Distributed Systems - SE3020

Alarm Monitoring System

Assignment 2

Group Members

IT18130508 – Sasmitha N. U. A

IT18007534 – Guruge P. P. L

IT18131048 – Silva W. J. T

IT18140330 – Gamage O. M.

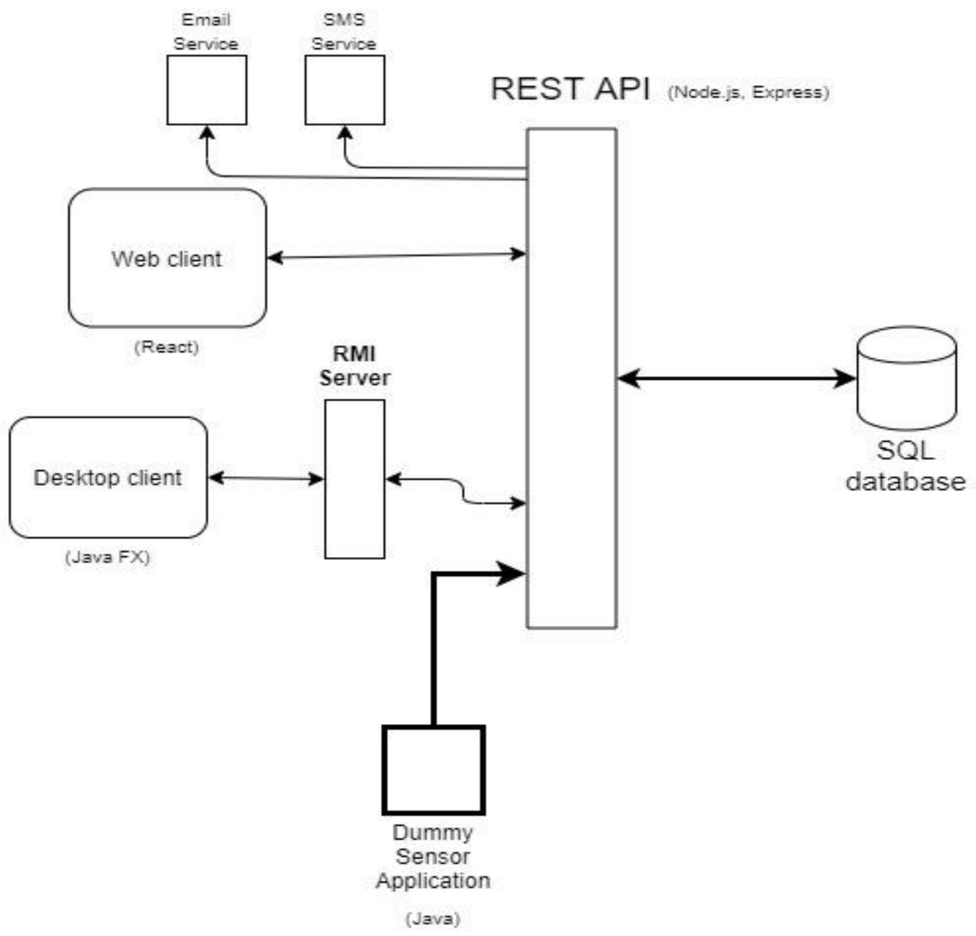
1. Introduction

This is a system developed for managing and monitoring fire alarms in a building. Users can add edit sensors and there is a dashboard for monitoring the overall status. This system consists of two end applications one is a web application and the other is a desktop application. The web client is developed using React since we do not have actual sensors to get data from, we created a dummy sensor app using java to periodically send data to database, backend server is developed using node.js and express, the desktop client is developed using java FX. The desktop client backend is an RMI java server.
(https://www.youtube.com/watch?v=vaaRTZtoQew&feature=youtu.be)

Github links,

Dummy Sensor App - https://github.com/amoda-sasmitha/Sensor-Dummy-App
RMI Server - https://github.com/amoda-sasmitha/RMI-Server
Desktop Client - https://github.com/amoda-sasmitha/SesorDesktopClient
Web Client - https://github.com/amoda-sasmitha/Monitor-System
Node Backend - https://github.com/amoda-sasmitha/alarmMonitoringSystemBackend

2. High-Level Architecture



This is the high-level architecture of the developed system. As shown in the diagram sensor is directly connected to the node server, like the database. The web client is connected to the node server through RMI server, and the web client is connected directly to the node server. Both email service and SMS service are connected to the node server so that users can send emails or text notifications through a web or desktop client.

3. Sequence Diagrams

● Admin Register

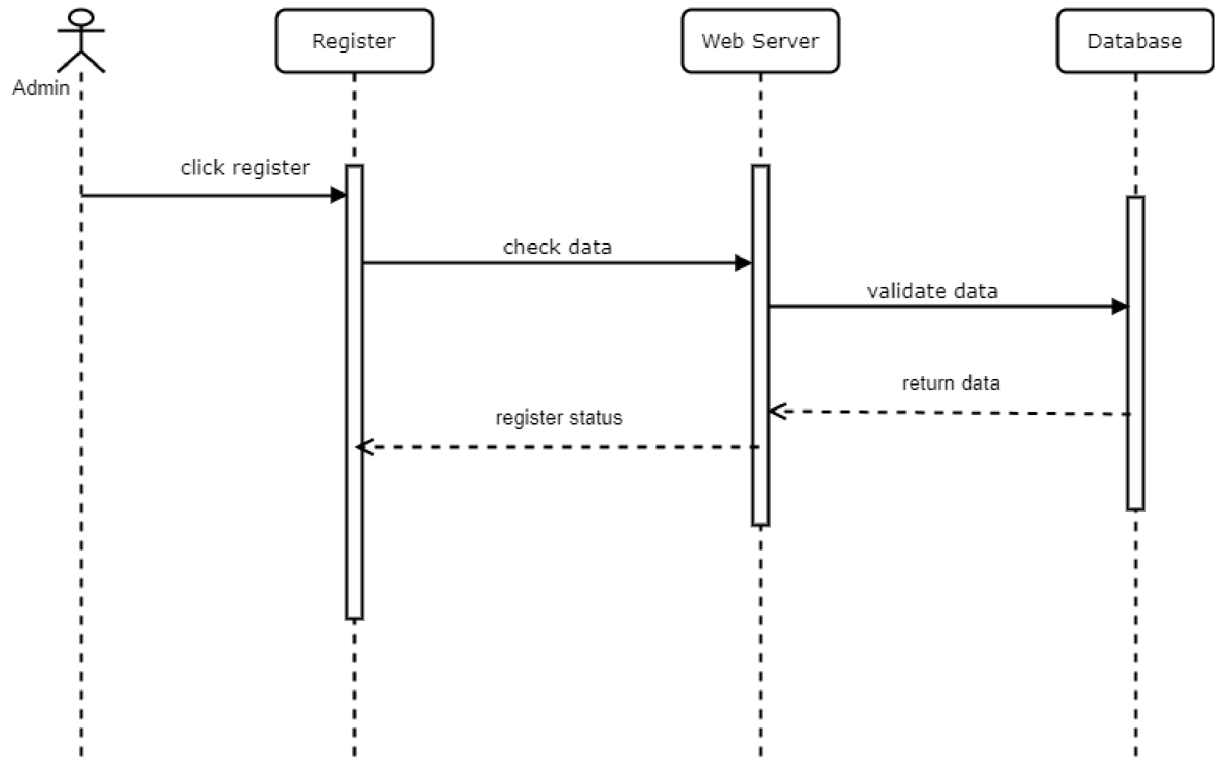


Figure 1 (Sequence Diagram – Admin Register)

In the admin registration process, the super admin can add admins to the system.
In the registration process super admin should input details of the admin who is going to register in the system and password automatically generated. After clicking the register button data goes to the database and validates if any of the users exist with these details, if these details are already in the database sent response (401) and if it is not in the database sent response (200). If the response is 200 then auto-generated password sent to the admin’s email.

● Admin Login

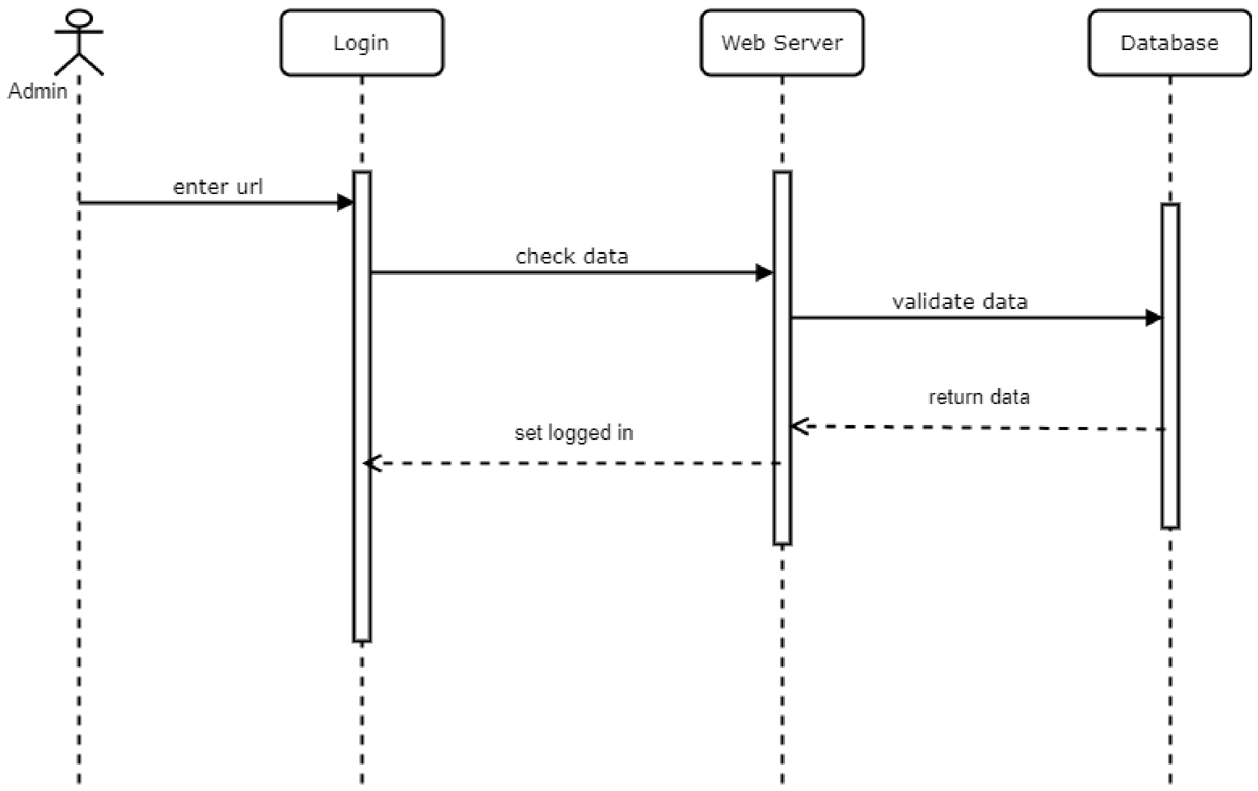


Figure 2 (Sequence Diagram – Admin Login)

In order to interact with the system first, the user needs to log in. this diagram shows how the user login will be validated and the user will be granted to access the system.

- Admin Remove

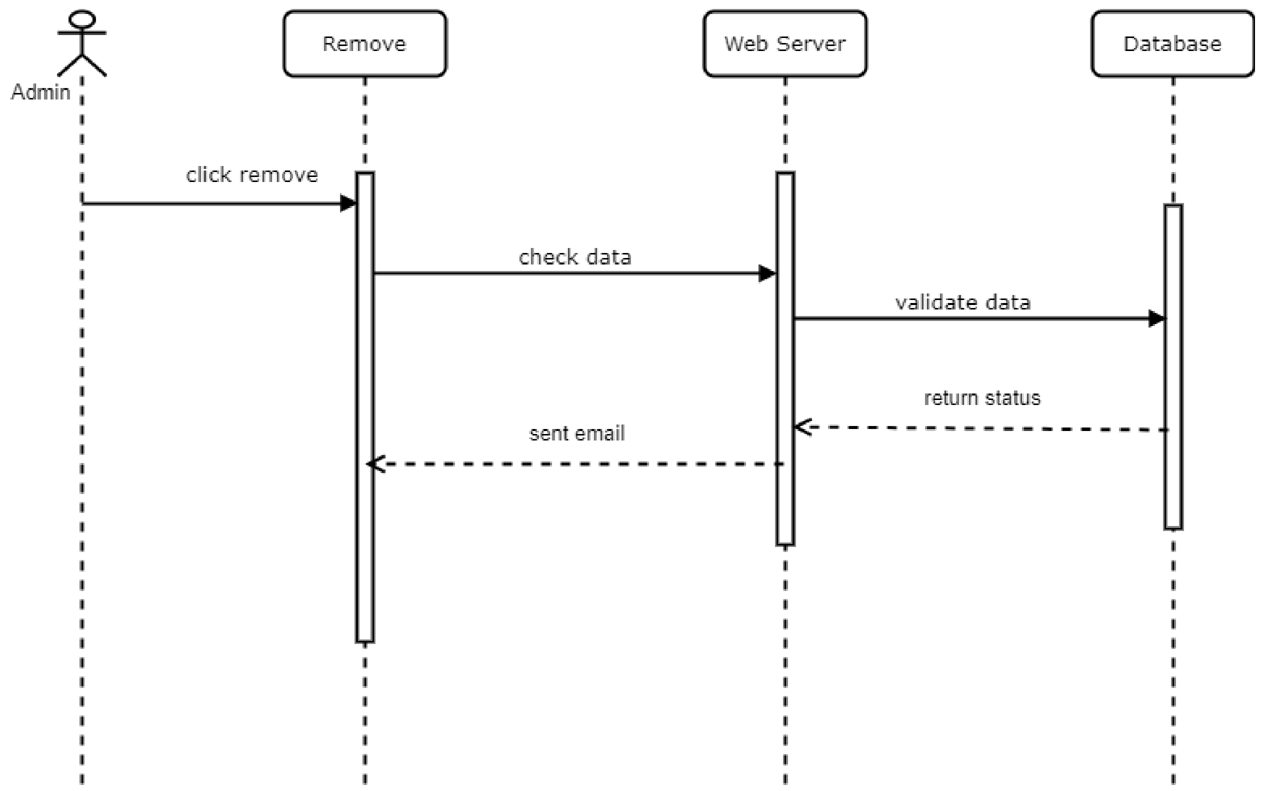


Figure 3 (Sequence Diagram – Admin Remove)

This diagram shows how to remove an admin from the database. Upon the delete, the user will be notified over an email.

- Add Sensor

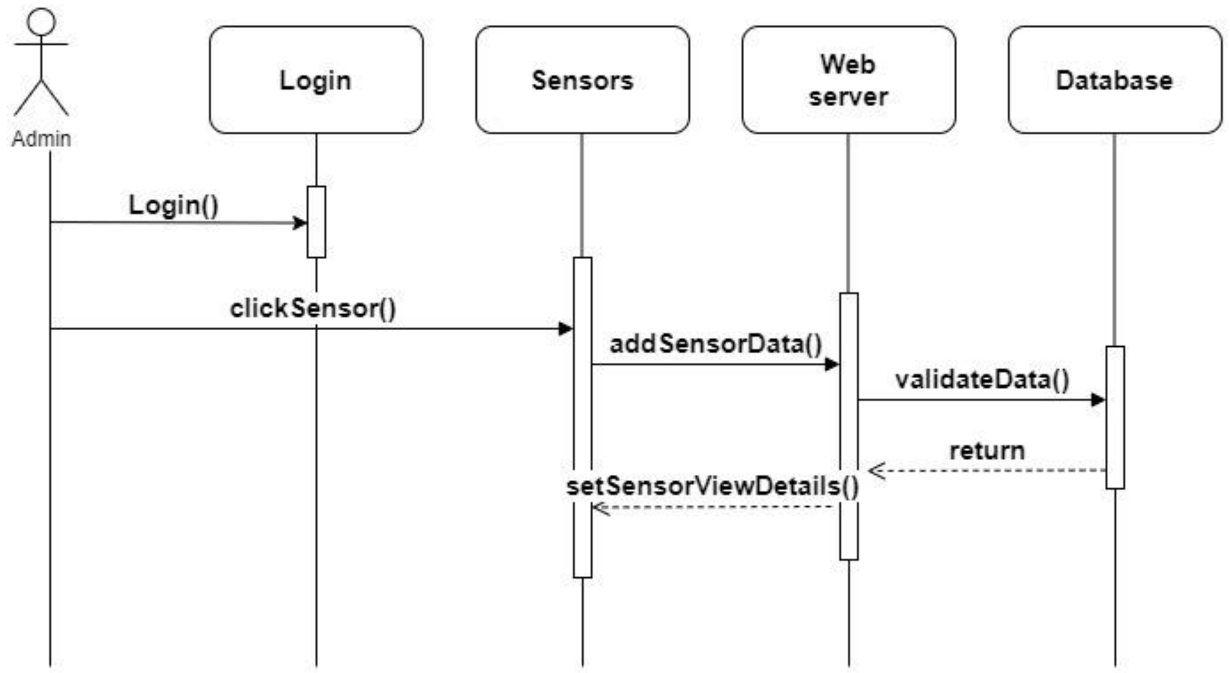


Figure 4 (Sequence Diagram – Admin add sensor)

Admin can add new sensors, in order to do that admin first needs to log in to the system. After successfully adding the new sensor it will appear on the dashboard alongside the previously added sensors. And all the values will be changed accordingly such as average co2 and smoke levels.

- View Sensor Details

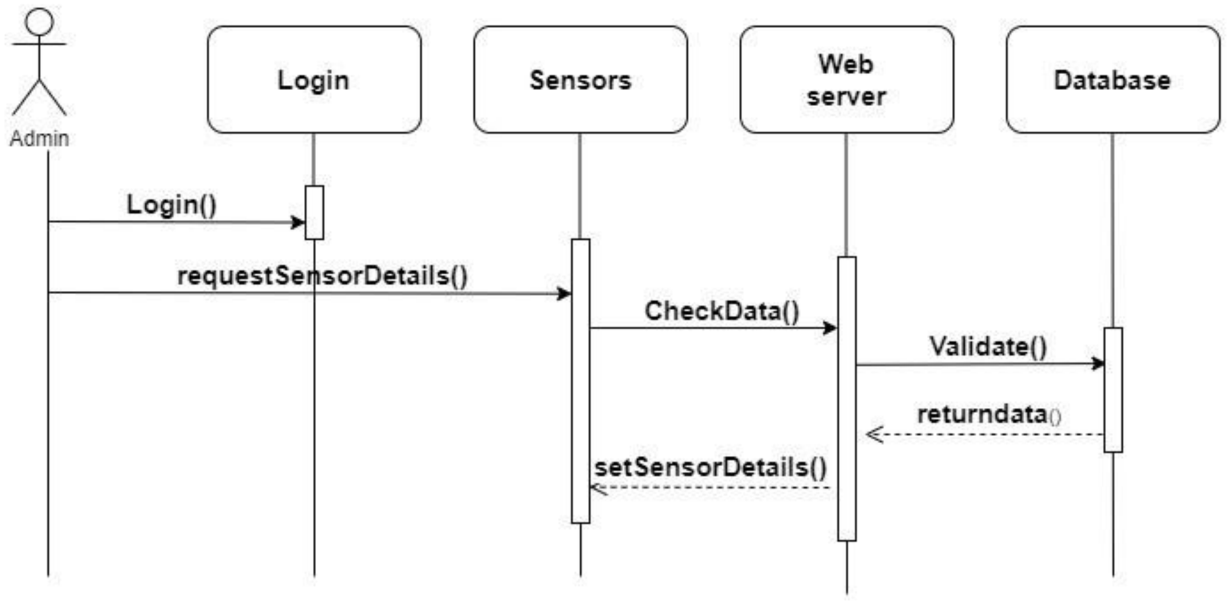
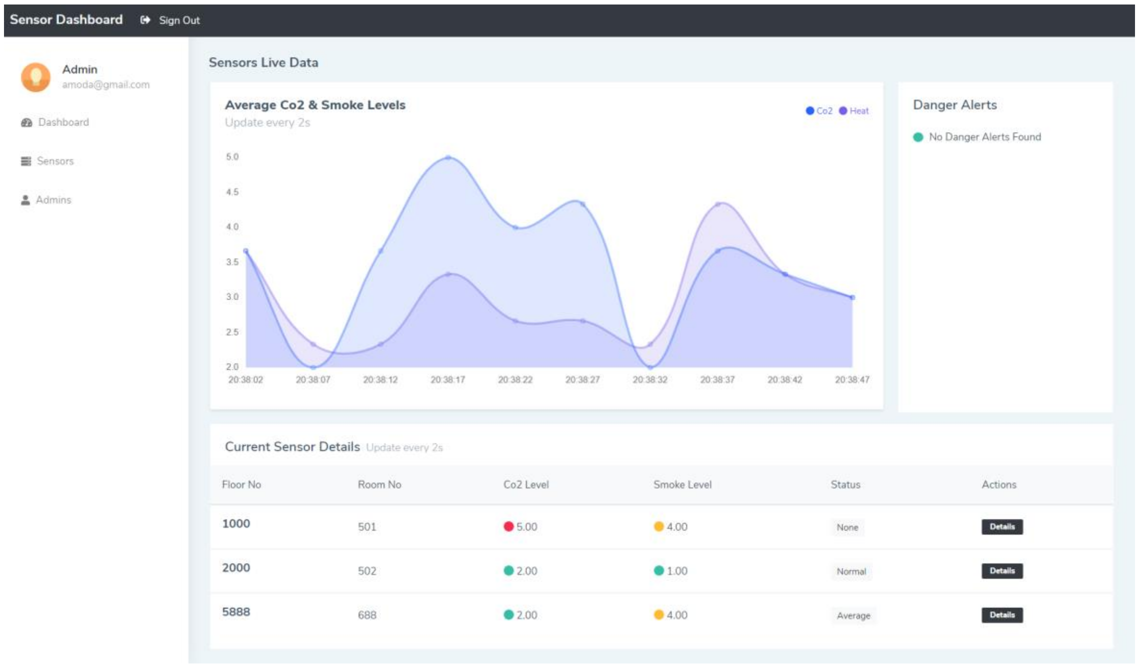


Figure 5 (Sequence Diagram – Admin View Sensor Details)

This function is to view sensor details to the admin. Admin must log in to the system to view sensor details. It will show how much CO2 and how much heat the particular sensor is reading; users can also see the graph changing in real-time.

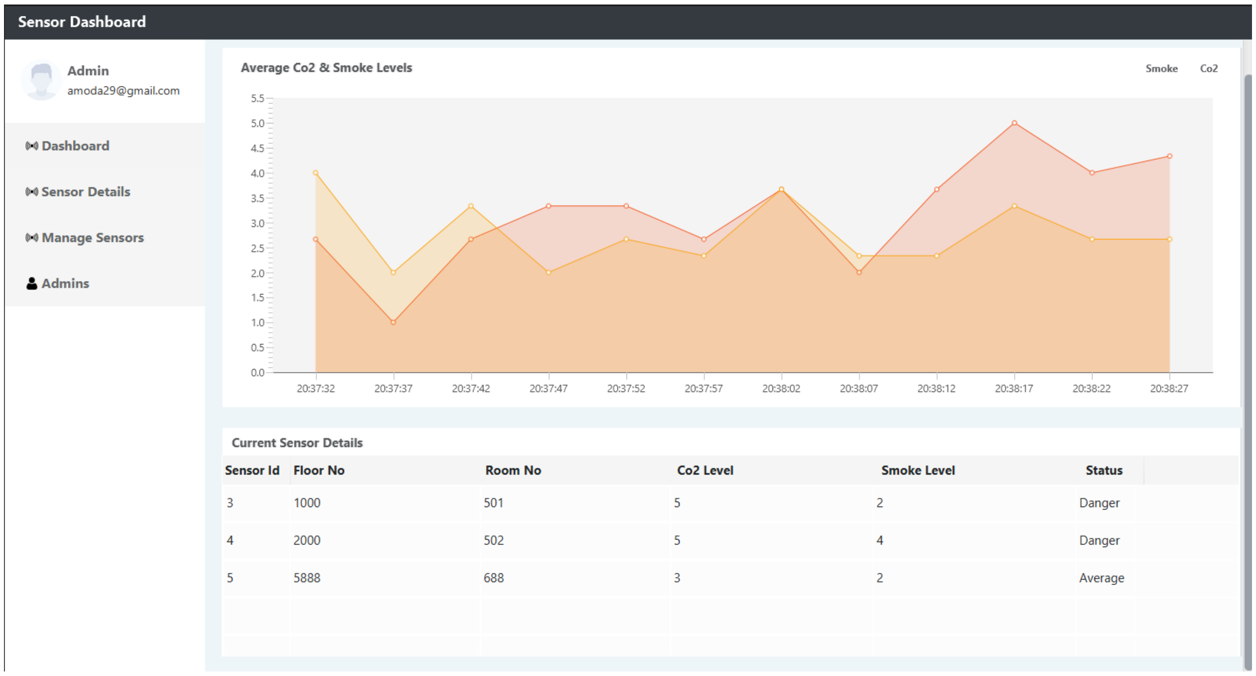
4. User Interface Screenshots



Current Sensor Details Update every 2s

Floor No	Room No	Co2 Level	Smoke Level	Status	Actions
1000	501	● 5.00	● 4.00	None	Details
2000	502	● 2.00	● 1.00	Normal	Details
5888	688	● 2.00	● 4.00	Average	Details

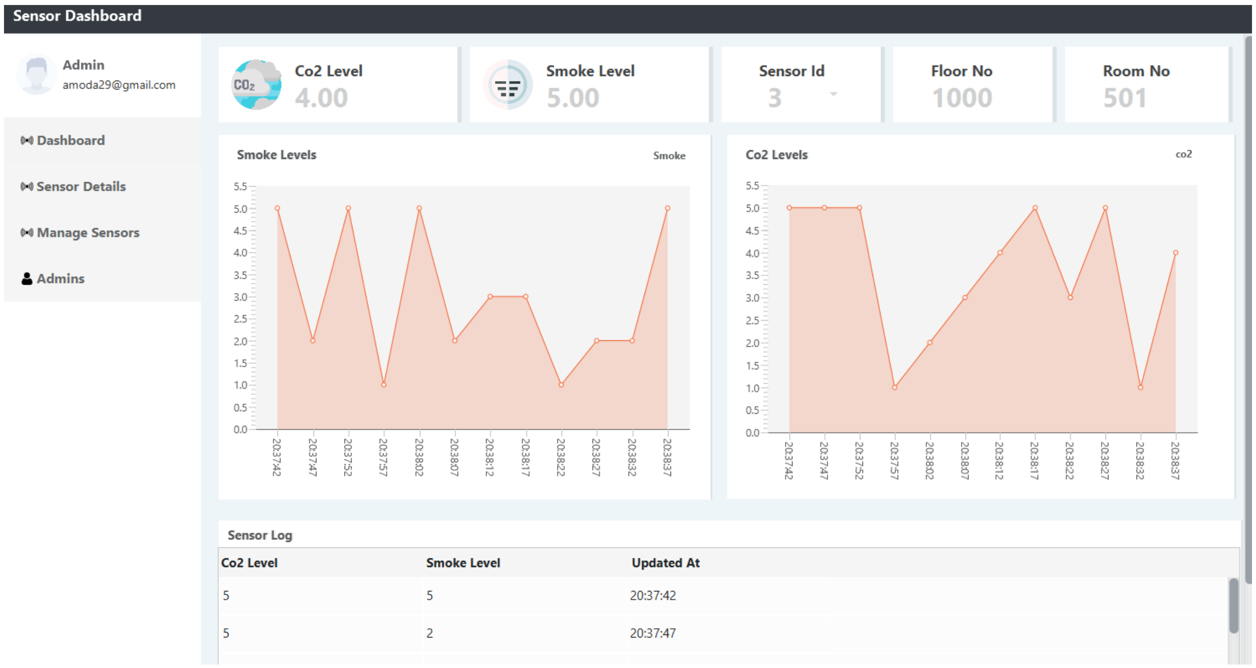
- This is the main dashboard of the web client, as you can see it shows the overall result in a graph, below that there is the individual summarized status of the sensors.



Current Sensor Details

Sensor Id	Floor No	Room No	Co2 Level	Smoke Level	Status
3	1000	501	5	2	Danger
4	2000	502	5	4	Danger
5	5888	688	3	2	Average

- This is the main page of the desktop client, it shows the overall result of sensors in the building just like in the web client. Below the main chart, this also shows the individual status of sensors.



Co2 Level	Smoke Level	Sensor Id	Floor No	Room No
4.00	5.00	3	1000	501

Sensor Log

Co2 Level	Smoke Level	Updated At
5	5	20:37:42
5	2	20:37:47

- When the user clicks on a particular sensor this window will show up. This shows the smoke level and co2 level that the sensor detected over time. It is available in both charts and raw data as a table at the bottom. It refreshes every five seconds.

5. System Description

The whole system consists of web applications and a desktop application. Users can add, edit, and delete sensors. At both web application dashboard and desktop application dashboard, users can see the overall co2 percentage and heat level of the building, below that user can see individual sensors. After clicking on one of them, the user will be directed to the individual information page for that sensor. If any sensor goes over the expected level the user will be notified over a text.
If the user decides to add a new admin. An email would be sent to their email with a suggested password and the user can proceed to register as an admin.
We've built a dummy sensor application using java to constantly send data to the database just like a real sensor would. REST API will take data from it and pass it to both web and desktop clients.

6. Appendix

Web Client

Frontend (React Js)

Admin Login - View

```
import React, { Component } from 'react';
// import top bar
import Topbar from '../../components/Topbar';

// import controll file
import A_Admin from '../../Controllers/Admin'

// imporrt css
import './login.css'

class Login extends Component {
    constructor() {
        super();
        this.state = {
            uEmail: '',
            uPass: ''

        };

    }

    // ---------------------------------------- functions ----------------
------------------------
    // ---------------------------------------- functions ----------------
------------------------
    // ---------------------------------------- functions ----------------
------------------------

    // user email
    onChangeEmail(e) {
        this.setState({
            uEmail: e.target.value
        })
    }
    // user passsword
    onChangePassword(e) {
        this.setState({
            uPass: e.target.value
        })
    }


    // sign in

    async OnSignIn(e) {
        e.preventDefault();

        const SignIn = {
            uEmail: this.state.uEmail,
            uPass: this.state.uPass
        }

        var signStatus = await A_Admin.adminSign(SignIn)
        await console.log(signStatus);

        switch (signStatus.status) {
            case 200:
                const SignedInUser = {
                    status: signStatus.data.status,
                    id: signStatus.data.user.id,
                    name: signStatus.data.user.name,
                    email: signStatus.data.user.email,
                    phone: signStatus.data.user.phone
                }
                A_Admin.setCookies(SignedInUser.status, SignedInUser.id,
SignedInUser.name, SignedInUser.email, SignedInUser.phone);
                window.location.replace("/dashboard");
                break;
            default:
                window.location.replace("/");
        }


    }


    // ---------------------------------------- render functions ---------
-------------------------------
    // ---------------------------------------- render functions ---------
-------------------------------
    // ---------------------------------------- render functions ---------
-------------------------------
    render() {
        return (
            <>
            <div className="wrapper hv-100 w-100 bg-darks" >
                <div className="container bg-light vh-100">
                    <div className="row mt-5">
                        <div className="col-sm-8 col-md-5  mx-auto mt-5">
                            <div className="card card-signin my-5
shadow">
                                <div className="card-body">
                                    <h5 className="card-title text-
center">Sign In</h5>
                                    <form className="form-signin"
onSubmit={(e) => this.OnSignIn(e)}>
                                        <div className="form-label-group">
                                            <label >Email address</label>
                                            <input type="email"
id="inputEmail" className="form-control" name="uEmail" placeholder="Email
address" required autoFocus onChange={(e) => this.onChangeEmail(e)} />
                                        </div>

                                        <div className="form-label-group">
                                            <label >Password</label>
                                            <input type="password"
id="inputPassword" className="form-control" name="uPass"
placeholder="Password" required onChange={(e) => this.onChangePassword(e)}
/>
                                        </div>
                                        <hr className="my-4"></hr>
                                        <button className="btn btn-md btn-
secondary btn-block text-uppercase" type="submit">Sign in</button>
                                    </form>
                                </div>
                            </div>
                        </div>
                    </div>
                </div>
            </div>
            </>
        );
    }
}


export default Login;
```

This is admin login form. Admin must input his/her email and password. If password or email is invalid proper error message will display

Admin Login - Controller

```
import Axios from 'axios';
import DATA from '../util/env'

// import cookies
import Cookies from "js-cookie";

class Admin {

    constructor() {
        this.api = {
            signin: "/users/login",
            register: "/users/register",
            getAllAdmins: "/users/a/all",
            removeAdmin: "/users/a/r",
            signOut: 'users/a/sign'

        };
    }

    // ------------------------------- Sign In ---------------------------
------------
    // ------------------------------- Sign In ---------------------------
------------
    // ------------------------------- Sign In ---------------------------
------------
    // ------------------------------- Sign In ---------------------------
------------
    async adminSign(Admin) {
        var requestData = {
            admin: Admin
        };
        console.log("REquset data", requestData);

        // var for store respose
        var resp = 600;
        var data = {};
        // sending request
        await Axios.post(
            `${DATA.API}${this.api.signin}`,
            requestData
        )
            .then(Response => {
                resp = Response.status;
                data = Response.data;
            })
            .catch(err => {
                console.error(err);

                try {
                    resp = err.response.status;
                } catch (error) {
                    //   network error
                    resp = 600;
                }
            });

        var returnObj = {
            status: resp,
            data: data
        };
        return returnObj;
    }



    // ------------------------------- Register ---------------------------------------
    // ------------------------------- Register ---------------------------------------
    // ------------------------------- Register ---------------------------------------
    // ------------------------------- Register ---------------------------------------

    async registerAdmin(Admin) {
        var requestData = {
            admin: Admin
        };
        // var for store respose
        var resp = 600;
        var data = {};
        // sending request
        await Axios.post(
            `${DATA.API}${this.api.register}`,
            requestData
        )
            .then(Response => {
                resp = Response.status;
                data = Response.data;
            })
            .catch(err => {
                console.error(err);

                try {
                    resp = err.response.status;
                } catch (error) {
                    //   network error
                    resp = 600;
                }
            });

        var returnObj = {
            status: resp,
            data: data
        };
        return returnObj;
    }




    // ------------------------------- Get all Admins -------------------------------------------
    // ------------------------------- Get all Admins -------------------------------------------
    // ------------------------------- Get all Admins -------------------------------------------
    // ------------------------------- Get all Admins -------------------------------------------


    getAllAdminDetails = async () => {
        var resp = 600;
        var data = {};
        // var for store respose
        var resp = 600;
        var data = {};
        // sending request
        await Axios.get(
            `${DATA.API}${this.api.getAllAdmins}`

        )
            .then(Response => {
                resp = Response.status;
                data = Response.data;
            })
            .catch(err => {
                console.error(err);

                try {
                    resp = err.response.status;
                } catch (error) {
                    //   network error
                    resp = 600;
                }
            });

        var returnObj = {
            status: resp,
            data: data.result
        };


        return returnObj;



    }

    // ------------------------------- Remove Admin ----------------------
-----------------
    // ------------------------------- Remove Admin ----------------------
-----------------
    // ------------------------------- Remove Admin ----------------------
-----------------
    // ------------------------------- Remove Admin ----------------------
-----------------



    async removeAdmin(admin) {

        console.log("Remove work");
        console.log(admin);

        if (admin != undefined || admin != null) {
            var resp = 600;
            var data = {};
            // sending request
            await Axios.post(
                `${DATA.API}${this.api.removeAdmin}`,
                admin
            )
                .then(Response => {
                    resp = Response.status;
                    data = Response.data;
                })
                .catch(err => {
                    console.error(err);

                    try {
                        resp = err.response.status;
                    } catch (error) {
                        //   network error
                        resp = 600;
                    }
                });

            var returnObj = {
                status: resp,
                data: data
            };
            return returnObj;
        }

    }








    // ------------------------------- Set Cookeis -----------------------
----------------
```

```
    // ------------------------------- Set Cookeis -----------------------
----------------
    // ------------------------------- Set Cookeis -----------------------
----------------
    // ------------------------------- Set Cookeis -----------------------
----------------


    // =======================================================
===========================================================================
======================================
    // ===============   set cookies when user login  start here
===========================================================================
=====================================
    // ========================================================
===========================================================================
======================================

    setCookies(status, id, name, email, phone) {
        var secureState = false;


        Cookies.set("cSta", status, { secure: secureState });
        Cookies.set("cI", btoa(id), { secure: secureState });
        Cookies.set("cN", btoa(name), { secure: secureState });
        Cookies.set("cE", btoa(email), { secure: secureState });
        Cookies.set("cP", btoa(phone), { secure: secureState });

    }

    // =======================================================
===========================================================================
======================================
    // ===============   set cookies when user login  end here
===========================================================================
=====================================
    // ========================================================
===========================================================================
======================================
    // =======================================================
==========================================================================
======================================
    // ===============   chekc signed in start here
==========================================================================
====================================
    // ========================================================
==========================================================================
======================================
    checkSignedIn() {
        if (
            Cookies.get("cSta") === false ||
            Cookies.get("cI") === undefined ||
            Cookies.get("cE") === undefined || Cookies.get("cSta") ===
undefined || Cookies.get("cSta") === null
        ) {
            return false;
        } else {
            return true;
        }
    }

    // =======================================================
==========================================================================
======================================
    // ===============   chekc signed in end   here
==========================================================================
====================================
    // ========================================================
==========================================================================
======================================



    // =======================================================
==========================================================================
======================================
    // ===============  get user details from cookies  start here
==========================================================================
===================================
    // ========================================================
==========================================================================
======================================

    // get name
    getName() {
        return atob(Cookies.get("cN"));
    }
    // get state
    getState() {
        return Cookies.get("cSta");
    }
    // get email
    getEmail() {
        return atob(Cookies.get("cE"));
    }
    // get created at
    getPhone() {
        return atob(Cookies.get("cP"));
    }
    // get created at
    getId() {
        return atob(Cookies.get("cI"));
    }



    // =======================================================
==========================================================================
======================================
    // ===============  get user details from cookies  end   here
==========================================================================
===================================
    // ========================================================
==========================================================================
======================================
    // =======================================================
===========================================================================
======================================
    // ===============  Sign out
===========================================================================
===================================
    // ========================================================
===========================================================================
======================================
    async signOut() {

        Cookies.remove("cSta");
        Cookies.remove("cI");
        Cookies.remove("cE");

        window.location.replace("/");

    }

}

var obj = new Admin();
export default obj;
```

This is a controller class in Admin (We use MVC Architecture). All functions which are related to admin in here. This file include all API calls which are related to Admin

Dashboard

```
import React from 'react';
import Sidebar from '../components/Sidebar';
import Topbar from '../components/Topbar';
import { FontAwesomeIcon } from '@fortawesome/react-fontawesome'
import { faCircle , faExclamationCircle} from '@fortawesome/free-solid-
svg-icons'
import axios from 'axios';
import { Line as LineChart } from 'react-chartjs-2';
import DATA from '../util/env'
import moment from 'moment'
import { Link } from "react-router-dom";

import A_Admin from '../Controllers/Admin'

class Dashboard extends React.Component {

    constructor(props) {
        super(props);
        this.state = {
            loading: false,
            sensors: [],
            labels: [],
            co2: [],
            smoke: [],
        }
    }

    async componentDidMount() {
        var State = await A_Admin.checkSignedIn();
        if (State === false) {
            await window.location.replace("/");
        } else {
        this.getDataFromApi();
        this._interval = await setInterval(() => {
            this.getDataFromApi();
        }, 2000);
        }
    }

    getDataFromApi = () => {
        axios.get(`${DATA.API}/sensors/getall/2`)
            .then(result => {

                let labels = [];
                let co2 = [];
                let smoke = [];
                // console.log(result.data.log);
                let dataarray = result.data.log;
                if (dataarray.length > 10) {
                    dataarray = dataarray.slice(Math.max(dataarray.length
- 10, 0))
                }
                // console.log(dataarray.length);

                dataarray.forEach(item => {
                    labels.push(moment(item.datetime).format('HH:mm:ss'));
                    co2.push(item.average_co2);
                    smoke.push(item.average_smoke);
                })

                this.setState({
                    sensors: result.data.current,
                    labels: labels,
                    co2: co2,
                    smoke: smoke
                });
            })
            .catch(err => {
                console.log(err);
            })
    }

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

    render() {
        const { sensors, labels, co2, smoke } = this.state;
        return (
            <>
            <Topbar/>
            <Sidebar/>
            <div className="page-wrapper pt-4">
                <div className="page-breadcrumb">
                    <div className="row align-items-center">
                        <div className="col-12">
                            <h4 className="page-title pt-4 pb-3 px-
2">Sensors Live Data</h4>
```

```
                        </div>
                    </div>

                    <div className="container-fluid">
                        <div className="row">
                            <div className="col-md-9 rounded">
                                <div className="card shadow-sm rounded">
                                    <div className="card-body">
                                        <div className="d-md-flex align-items-center">
                                            <div>
                                                <h4 className="card-title font-weight-bold">Average Co2 & Smoke Levels</h4>
                                                <h5 className="card-subtitle">Update every 2s</h5>
                                            </div>
                                            <div className="ml-auto d-flex no-block align-items-center">
                                                <ul className="list-inline font-12 dl m-r-15 m-b-0">
                                                    <li className="list-inline-item text-info"><FontAwesomeIcon icon={faCircle} /> Co2</li>
                                                    <li className="list-inline-item text-primary"><FontAwesomeIcon icon={faCircle} /> Heat</li>
                                                </ul>
                                            </div>
                                        </div>
                                    </div>
                                    <div className="row">

                                        <div className="col-lg-12">
                                            <div className="campaign ct-charts px-3">
                                            <LineChart data={{
                                                labels:labels ,
                                                datasets:[
                                                   {
                                                     label : "Co2",
                                                     backgroundColor: 'rgba(41,98,255,0.15)',
                                                     borderColor:
'rgba(41,98,255,0.4)',
                                                     data : co2
                                                   },
                                                   {
                                                    label : "Smoke",
                                                    backgroundColor:
'rgba(116,96,238,0.15)',
                                                    borderColor:
'rgba(116,96,238,0.4)',
                                                    data : smoke
                                                   }
                                                ]
                                            }}
                                            options={options}
                                            width="600" height="220"/>
                                            </div>

                                        </div>
                                    </div>
                                </div>
                            </div>
                            <div className="col-md-3">
                                <div className="card  h-100 pb-3" style={{
backgroundColor: "transparent" }}>
                                    <div className="card-body bg-white">
                                        <h4 className="card-title">Danger
Alerts</h4>
                                        <div className="feed-widget">
                                            <ul className="list-style-none
feed-body m-0 p-b-20">
                                            <li class="feed-item">
                                            <FontAwesomeIcon
icon={faExclamationCircle} className="text-danger mr-2 mt-1 mb-auto"/>
                                            <p> Sensor 4 smoke level
increased to danger zone
                                                <span class="font-12 text-
muted ml-2">Just Now</span>
                                            </p>
                                            </li>
```

```
                                            </ul>
                                        </div>
                                    </div>
                                </div>
                            </div>
                        </div>
                        <div className="row" >
                            <div className="col-12">
                                <div className="card">
                                    <div className="card-body pb-1">
                                    <div className="d-md-flex align-items-
center">
                                    <h4 className="card-title">Current
Sensor Details
                                    <span className="card-subtitle small
px-2">Update every 2s</span>
                                    </h4>
                                </div>
                                    </div>
                                    <div className="table-responsive">
                                        <table className="table v-middle"
id="td">
                                            <thead>
                                                <tr className="bg-light">
                                                    <th className="border-
top-0">Floor No</th>
                                                    <th className="border-
top-0">Room No</th>
                                                    <th className="border-
top-0">Co2 Level</th>
                                                    <th className="border-
top-0">Smoke Level</th>
                                                    <th className="border-
top-0">Status</th>
                                                    <th className="border-
top-0">Actions</th>
                                                </tr>
                                            </thead >
                                            <tbody >
```

```
                                                {sensors.map(sensor =>
this.renderSensorTable(sensor))}
                                            </tbody>
                                        </table>
                                    </div>
                                </div>
                            </div>
                        </div>
                    </div>
                </div>
                </div>
            </>
        );
    }

    renderSensorTable = item => {
        const status = (item.co2_level + item.smoke_level) / 2;
        return (<tr key={item.id}>
            <td>
                <div className="d-flex align-items-center">
                    <div className="">
                        <h4 className="m-b-0 font-16">{item.floor_id}</h4>
                    </div>
                </div>
            </td>
            <td>{item.room_id}</td>
            <td><FontAwesomeIcon icon={faCircle} className={`text-
${this.changeStyleColor(item.co2_level)} blink`} /> {item.co2_level}.00
</td>
            <td><FontAwesomeIcon icon={faCircle} className={`text-
${this.changeStyleColor(item.smoke_level)} blink`} />
{item.smoke_level}.00</td>
            <td><span className={`btn-sm bg-light text-
dark`}>{this.changestatus(status)}</span></td>
            <td><Link to={`/sensor/${item.id}`}><span className="label bg-
dark btn font-weight-bold">Details</span></Link> </td>
        </tr>
        );

    }
    changeStyleColor = number => {
        if (number >= 0 && number <= 2) {
            return 'success';
        } else if (number >= 3 && number <= 4) {
            return 'warning';
        } else if (number >= 5 && number <= 10) {
            return 'danger';
        } else {
            return 'secondary';
        }
    }

    changestatus = number => {
        if (number >= 0 && number <= 2) {
            return 'Normal';
        } else if (number >= 3 && number <= 4) {
            return 'Average';
        } else if (number >= 5 && number <= 10) {
            return 'Danger';
        } else {
            return 'None';
        }
    }



}



const options = {
    scaleShowGridLines: false,
    scaleGridLineColor: 'rgba(0,0,0,.05)',
    scaleGridLineWidth: 0,
    scaleShowHorizontalLines: true,
    scaleShowVerticalLines: true,
    bezierCurve: true,
    bezierCurveTension: 0.4,
    pointDot: true,
    pointDotRadius: 4,
    pointDotStrokeWidth: 1,
    pointHitDetectionRadius: 20,
    datasetStroke: true,
    datasetStrokeWidth: 2,
    datasetFill: true,
    legend: {
        display: false
    },
    scales: {
        xAxes: [{
            gridLines: {
                display: false
            }
        }],
        yAxes: [{
            gridLines: {
                display: false
            }
        }]
    }
}

export default Dashboard;
```

This file includes all dashboard functions. Mainly our dashboard has a sidebar, so admin can switch between functions easily. If an admin is not logged into the system , he/she can not load this page. This page shows all average sensor statistics using diagrams.

SensorData

```
import React, { Component } from 'react';
import Sidebar from '../components/Sidebar';
import Topbar from '../components/Topbar';
import axios from 'axios';
import { ToastContainer, toast } from 'react-toastify';
import 'react-toastify/dist/ReactToastify.css';
import DATA from '../util/env'
import moment from 'moment';
import { Link } from "react-router-dom";
import { FontAwesomeIcon } from '@fortawesome/react-fontawesome'
import { faEdit , faTrash } from '@fortawesome/free-solid-svg-icons'
import Modal from 'react-modal';

class SensorData extends Component {

    constructor() {
        super();
        this.state = {
            selectedItem : {},
            delete_modal_visible : false,
            update_mode : false,
            id : '' ,
            floor_id : '',
            room_id : '',
            sensors : [],
            loading : false,
            errors : { floor_id : '' , room_id : ''}
        };
    }

    componentDidMount(){
        this.getDataFromApi();
    }

    getDataFromApi = () => {
        axios.get(`${DATA.API}/sensors/get`)
            .then(result => {
                console.log(result.data);
                this.setState({ sensors: result.data});
            })
            .catch(err => {
                console.log(err);
            })
    }

    formValueChange = (e) => {
       this.setState({[e.target.name] : e.target.value });
    }

    onFormSubmit(e){
        e.preventDefault();
        if(this.validate()){
            if(this.state.update_mode){
                this.updateSensorData()
            }else{
                this.insertNewSensor();
            }
        }
    }


    updateSensorData = () => {
        const { floor_id , room_id , id } = this.state;
        axios.patch(`${DATA.API}/sensors/update` , {
            id : id,
            floor_id : floor_id,
            room_id : room_id,
        })
        .then( result => {
            console.log(result);
            toast("Sensor Successfully Updated");
            this.clearall();
            this.setState({ update_mode : false , selectedItem : {} })
            this.getDataFromApi();
        })
        .catch( err => {
            console.log(err);
            toast("Somthing Wrong Happend");
        })

    }

    insertNewSensor = () => {
        const { floor_id , room_id } = this.state;
        axios.post(`${DATA.API}/sensors/insert` , {
            floor_id : floor_id,
            room_id : room_id,
        })
        .then( result => {
            toast("Sensor Successfully Added");
            this.clearall();
            this.getDataFromApi();
        })
        .catch( err => {
            console.log(err);
            toast("Somthing Wrong Happend");
        })

    }

    deleteSensor = () => {
        const { selectedItem } = this.state;
        axios.delete(`${DATA.API}/sensors/delete/${selectedItem.id}`)
        .then( result => {
            toast("Sensor Successfully Deleted");
            this.setState({delete_modal_visible : false , selectedItem :
{} })
            this.clearall();
            this.getDataFromApi();
        })
        .catch( err => {
            console.log(err);
            toast("Somthing Wrong Happend");
        })

    }


    render() {
        const { sensors , update_mode , room_id , floor_id , errors , id}
= this.state;
        return (
        <>
        <Topbar />
        <Sidebar />
        <div className="page-wrapper pt-5 h-100">
            <div className="page-breadcrumb">
```

```
                <div className="row align-items-center">
                    <div className="col-12">
                        {/* <h4 className="page-title">Admin Register</h4>
*/}
                    </div>
                </div>
                <ToastContainer autoClose={1500} pauseOnFocusLoss={true}
hideProgressBar={true} />
                <div className="container-fluid">
                    <div className="row mt-2">

                        <div className="col-md-12">
                            <div className="card  h-100 pb-3" style={{
backgroundColor: "transparent" }}>
                                <div className="card-body bg-white">
                                    <div className="col-12 pl-0" >
                                    <h4 className="page-title">Sensors
Settings
                                    <span className="badge mx-2 font-
weight-bold px-2 badge-secondary">
                                        {update mode ? 'Update Mode' :
'Add Mode'}
                                    </span></h4>
                                        <br />
                                    </div>
                                     <form onSubmit={(e) =>
this.onFormSubmit(e)}>
                                        <div className="row">
                                            <div className="mt-2 col-md-
6">
                                                <div className="form-
label-group">
                                                    <label >Sensor ID
</label>
                                                    <input type="text"

className="form-control"
                                                           name="id"
                                                           value={id}
```

```
placeholder="Value will be auto genarated"
                                                            onChange={ (e)
=> this.formValueChange(e)}
                                                            readOnly
                                                      />
                                                  </div>
                                              </div>
                                              <div className="mt-2 col-md-
6">
                                                  <div className="form-
label-group">
                                                      <label >Floor No
</label>
                                                      <input
                                                            type="text"
                                                            className="form-
control"
                                                            name="floor_id"
                                                            value={floor_id}
                                                            onChange={ (e)
=> this.formValueChange(e)}

placeholder="Enter Floor No"  />
                                                            {
errors.floor_id.length > 0 &&
                                                              <h4
className="small text-danger mt-2 font-weight-bold mb-
0">{errors.floor_id}</h4>
                                                            }
                                                  </div>
                                              </div>
                                              <div className="mt-2 col-md-
6">
                                                  <div className="form-
label-group">
                                                      <label >Room No
</label>
                                                      <input type="text"
```

```
className="form-control"
                                                            onChange={ (e)
=> this.formValueChange(e)}
                                                            name="room_id"
                                                            value={room_id}

placeholder="Enter Room No"  />
                                                            {
errors.room_id.length > 0 &&
                                                             <h4
className="small text-danger mt-2 font-weight-bold mb-
0">{errors.room_id}</h4>
                                                            }
                                                 </div>
                                             </div>
                                             <div className="mt-auto  col-
md-6 ">
                                                 { !update_mode &&
                                                 <button
                                                     className="btn btn-
success  p-2 rounded btn-sm  font-weight-bold  text-uppercase"
                                                     type="submit">
                                                     Add Sensor
                                                 </button>
                                                 }
                                                 { update_mode &&
                                                 <div>
                                                     <button
                                                         className="btn
btn-secondary  p-2 rounded btn-sm  font-weight-bold   text-uppercase"
                                                         type="submit">
                                                         Update Sensor
                                                     </button>
                                                     <button
                                                         className="btn
btn-danger mx-2 p-2 rounded btn-sm  font-weight-bold   text-uppercase"
                                                         onClick={(e) =>
this.onPressUpdateModeCancel(e) }
                                                         type="cancel">
```

```
                                                        Cancel
                                                    </button>
                                                </div>
                                                }
                                            </div>
                                        </div>
                                    </form>
                                </div>
                            </div>
                        </div>
                    </div>
                    <div className="row mt-2" >
                        <div className="col-12">
                            <div className="card">
                                <div className="card-body pb-1">
                                    <div className="d-md-flex align-items-
center">
                                        <div>
                                            <h4 className="card-
title">Sensor Details</h4>
                                        </div>

                                    </div>
                                </div>
                                <div className="table-responsive">
                                    <table className="table v-middle"
id="td">
                                        <thead>
                                            <tr className="bg-light">
                                                <th className="border-top-
0">Sensor ID</th>
                                                <th className="border-top-
0">Floor No</th>
                                                <th className="border-top-
0">Room No</th>
                                                <th className="border-top-
0">Updated Date</th>
                                                <th className="border-top-
0">Time</th>
```

```
                                                <th className="border-top-
0">Actions</th>
                                            </tr>
                                        </thead >
                                        <tbody >
                                          { sensors.map( item =>
this.renderSensorTable(item) )}
                                        </tbody>
                                    </table>
                                </div>
                            </div>
                        </div>
                    </div>
                </div>
            </div>
        </div>
        <this.deleteModal/>
         </>
        );
    }

    renderSensorTable = item => {
        return (<tr key={item.id}>
            <td>
                <div className="d-flex align-items-center">
                    <div className="">
                        <h4 className="m-b-0 font-16">{item.id}</h4>
                    </div>
                </div>
            </td>
            <td>{item.floor_id}</td>
            <td>{item.room_id}</td>
            <td>{moment(item.updated_at).format('DD , MMMM YYYY') }</td>
            <td>{moment(item.updated_at).format('LT') }</td>
            <td>
                <Link to={`/sensor/${item.id}`}><span className="label py-
1 ml-2 bg-dark btn font-weight-bold">Details</span></Link>
                <span className="label py-1 bg-secondary btn font-weight-
bold ml-2"
```

```
                 onClick={() => this.onPressUpdate(item)}
><FontAwesomeIcon icon={faEdit} /></span>
                <span  onClick={() => this.onPressDelete(item)}
                 className="label py-1 bg-danger btn font-weight-bold ml-
2"><FontAwesomeIcon icon={faTrash} /></span>
            </td>
        </tr>
        );
    }

    deleteModal = () => (
        <Modal
            shouldCloseOnOverlayClick={true}
            style={customStyles}
            overlayClassName="Overlay"
            onRequestClose={() => this.setState({delete modal visible :
false}) }
            isOpen={this.state.delete modal visible}
        >
        <h4>Sensor Delete</h4>
        <p>Are you sure you want to delete
             <spam className="font-weight-bold mx-1">ID :
{this.state.selectedItem.id}</spam> sensor ? <br></br>
        <span className="text-danger" >This process can not be undone
!</span>
        </p>
        <div className="d-flex" >
        <button onClick={() => this.deleteSensor()}
        className="btn btn-danger px-2 ml-auto">Delete</button>
        <button
            onClick={() => this.setState({delete modal visible : false}) }
            className="btn btn-secondary px-2 mx-2">close</button>
        </div>
        </Modal>
    );

    validate = () => {
        let { errors , floor id , room id } = this.state;
        let count = 0;
        if( floor_id.length == 0 ){
            errors.floor_id = "Floor no can not be empty"
            count++
        }else{
            errors.floor_id = ""
        }
        if( room_id.length == 0 ){
            errors.room_id = "Room no can not be empty"
            count++
        }else{
            errors.room_id = ""
        }

        this.setState({errors});
        return count == 0;
    }

    onPressDelete = sensor =>{
        this.setState({
            selectedItem : sensor,
            delete_modal_visible : true,
            update_mode : false,
            floor_id : "",
            room_id : "",
            id : "",
            errors : { floor_id : '' , room_id : ''}
        });
    }

    onPressUpdate = sensor =>{
        this.setState({
            selectedItem : sensor,
            update_mode : true,
            floor_id : sensor.floor_id,
            room_id : sensor.room_id,
            id : sensor.id,
            errors : { floor_id : '' , room_id : ''}
        });
    }
    onPressUpdateModeCancel = (e) => {
        e.preventDefault();
        this.setState({ update mode : false , selectedItem : {} })
        this.clearall();
    }

    clearall = () => {
        this.setState({
            floor id : "",
            room id : "",
            id : "",
        });
    }
}

const customStyles = {
    content : {
      top                     : '25%',
      left                    : '50%',
      right                   : '10%',
      bottom                  : 'auto',
      transform               : 'translate(-50%, -50%)'
    }
  };
export default SensorData;
```

This component shows all sensor data. This component updates every 5 seconds. So users can get the latest updated details about sensors. We use special color to display sensor details for admins because admins can get an idea of the sensors easily.

SingleSensor

```
import React from 'react';
import Sidebar from '../components/Sidebar';
import Topbar from '../components/Topbar';
import {FontAwesomeIcon} from '@fortawesome/react-fontawesome'
import { faArrowUp , faArrowDown , faCircle } from '@fortawesome/free-
solid-svg-icons'
import axios from 'axios';
import {Line as LineChart} from 'react-chartjs-2';
import DATA from '../util/env'
import moment from 'moment'
import {Link } from "react-router-dom";
import ReactSpeedometer from "react-d3-speedometer"

class SingleSensor extends React.Component {

    constructor(props){
        super(props);
        this.state = {
            prevCo2 : 0,
            prevSmoke : 0,
            log : [],
            loading : false,
            sensor : { smoke_level : 0 , co2_level : 0 },
            labels : [],
            co2 : [],
            smoke : [],
        }
    }

    componentDidMount(){
        const { id } = this.props.match.params
        this.getDataFromApi(id);
        this._interval = setInterval(() => {
            this.getDataFromApi(id);
          }, 4000);

    }

    getDataFromApi = id => {
    axios.get(`${DATA.API}/sensors/getall/${id}/20`)
            .then( result => {

                let labels = [];
                let co2 = [];
                let smoke = [];

                let dataarray = result.data.data.log;
                if(dataarray.length > 6){
                    dataarray =  dataarray.slice(Math.max(dataarray.length
- 6 , 0))
                }

                let log = result.data.data.log;
                if(log.length > 12){
                    log =  log.slice(Math.max(log.length - 12 , 0))
                }

                dataarray.forEach( item => {
                    labels.push(moment(item.datetime).format('HH:mm:ss')
);

                    co2.push(item.co2_level);
                    smoke.push(item.smoke_level);
                })

                const prev = this.state.sensor;
                this.setState({
                    sensor : result.data.data.current ,
                    labels : labels ,
                    co2 : co2 ,
                    smoke : smoke ,
                    log : log,
                    prevCo2 : prev.co2_level,
                    prevSmoke : prev.smoke_level
                });

            })
            .catch( err => {
                console.log(err);
            })
    }

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

    render(){
        const {sensor , labels , co2 , smoke  , log , prevSmoke , prevCo2
} = this.state;
        return(
            <>
            <Topbar/>
            <Sidebar/>
            <div className="page-wrapper ">
                <div className="page-breadcrumb">
                    <div className="row align-items-center">
                        <div className="col-12">
                            <h4 className="page-title">Sensor Live
Data</h4>
                        </div>
                    </div>
                </div>

                <div className="container-fluid">
                    {/* -----------------card area---------------- */}
                    <div className="row mb-3">
                        <div className="mt-2 col-lg-3 col-md-6 col-sm-6
rounded">
                            <div className="h-100 bg-white shadow-sm pt-2
rounded ">
                            <div className="d-flex" >
                                <img src="images/co2.png" className="my-
auto ml-3 my-auto" width="60" height="60"/>
                                    <div className="pt-1 pl-3 pr-3 ">
                                        <h5 className="card-title mb-0
font-weight-bold">Co2 Level</h5>
                                        <h2 className={`text-
${this.changeStyleColor(sensor.co2_level)} card-title font-weight-bold mb-
0`}>{sensor.co2_level}.00</h2>
                                        {
this.changeLimit(sensor.co2_level , prevCo2 ) }
                                    </div>

                                </div>
                            </div>
                        </div>
                        <div className="mt-2 col-lg-3 col-md-6 col-sm-6
rounded">
                            <div className="h-100 bg-white mb-0 shadow-sm
pt-2 rounded ">
                            <div className="d-flex" >
                                <img src="images/sensor.png"
className="my-auto ml-3 my-auto" width="50" height="50"/>
                                    <div className="pt-1 pl-3 pr-3 ">
                                        <h5 className="card-title mb-0
font-weight-bold">Smoke Level</h5>
                                        <h2 className={`text-
${this.changeStyleColor(sensor.smoke_level)} card-title font-weight-bold
mb-0`}>{sensor.smoke_level}.00</h2>
                                        {
this.changeLimit(sensor.smoke_level , prevSmoke ) }
                                    </div>
                                    <div className="my-auto pt-3 pl-3">
                                        <h4 className="card-title mb-0
small text-danger font-weight-bold "></h4>

                                    </div>
                                </div>
                            </div>
                        </div>
                        <div className="mt-2 col-lg-2 col-md-4 col-sm-6
rounded">
                            <div className="h-100 bg-white mb-0  shadow-sm
pt-2 rounded ">
                            <div className="d-flex" >
                                    <div className="pt-3 px-3 mx-auto">
                                        <h5 className="card-title mb-0
font-weight-bold">Sensor Id</h5>
                                        <h2 className="card-title text-
secondary font-weight-bold">00{sensor.id}</h2>
                                    </div>
                                </div>
                            </div>
                        </div>
                        <div className="mt-2 col-lg-2 col-md-4 col-sm-6
rounded">
```

```
                            <div className="h-100 bg-white mb-0 shadow-sm
pt-2 rounded ">
                            <div className="d-flex" >
                                    <div className="pt-3 px-3 mx-auto">
                                        <h5 className="card-title mb-0
font-weight-bold">Floor No</h5>
                                        <h2 className="card-title text-
secondary font-weight-bold">{sensor.floor_id}</h2>
                                    </div>
                                </div>
                            </div>
                        </div>
                        <div className="mt-2 col-lg-2 col-md-4 col-sm-6
rounded">
                            <div className="h-100  bg-white mb-0  shadow-
sm pt-2 rounded ">
                            <div className="d-flex" >
                                    <div className="pt-3 px-3 mx-auto">
                                        <h5 className="card-title mb-0
font-weight-bold">Room No</h5>
                                        <h2 className="card-title text-
secondary font-weight-bold">{sensor.room_id}</h2>
                                    </div>
                                </div>
                            </div>
                        </div>
                    </div>
                    {/* ------------------------------------------ */}
                    <div className="row">
                        <div className="col-md-6 rounded">
                            <div className="card shadow-sm rounded">
                                <div className="card-body">
                                    <div className="d-md-flex align-items-
center">
                                        <div>
                                            <h4 className="card-title
font-weight-bold">Smoke Level</h4>
                                            <h5 className="card-
subtitle">Update every 2s</h5>
                                        </div>
                                        <div className="ml-auto d-flex no-
block align-items-center">
                                            <ul className="list-inline
font-12 dl m-r-15 m-b-0">
                                                <li className="list-
inline-item text-primary"><FontAwesomeIcon icon={faCircle} /> Smoke </li>
                                            </ul>
                                        </div>
                                    </div>
                                    <div className="row">

                                        <div className="col-lg-12">
                                            <div className="my-2 ct-
charts">

                                            <LineChart data={{
                                                labels: labels ,
                                                datasets:[{
                                                    label : "Smoke",
                                                    backgroundColor:
'rgba(116,96,238,0.15)',

                                                    borderColor:
'rgba(116,96,238,0.4)',

                                                    data : smoke
                                                   }]
                                            }}
                                            options={options}
                                            width={600} height={260}/>
                                            </div>
                                        </div>

                                    </div>
                                </div>
                            </div>
                        </div>

                        <div className="col-md-6 rounded">
                            <div className="card shadow-sm rounded">
                                <div className="card-body">
                                    <div className="d-md-flex align-items-
center">
```

```
                                        <div>
                                            <h4 className="card-title
font-weight-bold">Co2 Level</h4>
                                            <h5 className="card-
subtitle">Update every 2s</h5>
                                        </div>
                                        <div className="ml-auto d-flex no-
block align-items-center">
                                            <ul className="list-inline
font-12 dl m-r-15 m-b-0">
                                                <li className="list-
inline-item text-info"><FontAwesomeIcon icon={faCircle} /> Co2</li>
                                            </ul>
                                        </div>
                                    </div>
                                    <div className="row">

                                        <div className="col-lg-12">
                                            <div className="my-2 ct-
charts">
                                            <LineChart data={{
                                                labels: labels ,
                                                datasets:[{
                                                    label : "Co2",
                                                    backgroundColor:
'rgba(41,98,255,0.15)',
                                                    borderColor:
'rgba(41,98,255,0.4)',
                                                    data : co2
                                                   }]
                                            }}
                                            options={options}
                                            width={600} height={260}/>
                                            </div>
                                        </div>

                                    </div>
                                </div>
                            </div>
                        </div>
```

```
                    </div>
                    <div className="row" >
                    <div className="col-12">
                        <div className="card">
                            <div className="card-body pb-1">
                                <div className="d-md-flex align-items-
center">
                                    <h4 className="card-title">Sensor Log
                                    <span className="card-subtitle small
px-2">Last 02 min</span>
                                    </h4>
                                </div>
                            </div>
                            <div className="table-responsive">
                                <table className="table v-middle" id="td">
                                    <thead>
                                        <tr className="bg-light">
                                            <th className="border-top-
0">Date</th>

                                            <th className="border-top-
0">Time</th>

                                            <th className="border-top-
0">Co2 Level</th>

                                            <th className="border-top-
0">Smoke Level</th>

                                            <th className="border-top-
0">Status</th>
                                        </tr>
                                    </thead >
                                    <tbody >
                                        {log.slice(0).reverse().map(sensor
=> this.renderSensorTable(sensor))}
                                    </tbody>
                                </table>
                            </div>
                        </div>
                    </div>
                    </div>
                </div>
```

```
            </div>
            </>
        );
    }

    renderSensorTable = item => {

        const status = (item.co2_level + item.smoke_level) / 2;
        return (<tr key={item.id}>
            <td>
                <div className="d-flex align-items-center">
                    <div className="">
                        <h6 className="m-b-0 font-
16">{moment(item.datetime).format('DD , MMMM YYYY')}</h6>
                    </div>
                </div>
            </td>
            <td>{moment(item.datetime).format('hh:mm:ss')}</td>
            <td>
                <FontAwesomeIcon
                    icon={faCircle}
                    className={`text-
${this.changeStyleColor(item.co2_level)} `}
                /> {item.co2_level}.00
            </td>
            <td>
                <FontAwesomeIcon
                    icon={faCircle}
                    className={`text-
${this.changeStyleColor(item.smoke_level)} `}
                    /> {item.smoke_level}.00
            </td>
            <td>
                <span
                    className={`btn-sm bg-light text-dark`}>
                        {this.changestatus(status)}
                </span>
            </td>
        </tr>
        );
    }

    changeLimit = (current, prev ) => {
        if(current != prev && prev != 0 ){
        let increased = current > prev;
        let dif = Math.abs( current - prev ) / prev * 100
        return (
             dif > 0 ?
            <h4 className={`text-${increased && dif >= 500 ? 'danger' :
'dark'} small card-title font-weight-bold`}>
               {increased ? 'Increased' : 'Decreased'} By
                <FontAwesomeIcon icon={ increased ? faArrowUp :
faArrowDown } className="mx-1"/> {(Math.round( dif * 100) /
100).toFixed(2)}%
            </h4>
            : <h4 className={`text-danger small card-title font-weight-
bold`}></h4>

        );
        }
    }

    changeStyleColor = number => {
        if (number >= 0 && number <= 2) {
            return 'success';
        } else if (number >= 3 && number <= 4) {
            return 'warning';
        } else if (number >= 5 && number <= 10) {
            return 'danger';
        } else {
            return 'secondary';
        }
    }

    changestatus = number => {
        if (number >= 0 && number <= 2) {
            return 'Normal';
        } else if (number >= 3 && number <= 4) {
            return 'Average';
        } else if (number >= 5 && number <= 10) {
            return 'Danger';
        } else {
            return 'None';
        }
    }


}


const options = {
    scaleShowGridLines: false,
    scaleGridLineColor: 'rgba(0,0,0,.05)',
    scaleGridLineWidth: 0,
    scaleShowHorizontalLines: true,
    scaleShowVerticalLines: true,
    bezierCurve: true,
    bezierCurveTension: 0.4,
    pointDot: true,
    pointDotRadius: 4,
    pointDotStrokeWidth: 1,
    pointHitDetectionRadius: 20,
    datasetStroke: true,
    datasetStrokeWidth: 2,
    datasetFill: true,
    legend: {
        display: false
     },
    scales: {
        xAxes: [{
            gridLines: {
                display:false
            }
        }],
        yAxes: [{
            gridLines: {
                display:false
            }
        }]
    }
 }

export default SingleSensor;
```

This component shows all details about a single sensor which was admin selected. Admin can get all previous and present details about a single sensor. We use diagrams to show some statistics.

AdminRegister

```
import React, { Component } from 'react';

import Sidebar from '../../components/Sidebar';
import Topbar from '../../components/Topbar';
import A Admin from '../../Controllers/Admin'

import { ToastContainer, toast } from 'react-toastify';
import 'react-toastify/dist/ReactToastify.css';

import { Button, Modal } from 'react-bootstrap';

import UniqId from 'uniqid'

import './login.css'
class AdminRegister extends Component {
    constructor() {
        super();
        this.clearall = this.clearall.bind(this)
        this.state = {
            uName: '',
            uEmail: '',
            uCn: '',
            uPass: '',
            uConPass: '',
            passwordMatch: true,
            allusers: [],
            showModal: false,
            deleteId: '',
            deleteName: '',
            deletepw: ''
        };



    }


    async componentWillMount() {
        await this.getRegditedAdmins()
        await console.log(this.state.allusers);

    }

    // admin register
    onChnageName(e) {
        this.setState({
            uName: e.target.value
        })
    }
    onChangeEmail(e) {
        this.setState({
            uEmail: e.target.value
        })
    }

    onChangeCn(e) {
        this.setState({
            uCn: e.target.value
        })
    }
    // onChangePassword(e) {
    //     this.setState({

    //         uPass: e.target.value
    //     })
    // }


    // onChangeConPassword(e) {
    //     this.setState({

    //         uConPass: e.target.value
    //     }, () => this.checkPasswordMatch())
    // }

    //password match
    // async  checkPasswordMatch() {
    //     if (this.state.uPass != this.state.uConPass) {
    //         await this.setState({
    //             passwordMatch: false
    //         })
    //     } else {
    //         await this.setState({
    //             passwordMatch: true
    //         })
    //     }
    // }


    clearall() {
        this.setState({
            uName: '',
            uEmail: '',
            uCn: '',
            uPass: ''

        })
    }

    //register admin
    async registerAdmin(e) {
        // e.preventDefault()


        const Admin = {
            uName: this.state.uName,
            uEmail: this.state.uEmail,
            uCn: this.state.uCn,
            uPass: UniqId('ALS')
        }

        console.log(Admin);

        var addUser = await A Admin.registerAdmin(Admin);

        console.log(addUser);

        switch (addUser.status) {
            case 200:

                await this.getRegditedAdmins()
                await this.clearall()
                await this.notify()

                break;
            case 201:
                console.log("Already you have user");

            default:
                window.location.replace("/admin");
        }



    }


    // get registed admins
    async   getRegditedAdmins() {
        var alluser = await A_Admin.getAllAdminDetails()
        switch (alluser.status) {
            case 200:
                console.log("Data come");
                this.setState({
                    allusers: alluser.data
                })

                break;
            case 201:
                console.log("No data");

            default:
            // window.location.replace("/admin");
        }
    }




    // modal-----------------

    closeModal() {
        this.setState({
            showModal: false
        })
    }
    async  showModal(id, name) {
        await this.setState({
            showModal: true,
            deleteId: id,
            deleteName: name

        })

    }

    onChnageDeletePassword(e) {
        this.setState({
            deletepw: e.target.value
        })
    }

    async onDelete(e) {
        e.preventDefault()

        var deleteAccount = {
            id: this.state.deleteId,
            password: this.state.deletepw
        }

        var deleteState = await A_Admin.removeAdmin(deleteAccount)
        console.log(deleteState);

        switch (deleteState.status) {
            case 200:
                var title = "Successfully Removed"
                await this.setState({
                    showModal: false
                })
                await this.notifyB(title)
                await this.getRegditedAdmins()
                break;
            case 201:
                var title = "Something went wrong"
                await this.setState({
                    showModal: false
                })
                await this.notifyB(title)
                break;
            default:
                var title = "Something went wrong"
                await this.setState({
                    showModal: false
                })
                await this.notifyB(title)
                await this.getRegditedAdmins()
        }
    }



    // userSignOut(){
    //      var signOut = A Admin.signOut();
    // }


    // messages
    notify = () => toast("Successfully Added");
    notifyB = (title) => toast(title);

    render() {


        if (this.state.allusers != null || this.state.allusers !=
undefined) {

            var allAdmins = this.state.allusers.map((data, i) => {
                return (<tr key={data.id}>
                    <td>
                        {data.id}
                    </td>
                    <td>{data.name}</td>
                    <td>{data.email}</td>
                    <td>{data.phone}</td>
                    <td>
                        <button className="btn btn-sm btn-danger"
onClick={() => this.showModal(data.id, data.name)}>Remove</button>
                    </td>

                </tr>
                );
            });
        } else {
            return (<h3>No Admin</h3>)
        }
        return (
            <>
                <Topbar />
                <Sidebar />


                {/* Registerd Admin Views */}

                <div className="page-wrapper pt-5">
                    <div className="page-breadcrumb">
                        <div className="row align-items-center">
                            <div className="col-12">
                                {/* <h4 className="page-title">Admin
Register</h4> */}
                            </div>
                        </div>
                        <ToastContainer autoClose={1500}
pauseOnFocusLoss={true} hideProgressBar={true} />
                        <div className="container-fluid">
                            <div className="row">

                                <div className="col-md-12">
                                    <div className="card  h-100 pb-3"
style={{ backgroundColor: "transparent" }}>
                                        <div className="card-body bg-
white">
                                            <div className="col-12 pl-0" >
                                                <h4 className="page-
title">Register Admin</h4>
                                                <br />
                                            </div>
                                            <form onSubmit={(e) =>
this.registerAdmin(e)}>
                                                <div className="row">
                                                    <div className="col-
md-4">
                                                        <div
className="form-label-group">
                                                            <label >Name :
</label>
```

```
                                                                        <input
type="text" className="form-control" name="uName" placeholder="Name"
required autoFocus onChange={(e) => this.onChnageName(e)} />
                                                                    </div>
                                                                </div>
                                                                <div className="col-
md-4">
                                                                    <div
className="form-label-group">
                                                                        <label >Email
address : </label>
                                                                        <input
type="email" className="form-control" name="uEmail" placeholder="Email
address" required onChange={(e) => this.onChangeEmail(e)} />
                                                                    </div>
                                                                </div>
                                                                <div className="col-
md-4">
                                                                    <div
className="form-label-group">
                                                                        <label
>Contact No : </label>
                                                                        <input
type="tel" className="form-control" name="uCn" placeholder="Contact
Number" required onChange={(e) => this.onChangeCn(e)} />
                                                                    </div>
                                                                </div>
                                                                <div className="col-
md-4">
                                                                    <div
className="form-label-group">
                                                                        <label
>Password : </label>
                                                                        <input
type="password" className="form-control" name="uPass"
placeholder="Password sent to your email" disabled />
                                                                    </div>
                                                                </div>
                                                                {/* <div
className="col-md-4">
```

```
                                                        <div
className="form-label-group">
                                                            <label
>Confirm Password : </label>
                                                            <input
type="password" className="form-control" name="uConPass"
placeholder="Confirm Password" required onChange={(e) =>
this.onChangeConPassword(e)} />
                                                        </div>
                                                    </div> */}
                                                    <div className="col-
md-3 mt-2">
                                                        <br />
                                                        <button
className="btn btn-secondary btn-block text-uppercase"
type="submit">Register</button>
                                                    </div>
                                                    <div className="col-
md-12 mt-2">
                                                        <br />
                                                        <p
className="SignUp password not match" style={{ display:
this.state.passwordMatch === false ? 'block' : 'none' }}>Password and
Confrim Password did not match</p>
                                                    </div>


                                                </div>
                                            </form>

                                        </div>
                                    </div>
                                </div>
                            </div>
                            <div className="row" >
                                <div className="col-12">
                                    <div className="card">
                                        <div className="card-body pb-1">
                                            <div className="d-md-flex
align-items-center">
```

```
                                                <div>
                                                    <h4 className="card-
title">Admin Details</h4>
                                                </div>

                                            </div>
                                        </div>
                                        <div className="table-responsive">
                                            <table className="table v-
middle" id="td">
                                                <thead>
                                                    <tr className="bg-
light">
                                                        <th
className="border-top-0">ID</th>
                                                        <th
className="border-top-0">Name</th>
                                                        <th
className="border-top-0">Email</th>
                                                        <th
className="border-top-0">Contact No</th>
                                                        <th
className="border-top-0">Actions</th>
                                                    </tr>
                                                </thead >
                                                <tbody >
                                                    {allAdmins}
                                                </tbody>
                                            </table>
                                        </div>
                                    </div>
                                </div>
                            </div>
                        </div>
                    </div>


                    {/* ========================================= */}
                    {/* =================Delte Account ========== */}
                    {/* ========================================= */}
                    <Modal show={this.state.showModal} animation={false}>
                        <Modal.Header >
                            <Modal.Title>Delete Admin</Modal.Title>
                        </Modal.Header>
                        <Modal.Body>Hey {this.state.deleteName}, <br /> Do
you want to delete your account !</Modal.Body>
                        <form onSubmit={(e) => this.onDelete(e)}>
                            <div className="row m-2 p-2">
                                <div className="col-md-4">
                                    <div className="form-label-group">
                                        <label >Password :  </label>
                                    </div>
                                </div>
                                <div className="col-md-5">
                                    <div className="form-label-group">
                                        <input type="password"
className="form-control" name="delPass" required autoFocus onChange={(e)
=> this.onChnageDeletePassword(e)} />
                                    </div>
                                </div>
                            </div>

                            <Modal.Footer>
                                <Button variant="secondary" onClick={() =>
this.closeModal()}>
                                    Close
                                 </Button>
                                <Button variant="primary" type="submit" >
                                    Save Changes
                             </Button>
                            </Modal.Footer>
                        </form>
                    </Modal>
                </div>

            </>
        );
    }
}
```

```
export default AdminRegister;
```

In this component admin can create other admins. In this super admin must input sub admins details like Name, Email, Contact number. Then the system automatically generates a password and it is sent to the newly added admin's email. Sub admins can delete their account after the validate password.

router -> index

```
// import Login
import Login from '../views/Admin/Login'


let indexRoutes = [
  {
    path: "/*",
    name: "Login",
    component: Login,
  }
];

export default indexRoutes;
```

This router file includes a non protected router.

router -> protected

```

import Dashboard from '../views/Dashboard';
import SingleSensor from '../views/SingleSensor';
import SensorData from '../views/SensorData';

// import Login
import Login from '../views/Admin/Login'
import AdminVieww from '../views/Admin/AdminRegister'
```

```
let ProtectedindexRoutes = [
    {
        path: "/dashboard",
        name: "Dashboard",
        component: Dashboard,
        exact: true,
    },
    {
        path: "/sensor/:id",
        name: "SingleSensor",
        component: SingleSensor,
    },
    {
        path: "/sensordata",
        name: "Sensor Data",
        component: SensorData,
        exact: true,
    },
    {
        path: "/admin",
        name: "AdminView",
        component: AdminVieww,
    },
    {
        path: "/*",
        name: "Dashboard",
        component: Dashboard,
    },

];

export default ProtectedindexRoutes;
```

This file includes all protected routes. This route can access only who logged users in to the system.

Sidebar

```
import React from 'react';
import {FontAwesomeIcon} from '@fortawesome/react-fontawesome'
import { faTachometerAlt} from '@fortawesome/free-solid-svg-icons'
import { Link } from "react-router-dom";

class Sidebar extends React.Component {
    render(){
        return(
            <aside className="left-sidebar" data-sidebarbg="skin6">
                <div className="scroll-sidebar">
                <nav className="sidebar-nav">
                    <ul id="sidebarnav" className="px-2">
                        <li>
                            <div className="user-profile d-flex no-block
dropdown m-t-20">
                                <div className="user-pic">
                                    <img src="images/users/1.jpg"
alt="users" className="rounded-circle" width="40" />
                                </div>
                                <div className="user-content hide-menu ml-
3">
                                    <h5 className="mb-0 user-name font-
medium">Admin </h5>
                                    <span className="op-5 user-
email">amoda@gmail.com</span>
                                </div>
                            </div>
                        </li>
                        <li className="sidebar-item">
                            <Link to="/">
                            <div className="sidebar-link waves-effect
waves-dark sidebar-link"
                                href="/" aria-expanded="false">
                                <FontAwesomeIcon icon={faTachometerAlt} />
<span className="pl-2 hide-menu">
                                    Dashboard</span>
                            </div>
                            </Link>
                            <Link to="/sensordata">
                            <div className="sidebar-link waves-effect
waves-dark sidebar-link"
                                href="/" aria-expanded="false">
                                <FontAwesomeIcon icon={faTachometerAlt} />
<span className="pl-2 hide-menu">
                                    Sensors</span>
                            </div>
                            </Link>
                            <Link to="/admin">
                            <div className="sidebar-link waves-effect
waves-dark sidebar-link"
                                href="" aria-expanded="false">
                                <FontAwesomeIcon icon={faTachometerAlt} />
<span className="pl-2 hide-menu">
                                    Admins</span>
                            </div>
                            </Link>
                        </li>
                    </ul>
                </nav>
            </div>
        </aside>
        );
    }
}

export default Sidebar;
```

This sidebar of the system admin can switch between different views easily.

Topbar

```
import React from 'react';
import { FontAwesomeIcon } from '@fortawesome/react-fontawesome'

import { faSignOutAlt } from '@fortawesome/free-solid-svg-icons'

import A_Admin from '../Controllers/Admin'
import './topbar.css'

class Topbar extends React.Component {
    signOut() {
        var sg = A_Admin.signOut()
    }
    render() {
        return (

            <nav className="navbar navbar-dark bg-dark p-0 m-0 fixed-top">
                <div className="navbar-header" >
                    <a className="navbar-brand">
                        <h4 className=" text-light mt-1 mx-2">
                            Sensor Dashboard
                        </h4>

                    </a>
                    <a className="nbt sig" onClick={()=> this.signOut()} >
                    <FontAwesomeIcon icon={faSignOutAlt} style={{ color:
'white' }} />  <span className="pl-2 hide-menu " style={{ color: 'white'
}}>
                            Sign Out</span>

                    </a>

                </div>
            </nav>
        );
    }
}

export default Topbar;
```

This is the top bar of the system. This topbar has sign out button

App.js

```
import React from 'react';
import { BrowserRouter as Router, Route, Switch } from 'react-router-dom';
import indexRoutes from './routes/index'
import ProtectedindexRoutes from './routes/protected'
import A_Admin from './Controllers/Admin'
```

```
class App extends React.Component {

  render() {
    return (
      <>
        {A_Admin.checkSignedIn() == false ? <Router>
         <Switch>
            {indexRoutes.map((prop, key) => {
              return (
                <Route
                  path={prop.path}
                  key={key}
                  component={prop.component}
                  exact={prop.exact ? true : false}
                />
              );
            })}

          </Switch>
        </Router> : <Router>
            <Switch>

              {ProtectedindexRoutes.map((prop, key) => {
                return (
                  <Route
                    path={prop.path}
                    key={key}
                    component={prop.component}
                    exact={prop.exact ? true : false}
                  />
                );
              })}
            </Switch>
          </Router>}

      </>
    );
  }
}
```

```
export default App;
```

This is the main class in our system. This class include all routes and other details which are related to our system

Backend - Nodejs (REST API)

Controllers

```
//import Sensors model
const { GET_ALL, GET_SENSOR, INSERT_SENSOR, UPDATE_SENSORS,
    UPDATE_SENSORS_LOG, GET_SINGLE_ALL_BY_MINUTES, GET_ALL_BY_MINUTES } =
require('../models/sensors.model');
const connection = require('../../config/db.config');
const moment = require('moment');
// import email sent
const UtilObj = require('../util/util')


exports.getAll = function (req, res) {
    connection.query(GET_ALL, (err, result) => {
        if (err)
            throw err;
        res.end(JSON.stringify(result));
    });
};

exports.getAllForMinutes = function (req, res) {

    const minutes = req.params.minutes;
    const now = new Date();
    var to = moment(now).format('YYYY-MM-DD HH:mm:ss');
    var from = moment(now).subtract(minutes, "minutes").toDate();
    from = moment(from).format('YYYY-MM-DD HH:mm:ss');

    var data = {};
    data.from = from;
    data.to = to;

    connection.query(GET_ALL, (err, result) => {
        if (err)
            throw err;
        data.current = result;

        connection.query(GET_ALL_BY_MINUTES, [from, to], (err, result) =>
{
            if (err)
                throw err;

            data.log = result;
            res.end(JSON.stringify(data));
        });
    });
};

exports.get = function (req, res) {
    const param = [req.params.id];
    connection.query(GET_SENSOR, param, (err, result) => {
        if (err)
            throw err;
        if (result.length == 0) {
            res.end(JSON.stringify({ status: 'failed', message: 'record
not found!' }));
        } else {
            res.end(JSON.stringify({ status: 'success', data: result[0]
}));
        }
    });
};

exports.getAllSingle = function (req, res) {
    const id = [req.params.id];
    const minutes = req.params.minutes;
    const now = new Date();
    var to = moment(now).format('YYYY-MM-DD HH:mm:ss');
    var from = moment(now).subtract(minutes, "minutes").toDate();
    from = moment(from).format('YYYY-MM-DD HH:mm:ss');

    var data = {};
    data.from = from;
    data.to = to;

    connection.query(GET_SENSOR, id, (err, result) => {
        if (err)
            throw err;
        if (result.length == 0) {
            res.end(JSON.stringify({ status: 'failed', message: 'record
not found!' }));
        } else {

            data.current = result[0];
            connection.query(GET_SINGLE_ALL_BY_MINUTES, [from, to, id],
(err, result) => {
                if (err)
                    throw err;

                data.log = result;
                res.end(JSON.stringify({ status: 'success', data: data
}));
            });

        }
    });
};

exports.insert = function (req, res) {
    const data = req.body;
    if (data.floor_id != undefined && data.room_id != undefined) {
        connection.query(INSERT_SENSOR, [data.floor_id, data.room_id],
(err, result) => {
            if (err)
                throw err;
            res.end(JSON.stringify(result));
        });
    } else {
        res.end(JSON.stringify({ status: 'failed', message: 'wrong fields'
}));
    }
};

exports.updateall = function (req, res) {
    const data = req.body;
    let queries = "";
    let datetime = moment(new Date()).format('YYYY-MM-DD HH:mm:ss');
    let values = [];

    data.forEach(row => {
        queries += connection.format(UPDATE_SENSORS, [row.smoke_level,
row.co2_level, datetime, row.id]);
        values.push([row.id, row.smoke_level, row.co2_level, datetime]);
    });
    console.log(queries);
    connection.query(queries, (err, result) => {
        if (err)
            throw err;

        connection.query(UPDATE_SENSORS_LOG, [values], (err, result) => {
            if (err)
                throw err;
            res.end(JSON.stringify({ status: 'success', message: 'All
Fields Updated' }));
        })


    });

};






//
==========================================================================
================================================
//
==========================================================================
================================================
// ================================================ ALERT SERVICE
===========================================================
//
==========================================================================
================================================
//
==========================================================================
================================================


//sent email when sensor come to danger zone -----------------------------
-----------------------------------
exports.sentWarningEmail = function (req, res, next) {

    var uEmail = "padulaguruge@gmail.com";
    var id = "10";
    var co2 = "5";
    var smoke = "6";

    UtilObj.sentEmailDanSenesors(uEmail, id, co2, smoke)

}
//sent SMS when sensor come to danger zone -------------------------------
---------------------------------------
exports.sentWarningSMS = function (req, res, next) {

    var id = "10";
    var co2 = "5";
    var smoke = "6";

    UtilObj.sentSMSAlert(id, co2, smoke)

}
//sent Call when sensor come to danger zone ------------------------------
-------------------------------------------
exports.sentCallAlert = function (req, res, next) {

    UtilObj.sentCallAlert()

}
```

This includes all functions which are related to sensors. After call this function it will interact with database and return response

Users controller

```
const UtilObj = require('../util/util')

//import User model
const { GET_ALL_USERS, GET_USER_FROM_ID, GET_USER_FROM_NAME,
GET_USER_FROM_EMAIL, INSERT_USER, GET_USER_FROM_EMAIL_PASSWORD,
GET_USER_FROM_ID_EMAIL, GET_USER_FROM_ID_PASSWORD, DELETE_USER } =
require('../models//users.model');
const connection = require('../../config/db.config');
// test functions
exports.test = function (req, res) {
    res.json({ val: 'Greetings from the Test controller!', des: '1424',
kk: '45455' });
};


// register user ----------------------------------------------------------
--------------------------------------------------------------------
exports.insert = function (req, res, next) {
    const data = req.body.admin;
    console.log(data);
    if (data.uName != undefined && data.uPass != undefined && data.uEmail
!= undefined && data.uCn != undefined) {
        connection.query(GET_USER_FROM_EMAIL, [data.uEmail], (err, result)
=> {
            if (result < 1) {
                connection.query(INSERT_USER, [data.uName, data.uPass,
data.uEmail, data.uCn], (err, result) => {
                    if (err)
                        throw err;
                    UtilObj.sentEmailforRegisterUsers(data.uEmail,
data.uPass, data.uName)
                    res.status(200).json({
                        message: 'Registation success'
                    })
                    next()
                });
            } else {
                res.status(201).json({
                    message: 'Alread Registerd'
                })
                next()
            }
        });
    } else {
        res.status(202).json({
            message: 'Registation Unsucess'
        })
        next()
    }
};

//user login  ------------------------------------------------------------
----------------------------------------------------------------
exports.login = function (req, res, next) {
    const data = req.body.admin;

    console.log(data);
    if (data.uPass != undefined && data.uEmail != undefined) {
        connection.query(GET_USER_FROM_EMAIL_PASSWORD, [data.uEmail,
data.uPass], (err, result) => {
            if (err)
                throw err;
            if (result.length == 1) {
                res.status(200).json({
                    message: 'Login Sucess',
                    user: result[0],
                    status: true
                })
            } else {
                res.status(201).json({
                    message: 'No data',
                    status: false

                })
            }
        });

    } else {
        res.status(202).json({
            message: 'Please sent valid details',
            status: false
        })
        next()
    }
};


//get all admins ---------------------------------------------------------
------------------------------------------------------------------
exports.getAllAdmins = function (req, res) {
    connection.query(GET_ALL_USERS, (err, result) => {
        if (err)
            throw err;
        res.status(200).json({
            result
        })
    });
};

//remove admin -----------------------------------------------------------
------------------------------------------------------------------
exports.removeAdmin = function (req, res, next) {
    const data = req.body

    console.log(data.id);
    console.log(data.password);


    if (data.id != undefined && data.password != undefined) {
        connection.query(GET_USER_FROM_ID_PASSWORD, [data.id,
data.password], (err, result) => {
            if (err)
                throw err;
            if (result.length == 1) {
                console.log("-----------------------");


                console.log("-----------------------");

                UtilObj.sentEmailforDeletedUsers(result[0].email,
result[0].name)
                connection.query(DELETE_USER, [data.id], (err, result) =>
{
                    if (err)
                        throw err;
                    res.status(200).json({
                        message: 'Delete User',
                    })
                });
            } else {
                res.status(201).json({
                    message: 'No data',
                })
            }
        });

    } else {
        res.status(202).json({
            message: 'Please sent valid details',
            status: false
        })
        next()
    }
}
```

This controller has all functions related to the user(admins). After registering or deleting admin this function sends appropriate emails for admins.

Credentials

DBCredentials

```
class DBCredentias {
    constructor() {
        this.CredentialsOne = {
            host: 'localhost',
            port: 3306,
            user: 'root',
            password: 'admin',
            dbname: 'alarmsystem'
        }
        this.CredentialsTwo = {
            host: 'localhost',
            port: 3306,
            user: 'root',
            password: '',
            dbname: 'alarmsystem'
        }

    }

}
var DB_Credentials = new DBCredentias();
module.exports = DB_Credentials;
```
This file include all SQL database credentials in our system

Models

Senssor

```
//Queries used in Sensors
module.exports = {
    GET_ALL :  "SELECT * FROM sensors",
    GET_SENSOR :  "SELECT * FROM sensors where id = ?",
    INSERT_SENSOR : "INSERT INTO sensors (floor_id, room_id ) VALUES (? ,?
)",
    UPDATE_SENSORS : "UPDATE sensors SET smoke_level = ? , co2_level = ? ,
updated_at = ? WHERE id = ?; ",
    UPDATE_SENSORS_LOG : "INSERT INTO sensors_log (sensor_id, smoke_level,
co2_level, datetime) VALUES ?;",
    GET_ALL_BY_MINUTES : "SELECT s.datetime , AVG(s.smoke_level) as
'average_smoke' , AVG(s.co2_level) as 'average_co2' FROM sensors_log s
WHERE s.datetime >= ? AND s.datetime <= ? GROUP BY s.datetime",
    GET_SINGLE_ALL_BY_MINUTES : "SELECT s.datetime , s.smoke_level
,s.co2_level FROM sensors_log s WHERE s.datetime >= ? AND s.datetime <= ?
AND s.sensor_id = ?",
};
```

This file include all sql queries which are related to sensors

Users

```
// add sql qury here

module.exports = {
    GET_ALL_USERS: "SELECT * FROM users",
    GET_USER_FROM_ID: "SELECT * FROM users where id = ?",
    GET_USER_FROM_ID_EMAIL: "SELECT email FROM users where id = ?",
    GET_USER_FROM_ID_PASSWORD: "SELECT * FROM users where id = ? and
password =?",
    GET_USER_FROM_NAME: "SELECT * FROM users where name = ?",
    GET_USER_FROM_EMAIL: "SELECT * FROM users where email = ?",
    GET_USER_FROM_EMAIL_PASSWORD: "SELECT * FROM users where email = ? AND
password = ?",
    INSERT_USER: "INSERT INTO users(name, password, email, phone ) VALUES
(? ,?, ?, ? )",
    DELETE_USER: "DELETE FROM users WHERE id = ?;",
}
```

This file include all sql queries which are related to user(admin)

Routes

Sensor

```
const express = require('express');
const router = express.Router();


const SensorsController = require('../controllers/sensors.controller');

//===========================================================================
==============================
//===================================  GET REQUEST
===============================================
//===========================================================================
==============================

//get all sensors data
router.get('/get', SensorsController.getAll);
router.get('/getall/:minutes', SensorsController.getAllForMinutes);
//get selected sensor data
router.get('/get/:id', SensorsController.get);
router.get('/getall/:id/:minutes', SensorsController.getAllSingle);
//===========================================================================
==============================
//===================================  POST REQUEST
==============================================
//===========================================================================
==============================

//insert new sensor
router.post('/insert', SensorsController.insert);

//update sensors data
router.post('/updateall', SensorsController.updateall);
```

```
// sent warning email when sensor come danget level
router.get('/warning/email', SensorsController.sentWarningEmail);
// sent warning email when sensor come danget level
router.get('/warning/sms', SensorsController.sentWarningSMS);
// sent warning call when sensor come danget level
router.get('/warning/call', SensorsController.sentCallAlert);



//export router
module.exports = router
```

This file include all rest api details which are related to sensors

Users

```
const express = require('express');
const router = express.Router();


const  UsersController  = require('../controllers/users.controller');

//=========================================================================
==============================
//===================================  GET REQUEST
===============================================
//=========================================================================
==============================

//test router
router.get('/test', UsersController.test);
router.get('/a/all', UsersController.getAllAdmins);
```

```
//========================================================================
==============================
//===================================  POST REQUEST
==============================================
//========================================================================
==============================

router.post('/register', UsersController.insert);
router.post('/login', UsersController.login);
router.post('/a/r', UsersController.removeAdmin);








//export router
module.exports = router
```

This file include all rest apis which are related to users (admins)

Util

```
const nodemailer = require('nodemailer');
const hbs = require('nodemailer-express-handlebars')

class Util {

    constructor() {

        this.sms = {
            accountSid: "XXXXXXXXXXXXXXXXXXXXX",
            authToken: "XXXXXXXXXXXXXXXXXXXX",

        };
    }

    //
==========================================================================
=================
    // ===========================Email Sent
==================================================
    //
==========================================================================
=================

    // sent email for regiter users
    sentEmailforRegisterUsers(uEmail, password, name) {
        let transport = nodemailer.createTransport({
            host: 'smtp.gmail.com',
            port: 587,
            secure: false,
            auth: {
                user: 'cassertmusic@gmail.com',
                pass: 'XXXXXXXXXXX'
            }
        });
        const handlebarOptions = {
            viewEngine: {
                extName: '.hbs',
                partialsDir: './views/',
                layoutsDir: './views/',
                defaultLayout: 'index.hbs',
            },
            viewPath: './views/',
            extName: '.hbs',

        };
        transport.use('compile', hbs(handlebarOptions));
        const message = {
            from: 'cassertmusic@gmail.com',
            to: uEmail,
            subject: 'Welcome to Sensors Managment System',
            template: 'index',
            context: {                    // <=
                password: password,
                name: name
            }
        };
        transport.sendMail(message, function (err, info) {
            if (err) {
                console.log(err)

            } else {
                console.log(info);
            }
        });
    }


    // sent email for deleted users
    sentEmailforDeletedUsers(uEmail, name) {
        let transport = nodemailer.createTransport({
            host: 'smtp.gmail.com',
            port: 587,
            secure: false,
            auth: {
                user: 'cassertmusic@gmail.com',
                pass: 'XXXXXXXXX'
            }
        });
        const handlebarOptions = {
            viewEngine: {
                extName: '.hbs',
                partialsDir: './views/',
                layoutsDir: './views/',
                defaultLayout: 'delete.hbs',
            },
            viewPath: './views/',
            extName: '.hbs',

        };
        transport.use('compile', hbs(handlebarOptions));
        const message = {
            from: 'cassertmusic@gmail.com',
            to: uEmail,
            subject: 'Bye to Sensors Managment System',
            template: 'delete',
            context: {                    // <=

                name: name
            }
        };
        transport.sendMail(message, function (err, info) {
            if (err) {
                console.log(err)

            } else {
                console.log(info);
            }
        });
    }

    // sent email when sensor come to danger
    sentEmailDanSenesors(uEmail, id, co2, smoke) {
        let transport = nodemailer.createTransport({
            host: 'smtp.gmail.com',
            port: 587,
            secure: false,
            auth: {
                user: 'cassertmusic@gmail.com',
                pass: 'XXXXXXX'
            }
        });
        const handlebarOptions = {
            viewEngine: {
                extName: '.hbs',
                partialsDir: './views/',
                layoutsDir: './views/',
                defaultLayout: 'sensor.hbs',
            },
            viewPath: './views/',
            extName: '.hbs',

        };
        transport.use('compile', hbs(handlebarOptions));
        const message = {
            from: 'cassertmusic@gmail.com',
            to: uEmail,
            subject: 'Sensor Warning',
            template: 'delete',
            context: {
                id: id,
                co2: co2,
                smoke: smoke
            }
        };
        transport.sendMail(message, function (err, info) {
            if (err) {
                console.log(err)

            } else {
                console.log(info);

            }
        });
    }
    //
==========================================================================
=================
    // ===========================SMS  Sent
=================================================
    //
==========================================================================
=================


    sentSMSAlert(id, co2, smoke) {
        var twilio = require('twilio');
        var client = new twilio(this.sms.accountSid, this.sms.authToken);
        client.messages.create({
              body: `This is warning from Sensor Managment System !   Sensor
ID :  ${id}  |  Co2 Level :  ${co2} |  Smoke Level :  ${smoke}    `,
              to: '+94717269086',  // Text this number
              from: '+12512921823'  // From a valid Twilio number
          }).then((message) => console.log(message.sid));
     }

     //
==========================================================================
=================
     // ===========================Call  Sent
=================================================
     //
==========================================================================
=================
     sentCallAlert() {
          var twilio = require('twilio');
          var client = new twilio(this.sms.accountSid, this.sms.authToken);
          client.calls
              .create({
                   url: 'http://demo.twilio.com/docs/voice.xml',
                   to: '+94717269086',
                   from: '+12512921823'
              })
              .then(call => console.log(call.sid));
     }

}

var UtilObj = new Util();
module.exports = UtilObj;
```

This file includes all util functions like SMS send and Email send. We use third party paid sms sent service and we use node mail service to send email.

DBconfig

```
var mysql = require('mysql');
```

```
// Db Credentials

const DB_Credentials = require ('../app/Credentials/DBCredentials');
// change your credentials
const DB_Connection = DB_Credentials.CredentialsOne;
const connection = mysql.createConnection({
    host: DB_Connection.host,
    port: DB_Connection.port,
    user: DB_Connection.user,
    password: DB_Connection.password,
    database: DB_Connection.dbname,
    multipleStatements:true ,
});

connection.connect((err) => {
    if (err)
        throw err;
    console.log('Connected!');
});
module.exports = connection;
```

This file includes the db config function. This function we use to connect our backend to mysql database

Server

```
const express = require('express');
const app = express();
const bodyParser = require('body-parser');
const cors = require('cors');

//define server running port
let port = 4000;


//=========================================================================
==============================
//===================================import routes
=================================================
//=========================================================================
==============================
 const UsersRoutes    = require('./app/routes/users.router');
 const SensorsRoutes    = require('./app/routes/sensors.router');

//========================================================================
==============================
//===================================import config files
===============================================
//========================================================================
==============================

// import db
const connection = require('./config/db.config');

//========================================================================
==============================
//===================================open apps services
===============================================
//========================================================================
==============================
app.use(cors());
app.use(bodyParser.json());
app.use(bodyParser.urlencoded({ extended: false }));

//========================================================================
==============================
//=================================== defines routes
===============================================
//========================================================================
==============================


app.use('/users', UsersRoutes );
app.use('/sensors', SensorsRoutes );

//========================================================================
==============================
//=================================== critical functions
===========================================
//========================================================================
==============================


// Connecting to the database

// open server
app.listen(port, () => {
    console.log('Server is up and running on port numner ' + port);
});
```

This is our server configure file. We run our server on port 4000.

RMIServer

Model Class - Admin.java

```java
package com.model;

import java.io.Serializable;

public class Admin implements Serializable {

    private int id;
    private String name;
    private String password;
    private String email;
    private String phone;

    public Admin() {
    }

    public int getId() {
        return id;
    }

    public void setId(int id) {
        this.id = id;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public String getPassword() {
        return password;
    }

    public void setPassword(String password) {
        this.password = password;
    }

    public String getEmail() {
        return email;
    }

    public void setEmail(String email) {
        this.email = email;
    }

    public String getPhone() {
        return phone;
    }

    public void setPhone(String phone) {
        this.phone = phone;
    }
}
```

Model Class - Sensor.java

```java
package com.model;

import java.io.Serializable;

public class Sensor implements Serializable {

    private int id;
    private String floor_id;
    private String room_id;
    private int smoke_level;
    private int co2_level;
    private String updated_at;
    private String status;

    public int getCo2_level() {
        return co2_level;
    }

    public void setCo2_level(int co2_level) {
        this.co2_level = co2_level;
    }

    public String getStatus() {
        return status;
    }

    public void setStatus(String status) {
        this.status = status;
    }

    public Sensor() { }
    public int getId() {
        return id;
    }

    public void setId(int id) {
        this.id = id;
    }

    public String getFloor_id() {
        return floor_id;
    }

    public void setFloor_id(String floor_id) {
        this.floor_id = floor_id;
    }

    public String getRoom_id() {
        return room_id;
    }

    public void setRoom_id(String room_id) {
        this.room_id = room_id;
    }

    public int getSmoke_level() {
        return smoke_level;
    }

    public void setSmoke_level(int smoke_level) {
        this.smoke_level = smoke_level;
    }


    public String getUpdated_at() {
        return updated_at;
    }

    public void setUpdated_at(String updated_at) {
        this.updated_at = updated_at;
    }
```

```
}
```

Model class - SensorLog.java

```
package com.model;

import java.io.Serializable;

public class SensorLog implements Serializable {
    private int id;
    private String datetime;
    private int smoke_level;
    private int co2_level;
    private double average_smoke;
    private double average_co2;

    public double getAverage_smoke() {
        return average_smoke;
    }

    public void setAverage_smoke(double average_smoke) {
        this.average_smoke = average_smoke;
    }

    public double getAverage_co2() {
        return average_co2;
    }

    public void setAverage_co2(double average_co2) {
        this.average_co2 = average_co2;
    }

    public SensorLog() {
    }

    public int getId() {
        return id;
    }

    public void setId(int id) {
        this.id = id;
    }
    public String getDatetime() {
        return datetime;
    }

    public void setDatetime(String datetime) {
        this.datetime = datetime;
    }

    public int getSmoke_level() {
        return smoke_level;
    }

    public void setSmoke_level(int smoke_level) {
        this.smoke_level = smoke_level;
    }

    public int getCo2_level() {
        return co2_level;
    }

    public void setCo2_level(int co2_level) {
        this.co2_level = co2_level;
    }
}
```

Server class - Main.java

```
package com.server;

import com.model.Admin;
import com.model.Sensor;
import com.service.AdminServiceProvider;
import com.service.IAdminService;
import com.service.ISensorService;
import com.service.SensorServiceProvider;

import java.io.IOException;
import java.rmi.Naming;
import java.rmi.RMISecurityManager;
import java.rmi.RemoteException;
import java.rmi.registry.LocateRegistry;
import java.rmi.registry.Registry;
import java.rmi.server.UnicastRemoteObject;
import java.util.Timer;
import java.util.TimerTask;


public class Main  {

    public static void main(String[] args) {
        System.out.println("Working Directory = " +
System.getProperty("user.dir"));
        System.setProperty("java.security.policy",
"file:./allowall.policy");

        try {
            if(System.getSecurityManager() == null ){
                System.setSecurityManager( new RMISecurityManager() );
            }

            final ISensorService sensorServiceProvider = new
SensorServiceProvider();
            final IAdminService adminServiceProvicer = new
AdminServiceProvider();

            Registry registry = LocateRegistry.createRegistry(4500);

            registry.rebind("SensorService", sensorServiceProvider);
            registry.rebind("AdminService", adminServiceProvicer );
            System.out.println("Server Started..");

            Timer timer = new Timer();
            timer.schedule(new TimerTask() {
                @Override
                public void run() {
                    try {
                        sensorServiceProvider.updateValues();
                    } catch (IOException e) {
                        System.out.println(e);
                    }
                }
            }, 0, 5000);




        } catch (Exception e ) {
            System.out.println( "r " + e);
        }

    }
}
```

Service class - AdminServiceProvider.java

```
package com.service;

import com.google.gson.*;
import com.model.Admin;
import com.model.SensorLog;
import com.util.HttpController;

import java.io.IOException;
import java.rmi.RemoteException;
import java.rmi.server.UnicastRemoteObject;
import java.util.ArrayList;
import java.util.LinkedList;

public class AdminServiceProvider extends UnicastRemoteObject implements
IAdminService{

    public AdminServiceProvider() throws RemoteException {
    }

    public ArrayList<Admin> getAllAdmins() throws RemoteException {
        Gson gson = new Gson();
        ArrayList<Admin> admin_list = new ArrayList<Admin>();
        String response = null;
        try {
            response = HttpController.Get( "users/a/all" );
        } catch (IOException e) {
            e.printStackTrace();
        }

        JsonObject result = new
JsonParser().parse(response).getAsJsonObject();
        JsonArray array = result.getAsJsonArray("result");

        for (JsonElement item : array) {
            Admin admin = gson.fromJson(item.toString(), Admin.class);
            admin.setPassword("");
            admin_list.add(admin);
        }
        return admin_list;
    }

    public void insertAdmin(Admin admin) throws RemoteException {
        String response = null;
        Gson gson = new Gson();
        JsonObject adminJson = new JsonObject();
        adminJson.addProperty("uName", admin.getName() );
        adminJson.addProperty("uPass", "default123" );
        adminJson.addProperty("uEmail", admin.getEmail() );
        adminJson.addProperty("uCn", admin.getPhone() );
        JsonObject finaladmin = new JsonObject();
        finaladmin.add("admin", adminJson );

        try {
            response = HttpController.Post( "users/register" ,
finaladmin.toString() );
        } catch (IOException e) {
            e.printStackTrace();
        }
        System.out.println(response);
    }
    public void editAdmin(Admin admin) throws RemoteException {
    }

    public void deleteAdmin(int id, String password) throws
RemoteException {
        String response = null;

        JsonObject adminJson = new JsonObject();
        adminJson.addProperty("id", id );
        adminJson.addProperty("password", password);

        try {
            response = HttpController.Post( "users/a/r" ,
adminJson.toString() );
        } catch (IOException e) {
            e.printStackTrace();
        }
        System.out.println(response);
    }

    public boolean loginUser(String email, String password) throws
RemoteException {
        String response = null ;
        JsonObject adminJson = new JsonObject();
        adminJson.addProperty("uEmail", email );
        adminJson.addProperty("uPass", password);

        JsonObject finaladmin = new JsonObject();
        finaladmin.add("admin", adminJson );
        try {
            response = HttpController.Post( "users/login" ,
finaladmin.toString() );
        } catch (IOException e) {
            e.printStackTrace();
        }
        JsonObject result = new
JsonParser().parse(response).getAsJsonObject();
        JsonPrimitive message = result.getAsJsonPrimitive("message");

        System.out.println(message.getAsString() );
        if(message.getAsString().equalsIgnoreCase("Login Sucess") ){
            return true;
        }else{
            return false;
        }
    }
}
```

Service class - IAdminService.java

```
package com.service;

import com.model.Admin;

import java.rmi.Remote;
import java.rmi.RemoteException;
import java.util.ArrayList;

public interface IAdminService extends Remote {
    public ArrayList<Admin> getAllAdmins() throws RemoteException;
    public void insertAdmin(Admin admin) throws RemoteException;
    public void editAdmin(Admin admin) throws RemoteException;
    public void deleteAdmin(int id, String password) throws
RemoteException;
    public boolean loginUser(String email ,String password) throws
RemoteException;
}
```

Service class - ISensorService.java

```java
package com.service;

import com.model.Sensor;
import com.model.SensorLog;

import java.io.IOException;
import java.rmi.Remote;
import java.rmi.RemoteException;
import java.util.ArrayList;
import java.util.LinkedList;

public interface ISensorService extends Remote {

    public ArrayList<Sensor> getAllSensorsCurrentData() throws
RemoteException;

    public LinkedList<SensorLog> getAllSensorsLog() throws
RemoteException;

    public Sensor getSensorCurrentData(int id) throws RemoteException;

    public LinkedList<SensorLog> getSensorLog(int id) throws
RemoteException;

    public String test(String msg) throws RemoteException;

    public void insertSensor(Sensor sensor) throws RemoteException;

    public void editSensor(Sensor sensor) throws RemoteException;

    public void deleteSensor(int id) throws RemoteException;

    public void updateValues() throws IOException;
}
```

Service class - SensorServiceProvider.java

```java
package com.service;
```

```java
import com.google.gson.*;
import com.model.Sensor;
import com.model.SensorLog;
import com.util.HttpController;

import java.io.IOException;
import java.rmi.RemoteException;
import java.rmi.server.UnicastRemoteObject;
import java.text.DateFormat;
import java.text.ParseException;
import java.text.SimpleDateFormat;
import java.util.ArrayList;
import java.util.Calendar;
import java.util.LinkedList;

public class SensorServiceProvider extends UnicastRemoteObject implements
ISensorService{

    private  ArrayList<Sensor> sensorsdata ;
    private LinkedList<SensorLog> sensorslog;

    public SensorServiceProvider() throws RemoteException {
        this.sensorsdata = new ArrayList<Sensor>();
        this.sensorslog = new LinkedList<SensorLog>();
    }

    public ArrayList<Sensor> getAllSensorsCurrentData() throws
RemoteException {
        return sensorsdata;
    }

    public LinkedList<SensorLog> getAllSensorsLog() throws RemoteException
{
        return sensorslog;
    }

    public Sensor getSensorCurrentData(int id) throws RemoteException {
        for (Sensor sensor : sensorsdata) {
            if(sensor.getId() == id ){
                return sensor;
            }
        }
        return null;
    }

    public LinkedList<SensorLog> getSensorLog(int id) throws
RemoteException {
        System.out.println("called");
        LinkedList<SensorLog> single_sensorslog = new
LinkedList<SensorLog>();

        Gson gson = new Gson();
        DateFormat time = new SimpleDateFormat("HH:mm:ss");

        String response = null;
        try {
            response = HttpController.Get( "sensors/getall/"+ id +"/1" );
        } catch (IOException e) {
            e.printStackTrace();
        }
        JsonObject result = new
JsonParser().parse(response).getAsJsonObject();
        JsonObject data = result.getAsJsonObject("data");
        JsonArray log_array = data.getAsJsonArray("log");

        for (JsonElement  item : log_array) {
            SensorLog log = gson.fromJson( item.toString()  ,
SensorLog.class );

            Calendar cal =
javax.xml.bind.DatatypeConverter.parseDateTime(log.getDatetime());
            String timex  =  time.format( cal.getTime() );
            log.setDatetime(timex);

            single_sensorslog.add(log);
        }

        if(single_sensorslog.size() > 16){
            single_sensorslog =  new
LinkedList<SensorLog>(single_sensorslog.subList( single_sensorslog.size()
- 10 , single_sensorslog.size() ));

        }
        return single_sensorslog;
    }

    public String test(String msg) throws RemoteException {
        return "Server : " + msg;
    }

    public void insertSensor(Sensor sensor) throws RemoteException{
        String response = null;
        Gson gson = new Gson();

        try {
            response = HttpController.Post( "sensors/insert" ,
gson.toJson(sensor) );
            updateValues();
        } catch (IOException e) {
            e.printStackTrace();
        }
        System.out.println(response);
    }

    public void editSensor(Sensor sensor) throws RemoteException{
        String response = null;
        Gson gson = new Gson();

        try {
            response = HttpController.Post( "sensors/update" ,
gson.toJson(sensor) );
            updateValues();
        } catch (IOException e) {
            e.printStackTrace();
        }
        System.out.println(response);

    }
    public void deleteSensor(int id) throws RemoteException{
        String response = null;
        Gson gson = new Gson();

        try {
            response = HttpController.Post( "sensors/delete/"+ id , "" );
            updateValues();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    public  void updateValues() throws IOException {

        Gson gson = new Gson();

        DateFormat time = new SimpleDateFormat("HH:mm:ss");

        String response =  HttpController.Get( "sensors/getall/1" );
        JsonObject result = new
JsonParser().parse(response).getAsJsonObject();

        JsonArray current_array = result.getAsJsonArray("current");
        JsonArray log_array = result.getAsJsonArray("log");

        sensorsdata.clear();
        sensorslog.clear();
        System.out.println("------------------------------");
        for (JsonElement  item : log_array) {
            SensorLog log = gson.fromJson( item.toString()  ,
SensorLog.class );

            Calendar cal =
javax.xml.bind.DatatypeConverter.parseDateTime(log.getDatetime());
            String timex  =  time.format( cal.getTime() );

            log.setDatetime(timex);
            sensorslog.add(log);
        }
        if(sensorslog.size() > 16){
            sensorslog =  new LinkedList<SensorLog>(sensorslog.subList(
sensorslog.size() - 10 , sensorslog.size() ));

        }

        for (JsonElement  item : current_array) {
            Sensor sensor = gson.fromJson( item.toString() , Sensor.class
);
            sensor.setStatus( returnStatus(sensor.getCo2_level() ,
sensor.getSmoke_level() ));
            sensorsdata.add(sensor);
        }
    }

    private String returnStatus(int c02 , int smoke){
        float avg = (float) c02 + smoke / 2;
        if(avg >= 0 && avg <= 2 ){
            return "Normal";
        }else if( avg >= 3 && avg <= 4 ){
            return "Average";
        }else if( avg >= 5 && avg <= 10 ){
            return "Danger";
        }
        return "None";
    }
}
```

Util - HttpController.java

```
package com.util;

import org.apache.http.HttpEntity;
import org.apache.http.HttpResponse;
import org.apache.http.client.ClientProtocolException;
import org.apache.http.client.HttpClient;
import org.apache.http.client.methods.HttpGet;
import org.apache.http.client.methods.HttpPost;
import org.apache.http.entity.StringEntity;
import org.apache.http.impl.client.HttpClientBuilder;
import org.apache.http.util.EntityUtils;

import java.io.IOException;

public class HttpController  {
    public static final String BASE_URL  = "http://localhost:4000/";

    public static String Post(String url , String body) throws
ClientProtocolException, IOException {
        String final_url = BASE_URL + url;
        HttpClient httpClient    = HttpClientBuilder.create().build();
        HttpPost post           = new HttpPost(final_url);
        StringEntity postingString = new StringEntity(body);

        post.setEntity(postingString);
        post.setHeader("Content-type", "application/json");

        HttpResponse response = httpClient.execute(post);

        HttpEntity entity = response.getEntity();
        return EntityUtils.toString(entity);
    }

    public static String Get(String url ) throws ClientProtocolException,
IOException {
        String final_url = BASE_URL + url;
        HttpClient httpClient    = HttpClientBuilder.create().build();
        HttpGet get           = new HttpGet(final_url);
        get.setHeader("Content-type", "application/json");
        HttpResponse response = httpClient.execute(get);

        HttpEntity entity = response.getEntity();
        return EntityUtils.toString(entity);
    }
}
```

SensorDesktopClient

Model class - Admin.java

```java
package com.model;

import java.io.Serializable;

public class Admin implements Serializable {

    private int id;
    private String name;
    private String password;
    private String email;
    private String phone;

    public Admin() {
    }

    public int getId() {
        return id;
    }

    public void setId(int id) {
        this.id = id;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public String getPassword() {
        return password;
    }

    public void setPassword(String password) {
        this.password = password;
    }
    public String getEmail() {
        return email;
    }

    public void setEmail(String email) {
        this.email = email;
    }

    public String getPhone() {
        return phone;
    }

    public void setPhone(String phone) {
        this.phone = phone;
    }
}
```

Model class - Sensor.java

```
package com.model;

import java.io.Serializable;

public class Sensor implements Serializable {

    private int id;
    private String floor_id;
    private String room_id;
    private int smoke_level;
    private int co2_level;
    private String updated_at;
    private String status;

    public int getCo2_level() {
        return co2_level;
    }

    public void setCo2_level(int co2_level) {
        this.co2_level = co2_level;
    }
```

```
public String getStatus() {
    return status;
}

public void setStatus(String status) {
    this.status = status;
}

public Sensor() { }

public int getId() {
    return id;
}

public void setId(int id) {
    this.id = id;
}

public String getFloor_id() {
    return floor_id;
}

public void setFloor_id(String floor_id) {
    this.floor_id = floor_id;
}

public String getRoom_id() {
    return room_id;
}

public void setRoom_id(String room_id) {
    this.room_id = room_id;
}

public int getSmoke_level() {
    return smoke_level;
}

public void setSmoke_level(int smoke_level) {
        this.smoke_level = smoke_level;
    }


    public String getUpdated_at() {
        return updated_at;
    }

    public void setUpdated_at(String updated_at) {
        this.updated_at = updated_at;
    }
}
```

com.model class - SensorLog.java

```
package com.model;

import java.io.Serializable;

public class SensorLog implements Serializable {
    private int id;
    private String datetime;
    private int smoke_level;
    private int co2_level;
    private double average_smoke;
    private double average_co2;

    public double getAverage_smoke() {
        return average_smoke;
    }

    public void setAverage_smoke(double average_smoke) {
        this.average_smoke = average_smoke;
    }

    public double getAverage_co2() {
        return average_co2;
    }

    public void setAverage_co2(double average_co2) {
        this.average_co2 = average_co2;
    }

    public SensorLog() {
    }

    public int getId() {
        return id;
    }

    public void setId(int id) {
        this.id = id;
    }

    public String getDatetime() {
        return datetime;
    }

    public void setDatetime(String datetime) {
        this.datetime = datetime;
    }

    public int getSmoke_level() {
        return smoke_level;
    }

    public void setSmoke_level(int smoke_level) {
        this.smoke_level = smoke_level;
    }

    public int getCo2_level() {
        return co2_level;
    }

    public void setCo2_level(int co2_level) {
        this.co2_level = co2_level;
    }
}
```

com.model class - Users.java

```java
package com.model;

public class User {

    String email;
    private static User instance;

    public String getEmail() {
        return email;
    }

    public void setEmail(String email) {
        this.email = email;
    }

    public synchronized static User getInstance(){
        if(instance == null){
            instance = new User();
        }
        return instance;
    }

    public static void resetUser(){
        instance = null;
    }

}
```

com.service - IAdminService.java

```java
package com.service;

import com.model.Admin;

import java.rmi.Remote;
import java.rmi.RemoteException;
import java.util.ArrayList;

public interface IAdminService extends Remote {
    public ArrayList<Admin> getAllAdmins() throws RemoteException;
    public void insertAdmin(Admin admin) throws RemoteException;
    public void editAdmin(Admin admin) throws RemoteException;
    public void deleteAdmin(int id, String password) throws
RemoteException;
    public boolean loginUser(String email ,String password) throws
RemoteException;
}
```

com.service - ISensorService.java

```
package com.service;



import com.model.Sensor;
import com.model.SensorLog;

import java.io.IOException;
import java.rmi.Remote;
import java.rmi.RemoteException;
import java.util.ArrayList;
import java.util.LinkedList;

public interface ISensorService extends Remote {

    public ArrayList<Sensor> getAllSensorsCurrentData() throws
RemoteException;
    public LinkedList<SensorLog> getAllSensorsLog() throws
RemoteException;
    public Sensor getSensorCurrentData(int id) throws RemoteException;
    public LinkedList<SensorLog> getSensorLog(int id) throws
RemoteException;
```

```
    public String test(String msg) throws RemoteException;
    public void insertSensor(Sensor sensor) throws RemoteException;
    public void editSensor(Sensor sensor) throws RemoteException;
    public void deleteSensor(int id) throws RemoteException;
    public void updateValues() throws IOException;
}
```

Controller - AdminsController.java

```
package controllers;

import com.model.Admin;
import com.model.SensorLog;
import com.service.IAdminService;
import com.service.ISensorService;
import javafx.application.Platform;
import javafx.beans.value.ChangeListener;
import javafx.beans.value.ObservableValue;
import javafx.collections.FXCollections;
import javafx.event.ActionEvent;
import javafx.fxml.FXML;
import javafx.fxml.Initializable;
import javafx.scene.control.*;
import javafx.scene.control.cell.PropertyValueFactory;
import javafx.scene.input.InputMethodEvent;
import javafx.stage.StageStyle;

import java.io.IOException;
import java.net.MalformedURLException;
import java.net.URL;
import java.rmi.Naming;
import java.rmi.NotBoundException;
import java.rmi.RMISecurityManager;
import java.rmi.RemoteException;
import java.util.ArrayList;
import java.util.ResourceBundle;
import java.util.TimerTask;

public class AdminsController implements Initializable {
    @FXML
    private Button mode_btn;

    @FXML
    private TextField id_input;

    @FXML
    private TextField password_input;

    @FXML
    private TextField username_input;

    @FXML
    private TextField email_input;

    @FXML
    private TextField phone_input;

    @FXML
    private Button insert;

    @FXML
    private Button edit;

    @FXML
    private Button delete;

    @FXML
    private TableView<Admin> admin_table;

    @FXML
    private TableColumn<Admin,Integer> id;

    @FXML
    private TableColumn<Admin,String> username;

    @FXML
    private TableColumn<Admin,String> email;

    @FXML
    private TableColumn<Admin,String> phone;

    private String mode = "Register";
    private IAdminService service = null;
    private ArrayList<Admin> admins_array = new ArrayList<Admin>();
    private Admin activeItem;

    @Override
    public void initialize(URL location, ResourceBundle resources) {

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

        try {

            if(System.getSecurityManager() == null ){
                System.setSecurityManager( new RMISecurityManager() );
            }
            service = (IAdminService)
Naming.lookup("//localhost:4500/AdminService");
            admins_array = service.getAllAdmins();
            setTableData();

        } catch (NotBoundException e) {
            e.printStackTrace();
        } catch (MalformedURLException e) {
            e.printStackTrace();
        } catch (RemoteException e) {
            e.printStackTrace();
        }
    }

    public void onClickInsert(javafx.event.ActionEvent actionEvent) throws
RemoteException {
       if( email_input.getText().isEmpty() ||
phone_input.getText().isEmpty() || username_input.getText().isEmpty()){
           Alert alert = new Alert(Alert.AlertType.WARNING);
           alert.initStyle(StageStyle.UTILITY);
```

```
            alert.setTitle("Warning");
            alert.setContentText("All Fields Required");
            alert.showAndWait();
        }else{
            Admin admin = new Admin();
            admin.setName(username_input.getText().trim());
            admin.setPhone( phone_input.getText().trim());
            admin.setEmail( email_input.getText() );
            service.insertAdmin(admin);
            admins_array = service.getAllAdmins();
            setTableData();
        }

    }



    public void onClickDelete(javafx.event.ActionEvent actionEvent) throws
RemoteException {
        if(!id_input.getText().isEmpty()){
            if(password_input.getText().isEmpty()){
                Alert alert = new Alert(Alert.AlertType.WARNING);
                alert.initStyle(StageStyle.UTILITY);
                alert.setTitle("Warning");
                alert.setContentText("Current password is required to
remove an admin");
                alert.showAndWait();
            }else{
                service.deleteAdmin(
Integer.parseInt(id_input.getText().trim() ) ,
password_input.getText().trim() );
                admins_array = service.getAllAdmins();
                setTableData();
            }
        }
    }

    public void onChangeMode(ActionEvent actionEvent) {
        if(this.mode == "Register"){
            changeMode("Edit");
        }else{
            changeMode("Register");
        }
    }

    public void changeMode(String mode){
        this.mode = mode;
       if(mode == "Register"){
           email_input.setText("");
           phone_input.setText("");
           username_input.setText("");
           password_input.setText("");
           id_input.setText("");
           mode_btn.setText("Change to Edit");
           insert.setVisible(true);
           edit.setVisible(false);
           delete.setVisible(false);
           id_input.setPromptText("Id Will be Auto Genarated");
           id_input.setDisable(true);
           password_input.setDisable(true);
           email_input.setDisable(false);
           username_input.setDisable(false);
           phone_input.setDisable(false);
       }else if( mode == "Edit"){
           mode_btn.setText("Change to Register");
           insert.setVisible(false);
           edit.setVisible(false);
           delete.setVisible(true);
           id_input.setPromptText("Enter ID to Edit/Delete");
           id_input.setDisable(false);
           password_input.setDisable(false);
           email_input.setDisable(true);
           username_input.setDisable(true);
           phone_input.setDisable(true);

       }
    }

    public void onChangeId(){
        id_input.textProperty().addListener(new ChangeListener<String>() {
            @Override
            public void changed(ObservableValue<? extends String>
observable,
                                String oldValue, String newValue) {

                if(newValue.length() > 0 ){
                    if( isNumeric(newValue) ){
                        for (Admin admin : admins_array) {
                            if(admin.getId() ==
Integer.parseInt(newValue)){
                                activeItem = admin;
                                setActiveItem(admin);
                            }
                        }
                    }
                }else{
                    email_input.setText("");
                    phone_input.setText("");
                    username_input.setText("");
                    password_input.setText("");
                }
            }
        });
    }



    public void setTableData(){
        admin_table.getItems().removeAll();

        id.setCellValueFactory( new PropertyValueFactory<Admin,
Integer>("id"));
        username.setCellValueFactory( new PropertyValueFactory<Admin,
String>("name"));
        email.setCellValueFactory( new PropertyValueFactory<Admin,
String>("email"));
        phone.setCellValueFactory( new PropertyValueFactory<Admin,
String>("phone"));
```

```
admin_table.setItems(FXCollections.observableArrayList(admins_array));

        id.setStyle("-fx-size: 35.0px;-fx-min-height: 45.0px;");
        username.setStyle("-fx-size: 35.0px;-fx-min-height: 45.0px;");
        email.setStyle("-fx-size: 35.0px;-fx-min-height: 45.0px;");
        phone.setStyle("-fx-size: 35.0px;-fx-min-height: 45.0px;");
    }

    public static boolean isNumeric(String str) {
        return str.matches("-?\\d+(\\.\\d+)?");  //match a number with
optional '-' and decimal.
    }

    public void setActiveItem(Admin admin){
        if(mode == "Register"){
            email_input.setText("");
            phone_input.setText("");
            username_input.setText("");
            password_input.setText("");

        }else if( mode == "Edit"){
            email_input.setText(admin.getEmail());
            phone_input.setText(admin.getPhone());
            username_input.setText(admin.getName());
            password_input.setText("");
        }
    }

    public void onClickEdit(ActionEvent actionEvent) {
    }
}
```

Controller - DashboardController.java

```
package controllers;


import com.model.SensorLog;
import com.service.ISensorService;
import javafx.application.Platform;
```

```java
import javafx.collections.FXCollections;
import javafx.fxml.FXML;
import javafx.fxml.Initializable;
import com.model.Sensor;
import javafx.scene.chart.*;
import javafx.scene.control.TableColumn;
import javafx.scene.control.TableView;
import javafx.scene.control.cell.PropertyValueFactory;

import java.io.IOException;
import java.net.MalformedURLException;
import java.net.URL;
import java.rmi.Naming;
import java.rmi.NotBoundException;
import java.rmi.RMISecurityManager;
import java.rmi.RemoteException;
import java.util.*;

public class DashboardController implements Initializable {

    @FXML
    private AreaChart<String,Number> livechart;

    @FXML
    private CategoryAxis x;

    @FXML
    private NumberAxis y;

    @FXML
    private TableView<Sensor> sensortable;

    @FXML
    private TableColumn<Sensor,Integer> id;

    @FXML
    private TableColumn<Sensor,String> floor_no;

    @FXML
    private TableColumn<Sensor, String> room_no;
    @FXML
    private TableColumn<Sensor, Integer> co2_level;

    @FXML
    private TableColumn<Sensor,Integer> smoke_level;

    @FXML
    private TableColumn<Sensor,String> status;

    ArrayList<Sensor> sensorsData = new ArrayList<Sensor>();
    LinkedList<SensorLog> sensorLogs = new LinkedList<SensorLog>();


    @Override
    public void initialize(URL location, ResourceBundle resources) {

    }

    public void setLivechart(){
        XYChart.Series smoke = new XYChart.Series();
        XYChart.Series co2 = new XYChart.Series();
        co2.setName("co2");
        smoke.setName("smoke");

        for (int i = 0; i < sensorLogs.size(); i++) {
            co2.getData().add(new XYChart.Data(
sensorLogs.get(i).getDatetime() , sensorLogs.get(i).getAverage_co2() ));
            smoke.getData().add(new XYChart.Data(
sensorLogs.get(i).getDatetime(), sensorLogs.get(i).getAverage_smoke()));
        }
        livechart.getData().clear();
        livechart.layout();
        livechart.getData().addAll(co2, smoke);
        livechart.setLegendVisible(false);
    }

    public void setTableData(){
        sensortable.getItems().removeAll();
        id.setCellValueFactory( new PropertyValueFactory<Sensor,
Integer>("id"));
        floor_no.setCellValueFactory( new PropertyValueFactory<Sensor,
String>("floor_id"));
        room_no.setCellValueFactory( new PropertyValueFactory<Sensor,
String>("room_id"));
        co2_level.setCellValueFactory( new PropertyValueFactory<Sensor,
Integer>("co2_level"));
        smoke_level.setCellValueFactory( new PropertyValueFactory<Sensor,
Integer>("smoke_level"));
        status.setCellValueFactory( new PropertyValueFactory<Sensor,
String>("status"));


sensortable.setItems(FXCollections.observableArrayList(sensorsData));

        id.setStyle("-fx-size: 35.0px;-fx-min-height: 45.0px;");
        floor_no.setStyle("-fx-size: 35.0px;-fx-min-height: 45.0px;");
        room_no.setStyle("-fx-size: 35.0px;-fx-min-height: 45.0px;");
        co2_level.setStyle("-fx-size: 35.0px;-fx-min-height: 45.0px;");
        smoke_level.setStyle("-fx-size: 35.0px;-fx-min-height: 45.0px;");
    }

    public void init(Timer timer){
        ISensorService service = null;
        System.setProperty("java.security.policy",
"file:allowall.policy");

        try {

            if(System.getSecurityManager() == null ){
                System.setSecurityManager( new RMISecurityManager() );
            }

            service = (ISensorService)
Naming.lookup("//localhost:4500/SensorService");

            final ISensorService finalService = service;
            timer.schedule(new TimerTask() {
                @Override
                public void run() {
                    try {
                        sensorsData =
finalService.getAllSensorsCurrentData();
                        sensorLogs = finalService.getAllSensorsLog();
                        System.out.println(sensorLogs.size());
                        setTableData();
                        Platform.runLater(new Runnable() {
                            @Override
                            public void run() {
                                setLivechart();
                            }
                        });

                    } catch (IOException e) {
                        System.out.println(e);
                    }
                }
            }, 0, 5000);

        } catch (NotBoundException e) {
            e.printStackTrace();
        } catch (MalformedURLException e) {
            e.printStackTrace();
        } catch (RemoteException e) {
            e.printStackTrace();
        }

    }

}
```

Controller - LoginController.java

```
package controllers;

import com.model.User;
import com.service.IAdminService;
```

```java
import com.service.ISensorService;
import javafx.application.Platform;
import javafx.fxml.FXML;
import javafx.fxml.FXMLLoader;
import javafx.fxml.Initializable;
import javafx.scene.Node;
import javafx.scene.Parent;
import javafx.scene.Scene;
import javafx.scene.control.Label;
import javafx.scene.control.PasswordField;
import javafx.scene.control.TextField;
import javafx.stage.Stage;

import java.awt.event.ActionEvent;
import java.io.IOException;
import java.net.MalformedURLException;
import java.net.URL;
import java.rmi.Naming;
import java.rmi.NotBoundException;
import java.rmi.RMISecurityManager;
import java.rmi.RemoteException;
import java.util.ResourceBundle;
import java.util.TimerTask;

public class LoginController  implements Initializable {

    @FXML
    private Label lblUssername;

    @FXML
    private Label lblPassword;

    @FXML
    private Label error;

    @FXML
    private TextField txtUsername;

    @FXML
    private PasswordField txtPassword;
    @Override
    public void initialize(URL location, ResourceBundle resources) {

    }

    public void onClickSubmit(javafx.event.ActionEvent actionEvent) {

        if(txtUsername.getText().isEmpty() ||
txtPassword.getText().isEmpty() ) {

        }else{
            IAdminService service = null;
            System.setProperty("java.security.policy",
"file:allowall.policy");
            try {

                if(System.getSecurityManager() == null ){
                    System.setSecurityManager( new RMISecurityManager() );
                }

                service = (IAdminService)
Naming.lookup("//localhost:4500/AdminService");
                boolean result = service.loginUser(txtUsername.getText() ,
txtPassword.getText());
                System.out.println(result);

                if(result){
                    error.setText("");
                    User user = User.getInstance();
                    user.setEmail(txtUsername.getText());
                    Node node = (Node) actionEvent.getSource();
                    Stage stage = (Stage) node.getScene().getWindow();
                    stage.close();

                    Scene scene = new Scene((Parent)
FXMLLoader.load(getClass().getResource("/pages/index.fxml")));
                    stage.setScene(scene);
                    stage.show();
                }else{
                    error.setText("Login Failed !");
                }
            } catch (NotBoundException e) {
                e.printStackTrace();
            } catch (MalformedURLException e) {
                e.printStackTrace();
            } catch (RemoteException e) {
                e.printStackTrace();
            }catch (IOException e) {
            e.printStackTrace();
            }
        }
    }
}
```

Controller - MainController.java

```
package controllers;

import com.model.User;
import javafx.event.EventHandler;
import javafx.fxml.FXML;
import javafx.fxml.FXMLLoader;
import javafx.fxml.Initializable;

import java.io.IOException;
import java.net.URL;
import java.util.ResourceBundle;
import javafx.fxml.Initializable;
import javafx.scene.Parent;
import javafx.scene.control.Label;
import javafx.scene.input.MouseEvent;
import javafx.scene.layout.BorderPane;

import javax.xml.soap.Text;
import java.net.URL;
import java.util.ResourceBundle;
import java.util.Timer;

public class MainController implements Initializable {
```

```
Timer timer = new Timer();
@FXML
private BorderPane borderpane;
@FXML
private BorderPane mainpanel;

@FXML
private Label emailadmin;

public void initialize(URL location, ResourceBundle resources) {
    System.out.println("from dashboard controller");
    loadPage("dashboard" , mainpanel );

    emailadmin.setText(User.getInstance().getEmail());


}

@FXML
void onAdminClicked(MouseEvent event) {
    System.out.println("Admin Clicked");
    loadPage("admins" , mainpanel );
}

@FXML
void onDashboardClicked(MouseEvent event) {
    System.out.println("Dashboard Clicked");
    loadPage("dashboard" , mainpanel );
}

@FXML
void onSensorClicked(MouseEvent event) {
    System.out.println("Sensor Clicked");
    loadPage("sensors" , mainpanel );
}

public void loadPage(String page, BorderPane bp) {
    try {
        timer.cancel();
```

```
            FXMLLoader fxmlLoader = new
FXMLLoader(getClass().getResource("/pages/" + page + ".fxml"));
            Parent root = (Parent)fxmlLoader.load();

            if(page == "dashboard"){
                timer = new Timer();
                DashboardController dashboardController =
fxmlLoader.<DashboardController>getController();
                dashboardController.init(timer);
            }
            if(page == "sensors"){
                timer = new Timer();
                SensorsController sensorsController =
fxmlLoader.<SensorsController>getController();
                sensorsController.init(timer);
            }

            bp.setCenter(root);
        } catch (IOException ex) {
            System.out.println("Exception : " +ex);
        }

    }

    public void onManageSensorClicked(MouseEvent mouseEvent) {
        System.out.println("Manage Sensors Clicked");
        loadPage("manage_sensors" , mainpanel );
    }
}
```

Controller - ManageSensorsController.java

```
package controllers;

import com.model.Admin;
import com.model.Sensor;
import com.service.IAdminService;
import com.service.ISensorService;
import javafx.beans.value.ChangeListener;
import javafx.beans.value.ObservableValue;
```

```java
import javafx.collections.FXCollections;
import javafx.event.ActionEvent;
import javafx.fxml.FXML;
import javafx.fxml.Initializable;
import javafx.scene.control.*;
import javafx.scene.control.cell.PropertyValueFactory;
import javafx.stage.StageStyle;

import java.net.MalformedURLException;
import java.net.URL;
import java.rmi.Naming;
import java.rmi.NotBoundException;
import java.rmi.RMISecurityManager;
import java.rmi.RemoteException;
import java.text.DateFormat;
import java.text.SimpleDateFormat;
import java.util.ArrayList;
import java.util.Calendar;
import java.util.ResourceBundle;

public class ManageSensorsController implements Initializable {

    @FXML
    private Button mode_btn;

    @FXML
    private TextField id_input;

    @FXML
    private TextField floor_id_input;

    @FXML
    private TextField room_id_input;

    @FXML
    private Button insert;

    @FXML
    private Button edit;
    @FXML
    private Button delete;

    @FXML
    private TableView<Sensor> sensor_table;

    @FXML
    private TableColumn<Sensor, Integer> id;

    @FXML
    private TableColumn<Sensor, String> floor_id;

    @FXML
    private TableColumn<Sensor, String> room_id;

    @FXML
    private TableColumn<Sensor, String> updated_time;

    @FXML
    private TableColumn<Sensor, String> updated_time_s;

    private String mode = "Add";
    private ISensorService service = null;
    private ArrayList<Sensor> sensors_array = new ArrayList<Sensor>();
    private Sensor activeItem;

    @Override
    public void initialize(URL location, ResourceBundle resources) {
        changeMode("Add");
        onChangeId();

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

        try {

            if(System.getSecurityManager() == null ){
                System.setSecurityManager( new RMISecurityManager() );
            }
            service  = (ISensorService)
Naming.lookup("//localhost:4500/SensorService");
            sensors_array = service.getAllSensorsCurrentData();
            setTableData();

        } catch (NotBoundException e) {
            e.printStackTrace();
        } catch (MalformedURLException e) {
            e.printStackTrace();
        } catch (RemoteException e) {
            e.printStackTrace();
        }
    }

    private void setTableData() {
        DateFormat time = new SimpleDateFormat("HH:mm:ss");
        DateFormat date = new SimpleDateFormat("EEEEE dd MMMMM yyyy");
        for (Sensor sensor : sensors_array) {
            Calendar cal =
javax.xml.bind.DatatypeConverter.parseDateTime(sensor.getUpdated_at());
            sensor.setStatus(time.format( cal.getTime() ));
            sensor.setUpdated_at( date.format(cal.getTime()));
        }
        sensor_table.getItems().removeAll();

        id.setCellValueFactory( new PropertyValueFactory<Sensor,
Integer>("id"));
        floor_id.setCellValueFactory( new PropertyValueFactory<Sensor,
String>("floor_id"));
        room_id.setCellValueFactory( new PropertyValueFactory<Sensor,
String>("room_id"));
        updated_time.setCellValueFactory( new PropertyValueFactory<Sensor,
String>("updated_at"));
        updated_time_s.setCellValueFactory( new
PropertyValueFactory<Sensor, String>("status"));


sensor_table.setItems(FXCollections.observableArrayList(sensors_array));

        id.setStyle("-fx-size: 35.0px;-fx-min-height: 45.0px;");
        floor_id.setStyle("-fx-size: 35.0px;-fx-min-height: 45.0px;");
        room_id.setStyle("-fx-size: 35.0px;-fx-min-height: 45.0px;");
        updated_time.setStyle("-fx-size: 35.0px;-fx-min-height: 45.0px;");
        updated_time_s.setStyle("-fx-size: 35.0px;-fx-min-height:
45.0px;");
    }

    public void onChangeMode(ActionEvent actionEvent) {
        if(this.mode == "Add"){
            changeMode("Edit");
        }else{
            changeMode("Add");
        }
    }

    public void onClickInsert(ActionEvent actionEvent) throws
RemoteException {
        if( floor_id_input.getText().isEmpty() ||
room_id_input.getText().isEmpty() ){
            Alert alert = new Alert(Alert.AlertType.WARNING);
            alert.initStyle(StageStyle.UTILITY);
            alert.setTitle("Warning");
            alert.setContentText("All Fields Required");
            alert.showAndWait();
        }else{

            Sensor new_sensor = new Sensor();
            new_sensor.setFloor_id( floor_id_input.getText().trim());
            new_sensor.setRoom_id( room_id_input.getText().trim());
            service.insertSensor(new_sensor);
            sensors_array = service.getAllSensorsCurrentData();
            setTableData();
            floor_id_input.setText("");
            room_id_input.setText("");
        }
    }

    public void onClickEdit(ActionEvent actionEvent) throws
RemoteException {
        if(!id_input.getText().isEmpty()) {
            if (floor_id_input.getText().isEmpty() ||
room_id_input.getText().isEmpty()) {
                Alert alert = new Alert(Alert.AlertType.WARNING);
                alert.initStyle(StageStyle.UTILITY);
                alert.setTitle("Warning");
                alert.setContentText("All Fields Required");
                alert.showAndWait();
            } else {
                Sensor new_sensor = new Sensor();
                new_sensor.setFloor_id(floor_id_input.getText().trim());
                new_sensor.setRoom_id(room_id_input.getText().trim());
                new_sensor.setId( Integer.parseInt(
id_input.getText().trim()));
                service.editSensor(new_sensor);
                sensors_array = service.getAllSensorsCurrentData();
                setTableData();

            }
        }

    }

    public void onClickDelete(ActionEvent actionEvent) throws
RemoteException {
        if(!id_input.getText().isEmpty()){


service.deleteSensor(Integer.parseInt(id_input.getText().trim()));
            sensors_array = service.getAllSensorsCurrentData();
            setTableData();
        }
    }

    public void changeMode(String mode){
        this.mode = mode;
        if(mode == "Add"){
            floor_id_input.setText("");
            room_id_input.setText("");
            id_input.setText("");
            mode_btn.setText("Change to Edit");

            insert.setVisible(true);
            edit.setVisible(false);
            delete.setVisible(false);

            id_input.setPromptText("Id Will be Auto Genarated");
            id_input.setDisable(true);

        }else if( mode == "Edit"){
            mode_btn.setText("Change to Register");

            insert.setVisible(false);
            edit.setVisible(true);
            delete.setVisible(true);

            id_input.setPromptText("Enter ID to Edit/Delete");
            id_input.setDisable(false);

        }
    }

    public void onChangeId(){
        id_input.textProperty().addListener(new ChangeListener<String>() {
            @Override
            public void changed(ObservableValue<? extends String>
observable,
                                String oldValue, String newValue) {

                if(newValue.length() > 0 ){
                    if( isNumeric(newValue) ){
                        for (Sensor sensor : sensors_array) {
                            if(sensor.getId() ==
Integer.parseInt(newValue)){
                                activeItem = sensor;
                                setActiveItem(sensor);
                            }
                        }
                    }
                }else{
                    floor_id_input.setText("");
                    room_id_input.setText("");

                }
            }
        });
    }

    private void setActiveItem(Sensor sensor) {
        if(mode == "Add"){
            floor_id_input.setText("");
            room_id_input.setText("");

        }else if( mode == "Edit"){
            floor_id_input.setText(sensor.getFloor_id());
           room_id_input.setText(sensor.getRoom_id());
        }
    }

    public static boolean isNumeric(String str) {
        return str.matches("-?\\d+(\\.\\d+)?");  //match a number with
optional '-' and decimal.
    }


}
```

Controller - SensorsController.java

```
package controllers;

import java.io.IOException;
import java.net.MalformedURLException;
import java.rmi.Naming;
import java.rmi.NotBoundException;
import java.rmi.RMISecurityManager;
import java.rmi.RemoteException;
import java.util.*;

import com.model.Sensor;
```

```java
import com.model.SensorLog;
import com.service.ISensorService;
import javafx.application.Platform;
import javafx.beans.value.ChangeListener;
import javafx.beans.value.ObservableValue;
import javafx.collections.FXCollections;
import javafx.collections.ObservableList;
import javafx.fxml.FXML;
import javafx.scene.chart.AreaChart;
import javafx.scene.chart.CategoryAxis;
import javafx.scene.chart.NumberAxis;
import javafx.scene.chart.XYChart;
import javafx.scene.control.*;
import javafx.scene.control.cell.PropertyValueFactory;

public class SensorsController {

    @FXML
    private Label currentco2;

    @FXML
    private Label sensor_id_label;

    @FXML
    private Label currentsmoke;

    @FXML
    private ChoiceBox<String> sensor_id_selector;

    @FXML
    private Label flood_id;

    @FXML
    private Label room_id;

    @FXML
    private CategoryAxis smoke_x;

    @FXML
    private NumberAxis smoke_y;
    @FXML
    private CategoryAxis co2_x;

    @FXML
    private NumberAxis co2_y;

    @FXML
    private TableView<SensorLog> sensor_log;

    @FXML
    private TableColumn<SensorLog, Integer> co2_level;

    @FXML
    private TableColumn<SensorLog, Integer> smoke_level;

    @FXML
    private TableColumn<SensorLog, String> updated_at;

    @FXML
    private AreaChart<String,Number> smoke_chart;

    @FXML
    private AreaChart<String,Number> co2_chart;

    private int current_id;
    private ArrayList<Sensor> listofsensors = new ArrayList<Sensor>();
    private Sensor current_sensor = new Sensor();
    private LinkedList<SensorLog> current_sensor_log = new
LinkedList<SensorLog>();
    ISensorService service = null;
    @FXML
    void initialize() {

    }

    public void init(Timer timer){

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

```
        try {

            if(System.getSecurityManager() == null ){
                System.setSecurityManager( new RMISecurityManager() );
            }

            service = (ISensorService)
Naming.lookup("//localhost:4500/SensorService");
            listofsensors = service.getAllSensorsCurrentData();
            current_id = listofsensors.get(0).getId();
            initselector();

            final ISensorService finalService = service;

            timer.schedule(new TimerTask() {
                @Override
                public void run() {
                    try {
                        current_sensor =
finalService.getSensorCurrentData(current_id);
                        current_sensor_log =
finalService.getSensorLog(current_id);

                        Platform.runLater(new Runnable() {
                            @Override
                            public void run() {
                                setTableData();
                                setLivechart();
                                setCurrentInfo();
                            }
                        });

                    } catch (IOException e) {
                        System.out.println(e);
                    }
                }
            }, 0, 5000);

        } catch (NotBoundException e) {
            e.printStackTrace();
        } catch (MalformedURLException e) {
            e.printStackTrace();
        } catch (RemoteException e) {
            e.printStackTrace();
        }

    }

    public void setTableData(){
        sensor_log.getItems().removeAll();

        updated_at.setCellValueFactory( new
PropertyValueFactory<SensorLog, String>("datetime"));
        co2_level.setCellValueFactory( new PropertyValueFactory<SensorLog,
Integer>("co2_level"));
        smoke_level.setCellValueFactory( new
PropertyValueFactory<SensorLog, Integer>("smoke_level"));


sensor_log.setItems(FXCollections.observableArrayList(current_sensor_log))
;

        updated_at.setStyle("-fx-size: 35.0px;-fx-min-height: 45.0px;");
        co2_level.setStyle("-fx-size: 35.0px;-fx-min-height: 45.0px;");
        smoke_level.setStyle("-fx-size: 35.0px;-fx-min-height: 45.0px;");
    }

    public void setLivechart(){
        XYChart.Series smoke = new XYChart.Series();
        XYChart.Series co2 = new XYChart.Series();
        co2.setName("co2");
        smoke.setName("smoke");

        for (int i = 0; i < current_sensor_log.size(); i++) {
            co2.getData().add(new XYChart.Data(
current_sensor_log.get(i).getDatetime() ,
current_sensor_log.get(i).getCo2_level() ));
```

```
            smoke.getData().add(new XYChart.Data(
current_sensor_log.get(i).getDatetime(),
current_sensor_log.get(i).getSmoke_level() ));
        }
        smoke_chart.getData().clear();
        co2_chart.getData().clear();

        co2_chart.layout();
        smoke_chart.layout();

        smoke_chart.getData().add(smoke);
        co2_chart.getData().add(co2);

        smoke_chart.setLegendVisible(false);
        co2_chart.setLegendVisible(false);
    }

    public void setCurrentInfo(){
        currentco2.setText( current_sensor.getCo2_level() + ".00");
        currentsmoke.setText( current_sensor.getSmoke_level() + ".00");
        flood_id.setText(( current_sensor.getFloor_id() ));
        room_id.setText(( current_sensor.getRoom_id() ));
    }

    public void initselector(){

        ArrayList<String> list = new ArrayList<String>();
        for (Sensor listofsensor : listofsensors) {
            list.add( listofsensor.getId()+"");
        }
        ObservableList<String> options =
FXCollections.observableArrayList(list);

        sensor_id_selector.setValue(list.get(0)); // this statement shows
default value
        sensor_id_label.setText(list.get(0));

        sensor_id_selector.setItems(options); // this statement adds all
values in choiceBox
```

```
sensor_id_selector.getSelectionModel().selectedIndexProperty().addListener
(new ChangeListener<Number>() {
            @Override
            public void changed(ObservableValue<? extends Number>
observableValue, Number number, Number number2) {

System.out.println(sensor_id_selector.getItems().get((Integer) number2));
                current_id =
Integer.parseInt(sensor_id_selector.getItems().get((Integer) number2));
                sensor_id_label.setText("" + current_id );
                try {
                    current_sensor =
service.getSensorCurrentData(current_id);
                    current_sensor_log = service.getSensorLog(current_id);
                    setTableData();
                    setLivechart();
                    setCurrentInfo();

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



            }
        });
    }



}
```

Main.main class - main.java

```
package main;

import javafx.application.Application;
import javafx.event.EventHandler;
import javafx.fxml.FXMLLoader;
import javafx.scene.Parent;
import javafx.scene.Scene;
```

```java
import javafx.scene.effect.DropShadow;
import javafx.scene.input.MouseEvent;
import javafx.scene.paint.Color;
import javafx.stage.Stage;
import javafx.stage.StageStyle;

import java.awt.*;

public class Main extends Application {

    private double xOffset = 0;
    private double yOffset = 0;

    @Override
    public void start(final Stage primaryStage) throws Exception{

        Parent root =
FXMLLoader.load(getClass().getResource("/pages/login.fxml"));
        primaryStage.initStyle(StageStyle.UNDECORATED);

        root.setOnMousePressed(new EventHandler<MouseEvent>() {
            @Override
            public void handle(MouseEvent event) {
                xOffset = event.getSceneX();
                yOffset = event.getSceneY();
            }
        });

        root.setOnMouseDragged(new EventHandler<MouseEvent>() {
            @Override
            public void handle(MouseEvent event) {
                primaryStage.setX(event.getScreenX() - xOffset);
                primaryStage.setY(event.getScreenY() - yOffset);
            }
        });

        Scene dashboard = new Scene(root, 800, 515);

dashboard.getStylesheets().add("org/kordamp/bootstrapfx/bootstrapfx.css");
        primaryStage.setScene(dashboard);
        primaryStage.show();
    }

    public static void main(String[] args) {
        launch(args);
    }
}
```

SensorDataBuilder

Com - demo.java

```
package com;

import java.awt.Color;
import java.awt.GraphicsConfiguration;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.io.IOException;
import java.io.UnsupportedEncodingException;
import java.lang.management.PlatformLoggingMXBean;
import java.util.ArrayList;
import java.util.Timer;
import java.util.TimerTask;

import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.SwingConstants;

import org.apache.http.HttpEntity;
import org.apache.http.HttpResponse;
import org.apache.http.client.ClientProtocolException;
import org.apache.http.client.HttpClient;
import org.apache.http.client.methods.HttpPost;
import org.apache.http.entity.StringEntity;
import org.apache.http.impl.client.HttpClientBuilder;
import org.apache.http.util.EntityUtils;
import com.google.gson.Gson;

import Util.RandomGen;
import model.Sensor;

public class Demo {

    public static final String URL  =
"http://localhost:4000/sensors/updateall";
    public static boolean danger = false;
    static GraphicsConfiguration gc;
    public static JLabel label;
    public static JButton danger_btn;
    static ArrayList<Sensor> sensors = new ArrayList<Sensor>();

    public static void main(String[] args) {

        RandomGen random = new RandomGen(1, 5);
        Gson gson = new Gson();

        JFrame frame= new JFrame(gc);
        frame.setTitle("Dummy Sensor Data Generator");
        frame.setSize(400, 300);
        frame.setVisible(true);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setResizable(false);
        frame.setBackground(Color.white);
        danger_btn = new JButton("Simulate Threat" );
        danger_btn.setBounds(20,30, 150,20);

        danger_btn.addActionListener(new ActionListener(){
            public void actionPerformed(ActionEvent e){
                      danger = true;
                    }
                 });
        label = new JLabel("loading...", SwingConstants.CENTER  );
        frame.add(danger_btn);
        frame.add(label);
        Timer timer = new Timer();
        timer.schedule(new TimerTask() {
            @Override
            public void run() {
                try {
                    String data = gson.toJson(genarateData(random));
                    System.out.println("Request : " + data);
                    sendDataToAPI( URL  , data);
                    updateLabel();

                } catch (Exception e) {
                    System.out.println(e);
                }
            }
        }, 0, 5000);



    }

    public static void sendDataToAPI(String url , String body) throws
ClientProtocolException, IOException {

        HttpClient   httpClient    = HttpClientBuilder.create().build();
        HttpPost     post          = new HttpPost(url);
        StringEntity postingString = new StringEntity(body);
        post.setEntity(postingString);
        post.setHeader("Content-type", "application/json");
        HttpResponse  response = httpClient.execute(post);

        HttpEntity entity = response.getEntity();

        String content = EntityUtils.toString(entity);
        System.out.println("Request : " + content +"\n");
    }

    public static ArrayList<Sensor> genarateData(RandomGen random){
        sensors.clear();
        sensors.add( new Sensor(3 , random.nextInt() , random.nextInt()
));
```

```
        if(danger == true) {
            sensors.add( new Sensor(4 , 9 , 5 ));
            danger = false;
        }else {
            sensors.add( new Sensor(4 , random.nextInt() ,
random.nextInt() ));
        }
        sensors.add( new Sensor(5 , random.nextInt() , random.nextInt()
));
        return sensors;
    }

    public static void updateLabel() {
        String text = "<HTML><h3>Current Sensors data will display</h3>";
        for (Sensor s : sensors) {
            text += "<h2>";
            text += "ID : " + s.getId() + "   SMOKE LEVEL : " +
s.getSmoke_level() + "   CO2 LEVEL : " + s.getCo2_level();
            text += "</h2>";
        }
        text += "</HTML>";
        label.setText(text);
    }

}
```

Model - sensor.java

```
package model;

public class Sensor {

    private int id;
    private int smoke_level;
    private int co2_level;

    public Sensor() {
        this.id = 0;
        this.smoke_level = 0;
```

```java
        this.co2_level = 0;
    }

    public Sensor(int id, int smoke_level, int co2_level) {
        this.id = id;
        this.smoke_level = smoke_level;
        this.co2_level = co2_level;
    }

    public Sensor(int id) {
        this.id = id;
        this.smoke_level = 0;
        this.co2_level = 0;
    }

    public int getId() {
        return id;
    }

    public void setId(int id) {
        this.id = id;
    }

    public int getSmoke_level() {
        return smoke_level;
    }

    public void setSmoke_level(int smoke_level) {
        this.smoke_level = smoke_level;
    }

    public int getCo2_level() {
        return co2_level;
    }

    public void setCo2_level(int co2_level) {
        this.co2_level = co2_level;
    }
```

```
}
```

Util - RandomGen.java

```
package Util;

import java.util.PrimitiveIterator;
import java.util.Random;

public final class RandomGen {
    private PrimitiveIterator.OfInt randomIterator;


    public RandomGen(int min, int max) {
        randomIterator = new Random().ints(min, max + 1).iterator();
    }


    public int nextInt() {
        return randomIterator.nextInt();
    }
}
```