Sri Lanka Institute of Information Technology

A Distributed Fire Alarm System

Distributed Systems
SE3020 – 2020

Assignment 2- REST API

Table of Content

1. Introduction

This report includes the information about the architecture and services of a distributed fire alarm system built surrounding a reusable REST API. This system consists of these components.

- REST API
- RMI server and an RMI desktop client
- Web client
- Sensor emulator

The rest of the report discusses the high-level architecture, the workflows used in this system and the technologies used in this system. Finally, the Appendix contains all the source code of the system.

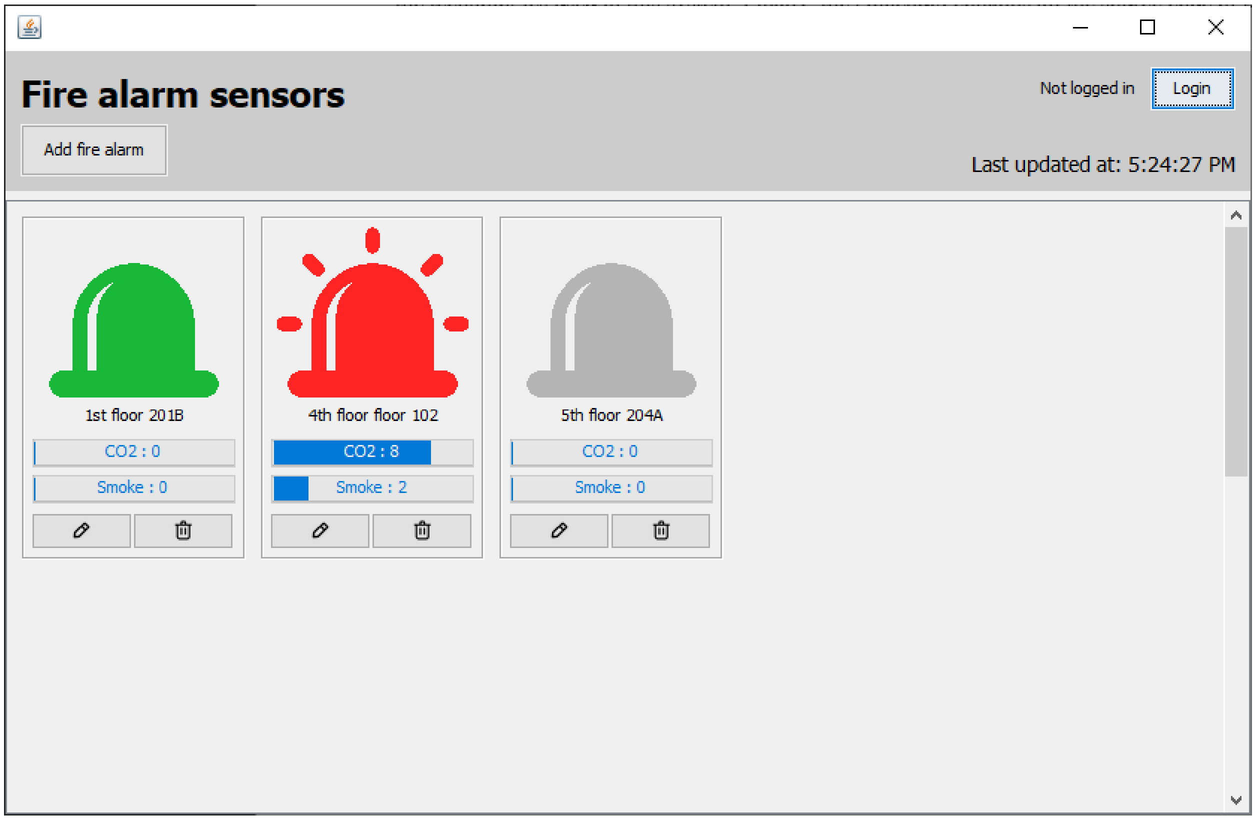
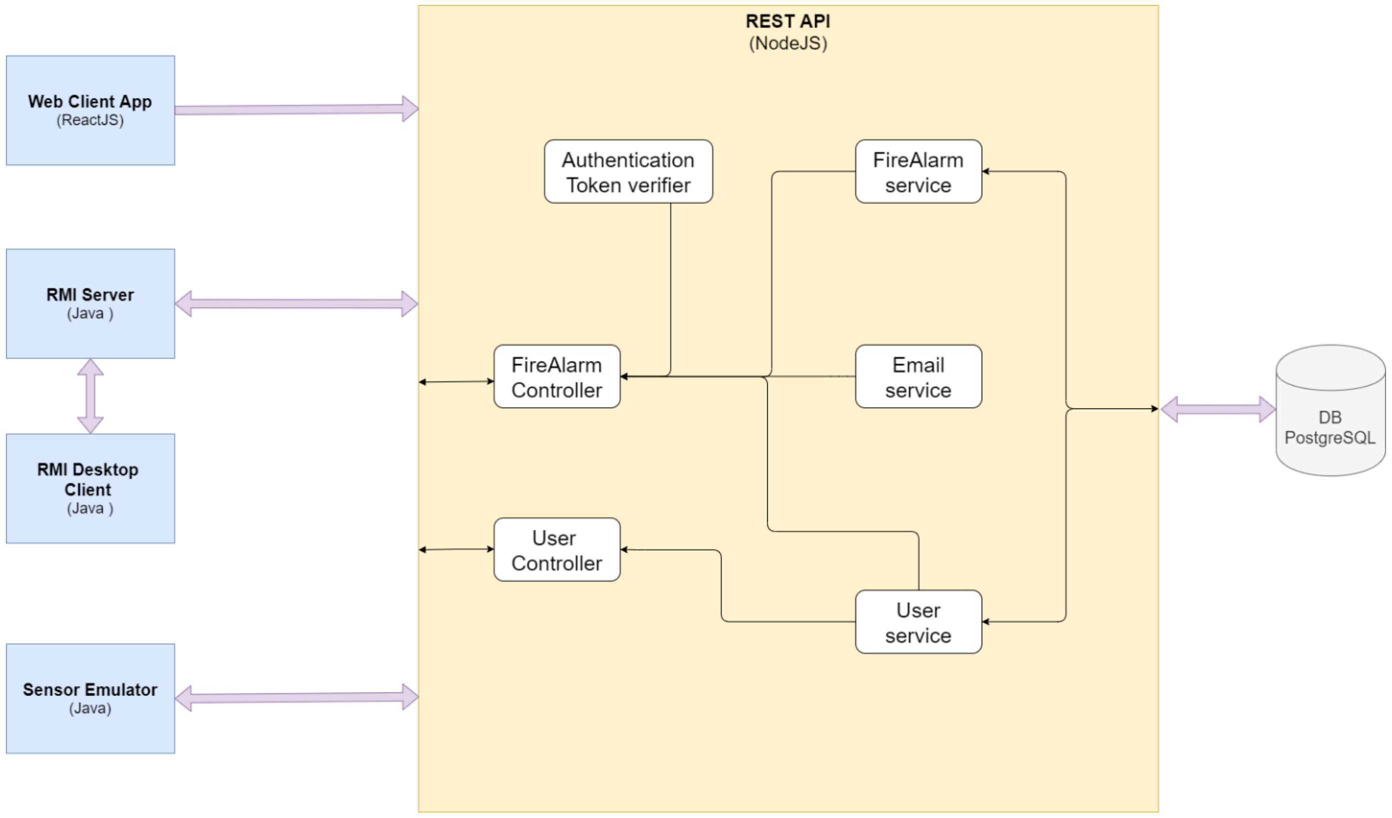


Figure 1RMI desktop client displaying the fire alarms

2. High-level Architecture



This diagram shows the high-level architecture of the complete system, the services included in the REST API and how they interconnect with each other.

3. REST API

This REST API is built using NodeJS and ExpressJS is used as a framework. For all the endpoints, this REST API accepts JSON data and all the responses will be in JSON format.

3.1 Service interfaces

3.1.1 User service

Login This endpoint can be used to authenticate an administrator	
Method: POST	*Path:* /api/v1/users/login

Request payload:

```
{
  "email": string,
  "password": string
}
```

Success response:

If login is successful a JWT (JSON Web Token) will be generated and included in the response JSON object of the below shape. 'isAuth' field represents whether the login credentials are valid.

```
HTTP Status: 200 OK
{
  "isAuth": boolean,
  "token": string
}
```

Error response:

If fields are missing in the request response an error response will be returned.

```
HTTP Status: 400 Bad Request
{
  "message": string
}
```

Sign up
This endpoint can be used to sign up an administrator

Method: POST | *Path:* /api/v1/users/signup

Request payload:

```
{
  “email”: string,
  “password”: string
}
```

Success response:
If signup is successful ‘status’ will be true and otherwise false.

```
HTTP Status: 200 OK
{
  “status”: boolean,
}
```

Error response:
If fields are missing in the request response an error response will be returned.

```
HTTP Status: 400 Bad Request
{
  “message”: string
}
```

Has an admin?
This endpoint can be used to check whether there is a registered administrator account.

Method: GET | *Path:* /api/v1/users/has-admin

Success response:
If there is a registered administrator, the ‘hasAdmin’ will be true and otherwise false.

```
HTTP Status: 200 OK
{
  “hasAdmin”: boolean,
}
```

3.1.2 Fire alarm service

Get all fire alarms
This endpoint can be used to retrieve all the fire alarms.

Method: GET	*Path:* /api/v1/fire-alarms

Success response: An array of fire alarms.

```
HTTP Status: 200 OK
[
  {
    “id”: number,
    “isActive”: boolean,
    “floor”: string,
    “room”: string,
    “smoke_level”: number,
    “co2_level”: number
  },
  ...
]
```

Get a fire alarm
This endpoint can be used to retrieve a fire alarm with the given id.

Method: GET	*Path:* /api/v1/fire-alarms/{fire-alarm-id}

Success response: A fire alarm.

```
HTTP Status: 200 OK
{
  “id”: number,
  “isActive”: boolean,
  “floor”: string,
  “room”: string,
  “smoke_level”: number,
  “co2_level”: number
}
```

Create a fire alarm
This endpoint can be used to create a fire alarm.

Method: POST	*Path:* /api/v1/fire-alarms

Request payload:

```
{
  “floor”: string,
  “room”: string
}
```

This is a protected route. Hence an authenticated JWT must be in the header ‘Authentication’ as a ‘bearer’ token.

Success response: The created fire alarm.

```
HTTP Status: 200 OK
{
  “id”: number,
  “isActive”: boolean,
  “floor”: string,
  “room”: string,
  “smoke_level”: number,
  “co2_level”: number
}
```

Error response:

- If fields are missing in the request.

  ```
  HTTP Status: 400 Bad Request
  {
    “message”: string
  }
  ```

- If the JWT token is missing in the request header

  ```
  HTTP Status: 401 Unauthorized
  {
    “message”: string
  }
  ```

Update a fire alarm
This endpoint can be used to update a fire alarm.

Method: PUT

Path: /api/v1/fire-alarms/{fire-alarm-id}

Request payload:

```
{
  “floor”: string,
  “room”: string,
  “isActive”: boolean,
  “smoke_level”: number,
  “co2_level”: number
}
```

This is a protected route. Hence an authenticated JWT must be in the header ‘Authentication’ as a ‘bearer’ token.

Success response: The updated fire alarm.

```
HTTP Status: 200 OK
{
  “id”: number,
  “isActive”: boolean,
  “floor”: string,
  “room”: string,
  “smoke_level”: number,
  “co2_level”: number
}
```

Error response:

- If fields are missing in the request or the given fire alarm id is not valid.

```
HTTP Status: 400 Bad Request
{
  “message”: string
}
```

- If the JWT token is missing in the request header

```
HTTP Status: 401 Unauthorized
{
  “message”: string
}
```

Update the status of a fire alarm
This endpoint can be used to update the status of a fire alarm of the given id.

Method: PATCH

Path: /api/v1/fire-alarms/{fire-alarm-id}

Request payload:

```
{
  “isActive”: boolean,
  “smoke_level”: number,
  “co2_level”: number
}
```

Success response: The updated fire alarm.

```
HTTP Status: 200 OK
{
  “id”: number,
  “isActive”: boolean,
  “floor”: string,
  “room”: string,
  “smoke_level”: number,
  “co2_level”: number
}
```

Error response:

- If fields are missing in the request or the given fire alarm id is not valid.

```
HTTP Status: 400 Bad Request
{
  “message”: string
}
```

Delete a fire alarm
This endpoint can be used to delete a fire alarm with the given id.

Method: DELETE

Path: /api/v1/fire-alarms/{fire-alarm-id}

Request payload:

This is a protected route. Hence an authenticated JWT must be in the header ‘Authentication’ as a ‘bearer’ token.

Success response: If the deletion is success ‘deleted’ will be true and otherwise false.

```
HTTP Status: 200 OK
{
  “deleted”: boolean
}
```

Error response:

- If the given fire alarm id is invalid.

```
HTTP Status: 400 Bad Request
{
  “message”: string
}
```

- If the JWT token is missing in the request header

```
HTTP Status: 401 Unauthorized
{
  “message”: string
}
```

Send notifications about a fire alarm
This endpoint can be used to notify the administrators about an alarm with higher warning levels. The REST API will decide whether to send emails, SMS or both.

Method: POST	*Path:* /api/v1/fire-alarms/{fire-alarm-id}/notify

Request payload:

```
{
  “message”: string,
}
```

Success response: The updated fire alarm.

```
HTTP Status: 200 OK
{
  “success”: boolean,
}
```

Error response:

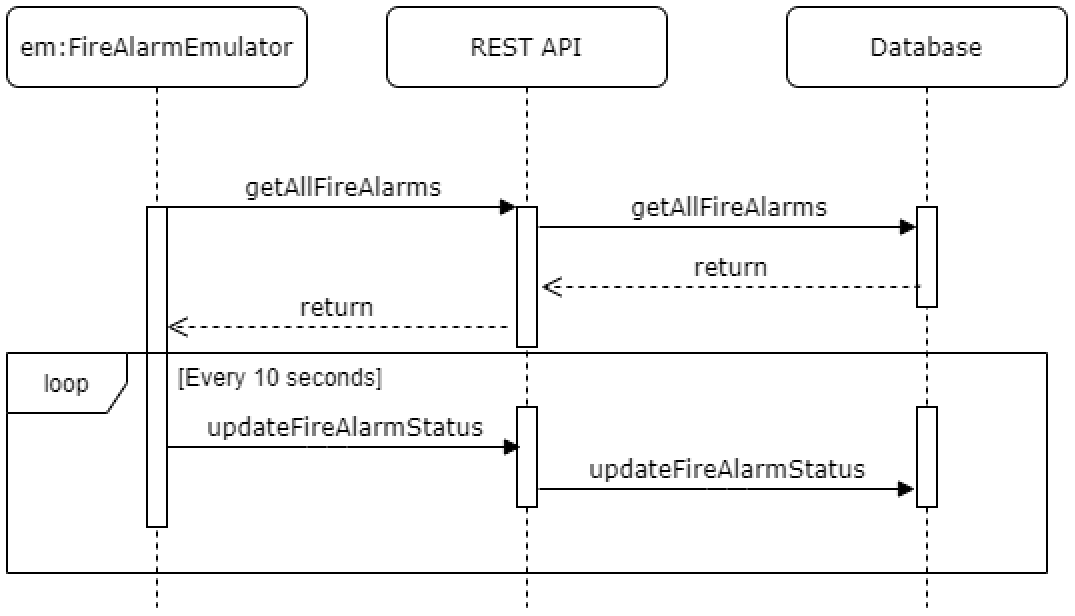
- If fields are missing in the request or the given fire alarm id is not valid.

```
HTTP Status: 400 Bad Request
{
  “message”: string
}
```

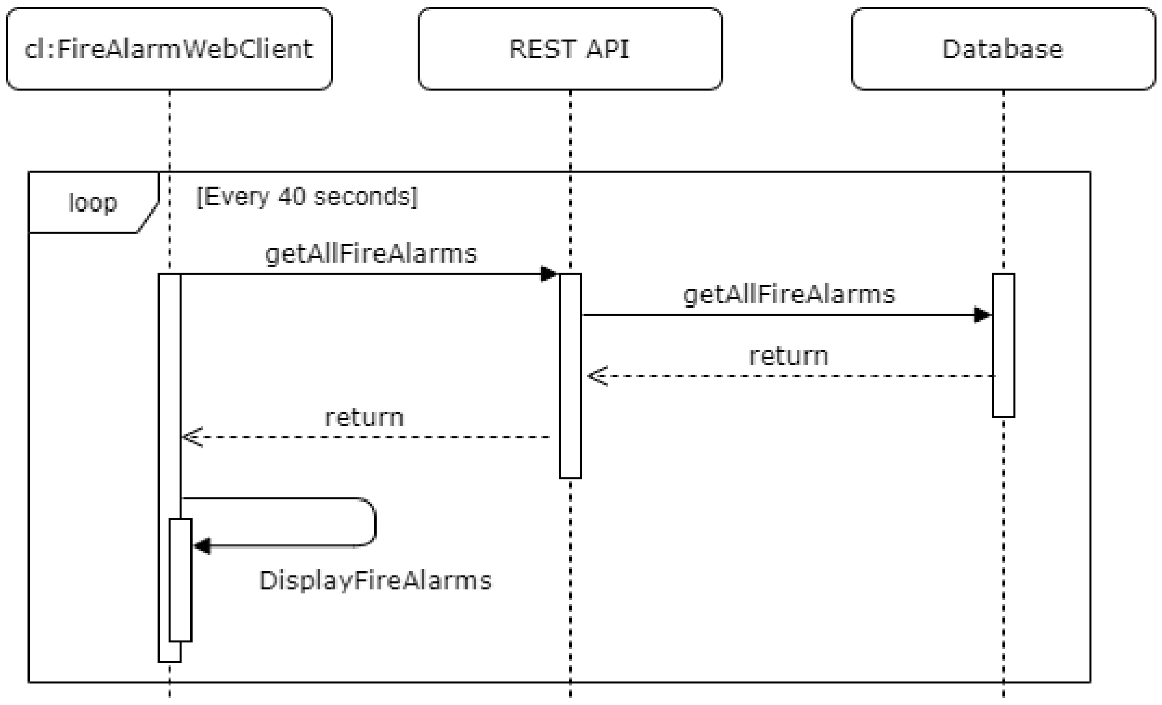
4. Workflows of the system

This section includes a list of sequence diagrams describing the workflows used in the system.

- **Fire alarm emulator workflow**
  Fire alarm emulator will fetch all the fire alarms from the REST API and allows the user to select a fire alarm to emulate. Once a fire alarm is selected, its status will be sent to the REST API in every 10 seconds.

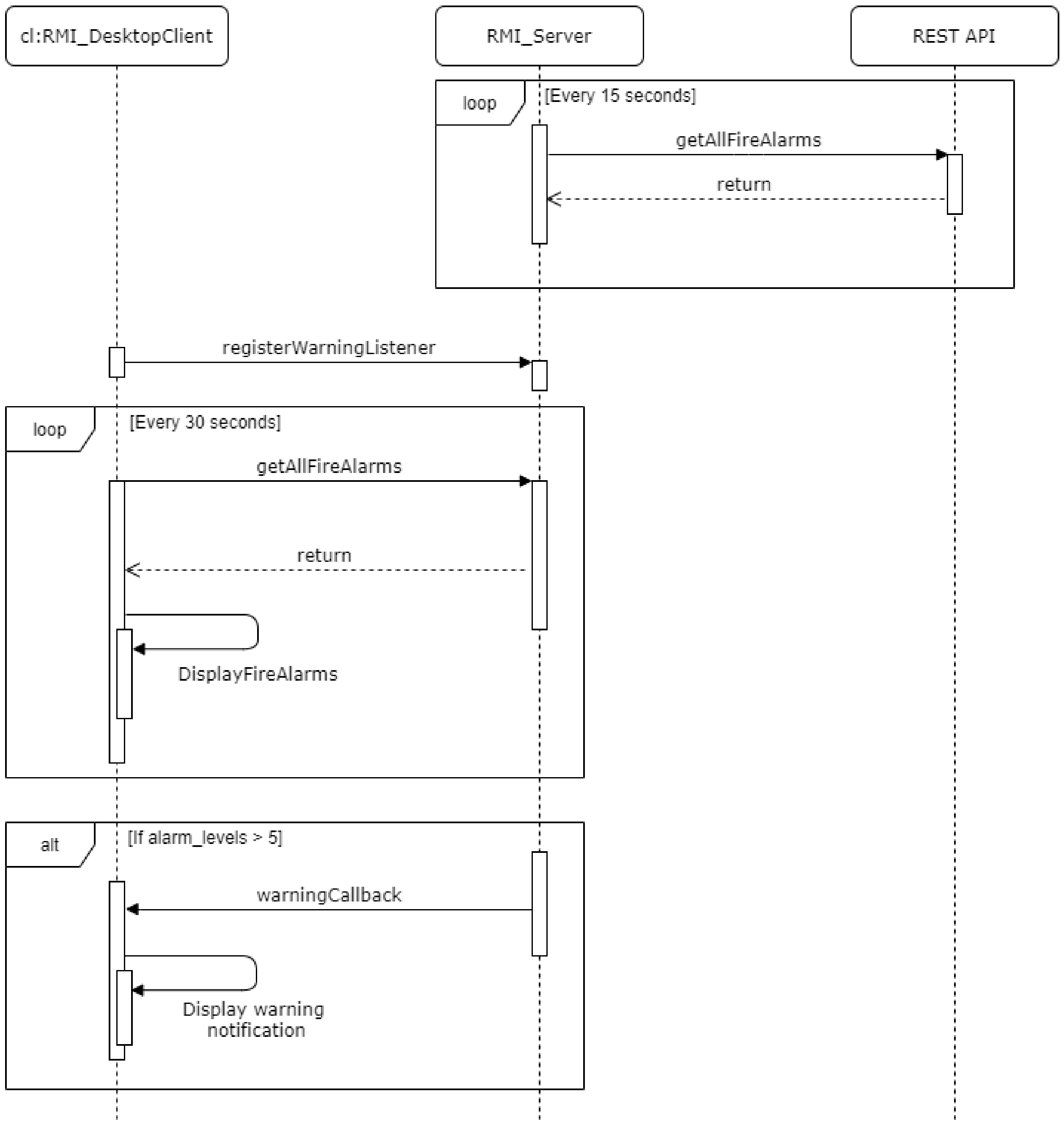


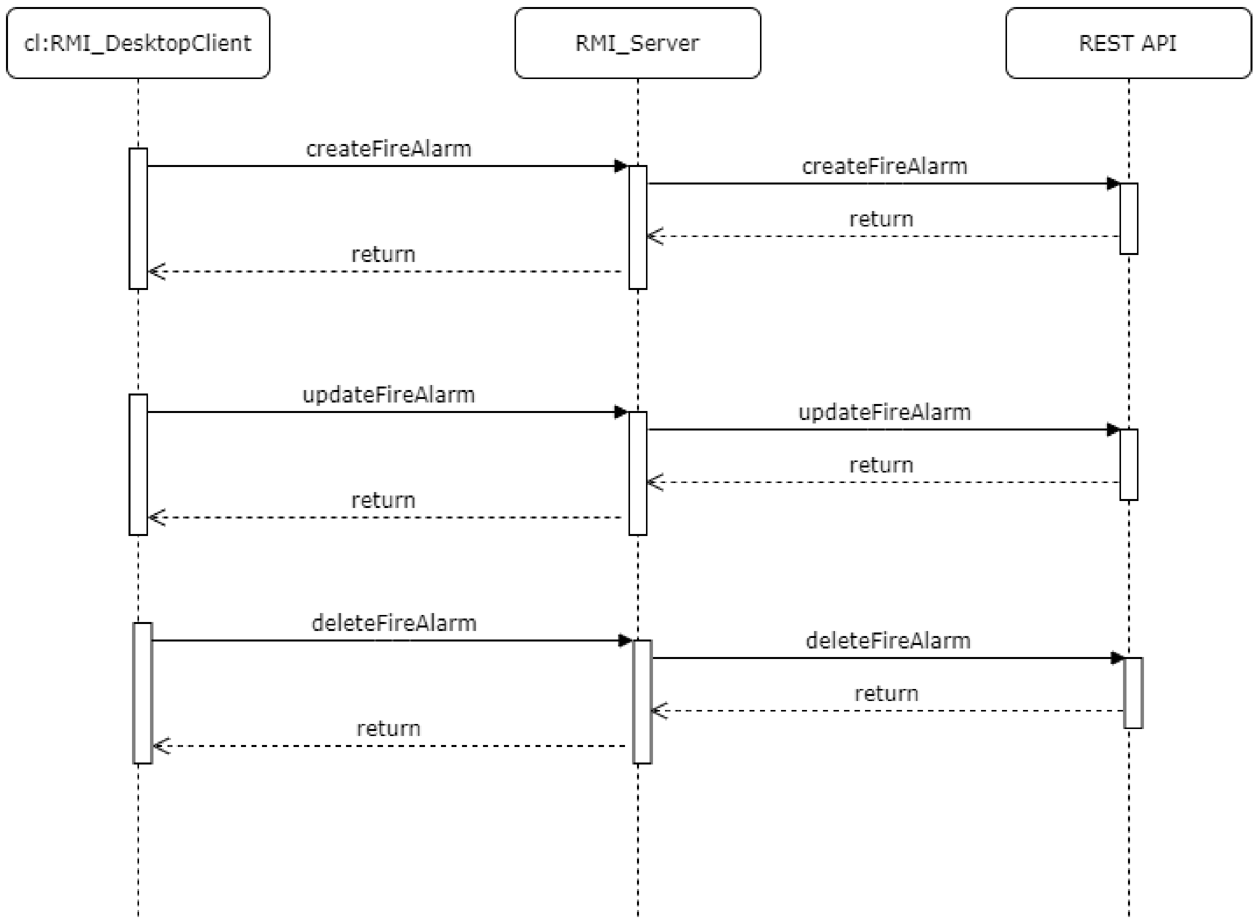
- **Fire alarm web client workflow**
  The web client will fetch all the fire alarms from the REST API and display the status of these alarms. If an alarm has a co2 or smoke level greater than 5, those alarms will blink in red color.



- **RMI server and the RMI client workflow**
  The RMI server will fetch all the fire alarms from the REST API in every 15 seconds and store them in an in-memory array. After each fetch, RMI server will check if there are any alarms with smoke or co2 levels greater than 5. If there are any such alarms, it will notify the opened RMI clients by calling an asynchronous callback function which is registered when an RMI client is started. Also, the RMI server will send a request for the REST API to send notifications to the administrators and other users. The mode of the notification is defined in the REST API whether it is an email or SMS notification.

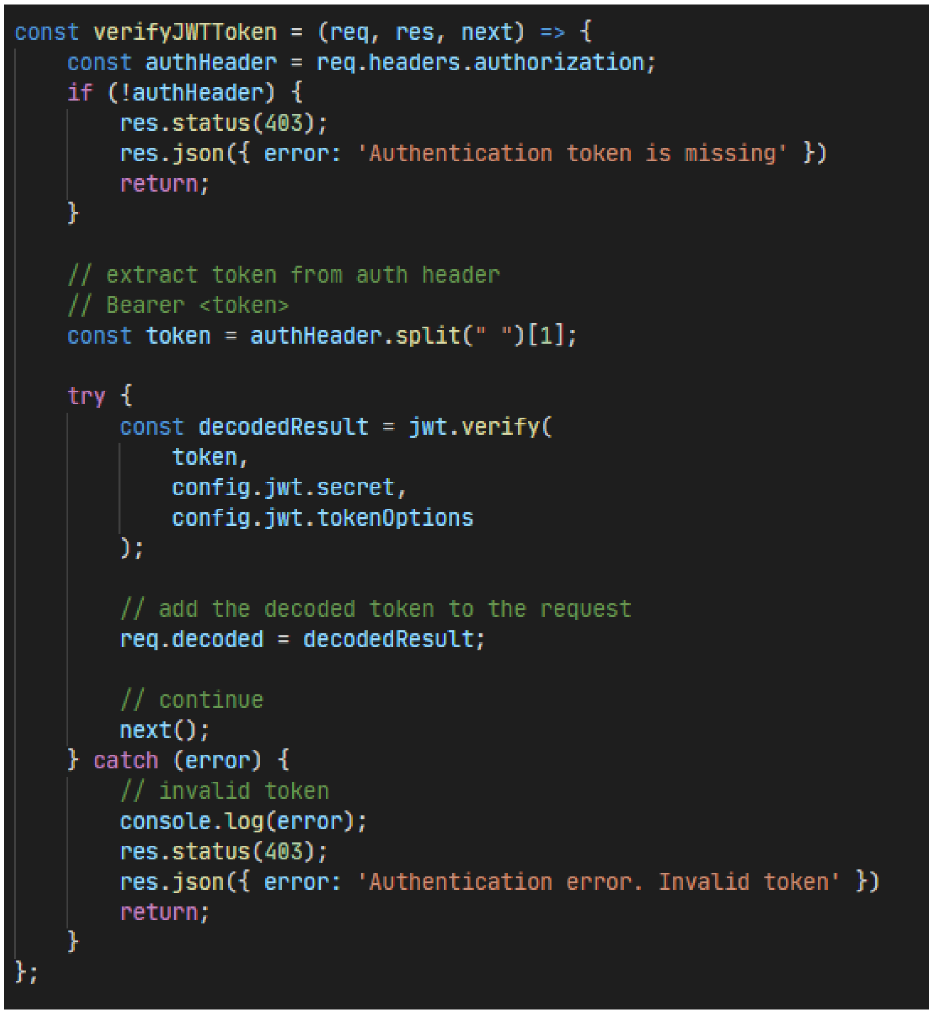
  The user can also login to as an administrator. When the system is running for the first time if there no administrators registered, a user can signup for an administrator through the RMI client. After that user can login and then the functionalities of adding, updating and deleting fire alarms will be activated.


cl:RMI_DesktopClient
RMI_Server
REST API
loop
[Every 15 seconds]
getAllFireAlarms
return
registerWarningListener
loop
[Every 30 seconds]
getAllFireAlarms
return
DisplayFireAlarms
alt
[If alarm_levels > 5]
warningCallback
Display warning notification


cl:RMI_DesktopClient
RMI_Server
REST API
createFireAlarm
createFireAlarm
return
return
updateFireAlarm
updateFireAlarm
return
return
deleteFireAlarm
deleteFireAlarm
return
return

5. Technologies used

- The REST API is built using NodeJS with the ExpressJS framework.
- The database used in the system is a PostgreSQL database.
- The web client application is built using ReactJS.
- The interfaces and the classes used in the RMI communication is encapsulated into a separate project and its generated JAR file is referenced in the RMI server project and the RMI client project to reduce the code duplication.
- The fire alarm emulator app is built using Java Swing.
- JWTs are used to implement the authentication features for the protected endpoints of the REST API.



```
const verifyJWTToken = (req, res, next) => {
    const authHeader = req.headers.authorization;
    if (!authHeader) {
        res.status(403);
        res.json({ error: 'Authentication token is missing' })
        return;
    }

    // extract token from auth header
    // Bearer <token>
    const token = authHeader.split(" ")[1];

    try {
        const decodedResult = jwt.verify(
            token,
            config.jwt.secret,
            config.jwt.tokenOptions
        );

        // add the decoded token to the request
        req.decoded = decodedResult;

        // continue
        next();
    } catch (error) {
        // invalid token
        console.log(error);
        res.status(403);
        res.json({ error: 'Authentication error. Invalid token' })
        return;
    }
};
```

Figure 2: A middleware used to verify the JWT

6. Appendix

- Source code of the REST API

```
//******** server.js
const app = require('./app');
const port = process.env.PORT || 5000;

app.listen(port, (err) => {
   if (err) {
      console.error(err);
   }

   console.log(`Listening on port ${port}`);

})


//******** app.js
require('dotenv').config();

const express = require('express');
const methodOverride = require('method-override');
const logger = require('morgan');
const cors = require('cors');
const fireAlarmRouter = require('./routes/fire-alarm');
const usersRouter = require('./routes/users');

const db = require('./db');

// init database
db.init();

const app = express();

// apply middleware
app.use(cors({ credentials: true }));
app.use(methodOverride('X-HTTP-Method-Override'));
app.use(logger('dev'));
app.use(express.json());
app.use(express.urlencoded({ extended: false }));


app.use('/user', usersRouter);
app.use('/fire-alarm', fireAlarmRouter);
module.exports = app;

//******** db.js

const Sequelize = require('sequelize');
const config = require('./config');

const sequelize = new Sequelize(config.db.url);
exports.init = () => {

    sequelize.authenticate()
        .then(() => {
            console.log('Database connected');
            require('./models/firealarm.model').init(sequelize);
            require('./models/user.model').init(sequelize);

            sequelize.sync();
        })
        .catch(err => {
            console.log('Failed to connect to database', err);
        });

}

exports.db = sequelize;


//******** config.js

module.exports = {
    jwt: {
        secret: process.env.JWT_SECRET || "fire-alarm-api",
        tokenOptions: {
            expiresIn: "1h",
            issuer: process.env.JWT_ISSUER || "fire-alarm-api",
        },
    },
    db: {
        host: process.env.DB_HOST || "localhost",
        database: process.env.DB_NAME || "firealarm",
        username: process.env.DB_USERNAME || "admin",
        password: process.env.DB_PASSWORD || "password",
        url: process.env.DATABASE_URL || 'postgres://postgres:9896@localhost:5432/firealarm'
    }
}
```

```
//******** fire-alarm.js
const express = require('express');
const router = express.Router();
const fireAlarmService = require('../services/fire-alarm.service');
const emailService = require('../services/email.service');
const userService = require('../services/user.service');
const { verifyJWTToken } = require('./middleware');


router.get('/', async (req, res, next) => {
  try {
    // get all fire alarms
    const fireAlarms = await fireAlarmService.getAllFireAlarms();
    res.json(fireAlarms);
  } catch (error) {
    console.error(error);
    res.json({ error: 'Failed to get all fire alarms' });
  }
});

router.get('/:id', async (req, res, next) => {
  // extract the id
  const id = req.params.id;
  try {
    // get fire alarm by id
    const fireAlarm = await fireAlarmService.getFireAlarm(id);
    return res.json(fireAlarm);
  } catch (error) {
    console.error(error);
    res.json({ error: 'Failed to get all fire alarms' });
  }
});

router.post('/', verifyJWTToken, async (req, res, next) => {
  // extract the floor and the room
  const { floor, room } = req.body;
  try {

    // create the alarm
    const fireAlarm = await fireAlarmService.createFireAlaram(floor, room);

    return res.json(fireAlarm);

  } catch (error) {
    console.error(error);
    res.json({ error: 'Failed to create' });
    }
});

router.post('/:id/notify', async (req, res, next) => {
    const { message } = req.body;
    const id = req.params.id;
    try {
        // find the fire alarm and the user emails
        const fireAlarm = await fireAlarmService.getFireAlarm(id);
        const userEmails = await userService.getUserEmails();

        // email all the users
        userEmails.forEach(email => {
            emailService.sendEmail(
                email,
                `WARNING: Status of the fire alarm in ${fireAlarm.room} room on
${fireAlarm.floor} floor`,
                message
            )
        })

        res.json({ success: true });

    } catch (error) {
        console.error(error);
        res.json({ error: 'Failed to notify' });
    }
});



router.put('/:id', verifyJWTToken, async (req, res, next) => {
    const { floor, room, is_active, smoke_level, co2_level } = req.body;
    const id = req.params.id;
    try {
        // update the fire alarm
        const updatedFireAlarm = await fireAlarmService.updateFireAlarm(id, floor, room,
is_active, smoke_level, co2_level);
        res.json(updatedFireAlarm);
    } catch (error) {
        console.error(error);
        res.json({ error: 'Failed to update fire alarm' });
    }
});

router.patch('/:id', async (req, res, next) => {
    // extract the fields
    const { is_active, smoke_level, co2_level } = req.body;
    const id = req.params.id;

    try {
        // update the fire alarm
        const updatedFireAlarm = await fireAlarmService.updateFireAlarmStatus(id, is_active,
smoke_level, co2_level);
        res.json(updatedFireAlarm);
    } catch (error) {
        console.error(error);
        res.json({ error: 'Failed to update smoke level' });
    }
});


router.delete('/:id', verifyJWTToken, async (req, res, next) => {
    // extract the id
    const id = req.params.id;
    try {
        await fireAlarmService.deleteFireAlarm(id);
        res.json({ deleted: true });
    } catch (error) {
        console.error(error);
        res.json({ error: 'Failed to update co2 level' });
    }
});


module.exports = router;


//******** users.js
const express = require('express');
const router = express.Router();

const userService = require('../services/user.service');


router.get('/has-admin', async (req, res, next) => {
  try {
    const hasAdmin = await userService.hasAdmin();
    res.json({ hasAdmin });
  } catch (error) {
    console.error(error);
    res.json({ error: 'Failed to get has admin' });
  }
});

router.post('/signup', async (req, res, next) => {
  // extract email and password
  const { email, password } = req.body;

  try {
    // sign up user
    const user = await userService.signup(email, password);
    if (user) {
      return res.json({ status: true });

    }
    return res.json({ status: false });
  } catch (error) {
    console.error(error);
    res.json({ error: 'Failed to signup user' });
  }
});

router.post('/login', async (req, res, next) => {
  // extract email and password
  const { email, password } = req.body;

  try {
    // check if credentials are valid
    const loginCorrect = await userService.login(email, password);
    if (loginCorrect) {
      const token = userService.generateToken(email);
      return res.json({ isAuth: true, token });

    }
    return res.json({ isAuth: false });
  } catch (error) {
    console.error(error);
    res.json({ error: 'Failed to get has admin' });
  }
});



module.exports = router;
```

```
//******** middleware.js

    const jwt = require("jsonwebtoken");
    const config = require("../config");


    const verifyJWTToken = (req, res, next) => {
      const authHeader = req.headers.authorization;
      if (!authHeader) {
        res.status(403);
        res.json({ error: 'Authentication token is missing' })
        return;
      }

      // extract token from auth header
      // Bearer <token>
      const token = authHeader.split(" ")[1];

      try {
        const decodedResult = jwt.verify(
          token,
          config.jwt.secret,
          config.jwt.tokenOptions
        );

        // add the decoded token to the request
        req.decoded = decodedResult;

        // continue
        next();
      } catch (error) {
        // invalid token
        console.log(error);
        res.status(403);
        res.json({ error: 'Authentication error. Invalid token' })
        return;
      }
    };

    module.exports = {
      verifyJWTToken,
    };
```

```
//******** firealarm.model.js
const Sequelize = require('sequelize');


class FireAlarmSensor extends Sequelize.Model { }


exports.init = (sequelize) => {

   FireAlarmSensor.init({
      id: {
         type: Sequelize.INTEGER,
         primaryKey: true,
         autoIncrement: true
      },
      isActive: {
         type: Sequelize.BOOLEAN,
         defaultValue: false
      },
      floor: {
         type: Sequelize.STRING,
      },
      room: {
         type: Sequelize.STRING
      },
      smoke_level: {
         type: Sequelize.INTEGER
      },
      co2_level: {
         type: Sequelize.INTEGER
      },

   }, {
      sequelize,
      modelName: 'fire-alarm-sensor',
      timestamps: false
   }
   )

}
exports.FireAlarmSensor = FireAlarmSensor;

//******** user.model.js
      const Sequelize = require('sequelize');
```

```
class User extends Sequelize.Model { }


exports.init = (sequelize) => {

    User.init({
        id: {
            type: Sequelize.INTEGER,
            primaryKey: true,
            autoIncrement: true
        },
        email: {
            type: Sequelize.STRING,
        },
        password: {
            type: Sequelize.STRING
        }

    }, {
        sequelize,
        modelName: 'user',
    }
    )

}
exports.User = User;
```

- Source code of the web client

```
//******** App.js
import React from 'react';
import './App.css';
import Alarms from './components/alarms.component';

function App() {
  return (
    <div className="container">
      <Alarms />
    </div>
  );
}

export default App;

//******** App.css
.App {
  text-align: center;
}

.App-logo {
  height: 40vmin;
  pointer-events: none;
}

@media (prefers-reduced-motion: no-preference) {
  .App-logo {
    animation: App-logo-spin infinite 20s linear;
  }
}

.App-header {
  background-color: #282c34;
  min-height: 100vh;
  display: flex;
  flex-direction: column;
  align-items: center;
  justify-content: center;
  font-size: calc(10px + 2vmin);
  color: white;
}

.App-link {
  color: #61dafb;
}

@keyframes App-logo-spin {
  from {
    transform: rotate(0deg);
  }
  to {
    transform: rotate(360deg);
  }
}
```

```css
body {
  margin: 0;
  font-family: -apple-system, BlinkMacSystemFont, 'Segoe UI', 'Roboto', 'Oxygen',
    'Ubuntu', 'Cantarell', 'Fira Sans', 'Droid Sans', 'Helvetica Neue',
    sans-serif;
  -webkit-font-smoothing: antialiased;
  -moz-osx-font-smoothing: grayscale;
  background: antiquewhite;
}

code {
  font-family: source-code-pro, Menlo, Monaco, Consolas, 'Courier New',
    monospace;
}

.container{
margin-top: 2%;

}

.inside {
  background-color: white;
}

.smoke {
  animation: change1 1s infinite;
} @keyframes change1 {
  0%{background-color: white;}
  50%{background-color: red;}
}

.co2 {
  animation: change2 1s infinite;
} @keyframes change2 {
  0%{background-color: white;}
  50%{background-color: red;}
}

.smokeandco2 {
  animation: change3 1s infinite;
} @keyframes change3 {
  0%{background-color: white;}
  50%{background-color: red;}
}
```

```
ul.demo {
  list-style-type: none;
  margin: 0;
  padding: 0;
}

.h3 {
  text-align: center;
}
.single-content {
  position: relative;
  transition: .3s;
  box-shadow: 10px 20px 20px rgba(0,0,0,0.8);
  border: 11px solid ;
}

.single-content .text-content,
.single-content:after {
  position: absolute;
  left: 20px;
  right: 20px;

}

.single-content:after {
  content: '';
  display: block;
  background: black;
  top: 20px;
  bottom: 20px;
  opacity: 0;
  transform: rotate3d(-1,1,0,100deg);
  transition: .4s;

}

.single-content:hover:after {
  opacity: .9;
  transform: rotate3d(0,0,0,0deg);

}

.single-content img {
  width:100%;
  height: auto;
}

.text-content {
  top: 45%;
  opacity: 0;
  z-index: 1;
  transform: translate(10%, -30%);
  transition: .3s;
  text-align: center;
  color: #fff;

}

.text-content h4 {
  font-family: big john ;

}

.single-content:hover .text-content {
  opacity: 1;
  transform: translate(0, -50%);
  transition-delay: .3s;
}

.container .card {
  position: relative;
}

.container .card .face {
  width: 300px;
  height:  200px;
  transition: 0.5s;

}

.container .card .face .face1 {
  position: relative;
  background: #333;
  display: flex;
  justify-content: center;
  align-items: center;
  z-index:1;
  transform: translateY(100px);


}
```

```
.container .card:hover .face .face1 {
  background: #ff0057;
  transform: translateY(0);
}


.container .card .face .face1 .content {
  opacity: 0.2;
  transition: 0.5s;
}

.container .card:hover .face .face1 .content {
  opacity: 1;

}

.container .card:hover .face .face1 .content img{
  max-width: 100px;

}

.container .card:hover .face .face1 .content h3{
  margin: 10px 0 0;
  padding: 0;
  color: #fff;
  text-align: center;
  font-size: 1.5em;
}



.container .card .face .face2 {
  position: relative;
  background: #fff;
  display: flex;
  justify-content: center;
  align-items: center;
  padding: 20px;
  box-sizing: border-box;
  box-shadow: 0 20px 50px rgba(0,0,0,0.8);
  transform: translateY(-100px);

}

.container .card:hover .face .face2 {
  transform: translateY(0);
```

```
}

.container .card .face .face2 .content p {
  margin: 0;
  padding: 0;

}

//******** header.component.js
import React, { Component } from 'react'

export default class Header extends Component {
    render() {
        return (
            <div>
                <h1 style={{ textAlign: 'center', textDecoration: 'bold' }}> Fire Alarm Viewer</h1>
                <hr></hr>
                <p style={{ fontSize: 30 }}>
                    This application will display all the fire alarms that available.<br></br><br />
                    <hr />
                    <ul class="list-group"> <br />
                        <li class="list-group-item"> If fire alarm blinks in <span style={{ color: 'red',
fontStyle: 'italic' }}>red</span> color, that means Smoke or/and co2 levels are <span style={{
textDecoration: 'underline' }}>high. </span></li><br />
                    </ul>

                You can see additional information about the fire alrms by hover it.

                </p>
            </div>
        )
    }
}


//******** alarms.component.js
import React, { Component } from 'react'
import axios from 'axios';

import '../App.css';
import Header from './header.component';

export default class Alarms extends Component {

    constructor(props) {
        super(props);
        this.state = { alarms: [] };
    }

    //fetch get request every 40 seconds
    componentDidMount() {
        this.fetchAlarms();
        setInterval(() => {
            this.fetchAlarms();
        }, 40000);
    }

    fetchAlarms() {
        axios.get('http://localhost:5000/fire-alarm').then(response => {
            this.setState({ alarms: response.data })
        }).catch(function (error) {
            console.log(error);
        });
    }

    //check whether smoke and co2 levels are increased
    ringingAlarm(smoke, co2) {
        if (smoke > 5 && co2 > 5) {
            return "smokeandco2";
        } else if (smoke > 5) {
            return "smoke";
        } else if (co2 > 5) {
            return "co2";
        } else {
            return "inside";
        }
    }

    seeLevels(level) {
        if (level > 5) {
            return "red";
        }
    }

    progressBarChanger(level) {
        if (level > 5) {
            return "progress-bar bg-danger";
        } else {
            return "progress-bar bg-success";
        }
    }
```

```
    render() {
        return (
            <div>
                <Header />
                <hr />
                <div class="row">
                    {this.state.alarms.map(alarm => (
                        <div class="col-md-3 col-sm-5">
                            <div class="single-content" style={{ borderColor: alarm.isActive ? '#4CAF50'
: 'lightgray' }}>
                                <div class={this.ringingAlarm(alarm.smoke_level, alarm.co2_level)}>
                                    <img src={require('../assets/alarm.png')} alt="alarm" /><hr />
                                    <h3 style={{ textAlign: 'center' }}>Room {alarm.room} on floor
{alarm.floor} </h3>
                                    <h4>smoke Level</h4>
                                    <div class="progress">
                                        <div class={this.progressBarChanger(alarm.smoke_level)}
role="progressbar" style={{ width: alarm.smoke_level + '0%' }} aria-
valuenow={alarm.smoke_level} aria-valuemin="0" aria-
valuemax="10">{alarm.smoke_level}</div>
                                    </div>
                                    <h4>Co2 Level</h4>
                                    <div class="progress">
                                        <div class={this.progressBarChanger(alarm.co2_level)}
role="progressbar" style={{ width: alarm.co2_level + '0%' }} aria-valuenow={alarm.co2_level}
aria-valuemin="0" aria-valuemax="10">{alarm.co2_level}</div>
                                    </div>

                                </div>
                                <div class="text-content"><br />
                                    <h4>Alarm ID : {alarm.id}</h4>
                                    <h5>Alarm Floor : {alarm.floor}</h5>
                                    <h5>Alarm Room : {alarm.room}</h5>
                                    <h5>Is active : {alarm.isActive}</h5>
                                    <h5 style={{ backgroundColor: this.seeLevels(alarm.smoke_level)
}}>Alarm Smoke Level : {alarm.smoke_level}</h5>
                                    <h5 style={{ backgroundColor: this.seeLevels(alarm.co2_level)
}}>Alarm Co2 level : {alarm.co2_level}</h5>

                                </div>
                            </div>
                            <hr></hr>
                        </div>
                    ))}
                </div>
            </div>
        )
    }
}
```

- Source code of the RMI API

```
//******** FireAlarmSensorService.java

package firealarm.rmi.api;

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

/**
 * This interface defines the API to interact
 * with the fire alarm service of the RMI server
 * @author Nuwan Karunarathna
 *
 */
public interface FireAlarmSensorService extends Remote {
	List<FireAlarmSensor> getAllFireAlarms() throws RemoteException;

	FireAlarmSensor getFireAlarm(int id) throws RemoteException;

	FireAlarmSensor createFireAlarm(String token, String floor, String room) throws
RemoteException;

	FireAlarmSensor updateFireAlarm(String token, FireAlarmSensor sensor) throws
RemoteException;

	boolean deleteFireAlarm(String token, int id) throws RemoteException;

    void registerWarningListener(FireAlarmSensorWarningListener listner) throws
RemoteException;

    void removeWarningListner(FireAlarmSensorWarningListener listner) throws
RemoteException;

}
```

```
//******** UserService.java
package firealarm.rmi.api;

import java.rmi.Remote;
import java.rmi.RemoteException;

/**
 * This interface defines the API to interact
 * with the user services of the RMI server
 * @author Nuwan Karunarathna
 *
 */
public interface UserService extends Remote {
	boolean hasAdmin() throws RemoteException;
	boolean createAdmin(String email, String password) throws RemoteException;


	/**Login the user
	 * @param email Email of the user
	 * @param password Password of the user
	 * @return A token if the email and password are correct or null otherwise
	 */
	String  login(String email, String password) throws RemoteException;
}

//******** FireAlarmSensorWarningListener.java
package firealarm.rmi.api;

import java.rmi.Remote;
import java.rmi.RemoteException;

/**
 *
 * @author nuwan
 */
public interface FireAlarmSensorWarningListener extends Remote{
    void notifyWarning(FireAlarmSensor sensor) throws RemoteException;
}

//******** FireAlarmSensor.java
package firealarm.rmi.api;

import java.io.Serializable;
import java.time.LocalDate;

/**
```

```
 * This class represents a Fire alarm sensor
 *
 * @author Nuwan Karunarathna
 *
 */

public class FireAlarmSensor implements Serializable {

    private int id;
    private String floor;
    private String room;
    private boolean isActive;
    private int smokeLevel;
    private int co2Level;
    private LocalDate createdDate;
    private LocalDate updatedDate;



    public FireAlarmSensor(int id, String floor, String room, int smokeLevel, int co2Level,
LocalDate createdDate,
                            LocalDate updatedDate) {
        super();
        this.id = id;
        this.floor = floor;
        this.room = room;
        this.smokeLevel = smokeLevel;
        this.co2Level = co2Level;
        this.createdDate = createdDate;
        this.updatedDate = updatedDate;
    }


    public FireAlarmSensor(int id, String floor, String room, int smokeLevel, int co2Level,
boolean isActive) {
        super();
        this.id = id;
        this.floor = floor;
        this.room = room;
        this.isActive = isActive;
        this.smokeLevel = smokeLevel;
        this.co2Level = co2Level;
    }


    public int getId() {
```

```
    return id;
}
public void setId(int id) {
    this.id = id;
}
public String getFloor() {
    return floor;
}
public void setFloor(String floor) {
    this.floor = floor;
}
public String getRoom() {
    return room;
}
public void setRoom(String room) {
    this.room = room;
}
public int getSmokeLevel() {
    return smokeLevel;
}
public void setSmokeLevel(int smokeLevel) {
    this.smokeLevel = smokeLevel;
}
public int getCo2Level() {
    return co2Level;
}
public void setCo2Level(int co2Level) {
    this.co2Level = co2Level;
}
public LocalDate getCreatedDate() {
    return createdDate;
}
public void setCreatedDate(LocalDate createdDate) {
    this.createdDate = createdDate;
}
public LocalDate getUpdatedDate() {
    return updatedDate;
}
public void setUpdatedDate(LocalDate updatedDate) {
    this.updatedDate = updatedDate;
}

 public boolean getIsActive() {
    return isActive;
}
```

```
   public void setIsActive(boolean isActive) {
      this.isActive = isActive;
   }

}


//******** APIServiceNames.java
package firealarm.rmi.api;

public enum APIServiceNames {
        FIRE_ALARM_SERVICE,
        USER_SERVICE
}
```

- Source code of the RMI server

```
//******** RMIServer.js
package com.firealarm.server;

import com.firealarm.helpers.APIHelper;
import com.firealarm.services.FireAlarmServiceImpl;
import com.firealarm.services.UserServiceImpl;
import firealarm.rmi.api.APIServiceNames;

import java.rmi.RemoteException;
import java.rmi.registry.LocateRegistry;
import java.rmi.registry.Registry;

public class RMIServer {
   public static void main(String[] args) {
      try {
         // create the registry
         Registry registry = LocateRegistry.createRegistry(Registry.REGISTRY_PORT);

         // create services
         FireAlarmServiceImpl fireAlarmService = new FireAlarmServiceImpl();
         UserServiceImpl userService = new UserServiceImpl();

         // bind the services

registry.rebind(APIServiceNames.FIRE_ALARM_SERVICE.toString(),fireAlarmService);
         registry.rebind(APIServiceNames.USER_SERVICE.toString(), userService);


         System.out.println("Services started");
```

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

//******** FireAlarmServiceImpl.java
package com.firealarm.services;

import com.firealarm.helpers.APIHelper;
import com.firealarm.helpers.JsonHelper;
import firealarm.rmi.api.FireAlarmSensor;
import firealarm.rmi.api.FireAlarmSensorService;
import firealarm.rmi.api.FireAlarmSensorWarningListener;


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

import java.util.concurrent.Executors;
import java.util.concurrent.ScheduledExecutorService;
import java.util.concurrent.TimeUnit;
import java.util.stream.Collectors;

import javax.json.*;


public class FireAlarmServiceImpl extends UnicastRemoteObject implements
FireAlarmSensorService, Runnable {
    private final static String FIRE_ALARM_URL = Constants.FIRE_ALARM_API_URL +
"/fire-alarm";

    private final List<FireAlarmSensor> sensorsList;
    private final List<FireAlarmSensorWarningListener> warningListeners;

    public FireAlarmServiceImpl() throws RemoteException {
        this.sensorsList = new ArrayList<>();
        this.warningListeners = new ArrayList<>();

        // schedule fire alarm fetching to run at every 15 seconds
        ScheduledExecutorService executorService =
Executors.newSingleThreadScheduledExecutor();
```

```
        executorService.scheduleAtFixedRate(this, 0, 15, TimeUnit.SECONDS);
    }

    @Override
    public void run() {
        this.fetchFireAlarmSensors();
    }

    private void fetchFireAlarmSensors(){
        StringBuffer res = null;
        try {
            res = APIHelper.get(FIRE_ALARM_URL);
        } catch (IOException e) {
            e.printStackTrace();
        }
        if(res == null){
            return;
        }

        // parse the response
        JsonArray jo =  JsonHelper.getJsonArrayFromString(res.toString());

        // get fire alarm sensor objects from json
        List<FireAlarmSensor> alarms = jo.stream().map(alarmJson -> {

            return JsonHelper.getFireAlamSensorFromJson((JsonObject) alarmJson);

        }).collect(Collectors.toList());

        List<FireAlarmSensor> originalSensorList = new ArrayList<>();
        originalSensorList.addAll(sensorsList);

        // update the sensor list
        this.sensorsList.removeAll(this.sensorsList);
        this.sensorsList.addAll(alarms);

        // send warning notification if needed
        this.checkSensorLevels(originalSensorList);
    }

    private void checkSensorLevels(List<FireAlarmSensor> originalSensorList){

        for (FireAlarmSensor updatedSensor :
                sensorsList) {
            FireAlarmSensor originalSensor = getSensorById(updatedSensor.getId(),
originalSensorList);
```

```
        if(originalSensor != null){
            // already have a sensor with this id
            // if the levels haven't change, we have already notified about this
            if(originalSensor.getCo2Level() == updatedSensor.getCo2Level()
            && originalSensor.getSmokeLevel() == updatedSensor.getSmokeLevel()){
                continue;
            }
        }

        // this is either a new sensor or an old sensor with new level(s)
        // notify if necessary
        if(updatedSensor.getCo2Level() > 5 || updatedSensor.getSmokeLevel() > 5){
            // issue a warning immediately to registered clients
            notifyWarningListeners(updatedSensor);

            // tell the REST API to send notifications to users
            notifyUsers(updatedSensor);

        }
    }


}

private void notifyWarningListeners(FireAlarmSensor sensor){
    for (FireAlarmSensorWarningListener listner :
            warningListeners) {
        try {
            listner.notifyWarning(sensor);
        } catch (RemoteException e) {
            e.printStackTrace();
        }

    }
}

private void notifyUsers(FireAlarmSensor sensor){

    StringBuilder msgBuilder = new StringBuilder();

    if(sensor.getSmokeLevel() > 5 && sensor.getCo2Level() > 5){
        msgBuilder.append("Smoke level and the CO2 level");
    }else if(sensor.getSmokeLevel() > 5){
        msgBuilder.append("Smoke level");
    }else if( sensor.getCo2Level() > 5){
        msgBuilder.append("CO2 level");
    }

    msgBuilder.append(
            " of the fire alarm sensor in "
                    + sensor.getFloor()
                    + " floor "
                    + sensor.getRoom()
                    + " room ");

    String msg = msgBuilder.toString();
    JsonObject sendNotificationParams = Json.createObjectBuilder()
            .add("message", msg)
            .build();
    try {
        APIHelper.post(
                FIRE_ALARM_URL + "/" + sensor.getId() + "/notify",
                sendNotificationParams,
                null);
    } catch (IOException e) {
        e.printStackTrace();
    }
}

private FireAlarmSensor getSensorById(int id){
    FireAlarmSensor sensorById = null;
    for (FireAlarmSensor sensor :
            sensorsList) {
        if(sensor.getId() == id){
            sensorById = sensor;
            break;
        }
    }
    return sensorById;
}

private FireAlarmSensor getSensorById(int id, List<FireAlarmSensor> list){
    FireAlarmSensor sensorById = null;
    for (FireAlarmSensor sensor :
            list) {
        if(sensor.getId() == id){
            sensorById = sensor;
            break;
        }
    }
    return sensorById;
}

@Override
public List<FireAlarmSensor> getAllFireAlarms() throws RemoteException {
    return this.sensorsList;
}

@Override
public FireAlarmSensor getFireAlarm(int id) throws RemoteException {
    FireAlarmSensor sensorById = getSensorById(id);
    return sensorById;
}

@Override
public FireAlarmSensor createFireAlarm(String token, String floor, String room) throws
RemoteException {
    JsonObject createFireAlamParms = Json.createObjectBuilder()
            .add("floor", floor)
            .add("room", room)
            .build();

    StringBuffer res = null;

    try {
        res = APIHelper.post(FIRE_ALARM_URL, createFireAlamParms, token);
    } catch (IOException e) {
        e.printStackTrace();
    }

    if(res == null){
        return null;
    }
    JsonObject alarmJson = JsonHelper.getJsonObjectFromString(res.toString());

    FireAlarmSensor sensor = JsonHelper.getFireAlamSensorFromJson(alarmJson);
    this.fetchFireAlarmSensors();
    return sensor;
}


@Override
public FireAlarmSensor updateFireAlarm(String token, FireAlarmSensor sensorTOUpdate)
throws RemoteException {
    JsonObject updateFireAlarmParams = Json.createObjectBuilder()
            .add("floor", sensorTOUpdate.getFloor())
            .add("room", sensorTOUpdate.getRoom())
            .add("is_active", sensorTOUpdate.getIsActive())
            .add("smoke_level", sensorTOUpdate.getSmokeLevel())
            .add("co2_level", sensorTOUpdate.getCo2Level())
            .build();

    StringBuffer res = null;

    try {
        res = APIHelper.put(FIRE_ALARM_URL + "/" + sensorTOUpdate.getId(),
updateFireAlarmParams, token);
    } catch (IOException e) {
        e.printStackTrace();
    }

    if(res == null){
        return null;
    }

    JsonObject alarmJson = JsonHelper.getJsonObjectFromString(res.toString());

    FireAlarmSensor updatedAlarm = JsonHelper.getFireAlamSensorFromJson(alarmJson);
    this.fetchFireAlarmSensors();
    return updatedAlarm;
}

@Override
public boolean deleteFireAlarm(String token, int alarmId) throws RemoteException {
    JsonObject deleteFireAlarmParams = Json.createObjectBuilder()
            .add("id", alarmId)
            .build();

    StringBuffer res = null;

    try {
        res = APIHelper.delete(FIRE_ALARM_URL + '/' + alarmId, deleteFireAlarmParams,
token);
    } catch (IOException e) {
        e.printStackTrace();
    }

    if(res == null){
        return false;
    }

    JsonObject deleteResult = JsonHelper.getJsonObjectFromString(res.toString());
    try {
        boolean isDeleted = deleteResult.getBoolean("deleted");
        if(isDeleted){
            fetchFireAlarmSensors();
        }
        return isDeleted;
    }catch (NullPointerException e){
        return false;
    }

}

public void registerWarningListener(FireAlarmSensorWarningListener listener) throws
RemoteException {
    this.warningListeners.add(listener);
}

public void removeWarningListner(FireAlarmSensorWarningListener listener) throws
RemoteException {
    this.warningListeners.remove(listener);
}
}



//******** UserServiceImpl.java

    package com.firealarm.services;

    import com.firealarm.helpers.APIHelper;
    import com.firealarm.helpers.JsonHelper;
    import firealarm.rmi.api.UserService;

    import javax.json.Json;
    import javax.json.JsonObject;
    import java.io.IOException;
    import java.rmi.RemoteException;
    import java.rmi.server.UnicastRemoteObject;

    public class UserServiceImpl extends UnicastRemoteObject implements UserService {

        private final static String USER_URL = Constants.FIRE_ALARM_API_URL +  "/user";

        public UserServiceImpl() throws RemoteException {

        }
```

```
@Override
public boolean hasAdmin() throws RemoteException {
   StringBuffer res = null;

   try {
      // call REST API
      res = APIHelper.get(USER_URL + "/has-admin");
   } catch (IOException e) {
      e.printStackTrace();
   }
   if(res == null){
      return false;
   }

   // convert response to java object
   JsonObject hasAdminResult = JsonHelper.getJsonObjectFromString(res.toString());

   boolean hasAdmin = hasAdminResult.getBoolean("hasAdmin");

   return hasAdmin;
}

@Override
public boolean createAdmin(String email, String password) throws RemoteException {
   // build parameter object
   JsonObject signupParams = Json.createObjectBuilder()
         .add("email", email)
         .add("password", password)
         .build();


   StringBuffer res = null;
   try {
      res = APIHelper.post(USER_URL + "/signup", signupParams, null);
   } catch (IOException e) {
      e.printStackTrace();
   }

   if(res == null){
      return false;
   }

   // convert response to java object
   JsonObject signupResponse = JsonHelper.getJsonObjectFromString(res.toString());
```

```
            boolean isSuccess = signupResponse.getBoolean("status");
            if(isSuccess){
              return true;
            }
            // signup failed
            return false;
        }

        @Override
        public String login(String email, String password) throws RemoteException {
            // build parameter object
            JsonObject loginParams = Json.createObjectBuilder()
                    .add("email", email)
                    .add("password", password)
                    .build();


            StringBuffer res = null;
            try {
                res = APIHelper.post(USER_URL + "/login", loginParams, null);
            } catch (IOException e) {
                e.printStackTrace();
            }

            if(res == null){
                return null;
            }

            JsonObject loginResponse = JsonHelper.getJsonObjectFromString(res.toString());

            boolean isAuth = loginResponse.getBoolean("isAuth");
            if(isAuth){
                // logged in
                String token = loginResponse.getString("token");
                return  token;
            }
            // logged in failed
            return null;
        }
    }

//******** Constants.java
    package com.firealarm.services;

    public class Constants {
```

```
    public static final String FIRE_ALARM_API_URL = "http://localhost:5000";
}


//******** APIHelper.java

package com.firealarm.helpers;

import javax.json.JsonObject;
import java.io.*;
import java.net.HttpURLConnection;
import java.net.URL;

public class APIHelper {

    /**
     * Perform a get requets
     * @param url url of the request
     * @return string representation of the response
     * @throws IOException
     */
    public static StringBuffer get(String url) throws IOException {
        // build url
        URL getUrl = new URL(url);

        // build connection
        HttpURLConnection con = (HttpURLConnection) getUrl.openConnection();
        // set method
        con.setRequestMethod("GET");

        int responseCode = con.getResponseCode();

        String readLine = null;
        // read the response
        if(responseCode == HttpURLConnection.HTTP_OK){
            BufferedReader in = new BufferedReader(new
InputStreamReader((con.getInputStream())));
            StringBuffer response = new StringBuffer();
            while((readLine = in.readLine()) != null){
                response.append(readLine);
            }
            in.close();
            return response;
        }
        return null;
    }
```

```
   /**
    * Perform a post request
    * @param url url of the request
    * @param params parameter data to send with the request
    * @param token An authentication token, this will be added to the header of
    *            the request
    * @return
    * @throws IOException
    */
   public static StringBuffer post(String url, JsonObject params, String token) throws
IOException {
       // get param string
       String parsedParams = JsonHelper.parseJsonToString(params);
       // parse url
       URL postUrl = new URL(url);

       // build connection
       HttpURLConnection con  = (HttpURLConnection)postUrl.openConnection();

       con.setRequestMethod("POST");
       // set token if there is any
       if(token != null){
           con.setRequestProperty("Authorization","Bearer " + token);
       }

       return fetchResponse(con, parsedParams);

   }


   /**
    * Performs a put request
    * @param url url of the request
    * @param params parameter data to send with the request
    * @param token an authentication token
    * @return a string representation of the response
    * @throws IOException
    */
   public static StringBuffer put(String url, JsonObject params, String token) throws
IOException {
       String parsedParams = JsonHelper.parseJsonToString(params);
       URL postUrl = new URL(url);

       HttpURLConnection con  = (HttpURLConnection)postUrl.openConnection();
```

```
        con.setRequestMethod("PUT");

        if(token != null){
            con.setRequestProperty("Authorization","Bearer " + token);
        }

        return fetchResponse(con, parsedParams);

    }

    /**
     * performs a delete request
     * @param url url of the request
     * @param params parameters to send with the request
     * @param token an authentication token
     * @return a string representation of the response
     * @throws IOException
     */
    public static StringBuffer delete(String url, JsonObject params, String token) throws
IOException {
        String parsedParams = JsonHelper.parseJsonToString(params);
        URL postUrl = new URL(url);

        HttpURLConnection con  = (HttpURLConnection)postUrl.openConnection();

        con.setRequestMethod("DELETE");

        if(token != null){
            con.setRequestProperty("Authorization","Bearer " + token);
        }

        return fetchResponse(con, parsedParams);

    }

    /**
     * builds a StringBuffer with the response stream
     * @param stream an input stream
     * @return a StringBuffer filled with the stream data
     * @throws IOException
     */
    private static StringBuffer getResponseFromStream(InputStream stream) throws
IOException {
        BufferedReader in = new BufferedReader(new InputStreamReader(stream));
        String inputLine;
        StringBuffer response = new StringBuffer();
```

```
            // read the stream
            while ((inputLine = in.readLine()) != null){
               response.append(inputLine);
            }
            in.close();
            return response;
         }

         /**
          * Extract response from the connection
          * @param con
          * @param params
          * @return
          * @throws IOException
          */
         private static StringBuffer fetchResponse(HttpURLConnection con, String params)
      throws IOException {
            con.setRequestProperty("Content-type", "application/json");

            // read from the stream
            con.setDoOutput(true);
            OutputStream os = con.getOutputStream();
            os.write(params.getBytes());
            os.flush();
            os.close();

            int responseCode = con.getResponseCode();

            if(responseCode == HttpURLConnection.HTTP_OK){
               BufferedReader in = new BufferedReader(new
      InputStreamReader(con.getInputStream()));
               return getResponseFromStream(con.getInputStream());
            }
            return null;
         }


      }


//********  JSONHelper.java
package com.firealarm.helpers;

import firealarm.rmi.api.FireAlarmSensor;
import javax.json.Json;
import javax.json.JsonArray;
import javax.json.JsonObject;
import javax.json.JsonReader;
import java.io.StringReader;
import java.io.StringWriter;
import java.io.Writer;

public class JsonHelper {

    /**
     * Convert a string to a JSON object
     * @param str
     * @return
     */
    public static JsonObject getJsonObjectFromString(String str){

        JsonReader reader = Json.createReader(new StringReader(str));

        JsonObject parsedObject =  reader.readObject();
        reader.close();
        return parsedObject;
    }

    /**
     * Conver a string to a JSON array
     * @param str
     * @return
     */
    public static JsonArray getJsonArrayFromString(String str){

        JsonReader reader = Json.createReader(new StringReader(str));

        JsonArray parsedArray =  reader.readArray();
        reader.close();
        return parsedArray;
    }

    /**
     * Convert a JSON object to a string
     * @param obj
     * @return
     */
    public static String parseJsonToString(JsonObject obj){
        String parsedString;
        Writer writer = new StringWriter();
        Json.createWriter(writer).write(obj);
        parsedString = writer.toString();
        return parsedString;
    }

    /**
     * Convert a JSON object to a FireAlarmSensor
     * @param alarmJson
     * @return
     */
    public static FireAlarmSensor getFireAlamSensorFromJson(JsonObject alarmJson){
        int id = ((JsonObject)alarmJson).getInt("id");
        String floor = ((JsonObject)alarmJson).getString("floor");
        String room = ((JsonObject)alarmJson).getString("room");
        int smokeLevel =  ((JsonObject)alarmJson).getInt("smoke_level");
        int co2Level =  ((JsonObject)alarmJson).getInt("co2_level");
        boolean isActive = ((JsonObject)alarmJson).getBoolean("isActive");


        FireAlarmSensor alarm = new FireAlarmSensor(
                id,
                floor,
                room,
                smokeLevel,
                co2Level,
                isActive
                );

        return alarm;
    }
}
```

- Source code of the RMI client (This project was created through Netbeans and the auto generated UI code is not included in here)

```
//********  FireAlarmDesktopiClientStarter.java

       package fire.alarm.desktopi.client;

       import firealarm.rmi.api.*;
       import java.rmi.NotBoundException;
       import java.rmi.RemoteException;
       import java.rmi.registry.LocateRegistry;
       import java.rmi.registry.Registry;
```

```
/**
 *
 * @author nuwan
 */
public class FireAlarmDesktopiClientStarter {

    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {

        try {
            // locate the registry
            Registry registry = LocateRegistry.getRegistry("localhost",
Registry.REGISTRY_PORT);

            // load the services
            FireAlarmSensorService fireAlarmSensorService = (FireAlarmSensorService)
registry.lookup(APIServiceNames.FIRE_ALARM_SERVICE.toString());
            UserService userService = (UserService)
registry.lookup(APIServiceNames.USER_SERVICE.toString());

            System.out.println("Services loaded");


            // start the client
           FireAlarmDesktopClient client = new
FireAlarmDesktopClient(fireAlarmSensorService, userService);
           client.displayWindow();


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

}
```

//******** FireAlarmDesktopiClient.java

```
package fire.alarm.desktopi.client;

import firealarm.rmi.api.FireAlarmSensor;
import firealarm.rmi.api.FireAlarmSensorService;
import firealarm.rmi.api.FireAlarmSensorWarningListener;
import firealarm.rmi.api.UserService;
import java.awt.AWTException;
import java.awt.Image;
import java.awt.SystemTray;
import java.awt.Toolkit;
import java.awt.TrayIcon;
import java.awt.event.WindowAdapter;
import java.awt.event.WindowEvent;
import java.io.Serializable;
import java.rmi.RemoteException;
import java.rmi.server.UnicastRemoteObject;
import java.util.logging.Level;
import java.util.logging.Logger;

/**
 *
 * @author nuwan
 */
public class FireAlarmDesktopClient extends UnicastRemoteObject implements
Serializable, FireAlarmSensorWarningListener {

    private FireAlarmSensorService fireAlarmService;

    transient private SensorWindow mainWindow;

    public FireAlarmDesktopClient(FireAlarmSensorService fireAlarmSensorService,
UserService userService) throws  RemoteException{
        this.fireAlarmService = fireAlarmSensorService;

        // change the default theme
        try {
            for (javax.swing.UIManager.LookAndFeelInfo info :
javax.swing.UIManager.getInstalledLookAndFeels()) {
                if ("Windows".equals(info.getName())) {
                    javax.swing.UIManager.setLookAndFeel(info.getClassName());
                    break;
                }
            }
        } catch (Exception e) {

        }

        // create the main window
        mainWindow = new SensorWindow(fireAlarmSensorService, userService);
        // register warning listner
        registerListeners();

    }


    private void registerListeners(){
        try {
            System.out.println("Registering warning listener");
            fireAlarmService.registerWarningListener(this);
        } catch (RemoteException ex) {
            Logger.getLogger(FireAlarmDesktopClient.class.getName()).log(Level.SEVERE,
null, ex);
        }

        mainWindow.addWindowListener(new WindowAdapter() {
            @Override
            public void windowClosing(WindowEvent e) {
                super.windowClosing(e);
                try {

FireAlarmDesktopClient.this.fireAlarmService.removeWarningListner(FireAlarmDesktopC
lient.this);
                    System.out.println("Removed warning listener");
                } catch (RemoteException ex) {

Logger.getLogger(FireAlarmDesktopClient.class.getName()).log(Level.SEVERE, null, ex);
                }
            }

        });
    }


    @Override
    public void notifyWarning(FireAlarmSensor sensor) throws RemoteException {


        mainWindow.forceFetch();

        showNotification(sensor);
    }

    private void showNotification(FireAlarmSensor sensor){

        SystemTray tray = SystemTray.getSystemTray();
        Image image = 
    Toolkit.getDefaultToolkit().createImage(getClass().getResource("/fire/alarm/desktopi/client
    /assets/alarm_warning.png"));

        TrayIcon trayIcon = new TrayIcon(image, "Tray Demo");

        trayIcon.setImageAutoSize(true);

        trayIcon.setToolTip("Fire alarm WARNING");
        try {
          tray.add(trayIcon);
        } catch (AWTException ex) {
          Logger.getLogger(SensorWindow.class.getName()).log(Level.SEVERE, null, ex);
        }

        String msg = "Fire alarm in " + sensor.getRoom() + " room at " + sensor.getFloor() + " 
    floor has a warning level";
        trayIcon.displayMessage("Fire alarm WARNING",   msg , 
    TrayIcon.MessageType.WARNING);

      }

      public void displayWindow(){

        mainWindow.setVisible(true);
      }


    }


//********  SensorWindow.java
    package fire.alarm.desktopi.client;

    import firealarm.rmi.api.*;
    import java.awt.*;
    import java.awt.event.ActionEvent;
    import java.rmi.RemoteException;
    import java.time.ZonedDateTime;
    import java.time.format.DateTimeFormatter;
    import java.time.format.FormatStyle;
    import java.util.ArrayList;
    import java.util.List;
    import java.util.concurrent.Executors;
    import java.util.concurrent.ScheduledExecutorService;
import java.util.concurrent.TimeUnit;
import java.util.logging.Level;
import java.util.logging.Logger;
import javax.swing.AbstractAction;
import javax.swing.JFrame;
import javax.swing.JOptionPane;

/**
 *
 * @author nuwan
 */
public class SensorWindow extends javax.swing.JFrame implements Runnable {

   private FireAlarmSensorService fireAlarmService;
   private UserService userService;

   private boolean hasAdmin = false;
   private String userAuthToken = null;
   private List<FireAlarmSensor> sensorList;

   /**
    * Creates new form SensorWindow
    */
   public SensorWindow(FireAlarmSensorService fireAlarmService, UserService
userService) {

       // draw the window
       initComponents(); // auto generated method by Netbeans

       this.fireAlarmService = fireAlarmService;
       this.userService = userService;

       // initialize sensor list
       sensorList = new ArrayList<>();

       // initialize button labels
       initLoginButton();

       // schedule fetching of sensors to run every 30 seconds
       ScheduledExecutorService executorService =
Executors.newSingleThreadScheduledExecutor();
       executorService.scheduleAtFixedRate(this, 0, 30, TimeUnit.SECONDS);
   }


private void loginButtonMouseClicked(java.awt.event.MouseEvent evt) {
        // TODO add your handling code here:
    }

    private void loginButtonActionPerformed(java.awt.event.ActionEvent evt) {
        handleLoginClicked();
    }

    private void addFireAlarmBtnActionPerformed(java.awt.event.ActionEvent evt) {
        if (userAuthToken == null) {
            // user not logged in
            // display error dialog
            showErrorDialog("Please login to create a fire alarm");

        } else {
            // display fire alarm dialog
            FireAlarmDialog alarmDialog = new FireAlarmDialog(this,
rootPaneCheckingEnabled, null);
            alarmDialog.setVisible(true);
            if (alarmDialog.getFormSubmitted()) {
                addFireAlarm(alarmDialog.getFloor(), alarmDialog.getRoom());
            }
        }

    }

    private void formWindowOpened(java.awt.event.WindowEvent evt) {
        setExtendedState(JFrame.MAXIMIZED_BOTH);
    }

    private void sensorPanelComponentAdded(java.awt.event.ContainerEvent evt) {
        resizeSensorPanel();
    }

    private void formComponentResized(java.awt.event.ComponentEvent evt) {
        resizeSensorPanel();
    }

    private void resizeSensorPanel(){
         Dimension d = new Dimension();
        d.setSize(jScrollPane1.getSize().getWidth() - 20, sensorPanel.getHeight());
        sensorPanel.setPreferredSize(d);
    }

    // Variables declaration - do not modify
    private javax.swing.JButton addFireAlarmBtn;
    private javax.swing.JPanel buttonPanel;
    private javax.swing.JLabel jLabel1;
    private javax.swing.JScrollPane jScrollPane1;
    private javax.swing.JLabel lastUpdateLabel;
    private javax.swing.JButton loginButton;
    private javax.swing.JPanel sensorPanel;
    private javax.swing.JLabel userInfo;
    // End of variables declaration

    /**
     * This method will fetch the fire alarms from the RMI server and add them
     * to the sensor list
     */
    private void fetchFireAlarms() {
        System.out.println("fetching fire alarms");
        try {
            List<FireAlarmSensor> sensors = fireAlarmService.getAllFireAlarms();

            sensorList.removeAll(sensorList);
            sensorList.addAll(sensors);

            populateSensorPanel();

            // update last updated time
            lastUpdateLabel.setText("Last updated at: " +
(ZonedDateTime.now().format(DateTimeFormatter.ofLocalizedTime(FormatStyle.MEDIU
M))));

        } catch (RemoteException ex) {
            Logger.getLogger(SensorWindow.class.getName()).log(Level.SEVERE, null, ex);
        }

    }

    /**
     * This method will clear the sensor panel and will populate it with the
     * current sensorList
     */
    private void populateSensorPanel() {
        // clear sensor panel
        sensorPanel.removeAll();

        for (FireAlarmSensor sensor : sensorList) {
            SensorItem item = new SensorItem();
            item.setSensor(sensor);

            // set a click listener for the edit button
            item.setOnEditClickListner(new AbstractAction() {
                @Override
                public void actionPerformed(ActionEvent e) {
                    int clickedAlarmId = Integer.valueOf(e.getActionCommand());
                    SensorWindow.this.handleEditClicked(clickedAlarmId);
                }
            });

            // set a click listener for the delete button
            item.setOnDeleteClickListner(new AbstractAction() {
                @Override
                public void actionPerformed(ActionEvent e) {
                    int clickedAlarmId = Integer.valueOf(e.getActionCommand());
                    SensorWindow.this.handleDeleteClicked(clickedAlarmId);
                }
            });

            // add item to the sensor panel
            sensorPanel.add(item);
        }

        // notify sensor panel
        sensorPanel.repaint();
        sensorPanel.revalidate();

        jScrollPane1.repaint();
        jScrollPane1.revalidate();
    }

    /**
     * This method will display the login form if the user is not logged in and
     * will logout the user otherwise
     */
    private void handleLoginClicked() {

        if (userAuthToken == null) {
            // not logged in
            // show login dialog
            LoginDialog d;
            if (hasAdmin) {
                // login mode
                d = new LoginDialog(this, rootPaneCheckingEnabled, false);
            } else {
                // sign up mode
                d = new LoginDialog(this, rootPaneCheckingEnabled, true);
            }
            d.setVisible(true);

            // check if user submitted the form or clicked cancel
            if (d.getLoginClicked()) {
                if (hasAdmin) {
                    login(d.getEmail(), d.getPassword());
                } else {
                    signUp(d.getEmail(), d.getPassword());
                }
            }
        } else {
            // logged in
            // so log out user
            userAuthToken = null;
            loginButton.setText("Login");
            userInfo.setText("Not logged in");
        }
    }

    private void handleEditClicked(int alarmId) {
        if (userAuthToken == null) {
            // not logged in
            showErrorDialog("Please log in to edit an alarm");
            return;
        }

        FireAlarmSensor sensorToUpdate = getFireAlarmById(alarmId);
        FireAlarmDialog d = new FireAlarmDialog(this, rootPaneCheckingEnabled,
sensorToUpdate);
        d.setVisible(true);
        if(!d.getFormSubmitted()){
            return;
        }
        sensorToUpdate.setFloor(d.getFloor());
        sensorToUpdate.setRoom(d.getRoom());

        updateFireAlarm(sensorToUpdate);
    }

    private void handleDeleteClicked(int alarmId) {
        if (userAuthToken == null) {
            // not logged in
            showErrorDialog("Please log in to delete an alarm");
            return;
        }
    FireAlarmSensor sensorToDelete = getFireAlarmById(alarmId);

    int deleteConfirmResult = JOptionPane.showConfirmDialog(
            this,
            "Delete this fire alarm of "
            + sensorToDelete.getFloor()
            + " floor "
            + sensorToDelete.getRoom() + " room?");

    if (deleteConfirmResult == JOptionPane.YES_OPTION) {
        deleteFireAlarm(sensorToDelete);
    }
}

/**
 * This method will call the necessary RMI server methods to perform the
 * login
 *
 * @param email Email entered by the user
 * @param password Password entered by the user
 */
private void login(String email, String password) {
    try {
        userAuthToken = userService.login(email, password);
        if (userAuthToken != null) {
            // login success
            loginButton.setText("Logout");
            userInfo.setText("User: " + email);
        } else {
            // login failed
            // display error dialog
            showErrorDialog("Failed to login. Please try again");

        }
    } catch (RemoteException ex) {
        Logger.getLogger(SensorWindow.class.getName()).log(Level.SEVERE, null, ex);
    }

}

/**
 * This method will call the necessary RMI server methods to perform the
 * sign up
 *
 * @param email Email entered by the user
 * @param password Password entered by the user
```

```
     */
    private void signUp(String email, String password) {
        try {
            boolean userCreated = userService.createAdmin(email, password);
            if (userCreated) {
                // sign up success
                hasAdmin = true;
                loginButton.setText("Login");
                handleLoginClicked();
            } else {
                // sign up failed
                // display error dialog
                showErrorDialog("Failed to sign up. Please try again");

            }
        } catch (RemoteException ex) {
            Logger.getLogger(SensorWindow.class.getName()).log(Level.SEVERE, null, ex);
        }
    }

    /**
     * This method will call the necessary RMI server methods to add a fire
     * alarm
     *
     * @param floor Floor entered by the user
     * @param room Room entered by the user
     */
    private void addFireAlarm(String floor, String room) {
        try {
            FireAlarmSensor createdSensor =
fireAlarmService.createFireAlarm(userAuthToken, floor, room);
            if (createdSensor != null) {
                // fire alarm creation succeded
                // add fire alarm to the list
                sensorList.add(createdSensor);
                // update sensor panel
                populateSensorPanel();

            } else {
                // fire alarm creation failed
                // display error dialog
                showErrorDialog("Failed to create fire alarm. Please try again");

            }
        } catch (RemoteException ex) {
            Logger.getLogger(SensorWindow.class.getName()).log(Level.SEVERE, null, ex);
    }
}

/**
 * This method will call the necessary RMI server methods to update a fire
 * alarm
 *
 * @param sensorToUpdate FireAlarm clicked by the user
 */
private void updateFireAlarm(FireAlarmSensor sensorToUpdate) {

    try {
        FireAlarmSensor updatedSensor = fireAlarmService.updateFireAlarm(
                userAuthToken,
                sensorToUpdate);

        if (updatedSensor == null) {
            // fire alarm update failed
            showErrorDialog("Failed to update fire alarm. Please try again");
        } else {
            // fire alarm update succeeded
            FireAlarmSensor oldSensor = getFireAlarmById(sensorToUpdate.getId());
            int indexOfOldSensor = sensorList.indexOf(oldSensor);
            if (indexOfOldSensor == -1) {
                // sensor list has already updated
                return;
            }
            // replace the old fire alarm sensor with the new one
            sensorList.set(indexOfOldSensor, updatedSensor);

            // update the sensor panel
            populateSensorPanel();
        }
    } catch (RemoteException ex) {
        Logger.getLogger(SensorWindow.class.getName()).log(Level.SEVERE, null, ex);
    }
}

/**
 * This method will call the necessary RMI server methods to delete a fire
 * alarm
 *
 * @param alarmToDelete Fire alarm clicked by the user
 */
private void deleteFireAlarm(FireAlarmSensor alarmToDelete) {
        try {
            boolean deleted = fireAlarmService.deleteFireAlarm(userAuthToken,
alarmToDelete.getId());
            if (deleted) {
                // deletion sucessful
                // update sensorList
                int indexOfDeletedAlarm = sensorList.indexOf(alarmToDelete);
                if (indexOfDeletedAlarm == -1) {
                    // fire alarm already removed from the list
                    return;
                }
                // remove sensor from the sensorList
                sensorList.remove(indexOfDeletedAlarm);

                // update sensor panel
                populateSensorPanel();

            } else {
                // failed to delete
                showErrorDialog("Failed to delete the fire alarm. Please try again later");
            }
        } catch (RemoteException ex) {
            Logger.getLogger(SensorWindow.class.getName()).log(Level.SEVERE, null, ex);
        }
    }

    /**
     * This method will update the label of the login button based on whether
     * there is a registered administrator account
     */
    private void initLoginButton() {
        try {
            hasAdmin = userService.hasAdmin();
            if (!hasAdmin) {
                loginButton.setText("Sign up");
            }
        } catch (RemoteException ex) {
            Logger.getLogger(SensorWindow.class.getName()).log(Level.SEVERE, null, ex);
        }
    }

    /**
     * This method will display a dialog with the given message and a OK button
     *
     * @param msg Message to display in the dialog
     */
    private void showErrorDialog(String msg) {
            JOptionPane.showMessageDialog(this, msg, "Warning",
  JOptionPane.WARNING_MESSAGE);
    }

    private FireAlarmSensor getFireAlarmById(int id) {
        FireAlarmSensor sensor = null;
        for (FireAlarmSensor fireAlarmSensor : sensorList) {
            if (fireAlarmSensor.getId() == id) {
                sensor = fireAlarmSensor;
                break;
            }
        }
        return sensor;
    }

    // implemented method of the Runnable interface
    @Override
    public void run() {
        fetchFireAlarms();
    }

    public void forceFetch() {
        fetchFireAlarms();
    }

  }




//********  SensorItem.java
package fire.alarm.desktopi.client;

import firealarm.rmi.api.FireAlarmSensor;
import javax.swing.AbstractAction;
import javax.swing.ImageIcon;

/**
 *
 * @author nuwan
 */
public class SensorItem extends javax.swing.JPanel {

    /**
```

```
     * Creates new form SensorItem
     */
    public SensorItem() {
        initComponents();
    }

    public void setSensor(FireAlarmSensor sensor){
        sensorLocation.setText(sensor.getFloor() + " floor " + sensor.getRoom());
        co2Level.setValue(sensor.getCo2Level() * 10);
        co2Level.setString("CO2 : " + sensor.getCo2Level());
        smokeLevel.setValue(sensor.getSmokeLevel() * 10);
        smokeLevel.setString("Smoke : " + sensor.getSmokeLevel());

        editBtn.setActionCommand(String.valueOf(sensor.getId()));
        deleteBtn.setActionCommand(String.valueOf(sensor.getId()));

        if(sensor.getIsActive()){
            // set active icon
            sensorIcon.setIcon(new
ImageIcon(getClass().getResource("/fire/alarm/desktopi/client/assets/alarm_active.png")));
        }

        if(sensor.getCo2Level() > 5 || sensor.getSmokeLevel() > 5){
            // set warning icon
            sensorIcon.setIcon(new
ImageIcon(getClass().getResource("/fire/alarm/desktopi/client/assets/alarm_warning.png")));
        }
    }

    public void setOnEditClickListner(AbstractAction action){

        editBtn.addActionListener(action);
    }

    public void setOnDeleteClickListner(AbstractAction action){
        deleteBtn.addActionListener(action);
    }

    // Variables declaration - do not modify
    private javax.swing.JProgressBar co2Level;
    private javax.swing.JButton deleteBtn;
    private javax.swing.JButton editBtn;
    private javax.swing.JPanel jPanel1;
    private javax.swing.JSeparator jSeparator1;
    private javax.swing.JSeparator jSeparator2;
    private javax.swing.JSeparator jSeparator3;
    private javax.swing.JSeparator jSeparator4;
    private javax.swing.JSeparator jSeparator5;
    private javax.swing.JLabel sensorIcon;
    private javax.swing.JLabel sensorLocation;
    private javax.swing.JProgressBar smokeLevel;
    // End of variables declaration
}


//********  LoginDialog.java
        package fire.alarm.desktopi.client;

        import java.awt.event.ActionEvent;
        import java.awt.event.ActionListener;

        /**
         *
         * @author nuwan
         */
        public class LoginDialog extends javax.swing.JDialog {

            private boolean loginClicked = false;
            private boolean signUpMode = false;

            /**
             * Creates new form LoginDialog
             */
            public LoginDialog(java.awt.Frame parent, boolean modal, boolean signUpMode) {
                super(parent, modal);
                initComponents();
                this.signUpMode = signUpMode;
                initLabels();

                setLocationRelativeTo(null);
            }

            private void initLabels(){
                if(signUpMode){
                    dialogLabel.setText("Sign up to fire alarm service");
                    loginBtn.setText("Sign up");
                }
            }

            public boolean getLoginClicked(){
                return this.loginClicked;
            }
            public String getEmail(){
                return this.emailTextbox.getText();
            }

            public String getPassword(){
                return this.passwordTextBox.getText();
            }

            public void clearInputs(){
                this.emailTextbox.setText("");
                this.passwordTextBox.setText("");
            }

        private void loginBtnActionPerformed(java.awt.event.ActionEvent evt) {
                this.loginClicked = true;
                this.setVisible(false);
            }

            private void cancelBtnActionPerformed(java.awt.event.ActionEvent evt) {
                this.setVisible(false);
            }



            // Variables declaration - do not modify
            private javax.swing.JButton cancelBtn;
            private javax.swing.JLabel dialogLabel;
            private javax.swing.JTextField emailTextbox;
            private javax.swing.JLabel jLabel1;
            private javax.swing.JLabel jLabel2;
            private javax.swing.JButton loginBtn;
            private javax.swing.JPasswordField passwordTextBox;
            // End of variables declaration
        }


//********  FireAlarmDialog.java
        package fire.alarm.desktopi.client;

        import firealarm.rmi.api.FireAlarmSensor;

        /**
         *
         * @author nuwan
         */
```

```
public class FireAlarmDialog extends javax.swing.JDialog {

    private boolean editMode = false;
    private FireAlarmSensor sensor = null;
    private boolean formSubmitted = false;

    /**
     * Creates new form FireAlarmDialog
     */
    public FireAlarmDialog(java.awt.Frame parent, boolean modal, FireAlarmSensor sensor )
{
        super(parent, modal);
        initComponents();

        if(sensor != null){
            editMode = true;
            this.sensor = sensor;
            fillSensorInformation();
            updateLabels();
        }

        setLocationRelativeTo(null);
    }

    private void fillSensorInformation(){
        floorTextBox.setText(sensor.getFloor());
        roomTextBox.setText(sensor.getRoom());
    }

    private void updateLabels(){
        dialogLabel.setText("Edit fire alarm");
        addBtn.setText("Update");
    }

    public String getFloor(){
        return this.floorTextBox.getText();
    }

    public String getRoom(){
        return this.roomTextBox.getText();
    }

    public boolean getIsEditMode(){
        return this.editMode;
    }
    public boolean getFormSubmitted(){
        return this.formSubmitted;
    }

private void addBtnActionPerformed(java.awt.event.ActionEvent evt) {
        formSubmitted = true;
        this.setVisible(false);
    }

    private void cancelBtnActionPerformed(java.awt.event.ActionEvent evt) {
        this.setVisible(false);
    }

    private void roomTextBoxActionPerformed(java.awt.event.ActionEvent evt) {
        // TODO add your handling code here:
    }


    // Variables declaration - do not modify
    private javax.swing.JButton addBtn;
    private javax.swing.JButton cancelBtn;
    private javax.swing.JLabel dialogLabel;
    private javax.swing.JTextField floorTextBox;
    private javax.swing.JLabel jLabel1;
    private javax.swing.JLabel jLabel2;
    private javax.swing.JLabel jLabel4;
    private javax.swing.JLabel jLabel5;
    private javax.swing.JTextField roomTextBox;
    // End of variables declaration
}
```

- Source code of the Sensor Emulator(This project was created through Netbeans and the auto generated UI code is not included in here)

```
//********  EmulatorWindow.java

package fire.alarm.emulator;

import firealarm.rmi.api.FireAlarmSensor;
import java.awt.event.ItemEvent;
import java.io.IOException;
import java.util.ArrayList;
import java.util.List;
import java.util.concurrent.Executors;
import java.util.concurrent.ScheduledExecutorService;
import java.util.concurrent.TimeUnit;
import java.util.stream.Collectors;
import javax.json.*;



/**
 *
 * @author nuwan
 */
public class EmulatorWindow extends javax.swing.JFrame implements Runnable{

    private final static String FIRE_ALARM_URL = "http://localhost:5000/fire-alarm";

    private List<FireAlarmSensor> alarmList;
    private FireAlarmSensor selectedSensor = null;

    /**
     * Creates new form EmulatorWindow
     */
    public EmulatorWindow() {

        initComponents();

        loadFireAlarms();
    }

    private void loadFireAlarms() {
        StringBuffer res = null;
        try {
            res = APIHelper.get(FIRE_ALARM_URL);
        } catch (IOException e) {
            e.printStackTrace();
        }
        if (res == null) {
            return;
        }

        // parse the response
        JsonArray jo = JsonHelper.getJsonArrayFromString(res.toString());

        // get fire alarm sensor objects from json
        alarmList = jo.stream().map(alarmJson -> {

            return JsonHelper.getFireAlamSensorFromJson((JsonObject) alarmJson);
        }).collect(Collectors.toList());

        populateComboBox();

    }

    private void populateComboBox() {
        sensorCombo.removeAllItems();
        for (FireAlarmSensor fireAlarmSensor : alarmList) {
            sensorCombo.addItem(fireAlarmSensor.getFloor() + " floor " +
fireAlarmSensor.getRoom() + " room");

        }
    }

    private void populateEmulatePanel() {
        if (selectedSensor == null) {
            return;
        }

        alarmNameLabel.setText(selectedSensor.getFloor() + " floor " +
selectedSensor.getRoom() + " room");
        if(selectedSensor.getIsActive()){
            activeStatusBtn.setText("Turn off");
            activeStatusLabel.setText("ON");
        }else{
            activeStatusBtn.setText("Turn on");
            activeStatusLabel.setText("OFF");
        }
        activeStatusBtn.setSelected(selectedSensor.getIsActive());
        co2Slider.setValue(selectedSensor.getCo2Level());
        smokeSlider.setValue(selectedSensor.getSmokeLevel());
    }

    private void scheduleStatusUpdate(){

        // schedule fetching of sensors to run every 30 seconds
        ScheduledExecutorService executorService =
Executors.newSingleThreadScheduledExecutor();
        executorService.scheduleAtFixedRate(this, 0, 5, TimeUnit.SECONDS);
    }

    private void updateStatus(){
        System.out.println("fire.alarm.emulator.EmulatorWindow.updateStatus() +  isActive "
+ selectedSensor.getIsActive());
```

```
        int co2Level = co2Slider.getValue();
        int smokeLevel = smokeSlider.getValue();

        if(!selectedSensor.getIsActive()){
            co2Level = 0;
            smokeLevel = 0;
        }

          JsonObject updateFireAlarmParams = Json.createObjectBuilder()
               .add("is_active", selectedSensor.getIsActive())
               .add("smoke_level", smokeLevel)
               .add("co2_level", co2Level)
               .build();

        StringBuffer res = null;

        try {
            APIHelper.patch(FIRE_ALARM_URL + "/" + selectedSensor.getId(),
updateFireAlarmParams);
        } catch (IOException e) {
            e.printStackTrace();
        }

    }

    private void emulateBtnActionPerformed(java.awt.event.ActionEvent evt) {
        selectedSensor = alarmList.get(sensorCombo.getSelectedIndex());
        populateEmulatePanel();
        scheduleStatusUpdate();
        mainPain.setVisible(false);
        alarmPane.setVisible(true);
    }

    private void activeStatusBtnActionPerformed(java.awt.event.ActionEvent evt) {

    }

    private void activeStatusBtnItemStateChanged(java.awt.event.ItemEvent evt) {
        int newState = evt.getStateChange();
        if (newState == ItemEvent.SELECTED) {
            activeStatusBtn.setText("Turn off");
            activeStatusLabel.setText("ON");
            co2Slider.setEnabled(true);
            smokeSlider.setEnabled(true);
            selectedSensor.setIsActive(true);
        }else{
            activeStatusBtn.setText("Turn on");
            activeStatusLabel.setText("OFF");
            co2Slider.setEnabled(false);
            smokeSlider.setEnabled(false);
            selectedSensor.setIsActive(false);
        }
    }


    // Variables declaration - do not modify
    private javax.swing.JToggleButton activeStatusBtn;
    private javax.swing.JLabel activeStatusLabel;
    private javax.swing.JLabel alarmNameLabel;
    private javax.swing.JPanel alarmPane;
    private javax.swing.JSlider co2Slider;
    private javax.swing.JButton emulateBtn;
    private javax.swing.JLabel jLabel1;
    private javax.swing.JLabel jLabel3;
    private javax.swing.JLabel jLabel4;
    private javax.swing.JLabel jLabel5;
    private javax.swing.JSeparator jSeparator1;
    private javax.swing.JSeparator jSeparator2;
    private javax.swing.JPanel mainPain;
    private javax.swing.JComboBox<String> sensorCombo;
    private javax.swing.JSlider smokeSlider;
    // End of variables declaration

    @Override
    public void run() {
        updateStatus();
    }
}
//********  FireAlarmEmulator.java
package fire.alarm.emulator;

/**
 *
 * @author nuwan
 */
public class FireAlarmEmulator {

    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {
        // change the default theme
        try {
            for (javax.swing.UIManager.LookAndFeelInfo info :
javax.swing.UIManager.getInstalledLookAndFeels()) {
                if ("Windows".equals(info.getName())) {
                    javax.swing.UIManager.setLookAndFeel(info.getClassName());
                    break;
                }
            }
        } catch (Exception e) {

        }

        // start the emulator window
        EmulatorWindow emulatorWindow = new EmulatorWindow();
        emulatorWindow.setVisible(true);

    }

}
```