Sri Lanka Institute of Information Technology

Assignement (II)

SE3020 – Distributed System

REST API Project

Hotel Fire Alarm System

Student Name	IT Number
Perera H.D.D.S	IT18006544
Kalubowila D.C	IT18028010

SLIIT
Discover Your Future

BSc (Hons) in Information Technology
Year 3

Assignment II – REST API Project

SE3020 – Distributed Systems **Semester 1**

Content

BSc (Hons) in Information Technology
Year 3

Assignment II – REST API Project

SE3020 – Distributed Systems **Semester 1**

1.0 Introduction

In this report, it will be discussed how components in a distributed system communicate with each other. First of all, it is necessary to find out what is a distributed system. A distributed system is a system where different components of a system are situated in a separate machine and these components will communicate with each other, while the end-user will experience a coherent system. The main reasons to use a distributed system are being able to scale the system horizontally, reliability, and performance.

And also it is necessary to understand what is a REST API. API stands for ''Application programming interface'', which defines a set of rules when the client communicates with the server. REST stands for "Representational State Transfer". In REST the rules state that when a certain URL has executed a set of data could be retrieved.

Another important part of a distributed system is invoking remote methods. .NET Remoting which is Microsoft's approach of invoking remote methods is used in this project. Common Object Request Broker Architecture (CORBA) and Remote Method Invocation of Java are similar technologies.

Web application, desktop application, sensor, and SMS gateway are the components of this distributed system. The desktop application will communicate with the remoting server to pass information to the API. The web application will communicate with API by using a proxy. Sensor and the SMS gateway has direct access to the API

Assignment II – REST API Project

SE3020 – Distributed Systems **Semester 1**

2.0 Tools and Technologies used.

1. Web Client.
    - React JS
    - Bootstrap
    - VS code

2. REST API
    - Node JS
    - Express JS
    - Mongo DB Atlas
    - Postman

3. Desktop Application, Sensor and Remoting server
    - C# .NET
    - Visual Studio Code

4. SMS gateway
    - PHP web API
    - Android

5. Mail Trapper (Email)
6. JSON Web Token

Assignment II – REST API Project

SE3020 – Distributed Systems Semester 1

3.0 Distributed System Components

3.1 Web Client

The frontend of the web application was developed using react and bootstrap. Using the Axios library, the frontend will communicate with the rest API to retrieve information. Here a method called set time interval() is used to mount the component very 30s thereby refresh the data that the user an view.

```
componentDidMount() {
    // need to make the initial call to autoFetch() to populate
    // data right away
    this.autoFetch();
    console.log(this.state.rooms)
    // Now we need to make it run at a specified interval
    setInterval(this.autoFetch, 30000); // runs every 30 seconds
}
```

Figure 1: Frontend component mounting

```
autoFetch = () =>{
  axios.get(`/api/room`)
  .then(res => {
    const rooms =  res.data;
    console.log(this.state.rooms)
    this.setState({rooms})
  })
}
```

Figure 2: Fetching data from api

From the client, web view users will be able to observe the details of all fire alarm sensors while getting the latest information every 30s. The alarm sensor will continuously send data to the database via the rest API. Smoke level and Co2 level will be measured from a scale of 1 – 10. Rooms with smoke level or Co2 level higher than 7 will be marked in dark red, smoke level, or Co2 level between 5 and 7 will be marked in light red and smoke level or CO2 level less than 5 will be marked in green color. These readings will monitor regularly.

Smoke Level	CO2 Level	Color
>= 7	>= 7	Dark red
>=5 and <7	>=5 and <7	Light red
< 5	< 5	Green

Table 1 :- Smoke and CO2 measures

And the user will also be able to observe the highest and lowest smoke level recorded, average smoke level, and the total number of rooms. These data will also be updated accordingly.

3.2 REST API

Rest API was developed using Node js, Express Js, and MongoDB Atlas as the database.

1. Models

- The mongoose schema for a room has 15 attributes. “roomNo” attribute will be unique for each room. When a new room is added to the system the fire alarm sensor status will be false by default. And the co2 level and smoke level will be 0 by default. Furthermore the SMS and Email status will be false by default because the customers are unavailable in the room when a new room is added to the system. When assigning a new user to the room the NIC, name, email, and mobile number of the user will be assigned to the user attribute of the room’s model.
- The mongoose schema for the user model has 5 attributes. The model is used to register administrators in the system. All five attribute is “required” fields.

2. Endpoints

- API/rooms/:- From this get request details about all the rooms can be retrieved. This API is re-used in both desktops as well as the web application. Implementation of email sending is also embedded in this API. But before sending the email it will check whether a specific room is existing, whether there is a user available in the specific room, whether the smoke level or co2 level is above level 5, and also checks whether the already a mail has been sent due to an issue. Once all these requirements are filled the mail will be sent. The desktop application will trigger this endpoint every 40s as well as the web application will trigger this every 30s.

- API/rooms/add room:- From this post request, a new room will be added to the system. From the request body which will be sent from the desktop application contains the room number, floor, and the floor number. Before adding a room to the system it will be first checked that whether a room is already available with the same room number sent through the request body. If a room is available an error message will be sent as the response. If not the room will be added to the system.

- Api/rooms/addCustomer/:roomNo:- From this put request a new customer will be added to a specific room. But before adding the customer to the room it will be checked whether the specified room is available in the system. In this API room number will send as a parameter and the nic, email, mobile, name of the customer will be sent through the request body by the desktop application. When a new customer is added to the room the status of the fire alarm will be switched to the active mode.

- Api/rooms/addSensor/:roomNo:- From this put request new sensor details can be updated to a specific room. Room number will be sent as a parameter and co2 level and the smoke level readings from the sensor will be sent through the request body. The existence of the specified room will be checked here as well.

- Api/rooms/alert:- From this get request all the room that needs SMS alerts will be triggered. Initially, all the rooms that have users will be fetched from the database and check each room's smoke and co2 level and also check whether already an SMS has been sent to the specific user. If these conditions are met, separate SMS stating the co2 level and smoke level will be sent to the specific user of the specific room.

  ```
  " url: `http://api.liyanagegroup.com/sms_api.php?sms=Room ${room[i].roomNo}-CO2 Alarm is Active&to=94${room[i].user.mobile}&usr=0766061689&pw=4873` "
  ```

  The above URL is used to enable the SMS component. Room number, message, co2, and smoke level will be appended to this URL.

- Api/rooms/deleteRoom/:roomNo:- From this route the administrator can delete a specific room. Here the room number will be sent as the parameter. Here also initially checking is done whether the specified room is available.

- Api/rooms/removeUser/:roomNo:- From this API is used to remove an existing user from a specific room. Here also room number will be sent as the parameter. After the customer is removed from the room, that room will be set to default values.

- Api/users/:- purpose of this API is to fetch all the user data from the database.

- Api/users/register:- This API will be accessed from the desktop application and this is used to add new administrators to the system. Name, email, password, the mobile number will be sent through the request body. A received password will be hashed using bcrypt js before saving the

administrator to the database.

- Api/users/login:- this API is used to do the login functionality of the desktop application. Email and password will be sent through the request body. Once again the bcrypt js is used decode and compare the password that was sent through the request body with the password which is coming from the database.

The following table will show the endpoints created for sending and retrieving data

Method	Endpoint	Purpose
GET request	Localhost:5000/api/rooms	Get all rooms
POST request	Localhost:5000/api/rooms/addRoom	Add a new Room
PUT request	Localhost:5000/api/rooms/addCustomer/:roomNo	Add a customer to a room
PUT request	Localhost:5000/api/rooms/addSensor/:roomNo	Update sensor details of a room
GET request	Localhost:5000/api/rooms/alert	Alert user about smoke and co2 level
DELETE request	Localhost:5000/api/rooms/deleteRoom/:roomNo	Delete a room
PUT request	Localhost:5000/api/rooms/removeUser/:roomNo	Remove user from a room
GET request	Localhost:5000/api/users/	Get details of the user
GET request	Localhost:5000/api/users/register	Register an administrator
GET request	Localhost:5000/api/users/login	Login a user

Table 2 :- API endpoint summary

3.3 SMS and Email Component

A custom-developed android application is used to implement the SMS sending component of this system. This application runs on a PHP API, which will be connected to the rest API. User mobile number information is gathered when assigning a room to a customer is used here. When co2 level or smoke level reaches above level 5 the customer will be alerted an SMS.

Even though the data refreshes every 10 seconds, and if the co2 level or smoke level remains above level 5, the customer will not be flooded with SMS. Once the co2 or smoke level get back to the safety level the user of that specific room will be notified again by an SMS

A web application called “Mail Trapper” is used as the email sending component for this system. Same as the SMS component, when co2 level or smoke level rise above level 5 customer will be notified by an email with information such as room No, co2 level and smoke level.

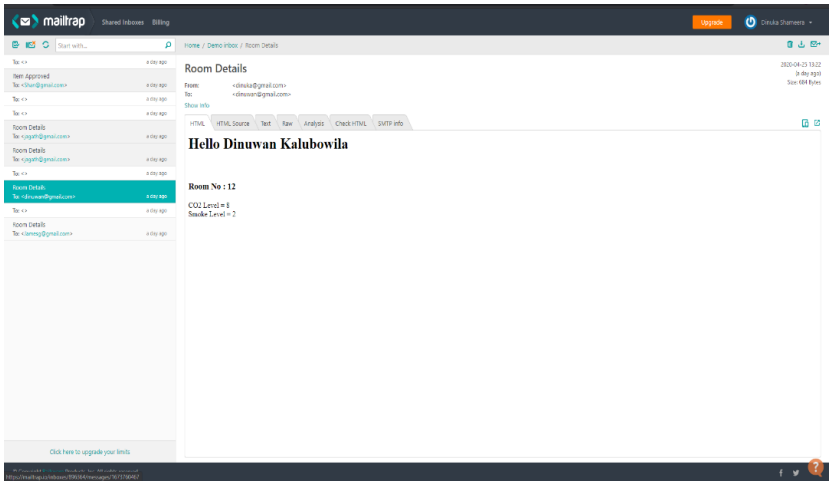

Figure 3.1: Email recieving for customers

Figure 4.2: SMS recieving for customers

3.4 Security and Authentication

Administrators of this system are supposed to add new rooms with a fire alarm system and assign new users to the rooms. Before being able to these operations the admin has to register to the system giving email and password as admin credentials. As the security aspect of this part, the provided password will be hashed before storing it to the database by using a library called “bycrpt.js”. Thereby the actual password will be stored as an encrypted password.

When a user logs into the system JSON web token are used to authenticate them. “JSON web token (jwt) is an open, industry-standard RFC 7519 method for representing claim securely between two parties”. Here the web token contains all the necessary information which is used when authenticating the user. This information will be attached to the request header before passing them to the backend.

3.5 Desktop Application Component

The desktop application components are developed using C# with .Net framework. In the desktop application, there are four major parts.

1. Client application
2. Server application
3. Remote interface
4. Remote method implementation

Here the client application and server application runs separately, but these two components will be connected using remote methods. Interfaces and method implementation are created separately where both server and client will not be able to see the method implementation. After the interfaces are created, a remote registration created inside the server thereby creating a reference to the remote methods, because of that the server can invoke the method which is remotely located.

When compared to Java's RMI, .Net remoting is different. In RMI method implementation is done inside the server therefore from the server side it is possible to see the implementation of the method, but in .Net remoting interfaces and method, implementation is located separately.

.NET Remoting is run on top of the TCP connection. Therefore we established Remoting Server and Remoting Client on TCP Connection.

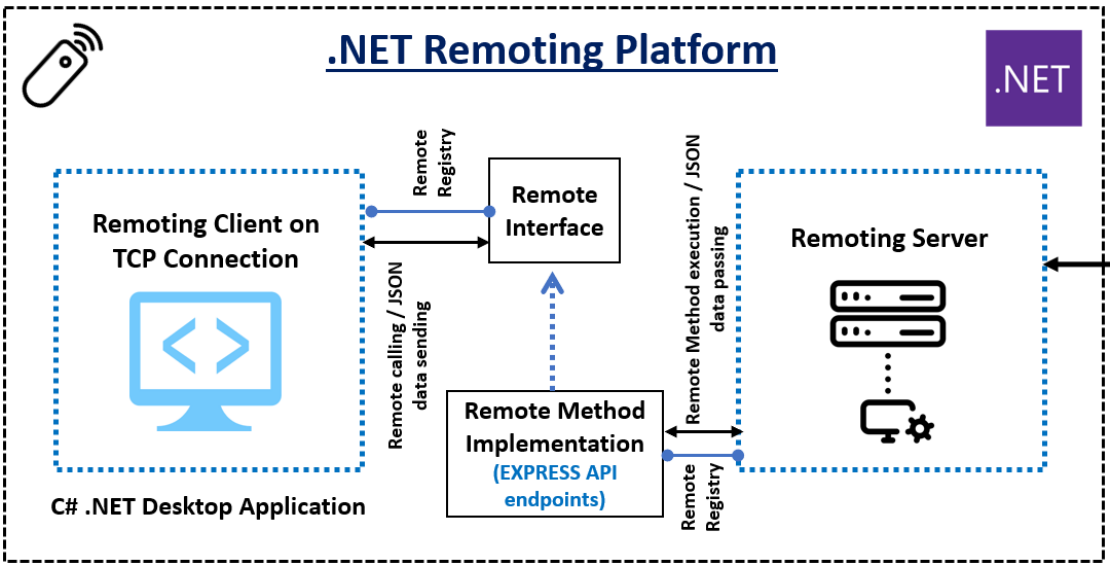

Figure 4: .NET Remoting Structure

Assignment II – REST API Project

SE3020 – Distributed Systems **Semester 1**

4.0 High-level Architectural diagram

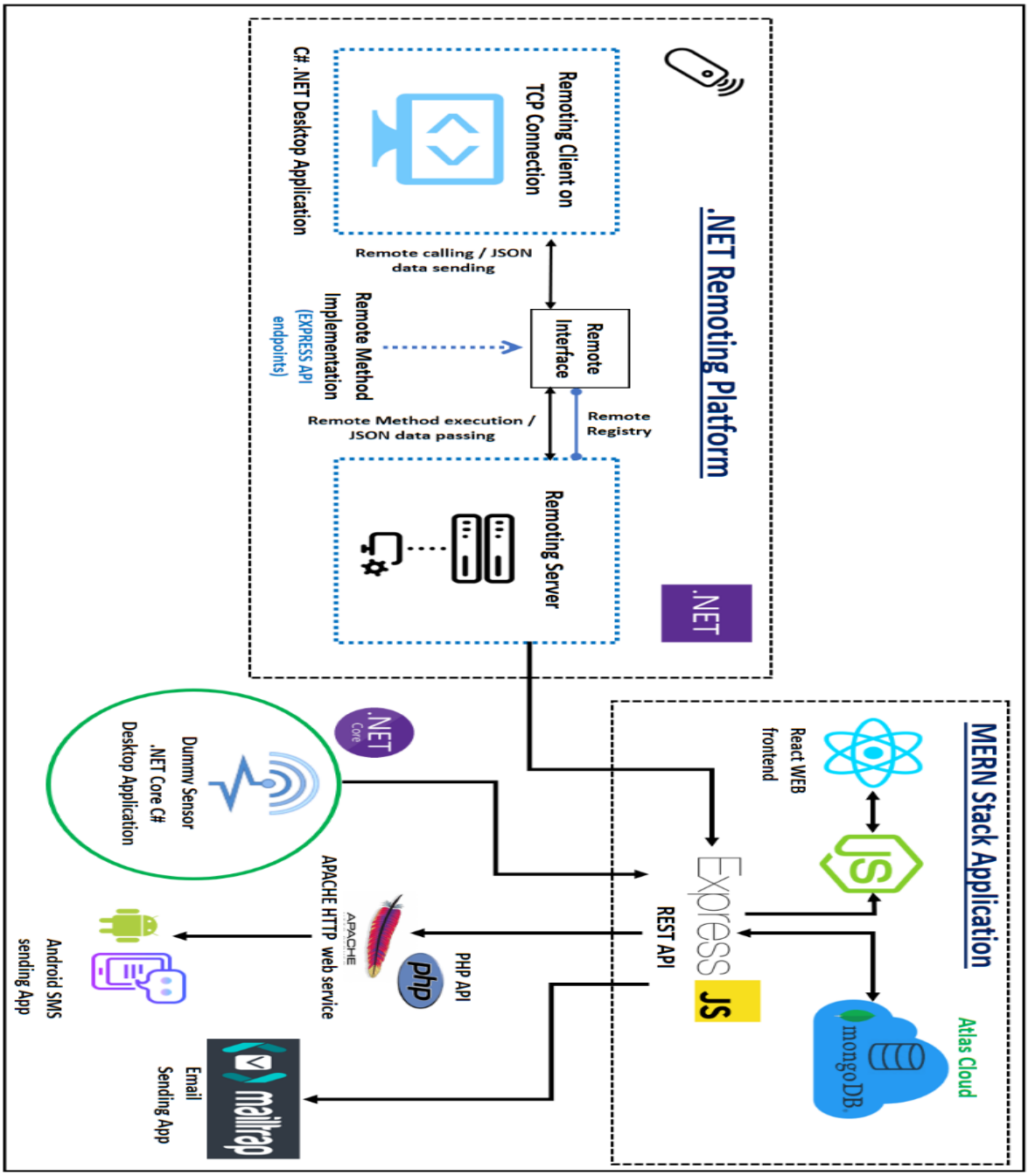

Figure 5: High Level architecture diagram

5.0 Sequence Diagram

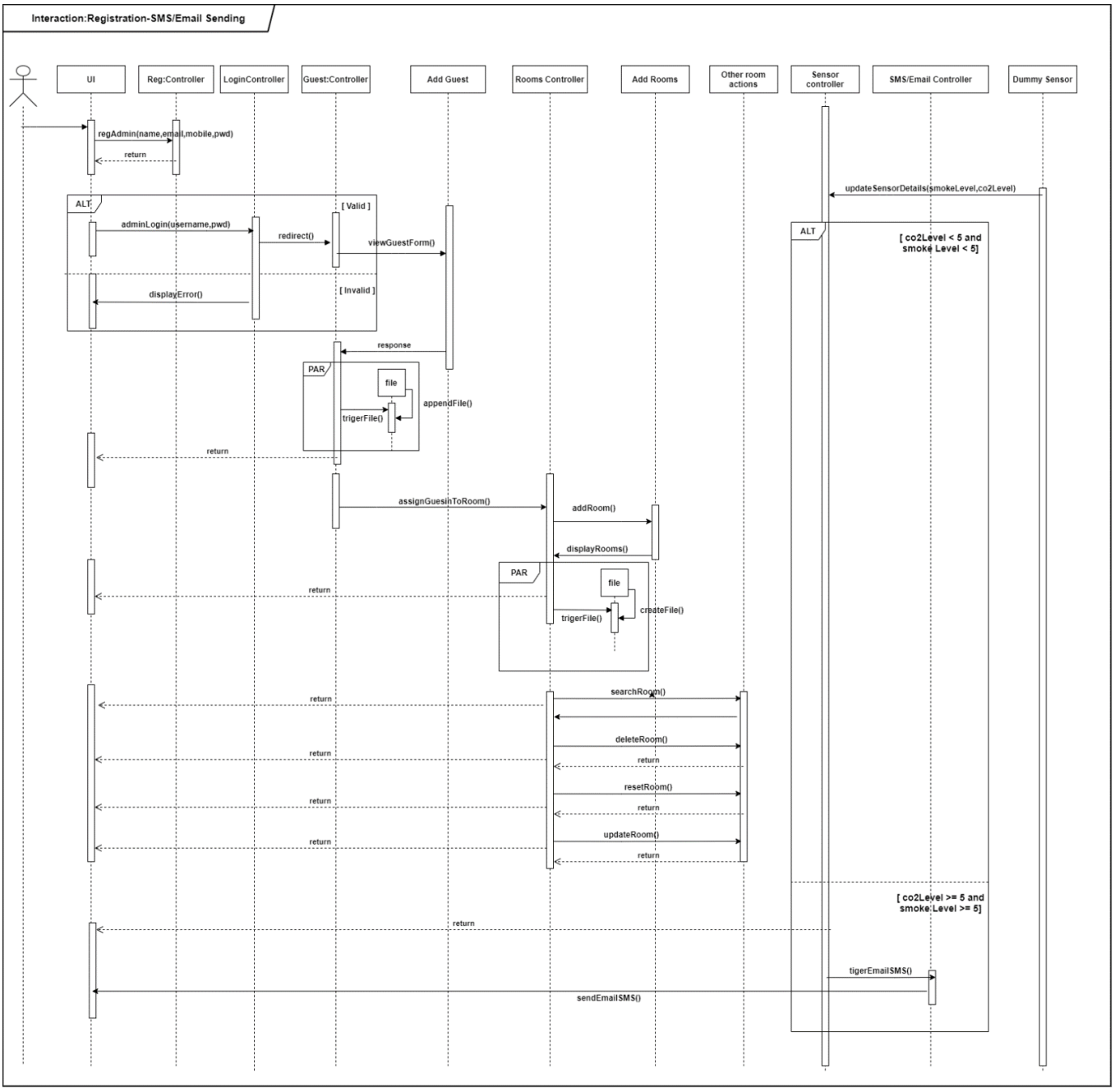

Figure 6: Sequence Diagram for the system

6.0 Class Diagrams

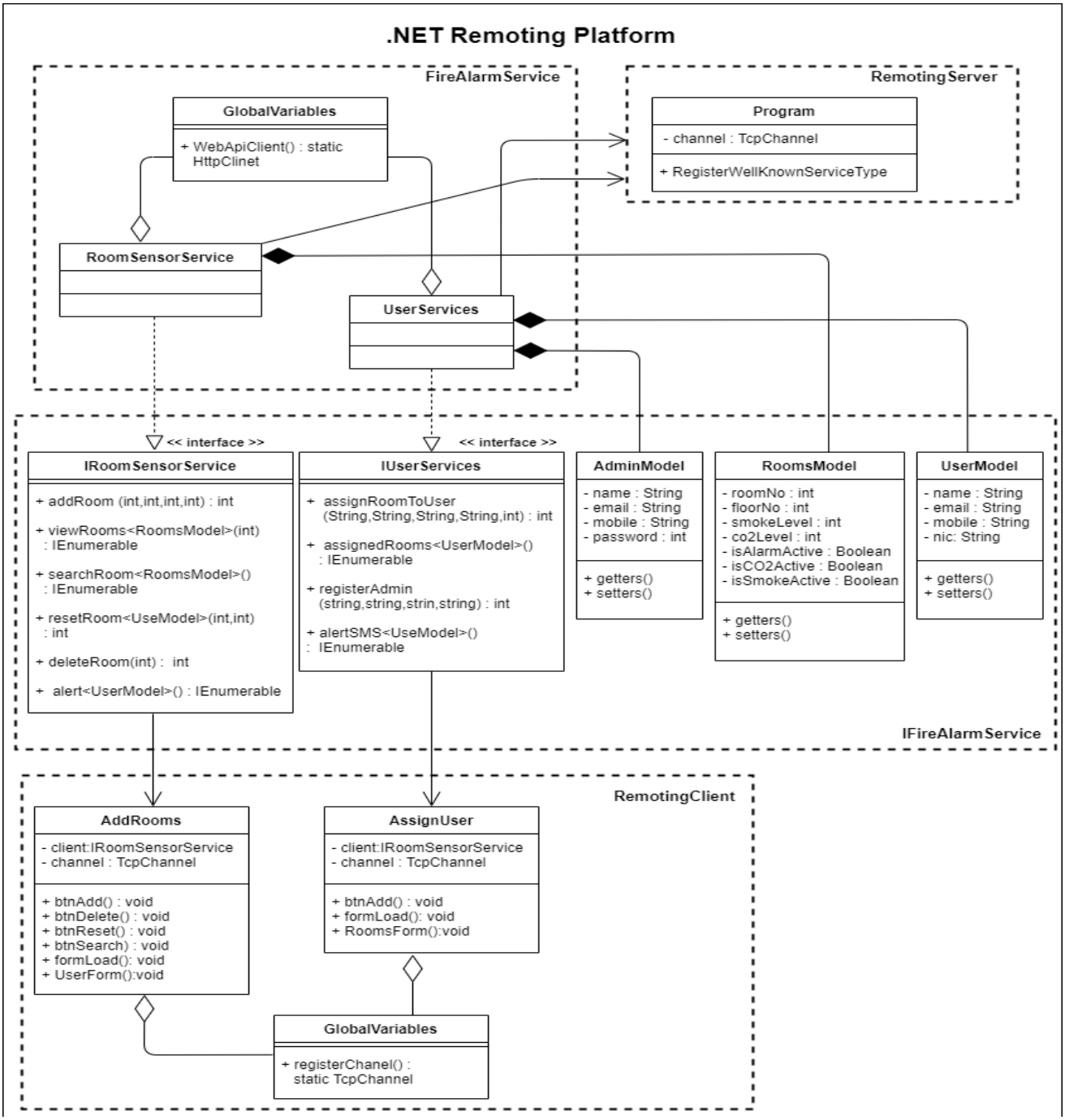

Figure 7: Class diagram for the desktop applciation

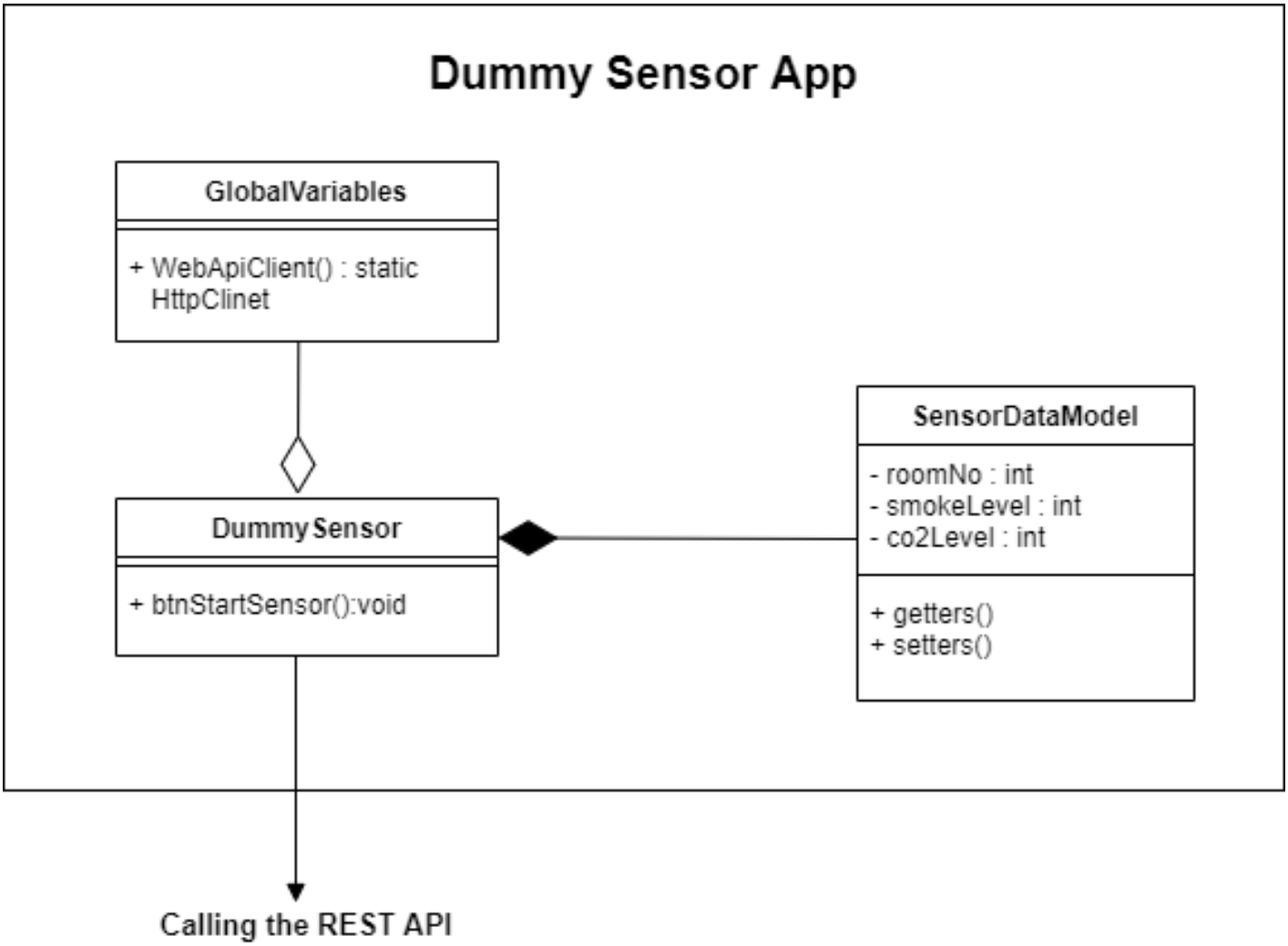

Figure 8: Class diagram for the sensor application

7.0 Appendix

Web Client

```
import React, { Component } from
"react";
import axios from "axios";
import { BrowserRouter as Router, Route, Switch }
from "react-router-dom";
import NavBar from
"./Components/NavComponent/navcomponent";
import Cardlist from
"./Components/CardComponent/CardListComponent";

import Table from
"./Components/TableComponent/Table";
import Chart from
"./Components/ChartComponent/Chart";
import Footer from "./Components/Footer/Footer";
import "./App.css";

class App extends Component {
  state = {
    rooms: [],
  };

  autoFetch = () => {
    axios.get(`/api/room`).then((res) => {
      const rooms = res.data;
      console.log(res.data);
      this.setState({ rooms });
      console.log(this.state.rooms)
    });
```

```
axios.put('/api/room/smsEmailStatus').then((res) =>
{
      console.log(res)
    }).catch((err) => console.error(err))


    axios.get('api/room/alert')
    .then(res=>{
      console.log(res);
    })
    .catch(err=>{
      console.log(err);
    })

  };

  componentDidMount() {
    // need to make the initial call to autoFetch()
to populate
    // data right away
    this.autoFetch();
    console.log(this.state.rooms);
    // Now we need to make it run at a specified
interval
    setInterval(this.autoFetch, 30000); // runs
every 30 seconds
  }

  render() {
    return (
      <Router>
        <div className="App">
          <NavBar />
          <div className="container">
            <div className="cardList">
              <Route
                exact
```

```
                    path="/"
                    component={() => <Cardlist
rooms={this.state.rooms} />}
                  />
                </div>
                <div className="tableContainer">
                  <Route
                    exact
                    path="/"
                    component={() => <Table
rooms={this.state.rooms} />}
                  />
                </div>

                <div className="chartcontainer">
                  {/* <Chart /> */}
                </div>
              </div>
              <Footer />
            </div>
          </Router>
        );
      }
    }

    export default App;
```

```
import React from "react";
import Card from "./Card";
export default function
CardListComponent({rooms}) {
   const array = [];
  let average = 0;
  rooms.map(room => {

array.push(room.smokeLevel)
  })
```

```
  for (let index = 0; index <
rooms.length; index++) {
    average =
parseFloat(average) +
parseFloat(rooms[index].co2Lev
el)

  }

  return (
    <div class="row">
      <Card
        count={rooms.length}
        color={"bg-info"}
        name={"Total Rooms"}
        icon={"fas fa-arrow-
circle-right"}
      />
      <Card

count={Math.max(...array)}
        color={"bg-success"}
        name={"Highest
Recorded Smoke Level"}
        icon={"fas fa-arrow-
circle-right"}
      />
      <Card
        count={(average /
rooms.length).toFixed(3)}
        color={"bg-warning"}
        name={"Average"}
        icon={"ion ion-person-
add"}
      />
      <Card

count={Math.min(...array)}
```

```
                color={"bg-danger"}
                name={"Lowest Recorded
Smoke Level"}
                icon={"ion ion-pie-
graph"}
              />
            </div>
          );
        }
```

```
import
React
from
"react";

export default function Card({count,name,color,icon}) {
  return (
    <div class="col-lg-3 col-6">
      <div class={`small-box ${color}`}>
        <div class="inner">
  <h3>{count}</h3>

  <p>{name}</p>
        </div>
        <div class="icon">
          <i class={icon}></i>
        </div>
        <a href="#" class="small-box-footer">
          More info <i class="fas fa-arrow-circle-right"></i>
        </a>
      </div>
    </div>
  );
}
```

```
import
React
```

```
from
"react";
           import {Link} from 'react-router-dom'
           export default function navcomponent() {
             return (
               <nav className="navbar navbar-expand-lg navbar-dark bg-dark">
                 <a className="navbar-brand" href="#">
                     Fire Alarm System
                 </a>
                 <button
                   className="navbar-toggler"
                   type="button"
                   data-toggle="collapse"
                   data-target="#navbarNavDropdown"
                   aria-controls="navbarNavDropdown"
                   aria-expanded="false"
                   aria-label="Toggle navigation"
                 >
                   <span className="navbar-toggler-icon"></span>
                 </button>
                 <div className="collapse navbar-collapse" id="navbarNavDropdown">
                   <ul className="navbar-nav">
                     <li className="nav-item active">
                       <Link to ='/' className="nav-link" href="#">
                         Admin <span className="sr-only">(current)</span>
                       </Link>
                     </li>
                     <li className="nav-item">
                       <a className="nav-link" href="#">
                         Client
                       </a>
                     </li>
                   </ul>
                 </div>
               </nav>
             );
           }
```

```
import
React
from
"react";

            export default function Footer() {
              return (
                <div className='bg-dark d-flex justify-content-center' style = {style.body}>
                  <div className="footer-dark ">
                    <footer>
                      <div className="container">
                        <div className="row">
                          <div className="col item social">
                            <a href="#">
                              <i className="fab fa-google-plus-square fa-3x  mr-3 mb-3 mt-3"
            style={style.icon}></i>
                            </a>
                            <a href="#">
                              <i className="fab fa-twitter-square fa-3x  mr-3 mb-3 mt-
            3"></i>
                            </a>
                            <a href="#">
                              <i className="fa fa-facebook-square fa-3x mr-3 mb-3 mt-3"></i>
                            </a>

                          </div>
                        </div>
                        <p className="copyright text-light">Company Name © 2020</p>
                      </div>
                    </footer>
                  </div>
                </div>
              );
            }
```

```
const style = {
    body : {
       height:'150px',
       marginTop : '75px'
    },
    icon : {
        size :'35px'
    }
}
```

Rest Api

```
const express =
require("express");
                    const mongoose = require("mongoose");
                    const users = require("./Routes/api/Users");
                    const rooms = require("./Routes/api/Rooms");

                    const app = express();


                    app.use(express.json());
                    app.use("/api/users", users);
                    app.use("/api/room", rooms);
                    //connecting to the dataBase
                    //mongodb+srv://ds123:ds123@dswebprojectcluster-
                    jvrhf.mongodb.net/test?retryWrites=true&w=majority
                    mongoose
                      .connect("mongodb://localhost:27017/ds_proj_remote", {
                        useNewUrlParser: true,
                        useUnifiedTopology: true
                      })
                      .then(() => console.log("connected to mongo DB"))
                      .catch((error) => console.error(error));

                    const PORT = process.env.PORT || 5000;
```

```
app.listen(PORT, () => console.log(`Listening to PORT ${PORT}`));
```

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

const userSchema = new mongoose.Schema({
  name:{
    type: String,
    required:true
  },
  email:{
    type: String,
    unique: true
  },
  password:{
    type: String,
    required:true
  },
  isAdmin:{
      type : Boolean,
      default : true
  },
  mobile:{
    type : String,
    required : true
  }
});

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

exports.User = User;
```

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

const roomSchema = new mongoose.Schema({
  roomNo: {
    type: Number,
    unique: true,
  },
  floorNo: {
    type: Number,
  },
  user: {
    nic: {
      type: String,
      default: null,
    },

    email: {
      type: String,
      default: null,
    },
    mobile: {
      type: String,
      default: null,
    },
    name:{
      type:String,
      default:null
    },
    password:{
      type:String,
      default:null
    }
  },
  isAlarmActive: {
    type: Boolean,
    default: false,
  },
```

```
    smokeLevel: {
      type: Number,
      default: 0,
    },
    co2Level: {
      type: Number,
      default: 0,
    },
    isCO2Active:{
      type:Boolean,
      default:false
    },
    isSmokeActive:{
      type:Boolean,
      default:false
    },
    isSmokeSMSSent:{
      type:Boolean,
      default:false
    },
    isCO2SMSSent:{
      type:Boolean,
      default:false
    },
    isMailSent:{
      type:Boolean,
      default:false
    }

});

const Room = mongoose.model("Room", roomSchema);

exports.Room = Room;
```

```
const express =
require("express"
);
                    const router = express.Router();
                    const { Room } = require("../../models/Rooms");
                    const nodemailer = require("nodemailer");
                    const axios = require("axios");

                    const fs = require('fs');
                    const path = require('path');
                    //getting Rooms
                    //getting Rooms
                    router.get("/", async (req, res) => {
                      try {
                        const room = await Room.find();

                        // //sending email to the client
                        for (let index = 0; index < room.length; index++) {
                          if (
                            ((room[index].co2Level >= 5 && room[index].co2Level <= 10) ||
                              (room[index].smokeLevel >= 5 && room[index].smokeLevel <=
                    10)) &&
                            room[index].isAlarmActive === true &&
                            room[index].isMailSent === false
                          ) {
                            let transporter = nodemailer.createTransport({
                              host: "smtp.mailtrap.io",
                              port: 2525,
                              auth: {
                                user: "cd1d2b8d863288",
                                pass: "41ff48f6db0ffa",
                              },
                            });

                            // send mail with defined transport object
                            let info = await transporter.sendMail({
                              from: "dinuka@gmail.com", // sender address
                              to: room[index].user.email, // list of receivers
```

```
            subject: "Item Approved", // Subject line
            text: "Hello world?", // plain text body
            html: `<h1>Hello ${room[index].user.name}</h1><br/>
                    <span><h3>Room No : ${room[index].roomNo}</h3></span>
                    <span>CO2 Level = ${room[index].co2Level}</span><br/>
                    <span>Smoke Level = ${room[index].smokeLevel}</span>
`, // html body
          });

          transporter.sendMail(info, function (err, info) {
            if (err) {
              console.log(err);
            } else {
              console.log(info);
            }
          });
          room[index].isMailSent = true;
          await room[index].save();
        }
      }

      res.send(room);
      //console.log(room);
    } catch (e) {
      console.log(e);
    }
});

router.put("/smsEmailStatus", async (req, res) => {
  try {
    const room = await Room.find();
    console.log("excuted The put method");
    setTimeout(async function () {
      try {
        for (let index = 0; index < room.length; index++) {
          if (
            ((room[index].co2Level >= 5 && room[index].co2Level <= 10)
||
```

```
              (room[index].smokeLevel >= 5 && room[index].smokeLevel <=
10)) &&
              room[index].isAlarmActive === true
            ) {
              room[index].isMailSent = true;
              await room[index].save();
            } else {
              room[index].isMailSent = false;
              await room[index].save();
            }
          }
        } catch (error) {}
      }, 2000);

      const result = await room.save();
      res.send(result);
    } catch (error) {}
});

//addd new Rooms
//just to add users to no sql
router.post("/addroom", async (req, res) => {
  console.log(req.body);
  try {
    //destructuring the req body
    const {
      roomNo,
      floorNo,
      user,
      isAlarmActive,
      smokeLevel,
      co2Level,
      isCO2Active,
      isSmokeActive,
      isSmokeSMSSent,
      isCO2SMSSent,
      isCO2MailSent,
      isSmokeMailSent,
```

```
    } = req.body;

    //checking whether the user with same room address exist

    let room = await Room.findOne({ roomNo });

    if (room) return res.status(400).send("Room already exists");

    room = new Room({
      roomNo,
      floorNo,
      user,
      isAlarmActive,
      smokeLevel,
      co2Level,
      isCO2Active,
      isSmokeActive,
      isSmokeSMSSent,
      isCO2SMSSent,
      isCO2MailSent,
      isSmokeMailSent,
    });
    const result = await room.save();


    //create folder
    fs.mkdir(path.join(`${__dirname}/res`, `/Room${roomNo}`),{},err =>{
      if(err) throw err;
      console.log('Folder Createted...');

      //Create and write to file

fs.writeFile(path.join(__dirname,`/res/Room${roomNo}`,'info.txt'),`Floo
r No : ${req.body.floorNo}`,err =>{
              if(err) throw err;
               console.log('File written to...');
          });
    });
```

```
    res.status(200).json(result);
  } catch (e) {
    res.send(e);
    console.log(e);
  }
});

//adding customers to the room
router.put("/addCustomer/:roomNo", async (req, res) => {
  try {
    //checking for the room existence
    //console.log(req);

    console.log("DOT NET sending request is : " ,req.body);


    let room = await Room.findOne({ roomNo: req.params.roomNo });
    if (!room) return res.status(400).send("No Such Room exist");

    //Append the file

fs.appendFile(path.join(__dirname,`/res/Room${req.params.roomNo}`,'info
.txt'),
    `\nUsername : ${req.body.name}\nNIC :
${req.body.nic}\nMobile:${req.body.mobile}\nEmail:${req.body.email}\nAd
ded date : ${Date(Date.now()).toString()}\n\n\nSensor Data\n\n`
    ,err =>{
      if(err) throw err;
      console.log('File Append to...');
    });

    room.user.nic = req.body.nic;
    room.user.email = req.body.email;
    room.user.mobile = req.body.mobile;
    room.user.name = req.body.name;
    room.user.password = req.body.password;
```

```
    room.isAlarmActive = true;

    await room.save();



    res.json(room);
  } catch (error) {
    res.send(error);
  }
});

//adding sensor details to the room
router.put("/addSensor/:roomNo", async (req, res) => {
  try {
    //checking for the room existence

    let room = await Room.findOne({ roomNo: req.params.roomNo });
    console.log(room);
    if (!room) return res.status(400).send("No Such Room exist");

    room.smokeLevel = req.body.smokeLevel;
    room.co2Level = req.body.co2Level;

      //Append the file

fs.appendFile(path.join(__dirname,`/res/Room${req.params.roomNo}`,'info
.txt'),
      `Smoke Level : ${req.body.smokeLevel}\t\tCO2 Level :
${req.body.co2Level}\t\tTime : ${Date(Date.now()).toString()}\n`
      ,err =>{
        if(err) throw err;
        console.log('File Append to...');
      });

    async function updateAlarms(id) {
      const updatedRoom = await Room.findById(id);
      if (!updatedRoom) {
```

```
        res.status(404).json({ msg: "Not Found" });
      }

      //chekcing the smoke level
      if (updatedRoom.smokeLevel <= 10 && updatedRoom.smokeLevel >= 5)
{
        updatedRoom.isSmokeActive = true;
      }else if(updatedRoom.smokeLevel >= 0 && updatedRoom.smokeLevel <=
4){
        updatedRoom.isSmokeActive = false;
      }

      //chekcing the CO2 level
      if (updatedRoom.co2Level <= 10 && updatedRoom.co2Level >= 5) {
        updatedRoom.isCO2Active = true;
      }else if(updatedRoom.co2Level >= 0 && updatedRoom.co2Level <= 4){
        updatedRoom.isCO2Active = false;
      }
      updatedRoom.save();
      res.sendStatus(200);
    }
    updateAlarms(room._id);

    await room.save();
    res.json(room);
  } catch (error) {}
});

//get rooms with users
router.get("/withUsers", async (req, res) => {
  try {
    let room = await Room.find({ user: { $ne: null } });
    console.log(room);
    res.send(room);
  } catch (error) {
    res.send(error);
  }
});
```

```
//alert needed rooms
router.get("/alert", async (req, res) => {
  try {
    const room = await Room.find();

    for (let i = 0; i < room.length; i++) {
      if (room[i].isCO2Active === true) {
        console.log("CO2 Active");

        if (room[i].isCO2SMSSent === false) {
          axios({
            method: "get",
            url: `http://api.liyanagegroup.com/sms_api.php?sms=Room
${room[i].roomNo}-CO2 Alarm is
Active&to=94${room[i].user.mobile}&usr=0766061689&pw=4873`,
          });
        }

        async function updateSmsStatus(id) {
          const smsRoom = await Room.findById(id);
          if (!smsRoom) {
            res.status(404).json({ msg: "Not Found" });
          }

          smsRoom.isCO2SMSSent = true;
          smsRoom.save();
        }

        updateSmsStatus(room[i]._id);
      }
      if (room[i].isSmokeActive === true) {
        console.log("Smoke Active");

        if (room[i].isSmokeSMSSent === false) {
          axios({
            method: "get",
```

```
            url: `http://api.liyanagegroup.com/sms_api.php?sms=Room
${room[i].roomNo}- Smoke Alarm is
Active&to=94${room[i].user.mobile}&usr=0766061689&pw=4873`,
          });
        }

        async function updateSmsStatus(id) {
          const smsRoom = await Room.findById(id);
          if (!smsRoom) {
            res.status(404).json({ msg: "Not Found" });
          }

          smsRoom.isSmokeSMSSent = true;
          smsRoom.save();
        }

        updateSmsStatus(room[i]._id);
      }
    }

    res.send(room);
  } catch (e) {
    console.log(e);
  }
});


//delete a room
router.delete("/deleteRoom/:roomNo",async(req,res)=>{
  try {
    console.log(req.body);
    const room = await Room.findOne({roomNo: req.params.roomNo});

    if (!room) {
      res.sendStatus(404)
    }

    await room.remove();
```

```
    res.sendStatus(200);

  } catch (error) {
    console.error(error);
    res.sendStatus(500);
  }
})


//removing user from the room and rest the room
router.put("/removeUser/:roomNo",async(req,res)=>{
  try {
    console.log(req.body);
    let room = await Room.findOne({ roomNo: req.params.roomNo });
    if (!room) return res.status(404).send("No Such Room exist");


    room.user.nic = "null";
    room.user.email = "null";
    room.user.mobile = "null";
    room.user.name = "null";
    room.user.password = "null";

    room.isAlarmActive = false;
    room.smokeLevel = 0;
    room.co2Level = 0;
    room.isCO2Active = false;
    room.isSmokeActive = false;
    room.isCO2SMSSent = false;
    room.isSmokeSMSSent = false;
    room.isMailSent = false;

    await room.save();
    res.status(200).json({"msg":"Room Reset"})
  } catch (error) {
    res.status(500).json(error)
  }
})
```

```
router.get('/getSingleRoom/:roomNo',async (req,res)=>{

  console.log(req.params)
  const room = await Room.find({roomNo:
req.params.roomNo}).select({floorNo:1,roomNo:1});

  if(!room){
    res.status(404).json({"msg":"Room Not Found"})
  }
  res.send(room);

});


router.put('/resetRoom/:roomNo/:floorNo',async (req,res)=>{
  console.log(req.params);

  try{

    const room = await Room.find({roomNo:req.params.roomNo})
    if(!room) res.status(404).json({"msg":"Invalid Room Rumber"});

    room.floorNo = req.params.floorNo;
    room.roomNo = req.params.floorNo;
    await room.save();

  }catch(err){
    res.status(500).json(err)
  }



})

router.get("/alertUserInformation", async (req, res) => {});

module.exports = router;
```

```
const express =
require("express");
                    const router = express.Router();
                    const { User } = require("../../models/User");
                    const bcrypt = require('bcryptjs');
                    const jwt = require('jsonwebtoken');
                    //getting user
                    router.get("/", async (req, res) => {
                      console.log('Api call from Dot Net Remote')
                      try {
                        const user = await User.find();
                        //res.send(user);
                       // console.log(user)
                        res.status(200).json(user)
                      } catch (e) {
                        console.log(e);
                      }
                    });



                    //registering users
                    //just to add users to no sql
                    router.post("/adduser", async (req, res) => {
                        console.log(req.body);
                        try {
                          //destructuring the req body
                          const { name, email, password,nic,
                    mobileNumber,isAdmin } = req.body;
                          //checking whether the user with same email
                    address exist
                          let user = await User.findOne({ email });
```

```
      if (user) return res.status(400).send("User
already exists");

      user = new User({
        name,
        email,
        password,
        nic,
        mobileNumber,
        isAdmin
      });
      const result = await user.save();
     res.status(200).json(result)
    } catch (e) {
      res.send(e);
    }
  });



//admin register
  router.post("/register",(req,res)=>{


    console.log(req.body);
    const adminData = {
      name:req.body.name,
      email:req.body.email,
      mobile:req.body.mobile,
      password:req.body.password
    }


    User.findOne({
      email:req.body.email
    })
    .then(user=>{
```

```
      if(!user){
        bcrypt.hash(req.body.password,10,(err,hash)=>{
          adminData.password = hash
          User.create(adminData)
          .then(user=>{
            res.sendStatus(200);
          })
          .catch(err=>{
            res.status(400).json({"error":err})
          })
        })
      }else{
        res.sendStatus(403);
      }
    })
    .catch(err=>{
      res.send(404).json({"error":err})
    })
  })



//admin login
router.post('/login',(req,res)=>{

    console.log(req.body)
    User.findOne({
      email:req.body.email
    })
    .then(user=>{
      if(user){

if(bcrypt.compareSync(req.body.password,user.password)){
          const payload={
            _id:user._id,
            name:user.name,
            email:user.email,
            mobile:user.mobile,
```

```
            }

            jwt.sign(
              payload,
              "secretkey",
              { expiresIn: "2000s" },
              (err, token) => {
                res.status(200).json({
                  'token' : token
                });
              }
            )
          }else{
            res.status(400).json({"msg":"Invalid
Password"});
          }
        }else{
          res.status(404).json({"msg":"User not Found"});
        }
      })
    })
```

.NET Remoting Platform

1) RemotingServer

Program.cs

```
TcpChannel channel = new TcpChannel(8080);


            ChannelServices.RegisterChannel(channel);

            RemotingConfiguration.RegisterWellKnownServiceType(typeof(UserServices),
"assignRoomToUser", WellKnownObjectMode.Singleton);
            RemotingConfiguration.RegisterWellKnownServiceType(typeof(UserServices),
"assignedRooms", WellKnownObjectMode.Singleton);
            RemotingConfiguration.RegisterWellKnownServiceType(typeof(UserServices),
"registerAdmin", WellKnownObjectMode.Singleton);
            RemotingConfiguration.RegisterWellKnownServiceType(typeof(UserServices),
"loginAdmin", WellKnownObjectMode.Singleton);
            RemotingConfiguration.RegisterWellKnownServiceType(typeof(UserServices),
"alertSMS", WellKnownObjectMode.Singleton);



            RemotingConfiguration.RegisterWellKnownServiceType(typeof(RoomSensorService),
"addRoom", WellKnownObjectMode.Singleton);
            RemotingConfiguration.RegisterWellKnownServiceType(typeof(RoomSensorService),
"viewRooms", WellKnownObjectMode.Singleton);
            RemotingConfiguration.RegisterWellKnownServiceType(typeof(RoomSensorService),
"searchRoom", WellKnownObjectMode.Singleton);
            RemotingConfiguration.RegisterWellKnownServiceType(typeof(RoomSensorService),
"deleteRoom", WellKnownObjectMode.Singleton);
            RemotingConfiguration.RegisterWellKnownServiceType(typeof(RoomSensorService),
"resetRoom", WellKnownObjectMode.Singleton);
            RemotingConfiguration.RegisterWellKnownServiceType(typeof(RoomSensorService),
"alert", WellKnownObjectMode.Singleton);


            Console.WriteLine("Remoting server started @ " + DateTime.Now);
            Console.ReadLine();
```

2) Remoting Client

2.1) GlobalVariables.cs

```
class GlobalVariables
{
        public static TcpChannel RegisterChannel()
        {
            TcpChannel channel = new TcpChannel();
            ChannelServices.RegisterChannel(channel);

            return channel;
        }
}
```

2.2) UserAssign.cs

```
public partial class UserForm : Form
    {
        IFireAlarmService.IUsersService client;
        TcpChannel channel;

        private static TimerCallback fillData;

        public UserForm()
        {
            InitializeComponent();


            channel = GlobalVariables.RegisterChannel();

           client = (IFireAlarmService.IUsersService)Activator.GetObject
           (typeof(IFireAlarmService.IUsersService),
"tcp://localhost:8080/assignRoomToUser");

            client = (IFireAlarmService.IUsersService)Activator.GetObject
                   (typeof(IFireAlarmService.IUsersService),
"tcp://localhost:8080/assignedRooms");

            client = (IFireAlarmService.IUsersService)Activator.GetObject
                   (typeof(IFireAlarmService.IUsersService),
"tcp://localhost:8080/alertSMS");
        }

        private void btnAssignUser_Click(object sender, EventArgs e)
        {

            int i = client.assignRoomToUser(txtUsername.Text, txtEmail.Text,
txtMobile.Text, txtNic.Text, "123456",Convert.ToInt32(txtRoomNo.Text.ToString()));
            if (i == 200)
            {
                MessageBox.Show("User Added to the Room Successfully!!!");
                txtUsername.Text = "";
                txtEmail.Text = "";
                txtMobile.Text = "";
                txtNic.Text = "";
                txtRoomNo.Text = "";
            }
            else
            {
                MessageBox.Show("Oops!!! Something went Wrong...");
            }
        }

        internal  void Run()
```

```
        {
            var autoEvent = new AutoResetEvent(false);
            int seconds = 15 * 1000;
            var timer = new System.Threading.Timer(FillData, autoEvent, 1000, seconds);
        }


        private void FillData(object state)
        {
            _ = client.alertSMS();
            string roomStatus, co2AlarmStatus, smokeAlarmStatus;


            IEnumerable<Usermodel> roomList = client.assignedRooms();
            dataGridView1.Rows.Clear();
            foreach (var row in roomList.ToList())
            {
                if (row.IsAlarmActive == true) roomStatus = "User In";
                else roomStatus = "User Free";

                if (row.IsCO2Active == true) co2AlarmStatus = "ON";
                else co2AlarmStatus = "OFF";

                if (row.IsSmokeActive == true) smokeAlarmStatus = "ON";
                else smokeAlarmStatus = "OFF";


                string[] userDataArray = { row.RoomNo.ToString(), row.FloorNo.ToString()+
" th Floor", row.Co2Level.ToString()+ " ml", row.SmokeLevel.ToString()+ "
ml",co2AlarmStatus,smokeAlarmStatus,roomStatus };
                dataGridView1.Rows.Add(userDataArray);


            }
        }


        private void btnreset_Click(object sender, EventArgs e)
        {
            channel.StopListening(null);
            RemotingServices.Disconnect(this);
            ChannelServices.UnregisterChannel(channel);
            channel = null;

            AddRooms Check = new AddRooms();
            Check.Show();
            this.Hide();
        }

        private void UserForm_Load(object sender, EventArgs e)
        {
```

```
            dataGridView1.ColumnCount = 7;
            dataGridView1.Columns[0].Name = "RoomNo";
            dataGridView1.Columns[1].Name = "FloorNo";
            dataGridView1.Columns[2].Name = "CO2 Level";
            dataGridView1.Columns[3].Name = "Smoke Level";
            dataGridView1.Columns[4].Name = "CO2 Sensor";
            dataGridView1.Columns[5].Name = "Smoke Sensor";
            dataGridView1.Columns[6].Name = "Room Status";

            Run();

        }

        private void dataGridView1_CellContentClick(object sender,
DataGridViewCellEventArgs e)
        {

        }
    }
```

2.3) AddRooms.cs

```
    public partial class AddRooms : Form
    {
        IFireAlarmService.IRoomSensorService client;
        TcpChannel channel;
        public AddRooms()
        {
            InitializeComponent();
            channel = GlobalVariables.RegisterChannel();

            client = (IFireAlarmService.IRoomSensorService)Activator.GetObject
             (typeof(IFireAlarmService.IRoomSensorService),
"tcp://localhost:8080/addRoom");

            client = (IFireAlarmService.IRoomSensorService)Activator.GetObject
                (typeof(IFireAlarmService.IRoomSensorService),
"tcp://localhost:8080/viewRooms");


            client = (IFireAlarmService.IRoomSensorService)Activator.GetObject
                (typeof(IFireAlarmService.IRoomSensorService),
"tcp://localhost:8080/searchRoom");


            client = (IFireAlarmService.IRoomSensorService)Activator.GetObject
                (typeof(IFireAlarmService.IRoomSensorService),
"tcp://localhost:8080/deleteRoom");


            client = (IFireAlarmService.IRoomSensorService)Activator.GetObject
                (typeof(IFireAlarmService.IRoomSensorService),
"tcp://localhost:8080/resetRoom");

            client = (IFireAlarmService.IRoomSensorService)Activator.GetObject
                (typeof(IFireAlarmService.IRoomSensorService),
"tcp://localhost:8080/alert");
        }

        private void btnadd_Click(object sender, EventArgs e)
        {
            int i = client.addRoom(Convert.ToInt32(txtRoomNo.Text.ToString()),
Convert.ToInt32(txtFloorNo.Text.ToString()),0,0);

            if (i == 200)
            {
                MessageBox.Show("Room added Successfully!!!");
                txtRoomNo.Text = "";
                txtFloorNo.Text = "";
            }
```

```
            else
            {
                MessageBox.Show("Room Already Exists...");
            }
        }

        private void btnUserForm_Click(object sender, EventArgs e)
        {
            channel.StopListening(null);
            RemotingServices.Disconnect(this);
            ChannelServices.UnregisterChannel(channel);
            channel = null;

            UserForm Check = new UserForm();
            Check.Show();
            this.Hide();
        }

        internal void Run()
        {
            var autoEvent = new AutoResetEvent(false);
            int seconds = 15 * 1000;
            var timer = new System.Threading.Timer(FillData, autoEvent, 1000, seconds);
        }


        private void FillData(object state)
        {
            string roomStatus;
            _ = client.alert();

            IEnumerable<Usermodel> roomList = client.viewRooms();
            dataGridView1.Rows.Clear();


            foreach (var row in roomList.ToList())
            {
              dataGridView1.FirstDisplayedScrollingRowIndex = dataGridView1.RowCount - 1;
              if (row.IsAlarmActive == true)
              {
                roomStatus = "User In";
              }
              else
              {
                roomStatus = "User Free";
              }
               string[] userDataArray = { row.RoomNo.ToString(), row.FloorNo.ToString() +
" th Floor", row.Co2Level.ToString() + " ml", row.SmokeLevel.ToString() + " ml" ,
roomStatus };
                dataGridView1.Rows.Add(userDataArray);
```

```
            }
        }


        private void AddRooms_Load(object sender, EventArgs e)
        {
            dataGridView1.ColumnCount = 5;
            dataGridView1.Columns[0].Name = "RoomNo";
            dataGridView1.Columns[1].Name = "FloorNo";
            dataGridView1.Columns[2].Name = "CO2 Level";
            dataGridView1.Columns[3].Name = "Smoke Level";
            dataGridView1.Columns[4].Name = "Status";
            dataGridView1.ScrollBars = ScrollBars.Both;
            Run();
        }

        private void label6_Click(object sender, EventArgs e)
        {

        }

        private void btnSearch_Click(object sender, EventArgs e)
        {
            IEnumerable<Usermodel> singleRoom =
client.searchRoom(Convert.ToInt32(txtSearch.Text.ToString()));

            if(singleRoom.ToList().Count == 1)
            {
                foreach (var row in singleRoom.ToList())
                {
                    txtRoomNo.Text = row.RoomNo.ToString();
                    txtFloorNo.Text = row.FloorNo.ToString();
                }
            }
            else
            {
                MessageBox.Show("Room Not Found");
            }

        }

        private void btnDelete_Click(object sender, EventArgs e)
        {
            int i = client.deleteRoom(Convert.ToInt32(txtRoomNo.Text.ToString()));
            if(i == 200)
            {
                MessageBox.Show("Room Deleted Successfully");
                txtSearch.Text = "";
                txtRoomNo.Text = "";
                txtFloorNo.Text = "";
            }
```

```
            else if(i == 404)
            {
                MessageBox.Show("Room Not Found");
            }
            else
            {
                MessageBox.Show("Oops!!! Something went Wrong");
            }
        }

        private void btnReset_Click(object sender, EventArgs e)
        {
            int i = client.resetRoom(Convert.ToInt32(txtRoomNo.Text.ToString()),
Convert.ToInt32(txtFloorNo.Text.ToString()));
            if(i == 200)
            {
                MessageBox.Show("Room Reset is Successfuly Done");
                txtSearch.Text = "";
                txtRoomNo.Text = "";
                txtFloorNo.Text = "";


            }else if(i == 404)
            {
                MessageBox.Show("Invalid Room Number!!!");
            }
            else
            {
                MessageBox.Show("Oops!!! Something Went Wrong!!!");
            }
        }
    }
```

2.4) Admin.cs

```
public partial class Admin : Form
    {

        IFireAlarmService.IUsersService client;
        TcpChannel channel;
        public Admin()
        {
            InitializeComponent();
            channel = GlobalVariables.RegisterChannel();

            client = (IFireAlarmService.IUsersService)Activator.GetObject
             (typeof(IFireAlarmService.IUsersService),
"tcp://localhost:8080/registerAdmin");

            client = (IFireAlarmService.IUsersService)Activator.GetObject
             (typeof(IFireAlarmService.IUsersService),
"tcp://localhost:8080/loginAdmin");
        }

        private void btnRegisterUser_Click(object sender, EventArgs e)
        {
            //string name,string email,string mobile,string password
            int i = client.registerAdmin(txtUsername.Text, txtEmail.Text, txtMobile.Text,
txtPassword.Text);
            if(i == 200)
            {
                MessageBox.Show("Registered Successfully");
                txtUsername.Text = "";
                txtMobile.Text = "";
                txtEmail.Text = "";
                txtPassword.Text = "";
            }
            else if (i == 403)
            {
                MessageBox.Show("User Email Already Registered");
            }
            else
            {
                MessageBox.Show("Oops!!! Something wet wrong");
            }
        }

        private void btnLogin_Click(object sender, EventArgs e)
        {
            int i = client.loginAdmin(txtEmailLogin.Text, txtPasswordLogin.Text);
            if(i == 200)
            {
                channel.StopListening(null);
```

```
                RemotingServices.Disconnect(this);
                ChannelServices.UnregisterChannel(channel);
                channel = null;

                UserForm Check = new UserForm();
                Check.Show();
                this.Hide();
            }
            else if(i == 400)
            {
                MessageBox.Show("Invalid Password!!!");
            }
            else if(i == 404)
            {
                MessageBox.Show("User Does Not Exsist in the System");
            }
        }

        private void txtPasswordLogin_TextChanged(object sender, EventArgs e)
        {
            txtPasswordLogin.PasswordChar = '*';

        }

        private void txtPassword_TextChanged(object sender, EventArgs e)
        {
            txtPassword.PasswordChar = '*';

        }
    }
```

3) IFireAlarmService

3.1.1) IUserService

```
    public interface IUsersService
    {
       int assignRoomToUser(string username, string email, string mobile, string nic,
string password, int roomNo);
       IEnumerable<Usermodel> assignedRooms();
       int registerAdmin(string name, string email, string mobile, string password);
       int loginAdmin(string email, string password);
       IEnumerable<UserModel> alertSMS();
    }
```

3.1.2) IRoomSensorService

```
    public interface IRoomSensorService
    {
        int addRoom(int roomNo, int floorNo, int smokeLevel, int co2Level);
        IEnumerable<Usermodel> viewRooms();
        IEnumerable<Usermodel> searchRoom(int roomNo);
        int resetRoom(int roomNo, int floorNo);
        int deleteRoom(int roomNo);
        IEnumerable<Usermodel> alert();

    }
```

3.2.1) Models/UserModel.cs

```
    [Serializable]
    public class UserModel
    {
        private String name;
        private String nic;
        private String email;
        private String password;
        private String mobile;
        public UserModel(String name, String nic, String email, String password, String
mobile)
        {
            this.name = name;
            this.nic = nic;
            this.email = email;
            this.password = password;
            this.mobile= mobile;
        }

        public string Name
        {
            get { return name; }
            set { name = value; }
        }

        public string Nic
        {
            get { return nic; }
            set { nic = value; }
        }

        public string Email
        {
            get { return email; }
            set { email = value; }
        }

        public string Password
        {
            get { return password; }
            set { password = value; }
        }

        public string Mobile
        {
            get { return mobile; }
            set { mobile = value; }
        }
    }
```

3.2.2) Models/RoomsModel.cs

```
    [Serializable]
    public class Usermodel
    {
        private int roomNo;
        private int floorNo;
        private int smokeLevel;
        private int co2Level;
        private Boolean isAlarmActive;
        private Boolean isCO2Active;
        private Boolean isSmokeActive;


        public Usermodel(int roomNo, int floorNo, int smokeLevel, int co2Level, Boolean
isAlarmActive)
        {
            this.roomNo = roomNo;
            this.floorNo = floorNo;
            this.smokeLevel = smokeLevel;
            this.co2Level = co2Level;
            this.isAlarmActive = isAlarmActive;
        }

        public Usermodel(int roomNo,int floorNo)
        {
            this.roomNo = roomNo;
            this.floorNo = floorNo;
        }


        public int RoomNo
        {
            get { return roomNo; }
            set { roomNo = value; }
        }

        public int FloorNo
        {
            get { return floorNo; }
            set { floorNo = value; }
        }

        public int SmokeLevel
        {
            get { return smokeLevel; }
            set { smokeLevel = value; }
        }
```

```
        public int Co2Level
        {
            get { return co2Level; }
            set { co2Level = value; }
        }

        public Boolean IsAlarmActive
        {
            get { return isAlarmActive; }
            set { isAlarmActive = value; }
        }

        public Boolean IsCO2Active
        {
            get { return isCO2Active; }
            set { isCO2Active = value; }
        }

        public Boolean IsSmokeActive
        {
            get { return isSmokeActive; }
            set { isSmokeActive = value; }
        }


    }
```

3.2.3) Models/AdminModel.cs

```
[Serializable]
    public class AdminModel
    {

        private String name;
        private String email;
        private String mobile;
        private String password;


        public AdminModel(string name,string email,string mobile,string password)
        {
            this.name = name;
            this.email = email;
            this.mobile = mobile;
            this.password = password;
        }

        public AdminModel(string email,string password)
        {
            this.email = email;
            this.password = password;
        }
        public String Name
        {
            get { return name; }
            set { name = value; }
        }
        public String Email
        {
            get { return email; }
            set { email = value; }
        }
        public String Mobile
        {
            get { return mobile; }
            set { mobile = value; }
        }
        public String Password
        {
            get { return password; }
            set { password = value; }
        }
    }
```

4) FireAlarmService

4.1) GlobalVariables.cs

```
public static class GlobalVariables
    {
        public static HttpClient WebApiClient = new HttpClient();
        static GlobalVariables()
        {
            WebApiClient.BaseAddress = new Uri("http://localhost:5000/api/");
            WebApiClient.DefaultRequestHeaders.Clear();
              WebApiClient.DefaultRequestHeaders.Accept.Add(new
       MediaTypeWithQualityHeaderValue("application/json"));
        }
    }
```

4.2) UserService

```
public class UserServices : MarshalByRefObject, IFireAlarmService.IUsersService
    {

        public int assignRoomToUser(string username, string email, string mobile, string
nic, string password,int roomNo)
        {
            UserModel userModel = new UserModel(username, nic, email, password, mobile);
            HttpResponseMessage response =
GlobalVariables.WebApiClient.PutAsJsonAsync("room/addCustomer/" + roomNo,
userModel).Result;
            return Convert.ToInt32(response.StatusCode);
        }

        public IEnumerable<Usermodel> assignedRooms()
        {
            IEnumerable<Usermodel> roomList;
            HttpResponseMessage response =
GlobalVariables.WebApiClient.GetAsync("room/").Result;
            roomList = response.Content.ReadAsAsync<IEnumerable<Usermodel>>().Result;
            return roomList;
        }

        public int registerAdmin(string name, string email, string mobile, string
password)
        {
            AdminModel adminModel = new AdminModel(name, email, mobile, password);
            HttpResponseMessage response =
GlobalVariables.WebApiClient.PostAsJsonAsync("users/register",adminModel).Result;
            return Convert.ToInt32(response.StatusCode);
        }

        public int loginAdmin(string email, string password)
        {
            AdminModel adminModel = new AdminModel(email,password);
            HttpResponseMessage response =
GlobalVariables.WebApiClient.PostAsJsonAsync("users/login",adminModel).Result;
            return Convert.ToInt32(response.StatusCode);
        }

        public IEnumerable<UserModel> alertSMS()
        {
            IEnumerable<UserModel> alert;
            HttpResponseMessage response =
GlobalVariables.WebApiClient.GetAsync("room/alert/").Result;
            alert = response.Content.ReadAsAsync<IEnumerable<UserModel>>().Result;
            return alert;
        }
    }
```

4.3) RoomService

```
public class RoomSensorService : MarshalByRefObject, IFireAlarmService.IRoomSensorService
    {
        public int addRoom(int roomNo, int floorNo, int smokeLevel, int co2Level)
        {
            Usermodel roomsModel = new
Models.Usermodel(roomNo,floorNo,smokeLevel,co2Level,false);
            HttpResponseMessage response =
GlobalVariables.WebApiClient.PostAsJsonAsync("room/addroom", roomsModel).Result;
            return Convert.ToInt32(response.StatusCode);
        }


        public IEnumerable<Usermodel> viewRooms()
        {
            IEnumerable<Usermodel> roomList;
            HttpResponseMessage response =
GlobalVariables.WebApiClient.GetAsync("room/").Result;
            roomList = response.Content.ReadAsAsync<IEnumerable<Usermodel>>().Result;
            return roomList;
        }

        public int deleteRoom(int roomNo)
        {
            HttpResponseMessage response =
GlobalVariables.WebApiClient.DeleteAsync("room/deleteRoom/" + roomNo).Result;
            return Convert.ToInt32(response.StatusCode);
        }

        public IEnumerable<Usermodel> searchRoom(int roomNo)
        {
            IEnumerable<Usermodel> singleRoom;
            HttpResponseMessage response =
GlobalVariables.WebApiClient.GetAsync("room/getSingleRoom/" + roomNo).Result;
            singleRoom = response.Content.ReadAsAsync<IEnumerable<Usermodel>>().Result;
            return singleRoom;
        }

        public int resetRoom(int roomNo, int floorNo)
        {
            Usermodel roomsModel = new Usermodel(roomNo,floorNo);
            HttpResponseMessage response =
GlobalVariables.WebApiClient.PutAsJsonAsync("room/removeUser/" + roomNo,
roomsModel).Result;
            return Convert.ToInt32(response.StatusCode);
        }

        public IEnumerable<Usermodel> alert()
        {
```

```
            IEnumerable<Usermodel> alert;
            HttpResponseMessage response =
GlobalVariables.WebApiClient.GetAsync("room/alert/").Result;
            alert = response.Content.ReadAsAsync<IEnumerable<Usermodel>>().Result;
            return alert;
        }


    }
```

Assignment II – REST API Project

SE3020 – Distributed Systems **Semester 1**

DummySensor App

1) GlobalVariables

```
public static class GlobalVariables
{
    public static HttpClient WebApiClient = new HttpClient();
    static GlobalVariables()
    {
        WebApiClient.BaseAddress = new Uri("http://localhost:5000/api/");
        WebApiClient.DefaultRequestHeaders.Clear();
        WebApiClient.DefaultRequestHeaders.Accept.Add(new
MediaTypeWithQualityHeaderValue("application/json"));
    }
}
```

2) SensoDataModel

```
[Serializable]
    public class SensorDataModel
    {
        private int roomNo;
        private int smokeLevel;
        private int co2Level;


        public SensorDataModel(int roomNo, int smokeLevel, int co2Level)
        {
            this.roomNo = roomNo;
            this.smokeLevel = smokeLevel;
            this.co2Level = co2Level;
        }

        public int RoomNo
        {
            get { return roomNo; }
            set { roomNo = value; }
        }

        public int SmokeLevel
        {
            get { return smokeLevel; }
            set { smokeLevel = value; }
        }

        public int Co2Level
        {
            get { return co2Level; }
            set { co2Level = value; }
        }
    }
```

3) Form1.cs (Dummy Interface)

```
4)  public partial class Form1 : Form
5)      {
6)          public Form1()
7)          {
8)              InitializeComponent();
9)          }
10)
11)         private void button1_Click(object sender, EventArgs e)
12)         {
13)             int roomNO = Convert.ToInt32(txtRoomNo.Text.ToString());
14)             int count = 1;
15)             int smoke = 0, co2 = 10;
16)
17)
18)             for (int i = 1; i <= 10; i++)
19)             {
20)                 smoke++;
21)                 co2--;
22)
23)                 lblCO2.Text = co2.ToString();
24)                 lblSmoke.Text = smoke.ToString();
25)
26)                 SensorDataModel sensorDataModel = new
    SensorDataModel(roomNO,smoke,co2);
27)                 HttpResponseMessage response =
    GlobalVariables.WebApiClient.PutAsJsonAsync("room/addSensor/"+sensorDataModel.Room
    No,sensorDataModel).Result;
28)                 Thread.Sleep(10000);
29)
30)             }
31)
32)
33)
34)         }
        }
```