

DS Assignment 2



IT18176070 - Perera P.R.H

IT18186284 - E.S.S. Soysa

IT18124736 – H R S De Silva

IT18186352 – A.R.P Tavarayan

May 6, 2020

Table of Contents

[1 Introduction 2](#_Toc602145)

[1.1 Overview 2](#_Toc602146)

[1.2 Technology Stack 2](#_Toc602147)

[2 ARCHITECTURE DIAGRAM 3](#_Toc602148)

[2.1 High level architectural diagram 3](#_Toc602149)

[2.3 Service interfaces 3](#_Toc602150)

[3 APENDIX 4](#_Toc602152)

[3.1 Source code of the restful web service 4](#_Toc602153)

[3.2 Source code of web client 4](#_Toc602154)

[3.3 Source code of rmi client and server 4](#_Toc602155)

[3.4 Source code of the sensor application 5](#_Toc602157)

[3.5 Source code of the email service 5](#_Toc602158)

[3.6 Source code of the sms service 5](#_Toc602158)

[4 Conclusion 8](#_Toc602168)

# Introduction

## Overview

Report is prepared for an implemented Fire alarm monitoring system. The main components the system are as follows; restful web service which connects to a cloud database (Mongo DB Atlas), RMI client and RMI server, an email service, sms service and a fire sensor application. These components would communicate through REST technology of Service Oriented Architecture (SOA). JSON is used as the message format for communication between components of the system.

The functions of the system are as follows. Web client is capable of viewing sensor information. This is done by sending REST calls to the restful web service. RMI Client desktop application could also view sensor information. It is capable of adding sensors and editing sensor information after login to the system as the Administrator. Fire sensors would periodically send updates of the smoke and Carbon Dioxide level to the restful web service. The restful web service is capable of adding, updating and retrieving stored sensor information from the database. When the level of carbon dioxide or the level of smoke rises above the level 5, the RMI server would send alerts to email service and sms service.

## Technology stack

Following are the technologies used in developing the fire alarm monitoring system.

|  |  |
| --- | --- |
| **Component** | **Technology** |
| Restful web service | Spring boot |
| Web Client | React Js |
| RMI Client and RMI Server | Java |
| Fire Sensor | Spring boot |
| Database | Mongo DB (Atlas) |

# System overview

## System Architecture Diagram

Database

WEB Client

REST Service

RMI Server

RMI Client

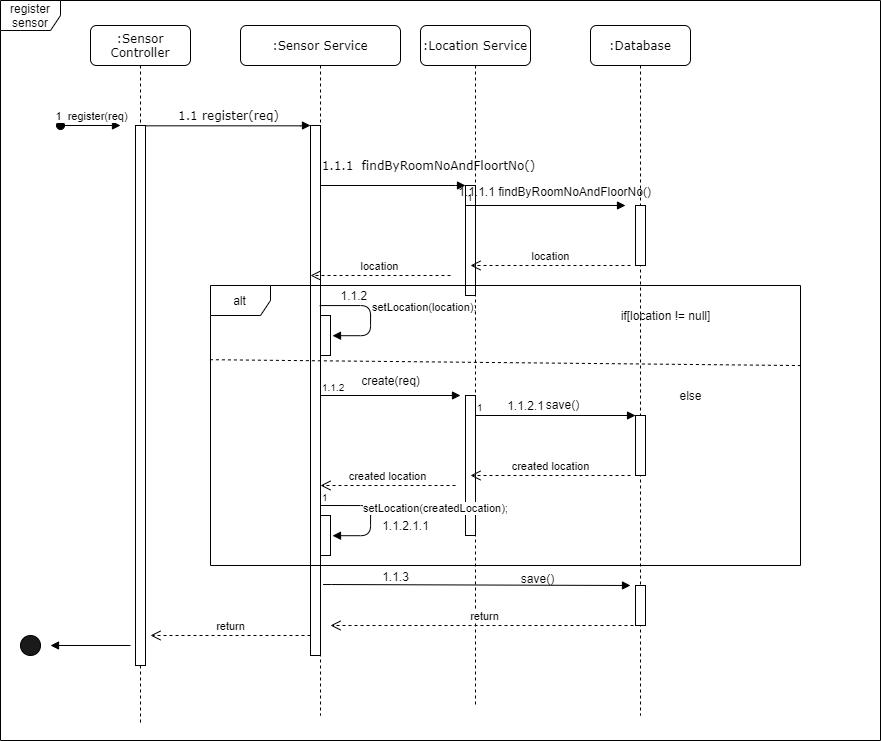
Sensor Application

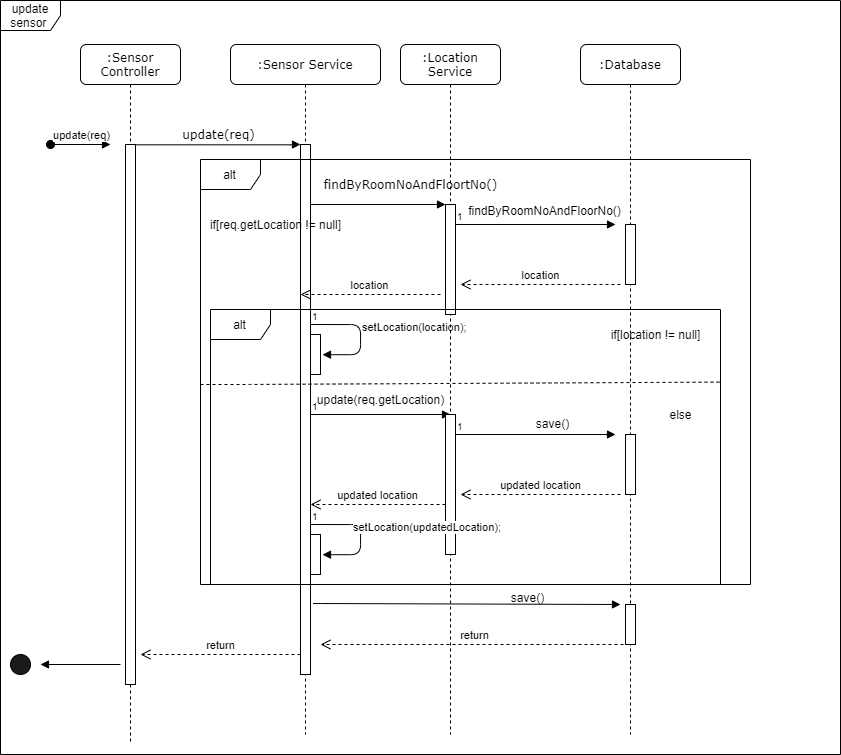
SMS Service

Email Service

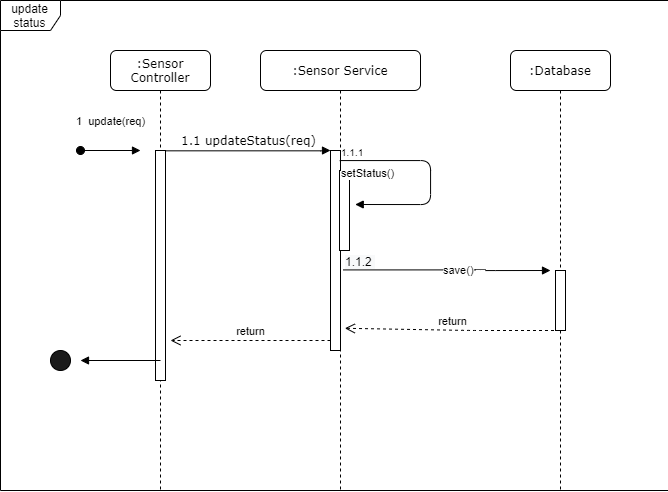
## Service Interfaces

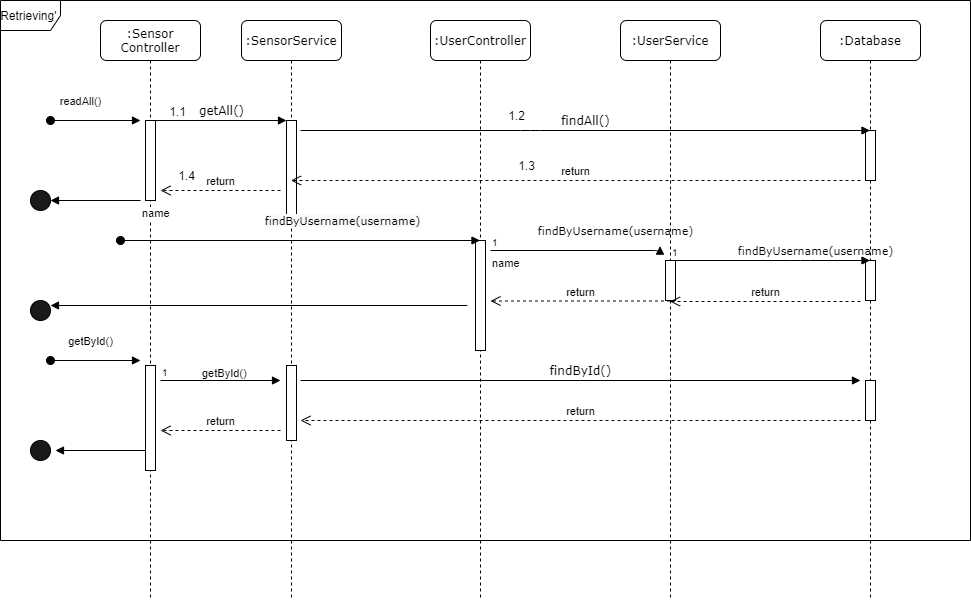
**2.2.1 RESTful Web Service: Register Sensor**



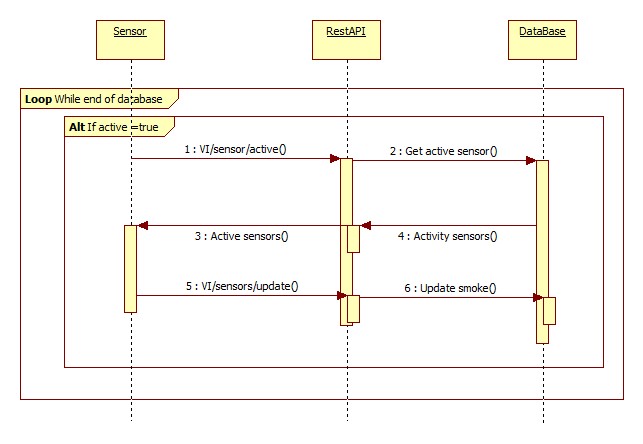
**2.2.2 RESTful Web Service: Update Sensor Information**

**2.2.3 RESTful Web Service: Update Sensor Status**

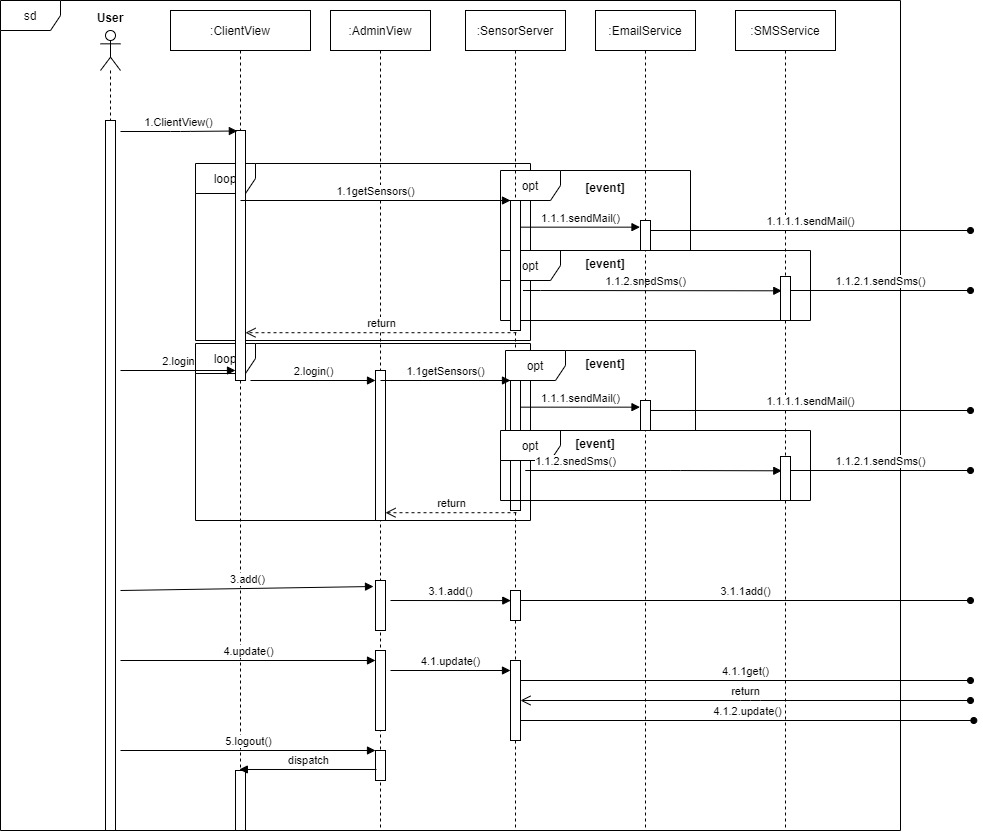


**2.2.4 RESTful Web Service: Retrieve Sensor Information**

**2.2.4 RESTful Web Service: Fire sensor Application**

****

**2.2.5 Desktop Client Application**

****

# APENDIX

## Source code of RESTful web service: (rest-service)

**Location.java**

**package** com.ds.restservice.models;

**import** org.springframework.data.annotation.Id;

**import** org.springframework.data.annotation.TypeAlias;

**import** org.springframework.data.mongodb.core.mapping.Document;

**import** org.springframework.data.mongodb.core.mapping.Field;

**import** lombok.AllArgsConstructor;

**import** lombok.Builder;

**import** lombok.Getter;

**import** lombok.NoArgsConstructor;

**import** lombok.Setter;

@NoArgsConstructor

@AllArgsConstructor

@Builder

@Getter

@Setter

@TypeAlias(value = "Location")

@Document

**public** **class** Location {

@Id

**private** String id;

@Field("room")

**private** **int** roomNo;

@Field("floor")

**private** **int** floorNo;

}

**Sensor.java**

**package** com.ds.restservice.models;

**import** org.springframework.data.annotation.Id;

**import** org.springframework.data.annotation.TypeAlias;

**import** org.springframework.data.mongodb.core.mapping.DBRef;

**import** org.springframework.data.mongodb.core.mapping.Document;

**import** org.springframework.data.mongodb.core.mapping.Field;

**import** lombok.AllArgsConstructor;

**import** lombok.Getter;

**import** lombok.NoArgsConstructor;

**import** lombok.Setter;

@NoArgsConstructor

@AllArgsConstructor

@Getter

@Setter

@TypeAlias(value = "Sensor")

@Document

**public** **class** Sensor {

@Id

**private** String id;

@Field("smoke\_level")

**private** **int** smokeLevel;

@Field("co2\_level")

**private** **int** co2Level;

@DBRef

**private** Location location;

**private** **boolean** active;

}

**User.java**

**package** com.ds.restservice.models;

**import** org.springframework.data.annotation.Id;

**import** org.springframework.data.annotation.TypeAlias;

**import** org.springframework.data.mongodb.core.mapping.Document;

**import** org.springframework.data.mongodb.core.mapping.Field;

**import** lombok.AllArgsConstructor;

**import** lombok.Getter;

**import** lombok.NoArgsConstructor;

**import** lombok.Setter;

@NoArgsConstructor

@AllArgsConstructor

@Getter

@Setter

@Document

@TypeAlias(value = "User")

**public** **class** User {

@Id

**private** String id;

**private** String username;

**private** String password;

@Field("sender\_email")

**private** String senderEmail;

@Field("receiver\_email")

**private** String recieverEmail;

@Field("sender\_password")

**private** String senderPassword;

@Field("sender\_mobile")

**private** String senderMobile;

@Field("receiver\_mobile")

**private** String recieverMobile;

}

**User Repository.java**

package com.ds.restservice.respositories;

import java.util.Optional;

import org.springframework.data.mongodb.repository.MongoRepository;

import org.springframework.stereotype.Repository;

import com.ds.restservice.models.User;

@Repository

public interface UserRepository extends MongoRepository<User,String>{

Optional<User> findById(String id);

Optional<User> findByUsername(String username);

}

Location Repositoy.java

package com.ds.restservice.respositories;

import java.util.List;

import java.util.Optional;

import org.springframework.data.mongodb.repository.MongoRepository;

import org.springframework.stereotype.Repository;

import com.ds.restservice.models.Location;

@Repository

public interface LocationRepository extends MongoRepository<Location, String>{

Optional<Location> findByRoomNoAndFloorNo(int roomNo, int floorNo);

Optional<Location> findById(String id);

List<Location> findAll();

}

Sensor Repository.java

package com.ds.restservice.respositories;

import java.util.List;

import java.util.Optional;

import org.springframework.data.mongodb.repository.MongoRepository;

import org.springframework.stereotype.Repository;

import com.ds.restservice.models.Location;

import com.ds.restservice.models.Sensor;

@Repository

public interface SensorRepository extends MongoRepository<Sensor, String>{

Optional<Sensor> findById(String id);

List<Sensor> findAll();

Sensor findByLocation(Location location);

List<Sensor> findByActive(boolean active);

}

**Sensor Service.java**

package com.ds.restservice.service;

import java.util.ArrayList;

import java.util.List;

import java.util.Optional;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.stereotype.Service;

import com.ds.restservice.dto.LocationRequestDto;

import com.ds.restservice.dto.LocationResponseDto;

import com.ds.restservice.dto.SensorRequestDto;

import com.ds.restservice.dto.SensorResponseDto;

import com.ds.restservice.models.Location;

import com.ds.restservice.models.Sensor;

import com.ds.restservice.respositories.SensorRepository;

@Service

public class SensorService {

@Autowired

private SensorRepository sensorRepository;

@Autowired

private LocationService locationService;

/\*\*

\* register new sensor

\* @param req

\* @return

\*/

public SensorResponseDto register(LocationRequestDto req) {

int floorNo = req.getFloorNo();

int roomNo = req.getRoomNo();

Location location;

//check if the location provided already exists in the database

Optional<Location> locationOpt = this.locationService.findByRoomNoAndFloor(roomNo, floorNo);

if(locationOpt.isPresent()) {

location = locationOpt.get();

}else {

//create new location object with room no and floor no

LocationResponseDto createdLocation = this.locationService.create(req);

location = getLocationObject(createdLocation);

}

Sensor sensor = new Sensor();

sensor.setActive(true);

sensor.setLocation(location);

Sensor savedSensor = this.sensorRepository.save(sensor);

SensorResponseDto reponseDto = buildSensorResponseDto(savedSensor);

LocationResponseDto locationResponseDto = buildLocationReponseDto(savedSensor.getLocation());

reponseDto.setLocation(locationResponseDto);

return reponseDto;

}

/\*\*

\* Update sensor information and related location information (room no and floor no)

\* @param req

\* @param id

\* @param locationId

\* @return

\* @throws Exception

\*/

public SensorResponseDto update(SensorRequestDto req, String id, String locationId) throws Exception {

Optional<Sensor> sensorOpt = this.sensorRepository.findById(id);

if(!sensorOpt.isPresent()) {

throw new Exception();

}

Sensor sensor = sensorOpt.get();

Location location = null;

if(req.getLocation() != null) {

LocationRequestDto locationRequest = req.getLocation();

//check if the location provided already exists in the database

Optional<Location> locationCheck =

this.locationService.findByRoomNoAndFloor(locationRequest.getRoomNo(), locationRequest.getFloorNo());

if(locationCheck.isPresent()) {

location = locationCheck.get();

}else {

//update existing location object with the new information provided

LocationResponseDto updatedLocation = this.locationService.update(locationRequest, locationId);

location = getLocationObject(updatedLocation);

}

}

//sensor is set active by default

sensor.setActive(req.getActive());

sensor.setLocation(location != null ? location : sensor.getLocation());

Sensor savedSensor = this.sensorRepository.save(sensor);

SensorResponseDto reponseDto = buildSensorResponseDto(savedSensor);

LocationResponseDto locationResponseDto = buildLocationReponseDto(savedSensor.getLocation());

reponseDto.setLocation(locationResponseDto);

return reponseDto;

}

/\*\*

\* Update sensor status by sensor id, i.e Carbon dioxide and smoke levels of a particular sensor

\* @param req

\* @param id

\* @return

\* @throws Exception

\*/

public SensorResponseDto updateStatus(SensorRequestDto req, String id) throws Exception {

Optional<Sensor> sensorOpt = this.sensorRepository.findById(id);

if(!sensorOpt.isPresent()) {

throw new Exception();

}

Sensor sensor = sensorOpt.get();

sensor.setCo2Level(req.getCo2Level());

sensor.setSmokeLevel(req.getSmokeLevel());

Sensor savedSensor = this.sensorRepository.save(sensor);

SensorResponseDto reponseDto = buildSensorResponseDto(savedSensor);

LocationResponseDto locationResponseDto = buildLocationReponseDto(savedSensor.getLocation());

reponseDto.setLocation(locationResponseDto);

return reponseDto;

}

/\*\*

\* Retreive all sensor information from the database

\* @return

\*/

public List<SensorResponseDto> getAll() {

List<Sensor> sensors = this.sensorRepository.findAll();

List<SensorResponseDto> responseList = new ArrayList<>();

for (Sensor sensor : sensors) {

SensorResponseDto sensorResponseDto = buildSensorResponseDto(sensor);

LocationResponseDto locationReponseDto = buildLocationReponseDto(sensor.getLocation());

sensorResponseDto.setLocation(locationReponseDto);

responseList.add(sensorResponseDto);

}

return responseList;

}

/\*\*

\* Retreive all active sensor information from the database

\* @return

\*/

public List<SensorResponseDto> getAllByActive() {

List<Sensor> sensors = this.sensorRepository.findByActive(true);

List<SensorResponseDto> responseList = new ArrayList<>();

for (Sensor sensor : sensors) {

SensorResponseDto sensorResponseDto = buildSensorResponseDto(sensor);

LocationResponseDto locationReponseDto = buildLocationReponseDto(sensor.getLocation());

sensorResponseDto.setLocation(locationReponseDto);

responseList.add(sensorResponseDto);

}

return responseList;

}

/\*\*

\* Get sensor information by sensor id

\* @param id

\* @return

\* @throws Exception

\*/

public SensorResponseDto getById(String id) throws Exception {

Optional<Sensor> optional = this.sensorRepository.findById(id);

if(!optional.isPresent()) {

throw new Exception();

}

SensorResponseDto sensorResponseDto = buildSensorResponseDto(optional.get());

LocationResponseDto locationReponseDto = buildLocationReponseDto(optional.get().getLocation());

sensorResponseDto.setLocation(locationReponseDto);

return sensorResponseDto;

}

// get a Location object from LocationResponseDto

private Location getLocationObject(LocationResponseDto createdLocation) {

Location location = new Location();

location.setId(createdLocation.getId());

location.setFloorNo(createdLocation.getFloorNo());

location.setRoomNo(createdLocation.getRoomNo());

return location;

}

// builds a Sensor response DTO

private SensorResponseDto buildSensorResponseDto(Sensor savedSensor) {

return SensorResponseDto.builder()

.id(savedSensor.getId())

.active(savedSensor.isActive())

.co2Level(savedSensor.getCo2Level())

.smokeLevel(savedSensor.getSmokeLevel())

.build();

}

// builds a Location response DTO

private LocationResponseDto buildLocationReponseDto(Location location) {

return LocationResponseDto.builder()

.id(location.getId())

.floorNo(location.getFloorNo())

.roomNo(location.getRoomNo())

.build();

}

}

**Location Service.java**

package com.ds.restservice.service;

import java.util.ArrayList;

import java.util.List;

import java.util.Optional;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.stereotype.Service;

import com.ds.restservice.dto.LocationRequestDto;

import com.ds.restservice.dto.LocationResponseDto;

import com.ds.restservice.models.Location;

import com.ds.restservice.respositories.LocationRepository;

@Service

public class LocationService {

@Autowired

private LocationRepository locationRepository;

/\*\*

\* Create new location providing room no and floor no

\* @param req

\* @return

\*/

public LocationResponseDto create(LocationRequestDto req) {

Location location = new Location();

location.setFloorNo(req.getFloorNo());

location.setRoomNo(req.getRoomNo());

Location savedLocation = this.locationRepository.save(location);

LocationResponseDto reponseDto = this.buildLocationReponseDto(savedLocation);

return reponseDto;

}

/\*\*

\* Update location information by location id

\* @param req

\* @param id

\* @return

\* @throws Exception

\*/

public LocationResponseDto update(LocationRequestDto req, String id) throws Exception {

Optional<Location> locationOpt = this.locationRepository.findById(id);

if(!locationOpt.isPresent()) {

throw new Exception();

}

Location location = locationOpt.get();

location.setFloorNo(req.getFloorNo());

location.setRoomNo(req.getRoomNo());

Location savedLocation = this.locationRepository.save(location);

LocationResponseDto reponseDto = this.buildLocationReponseDto(savedLocation);

return reponseDto;

}

/\*\*

\* Get location object by Id

\* @param id

\* @return

\* @throws Exception

\*/

public LocationResponseDto getById(String id) throws Exception {

Optional<Location> locationOpt = this.locationRepository.findById(id);

LocationResponseDto locationResponseDto = new LocationResponseDto();

if(locationOpt.isPresent()) {

throw new Exception();

}

Location location = locationOpt.get();

locationResponseDto.setId(location.getId());

locationResponseDto.setFloorNo(location.getFloorNo());

locationResponseDto.setRoomNo(location.getRoomNo());

return locationResponseDto;

}

/\*\*

\* Find location details by roomNo and floor no

\* @param roomNo

\* @param floorNo

\* @return

\*/

public Optional<Location> findByRoomNoAndFloor(int roomNo, int floorNo) {

Optional<Location> location = this.locationRepository.findByRoomNoAndFloorNo(roomNo, floorNo);

return location;

}

/\*\*

\* Retreive all location information

\* @return

\*/

public List<LocationResponseDto> getAll(){

List<LocationResponseDto> reponseList = new ArrayList<>();

List<Location> locations = this.locationRepository.findAll();

for (Location location : locations) {

LocationResponseDto reponseDto = buildLocationReponseDto(location);

reponseList.add(reponseDto);

}

return reponseList;

}

//build LocationResponseDto from Location entity object

private LocationResponseDto buildLocationReponseDto(Location location) {

return LocationResponseDto.builder()

.id(location.getId())

.floorNo(location.getFloorNo())

.roomNo(location.getRoomNo())

.build();

}

}

**User Service.java**

package com.ds.restservice.service;

import java.util.Optional;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.stereotype.Service;

import com.ds.restservice.dto.UserResponseDto;

import com.ds.restservice.models.User;

import com.ds.restservice.respositories.UserRepository;

@Service

public class UserService {

@Autowired

private UserRepository userRepository;

/\*\*\*

\* Retrieve user information by username

\* @param username

\* @return

\* @throws Exception

\*/

public UserResponseDto findByUsername(String username) throws Exception {

Optional<User> user = this.userRepository.findByUsername(username);

if(!user.isPresent()) {

throw new Exception();

}

UserResponseDto response = buildUserResponseDto(user.get());

return response;

}

//build UserResponseDto from User Entity Object

private UserResponseDto buildUserResponseDto(User user) {

return UserResponseDto.builder()

.id(user.getId())

.username(user.getUsername())

.password(user.getPassword())

.senderEmail(user.getSenderEmail())

.senderPassword(user.getSenderPassword())

.receiverEmail(user.getRecieverEmail())

.senderMobile(user.getSenderMobile())

.recieverMobile(user.getRecieverMobile())

.build();

}

}

**User Controller**

package com.ds.restservice.controller;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.http.HttpStatus;

import org.springframework.web.bind.annotation.GetMapping;

import org.springframework.web.bind.annotation.PathVariable;

import org.springframework.web.bind.annotation.RequestMapping;

import org.springframework.web.bind.annotation.RestController;

import com.ds.restservice.dto.UserResponseDto;

import com.ds.restservice.service.UserService;

import io.swagger.annotations.ApiOperation;

import io.swagger.annotations.ApiParam;

@RestController

@RequestMapping("v1/users")

public class UserController {

@Autowired

private UserService userService;

/\*\*\*

\* Get user details by username

\* @param username

\* @return

\*/

@ApiOperation(value = "Get user details from username")

@GetMapping("/{username}")

public UserResponseDto findByUsername(@ApiParam(name = "username")

@PathVariable String username) {

UserResponseDto response = null;

try {

response = this.userService.findByUsername(username);

} catch (Exception e) {

response = new UserResponseDto();

response.setErrorCode(HttpStatus.NOT\_FOUND.ordinal());

}

return response;

}

}

**Sensor Controller**

package com.ds.restservice.controller;

import java.util.List;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.http.HttpStatus;

import org.springframework.web.bind.annotation.GetMapping;

import org.springframework.web.bind.annotation.PathVariable;

import org.springframework.web.bind.annotation.PostMapping;

import org.springframework.web.bind.annotation.PutMapping;

import org.springframework.web.bind.annotation.RequestBody;

import org.springframework.web.bind.annotation.RequestMapping;

import org.springframework.web.bind.annotation.RestController;

import com.ds.restservice.dto.LocationRequestDto;

import com.ds.restservice.dto.SensorRequestDto;

import com.ds.restservice.dto.SensorResponseDto;

import com.ds.restservice.service.SensorService;

import io.swagger.annotations.ApiOperation;

import io.swagger.annotations.ApiParam;

@RestController

@RequestMapping("v1/sensors")

public class SensorController {

@Autowired

private SensorService sensorService;

/\*\*

\* Register a new sensor

\* @param request

\* @return

\*/

@ApiOperation(value = "create new sensor")

@PostMapping

public SensorResponseDto register(@RequestBody LocationRequestDto request) {

SensorResponseDto responseDto = this.sensorService.register(request);

return responseDto;

}

/\*\*\*

\* update sensor information

\* @param location\_id

\* @param id

\* @param request

\* @return

\*/

@ApiOperation(value = "update sensor information")

@PutMapping("/{id}/locations/{location\_id}")

public SensorResponseDto update(

@ApiParam(name = "location\_id") @PathVariable String location\_id,

@ApiParam(name = "id") @PathVariable String id,

@RequestBody SensorRequestDto request) {

SensorResponseDto responseDto = null;

try {

responseDto = this.sensorService.update(request, id, location\_id);

} catch (Exception e) {

responseDto = new SensorResponseDto();

responseDto.setErrorCode(HttpStatus.INTERNAL\_SERVER\_ERROR.ordinal());

}

return responseDto;

}

/\*\*

\* Update sensor status

\* @param id

\* @param request

\* @return

\*/

@ApiOperation(value = "update sensor status")

@PutMapping("/{id}")

public SensorResponseDto updateStatus(

@ApiParam(name = "id") @PathVariable String id,

@RequestBody SensorRequestDto request) {

SensorResponseDto responseDto = null;

try {

responseDto = this.sensorService.updateStatus(request, id);

} catch (Exception e) {

responseDto = new SensorResponseDto();

responseDto.setErrorCode(HttpStatus.INTERNAL\_SERVER\_ERROR.ordinal());

}

return responseDto;

}

/\*\*

\* Get sensor details by id

\* @param id

\* @return

\*/

@ApiOperation(value = "get sensor details by id")

@GetMapping("/{id}")

public SensorResponseDto getById(

@ApiParam(name = "id") @PathVariable String id) {

SensorResponseDto responseDto = null;

try {

responseDto = this.sensorService.getById(id);

} catch (Exception e) {

responseDto = new SensorResponseDto();

responseDto.setErrorCode(HttpStatus.NOT\_FOUND.ordinal());

}

return responseDto;

}

/\*\*

\* Get all sensor information

\* @return

\*/

@ApiOperation(value = "get all sensor information")

@GetMapping

public List<SensorResponseDto> readAll() {

List<SensorResponseDto> responseList = null;

try {

responseList = this.sensorService.getAll();

return responseList;

} catch (Exception e) {

return null;

}

}

/\*\*

\* Get all active sensor information

\* @return

\*/

@ApiOperation(value = "get all active sensor information")

@GetMapping("/active")

public List<SensorResponseDto> readAllByActive() {

List<SensorResponseDto> responseList = null;

try {

responseList = this.sensorService.getAllByActive();

return responseList;

} catch (Exception e) {

return null;

}

}

}

## Source code of Web Client

**Details.js**

import axios from 'axios';

import React, { Component } from 'react';

class Details extends Component {

constructor(props) {

super(props);

this.state = {

values: [],

}

}

componentDidMount() {

this.getDetails();

setInterval(()=> {

this.getDetails();

},40000)

}

getDetails(){

axios.get('http://localhost:8080/v1/sensors')

.then(response => {

this.setState({values: response.data}, console.log(this.state.values.active))

console.log(response)

// console.log("Hyyyyyyyyyyyyyyyyyyyyy")

})

.catch(error => {

console.log(error)

})

}

stateactive = (val) => {

if(val)

return 'ACTIVE';

else

return 'DEACTIVE'

}

render() {

const { values }=this.state

return (

<div>

<table className="table table-bordered">

<thead>

<tr>

<th colSpan="5">SENSOR DETAILS</th>

</tr>

</thead>

{values.map((post)=> <tbody key={post.id}>

<tr>

<td>Floor no {post.location.floor\_no}</td>

<td>Room no {post.location.room\_no}</td>

<td>Sensor status : {this.stateactive(post.active)}</td>

<td bgcolor={post.smoke\_level>5 ?"#FF0000":"#00FF00"}>smoke Level {post.smoke\_level}</td>

<td bgcolor={post.co2\_level>5 ?"#FF0000":"#00FF00"}>Co2 Level {post.co2\_level}</td>

</tr>

</tbody>)

}

</table>

</div>

)

}

}

export default Details

**App.js**

import React from 'react';

import './App.css';

import Details from "./components/Details";

//calls the Details component

function App() {

return (

<div>

<Details/>

</div>

);

}

export default App;

## Source code of RMI Server and Client

import java.awt.\*;

import java.awt.event.ActionEvent;

import java.awt.event.ActionListener;

import java.rmi.Naming;

import java.rmi.Remote;

import java.util.List;

import javax.swing.\*;

import java.util.Timer;

import java.util.TimerTask;

public class ClientFrame extends JFrame implements ActionListener {

/\*\*

\*

\*/

private static final long serialVersionUID = 1L;

//Initialise JFrame components

Container container = getContentPane();

JButton loginButton = new JButton("LOGOUT");

JButton add = new JButton("ADD");

JLabel floor = new JLabel("Floor");

JLabel room = new JLabel("Room");

JLabel co2 = new JLabel("CO2");

JLabel smoke = new JLabel("Smoke");

JLabel status = new JLabel("Status");

JLabel rmno = new JLabel("Room No:");

JLabel flno = new JLabel("Floor No:");

JLabel adminPanel = new JLabel("Admin Panel");

JTextField roomno = new JTextField(5);

JTextField floorno = new JTextField(5);

static ClientFrame frame = null;

List<Sensor> reading;

MyButton m = new MyButton();

public ClientFrame() {

contents();

}

//Set the JFrame components together

public void contents() {

setLayoutManager();

setLocationAndSize();

addComponentsToContainer();

addActionEvent();

adminPanel.setFont(new Font("", Font.PLAIN, 20));

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

try {

//get sensor information

String registration = "//localhost/SensorService";

Remote remoteService = Naming.lookup(registration);

SensorService sensor = (SensorService) remoteService;

//empty the JFrame first

container.removeAll();

addComponentsToContainer();

reading = sensor.getSensors();

//create rows for each sensor

int j = 130;

for (Sensor i : reading) {

JLabel floor = new JLabel();

JLabel room = new JLabel();

JLabel co2 = new JLabel();

JLabel smoke = new JLabel();

JLabel status = new JLabel();

MyButton edit = new MyButton();

edit.setText("EDIT");

floor.setOpaque(true);

room.setOpaque(true);

co2.setOpaque(true);

smoke.setOpaque(true);

status.setOpaque(true);

floor.setText(String.valueOf(i.getLocation().getFloor\_no()));

room.setText(String.valueOf(i.getLocation().getRoom\_no()));

co2.setText(String.valueOf(i.getCo2\_level()));

smoke.setText(String.valueOf(i.getSmoke\_level()));

status.setText(String.valueOf(i.isActive()));

edit.setSensorId(i.getId());

//update button click event

edit.addActionListener(new ActionListener() {

public void actionPerformed(ActionEvent e) {

// TODO Auto-generated method stub

Object source = e.getSource();

MyButton m = (MyButton)source;

EditSensor es = new EditSensor(m.getSensorId());

es.setTitle("Edit Form");

es.setVisible(true);

es.setBounds(500, 100, 390, 180);

es.setDefaultCloseOperation(JFrame.DISPOSE\_ON\_CLOSE );

es.setResizable(true);

dispose();

}

});

//check for fire alarm event

if(i.getCo2\_level() >= 5 || i.getSmoke\_level() >= 5) {

floor.setBackground(Color.RED);

room.setBackground(Color.RED);

co2.setBackground(Color.RED);

smoke.setBackground(Color.RED);

status.setBackground(Color.RED);

//send smms and email

//sensor.sendMail();

//sensor.sendSMS();

} else if(i.isActive() == true) {

floor.setBackground(Color.GREEN);

room.setBackground(Color.GREEN);

co2.setBackground(Color.GREEN);

smoke.setBackground(Color.GREEN);

status.setBackground(Color.GREEN);

}

//set JFrame components

floor.setBounds(10, j, 50, 30);

room.setBounds(60, j, 50, 30);

co2.setBounds(110, j, 50, 30);

smoke.setBounds(160, j, 50, 30);

status.setBounds(210, j, 50, 30);

edit.setBounds(270, j, 100, 30);

container.add(floor);

container.add(room);

container.add(edit);

container.add(co2);

container.add(smoke);

container.add(status);

j+=40;

}

} catch (Exception e) {

e.printStackTrace();

}

}

public static void main(String[] args) {

// TODO Auto-generated method stub

//create new JFrame

frame = new ClientFrame();

frame.setTitle("Admin Panel");

frame.setVisible(true);

frame.setBounds(500, 100, 400, 600);

frame.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

frame.setResizable(true);

//Refresh JFrame every 15 seconds

new Timer().scheduleAtFixedRate(new TimerTask() {

@Override

public void run() {

if(frame != null) {

frame.contents();

frame.revalidate();

frame.repaint();

}

}

}, 0, 15000);

}

//set JFrame layout

public void setLayoutManager() {

container.setLayout(null);

}

//set JFrame location and size

public void setLocationAndSize() {

loginButton.setBounds(270, 10, 100, 30);

add.setBounds(270, 50, 100, 30);

floor.setBounds(10, 90, 520, 30);

room.setBounds(60, 90, 50, 30);

co2.setBounds(110, 90, 50, 30);

smoke.setBounds(160, 90, 50, 30);

status.setBounds(210, 90, 50, 30);

flno.setBounds(10, 50, 520, 30);

floorno.setBounds(65, 50, 60, 30);

rmno.setBounds(130, 45, 520, 40);

roomno.setBounds(190, 50, 70, 30);

adminPanel.setBounds(90, 5, 120, 40);

}

//add components to JFrame

public void addComponentsToContainer() {

container.add(loginButton);

container.add(floor);

container.add(room);

container.add(add);

container.add(co2);

container.add(smoke);

container.add(status);

container.add(rmno);

container.add(flno);

container.add(roomno);

container.add(floorno);

container.add(adminPanel);

}

//add button events

public void addActionEvent() {

if (loginButton.getActionListeners().length<1){

loginButton.addActionListener(this);

}

if (add.getActionListeners().length<1){

add.addActionListener(this);

}

}

//button events

public void actionPerformed(ActionEvent e) {

// TODO Auto-generated method stub

//Client view/login

if(e.getSource() == loginButton) {

this.dispose();

LoginFrame frame = new LoginFrame();

frame.setTitle("Login Form");

frame.setVisible(true);

frame.setBounds(500, 100, 290, 600);

frame.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

frame.setResizable(true);

}

//get sensor id and pass to next frame

if(e.getSource() == add) {

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

try {

String registration = "//localhost/SensorService";

Remote remoteService = Naming.lookup(registration);

SensorService sensor = (SensorService) remoteService;

sensor.addSensor(Integer.parseInt(floorno.getText()), Integer.parseInt(roomno.getText()));

floorno.setText("");

roomno.setText("");

contents();

revalidate();

repaint();

}

catch (Exception ex) {

ex.printStackTrace();

}

}

}

}

import java.awt.Container;

import java.awt.Font;

import java.awt.event.ActionEvent;

import java.awt.event.ActionListener;

import java.rmi.Naming;

import java.rmi.Remote;

import javax.swing.JButton;

import javax.swing.JFrame;

import javax.swing.JLabel;

import javax.swing.JTextField;

//edit sensor detail frame

public class EditSensor extends JFrame implements ActionListener {

//declare variables

private String sensorId;

private String locationId;

Container container = getContentPane();

JLabel edit = new JLabel("Edit");

JLabel room = new JLabel("Room No:");

JLabel floor = new JLabel("Floor No:");

JLabel status = new JLabel("Active:");

JTextField roomNo = new JTextField();

JTextField floorNo = new JTextField();

JTextField active = new JTextField();

JButton update = new JButton("UPDATE");

//set JFrame layout

public void setLayoutManager() {

container.setLayout(null);

}

//set JFrame location and Size

public void setLocationAndSize() {

edit.setBounds(160, 10, 100, 30);

floor.setBounds(10, 50, 50, 30);

floorNo.setBounds(70, 50, 50, 30);

room.setBounds(130, 50, 60, 30);

roomNo.setBounds(200, 50, 50, 30);

status.setBounds(260, 50, 40, 30);

active.setBounds(310, 50, 50, 30);

update.setBounds(140, 90, 100, 30);

}

//add JFrame components

public void addComponentsToContainer() {

container.add(roomNo);

container.add(floorNo);

container.add(edit);

container.add(active);

container.add(update);

container.add(floor);

container.add(room);

container.add(status);

}

//add button eventlistener

public void addActionEvent() {

update.addActionListener(this);

}

//button events

public void actionPerformed(ActionEvent e) {

// TODO Auto-generated method stub

//update sensor

if(e.getSource() == update) {

try {

String registration = "//localhost/SensorService";

Remote remoteService = Naming.lookup(registration);

SensorService sensorService = (SensorService) remoteService;

sensorService.updateSensor(Integer.parseInt(floorNo.getText()), Integer.parseInt( roomNo.getText()), Boolean.parseBoolean(active.getText()), sensorId, locationId);

this.dispose();

//go back to admin panel

ClientFrame frame = new ClientFrame();

frame.setTitle("Login Form");

frame.setVisible(true);

frame.setBounds(500, 100, 400, 600);

frame.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

frame.setResizable(true);

} catch (Exception ex) {

ex.printStackTrace();

}

}

}

public EditSensor(String sensorId) {

this.sensorId = sensorId;

setLayoutManager();

setLocationAndSize();

addComponentsToContainer();

addActionEvent();

edit.setFont(new Font("", Font.PLAIN, 20));

//get sensor

try {

String registration = "//localhost/SensorService";

Remote remoteService = Naming.lookup(registration);

SensorService sensorService = (SensorService) remoteService;

Sensor sensor = sensorService.getSensorById(sensorId);

floorNo.setText(String.valueOf(sensor.getLocation().getFloor\_no()));

roomNo.setText(String.valueOf(sensor.getLocation().getRoom\_no()));

active.setText(String.valueOf(sensor.isActive()));

this.locationId = String.valueOf(sensor.getLocation().getId());

} catch (Exception e) {

e.printStackTrace();

}

}

}

//

//import java.rmi.Naming;

//import java.rmi.Remote;

//import java.rmi.RemoteException;

//import java.rmi.server.UnicastRemoteObject;

//

//public class ClientServiceProvider extends UnicastRemoteObject {

//

// protected ClientServiceProvider() throws RemoteException {

// super();

// // TODO Auto-generated constructor stub

// }

//

// public static void main(String[] args) {

// // TODO Auto-generated method stub

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

//

//

// try {

// String registration = "//localhost/SensorService";

//

// Remote remoteService = Naming.lookup(registration);

// SensorService sensor = (SensorService) remoteService;

//// Location reading = sensor.addSensor();

//// System.out.println("Original temp : " + reading);

//

// } catch (Exception e) {

// e.printStackTrace();

// }

// }

//

//}

import java.io.Serializable;

//Location object model

public class Location implements Serializable {

/\*\*

\*

\*/

private static final long serialVersionUID = 1L;

private String id;

public String getId() {

return id;

}

public void setId(String id) {

this.id = id;

}

private int floor\_no;

private int room\_no;

public int getFloor\_no() {

return floor\_no;

}

public void setFloor\_no(int floor\_no) {

this.floor\_no = floor\_no;

}

public int getRoom\_no() {

return room\_no;

}

public void setRoom\_no(int room\_no) {

this.room\_no = room\_no;

}

}

import java.awt.\*;

import java.awt.event.ActionEvent;

import java.awt.event.ActionListener;

import java.rmi.Naming;

import java.rmi.Remote;

import java.util.List;

import java.util.Timer;

import java.util.TimerTask;

import javax.swing.\*;

//Client view/Login

public class LoginFrame extends JFrame implements ActionListener {

//declare Jframe components and variable

Container container = getContentPane();

JButton loginButton = new JButton("LOGIN");

JLabel floor = new JLabel("Floor");

JLabel room = new JLabel("Room");

JLabel co2 = new JLabel("CO2");

JLabel smoke = new JLabel("Smoke");

JLabel status = new JLabel("Status");

JPasswordField password = new JPasswordField();

List<Sensor> reading;

LoginFrame() {

contents();

}

//set JFrame components

public void contents() {

setLayoutManager();

setLocationAndSize();

addComponentsToContainer();

addActionEvent();

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

//get all sensors

try {

String registration = "//localhost/SensorService";

Remote remoteService = Naming.lookup(registration);

SensorService sensor = (SensorService) remoteService;

//first clear Jframe

container.removeAll();

addComponentsToContainer();

reading = sensor.getSensors();

//create rows for sensors

int j = 90;

for (Sensor i : reading) {

JLabel floor = new JLabel();

JLabel room = new JLabel();

JLabel co2 = new JLabel();

JLabel smoke = new JLabel();

JLabel status = new JLabel();

floor.setOpaque(true);

room.setOpaque(true);

co2.setOpaque(true);

smoke.setOpaque(true);

status.setOpaque(true);

floor.setText(String.valueOf(i.getLocation().getFloor\_no()));

room.setText(String.valueOf(i.getLocation().getRoom\_no()));

co2.setText(String.valueOf(i.getCo2\_level()));

smoke.setText(String.valueOf(i.getSmoke\_level()));

status.setText(String.valueOf(i.isActive()));

//check for fire alarm event

if(i.getCo2\_level() >= 5 || i.getSmoke\_level() >= 5) {

floor.setBackground(Color.RED);

room.setBackground(Color.RED);

co2.setBackground(Color.RED);

smoke.setBackground(Color.RED);

status.setBackground(Color.RED);

//send sms and email

//sensor.sendMail();

//sensor.sendSMS();

} else if(i.isActive() == true) {

floor.setBackground(Color.GREEN);

room.setBackground(Color.GREEN);

co2.setBackground(Color.GREEN);

smoke.setBackground(Color.GREEN);

status.setBackground(Color.GREEN);

}

//set Jframe bounds and add

floor.setBounds(10, j, 50, 30);

room.setBounds(60, j, 50, 30);

co2.setBounds(110, j, 50, 30);

smoke.setBounds(160, j, 50, 30);

status.setBounds(210, j, 50, 30);

container.add(floor);

container.add(room);

container.add(co2);

container.add(smoke);

container.add(status);

j+=40;

}

} catch (Exception e) {

e.printStackTrace();

}

}

public static void main(String args[]) {

//create admin panel

LoginFrame frame = new LoginFrame();

frame.setTitle("Login Form");

frame.setVisible(true);

frame.setBounds(500, 100, 290, 600);

frame.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

frame.setResizable(true);

}

//set JFrame layout

public void setLayoutManager() {

container.setLayout(null);

}

//set JFrame component location and size

public void setLocationAndSize() {

loginButton.setBounds(160, 10, 100, 30);

floor.setBounds(10, 50, 520, 30);

room.setBounds(60, 50, 50, 30);

co2.setBounds(110, 50, 50, 30);

smoke.setBounds(160, 50, 50, 30);

status.setBounds(210, 50, 50, 30);

password.setBounds(10, 10, 130, 30);

}

//add JFrame components

public void addComponentsToContainer() {

container.add(loginButton);

container.add(floor);

container.add(room);

container.add(co2);

container.add(smoke);

container.add(status);

container.add(password);

}

//add button events

public void addActionEvent() {

loginButton.addActionListener(this);

}

//button events

public void actionPerformed(ActionEvent e) {

if (e.getSource() == loginButton) {

char[] pwd;

String pwdText;

pwd = password.getPassword();

pwdText = new String(pwd);

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

//check login validation

try {

String registration = "//localhost/SensorService";

Remote remoteService = Naming.lookup(registration);

SensorService sensor = (SensorService) remoteService;

if (sensor.adminCheck(pwdText)) {

this.dispose();

ClientFrame frame = new ClientFrame();

frame.setTitle("Admin Panel");

frame.setVisible(true);

frame.setBounds(500, 100, 400, 600);

frame.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

frame.setResizable(true);

} else {

JOptionPane.showMessageDialog(this, "Invalid Username or Password");

}

} catch (Exception ex) {

ex.printStackTrace();

}

}

}

}

import java.rmi.RemoteException;

import java.rmi.registry.LocateRegistry;

import java.rmi.registry.Registry;

import java.rmi.server.UnicastRemoteObject;

import java.util.List;

import com.fasterxml.jackson.core.type.TypeReference;

import com.fasterxml.jackson.databind.ObjectMapper;

import java.io.BufferedReader;

import java.io.IOException;

import java.io.InputStreamReader;

import java.io.OutputStream;

import java.net.HttpURLConnection;

import java.net.MalformedURLException;

import java.net.URL;

//RMI server

public class SensorServiceProvider extends UnicastRemoteObject implements SensorService {

protected SensorServiceProvider() throws RemoteException {

super();

// TODO Auto-generated constructor stub

}

public static void main(String[] args) {

//connect with the rmi registry

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

try {

SensorServiceProvider sensor = new SensorServiceProvider();

Registry registry = LocateRegistry.getRegistry();

System.out.println("Loading Sensor service");

registry.bind("SensorService", sensor);

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

}catch(Exception e){

e.printStackTrace();

}

}

//get all sensors

public List<Sensor> getSensors() {

try {

URL url = new URL("http://localhost:8080/v1/sensors/");

HttpURLConnection conn = (HttpURLConnection) url.openConnection();

conn.setDoOutput(true);

conn.setRequestMethod("GET");

conn.setRequestProperty("Content-Type", "application/json");

ObjectMapper objectMapper = new ObjectMapper();

String output;

BufferedReader br = new BufferedReader(new InputStreamReader(

(conn.getInputStream())));

StringBuilder sb = new StringBuilder();

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

sb.append(output+"\n");

}

output = sb.toString();

List<Sensor> l = objectMapper.readValue(output,new TypeReference<List<Sensor>>(){});

output = br.readLine();

conn.disconnect();

return l;

} catch (MalformedURLException e) {

e.printStackTrace();

} catch (IOException e) {

e.printStackTrace();

}

return null;

}

//add new sensor

public void addSensor(int flr, int rm) throws RemoteException {

// TODO Auto-generated method stub

try {

URL url = new URL("http://localhost:8080/v1/sensors/");

HttpURLConnection conn = (HttpURLConnection) url.openConnection();

conn.setDoOutput(true);

conn.setRequestMethod("POST");

conn.setRequestProperty("Content-Type", "application/json");

String input = "{\"floor\_no\":" + flr + ",\"room\_no\":\"" + rm + "\"}";

OutputStream os = conn.getOutputStream();

os.write(input.getBytes());

os.flush();

String output;

BufferedReader br = new BufferedReader(new InputStreamReader(

(conn.getInputStream())));

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

}

conn.disconnect();

} catch (MalformedURLException e) {

e.printStackTrace();

} catch (IOException e) {

e.printStackTrace();

}

}

//get sensor by id

public Sensor getSensorById(String id) throws RemoteException {

// TODO Auto-generated method stub

try {

URL url = new URL("http://localhost:8080/v1/sensors/" + id);

HttpURLConnection conn = (HttpURLConnection) url.openConnection();

conn.setDoOutput(true);

conn.setRequestMethod("GET");

conn.setRequestProperty("Content-Type", "application/json");

String output;

BufferedReader br = new BufferedReader(new InputStreamReader(

(conn.getInputStream())));

StringBuilder sb = new StringBuilder();

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

sb.append(output+"\n");

}

ObjectMapper objectMapper = new ObjectMapper();

output = sb.toString();

Sensor sensor = objectMapper.readValue(output, Sensor.class);

conn.disconnect();

return sensor;

} catch (MalformedURLException e) {

e.printStackTrace();

} catch (IOException e) {

e.printStackTrace();

}

return null;

};

//update sensor details

public void updateSensor(int floor, int room, boolean status, String sensorId, String locationId) throws RemoteException {

try {

URL url = new URL("http://localhost:8080/v1/sensors/" + sensorId + "/locations/" + locationId);

HttpURLConnection conn = (HttpURLConnection) url.openConnection();

conn.setDoOutput(true);

conn.setRequestMethod("PUT");

conn.setRequestProperty("Content-Type", "application/json");

String input = "{\"active\":" + status +",\"location\":{\"floor\_no\":" + floor + ",\"room\_no\":" + room + "}}";

OutputStream os = conn.getOutputStream();

os.write(input.getBytes());

os.flush();

String output;

BufferedReader br = new BufferedReader(new InputStreamReader(

(conn.getInputStream())));

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

}

conn.disconnect();

} catch (MalformedURLException e) {

e.printStackTrace();

} catch (IOException e) {

e.printStackTrace();

}

}

//validate admin login

public boolean adminCheck(String password) throws RemoteException {

try {

URL url = new URL("http://localhost:8080/v1/users/Admin");

HttpURLConnection conn = (HttpURLConnection) url.openConnection();

conn.setDoOutput(true);

conn.setRequestMethod("GET");

conn.setRequestProperty("Content-Type", "application/json");

String output;

BufferedReader br = new BufferedReader(new InputStreamReader(

(conn.getInputStream())));

StringBuilder sb = new StringBuilder();

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

sb.append(output+"\n");

}

ObjectMapper objectMapper = new ObjectMapper();

output = sb.toString();

User user = objectMapper.readValue(output, User.class);

conn.disconnect();

if(password.equals(user.getPassword())) {

return true;

} else {

return false;

}

} catch (MalformedURLException e) {

e.printStackTrace();

} catch (IOException e) {

e.printStackTrace();

}

return false;

}

//send email notification

public void sendMail() throws RemoteException {

// TODO Auto-generated method stub

try {

URL url = new URL("http://localhost:8080/v1/users/Admin");

HttpURLConnection conn = (HttpURLConnection) url.openConnection();

conn.setDoOutput(true);

conn.setRequestMethod("GET");

conn.setRequestProperty("Content-Type", "application/json");

String output;

BufferedReader br = new BufferedReader(new InputStreamReader(

(conn.getInputStream())));

StringBuilder sb = new StringBuilder();

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

sb.append(output+"\n");

}

ObjectMapper objectMapper = new ObjectMapper();

output = sb.toString();

User user = objectMapper.readValue(output, User.class);

SendEmail email = new SendEmail(user.getReceiver\_email(), user.getSender\_email(), user.getSender\_password());

conn.disconnect();

} catch (MalformedURLException e) {

e.printStackTrace();

} catch (IOException e) {

e.printStackTrace();

}

}

//send sms notification

public void sendSMS() throws RemoteException {

// TODO Auto-generated method stub

try {

URL url = new URL("http://localhost:8080/v1/users/Admin");

HttpURLConnection conn = (HttpURLConnection) url.openConnection();

conn.setDoOutput(true);

conn.setRequestMethod("GET");

conn.setRequestProperty("Content-Type", "application/json");

String output;

BufferedReader br = new BufferedReader(new InputStreamReader(

(conn.getInputStream())));

StringBuilder sb = new StringBuilder();

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

sb.append(output+"\n");

}

ObjectMapper objectMapper = new ObjectMapper();

output = sb.toString();

User user = objectMapper.readValue(output, User.class);

SmsSender sms = new SmsSender(user.getReceiver\_mobile(),user.getSender\_mobile());

conn.disconnect();

} catch (MalformedURLException e) {

e.printStackTrace();

} catch (IOException e) {

e.printStackTrace();

}

}

}

// Install the Java helper library from twilio.com/docs/libraries/java

import com.twilio.Twilio;

import com.twilio.rest.api.v2010.account.Message;

import com.twilio.type.PhoneNumber;

//SMS sender class

public class SmsSender {

// Find your Account Sid and Auth Token at twilio.com/console

public static final String ACCOUNT\_SID =

"AC4a8b0fa10c8ae4f70a6b0c74040b56b3";

public static final String AUTH\_TOKEN =

"07add7b4a46b9378b1a876567734b078";

public SmsSender(String to, String from) {

Twilio.init(ACCOUNT\_SID, AUTH\_TOKEN);

Message message = Message

.creator(new PhoneNumber(to), // to

new PhoneNumber(from), // from

"Alert: One or more of the Overnight Coders® sensors have been triggered")

.create();

}

}

import java.io.Serializable;

//User object model

public class User implements Serializable {

/\*\*

\*

\*/

private static final long serialVersionUID = 1L;

private String id;

private String username;

private String password;

private String receiver\_email;

private String sender\_password;

private String sender\_mobile;

private String receiver\_mobile;

private Integer errorCode;

private String sender\_email;

public static long getSerialversionuid() {

return serialVersionUID;

}

public String getId() {

return id;

}

public void setId(String id) {

this.id = id;

}

public String getUsername() {

return username;

}

public void setUsername(String username) {

this.username = username;

}

public String getPassword() {

return password;

}

public void setPassword(String password) {

this.password = password;

}

public String getReceiver\_email() {

return receiver\_email;

}

public void setReceiver\_email(String receiver\_email) {

this.receiver\_email = receiver\_email;

}

public String getSender\_password() {

return sender\_password;

}

public void setSender\_password(String sender\_password) {

this.sender\_password = sender\_password;

}

public String getSender\_mobile() {

return sender\_mobile;

}

public void setSender\_mobile(String sender\_mobile) {

this.sender\_mobile = sender\_mobile;

}

public String getReceiver\_mobile() {

return receiver\_mobile;

}

public void setReceiver\_mobile(String receiver\_mobile) {

this.receiver\_mobile = receiver\_mobile;

}

public Integer getError\_code() {

return errorCode;

}

public void setError\_code(Integer errorCode) {

this.errorCode = errorCode;

}

public String getSender\_email() {

return sender\_email;

}

public void setSender\_email(String sender\_Email) {

this.sender\_email = sender\_Email;

}

}

import java.util.Properties;

import javax.mail.Message;

import javax.mail.MessagingException;

import javax.mail.PasswordAuthentication;

import javax.mail.Session;

import javax.mail.Transport;

import javax.mail.internet.InternetAddress;

import javax.mail.internet.MimeMessage;

//Email sender class

public class SendEmail {

public SendEmail(String to, final String from, final String password) {

// TODO Auto-generated method stub

// Get system properties

Properties properties = new Properties();

properties.put("mail.smtp.auth", "true");

properties.put("mail.smtp.starttls.enable", "true");

properties.put("mail.smtp.host", "smtp.gmail.com");

properties.put("mail.smtp.port", 587);

// Get the default Session object.

Session session = Session.getDefaultInstance(properties, new javax.mail.Authenticator() {

protected PasswordAuthentication getPasswordAuthentication() {

return new PasswordAuthentication(from, password);

}

});

try {

// Create a default MimeMessage object.

MimeMessage message = new MimeMessage(session);

// Set From: header field of the header.

message.setFrom(new InternetAddress(from));

// Set To: header field of the header.

message.addRecipient(Message.RecipientType.TO, new InternetAddress(to));

// Set Subject: header field

message.setSubject("Alert");

// Now set the actual message

message.setText("Alert: One or more of the Overnight Coders® sensors have been triggered");

// Send message

Transport.send(message);

} catch (MessagingException mex) {

mex.printStackTrace();

}

}

// public static void main(String args[]) {

// SendEmail v = new SendEmail("ranulds@gmail.com","overnightcoders92@gmail.com","overnightcoders92admin");

// System.out.println("sent");

// }

}

import javax.swing.JButton;

//custom JButton class to get sensor details

public class MyButton extends JButton {

private String sensorId;

private String locationId;

public MyButton(){

super();

}

public String getSensorId() {

return sensorId;

}

public void setSensorId(String sensorId) {

this.sensorId = sensorId;

}

public String getLocationId() {

return locationId;

}

public void setLocationId(String locationId) {

this.locationId = locationId;

}

}

## Fire sensor Application

package sensor.model;

import java.util.List;

import java.util.Random;

import com.fasterxml.jackson.core.JsonProcessingException;

import com.fasterxml.jackson.databind.ObjectMapper;

import io.restassured.RestAssured;

import io.restassured.path.json.JsonPath;

import io.restassured.response.Response;

import io.restassured.specification.RequestSpecification;

public class Sensor {

public int getId() {

return \_id;

}

public void setId(int id) {

this.\_id = id;

}

public int getCo2() {

return co2\_level;

}

public void setCo2(int co2) {

this.co2\_level = co2;

}

public int getSmoke() {

return smoke\_level;

}

public void setSmoke(int smoke) {

this.smoke\_level = smoke;

}

public int getRoom\_number() {

return room\_number;

}

public void setRoom\_number(int room\_number) {

this.room\_number = room\_number;

}

public int getFoor\_number() {

return foor\_number;

}

public void setFoor\_number(int foor\_number) {

this.foor\_number = foor\_number;

}

public Sensor(int id, int co2, int smoke, int room\_number, int foor\_number) {

super();

this.\_id = id;

this.co2\_level = co2;

this.smoke\_level = smoke;

this.room\_number = room\_number;

this.foor\_number = foor\_number;

}

public Sensor() {

super();

}

private int \_id;

private int co2\_level;

private int smoke\_level;

private int room\_number;

private int foor\_number;

public void addSensor(Sensor sensor) {

Response response;

ObjectMapper mapper = new ObjectMapper();

String json;

try {

json = mapper.writeValueAsString(sensor);

RestAssured.baseURI="http://localhost:8080";

RequestSpecification request=RestAssured.given();

request.header("Content-Type","application/json");

request.body(json);

System.out.println(json);

response = request.post("/v1/sensors");

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

} catch (JsonProcessingException e) {

// TODO Auto-generated catch block

e.printStackTrace();

}

};

public void updateSensor(Sensor sensor) {

for(int i=0;i>=-1;i++) {

sensor.setCo2(getrandom(1,10));

sensor.setSmoke(getrandom(1,10));

Response response;

Response response2=null;

ObjectMapper mapper = new ObjectMapper();

String json;

try {

json = mapper.writeValueAsString(sensor);

RestAssured.baseURI="http://localhost:8080";

RequestSpecification request=RestAssured.given();

request.header("Content-Type","application/json");

request.body(json);

response2=RestAssured.given().when().get("/v1/sensors/active");

JsonPath jsonPathEvaluator = response2.jsonPath();

List values = jsonPathEvaluator.getList("\_id");

for (int k = 0; k < values.size(); k++) {

System.out.println("updated sensor value :"+json);

response = request.put("/v1/sensors/"+values.get(k)+"/co2/" + getrandom(1,10) + "/smoke/" + getrandom(1,10));

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

Thread.sleep(5000);

}

} catch (JsonProcessingException e) {

// TODO Auto-generated catch block

e.printStackTrace();

} catch (InterruptedException e) {

// TODO Auto-generated catch block

e.printStackTrace();

}

}

};

private static int getrandom(int min,int max) {

Random r=new Random();

return r.nextInt((max-min)+1)+min;

};

}