Sri Lanka Institute of Information Technology

Distributed System – SE3020

Assignment 2

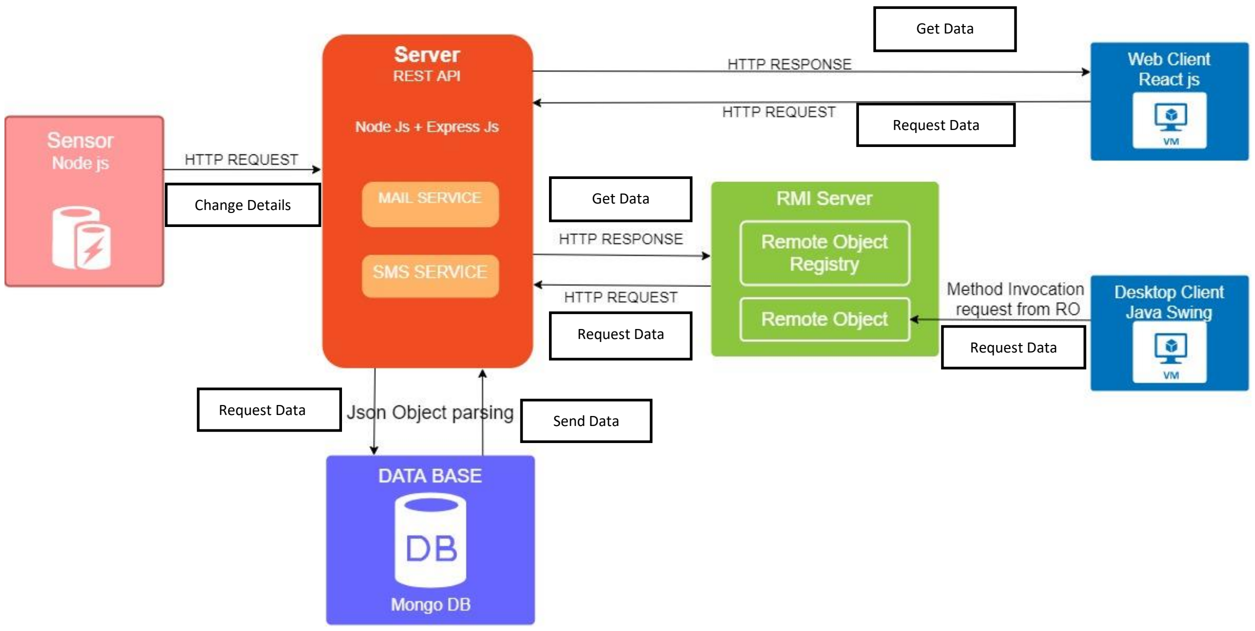
Final Report

	Student Registration Number	Student Name
1	IT18118032	Dissanayaka R.T
2	IT18118100	Dissanayake D.M.D.Y
3	IT18128246	Amalya H.A.V
4	IT18110180	Lakshan H.D.L

TABLE OF CONTENTS

High level Diagram Architecture

This diagram indicates the services and communicating protocols in between services. The Server and DB communicate through HTTP requests using JSON object parsing. The diagram doesn't Include Remote object registering in Remote Object registry and Client registering in Remote object registry.



System includes

1. REST API service
2. Sensor service
3. Web client service
4. Desktop client service
5. RMI service(Email + SMS service)

REST API

REST API is the main service bus of the system. All of the data transmitting over the network goes through the REST API. REST API connects the sensor, web client, desktop client along with the RMI server to the database. For the REST API Node JS and Express JS technologies were used.

REST API Services

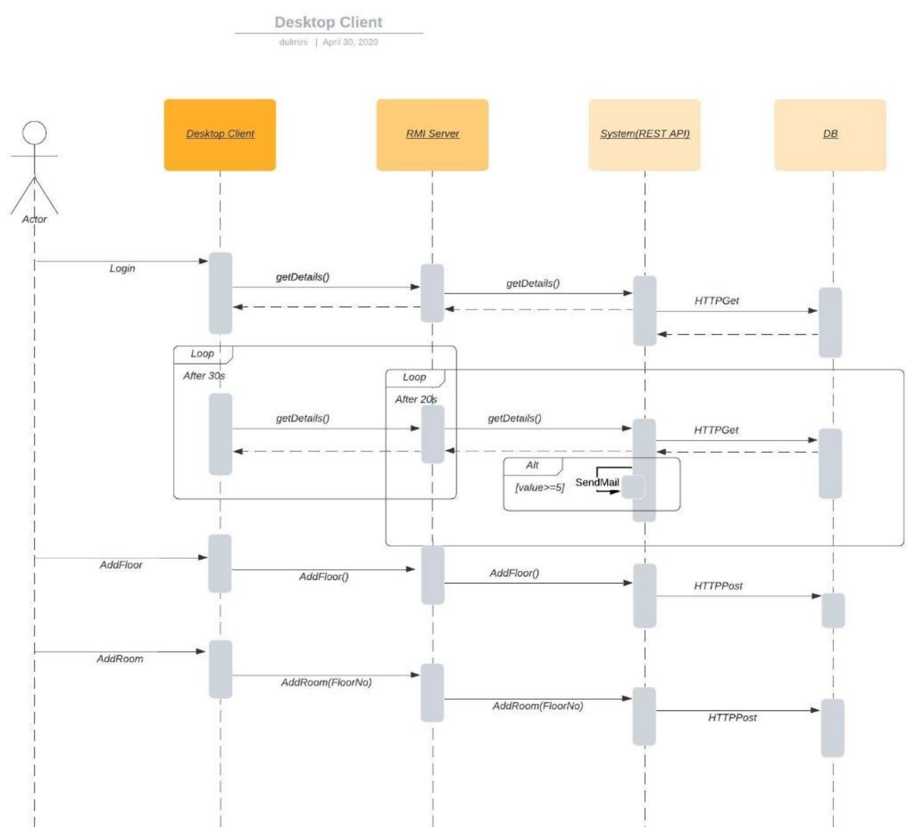
- Get details from the sensor in every 40 seconds.
- Send data to the web Client
- Send data to RMI Server
- Send email to the administrator when called in the RMI server.
- Send message to the administrator when called in the RMI server.

The Email and SMS services are embedded in the REST API.
The Email service uses NodeMailer.
The SMS service uses Twilio API.

Workflow:

- There are a couple of ways that data (in a JSON object) comes to the REST API and those are updated Sensor data and also as new data from the RMI server and sensor edited details. (Using HTTP request)
- The REST API processes data according to the routes and direct them to Email and SMS services which are triggered due to the RMI server calls.
- Update the database/ retrieve from the database
- Send back response as a JSON object

Desktop Client



The RMI server and the desktop client are developed using Java and Java Swing. Only the administrator can be logged into the Desktop Client.

Username: admin

Password: 123

Desktop Client Services

- Calling the RMI server.
- Displaying the information.
- Red boarder if any value is greater than 5.
- Blue boarder if there are any changes.

When the RMI server starts running it will call the REST API and will retrieve the data from the database coming through the REST API. These retrieved data will be saved in the RMI server.

After logging in to the Desktop client, the desktop client calls the RMI server through a remote method.

Then all the saved data in the RMI server will be sent to the desktop client by sending all inside a variable.

Admin can add a new floor and a new room to the system.

Admin can also change the state to switch on or off of a certain room in a floor. (This was taken as a static variable to satisfy the administrator can edit details requirement. Since all the details are anyway changing through the sensor dummy there is no point of changing details like CO2 level and smoke level.)

Then the Desktop client will call the remote method of a RMI server through the remote interface.

The desktop client will refresh the data every 30 seconds. If there was a change in data it will show a boarder with blue color and will show a red color boarder if the carbon dioxide level or smoke level is more than 5.

If the sensor is deactivated there will be no data shown in the relevant room.

RMI Server

RMI server is made using Java technology. This acts as a gateway for the Desktop client. All the data from the client to the REST API are communicated through this RMI Server. RMI server implements a remote interface which the Desktop client uses to call the methods from.

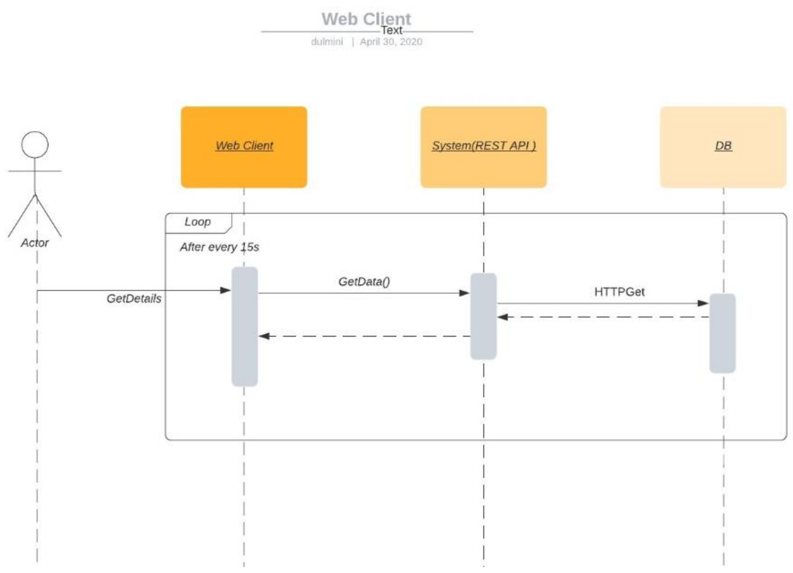
RMI Server services

- RMI Server provides the data in the database by retrieving them from the REST API.
- It provides interfaces to get those data for the desktop client by creating a remote communication between them.
- It sends updated the data by the desktop client to the REST API.
- It invokes the email and SMS service which get through REST API.

Workflow:

- RMI server gets data through REST API using http request within every 15 seconds and saves those data to a variable.
- Within this every 15 seconds RMI server invokes the email and the SMS service methods and sends respective messages about the details to the administrator.
- When RMI registry is activated, it builds a remote connection with the desktop client and provides interfaces with invoke methods to be used by the desktop client.
- The updating data such as adding rooms and floors and changing the states of sensors are passed through the RMI service back to the REST API using http response.

Web Client



ReactJS has been used to make the web client along with Axios to implement the connection to the REST API. Once the user runs the web client, all the data will be taken from MongoDB and will be displayed in the web application.

When the web user opens the web application, he will get the details on the rooms in a tabular form. The active rooms will be displayed in green color while the non-active ones will be displayed in yellow. The rooms that are available to a floor will be colored and the rest will be displayed as a white space.

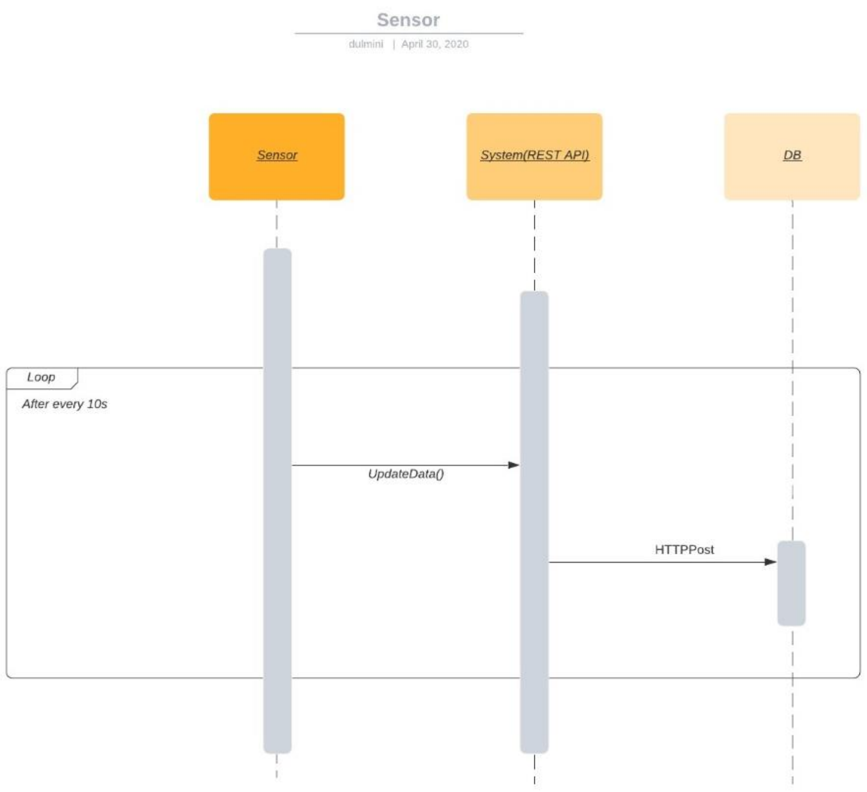
Web Client services

- Call the REST API through axios
- Display the retrieved data to the client

Workflow:

- Every 15 seconds Web client gets data through REST service using HTTP request

Sensor



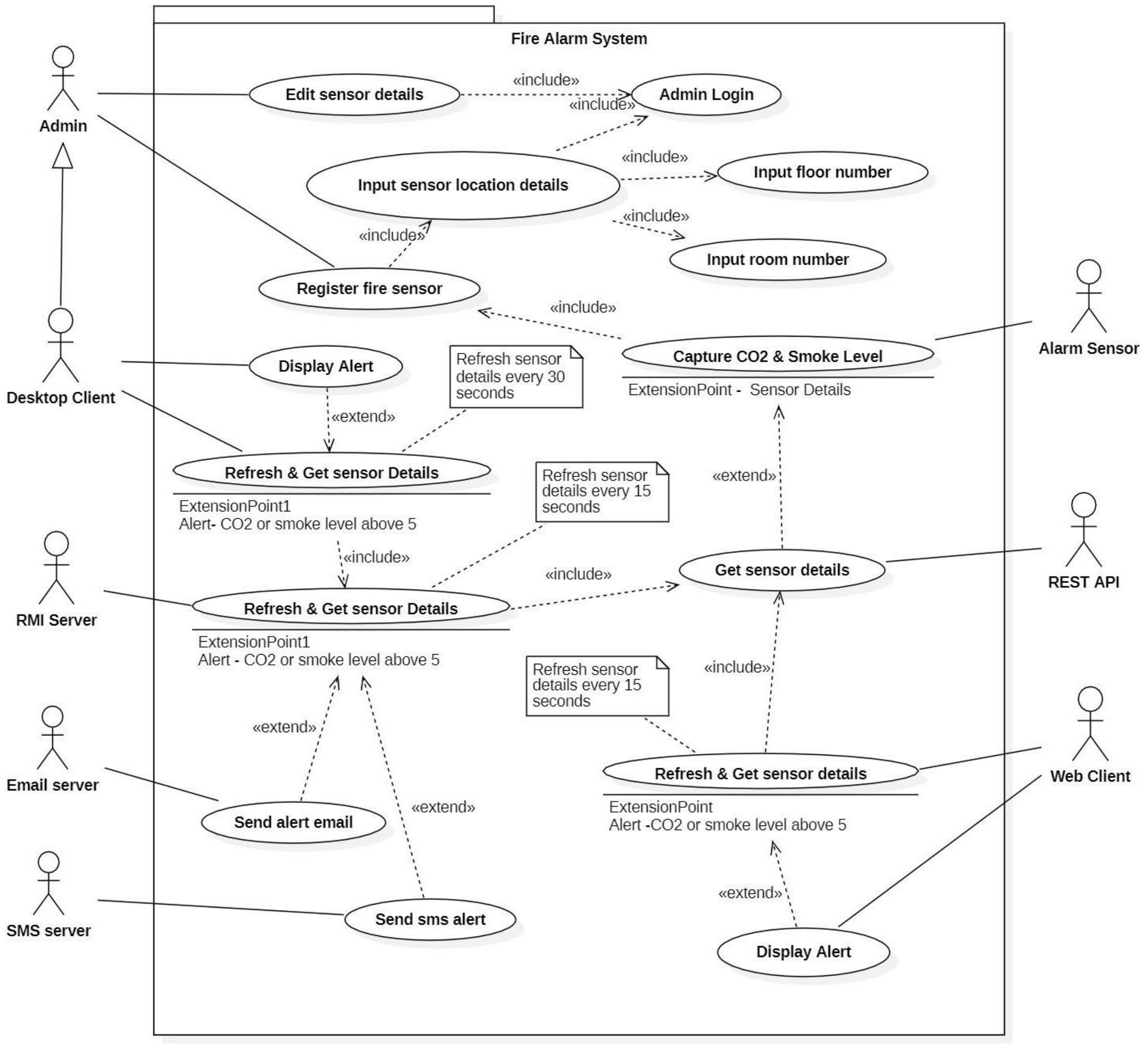
The sensor is created using nodeJS. As a dummy service, Sensor creates random smoke and Co2 levels by choosing a random room on the floor. After every 30 seconds two rooms update their smoke and Co2 levels and using the REST API interface it updates the database.

Workflow:

- Data flows only one way.
- Sensor calls rest API with dummy data

Other Diagrams

Use case diagrams to understand system scope and identify the functionalities.


Fire Alarm System
Admin
Desktop Client
RMI Server
Email server
SMS server
Alarm Sensor
REST API
Web Client
Edit sensor details
Admin Login
Input sensor location details
Input floor number
Input room number
Register fire sensor
Capture CO2 & Smoke Level
ExtensionPoint - Sensor Details
Display Alert
Refresh sensor details every 30 seconds
Refresh & Get sensor Details
ExtensionPoint1
Alert- CO2 or smoke level above 5
Refresh sensor details every 15 seconds
Refresh & Get sensor Details
ExtensionPoint1
Alert - CO2 or smoke level above 5
Get sensor details
Refresh sensor details every 15 seconds
Refresh & Get sensor details
ExtensionPoint
Alert -CO2 or smoke level above 5
Send alert email
Send sms alert
Display Alert
«include»
«extend»

Appendix

Code without import statement & Web client's HTML code

REST API

```
const config = dotenv.config().parsed;
const client = new Twilio(config.TWILIO_ACCOUNT_SID,
config.TWILIO_AUTH_TOKEN);

router.route('/all').get((req, res) => {
  Floor.find()
     .then(Floors => res.json(Floors))
     .catch(err => res.status(400).json('Error: ' + err));
});

router.route('/addFloor').post(async (req, res) => {
  const data = await
axios.get(`http://localhost:5000/getFloorCount`).then((resp
onse) => {
     console.log(response.data.count);
     const FloorNo = response.data.count + 1;
     const Rooms = [];
     const newFloor = new Floor({FloorNo, Rooms});

     newFloor.save().then(() => res.json('Floor
added')).catch(err => res.status(400).json('Error' + err));
  });
});

router.route('/addRoom/:no').post(async (req, res) => {
  const data = await
axios.get(`http://localhost:5000/getRoomsCount/${req.par
ams.no}`).then((response) => {
     Floor.update({FloorNo: req.params.no}, {
        $push: {
          Rooms: {
             "RoomNo": response.data.count + 1,
             "Active": true,
             "SmokeLevel": 0,
             "CO2Level": 0
          }
        }
     }).then(() => res.json('Done'));
  });
});

router.route('/getRoomsCount/:no').get((req, res) => {
  Floor.findOne({FloorNo: req.params.no})
     .then(Floor => {
        res.send({count: Floor.Rooms.length})
     });
});

router.route('/getFloorCount').get((req, res) => {
  Floor.find()
     .then(Floors => res.send({count: Floors.length}))
     .catch(err => res.status(400).json('Error: ' + err));
})

router.route('/update').post((req, res) => {
  Floor.findOneAndUpdate(
     {FloorNo: req.body.FloorNo, "Rooms.RoomNo":
req.body.RoomNo},
     {
        $set: {
          "Rooms.$.CO2Level": req.body.co2L,
          "Rooms.$.SmokeLevel": req.body.smL
        }
     },
     {new: true})
     .then(() => {
        res.sendStatus(200);
     })
     .catch(err => {
        console.error(err);
     });
});

router.route('/Off/:FloorNo/:RoomNo').post((req, res) => {

  Floor.findOneAndUpdate(
     {FloorNo:parseInt( req.params.FloorNo),
"Rooms.RoomNo": parseInt(req.params.RoomNo)},
     {
        $set: {
          "Rooms.$.Active": false
        }
     },
     {new: true})
     .then(() => {
        res.sendStatus(200);
     })
     .catch(err => {
        console.error(err);
     });
});

router.route('/On/:FloorNo/:RoomNo').post((req, res) => {
  Floor.findOneAndUpdate(
     {FloorNo:parseInt( req.params.FloorNo),
"Rooms.RoomNo": parseInt(req.params.RoomNo)},
     {
        $set: {
          "Rooms.$.Active": true
        }
     },
     {new: true})
     .then(() => {
        res.sendStatus(200);
     })
     .catch(err => {
        console.error(err);
     });
});

router.route('/MaxRoomCount').get((req, res) => {
  Floor.find()
     .then(Floors => {
        let maximumRoom = 0;
        Floors.map(floor => {
          if (floor.Rooms.length > maximumRoom) {
             maximumRoom = floor.Rooms.length
          }
        });
        res.send({maximumRoom: maximumRoom})
     })
     .catch(err => res.status(400).json('Error: ' + err));
});

router.route('/MailSender').get(async (req,res)=>{
  try {
      const res = await axios.get('http://localhost:5000/all');
      let floors = res.data;
      let listOfRooms = ``;

      floors.map((floor, i) => {
 listOfRooms += `Floor No:  ${i + 1}`;
         floor.Rooms.map((room, i) => {
            if (room.Active) {
               if (room.SmokeLevel > 5 || room.CO2Level >
5)
                  listOfRooms += `\nroom No: ${i + 1} `
               if (room.SmokeLevel > 5) {
                  listOfRooms += `#Smoke Level- warning\t`
               }
               if (room.CO2Level > 5) {
                  listOfRooms += `#CO2 Level- warning `
               }
            }
         })
         listOfRooms += "\n------------------------------------\n"
      })
      MailSender(listOfRooms);
   } catch (err) {
      console.error(err);
   }
   res.sendStatus(200);
});

router.route('/SMS-Sender').get(async (req,res)=>{
   try {
      const res1 = await axios.get('http://localhost:5000/all');
      let floors = res1.data;
      let listOfRooms = ``;

      floors.map((floor, i) => {
         listOfRooms += `Floor No:  ${i + 1}`;
         floor.Rooms.map((room, i) => {
            if (room.Active) {
               if (room.SmokeLevel > 5 || room.CO2Level >
5)
                  listOfRooms += `\nroom No: ${i + 1} `
               if (room.SmokeLevel > 5) {
                  listOfRooms += `#Smoke Level- warning\t`
               }
               if (room.CO2Level > 5) {
                  listOfRooms += `#CO2 Level- warning `
               }
            }
         })
         listOfRooms += "\n------------------------------------\n"
      })
      sendSms(listOfRooms,() => {
         console.log("successfully send this message \n " +
listOfRooms);
      })
   } catch (err) {
      console.error(err);
   }
   res.sendStatus(200);
});

function MailSender(text) {
   const transporter = nodemailer.createTransport({
      service: 'gmail',
      auth: {
         user: 'hanger24x7@gmail.com',
         pass: '1qaz2wsx@'
      }
   });
   var mailOptions = {
      from: 'hanger24x7@gmail.com',
      //change email address to test it
      to: 'lahirulakshan780@gmail.com',
      subject: 'Details From Sensors',
      text: text
   };

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

function sendSms(bodyMessage, done){

   let message = {
      to: config.TO_PHONE_NUMBER,
      from: config.FROM_PHONE_NUMBER,
      body: bodyMessage
   };

   client.messages.create(message, (err, message) => {
      if (err) return done(err);

      return done(null, message);
   });

}
module.exports = router;
```

Sensor Dummy

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

setInterval( function () {
   for (i=0;i<2;i++) {
      axios.get(`http://localhost:5000/all`).then(async
(response) => {
         const Floors = response.data;
         let FNo = Math.floor(Math.random() *
Floors.length);
         let RNo = Math.floor(Math.random() *
Floors[FNo].Rooms.length);

         let col2L = Math.floor(Math.random() * 10) + 1;
         let smL = Math.floor(Math.random() * 10) + 1;

         console.log(`Changing Smoke Level & CO2 Level
at Floor no: ${Floors[FNo].FloorNo} - Room no:
${Floors[FNo].Rooms[RNo].RoomNo} - Smoke Level:
${smL} - CO2 Level: ${col2L}`)

         try {
            const res = await
axios.post('http://localhost:5000/update', {
               "FloorNo": Floors[FNo].FloorNo,
               "RoomNo":
Floors[FNo].Rooms[RNo].RoomNo,
               "co2L": col2L,
               "smL": smL
            });
         } catch (err) {
            console.error(err);
          }

        }).catch((err) => {
          console.error(err);
        });
    }
}, 15000);
```

Web Client

```
class App extends Component{
 constructor(props) {
   super(props);
   this.state={
     FloorNo:0,
     NoofRooms:0,
     NoofRoomsList:[],
     RoomsCount:0,
     blocks:[{Rooms:[{Active:false},{Active: false}, {Active:
false}, {Active: false}, {Active: false}, {Active: false}],},
             {Rooms:[{Active:false},{Active: false}, {Active:
false}, {Active: false}, {Active: false}, {Active: false}],},
             {Rooms:[{Active:false},{Active: false}, {Active:
false}, {Active: false}, {Active: false}, {Active: false}],},
             {Rooms:[{Active:false},{Active: false}, {Active:
false}, {Active: false}, {Active: false}, {Active: false}],},
             {Rooms:[{Active:false},{Active: false}, {Active:
false}, {Active: false}, {Active: false}, {Active: false}],},
     ]}
 }
 getDetails() {
     //Get all the details from the REST API
   axios.get("http://localhost:5000/all").then(res => {
     this.setState({
       blocks: res.data
     });
     this.getFloorCount();
     this.getNoofRooms();

   });

 }
 getFloorCount(){
     //Getting the floor count from the REST API
   axios.get(`http://localhost:5000/getFloorCount`).then((re
sponse) => {
     this.setState({
       FloorNo:response.data.count + 1
     });
   });
 }

 getNoofRooms(){
     //Getting the maximum number of rooms compared to
rooms in each floor
     axios.get(`http://localhost:5000/MaxRoomCount/`).then
((response) => {
         this.setState({
           NoofRooms:response.data.maximumRoom,
         }, ()=>{
           for(var i=0; i<response.data.maximumRoom; i++){
             this.state.NoofRoomsList.push(i);
           }
         });
         console.log(this.state.NoofRooms);
     });
   }
```

```
 componentDidMount() {
     //Getting details when the component is mounted
     //Calling the getDetails method after 15s
   this.getDetails();
   setInterval(()=>{
     this.getDetails()
   }, 15000)
 }


 render() {

   return (
     <div>
       <table className="table table-bordered">
         <thead>
         <tr className="table-dark">
           <th>Number</th>
           {this.state.blocks.map((block,i)=>{
             return(
                 <th>
                   Room {i+1}
                 </th>
             )
           })}
           <th>Room 8</th>
         </tr>
         </thead>
         <tbody>
         {/*Sending data in a floor to the Block.js */}
         {this.state.blocks.map((block,i)=>{
           return(
               <tr>
                 <td className="table-dark" >Floor {i+1}</td>
             <Block o={this.state.blocks[i]} header={i} />
           </tr>
           )
         })}
         </tbody>
       </table>
       <div className="text-dark">
         Details:
         <p>If the sensor is active : It will be displayed in
Green Color</p>
         <p>If the sensor is not active : It will be displayed in
Yellow Color</p>
           <p>If there are no more rooms in the floor, rest
will be displayed in White Color</p>
         <p>If the CO2(Carbondioxide) level is higher than
or equal to level 5 : It will be displayed in Red Color</p>
         <p>If the Smoke level is higher than or equal to
level 5 : It will be displayed in Red Color</p>
       </div>
     </div>
   );
 }


}

export default App;

import React, {Component} from "react";
import BlockUnit from "./blockUnit";

export default class Block extends Component{
   constructor(props) {
        super(props);
        this.state={
            object:this.props.obj,
            header:false,
            length:0,
            blocks:[]
        }
    }
    componentWillReceiveProps(nextProps, nextContext) {
        //Getting the properties from the App.js
        this.setState({
            blocks:nextProps.o.Rooms,
            header:nextProps.header
        })

    }

    getBlockUnit(){
        //Sending the room data in the received floor data to
BlockUnit
        return this.state.blocks.map((res, i)=>{
            return  <BlockUnit header={this.state.header}
h={this.props.header} obj={this.state.blocks[i]} />
        })
    }

    render() {
        return(
            //{header? (<th scope="row" className="table-
active">Floor {h+1}</th>):(null)}
            this.getBlockUnit()
        );
    }
}

import React, {Component} from "react";

export default class BlockUnit extends Component{
    constructor(props) {
        super(props);
        this.state={
            object:this.props.obj,
            co2danger:false,
            smokedanger:false,
            length:0,
            blocks:[]
        }
    }

    componentWillReceiveProps(nextProps, nextContext) {
        //Getting the properties from Block
        this.setState({
            co2danger:false,
            smokedanger:false,
            blocks:nextProps.obj,
            /*length:nextProps.o.length*/
        });
        if(nextProps.obj.CO2Level>=5){
            this.setState({
                co2danger:true
            })
        }
        if(nextProps.obj.SmokeLevel>=5){
            this.setState({
                smokedanger:true
            })
        }
    }
    render() {
        const {  header, h , obj, co2, smoke}=this.props;
        return(
            <td className={this.state.blocks.Active? 'table-
success':'table-warning'} >
                <div style={{height: "150px", width:"150px"}}>
                    {this.state.blocks.Active ?
                        (<div>
                            <p>Floor = {this.props.h+1}  </p>
                            <p>Room =
{this.props.obj.RoomNo}  </p>
                            <p className={this.state.co2danger ?
'table-danger' : 'none'}>CO2 Level =
{this.props.obj.CO2Level}  </p>
                            <p
className={this.state.smokedanger ? 'table-danger' :
'none'}>Smoke Level = {this.props.obj.SmokeLevel}</p>
                        </div>) : (<div>
                            <p>Not Active</p>
                        </div>)
                    }
                </div>
            </td>
        );
    }
}
```

RMI Server

```
public class Server extends UnicastRemoteObject
implements Service {

        static int delay = 0; // delay for 0 sec.
        static int period = 15000; // repeat every 15 sec.
        static Timer timer = new Timer();
        String allData = "[{}]";
        int FloorCount = 0;
        int MaxRoomCount = 0;
        protected Server() throws RemoteException {
                super();

                timer.scheduleAtFixedRate(new
TimerTask()  // used for reload frame every 15 second
                                        {

          public void run()

          {
                callApi();
        }}, delay, period);
        }

        public static void main(String[] args) {
                // TODO Auto-generated method stub
                // set the policy file as the system
security policy

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

                try {
                        Server svr = new Server();
                         // Bind the remote object's
stub in the registry
                        Registry registry =
LocateRegistry.getRegistry();

        registry.bind("LevelService", svr);
```

```
                        System.out.println("Service
Strated........");
                }catch(RemoteException re){
      System.err.println(re.getMessage());
    }
    catch(AlreadyBoundException abe){
      System.err.println(abe.getMessage());
    }

        }

        @Override
        public String Getdata() throws RemoteException
{
                // TODO Auto-generated method stub
                return allData;
                        }

        @Override
        public int getFloorCount() throws
RemoteException {
                // get number of floors saved in
database
                return FloorCount;
        }

        @Override
        public int getRoomCount(String floornum)
throws RemoteException {
                // get numbers of rooms according to
relevant floor number
                String cnt = "";
                int roomcount = 0;
                try {
                        URL url = new
URL("http://localhost:5000/getRoomsCount/" + floornum);
            HttpURLConnection conn =
(HttpURLConnection) url.openConnection();
            conn.setRequestMethod("GET");
            conn.setRequestProperty("Accept",
"application/json");

            if (conn.getResponseCode() != 200) {
              throw new RuntimeException("Failed :
HTTP error code : "
                  + conn.getResponseCode());
            }
     //get JSON type data to string
            Scanner sc = new
Scanner(url.openStream());
            while(sc.hasNext()) {
              cnt += sc.nextLine();
            }

            //create JSON type object by string
            JSONObject jb = new JSONObject(cnt);
            roomcount = jb.getInt("count");
            conn.disconnect();

           } catch (MalformedURLException e) {

            e.printStackTrace();

           } catch (IOException e) {

            e.printStackTrace();

           } catch (JSONException e) {
                        // TODO Auto-generated
catch block
                        e.printStackTrace();
                }

                return roomcount;
        }

        @Override
        public void AddRoom(String floornum) throws
RemoteException {
                // server generate the next room
number according to given floor number and save in db

                try {

                        HttpClient client =
HttpClient.newHttpClient();
                      HttpRequest request =
HttpRequest.newBuilder()

                .uri(URI.create("http://localhost:5000/a
ddRoom/" + floornum ))

                .POST(HttpRequest.BodyPublishers.o
fString(""))
                        .build();

                      HttpResponse<String> response
= client.send(request,

                HttpResponse.BodyHandlers.ofString(
));


            System.out.println(response.body());
                callApi();
                } catch (Exception e) {
                        // TODO Auto-generated
catch block
                        e.printStackTrace();
                }

        }

        @Override
        public void AddFloor() throws RemoteException{
                // server generate the next floor
number and save in db
                try {

                        HttpClient client =
HttpClient.newHttpClient();
                      HttpRequest request =
HttpRequest.newBuilder()

                .uri(URI.create("http://localhost:5000/a
ddFloor"))

                .POST(HttpRequest.BodyPublishers.o
fString(""))
                        .build();

                      HttpResponse<String> response
= client.send(request,

                HttpResponse.BodyHandlers.ofString(
));
```

```
                        System.out.println(response.body());
                                    callApi();

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

            public void callApi() {

                        try {
                                    //-------------Get All Data by
Server -----------

                        URL url = new
URL("http://localhost:5000/all");
        HttpURLConnection conn = (HttpURLConnection)
url.openConnection();
        conn.setRequestMethod("GET");
        conn.setRequestProperty("Accept",
"application/json");

        if (conn.getResponseCode() != 200) {
            throw new RuntimeException("Failed : HTTP error
code : "
                    + conn.getResponseCode());
        }
        allData = "";
        // assign JSON type array to string
        Scanner sc = new Scanner(url.openStream());
        while(sc.hasNext()) {
            allData += sc.nextLine();
        }
        System.out.println(allData);
        conn.disconnect();

        //----------------Get Floors -------------------
        String fcnt = "";
                        URL url1 = new
URL("http://localhost:5000/getFloorCount");
                        HttpURLConnection conn1 =
(HttpURLConnection) url1.openConnection();
                        conn1.setRequestMethod("GET");
                        conn1.setRequestProperty("Accept",
"application/json");

                        if (conn1.getResponseCode() != 200) {
                            throw new RuntimeException("Failed :
HTTP error code : "
                                    + conn1.getResponseCode());
                        }

                        Scanner sc1 = new
Scanner(url1.openStream());
                        while(sc1.hasNext()) {
                            fcnt += sc1.nextLine();
                        }

                        JSONObject jb = new JSONObject(fcnt);
                        FloorCount = jb.getInt("count");
                        conn1.disconnect();

                        //------------------- Get MaxRoom ----------------
                        String maxroomcnt = "";
                                URL url2 = new
URL("http://localhost:5000/MaxRoomCount");
                        HttpURLConnection conn2 =
(HttpURLConnection) url2.openConnection();
                        conn2.setRequestMethod("GET");
                        conn2.setRequestProperty("Accept",
"application/json");

                        if (conn2.getResponseCode() != 200) {
                            throw new RuntimeException("Failed :
HTTP error code : "
                                    + conn2.getResponseCode());
                        }

                        Scanner sc2 = new
Scanner(url2.openStream());
                        while(sc2.hasNext()) {
                            maxroomcnt += sc2.nextLine();
                        }

                        JSONObject jb1 = new
JSONObject(maxroomcnt);
                        MaxRoomCount =
jb1.getInt("maximumRoom");
                        conn2.disconnect();

                        //------- Send Email -------------
                        URL url3 = new
URL("http://localhost:5000/MailSender");
                        HttpURLConnection conn3 =
(HttpURLConnection) url3.openConnection();
                        conn3.setRequestMethod("GET");
                        conn3.setRequestProperty("Accept",
"application/json");
                        url3.openStream();
                        conn3.disconnect();


                                    //------ Send Message -------
---
                                    URL url4 = new
URL("http://localhost:5000/SMS-Sender");
                HttpURLConnection conn4 =
(HttpURLConnection) url4.openConnection();
                conn4.setRequestMethod("GET");
                conn4.setRequestProperty("Accept",
"application/json");
                url4.openStream();
                conn4.disconnect();



                        } catch (MalformedURLException e) {

    e.printStackTrace();

   } catch (IOException e) {

    e.printStackTrace();

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

            }

            @Override
            public int getMaxRoomCOunt() throws
RemoteException {
                        return MaxRoomCount;
```

```
	}

	@Override
	public void SensorOff(String floornum, String
roomnum) throws RemoteException {
		try {

			HttpClient client =
HttpClient.newHttpClient();
			HttpRequest request =
HttpRequest.newBuilder()

		.uri(URI.create("http://localhost:5000/
Off/" +floornum + "/" +roomnum))

		.POST(HttpRequest.BodyPublishers.o
fString(""))
				.build();

			HttpResponse<String> response
= client.send(request,

		HttpResponse.BodyHandlers.ofString(
));


		System.out.println(response.body());
				callApi();

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

	@Override
	public void SensorOn(String floornum, String
roomnum) throws RemoteException {
		try {

			HttpClient client =
HttpClient.newHttpClient();
			HttpRequest request =
HttpRequest.newBuilder()

		.uri(URI.create("http://localhost:5000/
On/" +floornum + "/" +roomnum))

		.POST(HttpRequest.BodyPublishers.o
fString(""))
				.build();

			HttpResponse<String> response
= client.send(request,

		HttpResponse.BodyHandlers.ofString(
));


		System.out.println(response.body());
				callApi();

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

```
public interface Service  extends Remote{

	public String Getdata() throws
RemoteException;
	public int getFloorCount() throws
RemoteException;
	public int getRoomCount(String floornum)
throws RemoteException;
	public void AddRoom(String floornum) throws
RemoteException;
	public void AddFloor() throws RemoteException;
	public int getMaxRoomCOunt() throws
RemoteException;
	public void SensorOff(String floornum, String
roomnum) throws RemoteException;
	public void SensorOn(String floornum, String
roomnum) throws RemoteException;
}
```

Desktop Client

```
public class Login {

	private JFrame frame;
	private JTextField txtUsername;
	private JPasswordField pwdPassword;

	/**
	 * Launch the application.
	 */
	public static void main(String[] args) {
		EventQueue.invokeLater(new
Runnable() {
			public void run() {
				try {
					Login
window = new Login();

	window.frame.setVisible(true);
				} catch
(Exception e) {

	e.printStackTrace();
				}
			}
		});
	}

	/**
	 * Create the application.
	 */
	public Login() {
		initialize();
	}

	/**
	 * Initialize the contents of the frame.
	 */
	private void initialize() {
		frame = new JFrame();
		frame.setBounds(100, 100, 450, 300);

	frame.setDefaultCloseOperation(JFrame.EXIT_
ON_CLOSE);

	frame.getContentPane().setLayout(null);
```

```
				JLabel lblLogin = new
JLabel("Login");
				lblLogin.setBounds(189, 10, 63, 19);

		frame.getContentPane().add(lblLogin);

				JLabel lblUserName = new
JLabel("User Name");
				lblUserName.setBounds(81, 65, 63,
13);

		frame.getContentPane().add(lblUserName);

				JLabel lblPassword = new
JLabel("Password");
				lblPassword.setBounds(81, 119, 63,
13);

		frame.getContentPane().add(lblPassword);

				txtUsername = new JTextField();
				//txtUsername.setText('');
				txtUsername.setBounds(189, 62, 96,
19);

		frame.getContentPane().add(txtUsername);
				txtUsername.setColumns(10);

				pwdPassword = new
JPasswordField();
				//pwdPassword.setText("Password");
				pwdPassword.setBounds(189, 116,
96, 19);

		frame.getContentPane().add(pwdPassword);

				JButton btnSubmit = new
JButton("Login");
				btnSubmit.addActionListener(new
ActionListener() {
						public void
actionPerformed(ActionEvent e) {

		if(txtUsername.getText().toString().equalsIgnore
Case("admin") &&
pwdPassword.getText().toString().equalsIgnoreCase("123
") ) {

		ClientView cv = new ClientView();

		cv.setVisible(true);

		frame.setVisible(false);
								}else {

		JOptionPane.showMessageDialog(null, "Invalid
Login Detais","Login
Error",JOptionPane.ERROR_MESSAGE);
								}
						}
				});
				btnSubmit.setBounds(167, 171, 85,
21);

		frame.getContentPane().add(btnSubmit);
		}
}
```

```
public class ClientView extends JFrame {

		private JPanel contentPane;
		static int delay = 0; // delay for 0 sec.
		static int period = 30000; // repeat every 30 sec.
		static Timer timer = new Timer();
		 static ClientView frame ;
		 int mxrm = 10;

		public static void main(String[] args) {
				EventQueue.invokeLater(new
Runnable() {
						public void run() {
								try {
										frame
= new ClientView();

		frame.setVisible(true);

										} catch
(Exception e) {

		e.printStackTrace();
								}
						}
				});
		}

		/**
		 * Create the frame.
		 */
		JSONArray temparray = null;	// used for check
data changes after refresh
		JSONArray jsonarry = null;
		public ClientView() {


				timer.scheduleAtFixedRate(new
TimerTask()  // used for reload frame every 15 second
				{
				  public void run()
				   {


		setDefaultCloseOperation(JFrame.EXIT_ON_C
LOSE);
								setBounds(20,
20, 1325, 732);
								contentPane =
new JPanel();

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

		setContentPane(contentPane);

		contentPane.setLayout(null);


								JButton
btnAddFloor = new JButton("Add"); // button to go to add
floor or add room frame

		btnAddFloor.setBounds(805, 10, 95, 27);

		getContentPane().add(btnAddFloor);

		btnAddFloor.addActionListener(new
ActionListener() {
```

```
                                                public
void actionPerformed(ActionEvent e) {

          FloorAdd fa = new FloorAdd();

          fa.setVisible(true);

          setVisible(false);


                                                }
                                      });

                                      JButton
btnChangeState = new JButton("Change State"); // button
to change state of sensors

          btnChangeState.setBounds(920, 10, 125, 27);

          getContentPane().add(btnChangeState);

          btnChangeState.addActionListener(new
ActionListener() {
                                                public
void actionPerformed(ActionEvent e) {

          ChangeState cs = new ChangeState();

          cs.setVisible(true);

          setVisible(false);


                                                }
                                      });
                                      int p = 185;
                                      for(int v=0;
v<mxrm; v++) {  // create Room number labels
                                      JLabel
lblNewLabel_1 = new JLabel("Room " + (v+1));

          lblNewLabel_1.setFont(new Font("Times New
Roman", Font.BOLD | Font.ITALIC, 15));

          lblNewLabel_1.setBounds(p, 42, 81, 20);

          contentPane.add(lblNewLabel_1);
                                      p= p + 112;
                                      }


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

               Service service = null;

               try {

                 service = (Service)
Naming.lookup("//localhost/LevelService");
                 mxrm = service.getMaxRoomCOunt();
                 String alldata = service.Getdata(); //call
get all data method by rmi server
                         jsonarry = new
JSONArray(alldata); // assign return string value to  JSON
array

                 int y = 72;
```

```
             for(int j=0; j<jsonarry.length(); j++) {

                  JLabel jmain = new JLabel(); //
create rows according to floor numbers
                  jmain.setFont(new Font("Tahoma",
Font.PLAIN, 15));
                  jmain.setBorder(new
LineBorder(new Color(0, 0, 0)));
                  jmain.setBounds(25, y, 132, 72);
                  contentPane.add(jmain);
                          jmain.setText("FLOOR    "
+  String.valueOf(jsonarry.getJSONObject(j).getInt("FloorN
o")));
                          int x = 167;

                          for(int i=0;
i<jsonarry.getJSONObject(j).getJSONArray("Rooms").leng
th(); i++) {
                                 // create a label
to view  smoke level and co2 level, position of this label
change with floor and room of each floor
                          JLabel jlbSmLvl = new
JLabel("");
                          jlbSmLvl.setFont(new
Font("Tahoma", Font.PLAIN, 14));
                          jlbSmLvl.setBorder(new
LineBorder(new Color(0, 0, 0)));
                          jlbSmLvl.setBounds(x, y,
102, 32);
                    contentPane.add(jlbSmLvl);

                          JLabel jlCLvl = new
JLabel("");
                          jlCLvl.setFont(new
Font("Tahoma", Font.PLAIN, 14));
                          jlCLvl.setBorder(new
LineBorder(new Color(0, 0, 0)));
                          jlCLvl.setBounds(x, y+42,
102, 32);
                          contentPane.add(jlCLvl);
                                //get values for
relevant room by JSON array
                                    int
smokelevel =
jsonarry.getJSONObject(j).getJSONArray("Rooms").getJS
ONObject(i).getInt("SmokeLevel");
                                    int c02level
=
jsonarry.getJSONObject(j).getJSONArray("Rooms").getJS
ONObject(i).getInt("CO2Level");


          jlbSmLvl.setText("Smoke Level:" +
String.valueOf(smokelevel));

         jlCLvl.setText("Co2 Level  : "
+String.valueOf(c02level));

                                  boolean active
=
jsonarry.getJSONObject(j).getJSONArray("Rooms").getJS
ONObject(i).getBoolean("Active");
                                  //check sensor
activation
                                  if(!active) {

        jlbSmLvl.setText(null);

        jlCLvl.setText(null);
```

```
				jlbSmLvl.setBorder(new LineBorder(Color.RED));

				jlCLvl.setBorder(new LineBorder(Color.RED));
							}

							if(smokelevel >= 5) jlbSmLvl.setForeground(Color.RED); else jlbSmLvl.setForeground(Color.GREEN);
							if(c02level >= 5) jlCLvl.setForeground(Color.RED); else jlCLvl.setForeground(Color.GREEN);
							x = x +112;

							if(temparray != null && active) {

								if(temparray.getJSONObject(j).getJSONArray("Rooms").getJSONObject(i).getInt("SmokeLevel") != smokelevel)

									jlCLvl.setBorder(new LineBorder(Color.BLUE));

								if(temparray.getJSONObject(j).getJSONArray("Rooms").getJSONObject(i).getInt("CO2Level") != c02level)

									jlCLvl.setBorder(new LineBorder(Color.BLUE));
							}

						}
						y= y+86;
					}


				} catch (NotBoundException ex) {
					System.err.println(ex.getMessage());
				} catch (MalformedURLException ex) {
					System.err.println(ex.getMessage());
				} catch (RemoteException ex) {
					System.err.println(ex.getMessage());
				} catch (JSONException e) {
					// TODO Auto-generated catch block

				e.printStackTrace();
				}
				temparray = jsonarry;
			}}, delay, period);


	}


}

public class FloorAdd extends JFrame {

	private JPanel contentPane;
	static FloorAdd frame;
	/**
	 * Launch the application.
	 */
	public static void main(String[] args) {
		EventQueue.invokeLater(new Runnable() {
			public void run() {
				try {
					frame = new FloorAdd();

					frame.setVisible(true);
				} catch (Exception e) {

					e.printStackTrace();
				}
			}
		});
	}

	Service service = null;
	public FloorAdd() {
	  System.setProperty("java.security.policy", "file:allowall.policy");

	  int fct = 0;
		try {
			service = (Service) Naming.lookup("//localhost/LevelService");
			fct =service.getFloorCount();  // get number of floor by RMI SERVER

		}catch(Exception e) {

		e.printStackTrace();
		}

		String Floornum[] = new String[fct];
		//create string array by number of floors
		for(int f=0; f<fct ; f++) {
			Floornum[f] = String.valueOf(f+1);
		}


		setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
		setBounds(100, 100, 654, 464);
		contentPane = new JPanel();
		contentPane.setBorder(new EmptyBorder(5, 5, 5, 5));
		setContentPane(contentPane);
		contentPane.setLayout(null);
		//create button to add new floor
		JButton btnNewButton = new JButton("Add New Floor");
		btnNewButton.addActionListener(new ActionListener() {
			public void actionPerformed(ActionEvent e) {
				int a = JOptionPane.showConfirmDialog(contentPane, "Are You sure to add new floor ", "Confirm Add Floor", JOptionPane.YES_NO_OPTION);

				if(a==JOptionPane.YES_OPTION){
					try {

						service.AddFloor();

						ClientView c = new ClientView();

						c.setVisible(true);

						setVisible(false);
```

```
        }catch(Exception ec) {

        ec.printStackTrace();
                                }
                        }
                }
                });
                btnNewButton.setFont(new
Font("Ubuntu Mono", Font.BOLD | Font.ITALIC, 20));

        btnNewButton.setBackground(Color.RED);
                btnNewButton.setBounds(174, 26,
292, 45);
                contentPane.add(btnNewButton);
                //---------------add new Room -------------
--------
                JLabel lblNewLabel = new
JLabel("Add New Room ");
                lblNewLabel.setFont(new Font("Tw
Cen MT Condensed", Font.ITALIC, 21));

        lblNewLabel.setForeground(Color.BLUE);
                lblNewLabel.setBounds(65, 122, 226,
45);
                contentPane.add(lblNewLabel);

                JLabel lblSelectFloor = new
JLabel("Select Floor");
                lblSelectFloor.setFont(new
Font("Serif", Font.ITALIC, 16));
                lblSelectFloor.setBounds(174, 199,
92, 25);
                contentPane.add(lblSelectFloor);

                JComboBox comboBox = new
JComboBox(Floornum);
                comboBox.setBounds(298, 201, 168,
25);
                contentPane.add(comboBox);

                JButton btnAddNewRoom = new
JButton("Add New Room");

        btnAddNewRoom.addActionListener(new
ActionListener() {
                        public void
actionPerformed(ActionEvent e) {
                                String fnumber =
comboBox.getSelectedItem().toString();
                                int a =
JOptionPane.showConfirmDialog(contentPane, "Are You
sure to add new Room to Floor " + fnumber, "Confirm Add
Floor", JOptionPane.YES_NO_OPTION);

        if(a==JOptionPane.YES_OPTION){
                                        try {

         service.AddRoom(fnumber);
                                         ClientView c =
new ClientView();

         c.setVisible(true);

setVisible(false);


        }catch(Exception ec) {
```

```
        ec.printStackTrace();
                                        }
                                }
                        }
                });

        btnAddNewRoom.setForeground(Color.GREEN
);
                btnAddNewRoom.setFont(new
Font("Ubuntu Mono", Font.BOLD | Font.ITALIC, 20));

        btnAddNewRoom.setBackground(Color.RED);
                btnAddNewRoom.setBounds(233,
263, 204, 45);
                contentPane.add(btnAddNewRoom);
        }
}

public class ChangeState extends JFrame {

        /**
         * Launch the application.
         */
        public static void main(String[] args) {
                EventQueue.invokeLater(new
Runnable() {
                        public void run() {
                                try {

        ChangeState frame = new ChangeState();

        frame.setVisible(true);
                                } catch
(Exception e) {

        e.printStackTrace();
                                }
                        }
                });
        }

        /**
         * Create the frame.
         */
        static      boolean State;
        Service service = null;
        String ButtonLable;

        public ChangeState() {


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

           int floorCount = 0;
                try {
                                service =
(Service) Naming.lookup("//localhost/LevelService");
                                floorCount
=service.getFloorCount();

                }catch(Exception e) {

        e.printStackTrace();
                }
                 String Floornum[] = new
String[floorCount];
```

```
                        for(int f=0; f<floorCount ; f++) {
                              Floornum[f] =
String.valueOf(f+1);
                        }

                  getContentPane().setFont(new
Font("Times New Roman", Font.BOLD | Font.ITALIC, 21));

            setDefaultCloseOperation(JFrame.EXIT_ON_C
LOSE);
                  setBounds(100, 100, 604, 452);
                  getContentPane().setLayout(null);

                  JLabel lblUpdateRoom = new
JLabel("Update Room Sensor");
                  lblUpdateRoom.setFont(new
Font("Times New Roman", Font.BOLD | Font.ITALIC, 24));
                  lblUpdateRoom.setBounds(200, 10,
249, 41);

            getContentPane().add(lblUpdateRoom);

                  JLabel lblNewLabel = new
JLabel("Floor No.");
                  lblNewLabel.setFont(new
Font("Tahoma", Font.ITALIC, 18));
                  lblNewLabel.setBounds(108, 81, 108,
25);
                  getContentPane().add(lblNewLabel);

                  JLabel lblRoomNo = new
JLabel("Room No.");
                  lblRoomNo.setFont(new
Font("Tahoma", Font.ITALIC, 18));
                  lblRoomNo.setBounds(108, 138, 108,
25);
                  getContentPane().add(lblRoomNo);


                  JComboBox FloorcomboBox = new
JComboBox(Floornum);
                  FloorcomboBox.setBounds(273, 86,
132, 21);

            getContentPane().add(FloorcomboBox);

                  JComboBox RoomcomboBox = new
JComboBox();
                  RoomcomboBox.setBounds(273, 143,
132, 21);

            getContentPane().add(RoomcomboBox);

                  JButton SubmitButton = new
JButton("Select Floor & Room");
                  SubmitButton.setBounds(182, 232,
192, 32);
                  getContentPane().add(SubmitButton);
                  SubmitButton.setEnabled(false); //
Submit button disable till select room and floor

                  FloorcomboBox.addActionListener
(new ActionListener () {
                    public void
actionPerformed(ActionEvent e) {
                           String[] Roomnum;
                        try {
                          // Select relevant rooms for
selected Floor
```

```
                                    int
RoomCount =
service.getRoomCount(FloorcomboBox.getSelectedItem().
toString());

        Roomnum = new String[RoomCount];
                                    for(int
f=0; f<RoomCount ; f++) {

        Roomnum[f] = String.valueOf(f+1);
                                     }

        RoomcomboBox.setModel(new
DefaultComboBoxModel(Roomnum));



          }catch(Exception ec) {

        ec.printStackTrace();
                                    }


                      }
                    });


        RoomcomboBox.addActionListener(new
ActionListener () {

                        @Override
                        public void
actionPerformed(ActionEvent e) {
                                // TODO Auto-
generated method stub
                                 try {

//Check state of sensor

        String alldata = service.Getdata();

        JSONArray jsar = new JSONArray(alldata);

        SubmitButton.setEnabled(true);

        State =
jsar.getJSONObject(FloorcomboBox.getSelectedIndex()).
getJSONArray("Rooms").getJSONObject(RoomcomboBox
.getSelectedIndex()).getBoolean("Active");

        System.out.println(State + "-------------" +
FloorcomboBox.getSelectedIndex()+1);

        if(State) {

                ButtonLable = "Switch Off the
Sensor";

                SubmitButton.setText("Switch Off the
Sensor");


        SubmitButton.setBackground(Color.RED);

        }

        else {

                ButtonLable = "Switch On the
Sensor";
```

```
                        SubmitButton.setText("Switch On the
Sensor");

            SubmitButton.setBackground(Color.GREEN);

            }


                }catch(Exception ec) {

                        ec.printStackTrace();

            }
                                    }

                        });

                        SubmitButton.addActionListener(new
ActionListener() {
                                    public void
actionPerformed(ActionEvent e) {
                                                int a =
JOptionPane.showConfirmDialog(getContentPane(), "Are
You sure to  " + ButtonLable, "Confirm Sensor Change",
JOptionPane.YES_NO_OPTION);

            if(a==JOptionPane.YES_OPTION){
                                                            try {
            if(State)

            service.SensorOff(FloorcomboBox.getSelectedIt
em().toString(),
RoomcomboBox.getSelectedItem().toString());

            else


            service.SensorOn(FloorcomboBox.getSelectedIt
em().toString(),
RoomcomboBox.getSelectedItem().toString());


            ClientView c = new ClientView();

               c.setVisible(true);

               setVisible(false);

            }catch(Exception ec) {

            ec.printStackTrace();
                                                }
                                    }
                        }
            });


            }
}
```