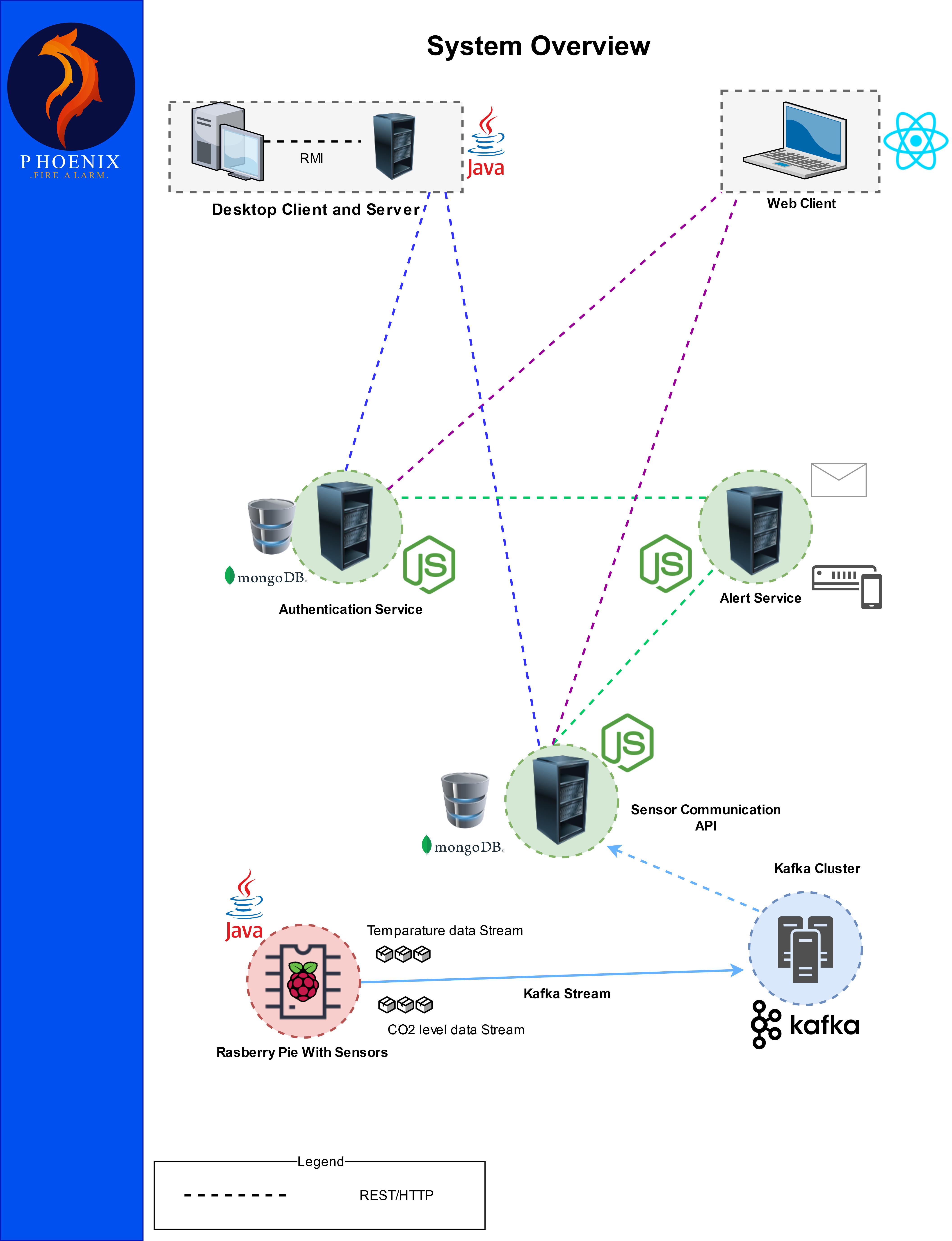
Fire Alarm Monitoring System Report

By:

* IT18050318 – M. A. Zeid
* IT17029896 – D. S. Jiffry
* IT18060690 – M. R. M. Rifan
* IT18200034 – M. A. F. Hasna

**Architectural Diagram**



**Services and their Interfaces Explained**

The System uses mongodb as it’s database mechanism, each service that uses it will have their own instance of mongodb.

1. RMI Server:

* + - * The server exposes the following to the RMI Client:
        + public boolean addSensor(int floorNumber, int roomNumber) throws RemoteException;
        + public boolean removeSensor(int floorNumber, int roomNumber) throws RemoteException;
        + public boolean changeState(int floorNumber, int roomNumber, boolean state) throws RemoteException;
        + public ArrayList<Sensor> viewSensors() throws RemoteException;
        + public boolean login(String username, String password) throws RemoteException;
        + public boolean checkAuthenticationServer() throws RemoteException;
      * The Client will first use checkAuthenticationServer() to make sure the Authentication service is reachable.
      * The login(…) will make the RMI Server validate the login by contacting the Authentication Server.
      * The addSensor(...), removeSensor(…) and changeState(…) are used to manage the sensors. The client will make the request to the RMI server who then talks to the Sensor Communication API and gets the reply which is then sent to the RMI Client.
      * ViewSensors() will make the RMI Server retrieve all the sensors in the Sensor Communication API for the Client. This is called every 30s to keep the data updated.

2. Authentication Service:

* <http://localhost:8080/loginUser> – Will validate the login of the user, expects a POST request with JSON keys “username” and “password”. Will return status 200 if login is valid otherwise status 404.
* <http://localhost:8080/register> – Will register a new user to the system, expects a POST request with JSON keys "username", "password", "email", "phoneNumber"and "type". (“type” can only have values of either “admin” or “user”). Passwords will be hash-coded before added to the database to avoid storing of plain-text passwords. Will return status 404 if username is taken, otherwise status 200 if register is sucessful.
* <http://localhost:8080/getEmails> – Will return a JSON array containing the emails of all the registered users, this is for the use of the Alert service to send out emails.
* <http://localhost:8080/getPhoneNumbers> – Will return a JSON array containing the phone numbers of all the registered users, this for use of the Alert Service to send out SMS.
* <http://localhost:8080/checkAuthenticationAlive> – Can be used to make sure the Authentication service is reachable. Will return status 200 if called.

3. Alert service:

* In our system we have identified the Alert Service and the sensor API as the critical services. In case one of the other services was down when a fire started we wanted to make sure that the alert would still go out to the users. To do this we implemented a local caching mechanism. When the alert service first comes online it would get the emails and phone numbers of the users from the Authentication service and store them locally.
* In case a fire occurred the sensor API would call on the Alert Service and then the alert service would fetch the emails and numbers from the Authentication service. Now in case the Authentication service is down when the Alert service needs the details. It would send out the emails and messages to the addresses and numbers it has cached. This way we can guarantee that even if all the users don’t get notified at least some of them will be. Since in this kind of safety system it is better to at least notify some people than none at all.
* This mechanism prevents our Alert service from being completely dependent on the Authentication service. When requesting emails and numbers of the Authentication service is reachable then the cache would also be updated.
* <http://localhost:8081/emailAlert> – Will send out an email alert. Expects a POST request with the JSON key “message”.
* <http://localhost:8081/smsAlert> – Will send out an SMS alert. Expects a POST request with the JSON key “message”.

**Appendix**

Alert Service:

let express = require("express");  
let app = express();  
let nodemailer = require('nodemailer');  
  
let fetch = require("node-fetch");  
app.use(express.json());  
  
let getEmailsURL = "http://localhost:8080/getEmails";  
let getphoneURL = "http://localhost:8080/getPhoneNumbers";  
  
// In case the authentication server is unreachable, the email/sms will be sent to these.  
let emailCache = [];  
let smsCache = [];  
  
/\*\*  
 \* Send email Alert  
 \* POST to http://localhost:8081/emailAlert  
 \* with JSON keys: "message"  
 \*/  
app.post('/emailAlert', async (req, res) => {  
 let message = req.body.message; // Body of the Email  
 let emailAddresses;  
  
 if (!(message)) {  
 return res.status(404).send('Error in JSON body');  
 }  
  
 emailAddresses = await fetch(getEmailsURL, {  
 method: 'POST',  
 headers: {  
 'Accept': 'application/JSON',  
 'Content-Type': 'application/JSON',  
 }  
 }).then((response) => {  
 if (response.ok) {  
 return response.json();  
 } else {  
 throw new Error('Something went wrong');  
 }  
 }).catch((err) => {  
 });  
  
 if (!(emailAddresses)) {  
 emailAddresses = emailCache.slice();  
 }  
 else  
 {  
 emailCache = emailAddresses.slice();  
 }  
  
 let testAccount = await nodemailer.createTestAccount();  
  
 let transporter = nodemailer.createTransport({  
 host: "smtp.ethereal.email",  
 port: 587,  
 secure: false, // true for 465, false for other ports  
 auth: {  
 user: testAccount.user, // generated ethereal user  
 pass: testAccount.pass // generated ethereal password  
 }  
 });  
  
 let info = await transporter.sendMail({  
 from: '"Fire Alarm Service" <Alert@FireAlarm.com>', // sender address  
 to: emailAddresses, // list of receivers  
 subject: "Fire Alarm Service Alert", // Subject line  
 text: message, // plain text body  
 });  
  
 console.log("Email sent");  
 return res.status(200).send(nodemailer.getTestMessageUrl(info)); //Will return a URL to preview the email that was sent  
});  
  
/\*\*  
 \* Send SMS Alert  
 \* POST to http://localhost:8081/smsAlert  
 \* JSON keys: "message"  
 \*/  
app.post('/smsAlert', async (req, res) => {  
 let message = req.body.message;  
 let smsNumbers;  
  
 if (!(message)) {  
 return res.status(404).send('Error in JSON body');  
 }  
  
 smsNumbers = await fetch(getphoneURL, {  
 method: 'POST',  
 headers: {  
 'Accept': 'application/JSON',  
 'Content-Type': 'application/JSON',  
 }  
 }).then((response) => {  
 if (response.ok) {  
 return response.json();  
 } else {  
 throw new Error('Something went wrong');  
 }  
 }).catch((err) => {  
  
 });  
  
 if (!(smsNumbers)) {  
 smsNumbers = smsCache.slice();  
 }  
 else  
 {  
 smsCache = smsNumbers.slice();  
 }  
  
 console.log("Sms sent");  
 return res.status(200).send("Sms message: " + message + " sent to: " + smsNumbers);  
  
});  
  
/\*\*  
 \* Maintaining a cache in case authorization server is down during a fire  
 \*/  
async function updateCache() {  
  
 emailCache = await fetch(getEmailsURL, {  
 method: 'POST',  
 headers: {  
 'Accept': 'application/JSON',  
 'Content-Type': 'application/JSON',  
 }  
 }).then((response) => {  
 if (response.ok) {  
 return response.json();  
 } else {  
 throw new Error('Something went wrong');  
 }  
 }).catch((err) => {  
  
 });  
  
 smsCache = await fetch(getphoneURL, {  
 method: 'POST',  
 headers: {  
 'Accept': 'application/JSON',  
 'Content-Type': 'application/JSON',  
 }  
 }).then((response) => {  
 if (response.ok) {  
 return response.json();  
 } else {  
 throw new Error('Something went wrong');  
 }  
 }).catch((err) => {  
  
 });  
  
 if (!(smsCache)) {  
 smsCache = ["119"];  
 }  
 if (!(emailCache)) {  
 emailCache = ["damn@weScrewed.com"];  
 }  
  
}  
  
  
//Keep server running on port  
const port = 8081;  
app.listen(port, () => {  
 updateCache();  
 console.log(`Server running on port:${port}`);  
});

Authentication Service:

let express = require("express");  
let encode = require('hashcode').hashCode;  
let app = express();  
let mongoose = require('mongoose/');  
  
app.use(express.json());  
  
//MongoDB url  
let url = "mongodb://localhost:27017/Fire\_Alarm\_Authentication";  
  
//Defining a Schema  
let userSchema = mongoose.Schema({  
 username: String,  
 password: String,  
 email: String,  
 phoneNumber: String,  
 type: String // either "user" or "admin"  
});  
  
//compiling schema to model  
let userModel = mongoose.model('User', userSchema, 'Users');  
  
/\*\*  
 \* Login Function for users  
 \* POST to http://localhost:8080/loginUser  
 \* With JSON Keys : "username", "password"  
 \*/  
app.post('/loginUser', (req, res) => {  
 let username = req.body.username;  
 //Hashcoding password before checking with DB  
 let password = encode().value(req.body.password);  
  
 if (!(username) || password == 0) {  
 return res.status(404).send('Error in JSON body');  
 }  
  
 connectToDB();  
  
 userModel.findOne({ username: username, password: password, type:"user"}, (err, user) => {  
 if (err) {  
 console.log(err);  
 }  
  
 //If a record is found  
 if (user) {  
 return res.status(200).send('Valid Login');  
 }  
  
 //If no record found  
 if (!user) {  
 console.log('Incorrect Login Details');  
 res.status(404).send('Incorrect Login Details');  
 }  
 });  
  
  
});  
  
/\*\*  
 \* Login Function for admins  
 \* POST to http://localhost:8080/loginAdmin  
 \* With JSON Keys : "username", "password"  
 \*/  
app.post('/loginAdmin', (req, res) => {  
 let username = req.body.username;  
 //Hashcoding password before checking with DB  
 let password = encode().value(req.body.password);  
  
 if (!(username) || password == 0) {  
 return res.status(404).send('Error in JSON body');  
 }  
  
 connectToDB();  
  
 userModel.findOne({ username: username, password: password, type:"admin"}, (err, user) => {  
 if (err) {  
 console.log(err);  
 }  
  
 //If a record is found  
 if (user) {  
 return res.status(200).send('Valid Login');  
 }  
  
 //If no record found  
 if (!user) {  
 console.log('Incorrect Login Details');  
 res.status(404).send('Incorrect Login Details');  
 }  
 });  
  
  
});  
  
/\*\*  
 \* Register New User Function  
 \* POST to http://localhost:8080/register  
 \* With JSON Keys : "username", "password", "email", "phoneNumber", "type"  
 \*/  
app.post('/register', (req, res) => {  
  
 let username = req.body.username;  
 let email = req.body.email;  
 let phone = req.body.phoneNumber;  
 let type = req.body.type; // either "user" or "admin"  
  
 //Hashcoding password for security before storing   
 let password = encode().value(req.body.password);  
  
 if (!(username) || password == 0 || !(email) || !(phone) || !(type)) {  
 return res.status(404).send('Error in JSON body');  
 }  
  
 if(type != "admin" && type != "user")  
 {  
 return res.status(406).send('incorrect type in JSON body. must be either "admin" or "user"');  
 }  
  
 connectToDB();  
  
 userModel.findOne({ username: username }, (err, user) => {  
 if (err) {  
 console.log(err);  
 }  
  
 if (user) {  
 console.log('Username is taken');  
 res.status(404).send('Username is taken');  
 }  
  
 if (!user) {  
 //reference to DB  
 let DB = mongoose.connection;  
  
 //Creating a Document  
 let userDoc = new userModel({  
 username: username,  
 password: password,  
 email: email,  
 phoneNumber: phone,  
 type: type  
 });  
  
 //Saving to DB  
 userDoc.save(function (err, user) {  
 if (err) {  
 return console.error(err);  
 }  
 console.log(user.username + " added to DB");  
  
 });  
  
 //Sending client to login page  
 return res.status(200).send(username + ' added to Database');  
 }  
 });  
  
  
  
  
});  
  
/\*\*  
 \* Get all emails  
 \* POST to http://localhost:8080/getEmails  
 \*/  
app.post('/getEmails', (req, res) => {  
  
 connectToDB();  
  
 userModel.find({}, '-\_id email', function (err, user) {  
 if (err) {  
 console.log(err);  
 }  
  
 let emails = [];  
  
 user.forEach(element => {  
 emails.push(element.email);  
 });  
  
 return res.status(200).send(emails);  
  
 });  
  
  
});  
  
  
/\*\*  
 \* Get all phone numbers  
 \* POST to http://localhost:8080/getPhoneNumbers  
 \*/  
app.post('/getPhoneNumbers', (req, res) => {  
  
 connectToDB();  
  
 userModel.find({}, '-\_id phoneNumber', function (err, user) {  
 if (err) {  
 console.log(err);  
 }  
  
 let numbers = [];  
  
 user.forEach(element => {  
 numbers.push(element.phoneNumber);  
 });  
  
 return res.status(200).send(numbers);  
  
 });  
  
  
  
  
  
  
  
  
  
});  
  
  
app.post('/checkAuthenticationAlive',(req,res) => {  
 return res.status(200).send();  
});  
  
/\*\*  
 \* Connecting to the Database  
 \*/  
function connectToDB() {  
 mongoose.connect(url)  
 .then(function () {  
 console.log('Connected to MongoDB');  
 })  
 .catch(function (err) {  
 console.log('Error in Connecting to MongoDB');  
 return;  
 });  
}  
  
// Keep server running on port  
const port = 8080;  
app.listen(port, () => {  
 console.log(`Server running on port:${port}`);  
});

RMI Client:

package Controllers;

import forms.Alert;

import java.net.MalformedURLException;

import java.rmi.Naming;

import java.rmi.NotBoundException;

import java.rmi.RemoteException;

import java.util.ArrayList;

import model.Sensor;

public class Client {

private ServerInterface service;

public Client() {

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

if (System.getSecurityManager() == null) {

System.setSecurityManager(new SecurityManager());

}

try {

service = (ServerInterface) Naming.lookup("rmi://localhost/rmiServer");

} catch (MalformedURLException | NotBoundException | RemoteException ex) {

Alert alert = new Alert("Connection error. Unable to reach RMI server");

}

}

/\*\*

\* Will as RMI server to add a server

\*

\* @param floorNumber

\* @param roomNumber

\* @return true if request was successful

\*/

public boolean addSensor(int floorNumber, int roomNumber) {

try {

return service.addSensor(floorNumber, roomNumber);

} catch (RemoteException ex) {

return false;

}

}

/\*\*

\* Will as RMI server to change the state of a sensor

\*

\* @param floorNumber

\* @param roomNumber

\* @param state true will make the sensor active, false will make it inactive

\* @return true if request was successful

\*/

public boolean changeState(int floorNumber, int roomNumber, boolean state) {

try {

return service.changeState(floorNumber, roomNumber, state);

} catch (RemoteException ex) {

return false;

}

}

/\*\*

\* Will make ask RMI server to delete a sensor

\*

\* @param floorNumber

\* @param roomNumber

\* @return true if request was successful

\*/

public boolean removeSensor(int floorNumber, int roomNumber) {

try {

return service.removeSensor(floorNumber, roomNumber);

} catch (RemoteException ex) {

return false;

}

}

/\*\*

\* Will request all the sensors in the database from the RMI server

\*

\* @return ArrayList of all the Sensors

\*/

public ArrayList<Sensor> getSensors(){

try {

return service.viewSensors();

} catch (RemoteException ignored) {

}

return new ArrayList<>();

}

/\*\*

\* Will check the login of the user

\*

\* @param username

\* @param password

\* @return true if login was successful

\*/

public boolean login(String username, String password) {

try {

return service.login(username, password);

} catch (RemoteException ex) {

return false;

}

}

/\*\*

\* Will check if the authorization server is reachable

\*

\* @return true if reachable

\*/

public boolean checkAuthenticationServer() {

try {

return service.checkAuthenticationServer();

} catch (RemoteException ex) {

return false;

}

}

}

RMI Server:

package Controllers;

import java.io.IOException;

import java.io.UnsupportedEncodingException;

import java.rmi.RemoteException;

import java.rmi.registry.LocateRegistry;

import java.rmi.registry.Registry;

import java.rmi.server.UnicastRemoteObject;

import java.util.ArrayList;

import java.util.HashMap;

import java.util.Map;

import org.apache.http.HttpResponse;

import org.apache.http.client.ClientProtocolException;

import org.apache.http.client.HttpClient;

import org.apache.http.client.methods.HttpDelete;

import org.apache.http.client.methods.HttpGet;

import org.apache.http.client.methods.HttpPost;

import org.apache.http.client.methods.HttpPut;

import org.apache.http.entity.StringEntity;

import org.apache.http.impl.client.BasicResponseHandler;

import org.apache.http.impl.client.HttpClientBuilder;

import model.Sensor;

import com.google.gson.Gson;

public class Server extends UnicastRemoteObject implements ServerInterface {

private final String SENSOR\_API\_URL = "https://webhook.site/72edc2d8-fd3a-4645-8b58-281572404a4e";

public Server() throws RemoteException {

super();

}

public static void main(String[] args) {

// Starting the RMI registry

try {

LocateRegistry.createRegistry(1099);

} catch (RemoteException ignored) {

// Means RMI registry is already running

}

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

try {

ServerInterface server = new Server();

Registry registry = LocateRegistry.getRegistry();

registry.bind("rmiServer", server);

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

} catch (Exception e) {

System.err.println(e.getMessage());

}

}

/\*\*

\* Will make a JSON request to add a Sensor to the database

\*

\* @return true if sucessful

\*/

@Override

public boolean addSensor(int floorNumber, int roomNumber) {

Map<String, String> body = new HashMap<>();

body.put("floorNumber", String.valueOf(floorNumber));

body.put("roomNumber", String.valueOf(roomNumber));

return makeRequest(body, "POST", SENSOR\_API\_URL);

}

/\*\*

\* Will make a JSON request to remove a Sensor from the database

\*

\* @return true if sucessful

\*/

@Override

public boolean removeSensor(int floorNumber, int roomNumber) throws RemoteException {

Map<String, String> body = new HashMap<>();

body.put("floorNumber", String.valueOf(floorNumber));

body.put("roomNumber", String.valueOf(roomNumber));

return makeRequest(body, "DELETE", SENSOR\_API\_URL); //TODO: add params to url

}

/\*\*

\* Will make a JSON request to change the state of a Sensor (active or inactive)

\*

\* @return true if sucessful

\*/

@Override

public boolean changeState(int floorNumber, int roomNumber, boolean state) throws RemoteException {

Map<String, String> body = new HashMap<>();

body.put("floorNumber", String.valueOf(floorNumber));

body.put("roomNumber", String.valueOf(roomNumber));

body.put("state", String.valueOf(state));

return makeRequest(body, "PUT", SENSOR\_API\_URL);

}

/\*\*

\* Will make a JSON request to retreive all sensors on database

\*

\* @return sensor Arraylist of the sensor details

\*/

@Override

public ArrayList<Sensor> viewSensors() throws RemoteException {

ArrayList<Sensor> sensors = new ArrayList<>();

HttpClient httpClient = HttpClientBuilder.create().build();

HttpGet get = new HttpGet(SENSOR\_API\_URL);

get.setHeader("Content-type", "application/json");

get.setHeader("Accept", "application/json");

HttpResponse response = null;

String responseString = null;

try {

response = httpClient.execute(get);

responseString = new BasicResponseHandler().handleResponse(response);

} catch (ClientProtocolException e) {

e.printStackTrace();

} catch (IOException e) {

e.printStackTrace();

}

for (String string : responseString.split("\".\*\":\\s\\{")) {

string = string.replaceAll("[{}]|(\\r\\n|\\r|\\n).\*}.\*,", "");

if (string.trim().length() == 0) {

continue;

}

string = "{\n" + string.replace("},", "").trim() + "\n}";

Sensor sensor = new Gson().fromJson(string, Sensor.class);

sensors.add(sensor);

}

return sensors;

}

/\*\*

\* Checks the login with the authorization api

\* @param username

\* @param password

\* @return true if valid login

\*/

@Override

public boolean login(String username, String password) throws RemoteException {

Map<String, String> body = new HashMap<>();

body.put("username", String.valueOf(username));

body.put("password", String.valueOf(password));

return makeRequest(body, "POST", "http://localhost:8080/loginAdmin");

}

/\*\*

\* Makes sure the authorization server is reachable

\* @return true if reachable

\*/

@Override

public boolean checkAuthenticationServer() throws RemoteException {

Map<String, String> body = new HashMap<>();

return makeRequest(body, "POST", "http://localhost:8080/checkAuthenticationAlive");

}

/\*\*

\* Used by add/remove/change methods to make requests.

\*/

public boolean makeRequest(Map<String, String> body, String RequestType, String URL) {

HttpResponse response = null;

try {

HttpClient httpClient = HttpClientBuilder.create().build();

StringEntity postingString = new StringEntity(new Gson().toJson(body));

switch (RequestType) {

case "POST":

HttpPost post = new HttpPost(URL);

post.setEntity(postingString);

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

response = httpClient.execute(post);

break;

case "PUT":

HttpPut put = new HttpPut(URL);

put.setEntity(postingString);

put.setHeader("Content-type", "application/json");

response = httpClient.execute(put);

break;

case "DELETE":

HttpDelete delete = new HttpDelete(URL);

delete.setHeader("Content-type", "application/json");

response = httpClient.execute(delete);

break;

}

} catch (UnsupportedEncodingException e) {

e.printStackTrace();

} catch (ClientProtocolException e) {

e.printStackTrace();

} catch (IOException e) {

e.printStackTrace();

}

if (response == null || response.getStatusLine().getStatusCode() > 399) {

return false;

}

return true;

}

private static final long serialVersionUID = 1L;

}