FIRE SENSOR MONITORING APPLICATION

1. IT18140194@my.sliit.lk - M.R.M. Tharik
2. IT18149036@my.sliit.lk - V.M. Siriwardena
3. IT18153132@my.sliit.lk - L.M.D. Perera
4. IT18109122@my.sliit.lk - S. Sharanjan

Date 4/5/2020

Group details

Distributed system

ASSIGNMENT 2

Contents

[**Introduction** 2](#_Toc39609886)

[**High Level Architectural Diagram** 3](#_Toc39609887)

[**Desktop Client** 4](#_Toc39609888)

[**Sensor Application** 8](#_Toc39609889)

[**Web Client** 9](#_Toc39609890)

[**Appendix** 11](#_Toc39609891)

# **Introduction**

This project is based on a fire alarm monitoring system. The fire alarm monitoring system developed by our team consists of four main components namely desktop application, web client, fire sensor application and a Rest API.

When talking about the technologies used to develop these components, the desktop application was built using JAVA swing whereas the web application was built using React JS. Laravel PHP framework was used to build the REST API with the help of the passport plugin. The RMI server which stands in between the desktop application and the REST API was built using the JAVA programming language. GSON was used to convert the JSON data coming from the REST API to an object. The sensor application which is used to mimic the functionality of an actual sensor was built using electron.

This system can be used by two types of users, the client and the administrator. Both the administrator and the client are able to observe the alerts and receive emails and SMS notifications when the CO2 level or smoke level values are greater than 5. Several security mechanisms have been adopted to ensure that only an administrator has the authority to add, edit and delete sensors. These security mechanisms and the other validations adopted will be broadly explained in the subsequent chapters.

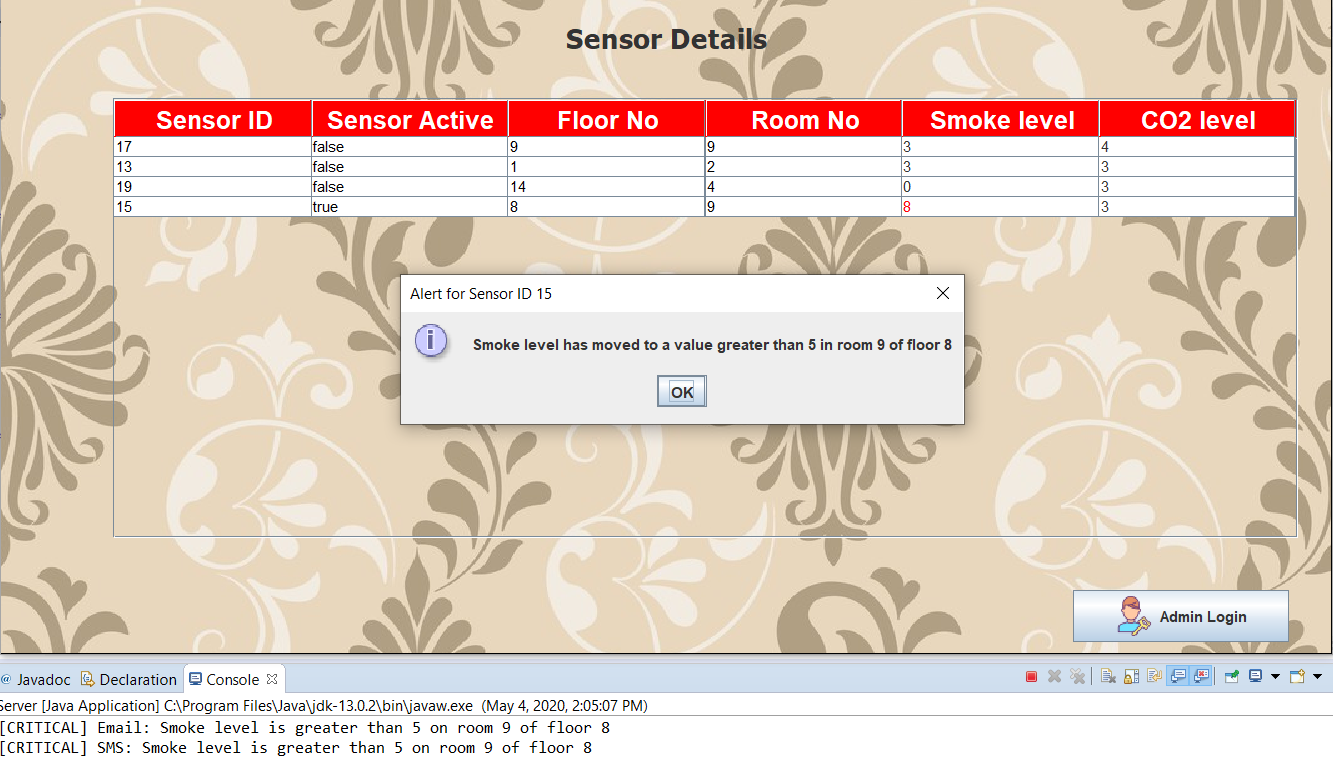
# **High Level Architectural Diagram**

**A close up of a piece of paper

Description automatically generated**

# **Desktop Client**

In the desktop client, a table consisting of all sensor details is displayed to the user. The data in the table is refreshed every 15 seconds and an alert will pop up when the CO2 level or Smoke level moves to a value greater than 5. If the CO2 level or Smoke level is greater than 5, they are displayed in red and an email and sms notification is sent in such an occasion as shown below in the diagram.

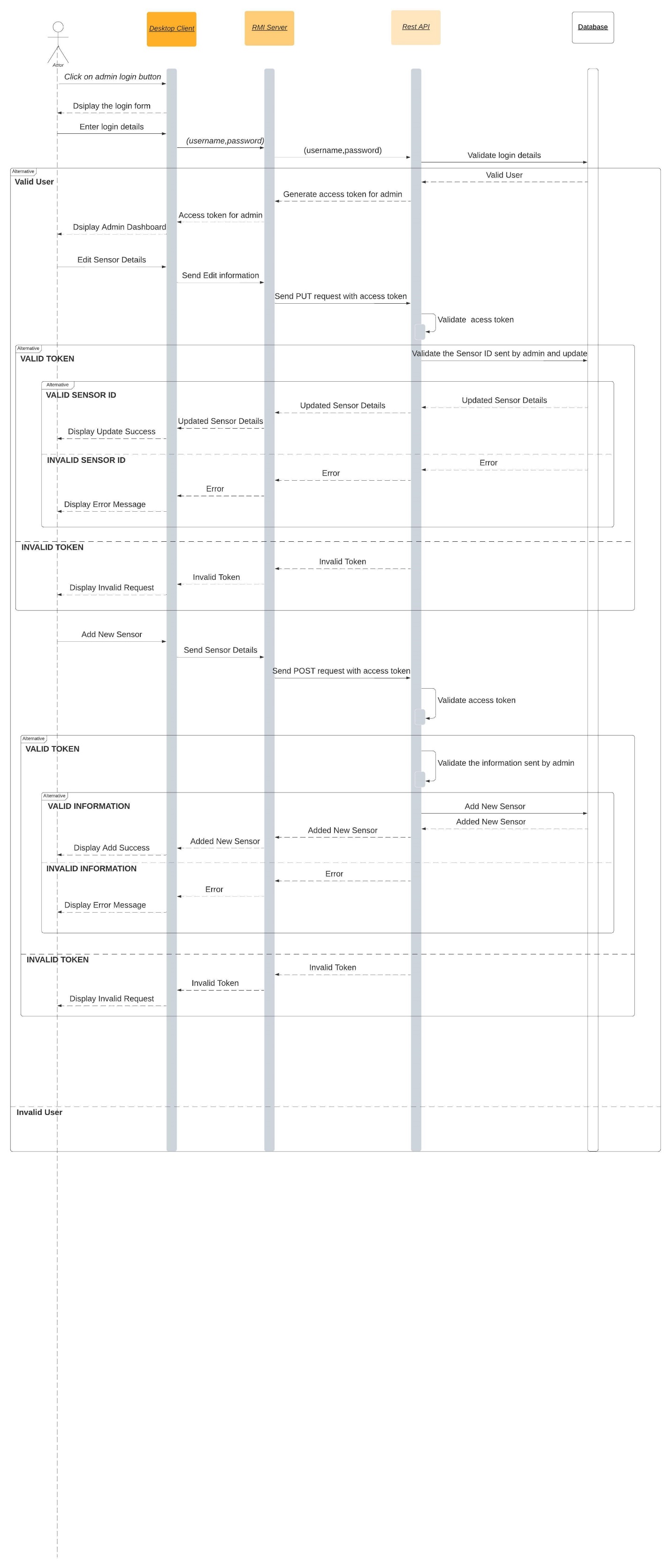


By clicking on the “Admin Login” button, the user is provided with a login form where the user has to enter the username and password in order to login as an administrator. After validating the user, the REST API generates an access token for admin. Afterwards, the user is provided with an admin dashboard where the user is given the administrative privileges to add, edit and delete the sensor details. Admin has the authority to edit the floor number and room number of the sensors. The admin dashboard also consists of a table consisting of all the sensor details which is refreshed every 15 seconds.

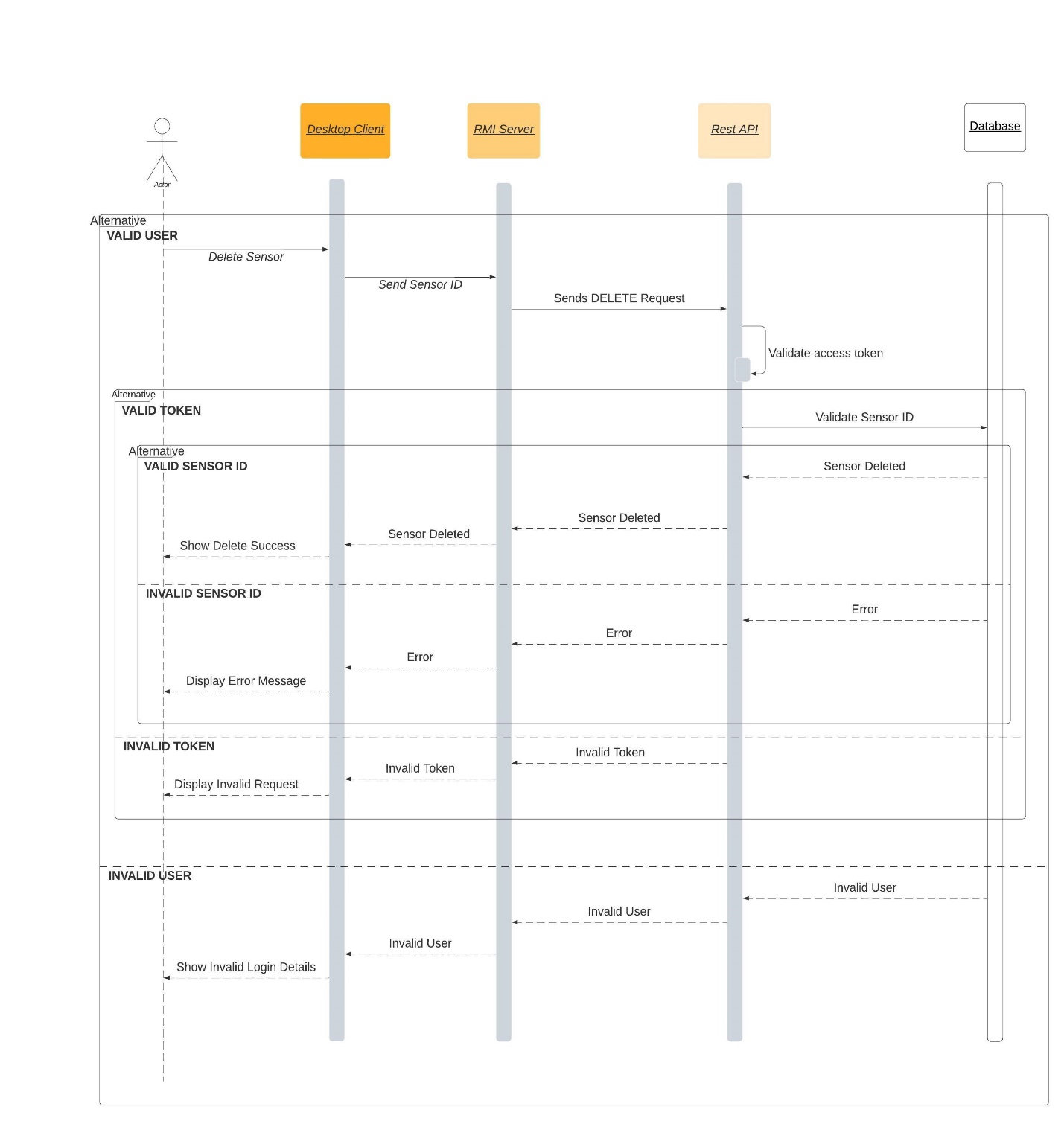
All the forms related to login, add, edit and delete are validated so that the user can submit only valid data. For example, when adding and editing sensors, the admin is only able to input numerical values to the floor number and room number input fields. Validations have also been applied to ensure that empty forms cannot be submitted.

API: https://fire-alarm-api-ds.herokuapp.com

Below sequence diagram explains the workflow of the desktop client.



**…. continuation of the previous sequence diagram**



As shown in the above sequence diagram, when the user submits the login form with the login details, the entered login details are validated. If such an user exists in the database, an access token which is specific to an admin is generated at the REST API and an admin dashboard is displayed to the user giving the administrative privileges to add, edit and delete sensor details. These administrative privileges are given to the user after validating the admin access token, and if it is invalid, the user is unable to add, edit and delete sensor details.

**API Response for Login Request:**

**A screenshot of a cell phone

Description automatically generated**

**Adding a new Sensor**

When a user tries to add a new sensor, the desktop client sends the sensor details entered by the user to the RMI. Thereafter, the RMI sends those information to the webserver which hosts the REST API as a POST request with the admin access token which was received during login. Then on the web server, the admin access token is validated first to ensure that the request is authorized to add a new sensor. If the authorization fails, an error message is sent to the RMI. If the authorization is successful, then the information entered by user is sent for validation. If the validation is successful, then the new sensor details are added to the database and new sensor ID is generated automatically.

A screenshot of a cell phone

Description automatically generated

**Editing a Sensor**

In the event of editing the sensor details, the desktop client sends the information entered by the user to the RMI. Thereafter, the RMI sends those information to the webserver as PUT request with the access token of the admin. Inside the webserver, the admin access token is validated to ensure that the admin has the authorization to send a PUT request. After successfully validating the admin access token, the information entered by user is passed for validation. During the validation, it is checked whether a sensor with the ID sent by the admin exists. If such a sensor exits, the sensor details of that specific sensor is updated. If there is no sensor matching the sensor ID sent by the admin, then an error message is displayed.

A screenshot of a cell phone

Description automatically generated

**Deleting a Sensor**

In case the user wants to delete a sensor, the desktop client sends the id of the sensor that needs to be deleted to the RMI. Afterwards, the RMI sends the information to the web server as a DELETE request with the admin access token. Then in the server, the admin access token is validated to ensure that the request is authorized to delete an existing sensor. If the authorization fails, an error message is sent to the RMI server. If the authorization is successful, the sensor ID that needs to be deleted is sent for validation. Here, the server checks whether a sensor with the given ID exists in the database, and if such a sensor exists, that particular sensor is deleted and a success message is sent back as a response. If such a sensor does not exist, then an error message is sent as a response to the RMI server.

A screenshot of a cell phone

Description automatically generated

# **Sensor Application**

Sensor application mimics the functionality of an actual sensor. It sends Sensor Data such as CO2 and Smoke levels, to the REST API every 10 seconds. Before sending the data, the sensor needs to be authenticated as valid sensor. If the sensor is not a sensor which is registered by the admin, then the authentication will fail, but if you get room number or the floor number is wrong, then the sensor will auto correct itself. You will be able to change the data manually.

A screenshot of a cell phone

Description automatically generated

**Response of a server PUT request sent by sensor to update the CO2 and Smoke level.**

**A screenshot of a cell phone

Description automatically generated**

**How the REST API determine if the sensor is active or inactive**

The API records the time stamp of the last request sent by the Sensor along with the levels. Then if there is a time gap more than 10 seconds between the current time and the last sensor data updated time, then the server decides that the sensor is inactive.

# **Web Client**

In the web client application, users can view the status of all fire alarm sensors. For each sensor, the web application displays whether the fire alarm sensor is active, the location (floor no, room no), and the smoke level (1-10) and the CO2 level (1-10 scale).These details are retrieved form the Rest Api. If the smoke level or CO2 level is above 5, then that particular sensor is marked in red. This application is refreshed every 40 seconds.

**Sequence diagram for the web client Application**

A close up of text on a white background

Description automatically generated

**Fetching Data from the Rest API every 40s by the web Client**

A screenshot of a cell phone

Description automatically generated

**Displaying sensor According to the smoke and co2 level**

A screen shot of a computer

Description automatically generated

# **Appendix**

**REST API**

**api.php**

<?php

use App\SensorInfo;

use Illuminate\Http\Request;

use Illuminate\Support\Facades\Route;

/\*

|--------------------------------------------------------------------------

| API Routes

|--------------------------------------------------------------------------

|

| Here is where you can register API routes for your application. These

| routes are loaded by the RouteServiceProvider within a group which

| is assigned the "api" middleware group. Enjoy building your API!

|

\*/

Route::post('/login', 'AuthLoginController@login')->name('api-login-route');

Route::middleware('auth:api')->get('/user', function (Request $request) {

return $request->user();

});

Route::middleware('auth:api')->get('/get', function (){

return \App\User::all()->toJson();

});

Route::resource("/sensorinfo", "SensorInfoController");

Route::get('/isregistered/{id}', 'SensorInfoController@isRegistered');

Route::middleware('auth:api')->put('/update/{sensor}', 'SensorInfoController@adminUpdate');

Route::middleware('auth:api')->post('/sensorinfo', 'SensorInfoController@store');

Route::middleware('auth:api')->delete('/sensorinfo/{sensor}', 'SensorInfoController@destroy');

**Channels.php**

<?php

use Illuminate\Support\Facades\Broadcast;

/\*

|--------------------------------------------------------------------------

| Broadcast Channels

|--------------------------------------------------------------------------

|

| Here you may register all of the event broadcasting channels that your

| application supports. The given channel authorization callbacks are

| used to check if an authenticated user can listen to the channel.

|

\*/

Broadcast::channel('App.User.{id}', function ($user, $id) {

return (int) $user->id === (int) $id;

});

**Console.php**

<?php

use Illuminate\Foundation\Inspiring;

use Illuminate\Support\Facades\Artisan;

/\*

|--------------------------------------------------------------------------

| Console Routes

|--------------------------------------------------------------------------

|

| This file is where you may define all of your Closure based console

| commands. Each Closure is bound to a command instance allowing a

| simple approach to interacting with each command's IO methods.

|

\*/

Artisan::command('inspire', function () {

$this->comment(Inspiring::quote());

})->describe('Display an inspiring quote');

**Web.php**

<?php

use Illuminate\Support\Facades\Route;

/\*

|--------------------------------------------------------------------------

| Web Routes

|--------------------------------------------------------------------------

|

| Here is where you can register web routes for your application. These

| routes are loaded by the RouteServiceProvider within a group which

| contains the "web" middleware group. Now create something great!

|

\*/

//Route::post('/login', 'AuthLoginController@login')->name('api-login-route');

Route::get('/', function () {

return view('welcome');

});

Auth::routes();

Route::match(['get', 'post'], 'login', function(){

return redirect('/');

});

Route::match(['get', 'post'], 'register', function(){

return redirect('/');

});

Route::get('/home', 'HomeController@index')->name('home');

**Sensorinfo.php**

<?php

namespace App;

use Carbon\Carbon;

use Illuminate\Database\Eloquent\Model;

class SensorInfo extends Model

{

protected $guarded = [];

protected $appends = ["is\_active"];

// if the difference between the last value updated time and the current time is greater than 10 Sec,

// assume the sensor as inactive

public function getIsActiveAttribute($value){

// if updated\_at is null, then return false

if($this->updated\_at === null) return false;

$timeGap = Carbon::now()->diffInSeconds($this->updated\_at);

$TIME\_GAP = 10;

if($timeGap > $TIME\_GAP){

return false;

}else{

return true;

}

}

}

**SensorInfoController.php**

<?php

namespace App\Http\Controllers;

use App\SensorInfo;

use Carbon\Carbon;

use Illuminate\Http\Request;

use function GuzzleHttp\Promise\all;

class SensorInfoController extends Controller

{

/\*\*

\* Display a listing of the resource.

\*

\* @return SensorInfo[]|\Illuminate\Database\Eloquent\Collection

\*/

public function index()

{

return SensorInfo::all();

}

/\*\*

\* Store a newly created resource in storage.

\*

\* @param \Illuminate\Http\Request $request

\* @return SensorInfo

\*/

public function store(Request $request)

{

// Server side validation

$validatedData = $request->validate([

'floor\_no' => 'required|numeric|min:0',

'room\_no' => 'required|numeric|min:0'

]);

// only the room number and the floor number are required for SensorRegistration(other values are updated separately)

$s = new SensorInfo();

$s->smoke\_level = 0;

$s->co2\_level = 0;

$s->room\_no = $validatedData["room\_no"];

$s->floor\_no = $validatedData["floor\_no"];

// $s->is\_active = false;

$s->updated\_at = null;

$s->save();

return $s;

}

/\*\*

\* Display the specified resource.

\*

\* @param \App\SensorInfo $sensorInfo

\* @return SensorInfo

\*/

public function show(SensorInfo $sensorinfo)

{

return $sensorinfo;

}

/\*\*

\* Display the specified resource.

\*

\* @param \App\SensorInfo $sensorInfo

\* @return array

\*/

public function isRegistered($id)

{

// find the sensor from the Id

$user = SensorInfo::find($id);

// if the Sensor with that ID is not available send back this

if($user === null) return ["isAvailable" => false];

// if the sensor is available send

return ["isAvailable" => true, "info" => $user];

}

/\*\*

\* Update the specified resource in storage.

\*

\* @param \Illuminate\Http\Request $request

\* @param \App\SensorInfo $sensorinfo

\* @return SensorInfo

\*/

public function update(Request $request, SensorInfo $sensorinfo)

{

$validatedData = $request->validate([

'smoke\_level' => 'required|numeric|min:0|max:10',

'co2\_level' => 'required|numeric|min:0|max:10'

]);

$sensorinfo->co2\_level = $validatedData["co2\_level"];

$sensorinfo->smoke\_level = $validatedData["smoke\_level"];

$sensorinfo->updated\_at = Carbon::now();

$sensorinfo->save();

return $sensorinfo;

}

/\*\*

\* Update the specified attributes as admin resource in storage.

\*

\* @param \Illuminate\Http\Request $request

\* @param \App\SensorInfo $sensorinfo

\* @return SensorInfo

\*/

public function adminUpdate(Request $request, SensorInfo $sensor)

{

$validatedData = $request->validate([

'floor\_no' => 'required|numeric|min:0',

'room\_no' => 'required|numeric|min:0'

]);

$sensor->room\_no = $validatedData["room\_no"];

$sensor->floor\_no = $validatedData["floor\_no"];

// make sure that the updated\_at timestamp on floor and room number updates

$sensor->timestamps = false;

$sensor->save();

return $sensor;

}

/\*\*

\* Remove the specified resource from storage.

\*

\* @param \App\SensorInfo $sensor

\* @return SensorInfo

\* @throws \Exception

\*/

public function destroy(SensorInfo $sensor)

{

$sensor->delete();

return $sensor;

}

}

**User.php**

<?php

namespace App;

use Illuminate\Contracts\Auth\MustVerifyEmail;

use Illuminate\Foundation\Auth\User as Authenticatable;

use Illuminate\Notifications\Notifiable;

use Laravel\Passport\HasApiTokens;

class User extends Authenticatable

{

use HasApiTokens, Notifiable;

/\*\*

\* The attributes that are mass assignable.

\*

\* @var array

\*/

protected $fillable = [

'name', 'email', 'password',

];

/\*\*

\* The attributes that should be hidden for arrays.

\*

\* @var array

\*/

protected $hidden = [

'password', 'remember\_token',

];

/\*\*

\* The attributes that should be cast to native types.

\*

\* @var array

\*/

protected $casts = [

'email\_verified\_at' => 'datetime',

];

}

**Migrations**

**1.create\_user\_table**

<?php

use Illuminate\Database\Migrations\Migration;

use Illuminate\Database\Schema\Blueprint;

use Illuminate\Support\Facades\Schema;

class CreateUsersTable extends Migration

{

/\*\*

\* Run the migrations.

\*

\* @return void

\*/

public function up()

{

Schema::create('users', function (Blueprint $table) {

$table->id();

$table->string('name');

$table->string('email')->unique();

$table->timestamp('email\_verified\_at')->nullable();

$table->string('password');

$table->rememberToken();

$table->timestamps();

});

// add the admin user

$user = new \App\User();

$user->name = "Admin";

$user->email = "admin@admin.com";

$user->password = bcrypt("password");

$user->save();

}

/\*\*

\* Reverse the migrations.

\*

\* @return void

\*/

public function down()

{

Schema::dropIfExists('users');

}

}

**2.create\_password\_reset\_table**

<?php

use Illuminate\Database\Migrations\Migration;

use Illuminate\Database\Schema\Blueprint;

use Illuminate\Support\Facades\Schema;

class CreatePasswordResetsTable extends Migration

{

/\*\*

\* Run the migrations.

\*

\* @return void

\*/

public function up()

{

Schema::create('password\_resets', function (Blueprint $table) {

$table->string('email')->index();

$table->string('token');

$table->timestamp('created\_at')->nullable();

});

}

/\*\*

\* Reverse the migrations.

\*

\* @return void

\*/

public function down()

{

Schema::dropIfExists('password\_resets');

}

}

**3.create\_failed\_jobs\_table**

<?php

use Illuminate\Database\Migrations\Migration;

use Illuminate\Database\Schema\Blueprint;

use Illuminate\Support\Facades\Schema;

class CreateFailedJobsTable extends Migration

{

/\*\*

\* Run the migrations.

\*

\* @return void

\*/

public function up()

{

Schema::create('failed\_jobs', function (Blueprint $table) {

$table->id();

$table->text('connection');

$table->text('queue');

$table->longText('payload');

$table->longText('exception');

$table->timestamp('failed\_at')->useCurrent();

});

}

/\*\*

\* Reverse the migrations.

\*

\* @return void

\*/

public function down()

{

Schema::dropIfExists('failed\_jobs');

}

}

**4.create\_sensor\_info\_table**

**<?php**

**use Illuminate\Database\Migrations\Migration;**

**use Illuminate\Database\Schema\Blueprint;**

**use Illuminate\Support\Facades\Schema;**

**class CreateSensorInfosTable extends Migration**

**{**

**/\*\***

**\* Run the migrations.**

**\***

**\* @return void**

**\*/**

**public function up()**

**{**

**Schema::create('sensor\_infos', function (Blueprint $table) {**

**$table->id();**

**$table->integer('smoke\_level');**

**$table->integer('co2\_level');**

**$table->integer('room\_no');**

**$table->integer('floor\_no');**

**$table->timestamps();**

**});**

**}**

**/\*\***

**\* Reverse the migrations.**

**\***

**\* @return void**

**\*/**

**public function down()**

**{**

**Schema::dropIfExists('sensor\_infos');**

**}**

**}**

**Fire Sensor Application (dummy sensor)**

**main.js**

const {app, BrowserWindow} = new require("electron");

function createWindow(){

let win = new BrowserWindow({

width: 430,

height: 530,

webIntegration:{

nodeIntegration: true

}

});

win.loadFile('index.html');

win.setMenu(null);

// win.setResizable(false);

}

app.whenReady().then(createWindow);

// Quit when all windows are closed.

app.on('window-all-closed', () => {

// On macOS it is common for applications and their menu bar

// to stay active until the user quits explicitly with Cmd + Q

if (process.platform !== 'darwin') {

app.quit()

}

});

app.on('activate', () => {

// On macOS it's common to re-create a window in the app when the

// dock icon is clicked and there are no other windows open.

if (BrowserWindow.getAllWindows().length === 0) {

createWindow()

}

});

**Frontend.js**

const form = document.getElementById('form');

const sendData = document.getElementById('sendData');

sendData.hidden = true;

let url, id, roomNo, floorNo;

const smoke = document.getElementById("smoke");

const co2 = document.getElementById("co2");

function onSubmit(e) {

e.preventDefault();

// alert();

url = e.target["api"].value + "/api";

id = e.target["id"].value;

roomNo = parseInt(e.target["roomNo"].value);

floorNo = parseInt(e.target["floorNo"].value);

return fetch(url + "/isregistered/" + id)

.then(response => response.json())

.then(res => {

if (res.isAvailable === true) {

if (res.info.room\_no !== roomNo || res.info.floor\_no !== floorNo) {

alert("Room No, or the Floor No. You entered was wrong, they have been corrected.");

// set the correct room no and floor no

roomNo = res.info.room\_no;

floorNo = res.info.floor\_no;

}

smoke.value = res.info.smoke\_level;

co2.value = res.info.co2\_level;

const roomFloor = document.getElementById('roomFloor');

roomFloor.innerText = `Room No: ${roomNo} | Floor No: ${floorNo}`;

updateStatus(co2.value, smoke.value);

sendData.hidden = false;

form.hidden = true;

alert("Authenticated!");

updateLevels();

return true;

}else{

alert(`Authentication of the Sensor ID: ${id} failed\nNo Sensor Registered with that ID(${id})`);

return false;

}

})

.catch(err => {

alert("Error: " + err);

return false;

});

}

let timer = null;

function updateLevels() {

timer = setInterval(() => {

const bodyO = {

smoke\_level: parseInt(smoke.value),

co2\_level: parseInt(co2.value)

};

const updateUri = `${url}/sensorinfo/${id}`;

fetch(updateUri, {

method: 'PUT',

headers: {

'Content-Type': 'application/json'

},

body: JSON.stringify(bodyO)

})

.then(response => response.json())

.then(res => {

updateStatus(res.co2\_level, res.smoke\_level);

})

.catch(err => {

console.log("Error: " + err);

alert("Error from the server...\nExit?");

clearTimeout(timer);

window.close();

})

}, 5000);

}

function updateStatus(co2, smoke) {

const status = document.getElementById('status');

if (smoke < 5 && co2 < 5) {

status.innerText = "Good";

status.className = "good";

} else if (smoke > 5 || co2 > 5) {

status.innerText = "Dangerous";

status.className = "danger";

} else {

status.innerText = "Average";

status.className = "";

}

}

**Index.js**

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<title>Sensor</title>

<link rel="stylesheet" href="node\_modules/bootstrap/dist/css/bootstrap.min.css">

<script defer src="fontend.js"></script>

<style>

.danger{

color: firebrick;

}

.good{

color: green;

}

</style>

</head>

<body style=" user-select: none;">

<div class="container">

<div id="form">

<h2 style="text-align: center">Enter Sensor Details:</h2>

<form onsubmit="return onSubmit(event)">

<div class="form-group">

<label for="api">API Link:</label>

<input type="text" class="form-control" id="api" name="api" aria-describedby="api" value="https://fire-alarm-api-ds.herokuapp.com" required>

</div>

<div class="form-group">

<label for="sensorId">Sensor ID:</label>

<input type="number" class="form-control" id="sensorId" name="id" aria-describedby="id" value="1" required>

</div>

<div class="form-group">

<label for="roomNo">Room No:</label>

<input type="number" class="form-control" id="roomNo" name="roomNo" aria-describedby="roomNo" required>

</div>

<div class="form-group">

<label for="floorNo">Floor No:</label>

<input type="number" class="form-control" id="floorNo" name="floorNo" aria-describedby="floorNo" required>

</div>

<div class="form-group">

<button type="submit" class="btn btn-primary form-control">Submit</button>

</div>

</form>

</div>

<div id="sendData">

<div class="mb-2" style="text-align: center">

<h1>Send Data</h1>

<p id="roomFloor"></p>

</div>

<div class="form-group">

<label for="smoke">Smoke Level:</label>

<input type="number" class="form-control" id="smoke" aria-describedby="smoke" min="0" max="10" required>

</div>

<div class="form-group">

<label for="co2">CO2 Level:</label>

<input type="number" class="form-control" id="co2" aria-describedby="co2" min="0" max="10" required>

</div>

<br>

<div style="text-align: center">

<h3>Status: </h3>

<h3 id="status"></h3>

</div>

</div>

</div>

</body>

</html>

**Desktop Application**

**Server.java**

**package** rmiserver;

**//code related to imports removed**

**public** **class** Server **extends** UnicastRemoteObject **implements** IServer{

/\*\*

\*

\*/

**private** **static** **final** **long** ***serialVersionUID*** = 1L;

// flag to remember which fire sensors have been already alerted

**private** HashMap<Integer, Boolean> hasAlertedCO2 = **new** HashMap<Integer, Boolean>();

**private** HashMap<Integer, Boolean> hasAlertedSmoke = **new** HashMap<Integer, Boolean>();

**protected** Server() **throws** RemoteException {

**super**();

}

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

// set the policy file as the system security policy

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

**try** {

Server svr = **new** Server();

Registry registry = LocateRegistry.*getRegistry*();

registry.rebind("RMIServer", svr);

System.***out***.println("Server started....");

} **catch** (RemoteException re) {

System.***err***.println(re.getMessage());

}

}

// IServer implementations

// All the WebRequest implementations are in WebRequest Class

@Override

**public** **boolean** login(String username, String password) {

// System.out.println("Calling login");

**try** {

**return** WebRequest.*checkLogin*(username, password);

} **catch** (IOException e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

}

**return** **false**;

}

@Override

**public** List<SensorInfo> getSensorInfo() {

List<SensorInfo> data = WebRequest.*getSensorInfo*();

**for**(SensorInfo s : data) {

**if**(!hasAlertedCO2.containsKey(s.id)) {

hasAlertedCO2.put(s.id, **false**);

}

**if**(!hasAlertedSmoke.containsKey(s.id)) {

hasAlertedSmoke.put(s.id, **false**);

}

**if** (s.is\_active) {

// check for CO2 Level

**if** (s.co2\_level > 5 && !hasAlertedCO2.get(s.id)) {

String msg = "CO2 level is greater than 5 on room "+ s.room\_no + " of floor "+s.floor\_no;

SendEmail(msg);

SendSMS(msg);

// update the flag

hasAlertedCO2.put(s.id, **true**);

}**else** **if**(s.co2\_level <= 5) {

// reset the flag

hasAlertedCO2.put(s.id, **false**);

}

// Check for Smoke Level

**if** (s.smoke\_level > 5 && !hasAlertedSmoke.get(s.id)) {

String msg = "Smoke level is greater than 5 on room "+ s.room\_no + " of floor "+s.floor\_no;

SendEmail(msg);

SendSMS(msg);

// update the flag

hasAlertedSmoke.put(s.id, **true**);

}**else** **if**(s.smoke\_level <= 5) {

// reset the flag

hasAlertedSmoke.put(s.id, **false**);

}

}

}

**return** data;

}

@Override

**public** **void** addNewSensor(SensorInfo sensorInfo) {

WebRequest.*addSensor*(sensorInfo);

}

@Override

**public** **void** updateSensor(**int** id, SensorInfo updatedSensorInfo) {

WebRequest.*updateSensorInfo*(id, updatedSensorInfo);

}

@Override

**public** **void** logout(String username, String password) **throws** RemoteException {

WebRequest.*logout*();

}

// delete the sensor by id

@Override

**public** **void** deleteSensor(**int** id) **throws** RemoteException {

WebRequest.*deleteSensorById*(id);

}

// method to send an email

**private** **void** SendEmail(String msg) {

System.***out***.println("[CRITICAL] Email: "+msg);

}

// method to send SMS

**private** **void** SendSMS(String msg) {

System.***out***.println("[CRITICAL] SMS: "+msg);

}

}

**Token.java**

**package** rmiserver;

**public** **class** Token {

**public** String error = **null**;

**public** String token\_type;

**public** String access\_token;

}

**TokenRequest.java**

**package** rmiserver;

**public** **class** TokenRequest {

**public** **final** String grant\_type;

**public** **final** String client\_id;

**public** **final** String client\_secret;

**public** **final** String username;

**public** **final** String password;

**public** TokenRequest(String grant\_type, String client\_id, String client\_secret, String username, String password) {

**super**();

**this**.grant\_type = grant\_type;

**this**.client\_id = client\_id;

**this**.client\_secret = client\_secret;

**this**.username = username;

**this**.password = password;

}

}

**WebRequest.java**

**package** rmiserver;

**//removed the code relating to imports**

**import** common.SensorInfo;

**public** **class** WebRequest {

// OAuth 2.0

**private** **static** String *adminToken* = "";

**private** **final** **static** String ***BASE\_URL*** = "https://fire-alarm-api-ds.herokuapp.com";

**private** **final** **static** String ***CLIENT\_ID*** = "2";

**private** **final** **static** String ***CLIENT\_SECRET*** = "geanPhlSATg433c1Uf7qbzBb7gSjhJeTSrlEoDdd";

// make web request to login

**public** **static** **boolean** checkLogin(String username, String password) **throws** IOException {

Gson gson = **new** Gson();

String requestBody = gson.toJson(**new** TokenRequest("password", ***CLIENT\_ID***, ***CLIENT\_SECRET***, username, password));

HttpClient client = HttpClient.*newHttpClient*();

HttpRequest request = HttpRequest.*newBuilder*()

.uri(URI.*create*(***BASE\_URL***+"/oauth/token"))

.POST(HttpRequest.BodyPublishers.*ofString*(requestBody))

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

.header("Accept", "application/json")

.build();

HttpResponse<String> response = **null**;

**try** {

response = client.send(request, HttpResponse.BodyHandlers.*ofString*());

} **catch** (IOException | InterruptedException e) {

e.printStackTrace();

}

**if**(response == **null**) **return** **false**;

String res = response.body();

Token token = gson.fromJson(res, Token.**class**);

**if**(token.error != **null**) {

System.***out***.println("Username or Password is wrong!!");

**return** **false**;

}

**else** {

*adminToken* = token.access\_token;

System.***out***.println("Admin authentication token has been retrieved...");

System.***out***.println("Authentication successful...");

**return** **true**;

}

}

// logout locally

**public** **static** **void** logout() {

*adminToken* = "";

}

// retrieve current list of sensor information

**public** **static** List<SensorInfo> getSensorInfo(){

String url = ***BASE\_URL***+"/api/sensorinfo";

HttpClient client = HttpClient.*newHttpClient*();

HttpRequest request = HttpRequest.*newBuilder*(URI.*create*(url)).build();

List<SensorInfo> data = client.sendAsync(request, HttpResponse.BodyHandlers.*ofString*())

.thenApply(HttpResponse::body)

.thenApply(WebRequest::*parse*)

.join();

**return** data;

}

// update current sensor information

**public** **static** **void** updateSensorInfo(**int** id, SensorInfo updatedSensor) {

// if not logged in, return

**if**(*adminToken*.length() == 0) {

System.***out***.println("Not logged in...");

**return**;

}

Gson gson = **new** Gson();

String requestBody = gson.toJson(updatedSensor);

HttpClient client = HttpClient.*newHttpClient*();

HttpRequest request = HttpRequest.*newBuilder*()

.uri(URI.*create*(***BASE\_URL***+"/api/update/"+id))

.PUT(HttpRequest.BodyPublishers.*ofString*(requestBody))

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

.header("Accept", "application/json")

.header("Authorization", "Bearer "+*adminToken*)

.build();

HttpResponse<String> response = **null**;

**try** {

response = client.send(request, HttpResponse.BodyHandlers.*ofString*());

} **catch** (IOException | InterruptedException e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

}

String res = response.body();

System.***out***.println("Updated = "+ res);

}

// add new sensor

**public** **static** **void** addSensor(SensorInfo newSensor) {

// if not logged in, return

**if**(*adminToken*.length() == 0) {

System.***out***.println("Not logged in...");

**return**;

}

Gson gson = **new** Gson();

String requestBody = gson.toJson(newSensor);

HttpClient client = HttpClient.*newHttpClient*();

HttpRequest request = HttpRequest.*newBuilder*()

.uri(URI.*create*(***BASE\_URL***+"/api/sensorinfo"))

.POST(HttpRequest.BodyPublishers.*ofString*(requestBody))

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

.header("Accept", "application/json")

.header("Authorization", "Bearer "+*adminToken*)

.build();

HttpResponse<String> response = **null**;

**try** {

response = client.send(request, HttpResponse.BodyHandlers.*ofString*());

} **catch** (IOException | InterruptedException e) {

e.printStackTrace();

}

String res = response.body();

System.***out***.println("Added = "+ res);

}

// delete the sensor by ID

**public** **static** **void** deleteSensorById(**int** id) {

// if not logged in, return

**if**(*adminToken*.length() == 0) {

System.***out***.println("Not logged in...");

**return**;

}

String url = ***BASE\_URL*** + "/api/sensorinfo/"+id;

HttpClient client = HttpClient.*newHttpClient*();

HttpRequest request = HttpRequest.*newBuilder*()

.uri(URI.*create*(url))

.DELETE()

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

.header("Accept", "application/json")

.header("Authorization", "Bearer "+*adminToken*)

.build();

HttpResponse<String> response = **null**;

**try** {

response = client.send(request, HttpResponse.BodyHandlers.*ofString*());

} **catch** (IOException | InterruptedException e) {

e.printStackTrace();

}

**if**(response == **null**) {

System.***out***.println("Error when sending the delete request...");

**return**;

}

String res = response.body();

**if**(res != **null**)System.***out***.println("Deleted the sensor "+ res);

**else** System.***out***.println("Error in deleting the sensor...");

}

// TESTING

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

// try {

// System.out.println("Access Granted: "+checkLogin("admin@admin.com", "password"));

//// addSensor(new SensorInfo(1,5,5,5,5,true, "",""));

//// updateSensorInfo(0, new SensorInfo(1,10,10,10,10,true, "",""));

// } catch (IOException e) {

// e.printStackTrace();

// }

// }

// parse the JSON data list

**private** **static** List<SensorInfo> parse(String json) {

Gson gson = **new** Gson();

Type collectionType = **new** TypeToken<List<SensorInfo>>(){}.getType();

List<SensorInfo> obj = gson.fromJson(json, collectionType);

**return** obj;

}

}

**Iserver.java**

**package** common;

//removed imports

**public** **interface** IServer **extends** Remote {

**public** **boolean** login(String username, String password)**throws** RemoteException;

**public** **void** logout(String username, String password)**throws** RemoteException;

**public** List<SensorInfo> getSensorInfo() **throws** RemoteException;

**public** **void** addNewSensor(SensorInfo sensorInfo)**throws** RemoteException;

**public** **void** updateSensor(**int** id, SensorInfo updatedSensorInfo)**throws** RemoteException;

**public** **void** deleteSensor(**int** id)**throws** RemoteException;

}

**SensorInfor.java**

**package** common;

**import** java.io.Serializable;

**public** **class** SensorInfo **implements** Serializable{

/\*\*

\*

\*/

**private** **static** **final** **long** ***serialVersionUID*** = 1L;

**public** **int** id;

**public** **int** smoke\_level;

**public** **int** co2\_level;

**public** **int** room\_no;

**public** **int** floor\_no;

**public** **boolean** is\_active;

**public** String created\_at;

**public** String updated\_at;

**public** SensorInfo() {

}

**public** SensorInfo(**int** id, **int** smoke\_level, **int** co2\_level, **int** room\_no, **int** floor\_no, **boolean** is\_active,

String created\_at, String updated\_at) {

**this**.id = id;

**this**.smoke\_level = smoke\_level;

**this**.co2\_level = co2\_level;

**this**.room\_no = room\_no;

**this**.floor\_no = floor\_no;

**this**.is\_active = is\_active;

**this**.created\_at = created\_at;

**this**.updated\_at = updated\_at;

}

**public** **int** getId() {

**return** id;

}

**public** **void** setId(**int** id) {

**this**.id = id;

}

**public** **int** getSmoke\_level() {

**return** smoke\_level;

}

**public** **void** setSmoke\_level(**int** smoke\_level) {

**this**.smoke\_level = smoke\_level;

}

**public** **int** getCo2\_level() {

**return** co2\_level;

}

**public** **void** setCo2\_level(**int** co2\_level) {

**this**.co2\_level = co2\_level;

}

**public** **int** getRoom\_no() {

**return** room\_no;

}

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

**this**.room\_no = room\_no;

}

**public** **int** getFloor\_no() {

**return** floor\_no;

}

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

**this**.floor\_no = floor\_no;

}

**public** **boolean** isIs\_active() {

**return** is\_active;

}

**public** **void** setIs\_active(**boolean** is\_active) {

**this**.is\_active = is\_active;

}

**public** String getCreated\_at() {

**return** created\_at;

}

**public** **void** setCreated\_at(String created\_at) {

**this**.created\_at = created\_at;

}

**public** String getUpdated\_at() {

**return** updated\_at;

}

**public** **void** setUpdated\_at(String updated\_at) {

**this**.updated\_at = updated\_at;

}

}

**Sensorinformation.java**

**package** clientApplication;

**//code related to imports removed**

**public** **class** Sensorinformation {

**//code relating to initializing JFrame and JTable removed**

**private** IServer service = **null**;

**private** **static** HashMap<Integer, Boolean> *hasAlertedCO2* = **new** HashMap<Integer, Boolean>();

**private** **static** HashMap<Integer, Boolean> *hasAlertedSmoke* = **new** HashMap<Integer, Boolean>();

/\*\*

\* Launch the application.

\*/

//auto generated code removed

/\*\*

\* Create the application.

\*/

**public** Sensorinformation() {

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

**try** {

service = (IServer) Naming.*lookup*("rmi://localhost/RMIServer");

} **catch** (NotBoundException ex) {

System.***err***.println(ex.getMessage());

} **catch** (MalformedURLException ex) {

System.***err***.println(ex.getMessage());

} **catch** (RemoteException ex) {

System.***out***.println("error");

System.***err***.println(ex.getMessage());

}

initialize();

sensorJTable();

refreshTable();

}

**public** List<SensorInfo> Sensorlist()

{

ArrayList<SensorInfo> sensors = **new** ArrayList<SensorInfo>();

**try** {

**return** service.getSensorInfo();

} **catch** (RemoteException e) {

e.printStackTrace();

}

**return** sensors;

}

**public** **void** sensorJTable() //adding data to the table

{

**if**(frame == **null**) **return**;

DefaultTableModel model = (DefaultTableModel)table.getModel();

List<SensorInfo> sensors = Sensorlist();

Object sensorData[] = **new** Object[6];

**for** (**int** counter = 0; counter < sensors.size(); counter++) {

sensorData[0] = sensors.get(counter).id;

sensorData[1] = sensors.get(counter).is\_active;

sensorData[2] = sensors.get(counter).floor\_no;

sensorData[3] = sensors.get(counter).room\_no;

sensorData[4] = sensors.get(counter).smoke\_level;

sensorData[5] = sensors.get(counter).co2\_level;

model.addRow(sensorData);

}

**for**(SensorInfo s : sensors) { //alert when the CO2 or smoke level passes 5

**if**(!*hasAlertedCO2*.containsKey(s.id)) {

*hasAlertedCO2*.put(s.id, **false**);

}

**if**(!*hasAlertedSmoke*.containsKey(s.id)) {

*hasAlertedSmoke*.put(s.id, **false**);

}

**if** (s.is\_active) {

// check for CO2 Level

**if** (s.co2\_level > 5 && !*hasAlertedCO2*.get(s.id)) {

String alertmsg = "CO2 level has moved to a value greater than 5 in room " + s.room\_no + " of floor " + s.floor\_no;

JOptionPane.*showMessageDialog*(**null**,alertmsg,"Alert for Sensor ID " + s.id, JOptionPane.***INFORMATION\_MESSAGE***);

// update the flag

*hasAlertedCO2*.put(s.id, **true**);

}**else** **if**(s.co2\_level <= 5) {

// reset the flag

*hasAlertedCO2*.put(s.id, **false**);

}

// Check for Smoke Level

**if** (s.smoke\_level > 5 && !*hasAlertedSmoke*.get(s.id)) {

String alertmsg = "Smoke level has moved to a value greater than 5 in room " + s.room\_no + " of floor " + s.floor\_no;

JOptionPane.*showMessageDialog*(**null**,alertmsg,"Alert for Sensor ID " + s.id, JOptionPane.***INFORMATION\_MESSAGE***);

// update the flag

*hasAlertedSmoke*.put(s.id, **true**);

}**else** **if**(s.smoke\_level <= 5) {

// reset the flag

*hasAlertedSmoke*.put(s.id, **false**);

}

}

}

}

**public** **void** refreshTable() { //Refreshes the table every 15 seconds

Timer timer;

timer = **new** Timer(15000, **new** ActionListener() {

@Override

**public** **void** actionPerformed(ActionEvent e) {

DefaultTableModel model = (DefaultTableModel)table.getModel();

model.setRowCount(0);

sensorJTable();

}

});

timer.start();

}

/\*\*

\* Initialize the contents of the frame.

\*/

**private** **void** initialize() {

//auto generated code removed

scrollPane.setOpaque( **false** );

scrollPane.getViewport().setOpaque( **false** ); //sets the non-content area of the table same as the background image

//cod related to table creation removed

TableCellRedColourRenderer renderer = **new** TableCellRedColourRenderer();

table.setDefaultRenderer(Object.**class**, renderer);

//code relating to button and labels removed

}

}

**Login.java**

**package** clientApplication;

**//code relating to imports removed**

**public** **class** Login **extends** JFrame {

**//code removed**

/\*\*

\* Launch the application.

\*/

**//code relating to the launching of application removed**

/\*\*

\* Create the frame.

\*/

**public** Login() {

//code relating to buttons, text fields,labels, keylisteners removed

**if** (e.getKeyCode()==KeyEvent.***VK\_ENTER***){

btnNewButton.doClick();

}

}

//auto generated code removed

});

**if** (e.getKeyCode()==KeyEvent.***VK\_ENTER***){

btnNewButton.doClick();

}

}

@Override

//auto generated code removed });

//code relating to action listener removed

String password = txtPassword.~~getText~~().trim();

String username = txtUsername.getText().trim();

**if** (password.isEmpty() && username.isEmpty()) { //validations are done for empty fields

lbl\_username.setText("Username field canno't be empty");

lbl\_pass.setText("Password field canno't be empty");

}

**else** **if**(username.isEmpty()) {

lbl\_username.setText("Username field canno't be empty");

}

**else** **if**(password.isEmpty()) {

lbl\_pass.setText("Password field canno't be kept empty");

}

**else** {

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

// System.setSecurityManager (new RMISecurityManager());

// }

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

**try** {

IServer service = (IServer) Naming.*lookup*("rmi://localhost/RMIServer");

**if**(service.login(username, password)) {

contentPane.setVisible(**false**);

dispose();

AdminPage admpage = **new** AdminPage();

admpage.setVisible(**true**); //closes the current frame and opens AdminPage frame

}**else** {

**return**;

}

} **catch** (NotBoundException ex) {

System.***err***.println(ex.getMessage());

} **catch** (MalformedURLException ex) {

System.***err***.println(ex.getMessage());

} **catch** (RemoteException ex) {

System.***err***.println(ex.getMessage());

}

}

}

});

//code relating to buttons and labels removed

}

}

**AdminPage.java**

**package** clientApplication;

//code relating to imports removed

**public** **class** AdminPage **extends** JFrame {

//code relating to initializing buttons, labels and text fields removed

/\*\*

\* Launch the application.

\*/

//code relating to launching the application removed

**private** HashMap<Integer, Boolean> hasAlertedCO2 = **new** HashMap<Integer, Boolean>();

**private** HashMap<Integer, Boolean> hasAlertedSmoke = **new** HashMap<Integer, Boolean>();

**private** IServer service = **null**;

**public** List<SensorInfo> Sensorlist()

{

ArrayList<SensorInfo> sensors = **new** ArrayList<SensorInfo>();

**try** {

**return** service.getSensorInfo();

} **catch** (RemoteException e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

}

**return** sensors;

}

**public** **void** sensorJTable() //adding data to the table

{

DefaultTableModel model = (DefaultTableModel)table.getModel();

List<SensorInfo> sensors = Sensorlist();

Object sensorData[] = **new** Object[6];

**for** (**int** counter = 0; counter < sensors.size(); counter++) {

sensorData[0] = sensors.get(counter).id;

sensorData[1] = sensors.get(counter).is\_active;

sensorData[2] = sensors.get(counter).floor\_no;

sensorData[3] = sensors.get(counter).room\_no;

sensorData[4] = sensors.get(counter).smoke\_level;

sensorData[5] = sensors.get(counter).co2\_level;

model.addRow(sensorData);

}

**for**(SensorInfo s : sensors) { //alert when the CO2 or smoke level passes 5

**if**(!hasAlertedCO2.containsKey(s.id)) {

hasAlertedCO2.put(s.id, **false**);

}

**if**(!hasAlertedSmoke.containsKey(s.id)) {

hasAlertedSmoke.put(s.id, **false**);

}

**if** (s.is\_active) {

// check for CO2 Level

**if** (s.co2\_level > 5 && !hasAlertedCO2.get(s.id)) {

String alertmsg = "CO2 level has moved to a value greater than 5 in room " + s.room\_no + " of floor " + s.floor\_no;

JOptionPane.*showMessageDialog*(**null**,alertmsg,"Alert for Sensor ID " + s.id, JOptionPane.***INFORMATION\_MESSAGE***);

// update the flag

hasAlertedCO2.put(s.id, **true**);

}**else** **if**(s.co2\_level <= 5) {

// reset the flag

hasAlertedCO2.put(s.id, **false**);

}

// Check for Smoke Level

**if** (s.smoke\_level > 5 && !hasAlertedSmoke.get(s.id)) {

String alertmsg = "Smoke level has moved to a value greater than 5 in room " + s.room\_no + " of floor " + s.floor\_no;

JOptionPane.*showMessageDialog*(**null**,alertmsg,"Alert for Sensor ID " + s.id, JOptionPane.***INFORMATION\_MESSAGE***);

// update the flag

hasAlertedSmoke.put(s.id, **true**);

}**else** **if**(s.smoke\_level <= 5) {

// reset the flag

hasAlertedSmoke.put(s.id, **false**);

}

}

}

}

**public** **void** refreshTable() { //refreshes the table every 15 seconds

Timer timer;

timer = **new** Timer(15000, **new** ActionListener() {

@Override

**public** **void** actionPerformed(ActionEvent e) {

DefaultTableModel model = (DefaultTableModel)table.getModel();

model.setRowCount(0);

sensorJTable();

}

});

timer.start();

}

/\*\*

\* Create the frame.

\*/

**public** AdminPage() {

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

**try** {

service = (IServer) Naming.*lookup*("rmi://localhost/RMIServer");

} **catch** (NotBoundException ex) {

System.***err***.println(ex.getMessage());

} **catch** (MalformedURLException ex) {

System.***err***.println(ex.getMessage());

} **catch** (RemoteException ex) {

System.***out***.println("Error");

System.***err***.println(ex.getMessage());

}

//code relating to creation of buttons, labels and table removed

JTableHeader Tableheader = table.getTableHeader();

Tableheader.setBackground(Color.***red***); // changes the Background color

Tableheader.setForeground(Color.***WHITE***); //changes the font color to white

TableCellRedColourRenderer renderer = **new** TableCellRedColourRenderer();

table.setDefaultRenderer(Object.**class**, renderer);

//code related to creating labels removed

//code relating to keyadapters removed

edID.setEditable(**false**);

lbl\_sensorID.setText("Please select a row in the table to edit floor no and room no");

}

});

//code relating to initializing text fields and buttons removed

//code relating to key adapters removed

edStat.setEditable(**false**);

lbl\_sensorstatus.setText("Please select a row in the table to edit floor no and room no");

}

});

//code removed

//code relating to key adapters removed

**if**(Character.*isLetter*(a)) { //ensures that only numbers are entered in the Floor Number field

edFloorNo.setEditable(**false**);

lbl\_floorno.setText("Only numbers are allowed in this field");

}

**else** {

edFloorNo.setEditable(**true**);

lbl\_floorno.setText("");

lbl\_sensorID.setText("");

lbl\_sensorstatus.setText("");

}

}

});

//code removed

//code relating to key adapters removed

**if**(Character.*isLetter*(a)) { //ensures that only numbers are entered in the Room Number field

edRoomNo.setEditable(**false**);

lbl\_roomno.setText("Only numbers are allowed in this field");

}

**else** {

edRoomNo.setEditable(**true**);

lbl\_roomno.setText("");

lbl\_sensorID.setText("");

lbl\_sensorstatus.setText("");

}

}

});

//code related to initializing buttons, lables and ActionListeners removed

**if**(edID.getText().trim().isEmpty() && edStat.getText().trim().isEmpty() && edFloorNo.getText().trim().isEmpty() && edRoomNo.getText().trim().isEmpty()) { //checks whether all the fields are empty

lbl\_sensorID.setText("Sensor ID field canno't be kept empty");

lbl\_sensorstatus.setText("Sensor status field canno't be kept empty");

lbl\_floorno.setText("Floor Number field canno't be kept empty");

lbl\_roomno.setText("Room Number field canno't be kept empty");

}

**else** **if**(edID.getText().trim().isEmpty()){ //checks whether the Sensor ID field is empty

lbl\_sensorID.setText("Sensor ID field canno't be kept empty");

}

**else** **if**(edStat.getText().trim().isEmpty()) { //checks whether the sensor status field is empty

lbl\_sensorstatus.setText("Sensor status field canno't be kept empty");

}

**else** **if**(edFloorNo.getText().trim().isEmpty()) { //checks whether the floor number field is empty

lbl\_floorno.setText("Floor Number field canno't be kept empty");

}

**else** **if**(edRoomNo.getText().trim().isEmpty()) {

lbl\_roomno.setText("Room Number field canno't be kept empty"); //checks whether the room number field is empty

}

**else** {

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

**try** {

IServer service = (IServer) Naming.*lookup*("rmi://localhost/RMIServer");

**int** id = Integer.*parseInt*(edID.getText().trim());

**int** room\_no = Integer.*parseInt*(edRoomNo.getText().trim());

**int** floor\_no = Integer.*parseInt*(edFloorNo.getText().trim());

SensorInfo updatedSensorInfo = **new** SensorInfo(id, 0, 0, room\_no, floor\_no, **true**, "", "");

service.updateSensor(id, updatedSensorInfo);

JOptionPane.*showMessageDialog*(**null**,"Saved Successfully");

} **catch** (NotBoundException ex) {

System.***err***.println(ex.getMessage());

} **catch** (MalformedURLException ex) {

System.***err***.println(ex.getMessage());

} **catch** (RemoteException ex) {

System.***err***.println(ex.getMessage());

}

}

}

});

//code relating to initializng buttons, labels and action listeners removed

**if**(edID.getText().trim().isEmpty() && edStat.getText().trim().isEmpty() && edFloorNo.getText().trim().isEmpty() && edRoomNo.getText().trim().isEmpty()) { //checks whether all the fields are empty

lbl\_sensorID.setText("Sensor ID field canno't be kept empty");

lbl\_sensorstatus.setText("Sensor status field canno't be kept empty");

lbl\_floorno.setText("Floor Number field canno't be kept empty");

lbl\_roomno.setText("Room Number field canno't be kept empty");

}

**else** **if**(edID.getText().trim().isEmpty()){ //checks whether the Sensor ID field is empty

lbl\_sensorID.setText("Sensor ID field canno't be kept empty");

}

**else** **if**(edStat.getText().trim().isEmpty()) { //checks whether the sensor status field is empty

lbl\_sensorstatus.setText("Sensor status field canno't be kept empty");

}

**else** **if**(edFloorNo.getText().trim().isEmpty()) { //checks whether the floor number field is empty

lbl\_floorno.setText("Floor Number field canno't be kept empty");

}

**else** **if**(edRoomNo.getText().trim().isEmpty()) {

lbl\_roomno.setText("Room Number field canno't be kept empty"); //checks whether the room number field is empty

}

**else**

{

**int** id = Integer.*parseInt*(edID.getText().trim());

**if**(id > 0) {

**try** {

service.deleteSensor(id);

JOptionPane.*showMessageDialog*(**null**,"Deleted Successfully");

// to update the table immediately

DefaultTableModel model = (DefaultTableModel)table.getModel();

model.setRowCount(0);

sensorJTable();

} **catch** (RemoteException e1) {

// **TODO** Auto-generated catch block

e1.printStackTrace();

JOptionPane.*showMessageDialog*(**null**,"Error when deleting the sensor...");

}

}

}

}

});

//code relating to buttons and labels removed

sensorJTable();

refreshTable();

}

}

**AdminAddSensor.java**

**package** clientApplication;

//code relating to imports removed

**public** **class** AdminAddSensor **extends** JFrame {

//code relating to initializing text fields an labels removed

/\*\*

\* Launch the application.

\*/

//code relating to launching the application removed

/\*\*

\* Create the frame.

\*/

**public** AdminAddSensor() {

//code relating to initializing contentpanes, labels, textfields removed

//code relating to initializing buttons, text fields, labels and action listeners removed

dispose();

AdminPage admpage = **new** AdminPage();

admpage.setVisible(**true**);

}

});

//code relating to buttond and key adapters removed

**if**(Character.*isLetter*(a)) { //ensures that only numbers are entered in the Floor Number field

floornm.setEditable(**false**);

lbl\_floornum.setText("Only numbers are allowed in this field");

}

**else** {

floornm.setEditable(**true**);

lbl\_floornum.setText("");

}

}

});

**if**(Character.*isLetter*(a)) { //ensures that only numbers are entered in the Room Number field

roomnm.setEditable(**false**);

lbl\_roomnum.setText("Only numbers are allowed in this field");

}

**else** {

roomnm.setEditable(**true**);

lbl\_roomnum.setText("");

}

}

});

//code relating to action listener of button removed

**if**(floornm.getText().trim().isEmpty() && roomnm.getText().trim().isEmpty()) { //checks whether all the fields are empty

lbl\_floornum.setText("Floor Number field canno't be kept empty");

lbl\_roomnum.setText("Room Number field canno't be kept empty");

}

**else** **if**(floornm.getText().trim().isEmpty()) { //checks whether the floor number field is empty

lbl\_floornum.setText("Floor Number field canno't be kept empty");

}

**else** **if**(roomnm.getText().trim().isEmpty()) { //checks whether the room number field is empty

lbl\_roomnum.setText("Room Number field canno't be kept empty");

}

**else** {

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

**try** {

IServer service = (IServer) Naming.*lookup*("rmi://localhost/RMIServer");

**int** id = 0;

**int** room\_no = Integer.*parseInt*(roomnm.getText().trim());

**int** floor\_no = Integer.*parseInt*(floornm.getText().trim());

SensorInfo newSensor = **new** SensorInfo(id, 0, 0, room\_no, floor\_no, **true**, "", "");

service.addNewSensor(newSensor);

JOptionPane.*showMessageDialog*(**null**,"Sensor added successfully");

} **catch** (NotBoundException ex) {

System.***err***.println(ex.getMessage());

} **catch** (MalformedURLException ex) {

System.***err***.println(ex.getMessage());

} **catch** (RemoteException ex) {

System.***err***.println(ex.getMessage());

}

}

}

});

}

}

**TableCellRedColourRenderer.java**

**package** clientApplication;

**//code related to imports removed**

**public** **class** TableCellRedColourRenderer **implements** TableCellRenderer{

**//auto generated method removed**

**if**(column == 5 || column == 4) { //checks for CO2 level, Smoke level columns

Object result = table.getModel().getValueAt(row, column); //gets the value in the particular cell

**int** resultvalue = Integer.*parseInt*(result.toString());

Color color = **null**;

**if**(resultvalue > 5) {

color = Color.***RED***;

}

comp.setForeground(color); //if the value is greater than 5, set the font color to red

} **else** {

comp.setForeground(Color.***BLACK***);

}

}

}

**Web client**

Header

import React from "react";

const DangerAndNormal = () => {

const headerStyle = {

display: "flex",

flexDirection: "column",

alignItems: "center",

justifyContent: "center",

};

return (

<h3 style={headerStyle}>

If Sensor Normal <span className="badge badge-success"> GREEN</span>

<br />

If Sensor Danger <span className="badge badge-danger"> RED </span>

</h3>

);

};

export default DangerAndNormal;

import React from "react";

const Header = () => {

const headerStyle = {

display: "flex",

flexDirection: "column",

alignItems: "center",

justifyContent: "center",

};

return (

<h1 style={headerStyle} className="display-4">

Fire Alarm App

</h1>

);

};

export default Header;

Sensor Component

import React from "react";

const Sensor = (props) => {

const sensorDetails = props.sensorDetails;

let sensorState;

if (

!sensorDetails.is\_active ||

sensorDetails.co2\_level >= 5 ||

sensorDetails.smoke\_level >= 5

) {

sensorState = "danger";

} else {

sensorState = "success";

}

const boarderClassColor = `card border-${sensorState} mb-3`;

const badgeStateColor = ` badge badge-${sensorState}`;

return (

<div className={boarderClassColor} style={{ maxWidth: "15rem" }}>

<div className="card-header container">

<div className="row">

<div className="col">Sensor {sensorDetails.id}</div>

<div className="col">

<span className={badgeStateColor}>

{sensorDetails.is\_active ? "active" : "Inactive"}

</span>

</div>

</div>

</div>

<div className="card-body text-primary">

<div className="container">

<div className="row">

<div className="col">

<h5 className="card-title">Location </h5>

</div>

<div className="col">

<span className={badgeStateColor}>

floor: {sensorDetails.floor\_no}{" "}

</span>

<span style={{ margin: "5px" }} className={badgeStateColor}>

room: {sensorDetails.room\_no}

</span>

</div>

</div>

<div className="row">

<div className="col">

<h5 className="card-title">CO2 Level </h5>

</div>

<div className="col">

<span style={{ padding: "10px" }} className={badgeStateColor}>

{sensorDetails.co2\_level}

</span>

</div>

</div>

<div className="row">

<div className="col">

<h5 className="card-title">Smoke Level </h5>

</div>

<div className="col">

<span style={{ padding: "10px" }} className={badgeStateColor}>

{sensorDetails.smoke\_level}

</span>

</div>

</div>

</div>

</div>

</div>

);

};

export default Sensor;

Sensor List component

import React, { Component } from "react";

import Sensor from "./sensor";

class SensorList extends Component {

state = {

sensors: [],

};

fetchingData = () => {

let fetchedData;

fetch(" https://fire-alarm-api-ds.herokuapp.com/api/sensorinfo")

.then((res) => res.json())

.then((data) => {

fetchedData = data;

this.setState({

sensors: fetchedData,

});

// sorting the sensors

this.setState(this.state.sensors.sort((s1, s2) => s1.id - s2.id));

});

};

componentDidMount() {

this.fetchingData();

//making the page refresh every 40 seconds

setInterval(this.fetchingData, 40000);

}

render() {

const sensorComponents = this.state.sensors.map((sensor) => (

<Sensor sensorDetails={sensor} />

));

return (

<div className="grid-container">

{sensorComponents.map((comp) => (

<div key={comp.props.sensorDetails.id}>{comp}</div>

))}

</div>

);

}

}

export default SensorList;