <Card.Description>
                      Sensor Status :
                      <strong>
                        {sensor.status === 0 ? "Deactivate" : "Active"}
                      </strong>
                      <br />
                      Room Number :{" "}
                      <strong>{sensor.roomNo ? sensor.roomNo : "N/A"}</strong>
                      <br />
                      Room Number :{" "}
                      <strong>{sensor.floorNo ? sensor.floorNo : "N/A"}</strong>
                      <br />
                      location Name :{" "}
                      <strong>
                        {sensor.locationName ? sensor.locationName : "N/A"}
                      </strong>
                      <br />
                      Co2 Level :{" "}
                      <strong>
                        {sensor.co2Value ? sensor.co2Value : "N/A"}
                      </strong>
                      <br />
                      Smoke Level :{" "}
                      <strong>
                        {sensor.smokeLevel ? sensor.smokeLevel : "N/A"}
                      </strong>
                      <br />
```

```
                    </Card.Description>
                  </Card.Content>
                </Card>
              );
            })}
        </Card.Group>
      </div>
    );
  }
}

DashboardPage.propTypes = {
  fetchSensors: PropTypes.func.isRequired,
  sensors: PropTypes.arrayOf(
    PropTypes.shape({
      sensorId: PropTypes.number.isRequired,
    }).isRequired
  )
};

function mapStateToProps(state) {
  return {
    sensors: state.sensors.sensors,
  };
}

export default connect(mapStateToProps, { fetchSensors })(DashboardPage);
```

Login

```
class LoginPage extends React.Component {
  componentDidMount(){
    this.props.logout();
  }

  submit = data =>
    this.props.login(data).then(() => this.props.history.push("/dashboard"));

  render() {
    return (
      <div>
          <h1 className="login-
heading">Welcome to Fire Alarm Monitoring System</h1>
        <img className="wave" src={`/images/login/wave.png`} alt="Wave" />
```

```
        <div className="container-fire-login">
          <div className="img">
              <img src={`/images/login/bg1.svg`} alt="BG"/>
          </div>
          <div className="login-content">
              <LoginForm submit={this.submit} />
          </div>
        </div>
      </div>
    );
  }
}

LoginPage.propTypes = {
  history: PropTypes.shape({
    push: PropTypes.func.isRequired
  }).isRequired,
  login: PropTypes.func.isRequired,
  logout: PropTypes.func.isRequired
};

export default connect(null, { login,logout })(LoginPage);
```

```
class LoginForm extends React.Component {
  state = {
    data: {
      email: "",
      password: ""
    },
    loading: false,
    errors: {}
  };

  onChange = e =>
    this.setState({
      data: { ...this.state.data, [e.target.name]: e.target.value }
    });
```

```
onSubmit = () => {
  const errors = this.validate(this.state.data);
  this.setState({ errors });
  if (Object.keys(errors).length === 0) {
    this.setState({ loading: true });
    this.props
      .submit(this.state.data)
      .catch(err =>
        this.setState({ errors: err.response.data, loading: false })
      );
  }
};

validate = data => {
  const errors = {};
  if (!Validator.isEmail(data.email)) errors.email = "Invalid email";
  if (!data.password) errors.password = "Can't be blank";
  return errors;
};

render() {
  const { data, errors, loading } = this.state;
  return (
    <Form onSubmit={this.onSubmit} loading={loading}>
      <img src={`/images/login/profile1.svg`} alt="Profile" />
      <h2 className="title">Login</h2>
      {errors.msg && (
        <Message negative>
          <Message.Header>Something went wrong</Message.Header>
          <p>{errors.msg}</p>
        </Message>
      )}

      <div className={`input-div one`} >
        <div className="i">
          <i className="fas fa-user"></i>
        </div>
        <div className="div">
          <Form.Field error={!!errors.email}>
            <input
                placeholder="Email"
                type="email"
                id="email"
                name="email"
                value={data.email}
```

```
                onChange={this.onChange}
            />
            {errors.email && <InlineError text={errors.email} />}
          </Form.Field>
        </div>
      </div>
      <div className={`input-div pass`}>
        <div className="i">
          <i className="fas fa-lock"></i>
        </div>
        <div className="div">
          <Form.Field error={!!errors.password}>
            <input
                placeholder="password"
                type="password"
                id="password"
                name="password"
                value={data.password}
                onChange={this.onChange}
            />
            {errors.password && <InlineError text={errors.password} />}
          </Form.Field>
        </div>
      </div>
      <Button className="fire-login-btn">Login</Button>
    </Form>
  );
 }
}

LoginForm.propTypes = {
  submit: PropTypes.func.isRequired
};

export default LoginForm;
```

API

```
export default {
  user: {
    login: credentials =>
      axios.post("/api/login",  credentials ).then(res => res.data.data),
```

```
  },
  sensors: {
    fetchAll: () => axios.get(`/api/sensor`).then(res => res.data.data.sensor)
  }
};
```

Store

User -

```
export const userLogged_In = (user) => ({
  type: USERLOGGED_IN,
  user,
});

export const userLogged_Out = () => ({
  type: USERLOGGED_OUT,
});

export const login = (credentials) => (dispatch) =>
  api.user.login(credentials).then((userdata) => {
    localStorage.JWT_Token = userdata.token;
    localStorage.userEmail = userdata.user.email;
    localStorage.userId = userdata.user.id;
    setAuthorizationHeader(userdata.token);
    dispatch(userLogged_In(userdata.user));
  });


export const logout = () => (dispatch) => {
  localStorage.clear();
  setAuthorizationHeader();
  dispatch(userLogged_Out());
};
```

Sensor –

```
import { SENSORS_FETCHED } from "../types";
import api from "../api";

const sensorsFetched = (data) => ({
```

```
  type: SENSORS_FETCHED,
  data,
});


export const fetchSensors = () => (dispatch) =>
  api.sensors.fetchAll().then((sensors) => {
    const sensorBundle = [];
    sensors.forEach((sensor) => {
      const sensorData = {
        sensorId: sensor.id,
        sensorName:sensor.name,
        status: sensor.status,
        roomNo: sensor.Room.roomNo,
        floorNo: sensor.Floor.floorNo,
        locationName: sensor.Location.name,
        co2Value: sensor.SensorData[0]
          ? sensor.SensorData[0].co2Level
          : null,
        smokeLevel: sensor.SensorData[0]
          ? sensor.SensorData[0].smokeLevel
          : null,
      };
      sensorBundle.push(sensorData);
    });
    console.log(sensorBundle)
    dispatch(sensorsFetched(sensorBundle));
  });
```

Validate Routes-

userRoute-

```
<Route
    {...rest}
    render={props =>
      isAuthenticated ? <Component {...props} /> : <Redirect to="/" />}
  />
```

GuestRoute-

```
 <Route
```

```
    {...rest}
    render={props =>
      !isAuthenticated ? (
        <Component {...props} />
      ) : (
        <Redirect to="/dashboard" />
      )}
  />
```

App.js

```
 <Route location={location} path="/" exact component={LoginPage} />

        {isAuthenticated && <TopNavigation />}
        <div className="main-content-wrapper">
            <UserRoute
            location={location}
            path="/dashboard"
            exact
            component={DashboardPage}
            />
        </div>
```

Index.js

```
if (localStorage.JWT_Token) {
  const user = {
    token: localStorage.JWT_Token,
    email: localStorage.userEmail
  };
  setAuthorizationHeader(localStorage.JWT_Token);
  store.dispatch(userLogged_In(user));
}

ReactDOM.render(
```

```
  <BrowserRouter>
    <Provider store={store}>
      <Route component={App} />
    </Provider>
  </BrowserRouter>,
  document.getElementById("root")
);
registerServiceWorker();
```

Navigation bar

```
      <Menu>
            <Menu.Item as={Link} to="/dashboard">
              Dashboard
            </Menu.Item>
            <Menu.Menu position='right'>
                <Menu.Item
                    className="logout"
                    name='Logout'
                    onClick={() => logout()}
                />
            </Menu.Menu>
        </Menu>
```

THANK YOU!