Sri Lanka Institute of Information Technology

Assignment 2
REST API

Distributed Systems – SE3020
2020

IT18110876– Ariyathilake G.P.P.

Table of Contents

Introduction

This Report describes each and every component in the Fire Alarm System and how they connect with each other to process the sensor data gathered by Fire Alarm Sensors. In the First Section of the report it describes what are the components in the system and how are they used in the System.
First in this section it is described how the data is stored in the database. Then the RESTful web services API is described using the Service interface it provides to perform actions relevant to users and fire alarm sensor data. When describing the REST API, it is also mentioned what are the data that each of the method need to make a request and also what can be the response from them. This task is achieved using a YAML to PDF converter. And also in each REST API interface it also shows how the system process the HTTP request using sequence diagrams. Then in the next part it describes how the SMS service and the Email service is implemented. And how to configure them and use them in the system. The next component described is the fire alarm sensor. This describes how the fire alarm sensor sends data to the REST API and also how the Email and the SMS service works when a Fire Alarm sensor exceeds its CO2 or Smoke limit. RMI Server and the Client is described in the last of the first section. It is described using Class Diagrams and Sequence diagrams.
In the next section is a Component Diagram of the system. It displays how the above-mentioned components are connected and working as one whole system.
Then in the last section of the Report describes the authentication mechanisms used in the system to implement user login and registration.

The System Architecture

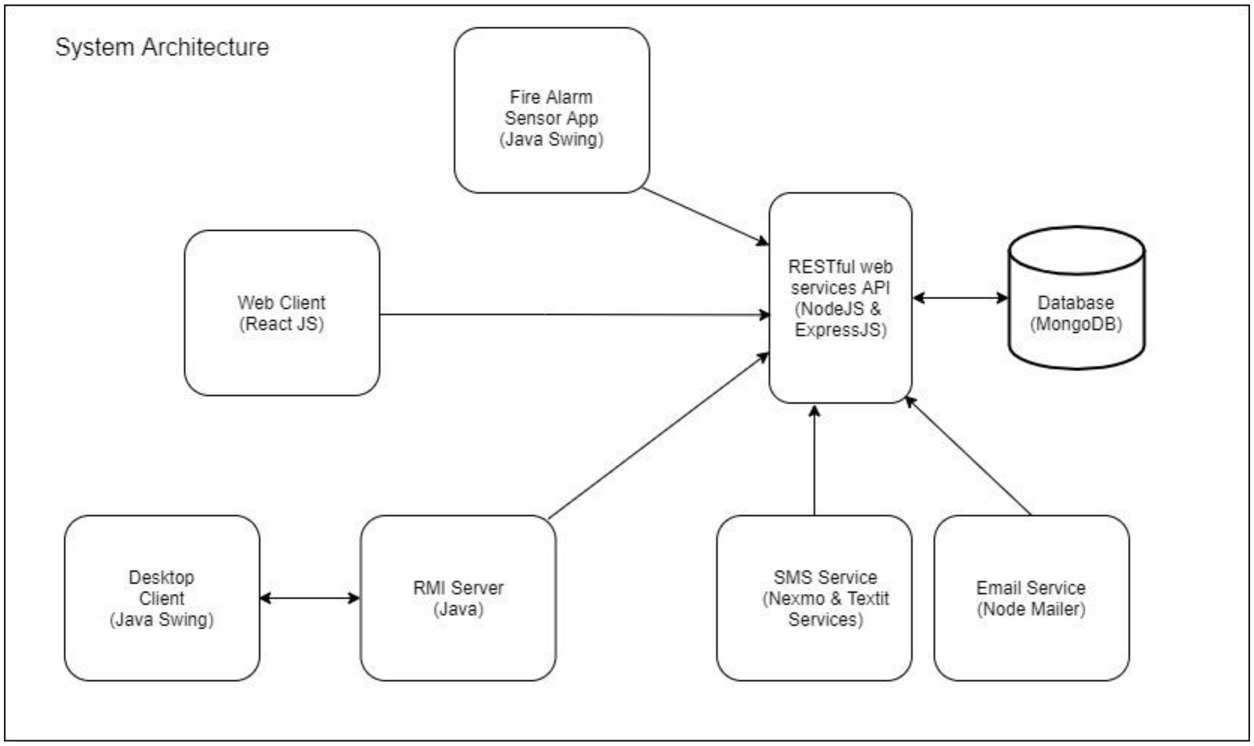

System Architecture
Fire Alarm Sensor App (Java Swing)
Web Client (React JS)
RESTful web services API (NodeJS & ExpressJS)
Database (MongoDB)
Desktop Client (Java Swing)
RMI Server (Java)
SMS Service (Nexmo & Textit Services)
Email Service (Node Mailer)

Figure 1 The System Architecture

System is developed using the above System Architecture. The System has main 7 Components as follows,

1. Database
2. RESTful Web Services API
3. SMS Service
4. Email Service
5. Fire Alarm Sensor
6. Web Client
7. RMI Server & Desktop Client

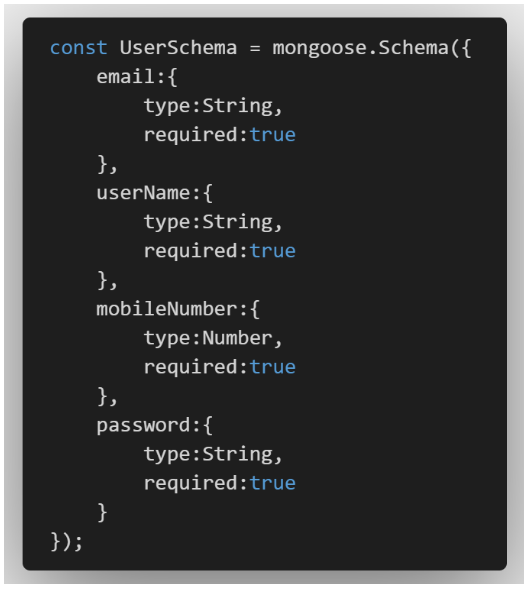
Every Other Component is connected to the RESTful web services API. The RESTful web services API provides the required functions to the other components using interfaces.

Database

The database is developed using MongoDB. MongoDB is a NoSQL database. In the Fire Alarm System, the database stores 2 types of data collections.

- User Details Collection
- Fire Alarm Sensor Details Collection

The User Details are stored in the database to introduce the Administrator and Guest user privileges to the system. Figure 2 shows the MongoDB schema used to add a new user to the system.

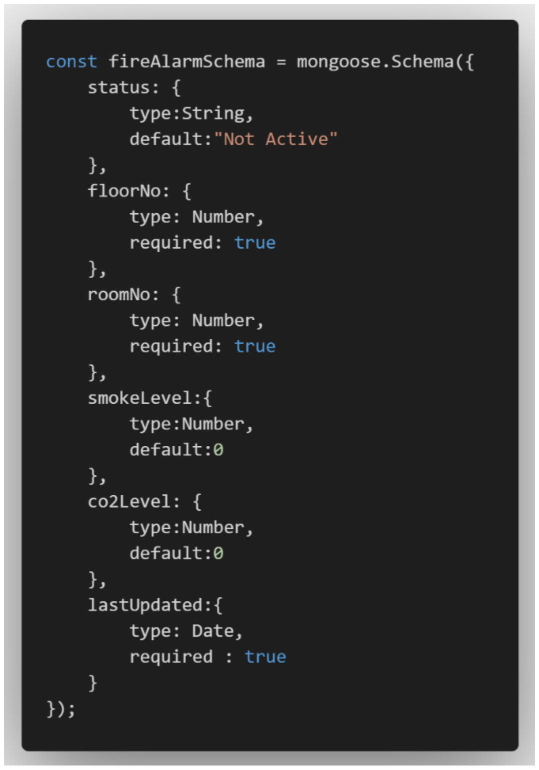


```
const UserSchema = mongoose.Schema({
    email:{
        type:String,
        required:true
    },
    userName:{
        type:String,
        required:true
    },
    mobileNumber:{
        type:Number,
        required:true
    },
    password:{
        type:String,
        required:true
    }
});
```

Figure 2: User MongoDB Schema

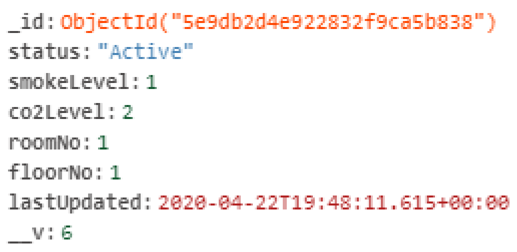
As shown in the Figure 2 to add a new data document to the Database all the 4 data fields should be provided. The User Name and the Password is required to Login to the System. And if need the Email address can also be used to login to the system. The Mobile Number and the Email address is required to register a user because the system has to send an Email Notification and a SMS Notification when a Fire Alarm Sensor exceeds its CO2 Level or Smoke Level.

The Fire Alarm Sensor Details is the Main Collection of the database. It stores all the important data per a Fire Alarm Sensor. Figure 3 display the Mongo Schemas used to Create collections. When a Fire Alarm Sensor document is first created only the Room Number and the Floor Number is required to enter manually. The last updated time must be set by the system that enters the data to the database automatically. By default, when a fire alarm is added the status of the fire alarm is "Not Active". Which means that the fire alarm must be activated.
When updating Fire Alarm Sensor data, the version key (__v) must also be updated. The Version key (__v) in MongoDB is used to mitigate the possibility of race conditions in an array. But in here it can also be used to keep track of the number of revisions to a document. To do that when updating a document in MongoDB the version key must be manually incremented by one. And when updating a document then a user can use the version key to check whether the document is already updated by another user.



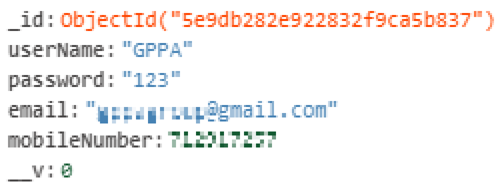
```
const fireAlarmSchema = mongoose.Schema({
    status: {
        type:String,
        default:"Not Active"
    },
    floorNo: {
        type: Number,
        required: true
    },
    roomNo: {
        type: Number,
        required: true
    },
    smokeLevel:{
        type:Number,
        default:0
    },
    co2Level: {
        type:Number,
        default:0
    },
    lastUpdated:{
        type: Date,
        required : true
    }
});
```

Figure 3: Fire Alarm Sensor MongoDB Schema



```
_id: ObjectId("5e9db2d4e922832f9ca5b838")
status: "Active"
smokeLevel: 1
co2Level: 2
roomNo: 1
floorNo: 1
lastUpdated: 2020-04-22T19:48:11.615+00:00
__v: 6
```

Figure 4: Fire Alarm Sensor Data in the Database



```
_id: ObjectId("5e9db282e922832f9ca5b837")
userName: "GPPA"
password: "123"
email: "[redacted]@gmail.com"
mobileNumber: [redacted]
__v: 0
```

Figure 5: User Data in the Database

RESTful Web Services API

The Swagger documentation related to this API and the Markdown file are located in the GitHub project branch under “REST API server” directory. In the markdown file it is also explained how to install and test the REST API.

The REST API exposes 2 types of interfaces in this system.

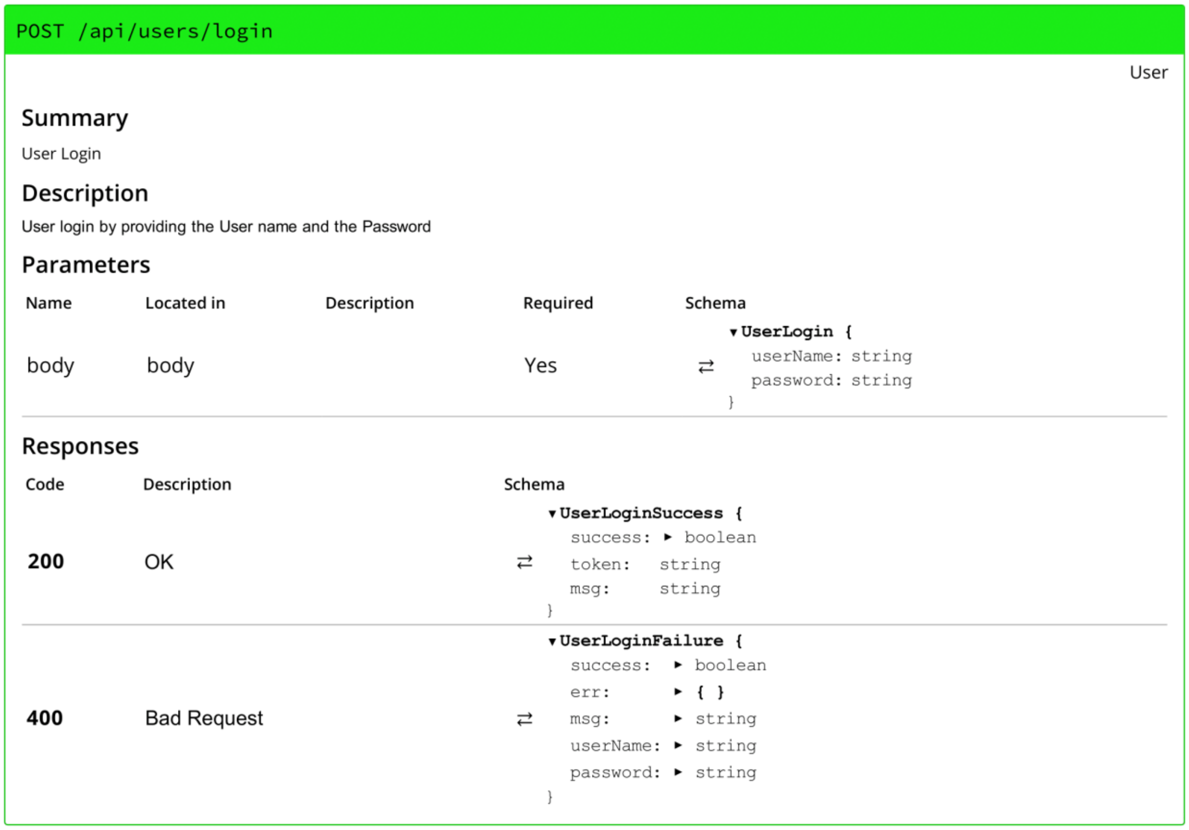
1. Users
2. Fire Alarms

Users

Users type interfaces are to perform the operations that are relevant to users. Such as,

- Logging using user credentials
- Registering a user account

1. User Login



POST /api/users/login

User

Summary

User Login

Description

User login by providing the User name and the Password

Parameters

Name	Located in	Description	Required	Schema
body	body		Yes	▾UserLogin { userName: string password: string }

Responses

Code	Description	Schema
200	OK	▾UserLoginSuccess { success: ▸ boolean token: string msg: string }
400	Bad Request	▾UserLoginFailure { success: ▸ boolean err: ▸ { } msg: ▸ string userName: ▸ string password: ▸ string }

Figure 6 : Login REST API Interface

As shown in Figure 6 the Login interface requires content-type JSON data that includes User Name and the Password. This User Name data can be user name or email. But the data field name is user name. If the Request is success the server returns a 200 HTTP status code and the authentication token for the user. If the Request is unsuccessful the Server returns a 400 HTTP status code and an error message.

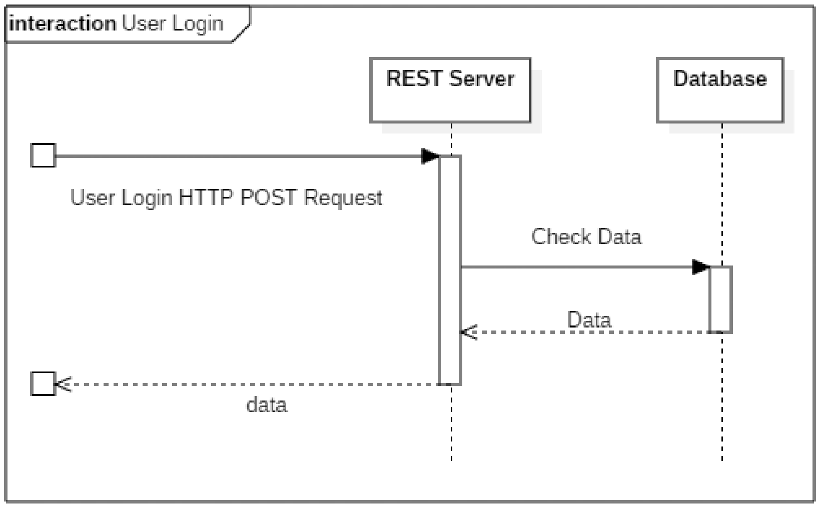
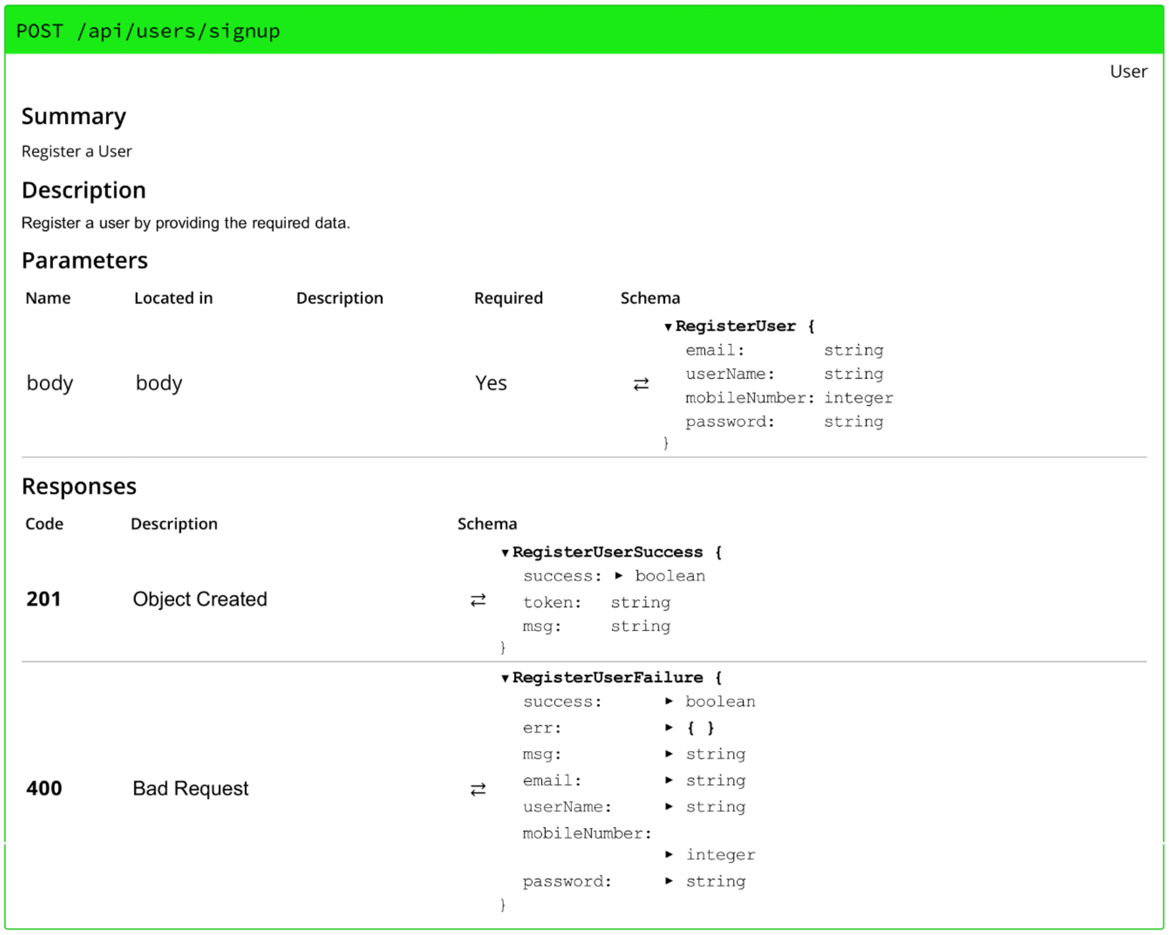


Figure 7: User Login REST Interface Sequence Diagram

2. User Registration



POST /api/users/signup

User

Summary

Register a User

Description

Register a user by providing the required data.

Parameters

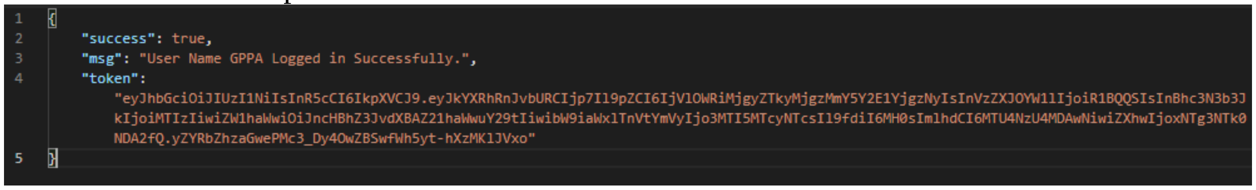
Name	Located in	Description	Required	Schema
body	body		Yes	⇄ ▾RegisterUser { email: string userName: string mobileNumber: integer password: string }

Responses

Code	Description	Schema
201	Object Created	⇄ ▾RegisterUserSuccess { success: ▸ boolean token: string msg: string }
400	Bad Request	⇄ ▾RegisterUserFailure { success: ▸ boolean err: ▸ { } msg: ▸ string email: ▸ string userName: ▸ string mobileNumber: ▸ integer password: ▸ string }

Figure 8: Register REST API Interface

As shown in Figure 8 to use this interface the user should provide a JSON type data that contains the user name, password, email and the mobile number. If one of the data fields are invalid or empty the server responds a 400 HTTP status code with failure message. If the Registration is success the Server responds a 201 HTTP status code with the authentication token.



```
{
    "success": true,
    "msg": "User Name GPPA Logged in Successfully.",
    "token":
        "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJkYXRhRnJvbURCIjp7Il9pZCI6IjV1OWRiMjgyZTkyMjgzMmY5Y2E1YjgzNyIsInVzZXJOYW1lIjoiR1BQQSIsInBhc3N3b3J
        kIjoiMTIzIiwiZW1haWwiOiJncHBhZ3JvdXBAZ21haWwuY29tIiwibW9iaWxlTnVtYmVyIjo3MTI5MTcyNTcsIl9fdiI6MH0sImlhdCI6MTU4NzU4MDAwNiwiZXhwIjoxNTg3NTk0
        NDA2fQ.yZYRbZhzaGwePMc3_Dy4OwZBSwfWh5yt-hXzMK1JVxo"
}
```

Figure 9: Sample HTTP Response body of a login

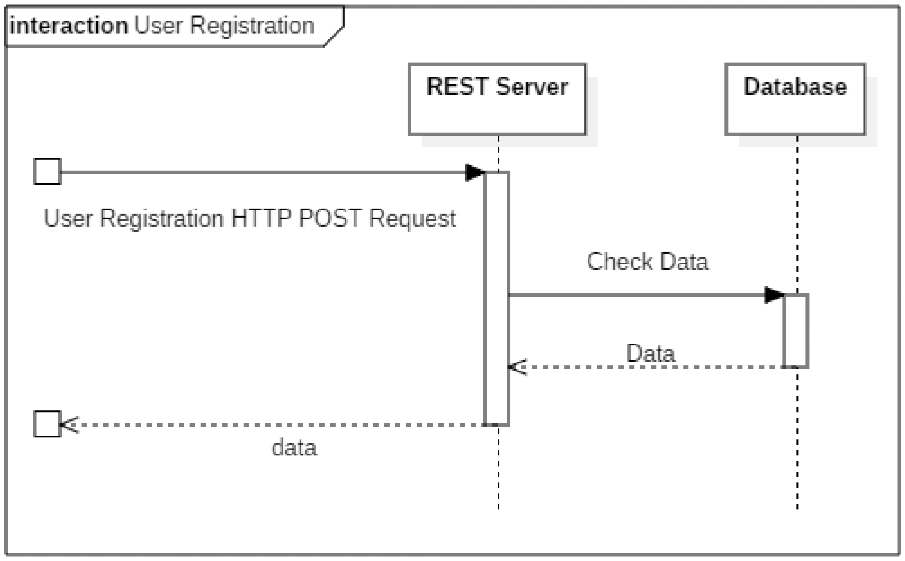

interaction User Registration
REST Server
Database
User Registration HTTP POST Request
Check Data
Data
data

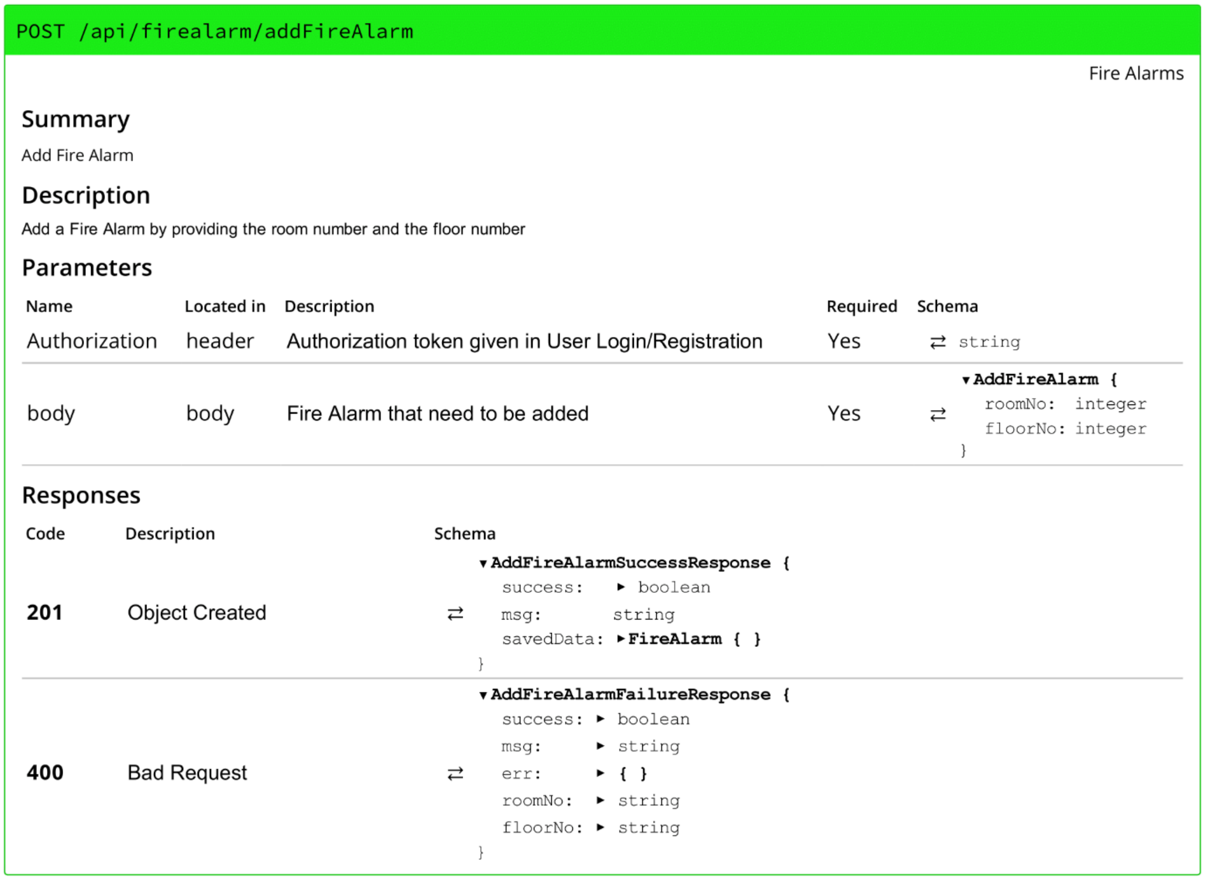
Figure 10: User Registration REST Interface Sequence Diagram

Fire Alarm Sensors

Fire Alarm type interfaces takes a major role in the system. There are 5 interfaces provided by the REST API in order to manage Fire Alarm Sensors.

1. Add Fire Alarm Sensors
2. Edit Fire Alarm Sensors
3. Delete Fire Alarm Sensors
4. Update Sensor Data of Fire Alarm Sensors
5. Get Sensor Data of Fire Alarm Sensors

1. Add Fire Alarm Sensors



```
POST /api/firealarm/addFireAlarm
```

Fire Alarms

Summary

Add Fire Alarm

Description

Add a Fire Alarm by providing the room number and the floor number

Parameters

Name	Located in	Description	Required	Schema
Authorization	header	Authorization token given in User Login/Registration	Yes	⇄ string
body	body	Fire Alarm that need to be added	Yes	⇄ ▾AddFireAlarm { roomNo: integer floorNo: integer }

Responses

Code	Description	Schema
201	Object Created	⇄ ▾AddFireAlarmSuccessResponse { success: ▸ boolean msg: string savedData: ▸FireAlarm { } }
400	Bad Request	⇄ ▾AddFireAlarmFailureResponse { success: ▸ boolean msg: ▸ string err: ▸ { } roomNo: ▸ string floorNo: ▸ string }

Figure 11: Add Fire Alarm Sensor Interface

As shown in Figure 11, when adding a new fire alarm sensor, the user must provide the Authentication and also the room number and the floor number. If one of these details are missing or invalid the server responds with a 400 HTTP status code. If the request is valid and the data gets inserted to the database and responds with a 201 HTTP status code.

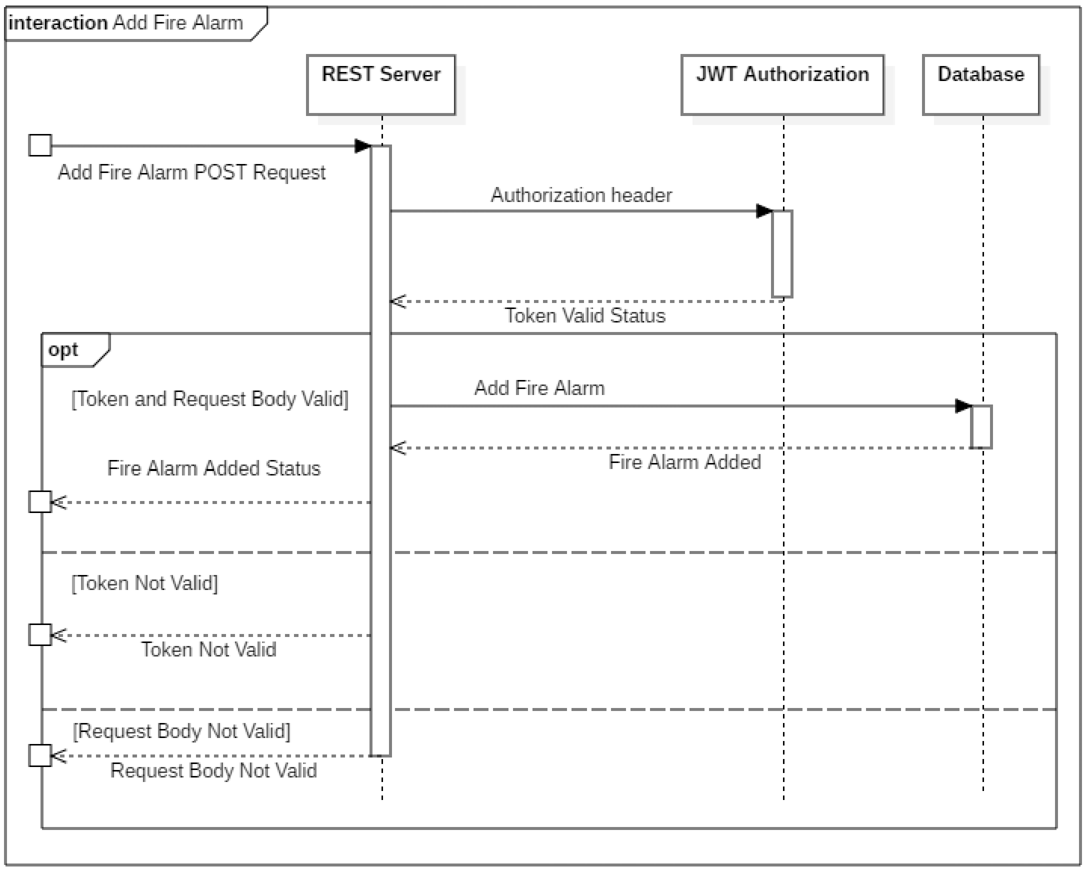
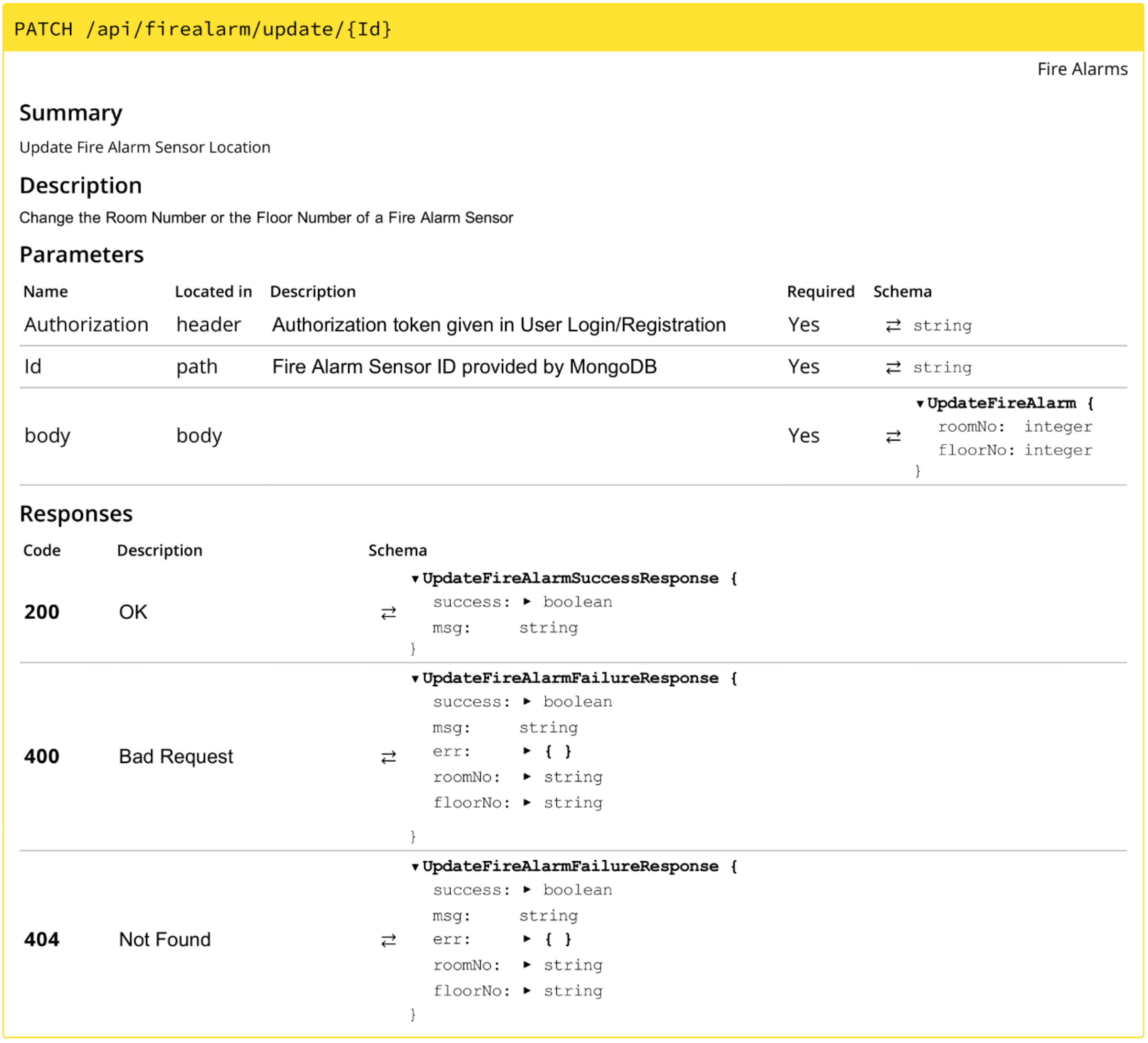


Figure 12: Add Fire Alarm REST Interface Sequence Diagram

2. Edit Fire Alarm Sensors



PATCH /api/firealarm/update/{Id}

Fire Alarms

Summary

Update Fire Alarm Sensor Location

Description

Change the Room Number or the Floor Number of a Fire Alarm Sensor

Parameters

Name	Located in	Description	Required	Schema
Authorization	header	Authorization token given in User Login/Registration	Yes	⇄ string
Id	path	Fire Alarm Sensor ID provided by MongoDB	Yes	⇄ string
body	body		Yes	⇄ ▾UpdateFireAlarm { roomNo: integer floorNo: integer }

Responses

Code	Description	Schema
200	OK	⇄ ▾UpdateFireAlarmSuccessResponse { success: ▸ boolean msg: string }
400	Bad Request	⇄ ▾UpdateFireAlarmFailureResponse { success: ▸ boolean msg: string err: ▸ { } roomNo: ▸ string floorNo: ▸ string }
404	Not Found	⇄ ▾UpdateFireAlarmFailureResponse { success: ▸ boolean msg: string err: ▸ { } roomNo: ▸ string floorNo: ▸ string }

Figure 13: Fire Alarm Sensor Edit interface

The edit Fire alarm HTTP request method is PATCH. The major difference PUT and PATCH HTTP method is that PUT requests need the whole document where the data need to be modified. But in PATCH requests it only requires the relevant part that is going to be updated. Hence PATCH requests save network resources.
To edit a fire alarm sensor the user needs to provide 3 key information. The id of the fire alarm sensor, authentication token and the modified data. If one or more of this information are missing the server responds a 400 HTTP status code. If the fire alarm id provided does not match a fire alarm existing in the database the server responds with a 404 HTTP status code. If the operation is successfully completed the server responds with a 200 HTTP status code and a success message.

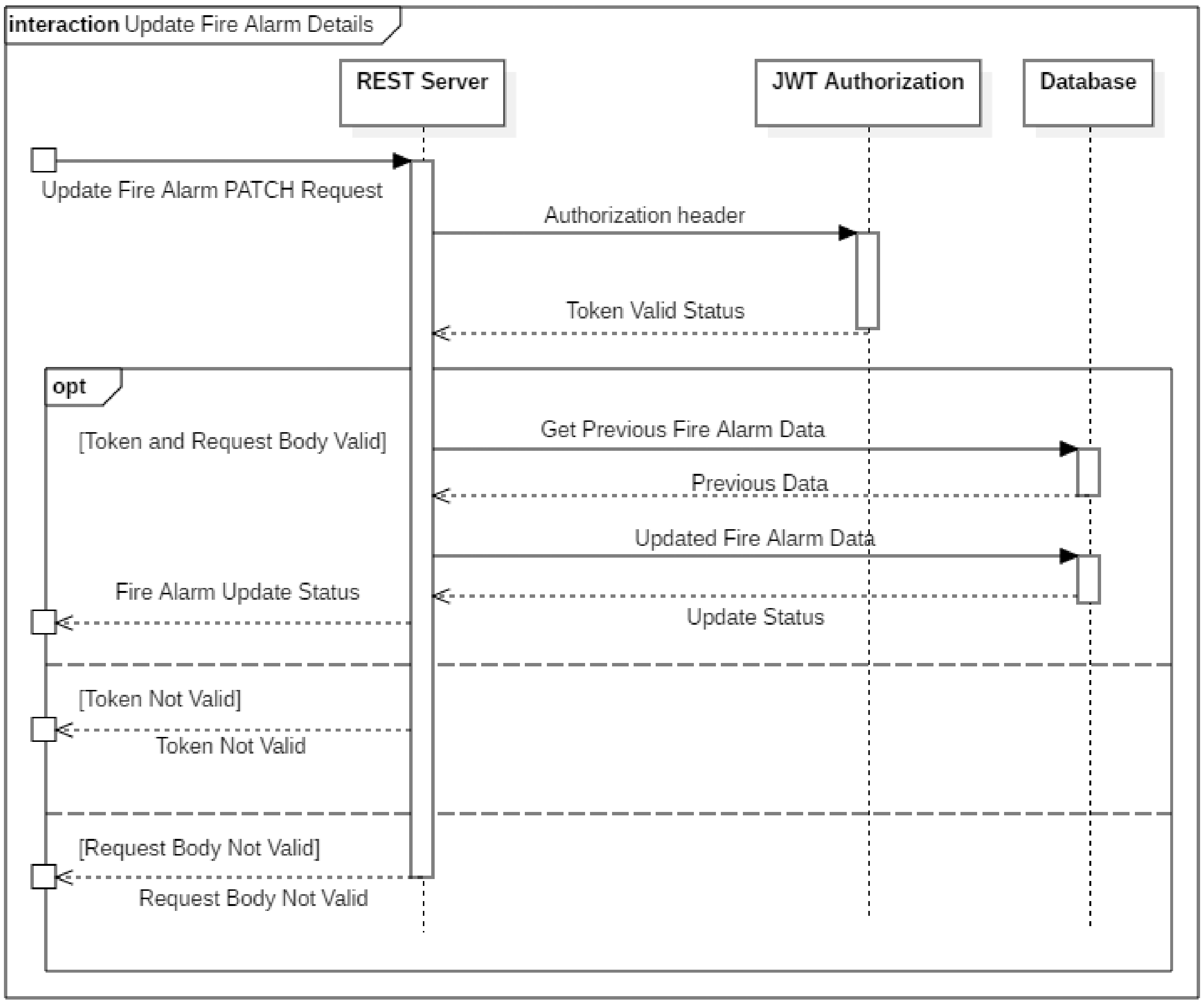
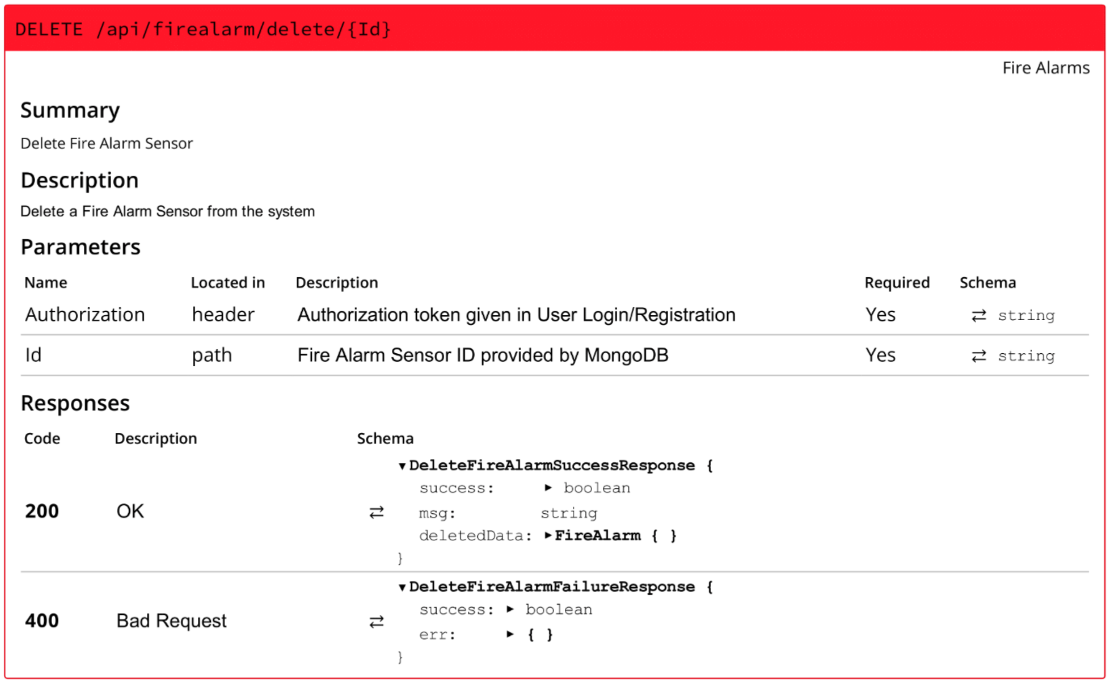


Figure 14: Edit Fire Alarm Sensor Details REST Interface Sequence Diagram

3. Delete Fire Alarm Sensor



DELETE /api/firealarm/delete/{Id}

Fire Alarms

Summary

Delete Fire Alarm Sensor

Description

Delete a Fire Alarm Sensor from the system

Parameters

Name	Located in	Description	Required	Schema
Authorization	header	Authorization token given in User Login/Registration	Yes	⇄ string
Id	path	Fire Alarm Sensor ID provided by MongoDB	Yes	⇄ string

Responses

Code	Description	Schema
200	OK	⇄ ▾**DeleteFireAlarmSuccessResponse** { success: ▸ boolean msg: string deletedData: ▸**FireAlarm** { } }
400	Bad Request	⇄ ▾**DeleteFireAlarmFailureResponse** { success: ▸ boolean err: ▸ { } }

Figure 15: Delete Fire Alarm Sensor Interface

To delete a fire alarm sensor from the database the user needs to provide 2 key information. Fire alarm sensor ID and the authentication token. If there was an error the system sends a 400 HTTP status code and an error message. If the fire alarm sensor document is successfully deleted or if the provided fire alarm sensor id does not exist in the database the system responds with a 200 HTTP status code. The servers send the response as OK even when the id is not existing is because the idempotent property of the DELETE HTTP method.

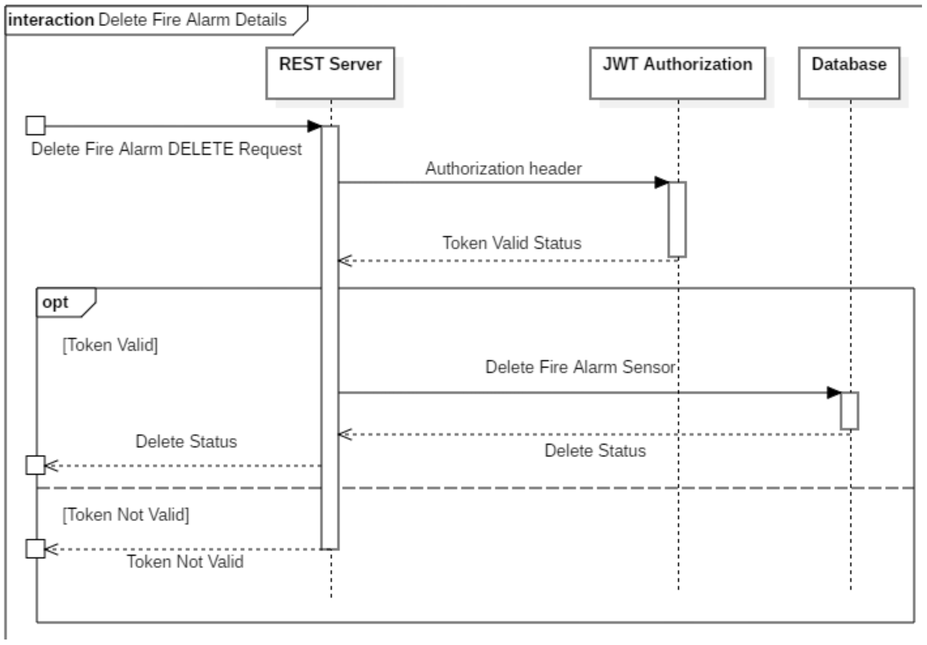
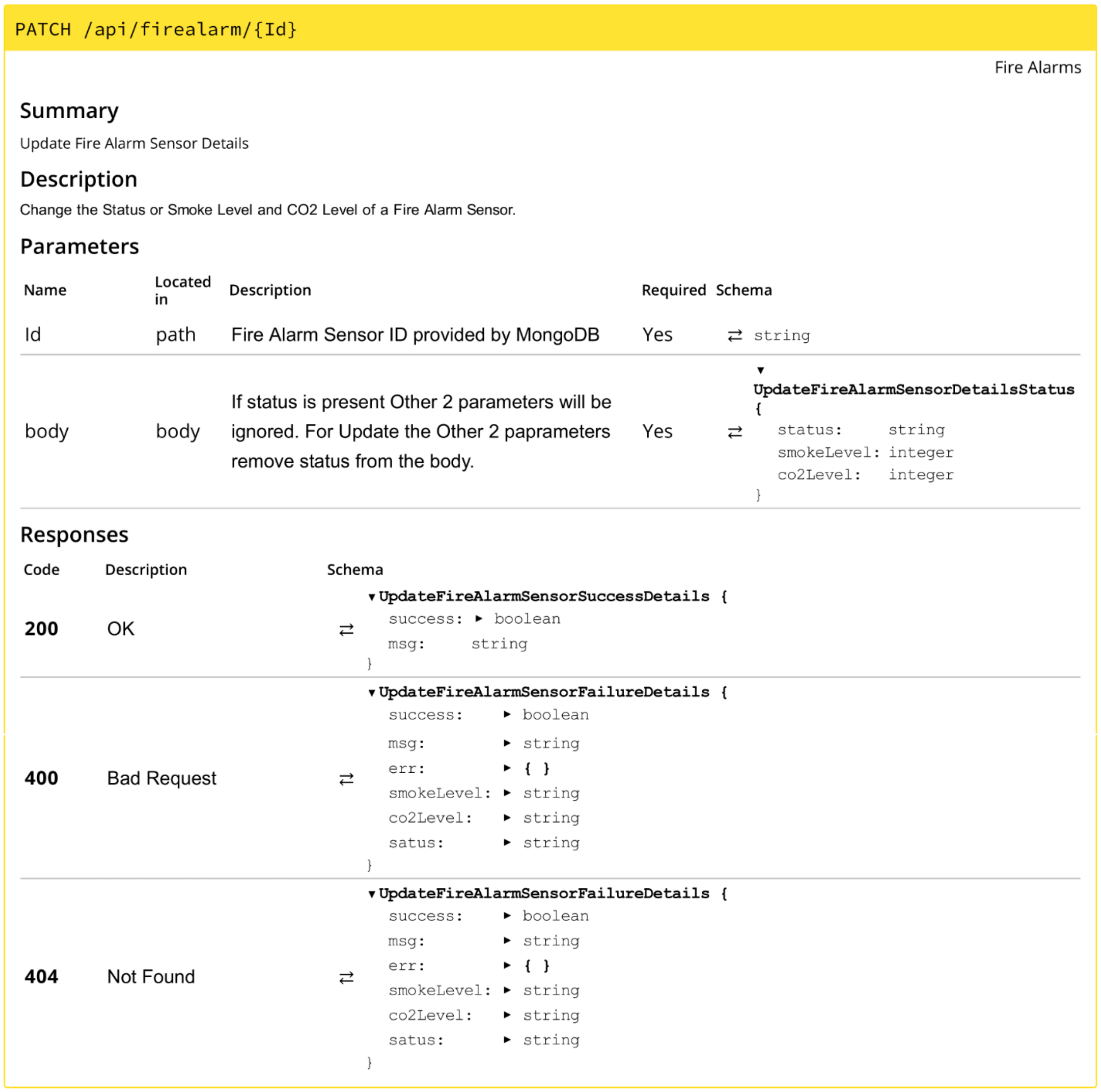


Figure 16: Delete Fire Alarm REST Interface Sequence Diagram

4. Update Sensor Data of Fire Alarm Sensors



PATCH /api/firealarm/{Id}

Fire Alarms

Summary

Update Fire Alarm Sensor Details

Description

Change the Status or Smoke Level and CO2 Level of a Fire Alarm Sensor.

Parameters

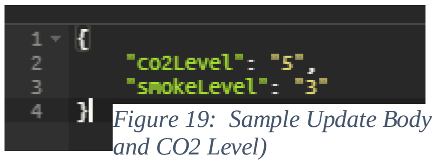
Name	Located in	Description	Required	Schema
Id	path	Fire Alarm Sensor ID provided by MongoDB	Yes	⇄ string
body	body	If status is present Other 2 parameters will be ignored. For Update the Other 2 paprameters remove status from the body.	Yes	⇄ ▾ UpdateFireAlarmSensorDetailsStatus { status: string, smokeLevel: integer, co2Level: integer }

Responses

Code	Description	Schema
200	OK	⇄ ▾ UpdateFireAlarmSensorSuccessDetails { success: ▸ boolean, msg: string }
400	Bad Request	⇄ ▾ UpdateFireAlarmSensorFailureDetails { success: ▸ boolean, msg: ▸ string, err: ▸ { }, smokeLevel: ▸ string, co2Level: ▸ string, satus: ▸ string }
404	Not Found	⇄ ▾ UpdateFireAlarmSensorFailureDetails { success: ▸ boolean, msg: ▸ string, err: ▸ { }, smokeLevel: ▸ string, co2Level: ▸ string, satus: ▸ string }

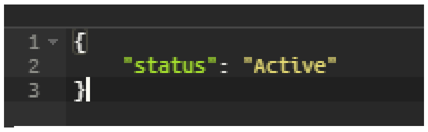
Figure 17: Update Sensor Data of Fire Alarm Sensor Interface

This Interface is used by the Fire Alarm Sensors to update the CO2 Level and Smoke Level and also the Status of the Fire Alarm Sensors. If a Fire Alarm Sensor status is needed to be updated status type must be provided in the body of the HTTP request. If the CO2 level or the Smoke Level is needed to be updated the status type should not be included in the body. The Smoke level and the CO2 level must be between 0 and 10.



```
{
    "co2Level": "5",
    "smokeLevel": "3"
}
```

Figure 19: Sample Update Body (Smoke and CO2 Level)



```
{
    "status": "Active"
}
```

Figure 18: Sample Status Update Request Body

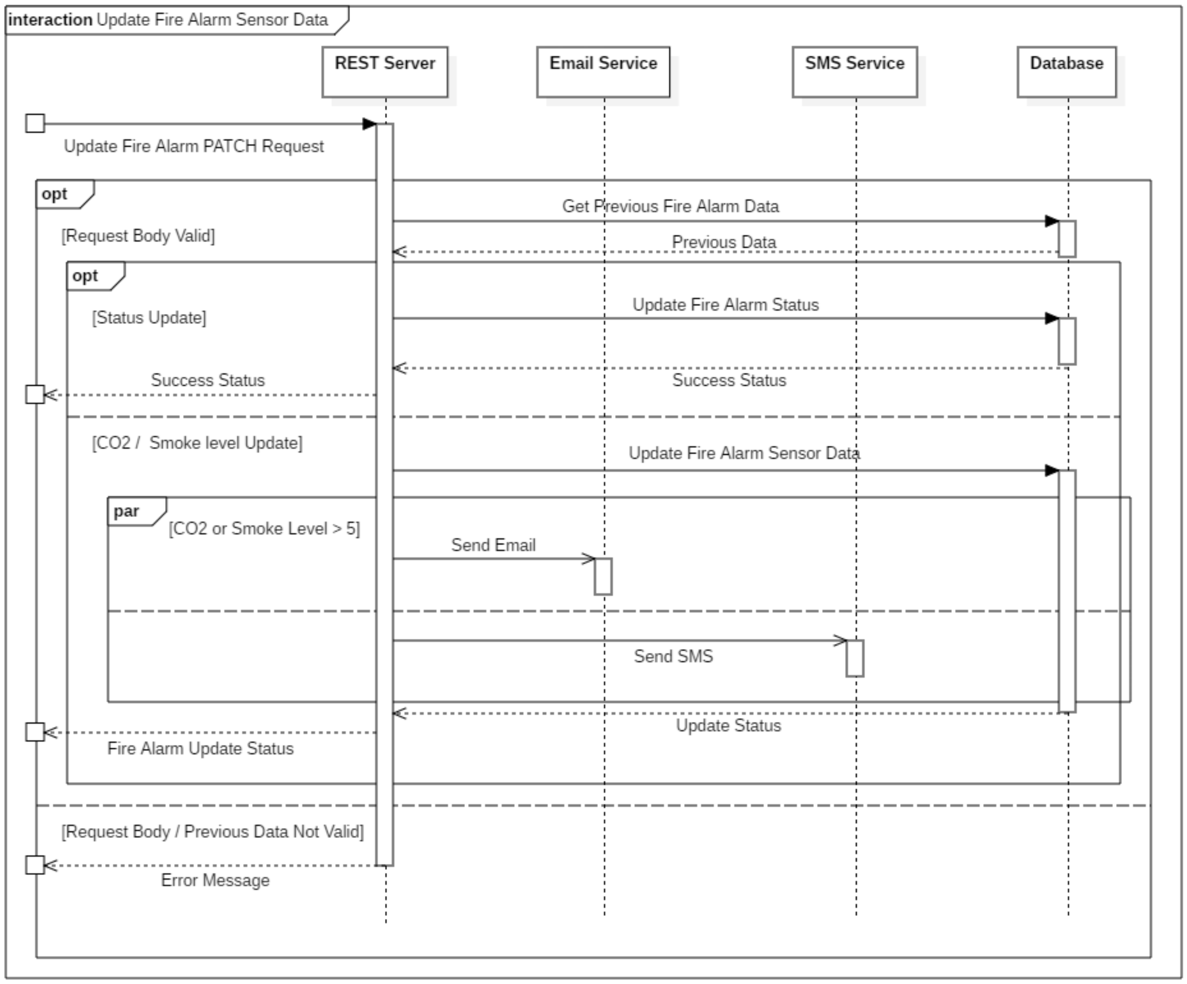
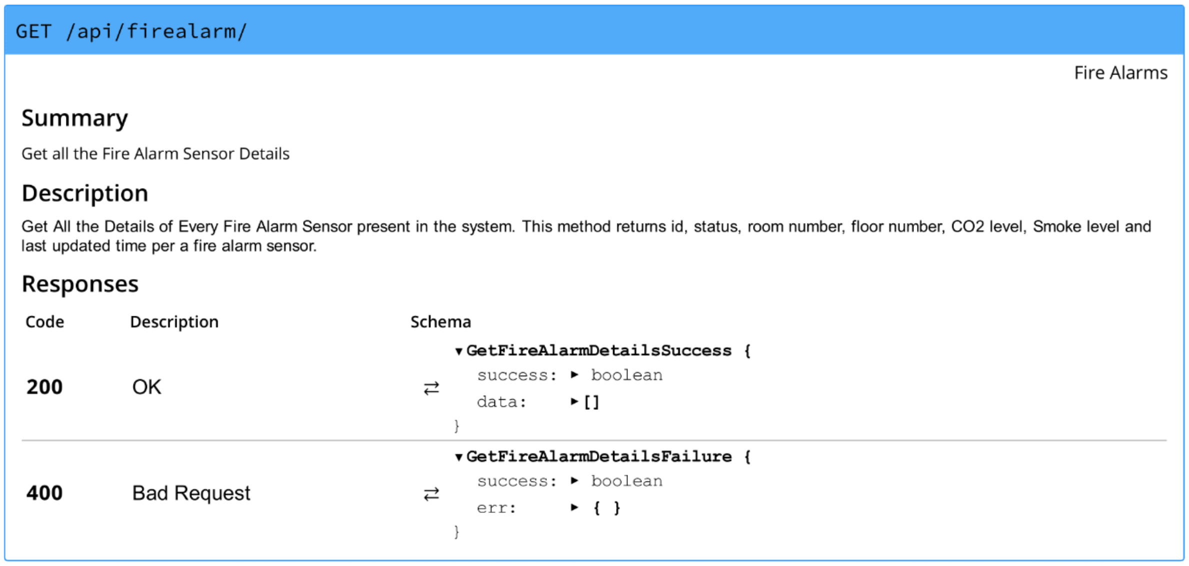


Figure 20: Fire Alarm Sensor Data Update REST Interface Sequence Diagram

5. Get Sensor Data of Fire Alarm Sensors



```
GET /api/firealarm/
```

Fire Alarms

Summary

Get all the Fire Alarm Sensor Details

Description

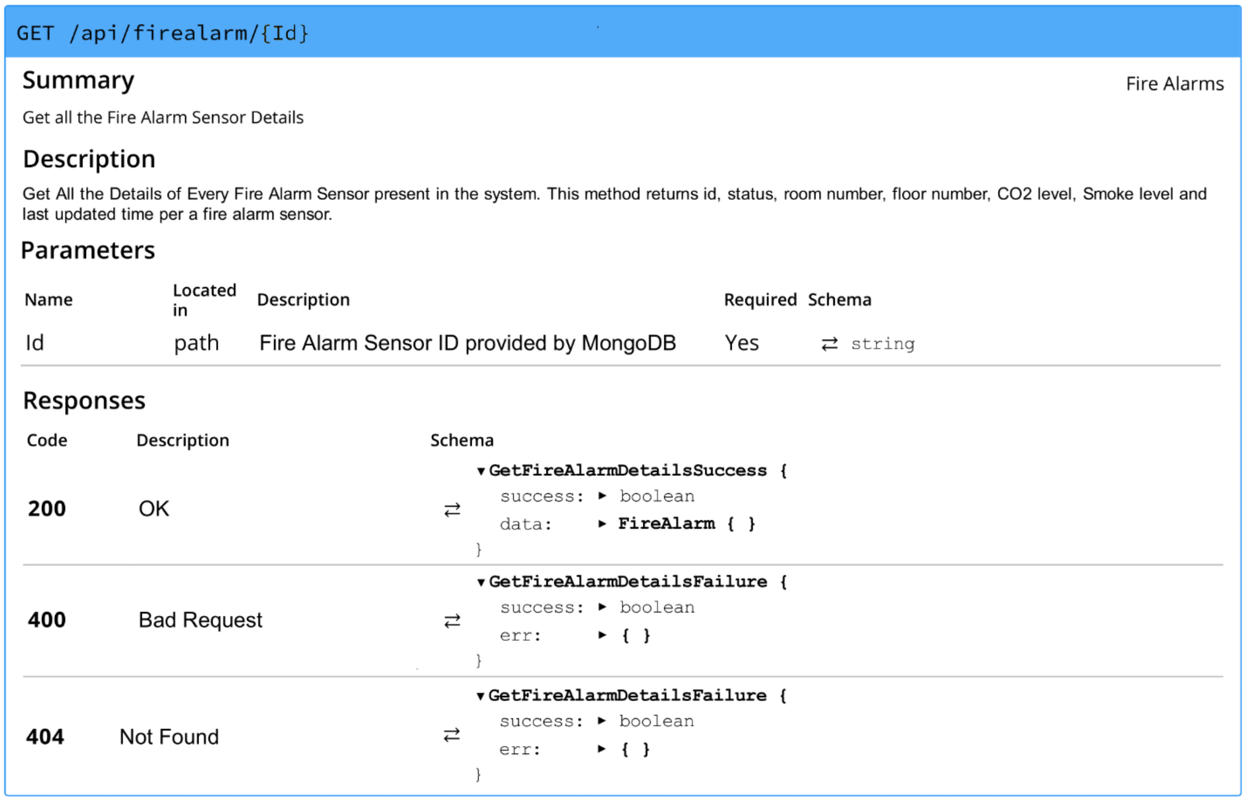
Get All the Details of Every Fire Alarm Sensor present in the system. This method returns id, status, room number, floor number, CO2 level, Smoke level and last updated time per a fire alarm sensor.

Responses

Code	Description	Schema
200	OK	▾GetFireAlarmDetailsSuccess { success: ▸ boolean data: ▸[] }
400	Bad Request	▾GetFireAlarmDetailsFailure { success: ▸ boolean err: ▸ { } }

Figure 21: The Interface to get Sensor data of Fire Alarm Sensors

This interface can be accessed by any user of the system including guests to the system. This method returns an Array of Fire Alarm Sensor document. This Request does not need any external parameters. If the user requires a particular Fire Alarm Sensor Details he can use the following interface.



GET /api/firealarm/{Id}

Summary

Fire Alarms

Get all the Fire Alarm Sensor Details

Description

Get All the Details of Every Fire Alarm Sensor present in the system. This method returns id, status, room number, floor number, CO2 level, Smoke level and last updated time per a fire alarm sensor.

Parameters

Name	Located in	Description	Required	Schema
Id	path	Fire Alarm Sensor ID provided by MongoDB	Yes	⇄ string

Responses

Code	Description	Schema
200	OK	⇄ ▾GetFireAlarmDetailsSuccess { success: ▸ boolean data: ▸ FireAlarm { } }
400	Bad Request	⇄ ▾GetFireAlarmDetailsFailure { success: ▸ boolean err: ▸ { } }
404	Not Found	⇄ ▾GetFireAlarmDetailsFailure { success: ▸ boolean err: ▸ { } }

Figure 22: Searching for a Fire Alarm Sensor Details Using ID

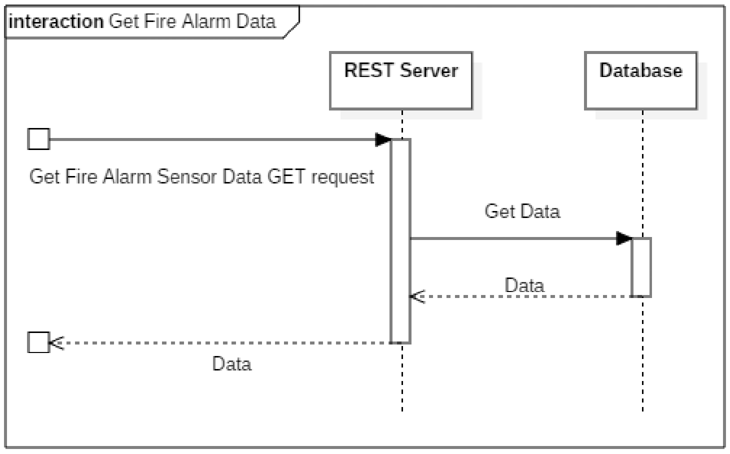
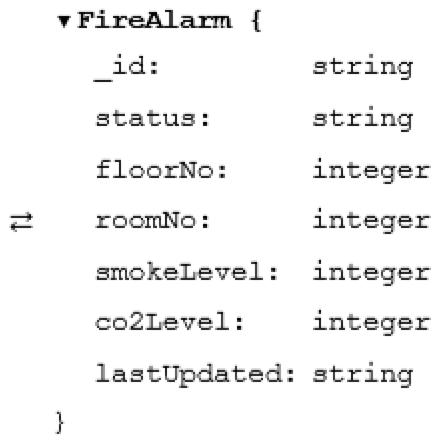

interaction Get Fire Alarm Data
REST Server
Database
Get Fire Alarm Sensor Data GET request
Get Data
Data
Data

Figure 23: Fire Alarm Data Retrieve REST Interface Sequence Diagram

In many HTTP method response when returning a Fire Alarm Sensor details it returns a following type JSON document. In GET request it returns an Array of this documents.



```
▾FireAlarm {
    _id:          string
    status:       string
    floorNo:      integer
⇄   roomNo:       integer
    smokeLevel:   integer
    co2Level:     integer
    lastUpdated:  string
}
```

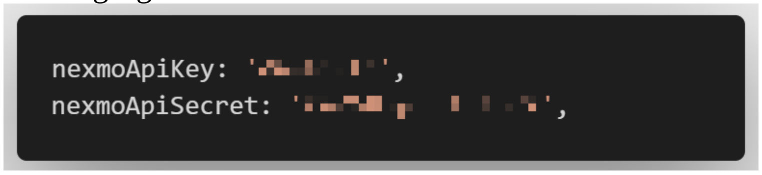
Figure 24: Fire Alarm Sensor JSON Modal

The REST API server also facilitates the services to Send Email and SMS Notifications. These services are implemented using some external services. Such as Nexmo, TextIt and Nodemailer. The both SMS service and the Email service is implemented in a way that they can be used by any function that need to send Email or SMS to a certain number or an Email address.

SMS Service

1. Nexmo SMS Service

To send SMS using the system there are some prerequisites. When using Nexmo the user must first have a User Account in Nexo. A user can create a user account using this link. Then after creating the account user must add the shown API key and the API Secret key to the config file like shown in the following figure.



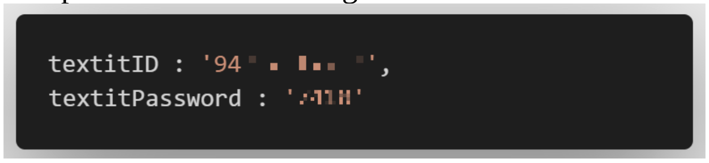
```
nexmoApiKey: '[redacted]',
nexmoApiSecret: '[redacted]',
```

Figure 25: Adding Nexmo API key and Secret Key

Even after adding the credentials to send a SMS using Nexmo the mobile number that the message is sent can only be the number the user provided during signing in. To send messages to other numbers user needs to add them to the “Test Number” in Nexmo Dashboard or purchase a plan from Nexmo.

2. TextIt SMS Service

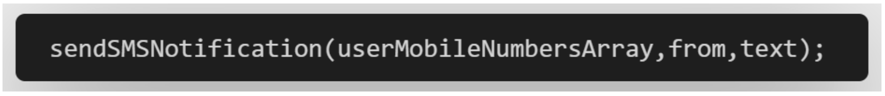
TextIT is a Sri Lankan company. When Sending a SMS using TextIt the user need to create an account in TextIt. A user can sign up to TextIt using this link. After signing up the user needs to add the TextIt ID and the password to the Config file as shown below.



```
textitID : '94[redacted]',
textitPassword : '[redacted]'
```

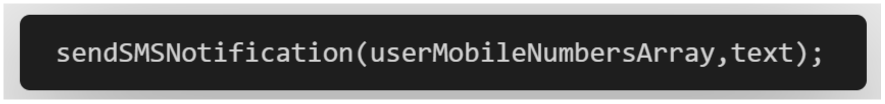
Figure 26: Adding TextIT Id and Password to config file

The above both SMS services can be used in any function that need to send SMS messages to a number/s. In Nexmo the SMS sender name can be changed. Since that the functions to send the SMS in TextIT and Nexmo are different.



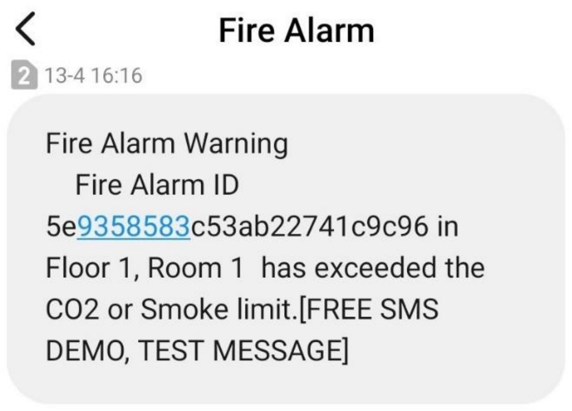
```
sendSMSNotification(userMobileNumbersArray,from,text);
```

Figure 27: Function to send SMS using Nexmo



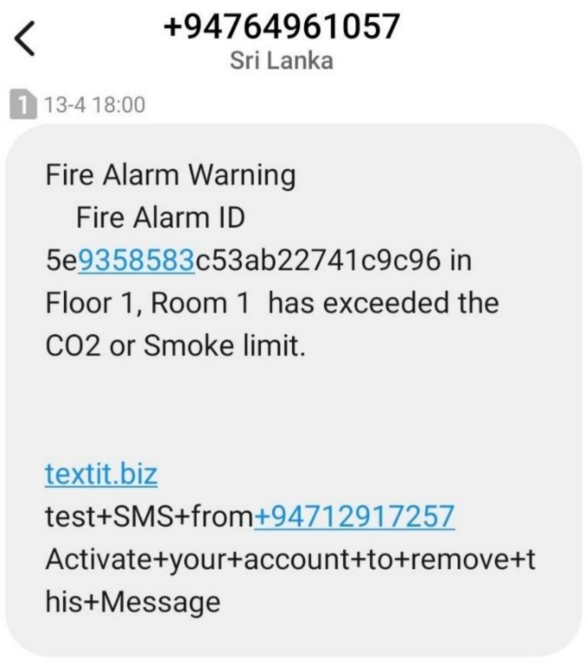
```
sendSMSNotification(userMobileNumbersArray,text);
```

Figure 28: Function to send SMS using TextIt



Fire Alarm Warning
Fire Alarm ID
5e9358583c53ab22741c9c96 in
Floor 1, Room 1 has exceeded the
CO2 or Smoke limit.[FREE SMS
DEMO, TEST MESSAGE]

Figure 29: SMS Sent using Nexmo SMS Service



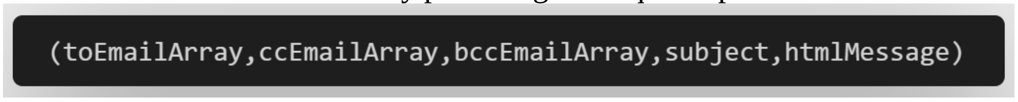
Fire Alarm Warning
Fire Alarm ID
5e9358583c53ab22741c9c96 in
Floor 1, Room 1 has exceeded the
CO2 or Smoke limit.

textit.biz
test+SMS+from+94712917257
Activate+your+account+to+remove+this+Message

Figure 30: SMS Sent Using TextIT SMS Service

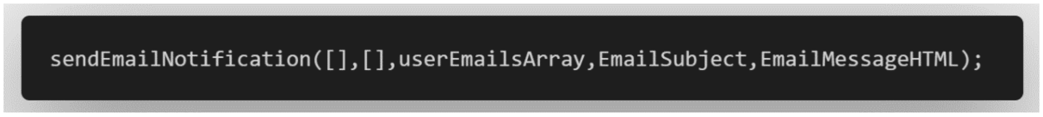
Email Service

The Email service is implemented using Nodemailer. Nodemailer is a module which makes sending Email with Node easy. The Email sending function can be used in any function that requires to send Emails. It can be used by providing the required parameters.



```
(toEmailArray,ccEmailArray,bccEmailArray,subject,htmlMessage)
```

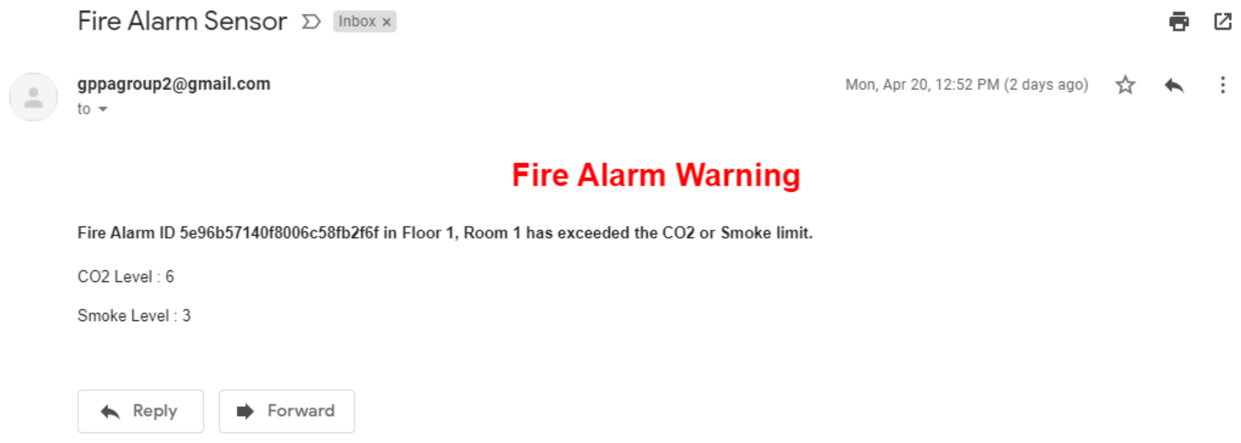
Figure 31: Required Data to send an Email using the Email Service



```
sendEmailNotification([],[],userEmailsArray,EmailSubject,EmailMessageHTML);
```

Figure 32: How the Email function is used in the Fire Alarm System

The REST API uses the Email and SMS services to send a notification to all the administrator accounts when a Fire Alarm Sensor's CO2 level or Smoke level gets higher than 5. This process is shown using a sequence diagram in Figure 20.



Fire Alarm Sensor Inbox ×

gppagroup2@gmail.com
to

Mon, Apr 20, 12:52 PM (2 days ago)

Fire Alarm Warning

Fire Alarm ID 5e96b57140f8006c58fb2f6f in Floor 1, Room 1 has exceeded the CO2 or Smoke limit.

CO2 Level : 6

Smoke Level : 3

Figure 33: Fire Alarm Warning Email Sent using the Email Service

Fire Alarm Sensor

Fire Alarm Sensor is a major component of the system since the processing of the data that are gathered using the Fire Alarm Sensors are the most crucial data in this system. The Fire Alarm sensors can use the REST API web interface provided to update the data. The data that are required to make the HTTP request is shown in the Figure 17.
In the system to emulate a Fire Alarm a simple dummy application was developed. This was developed using JAVA Swing. This dummy app sends the data to the system in 10 second intervals. Many instances of the application can be run.

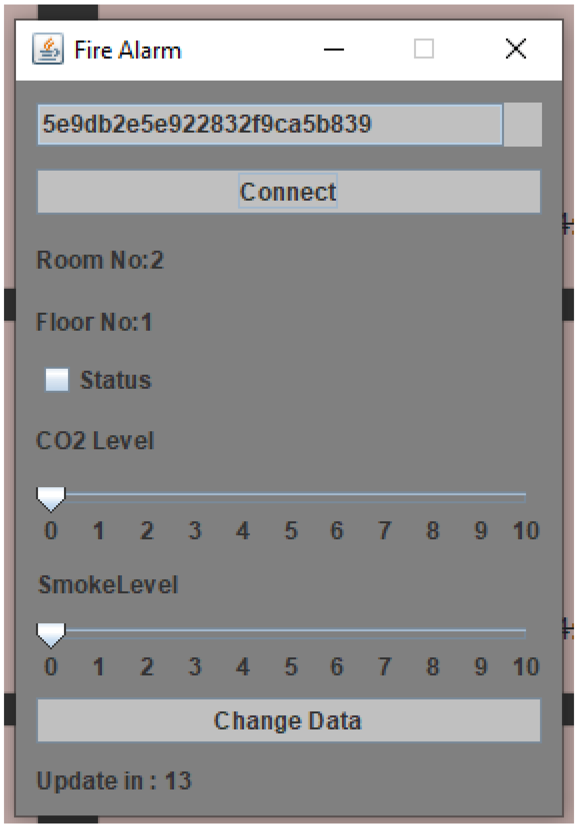
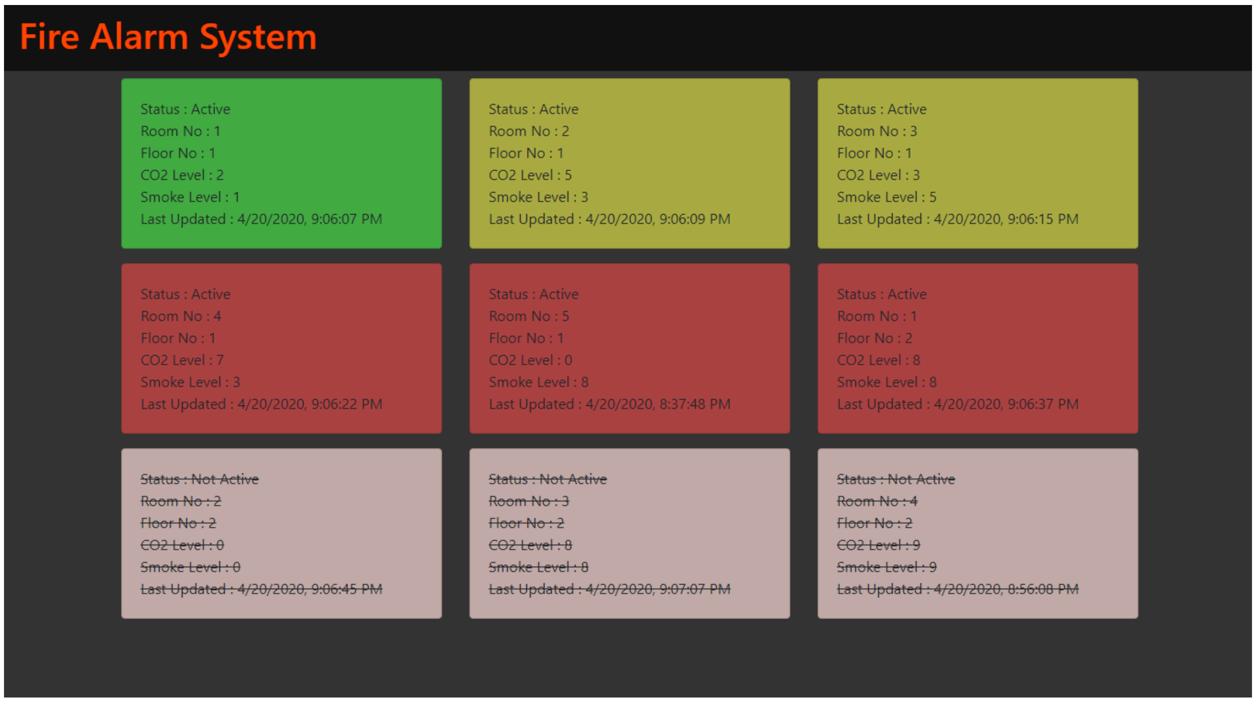


Figure 34: Dummy App to emulate a Fire Alarm Sensor

Web Client

The web client was developed using React JS. React JS is a Java Script library used to develop single page application. Since in the Fire Alarm System using the web UI is only limited to displaying the sensor data gathered by the fire alarm sensors only a single page web application is sufficient to accomplish the task.
Since React JS uses a Virtual DOM to track updates in the DOM and only render the necessary components, in the web interface, using react the Fire Alarm Sensor data can be updated without refreshing the whole web page which saves network resources.
Since JavaScript is made asynchronous using callbacks and promises, the revising of the fire alarm sensor data can be done without blocking the application execution.


Fire Alarm System

Status : Active
Room No : 1
Floor No : 1
CO2 Level : 2
Smoke Level : 1
Last Updated : 4/20/2020, 9:06:07 PM

Status : Active
Room No : 2
Floor No : 1
CO2 Level : 5
Smoke Level : 3
Last Updated : 4/20/2020, 9:06:09 PM

Status : Active
Room No : 3
Floor No : 1
CO2 Level : 3
Smoke Level : 5
Last Updated : 4/20/2020, 9:06:15 PM

Status : Active
Room No : 4
Floor No : 1
CO2 Level : 7
Smoke Level : 3
Last Updated : 4/20/2020, 9:06:22 PM

Status : Active
Room No : 5
Floor No : 1
CO2 Level : 0
Smoke Level : 8
Last Updated : 4/20/2020, 8:37:48 PM

Status : Active
Room No : 1
Floor No : 2
CO2 Level : 8
Smoke Level : 8
Last Updated : 4/20/2020, 9:06:37 PM

~~Status : Not Active~~
~~Room No : 2~~
~~Floor No : 2~~
~~CO2 Level : 0~~
~~Smoke Level : 0~~
~~Last Updated : 4/20/2020, 9:06:45 PM~~

~~Status : Not Active~~
~~Room No : 3~~
~~Floor No : 2~~
~~CO2 Level : 8~~
~~Smoke Level : 8~~
~~Last Updated : 4/20/2020, 9:07:07 PM~~

~~Status : Not Active~~
~~Room No : 4~~
~~Floor No : 2~~
~~CO2 Level : 9~~
~~Smoke Level : 9~~
~~Last Updated : 4/20/2020, 8:56:08 PM~~

Figure 35: Web Interface of the System

The Color coding of the cards are as follows,

- Green - CO2 and Smoke level under level 5
- Yellow – CO2 or Smoke or both levels are at level 5
- Red - CO2 or Smoke or both levels have exceeded level 5
- Pink and Text cut through – Fire Alarm Sensor is Not Active

RMI Server and Desktop Client

RMI Server connects the Desktop Client to the RESTful web services API. Using the RMI Server the Desktop client can use all the interfaces provided by the RESTful web services API.

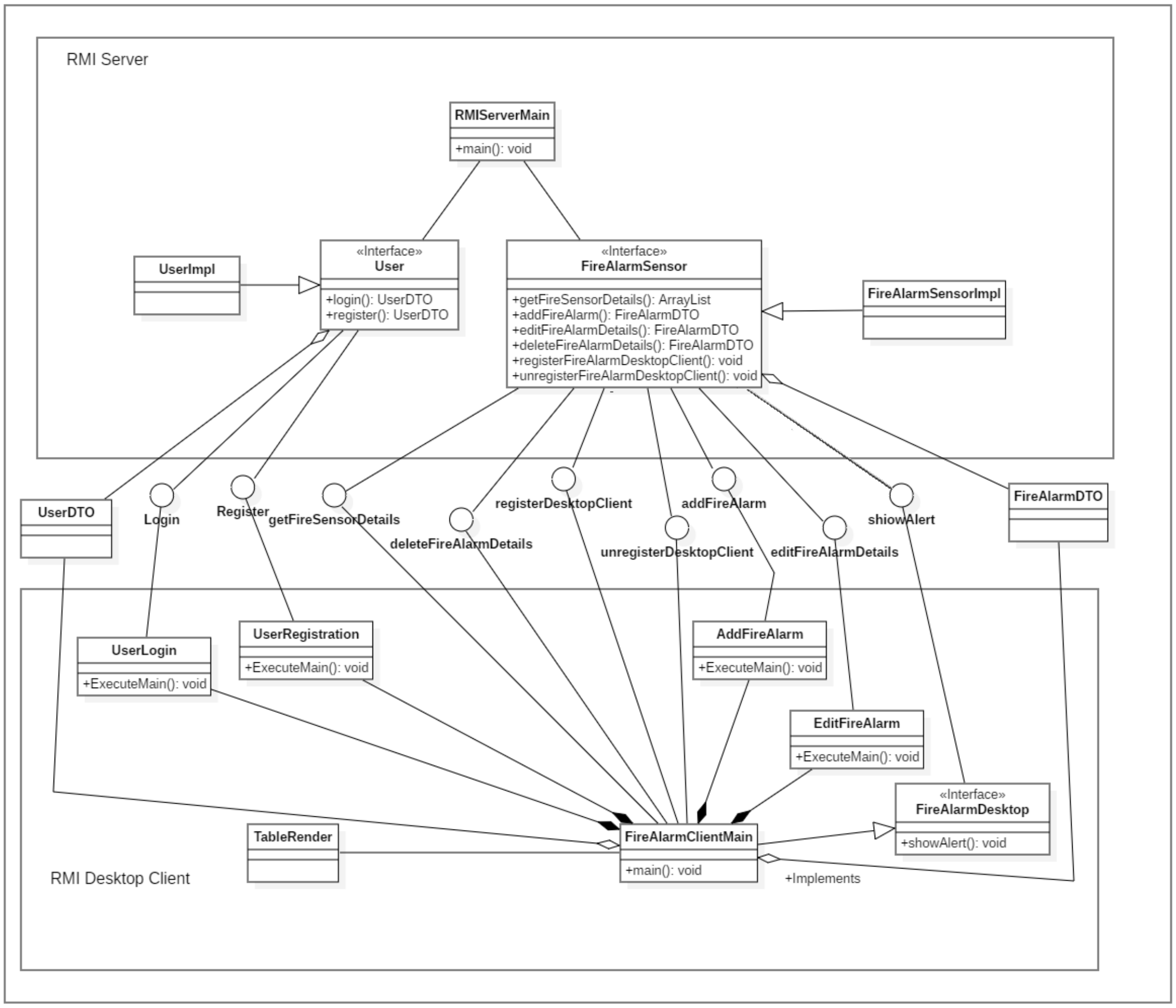


Figure 36: RMI Server and Desktop Client Class Diagram

As shown in the above figure the RMI Server and the Desktop client use 3 interface Classes and 2 Data Transfer Object Classes.
The 2 DTO Classes are as follows,

1. Fire Alarm DTO
2. User DTO

The Fire Alarm DTO and the User DTO are used to transfer between RMI Server and the Desktop Client. When JSON type response is received to the RMI Server after a HTTP request to the REST API the response data is stored in a FireAlarmDTO or UserDTO object. This object is then serialized and sent over the network.

The 3 Interfaces are as follows,

1. User
2. Fire Alarm Sensor
3. Fire Alarm Desktop

User Interface

The User interface provide the remote methods to login and register a user.

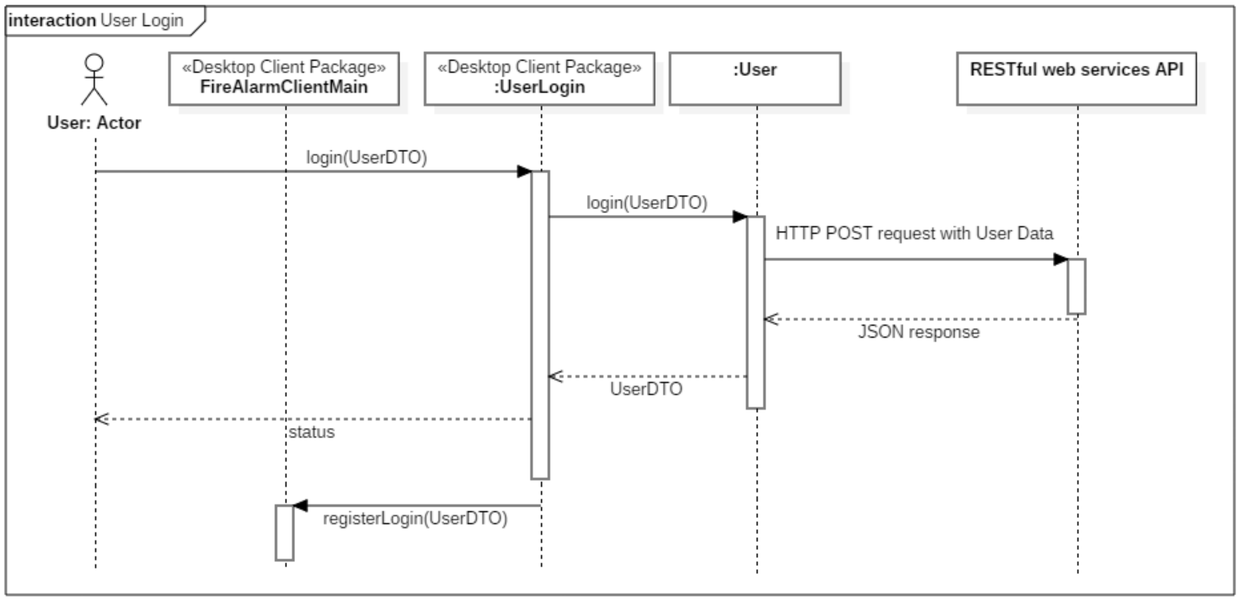


Figure 37:Sequence Diagram for User Login

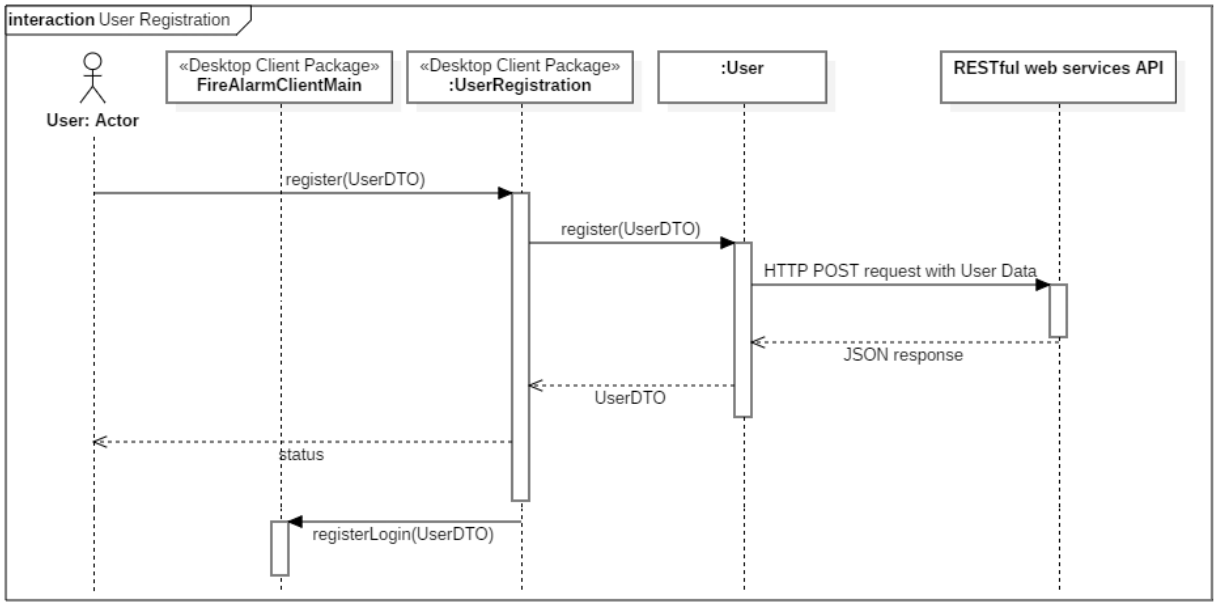

interaction User Registration
User: Actor
«Desktop Client Package» FireAlarmClientMain
«Desktop Client Package» :UserRegistration
:User
RESTful web services API
register(UserDTO)
register(UserDTO)
HTTP POST request with User Data
JSON response
UserDTO
status
registerLogin(UserDTO)

Figure 38: Sequence Diagram for User Registration

Fire Alarm Sensor Interface

Fire Alarm Sensor Interface provides method to Get Fire Alarm Sensor Data, Add Fire Alarm Sensors, Edit Fire Alarm Sensors, and Delete Fire Alarm Sensors. There is a method in RMI Server which gets Fire alarm sensor data from the RESTful web services API periodically and shows a notification in the registered Desktop Client Interface. The RMI Server provides methods to register and unregister from this service too.

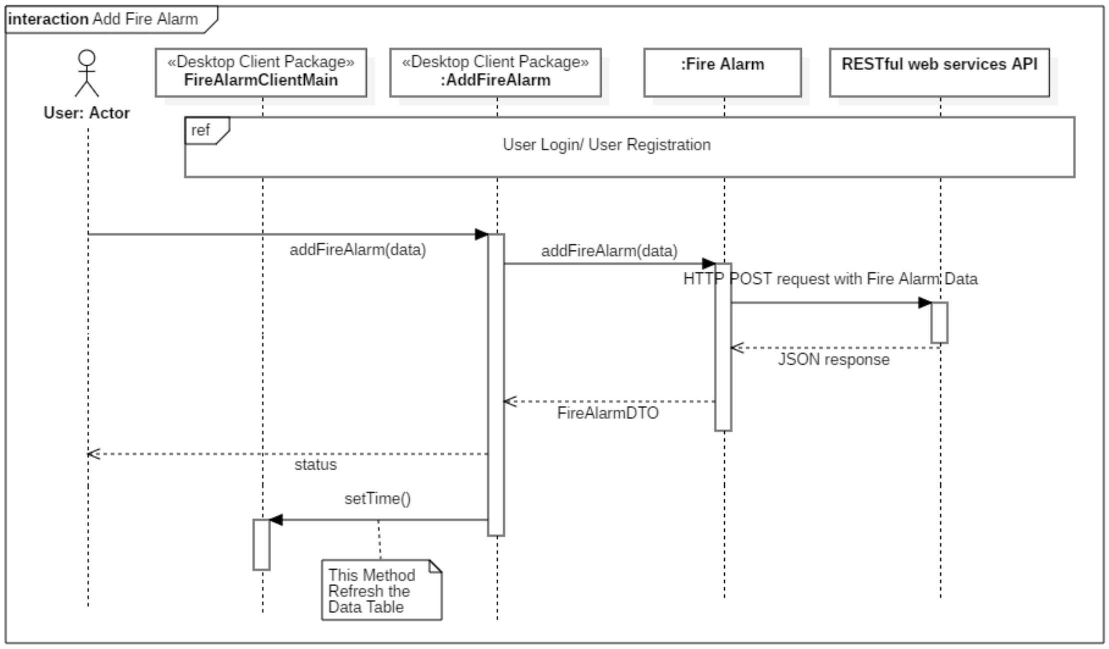

interaction Add Fire Alarm
User: Actor
«Desktop Client Package» FireAlarmClientMain
«Desktop Client Package» :AddFireAlarm
:Fire Alarm
RESTful web services API
ref
User Login/ User Registration
addFireAlarm(data)
addFireAlarm(data)
HTTP POST request with Fire Alarm Data
JSON response
FireAlarmDTO
status
setTime()
This Method Refresh the Data Table

Figure 39: Sequence Diagram for Adding Fire Alarms

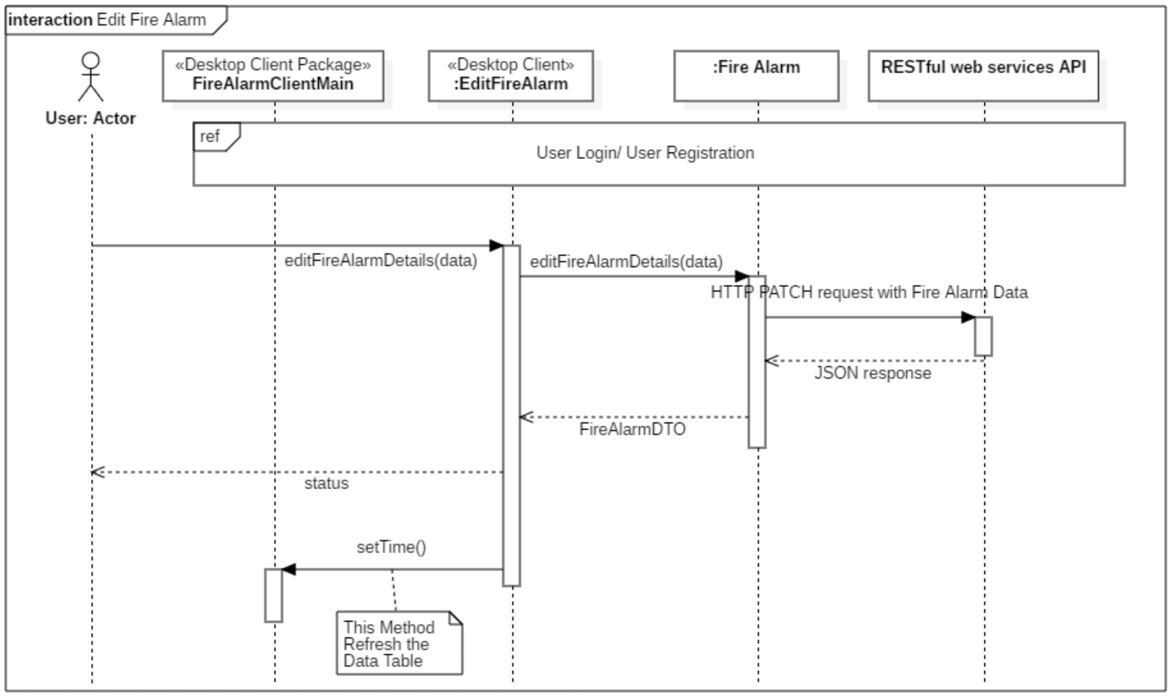

interaction Edit Fire Alarm
User: Actor
«Desktop Client Package» FireAlarmClientMain
«Desktop Client» :EditFireAlarm
:Fire Alarm
RESTful web services API
ref
User Login/ User Registration
editFireAlarmDetails(data)
editFireAlarmDetails(data)
HTTP PATCH request with Fire Alarm Data
JSON response
FireAlarmDTO
status
setTime()
This Method Refresh the Data Table

Figure 40: Sequence Diagram for Update Fire Alarm Sensor Details

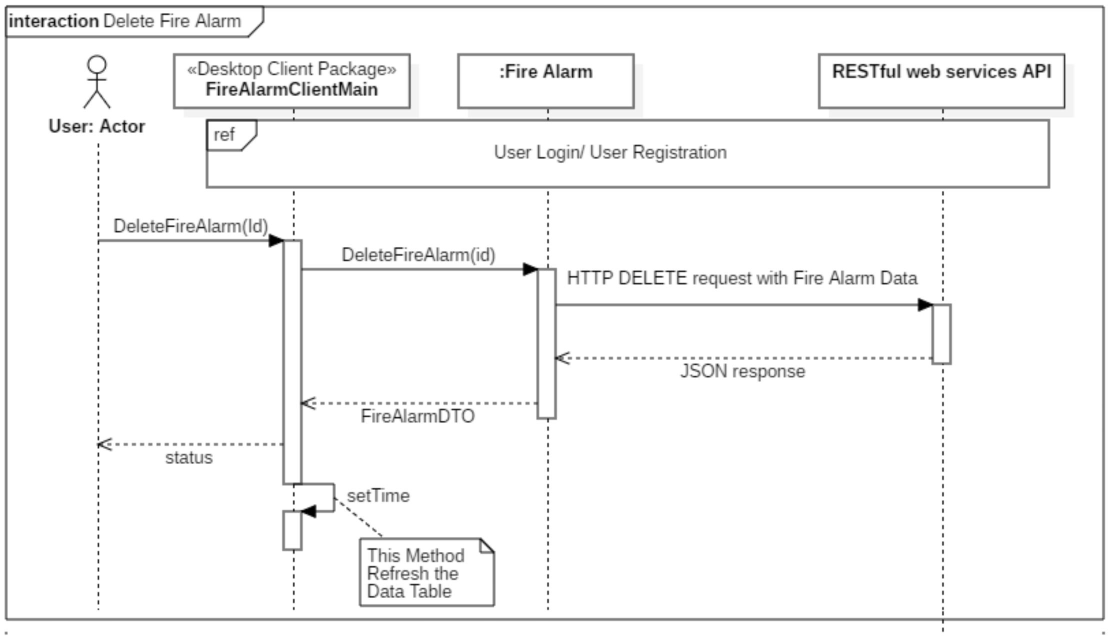

interaction Delete Fire Alarm
User: Actor
«Desktop Client Package» FireAlarmClientMain
:Fire Alarm
RESTful web services API
ref
User Login/ User Registration
DeleteFireAlarm(Id)
DeleteFireAlarm(id)
HTTP DELETE request with Fire Alarm Data
JSON response
FireAlarmDTO
status
setTime
This Method Refresh the Data Table

Figure 41: Sequence Diagram for Delete Fire Alarm Sensor

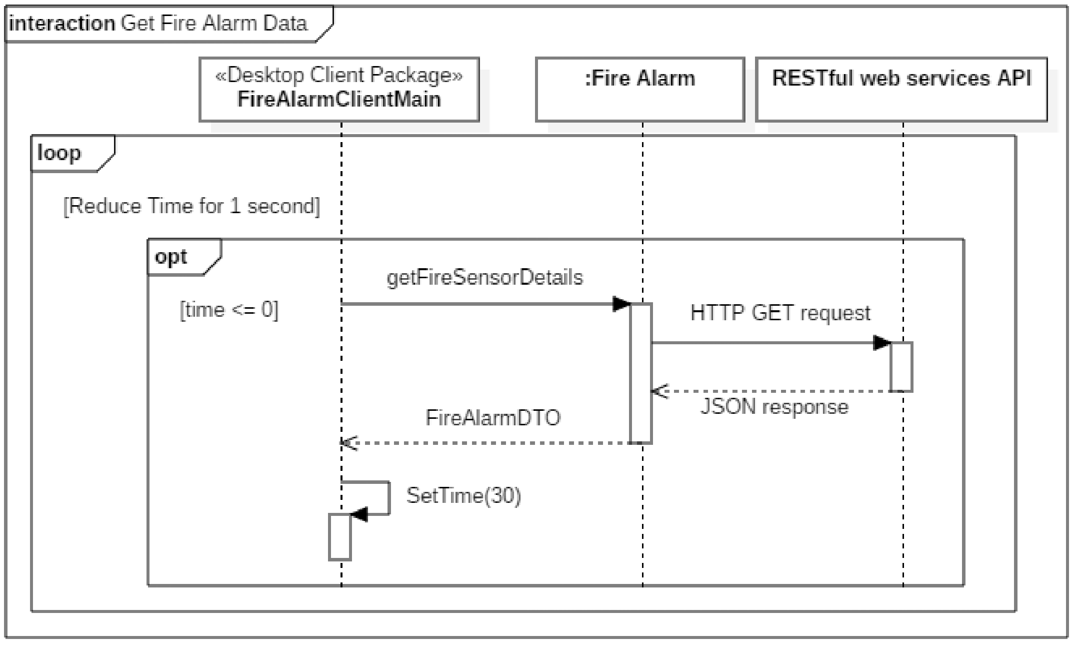


Figure 42:Sequence Diagram for getting Fire Alarm Sensor Data

Fire Alarm Desktop Interface

This interface is used to show an alert in the desktop client when one of the fire alarm sensors smoke level or CO2 level exceeds level 5. To receive this alert the desktop client must first register to the service using Fire Alarm Sensor interface. The system must unregister when closing the desktop client.

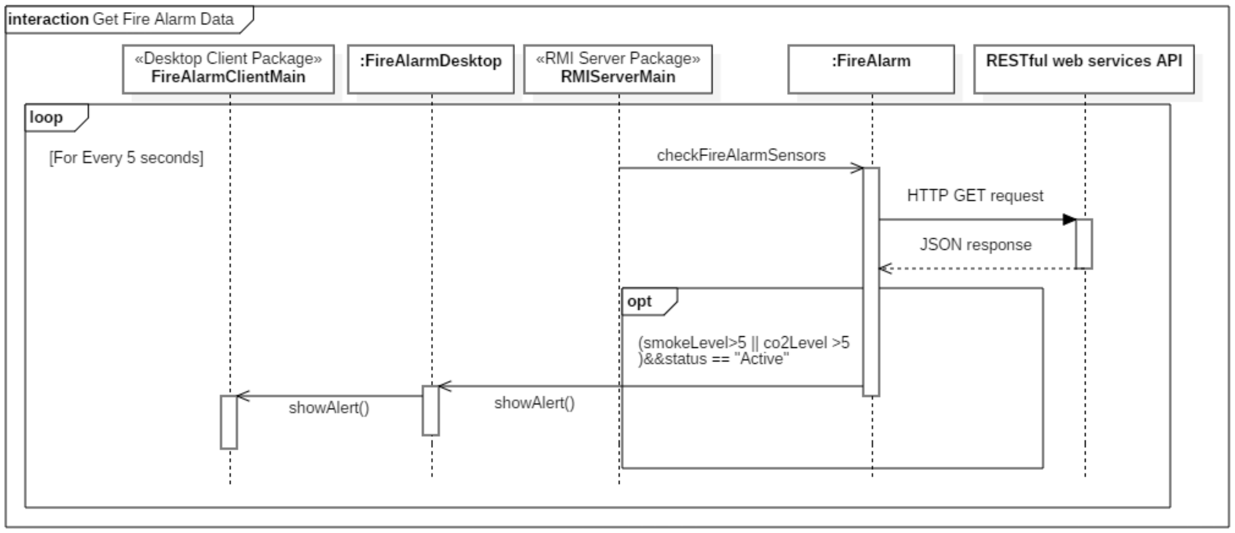
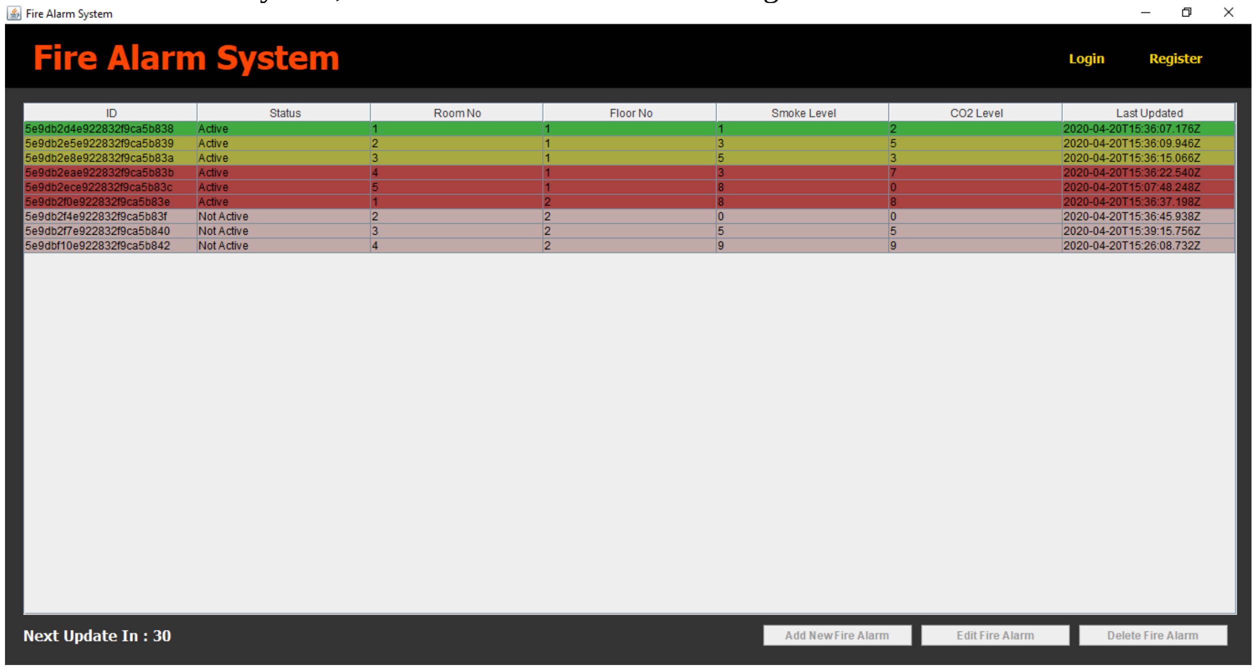

interaction Get Fire Alarm Data
«Desktop Client Package» FireAlarmClientMain
:FireAlarmDesktop
«RMI Server Package» RMIServerMain
:FireAlarm
RESTful web services API
loop
[For Every 5 seconds]
checkFireAlarmSensors
HTTP GET request
JSON response
opt
(smokeLevel>5 || co2Level >5)&&status == "Active"
showAlert()
showAlert()

Figure 43: Sequence Diagram for Showing alerts in desktop client

All the above-mentioned Interface and Data Transfer Object java files are stored in both RMI Server and the Java Client directories for easier development purposes. But when Deploying the system, the Client can be made thin using a RMI Codebase.



ID	Status	Room No	Floor No	Smoke Level	CO2 Level	Last Updated
5e9db2d4e922832f9ca5b838	Active	1	1	1	2	2020-04-20T15:36:07.176Z
5e9db2e5e922832f9ca5b839	Active	2	1	3	5	2020-04-20T15:36:09.946Z
5e9db2e8e922832f9ca5b83a	Active	3	1	5	3	2020-04-20T15:36:15.066Z
5e9db2eae922832f9ca5b83b	Active	4	1	3	7	2020-04-20T15:36:22.540Z
5e9db2ece922832f9ca5b83c	Active	5	1	8	0	2020-04-20T15:07:48.248Z
5e9db2f0e922832f9ca5b83e	Active	1	2	8	8	2020-04-20T15:36:37.198Z
5e9db2f4e922832f9ca5b83f	Not Active	2	2	0	0	2020-04-20T15:36:45.938Z
5e9db2f7e922832f9ca5b840	Not Active	3	2	5	5	2020-04-20T15:39:15.756Z
5e9dbf10e922832f9ca5b842	Not Active	4	2	9	9	2020-04-20T15:26:08.732Z

Next Update In : 30

Figure 44: User Interface of the Desktop Client

Component Diagram of the System

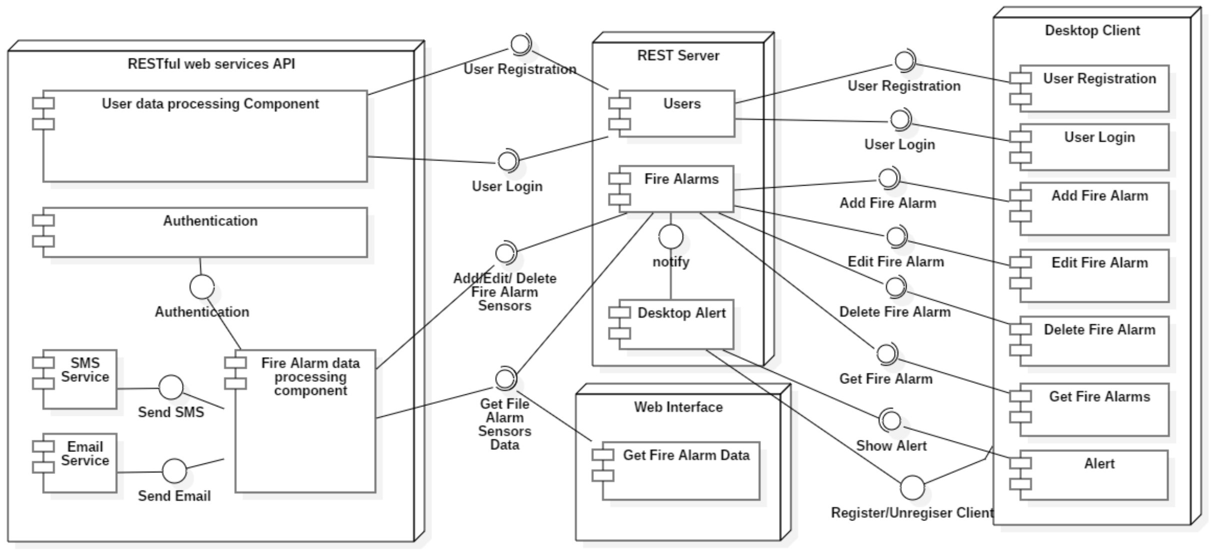
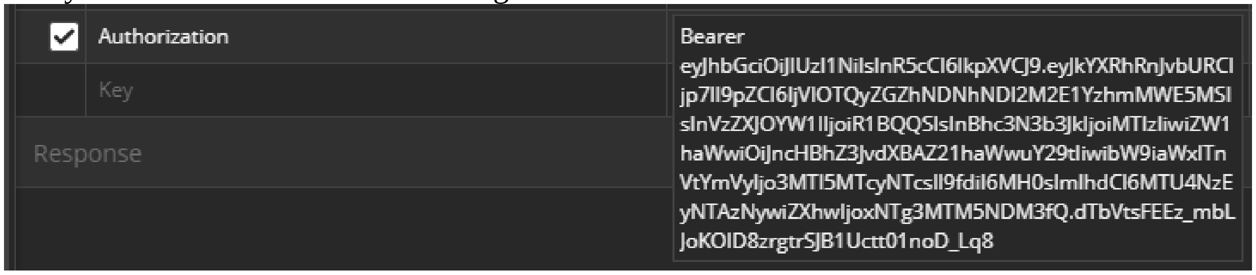


Figure 45: Component Diagram of the System

Authentication Mechanisms

When adding, updating or deleting a Fire Alarm Sensor from the system the user must first be verified. In the RESTful web services API this process is achieved using JSON Web Tokens. JWT generates a token using a secret key which is defined by the system developer with given data. This token can be made to expire after some time. JWT can also decode the token using the same secret key and return the JSON data.

Using the JWT the fire alarm system achieves user authentication. When a User registers or logins the REST API make a token with the user details. This token is then passed in the response. When the user makes another HTTP request that needs authentication the token must be included in the HTTP authentication header. If the token is not entered/ Invalid or timed out the system then returns an error message.



```
Bearer
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJkYXRhRnJvbURCI
jp7Il9pZCI6IjVlOTQyZGZhNDNhNDI2M2E1YzhmMWE5MSI
sInVzZXJOYW1lIjoiR1BQQSIsInBhc3N3b3JkIjoiMTIzIiwiZW1
haWwiOiJncHBhZ3JvdXBAZ21haWwuY29tIiwibW9iaWxlTn
VtYmVyIjo3MTI5MTcyNTcsIl9fdiI6MH0sImlhdCI6MTU4NzE
yNTAzNywiZXhwIjoxNTg3MTM5NDM3fQ.dTbVtsFEEz_mbL
JoKOID8zrgtrSJB1Uctt01noD_Lq8
```

Figure 46: Setting the Authentication header for HTTP Request in Postman

Appendix

REST API

Config

```
module.exports = {
    //Running of the System
    //If the port number is changed, there are many places to change the number in other applications
    //beacuse of that of port 5000 is in use, stop the other app using the port and use the port 5000
    //to start this app
    //if any case the port number is changed,
    //the user must change the HTTP Request URIs in following places
    //Website/src/Components/firealarm/fireAlarmMain.jsx - In axios.get() - change the 500 to the changed port
    //Desktop Application/Fire Alarm Sensor/src/fireAlarmSensor/FireAlarmImpl.java - find and replace all port 5000 to changed port
    //Desktop Application/Fire Alarm RMI Sensor/src/FireAlarmSensorImpl.java - find and replace all port 5000 to changed port
    //Desktop Application/User RMI Sensor/src/UserImpl.java - find and replace all port 5000 to changed port
    PORT : 5000,

    //mongoDB uri in the current system
    //the default port is 27017
    //if the system is changed, Change this in order to run the database
    mongodb: "mongodb://localhost:27017/fireAlarmSystem",

    //secret key to generate JSON Web Tokens
    secret: "firealertusertokensecret",

    //Gmail Credentials
    //First turn on Allow less secure apps in GMAIL account using the below link
    //https://myaccount.google.com/u/0/lesssecureapps
    userName: "XXXXXXXX@gmail.com",
    password: "XXXXXXXX",


    //Nexmo Credentials
    //Used to send SMS
    //First create an account in nexmo using the below link
    //https://dashboard.nexmo.com/sign-in
    nexmoApiKey: 'XXXXXXXX',
    nexmoApiSecret: 'XXXXXXXXXXXXXXX',
    //textit credentials
    //signup using the below link
    //http://textit.biz/signup_1.php
    textitID : '94XXXXXXXXX',
    textitPassword : 'XXXX'
}
```

Authentication Middleware

```
//Importing JSON web token package to admin authorization
const jwt = require('jsonwebtoken');
//Importing the Secret Key required to generate/Verify the Token for a User
const secret = require('../config/config').secret;

//Exporting middleware functions
module.exports = {
    //Function to authenticate user based on the JSON web token
    //If the JSON Web token is expired or not provided it will provide a
    //401 Unauthorized error status and a message
    //This middelware is used in every HTTP method where an admin authorizarion i
s requried
    verifyUser : (req,res,next)=>{
        //getting the authorization token from the HTTP Request
        let token = req.headers['authorization'];
        //Checking if the Token is empty or not
        if(token){
            //Checking for the validity of the token and if valid get the token s
liced
            if(token.startsWith('Bearer ')){
                token = token.slice(7,token.length);
            }
            //verifying the token using the secret key and getting the embedded u
ser data
            jwt.verify(token,secret,(err,authData)=>{
                //Returining an error if the token is not valid
                //401 = Unauthorized access
                if(err){
                    return res.status(401).send({success:false,msg:"Token is not 
Valid"})
                }else{
                    //Sending the user data to the REST process for further funct
ion execution
                    //by embeding it to the req header
                    req.authData = authData;
                    next();
                }
            });
        }else{
            //Sending an Unauthorized access status if the Token is not provided
            return res.status(401).send({success:false,msg:"Authorization Token n
ot provided. Login first."})
        }
    }
}
```

Send Email Notifications Function

```
//importing nodemailer package
const nodemailer = require('nodemailer');

//import GMAIL credentials
const userName = require('../config/config').userName;
const password = require('../config/config').password;

//This function takes all the required details tp send an Email
//This function can be used by any service that provides the relevent data needed
 to send an Email
module.exports = async (toEmailArray,ccEmailArray,bccEmailArray,subject,htmlMessa
ge) => {
    //Creating a NodeMailer Transport using the Gmail credentials
    const transporter = await nodemailer.createTransport({
        service: "Gmail",
        auth: {
            user: userName,
            pass: password
        }
    });
    //Sending the Email
    let info = await transporter.sendMail({
        to: toEmailArray.toString(),
        cc: ccEmailArray.toString(),
        bcc: bccEmailArray.toString(),
        subject: subject,
        html:htmlMessage

    });
    console.log("Email Sent");
}
```

Send SMS Function

```
//Importing Nexmo to send sms
const Nexmo = require('nexmo');
```

```
//Importing Api Key and Secret key from config
const apiKey = require('../config/config').nexmoApiKey;
const apiSecret = require('../config/config').nexmoApiSecret;

//Since this is only a test version the message are only sent to the registered t
est numbers.
//So to test the number you first need to sign up to the nexmo using the link in
the config file.


//Using the Nexmo Credentials to create the initial nexmo Object
const nexmo = new Nexmo({
    apiKey,
    apiSecret
});

//This function takes all the relavent details required to send a SMS
//This service can be used by anyone that provides the relavent information
module.exports = async (userMobileNumbersArray,from,text) => {
    //Taking all the Mobile Numbers provided one by one and sending the Message
    userMobileNumbersArray.map((value) => {
        //Sending the SMS
        nexmo.message.sendSms(from, value, text, (err, responseData) => {
            if (err) {
                console.log(err);
            } else {
                if (responseData.messages[0]['status'] === "0") {
                    console.log("Message sent successfully.");
                } else {
                    console.log(`Message failed with error: ${responseData.messag
es[0]['error-text']}`);
                }
            }
        });

    });
}
```

Send SMS Function 2

```
//Importing http package to send sms via a REST Api
const http = require('http');

//importing textit account credentials
const id = require('../config/config').textitID;
const pass = require('../config/config').textitPassword;


//This function takes all the relavent details required to send a SMS
//This service can be used by anyone that provides the relavent information
module.exports = async (userMobileNumbersArray,text) => {
    //Encoding the Text to a URI format since TextIt uses the URI to get the mobi
le Number and the text message
    //The encoing type is the percentage encoding
    text = encodeURIComponent(text);

    //Uncomment the code to test the system
    // userMobileNumbersArray.map((value) => {
    //     http.get(`http://textit.biz/sendmsg/index.php?id=${id}&pw=${pass}&to=$
{value}&text=${text}`, (response) => {
    //         console.log(response.statusCode);
    //         if the returned status code is 200 the SMS is successfuly sent
    //     });
    // });

    console.log("Message Sent");
}
```

Mongo Schema Users

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

const UserSchema = mongoose.Schema({
    email:{
        type:String,
        required:true
    },
    userName:{
        type:String,
        required:true
    },
    mobileNumber:{
        type:Number,
        required:true
    },
    password:{
        type:String,
        required:true
    }
});

module.exports = User = mongoose.model('User',UserSchema);
```

Mongo Schema Fire Alarms

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

const fireAlarmSchema = mongoose.Schema({
    status: {
        type:String,
        default:"Not Active"
    },
    floorNo: {
        type: Number,
        required: true
    },
    roomNo: {
        type: Number,
        required: true
    },
    smokeLevel:{
        type:Number,
        default:0
    },
    co2Level: {
        type:Number,
        default:0
    },
    lastUpdated:{
        type: Date,
        required : true
    }
});

module.exports = FireAlarm = mongoose.model('FireAlarm', fireAlarmSchema);
```

Fire Alarm API Implementation

```
//Importing express js required modules
const express = require('express');
const router = express.Router();

//Importing Fire Alarm Mongoose model
const fireAlarm = require('../../models/FireAlarm');

//importing User mongose modal
const user = require('../../models/User');
```

```
//Importing middleware to authorize admin login
const userAuthorization = require('../../middleware/FireAlarmsMiddleware').verify
User;

//Importing SMS and Email Sender functions
const sendEmailNotification = require('../../middleware/SendEmailNotification');
const sendSMSNotification = require('../../middleware/SendSMSNotification2');
// const sendSMSNotification = require('../../middleware/SendSMSNotification');

//Adding a Fire Alram
//HTTP POST request
//Adding a firealram if Room number and floor number are provided.
//The default status value is not active and
//CO2 level and Smoke Level is 0
//The last Updated time is the added time
//Only the admin can add fire alarms. JWT tokens take care of that
//returns a JSON with a attribute msg with the Fire Alarm details
//returns a JSON with the not provided Details if one of the details are not prov
ided
router.post('/addFireAlarm', userAuthorization, (req, res) => {
    const data = req.body;
    if (data.roomNo && data.floorNo) {
        let fireAlarmData = {
            roomNo: data.roomNo,
            floorNo: data.floorNo,
            lastUpdated: new Date()
        }
        //Inserting data to MongoDB
        fireAlarm.create(fireAlarmData, (err, savedData) => {
            if (err) return res.status(400).send({ success: false, err: err });
            res.status(201).send({
                msg: `FireAlarm Created at Floor ${savedData.floorNo} Room ${save
dData.roomNo} @${savedData.lastUpdated}.`,
                success: true,
                savedData
            })
        });

    } else {
        //sending a 400 (Bad Request) error when input data is not correct
        res.status(400).send({
            success: false,
            msg: "roomNo or floorNo or both are not provided",
            roomNo: data.roomNo ? "Valid" : "Empty",
            floorNo: data.floorNo ? "Valid" : "Empty"
        });
    }

});


//Update Fire Alarm Details
//HTTP PATCH request
//PATCH Req means only certain data are changed.
//Not the Whole Fire Alram
//Need Admin Authorization
//At least one parameter must be changed
//The Room Number, Floor Number can be changes.
//If the id didnt match any existing fire alarm ID returns a 404 not found error
router.patch('/update/:id', userAuthorization, async (req, res) => {
    const fireAlarmId = req.params.id;
    const data = req.body;
    if (data.roomNo || data.floorNo) {
        let error = false;
        //getting the previous data
        let fireAlarmPervioudData = await fireAlarm.findOne({ _id: fireAlarmId })
.catch((err) => { error = true; return res.status(404).send({ success: false, err
: err }); });
        if (error) {
            return;
        }
        //updating the required fields
        if (fireAlarmPervioudData) {
            let fireAlarmNewData = {
                "roomNo": data.roomNo ? data.roomNo : fireAlarmPervioudData.roomN
o,
                "floorNo": data.floorNo ? data.floorNo : fireAlarmPervioudData.fl
oorNo,
            }
            //updating the data in the mongoDB
            fireAlarm.updateOne({ _id: fireAlarmId }, fireAlarmNewData, (err) =>
{
                //sending a 400(Bad Request) error if data types doesn't match or
 any other mongo error
                if (err) {
                    return res.status(400).send({ success: false, err: err });
                }
                //returning a success message
                res.status(200).send({
                    success: true,
                    msg: 'Fire Alarm Updated.'
                });
            });
        } else {
            //returning 404 (Not found) error if the id is invalid
            res.status(404).send({ success: false, msg: "Not Found" });
        }
    } else {
        //returing 400 (Bad Request ) error if the input fields are not valid
        res.status(400).send({
            success: false,
            msg: "roomNo and floorNo are not provided. Provide one of the field o
r both.",
            roomNo: "Empty",
            floorNo: "Empty"
        });
    }
});

//Delete an Fire Alarm
//HTTP DELETE request
//Need admin authentication
router.delete("/:id", userAuthorization, (req, res) => {
    const fireAlarmID = req.params.id;
    fireAlarm.findByIdAndDelete(fireAlarmID, (err, deletedData) => {
        if (err) {
            return res.status(400).send({ success: false, err: err });
        } else {
            if (deletedData) {
                //sending a 200 (Ok) if the firealarm deleted successfully
                res.status(200).send({ success: true, msg: `FireAlarm ID ${fireAl
armID} deleted successfully.`, deletedData });
            } else {
                //since the firealarm data is not existing in the DB the process
is a success
                //This is the Idempotent property of the DELETE HTTP request
                res.status(200).send({ success: true, msg: "Already Deleted or In
valid firealarm ID" });
            }

        }

    });
});
```

```
//update firealarm details
//HTTP PATCH requets
//used to update CO2 Level, Smoke Level, Status of a fire alarm
//CO2 level, Smoke level are required
//Don't need authentication
//Will automatically update the lastupdated time
router.patch('/:id', async (req, res) => {
    let fireAlarmID = req.params.id;
    let data = req.body;
    let error = false;
    let fireAlarmPrevious = await fireAlarm.findById(fireAlarmID).catch((err) =>
{ if (err) error = true; return res.status(404).send({ success: false, err: err }
) });
    if (error) {
        return
    }
    if (fireAlarmPrevious) {
        if (data.status) {
            fireAlarm.findByIdAndUpdate(fireAlarmID, { status: data.status ,$inc:
{__v:+1}}, (err) => {
                if (err) {
                    return res.status(400).send({ success: false, err: err });
                } else {
                    res.status(200).send({ success: true, msg: "Updated" });
                }
            });
        } else if (data.co2Level && data.smokeLevel) {
            if (data.co2Level <= 10 && data.smokeLevel <= 10 && data.co2Level >=
0 && data.smokeLevel >= 0) {
                let fireAlarmNew = {
                    "co2Level": data.co2Level,
                    "smokeLevel": data.smokeLevel,
                    "lastUpdated": new Date(),
                    $inc : {__v:+1}
                }
                if (data.co2Level > 5 || data.smokeLevel > 5) {
                    //sending the Email and the SMS only one time
                    //The email & sms is sent again only if the co2 level and smo
ke level
                    //droped under 5 and get increased again
                    if (fireAlarmPrevious.co2Level <= 5 && fireAlarmPrevious.smok
eLevel <= 5) {
                        //Assign the previous data to the new data
                        //This is done because the new data should have the id, r
oom no, floor no in order to send the
```

```
                        //SMS and the email
                        fireAlarmNew = fireAlarmPrevious;
                        //replace the co2 level and the smoke level with the new 
data
                        fireAlarmNew.co2Level = data.co2Level;
                        fireAlarmNew.smokeLevel = data.smokeLevel;

                        let userEmailAndMobile = await user.find({}, { _id: 0, "e
mail": 1, "mobileNumber": 1  }).catch((err) =>{error=true; res.status(500).send({
 success: false, err: err })});
                        if(error){
                            return;
                        }
                        let userEmailsArray = userEmailAndMobile.map((value) => {
                            return value.email;
                        });
                        userEmailsArray = userEmailsArray.filter((value, index, s
elf) => {
                            return self.indexOf(value) === index;
                        });

                        let EmailMessageHTML=`<h1 style="color: red;font-
weight: bold;text-align:center">Fire Alarm Warning</h1>
                        <h4>Fire Alarm ID ${fireAlarmNew._id} in Floor ${fireAlar
mNew.floorNo}, Room ${fireAlarmNew.roomNo}  has exceeded the CO2 or Smoke limit.<
/h4>
                        <p>CO2 Level : ${fireAlarmNew.co2Level}</p>
                        <p>Smoke Level : ${fireAlarmNew.smokeLevel}</p>`;
                        let EmailSubject = "Fire Alarm System";
                        sendEmailNotification([],[],userEmailsArray,EmailSubject,
EmailMessageHTML);


                        let userMobileNumbersArray = userEmailAndMobile.map((valu
e) => {
                            return ('94' + value.mobileNumber);
                        });
                        userMobileNumbersArray = userMobileNumbersArray.filter((v
alue, index, self) => {
                            return self.indexOf(value) === index;
                        });
                        const from = 'Fire Alarm Warning';

                        const text = `Fire Alarm Warning
                        Fire Alarm ID ${fireAlarmNew._id} in Floor ${fireAlarmNew
.floorNo}, Room ${fireAlarmNew.roomNo}  has exceeded the CO2 or Smoke limit.
                          `;

                        //This method is if the SMS is sent via Nexmo
                        // sendSMSNotification(userMobileNumbersArray,from,text);
                        //This method is if the SMS is sent via TextIt
                        sendSMSNotification(userMobileNumbersArray,text);
                    }
                }
                fireAlarm.findByIdAndUpdate(fireAlarmID, fireAlarmNew, (err) => {
                    if (err) {
                        return res.status(400).send({ success: false, err: err })
;
                    } else {
                        res.status(200).send({ success: true, msg: "Updated" });
                    }
                });
            } else {
                res.status(400).send({
                    success: false,
                    msg: "co2level and smokeLevel must be between 0 and 10",
                    co2Level: data.co2Level >= 0 && data.co2Level <= 10 ? "Valid"
 : "CO2 Level must be between 0 and 10",
                    smokeLevel: data.smokeLevel >= 0 && data.smokeLevel <= 10 ? "
Valid" : "Smoke Level must be between 0 and 10",
                });
            }
        } else {
            res.status(400).send({
                success: false,
                msg: "co2level and smokeLevel or status must be provided",
                co2Level: data.co2Level ? "Valid" : "Empty",
                smokeLevel: data.smokeLevel ? "Valid" : "Empty",
                status: data.status ? "Valid" : "Empty",
            });
        }
    } else {
        res.status(400).send({ success: false, msg: "Invalid Fire Alarm ID." });
    }
});

//Get the fire Alarm Details
//HTTP GET Request
//no authorization needed
```

```
router.get("/", (req, res) => {
    fireAlarm.find((err, data) => {
        if (err) {
            return res.status(400).send({ success: false, err: err });
        } else {
            res.status(200).send({ success: true, data });
        }
    });

});

//Get a particular Fire Alarm sensor Details
//HTTP GET Request
//No Authorization needed
router.get("/:id", (req, res) => {
    let id = req.params.id;
    fireAlarm.findById(id,(err, data) => {
        if (err) {
            return res.status(400).send({ success: false, err: err });
        } else {
            if(data){
                res.status(200).send({ success: true, data });
            }else{
                res.status(404).send({success:false,msg:"Not Found"});
            }

        }
    });

});

module.exports = router;
```

User API implementation

```
//importing required express modules
const express = require('express');
const router = express.Router();

//importing JSON Web Token for user account authorization
const jwt = require('jsonwebtoken');
//importing JSON Web Token Secret Key from config file
const secret = require('../../config/config').secret;

//importing user Mongoose Model
const User = require('../../models/User');
```

```
//Adding a User Account
//HTTP POST Request
//requried details are UserName, Password, Email, Mobile Number
//The email and Mobile Number is required since there is a process to
//inform the user in case of an emergency
//mobile number is 9 digits number without 0
//And this service is only valid for Sri lankan numbers since before sending +94
is added to the number
//Every detail is requred in order to make an user account
//If some details are not provided the system will provide a 400 Bad Request stat
us and
//a message with the unavailavle parameters
//When signedUp the server provide an authorization token to the user.
//This token will be stored in the session storage in the browser
//In the case of a desktop application it will be stored in a Global Variable.
//The provided token expires in 4 hours after generation
router.post('/signup', (req, res) => {
    const data = req.body;
    if (data.userName && data.password && data.email && data.mobileNumber) {
        let UserData = {
            userName: data.userName,
            password: data.password,
            email: data.email,
            mobileNumber: data.mobileNumber
        }
        User.create(UserData, (err, savedData) => {
            if (err) {
                res.status(400).send({ success: false, err: err });
            } else {
                let token = jwt.sign({ savedData }, secret, {
                    expiresIn: '4h'
                });
                res.status(201).send({ success: true, msg: `User ${savedData.user
Name} created.`, token })
            }
        });


    } else {
        return res.status(400).send({
            success: false,
            msg: "userName, password,email,mobileNumber must be provided.",
            "userName": data.userName ? "Valid" : "Not Provided",
            "password": data.password ? "Valid" : "Not Provided",
            "email": data.email ? "Valid" : "Not Provided",
            "mobileNumber": data.mobileNumber ? "Valid" : "Not Provided",
        })
    }

});


//Login using admin credentials
//HTTP POST request
//Username and password must be provided
//if not error 400  Bad Request will be returned
//if the credentials are valid and a user exists in the user database
//a Authorization token will bre provided
router.post('/login', (req, res) => {
    const data = req.body;
    if (data.userName && data.password) {
        User.findOne({$or:[{userName:data.userName},{email:data.userName}],passwo
rd:data.password}, (err, dataFromDB) => {
            if (err) {
                res.status(404).send({ success: false, err: err });
            } else {
                if (dataFromDB) {
                    let token = jwt.sign({ dataFromDB }, secret, {
                        expiresIn: '4h'
                    });
                    res.status(200).send({ success: true, msg: `User Name ${dataF
romDB.userName} Logged in Successfully.`, token });
                } else {
                    res.status(404).send({ success: false, msg: "Not Found" });
                }
            }
        });

    } else {
        res.status(400).send({
            success: false,
            msg: "userName, password must be provided.",
            "userName": data.userName ? "Provided" : "Not Provided",
            "password": data.password ? "Provided" : "Not Provided"
        })
    }
})


module.exports = router;
```

Web Interface

Fire Alarm CSS

```css
p{
    margin-bottom: 0;
}
.card-text{
    margin-bottom: 1rem;
}

.green-card{
    background-color: rgb(65, 170, 65);
}
.red-card{
    background-color: rgb(170, 65, 65);
}
.yello-card{
    background-color: rgb(168, 170, 65);
}

.not-active-card{
    background-color: rgb(192, 169, 166);
    text-decoration: line-through;
}
```

Fire Alarm Card Component

```javascript
//Importing React and React.Component
import React, { Component } from 'react';
//StyleSheet required to fire alarm card
import './fireAlarm.css';


class fireAlarmCard extends Component {

    //getting the card classnames according to the co2 and smoke levels
    getClass = () => {
        if (this.props.data.status === 'Not Active') {
            return (
                'card not-active-card'
            )
        } else if (this.props.data.co2Level < 5 && this.props.data.smokeLevel < 5
) {
            return (
                'card green-card'
            );
        } else if ((this.props.data.smokeLevel < 5 && this.props.data.co2Level ==
= 5) || (this.props.data.smokeLevel === 5 && this.props.data.co2Level < 5)) {
            return (
                'card yello-card'
            );
        } else if (this.props.data.co2Level > 5 || this.props.data.smokeLevel > 5
) {
            return (
                'card red-card'
            );
        }
    }


    render() {
        let lastUpdated = new Date(this.props.data.lastUpdated);
        lastUpdated = lastUpdated.toLocaleString();
        return (
            <div className="col-sm-4 py-2 ">
                <div className={this.getClass()}>
                    <div className="card-body ">
                        <div className='card-text' >
                            <p>Status       : {this.props.data.status}</p>
                            <p>Room No      : {this.props.data.roomNo}</p>
                            <p>Floor No     : {this.props.data.floorNo}</p>
                            <p>CO2 Level    : {this.props.data.co2Level}</p>
                            <p>Smoke Level  : {this.props.data.smokeLevel}</p>
                            <p>Last Updated : {lastUpdated}</p>
                        </div>
                    </div>
                </div>
            </div>
        );
    }
}

export default fireAlarmCard;
```

Fire Alarm Main Component

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

import FireAlarmCard from './fireAlarmCard';

import axios from 'axios';
//Importing sweet alert and sweet alert dark theme
import Swal from 'sweetalert2';
import '@sweetalert2/theme-dark/dark.css';


class fireAlarmMain extends Component {
    constructor(props) {
        super(props);
        this.state = {
            data: [],
        }
    }

    getData = () => {
        //Getting the FireAlarm data from the REST Api
        axios.get('http://localhost:5000/api/firealarm/')
            .then((res) => {
                if (res.data.success) {
                    this.setState({
                        data: res.data.data,
                    });
                } else {
                    //Showing a error dialog upon errors
                    Swal.fire({
                        icon: 'error',
                        title: 'Oops...',
                        text: res.data.msg,
                    })
                }
            })
            .catch(err => {
                Swal.fire({
                    icon: 'error',
                    title: 'Oops...',
                    text: err
                })
            });
    }


    //Refresing the webSite data in 40 second intervals
    componentDidMount() {
        this.getData();
        this.interval = setInterval(()=>this.getData(), 40*1000);
    }
    //Clearing the interval is the Component get Unmounted
    //This will avoid possible data leaks and memory use errors
    componentWillUnmount() {
        clearInterval(this.interval);
    }

    render() {
        return (
            <div className='container'>
                <div className="row">
                    {this.state.data.map((data, index, self) =>
                        <FireAlarmCard data={data} key={index} />
                    )}
                </div>
            </div>
        );
    }
}

export default fireAlarmMain;
```

Header CSS

```
#header-nav-bar{
    background-color: #111111;
}
.navbar-title{
    color: orangered;
    text-align: center;

}
.nav-link{
    text-decoration: none;
    color: gold;
}
```

Header Component

```
import React, { Component } from 'react';
import './header.css';

class header extends Component {
    render() {
        return (
            <div>
                <nav id="header-nav-bar" className="navbar ">
                    <h1 className="navbar-title">Fire Alarm System</h1>
                </nav>
            </div>
        );
    }
}

export default header;
```

Desktop Client

```
clientMain = new FireAlarmClientMain();
                                 clientMain.frame.setVisible(true);

                                 // Getting the Remote Objects from the RMI Registry
                                 int PORT = 3000;
                                 String fireAlarmRegistration = "//localhost:" + PORT
+ "/FireAlarm";
                                 String userRegistration = "//localhost:" + PORT +
"/User";

                                 Remote fireAlarmService =
Naming.lookup(fireAlarmRegistration);
                                 fireAlarm = (FireAlarmSensor) fireAlarmService;

                                 Remote userService =
Naming.lookup(userRegistration);
                                 user = (User) userService;

                                 // Registering the Client Application in the RMI
Server in order to get
                                 // alerts when a fire alarm sensor exceeds the
warning limit

      fireAlarm.registerFireAlarmDesktopClient(clientMain);

lblLogin.addMouseListener(new MouseAdapter() {
                    @Override
                    public void mouseEntered(MouseEvent e) {
                          // Changing the Color On mouse hover

      lblLogin.setForeground(Color.getHSBColor(loginRegeisterMouseFloatColorHSB[0],
                                       loginRegeisterMouseFloatColorHSB[1],
loginRegeisterMouseFloatColorHSB[2]));
                    }

                    @Override
                    public void mouseExited(MouseEvent e) {

      lblLogin.setForeground(Color.getHSBColor(loginRegeisterColorHSB[0],
loginRegeisterColorHSB[1],
                                       loginRegeisterColorHSB[2]));
                    }

                    @Override
                    public void mouseClicked(MouseEvent e) {
                          UserLogin.executeMain(user);
                    }
             });
lblRegister.addMouseListener(new MouseAdapter() {
                    @Override
                    public void mouseEntered(MouseEvent e) {

        lblRegister.setForeground(Color.getHSBColor(loginRegeisterMouseFloatColorHSB[0
],
                                        loginRegeisterMouseFloatColorHSB[1],
loginRegeisterMouseFloatColorHSB[2]));
                    }

                    @Override
                    public void mouseExited(MouseEvent e) {

        lblRegister.setForeground(Color.getHSBColor(loginRegeisterColorHSB[0],
loginRegeisterColorHSB[1],
                                        loginRegeisterColorHSB[2]));
                    }

                    @Override
                    public void mouseClicked(MouseEvent e) {
                          UserRegistration.executeMain(user);
                    }
             });
btnEditFireAlarm.addActionListener(new ActionListener() {
                    public void actionPerformed(ActionEvent e) {
                          if (table_1.getSelectedRow() != -1) {
                                int row = table_1.getSelectedRow();
                                EditFireAlarm.executeMain(fireAlarm, UserDTO,
(String) table_1.getValueAt(row, 0),
                                            (Integer.valueOf((String)
table_1.getValueAt(row, 2))),
                                            (Integer.valueOf((String)
table_1.getValueAt(row, 3))));
                          } else {
                                JOptionPane.showMessageDialog(frame, "Select a Row
to Edit.", "Error", JOptionPane.ERROR_MESSAGE);

                          }

                    }
btnDeleteFireAlarm.addActionListener(new ActionListener() {
                    public void actionPerformed(ActionEvent e) {
                          if (table_1.getSelectedRow() != -1) {
                                int row = table_1.getSelectedRow();
                                try {

                                      if (JOptionPane.showConfirmDialog(frame, "Do
you want to delete this Fire Alarm Sensor ?",
                                                  "Confirm",
JOptionPane.WARNING_MESSAGE) == JOptionPane.YES_OPTION) {
                                            FireAlarmDTO deleteDTO =
fireAlarm.deleteFireAlarmDetails((String) table_1.getValueAt(row, 0),
                                                        UserDTO.getToken());
                                            if (deleteDTO.isSuccess()) {

        JOptionPane.showMessageDialog(frame, deleteDTO.getMsg(), "Deleted",
	JOptionPane.ERROR_MESSAGE);
						setTime(0);
					} else {
						if (deleteDTO.getMsg() != null)
{

	JOptionPane.showMessageDialog(frame, "Error Delete. Error - " +
deleteDTO.getMsg(),
								"Error",
JOptionPane.ERROR_MESSAGE);
						} else {

	JOptionPane.showMessageDialog(frame, "Error Delete", "Error",

	JOptionPane.ERROR_MESSAGE);
						}
					}
				}
			} catch (RemoteException e1) {
				JOptionPane.showMessageDialog(frame, "Error
Delete", "Error", JOptionPane.ERROR_MESSAGE);
				e1.printStackTrace();
			} catch (IOException e1) {
				JOptionPane.showMessageDialog(frame, "Error
Delete", "Error", JOptionPane.ERROR_MESSAGE);
				e1.printStackTrace();
			}catch (Exception e1) {
				JOptionPane.showMessageDialog(frame, "Error
Delete", "Error", JOptionPane.ERROR_MESSAGE);
				e1.printStackTrace();
			}
		} else {
			JOptionPane.showMessageDialog(frame, "Select a Row
to Delete.", "Error", JOptionPane.ERROR_MESSAGE);

		}
	}
	});
try {
						// Emptying the Table data
						tdm.setRowCount(0);
						// Adding the new Data to the
Table
						for (FireAlarmDTO fdto :
fireAlarm.getFireSensorDetails()) {
							Object[] obj = {
fdto.getId(), fdto.getStatus(), fdto.getRoomNo(),

	fdto.getFloorNo(), fdto.getSmokeLevel(), fdto.getCo2Level(),

	fdto.getLastUpdated() };
							tdm.addRow(obj);
						}
					} catch (RemoteException e1) {
```

```
UserDTO = user;
             btnAddNewFire.setEnabled(true);
             btnEditFireAlarm.setEnabled(true);
             lblLogin.setVisible(false);
             lblRegister.setVisible(false);   btnDeleteFireAlarm.setEnabled(true);

Override
      public void showAlert(FireAlarmDTO fireDto) throws RemoteException {
             JOptionPane.showMessageDialog(frame, "Room No " + fireDto.getRoomNo() +
", Floor No " + fireDto.getFloorNo()
                          + " has exceeded CO2 or Smoke Level Limit", "Alert",
JOptionPane.WARNING_MESSAGE);
             setTime(0);

      }
float backgroudnColorHSBGreen[] = Color.RGBtoHSB(65, 170, 65, null);
             float backgroudnColorHSBYello[] = Color.RGBtoHSB(168, 170, 65, null);
             float backgroudnColorHSBRed[] = Color.RGBtoHSB(170, 65, 65, null);
             float backgroudnColorHSBPink[] = Color.RGBtoHSB(192, 169, 166, null);
btnAddFireAlarm.addActionListener(new ActionListener() {
                   public void actionPerformed(ActionEvent e) {
                          if(!txtRoomNo.getText().equals("")) {
                                 if(!txtFloorNo.getText().equals("")) {
                                       try {
                                              int room =
Integer.parseInt(txtRoomNo.getText());
                                              int floor =
Integer.parseInt(txtFloorNo.getText());
                                              if(user.getToken()!=null) {
                                                     try {
                                                            FireAlarmDTO fireaddedDTO
= fire.addFireAlarm(user.getToken(), room, floor);

      if(fireaddedDTO.isSuccess()) {

      lblError.setText("Fire Alarm Added Successfully");

      txtFloorNo.setText("");

      txtRoomNo.setText("");

      FireAlarmClientMain.setTime(0);
                                                            }else {

      if(fireaddedDTO.getMsg()!=null) {

      lblError.setText("Error - "+fireaddedDTO.getMsg());
                                                                   }else {

      lblError.setText("Error Adding Fire Alarm");
                                                                   }
                                                            }
                                                     } catch (RemoteException e1) {
                                                            lblError.setText("Error
Adding Fire Alarm");
```

```
                                                                e1.printStackTrace();
                                                         } catch (IOException e1) {
                                                                lblError.setText("Error
Adding Fire Alarm");

                                                                e1.printStackTrace();
                                                         } catch (Exception e1) {
                                                                lblError.setText("Error
Adding Fire Alarm");

                                                                e1.printStackTrace();
                                                         }
                                                  }else {
                                                         lblError.setText("Admin Access
Required");
                                                  }
                                           }catch ( NumberFormatException ex){
                                                  lblError.setText("Room No and Floor No
must be Numbers");
                                           }
                                    }else {
                                           lblError.setText("Floor No is invalid");
                                    }
                             }else {
                                    lblError.setText("Room No is invalid");
                             }
                      }
               });
btnEditFireAlarm.addActionListener(new ActionListener() {
                      public void actionPerformed(ActionEvent e) {
                             if (!txtRoomNo.getText().equals("")) {
                                    if (!txtFloorNo.getText().equals("")) {
                                           try {
                                                  int room =
Integer.parseInt(txtRoomNo.getText());
                                                  int floor =
Integer.parseInt(txtFloorNo.getText());
                                                  if (user.getToken() != null) {
                                                         try {
                                                                FireAlarmDTO fireaddedDTO
= fire.editFireAlarmDetails(id, user.getToken(), room,
                                                                              floor);
                                                                if
(fireaddedDTO.isSuccess()) {

       lblError.setText("Fire Alarm Edited SUccessfully");

       txtFloorNo.setText("");

       txtRoomNo.setText("");
                                                                       dispose();

       FireAlarmClientMain.setTime(0);
                                                                } else {
                                                                       if
(fireaddedDTO.getMsg() != null) {
        lblError.setText("Error - " + fireaddedDTO.getMsg());
                                                            } else {

        lblError.setText("Error Editing Fire Alarm");
                                                            }
                                                        }
                                                } catch (RemoteException e1) {
                                                        lblError.setText("Error
Editing Fire Alarm");
                                                        e1.printStackTrace();
                                                } catch (IOException e1) {
                                                        lblError.setText("Error
Editing Fire Alarm");
                                                        e1.printStackTrace();
                                                } catch (Exception e1) {
                                                        lblError.setText("Error
Editing Fire Alarm");
                                                        e1.printStackTrace();
                                                }
                                        } else {
                                                lblError.setText("Admin Access
Required");
                                        }
                                } catch (NumberFormatException ex) {
                                        lblError.setText("Room No and Floor No
must be Numbers");
                                }
                        } else {
                                lblError.setText("Floor No is invalid");
                        }
                } else {
                        lblError.setText("Room No is invalid");
                }
        }
    });
public interface FireAlarmDesktop extends Remote{


        public void showAlert(FireAlarmDTO fireDto) throws RemoteException;

}

public class FireAlarmDTO implements Serializable {
        /**
         *
         */
        private static final long serialVersionUID = 1620796167638083179L;
        private String _id;
        private String floorNo;
        private String roomNo;
        private String lastUpdated;
        private String status;
        private int co2Level;
        private int smokeLevel;
```

```
	private boolean success;
	private String msg;
	private int statusCode;

	private List<FireAlarmDTO> data;

	private FireAlarmDTO savedData;

	public String getId() {
		return _id;
	}

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

	public String getFloorNo() {
		return floorNo;
	}

	public void setFloorNo(String floorNo) {
		this.floorNo = floorNo;
	}

	public String getRoomNo() {
		return roomNo;
	}

	public void setRoomNo(String roomNo) {
		this.roomNo = roomNo;
	}

	public String getLastUpdated() {
		return lastUpdated;
	}

	public void setLastUpdated(String lastUpdated) {
		this.lastUpdated = lastUpdated;
	}

	public String getStatus() {
		return status;
	}

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

	public int getCo2Level() {
		return co2Level;
	}

	public void setCo2Level(int co2Level) {
		this.co2Level = co2Level;
	}

	public int getSmokeLevel() {
		return smokeLevel;
	}

	public void setSmokeLevel(int smokeLevel) {
		this.smokeLevel = smokeLevel;
	}

	public boolean isSuccess() {
		return success;
	}

	public void setSuccess(boolean success) {
		this.success = success;
	}

	public String getMsg() {
		return msg;
	}

	public void setMsg(String msg) {
		this.msg = msg;
	}

	public int getStatusCode() {
		return statusCode;
	}

	public void setStatusCode(int statusCode) {
		this.statusCode = statusCode;
	}

	public List<FireAlarmDTO> getData() {
		return data;
	}

	public void setData(List<FireAlarmDTO> data) {
		this.data = data;
	}


	public FireAlarmDTO getSavedData() {
		return savedData;
	}

	public void setSavedData(FireAlarmDTO savedData) {
		this.savedData = savedData;
	}



}
```

```
public interface FireAlarmSensor extends java.rmi.Remote{

	public ArrayList<FireAlarmDTO> getFireSensorDetails() throws
java.rmi.RemoteException , IOException;

	public FireAlarmDTO addFireAlarm(String token,int roomNo,int floorNo) throws
java.rmi.RemoteException , IOException;

	public void registerFireAlarmDesktopClient(FireAlarmDesktop fireAlarm) throws
java.rmi.RemoteException , IOException;

	public void unregisterFireAlarmDesktopClient(FireAlarmDesktop fireAlarm)
throws java.rmi.RemoteException , IOException;

	public FireAlarmDTO editFireAlarmDetails(String id,String token,int roomNo,int
floorNo) throws java.rmi.RemoteException , IOException;

	public FireAlarmDTO deleteFireAlarmDetails(String id,String token) throws
java.rmi.RemoteException , IOException;

}
btnLogin.addActionListener(new ActionListener() {
			public void actionPerformed(ActionEvent e) {
				if (!txtUsername.getText().equals("")) {
					if (!txtPassword.getText().equals("")) {
						UserDTO usdto = new UserDTO();
						usdto.setUserName(txtUsername.getText());
						usdto.setPassword(txtPassword.getText());
						try {
							UserDTO loggedInDTO =
user.login(usdto);
							if (loggedInDTO.getSuccess()) {
								lblError.setText("Logged In
Successfully");

	FireAlarmClientMain.registerLogin(loggedInDTO);
								dispose();
							} else {
								if (loggedInDTO.getMsg() !=
null) {

									lblError.setText("Error -
" + loggedInDTO.getMsg());
								} else {
									lblError.setText("Error");
								}
							}
						} catch (RemoteException e1) {
							lblError.setText("Error Login");
							e1.printStackTrace();
						} catch (IOException e1) {
							lblError.setText("Error Login");
							e1.printStackTrace();
						} catch (Exception e1) {
							lblError.setText("Error Login");
							e1.printStackTrace();
```

```
                                        }

                                } else {
                                        lblError.setText("Password is invalid");
                                }
                        } else {
                                lblError.setText("Username is invalid");
                        }
                }
        });
btnSignup.addActionListener(new ActionListener() {
                public void actionPerformed(ActionEvent e) {
                        if (!txtUsername.getText().equals("")) {
                                if (!txtEmail.getText().equals("")) {
                                        if (!txtPassword.getText().equals("")) {
                                                if
(!txtConfirmpassword.getText().equals("")) {
                                                        if
(!txtMobile.getText().equals("")) {
                                                                if
(txtPassword.getText().equals(txtConfirmpassword.getText())) {
                                                                        try {
                                                                                UserDTO usd =
new UserDTO();

        usd.setUserName(txtUsername.getText());

        usd.setEmail(txtEmail.getText());

        usd.setPassword(txtPassword.getText());

        usd.setMobileNumber(String.valueOf(Integer.parseInt(txtMobile.getText())));
                                                                                try {

        UserDTO registerDTO = user.register(usd);
                                                                                        if
(registerDTO.getSuccess()) {

        lblError.setText("User Registered");

        FireAlarmClientMain.registerLogin(registerDTO);

        dispose();
                                                                                        } else
{

        if (registerDTO.getMsg() != null) {

        lblError.setText("Error - " + registerDTO.getMsg());

        } else {

        lblError.setText("Error Registration");

        }
```

```
                                                                                   }
                                                                             } catch
(RemoteException e1) {

      lblError.setText("Error Registration");

      e1.printStackTrace();
                                                                             } catch
(IOException e1) {

      lblError.setText("Error Registration");

      e1.printStackTrace();

      }catch(Exception e1) {

      lblError.setText("Error Registration");

      e1.printStackTrace();

                                                                             }
                                                                       } catch
(NumberFormatException ex) {

      lblError.setText("Mobile Number must be a Number");

      ex.printStackTrace();
                                                                       } catch(Exception
ex1) {

      lblError.setText("Error Registration");

      ex1.printStackTrace();
                                                                       }
                                                                 } else {

      lblError.setText("Password is mismatch");
                                                                 }
                                                           } else {
                                                                 lblError.setText("Mobile
Number is invalid");
                                                           }
                                                     } else {
                                                           lblError.setText("Password is
mismatch");
                                                     }
                                               } else {
                                                     lblError.setText("Password is
invalid");
                                               }
                                         } else {
                                               lblError.setText("Email is invalid");
                                         }
                                   } else {
                                         lblError.setText("Username is invalid");
                                   }
```

```
                    }
            });

public interface User extends java.rmi.Remote{

        public UserDTO login(UserDTO user) throws
java.rmi.RemoteException,IOException;

        public UserDTO register(UserDTO user) throws
java.rmi.RemoteException,IOException;

}
public class UserDTO implements Serializable{

        /**
         *
         */
        private static final long serialVersionUID = -1464002385204833616L;
        @SerializedName("success")
        @Expose
        private Boolean success;
        @SerializedName("msg")
        @Expose
        private String msg;
        @SerializedName("token")
        @Expose
        private String token;
        @SerializedName("userName")
        @Expose
        private String userName;
        @SerializedName("password")
        @Expose
        private String password;
        @SerializedName("email")
        @Expose
        private String email;
        @SerializedName("mobileNumber")
        @Expose
        private String mobileNumber;

        public Boolean getSuccess() {
                return success;
        }

        public void setSuccess(Boolean success) {
                this.success = success;
        }

        public String getMsg() {
                return msg;
        }

        public void setMsg(String msg) {
                this.msg = msg;
        }
```

```
	public String getToken() {
		return token;
	}

	public void setToken(String token) {
		this.token = token;
	}

	public String getUserName() {
		return userName;
	}

	public void setUserName(String userName) {
		this.userName = userName;
	}

	public String getPassword() {
		return password;
	}

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

	public String getEmail() {
		return email;
	}

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

	public String getMobileNumber() {
		return mobileNumber;
	}

	public void setMobileNumber(String mobileNumber) {
		this.mobileNumber = mobileNumber;
	}

}

public class FireAlarmSensorImpl extends UnicastRemoteObject implements
FireAlarmSensor {

	/**
	 * This is where the RMI Server functions are implemented
	 *
	 */
	private static final long serialVersionUID = 1L;

	// Store the RMI Client list to send alerts
	private static volatile List<FireAlarmDesktop> fireAlaramClientList = new
ArrayList<FireAlarmDesktop>();
```

```
	public FireAlarmSensorImpl() throws RemoteException {
	}

	// This method is used to get all the fire alarm sensor details
	// It exports a Fire Alarm DTO array
	@Override
	public ArrayList<FireAlarmDTO> getFireSensorDetails() throws RemoteException, 
IOException {
		// Setting the REST Api URL whichis used to do the GET request
		URL url = new URL("http://localhost:5000/api/firealarm/");
		HttpURLConnection con = (HttpURLConnection) url.openConnection();
		con.setRequestMethod("GET");
		// Setting the Request header to accept response in JSON Format
		con.setRequestProperty("Accept", "application/json");

		// getting the response status code
		int responseCode = con.getResponseCode();

		Reader br = null;
		// checking the response is a success or an error
		// Reading the success or error response
		if (responseCode >= 200 && responseCode <= 299) {
			br = new BufferedReader(new 
InputStreamReader(con.getInputStream(), "utf-8"));
		} else {
			br = new BufferedReader(new 
InputStreamReader(con.getErrorStream(), "utf-8"));
		}

		// parsing the JSON response to a Java Object
		Gson gson = new Gson();
		try {
			FireAlarmDTO fireAlarmDTO = gson.fromJson(br, 
FireAlarmDTO.class);
			if (fireAlarmDTO != null) {

				return (ArrayList<FireAlarmDTO>) fireAlarmDTO.getData();

			}
		} catch (Exception e) {
			System.out.println(e);
		}
		return null;
	}

	// This method is used to add a new fire alarm
	// It returns the success status and success message if success
	// or a fail message if failed
	@Override
	public FireAlarmDTO addFireAlarm(String token, int roomNo, int floorNo) throws 
RemoteException, IOException {
		// Setting the URL to post request to add the new fire alarm
		URL url = new URL("http://localhost:5000/api/firealarm/addFireAlarm");
		// Opening a Connection
		HttpURLConnection con = (HttpURLConnection) url.openConnection();
		// Setting the Request Method
		con.setRequestMethod("POST");
		// Setting the Request Content Type
		con.setRequestProperty("Content-Type", "application/json; charset=UTF-
8");
		// Set the Authorization header
		con.setRequestProperty("Authorization", "Bearer " + token);
		// Setting the Request header to accept response in JSON Format
		con.setRequestProperty("Accept", "application/json");
		// Enabling the writing to the connection output stream
		con.setDoOutput(true);
		// Creating the Data in the request body and writing it to output stream
		String data = "{\r\n" + "    \"roomNo\": \"" + roomNo + "\",\r\n" + "
\"floorNo\": \"" + floorNo + "\"\r\n"
				+ "}";
		OutputStream os = con.getOutputStream();
		byte[] input = data.getBytes("utf-8");
		os.write(input, 0, input.length);

		// getting the response status code
		int responseCode = con.getResponseCode();
		// Reading the response
		Reader br = null;
		// checking the response is a success or an error
		if (responseCode >= 200 && responseCode <= 299) {
			br = new BufferedReader(new
InputStreamReader(con.getInputStream(), "utf-8"));
		} else {
			br = new BufferedReader(new
InputStreamReader(con.getErrorStream(), "utf-8"));
		}

		// parsing the JSON response to a Java Object
		Gson gson = new Gson();
		FireAlarmDTO fireAlarmDTO = gson.fromJson(br, FireAlarmDTO.class);
		if (fireAlarmDTO != null) {
			return fireAlarmDTO;
		}

		return null;

	}

	// This method is used to register a new desktop client
	@Override
	public void registerFireAlarmDesktopClient(FireAlarmDesktop fireAlarm) throws
RemoteException, IOException {
		fireAlaramClientList.add(fireAlarm);
	}

	// THis method is used to unregister a desktop client
	@Override
	public void unregisterFireAlarmDesktopClient(FireAlarmDesktop fireAlarm)
throws RemoteException, IOException {
		fireAlaramClientList.remove(fireAlarm);
	}

	// This method is used to edit a fire alarm details
	// It uses PATCH HTTP request type
	// Since it is not provided in HTTP URL Connection
	// THis method uses a way around the problem
	// Using this prints a warning but the PATCH request works
	// Method returns a Success status and a message
	@Override
	public FireAlarmDTO editFireAlarmDetails(String id, String token, int roomNo, 
int floorNo)
			throws RemoteException, IOException {

		// Setting the URL to Make the PATCH request
		URL url = new URL("http://localhost:5000/api/firealarm/update/" + id);
		// Opening a Connection
		HttpURLConnection con = (HttpURLConnection) url.openConnection();
		// Setting the Request Method
		// Since PATCH is not available in HttpURLConnection this is a 
workaround
		allowMethods("PATCH");
		con.setRequestMethod("PATCH");
		// Setting the Request Content Type
		con.setRequestProperty("Content-Type", "application/json; charset=UTF-
8");
		// Setting the Authorization header
		con.setRequestProperty("Authorization", "Bearer " + token);
		// Setting the Request header to accept response in JSON Format
		con.setRequestProperty("Accept", "application/json");
		// Enabling the writing to the connection output stream
		con.setDoOutput(true);
		// Creating the Data in the request body and writing it to output stream
		String data = "{\r\n" + "    \"roomNo\": \"" + roomNo + "\",\r\n" + "    
\"floorNo\": \"" + floorNo + "\"\r\n"
				+ "}";
		OutputStream os = con.getOutputStream();
		byte[] input = data.getBytes("utf-8");
		os.write(input, 0, input.length);

		// getting the response status code
		int responseCode = con.getResponseCode();
		// Reading the response
		Reader br = null;
		// checking the response is a success or an error
		if (responseCode >= 200 && responseCode <= 299) {
			br = new BufferedReader(new 
InputStreamReader(con.getInputStream(), "utf-8"));
		} else {
			br = new BufferedReader(new 
InputStreamReader(con.getErrorStream(), "utf-8"));
		}

		// parsing the JSON response to a Java Object
		Gson gson = new Gson();
```

```
			FireAlarmDTO fireAlarmDTO = gson.fromJson(br, FireAlarmDTO.class);
			if (fireAlarmDTO != null) {
				return fireAlarmDTO;
			}

			return null;
	}

	// This method is to delete FIre Alarm Sensors
	// This method returns a success status and a message
	@Override
	public FireAlarmDTO deleteFireAlarmDetails(String id, String token) throws
RemoteException, IOException {
			// Setting the DELETE HTTP Request REST API URL
			URL url = new URL("http://localhost:5000/api/firealarm/" + id);
			HttpURLConnection con = (HttpURLConnection) url.openConnection();
			con.setRequestMethod("DELETE");
			// Setting the Request header to accept response in JSON Format
			con.setRequestProperty("Accept", "application/json");
			// Setting the Authorization token
			con.setRequestProperty("Authorization", "Bearer " + token);
			// getting the response status code
			int responseCode = con.getResponseCode();

			// Reading the response
			Reader br = null;
			// checking the response is a success or an error
			if (responseCode >= 200 && responseCode <= 299) {
				br = new BufferedReader(new
InputStreamReader(con.getInputStream(), "utf-8"));
			} else {
				br = new BufferedReader(new
InputStreamReader(con.getErrorStream(), "utf-8"));
			}

			// parsing the JSON response to a Java Object
			Gson gson = new Gson();
			try {
				FireAlarmDTO fireAlarmDTO = gson.fromJson(br,
FireAlarmDTO.class);
				if (fireAlarmDTO != null) {
					return fireAlarmDTO;
				}
			} catch (Exception e) {
				System.out.println(e);
			}
			return null;
	}

	//This method is periodically run by the RMI Server Main method to check for
fire alarm smoke and CO2 levels
	//If one or many fire alarms exceeds the CO2 or smoke level
	//this method runs the notify all listeners method which shows an alert in the
desktop client
```

```
	//This method only invokes the listeners if the fire alarm sensors status is 
active
	public void checkFireAlarmSensors() throws IOException {
		// getting all fireAlarmDetails
		URL url = new URL("http://localhost:5000/api/firealarm/");
		HttpURLConnection con = (HttpURLConnection) url.openConnection();
		con.setRequestMethod("GET");
		// Setting the Request header to accept response in JSON Format
		con.setRequestProperty("Accept", "application/json");

		// getting the response status code
		int responseCode = con.getResponseCode();
		// Reading the response
		Reader br = null;
		// checking the response is a success or an error
		if (responseCode >= 200 && responseCode <= 299) {
			br = new BufferedReader(new 
InputStreamReader(con.getInputStream(), "utf-8"));
		} else {
			br = new BufferedReader(new 
InputStreamReader(con.getErrorStream(), "utf-8"));
		}

		// parsing the JSON response to a Java Object
		Gson gson = new Gson();
		try {
			FireAlarmDTO fireAlarmDTO = gson.fromJson(br, 
FireAlarmDTO.class);
			if (fireAlarmDTO != null) {
				for (FireAlarmDTO fdt : fireAlarmDTO.getData()) {
					if (fdt.getSmokeLevel() > 5 || fdt.getCo2Level() > 
5) {

						if (fdt.getStatus().equals("Active")) {
							notifyAllListners(fdt);
						}
					}
				}

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

	//This method is used to access the remote method in Desktop clients to 
	//show the alert
	//This method is invoked by the checkFireAlarmSensors Method
	public void notifyAllListners(FireAlarmDTO fireDto) {
		for (FireAlarmDesktop fireAlarmDesktop : fireAlaramClientList) {
			try {
				fireAlarmDesktop.showAlert(fireDto);
			} catch (RemoteException e) {
				e.printStackTrace();
			}
		}
		}
}

//Registering In RMI registry
			int PORT = 3000;
			LocateRegistry.createRegistry(PORT);
			String registry = "localhost:" + PORT;

			String fireAlarmRegistration = "rmi://" + registry + 
"/FireAlarm";
			String userRegistration = "rmi://" + registry + "/User";

			Naming.rebind(fireAlarmRegistration, sensor);
			Naming.rebind(userRegistration, user);

			System.out.println("RMI Server Running");
			//Used to check the fire alarm sensor details in intervals
			int time = 15000;
			checkFireAlarms(time);
			System.out.println("Checking for Fire Alarm CO2, Smoke Level 
Exceeds in Every " + time + "ms");
private static FireAlarmSensorImpl sensor ;
	public static void checkFireAlarms(int time) {
		try {
			sensor = new FireAlarmSensorImpl();
		} catch (RemoteException e1) {
			// TODO Auto-generated catch block
			e1.printStackTrace();
		}
		Thread th = new Thread(new Runnable() {
			@Override
			public void run() {

				for (;;) {
					try {
						sensor.checkFireAlarmSensors();
						Thread.sleep(time);
					} catch (IOException e) {
						e.printStackTrace();
					} catch (InterruptedException e) {
						e.printStackTrace();
					}
				}

			}
		});

		th.start();
	}
public class UserImpl extends UnicastRemoteObject implements User {

	/**
	 * This is where the RMI Server user interface implemented
	 */
	private static final long serialVersionUID = 1L;
	public UserImpl() throws RemoteException {

	}

	//This method is used to register a client
	//When success it returns a user DTO with the success status,message and the 
authorization token
	//When failure it returns a user DTO with the success status and a message
	@Override
	public UserDTO register(UserDTO user) throws RemoteException, IOException {
		// Setting the REST API URL to register a client
		URL url = new URL("http://localhost:5000/api/users/signup");
		// Opening a Connection
		HttpURLConnection con = (HttpURLConnection) url.openConnection();
		// Setting the Request Method
		con.setRequestMethod("POST");
		// Setting the Request Content Type
		con.setRequestProperty("Content-Type", "application/json; charset=UTF-
8");
		// Setting the Request header to accept response in JSON Format
		con.setRequestProperty("Accept", "application/json");
		// Enabling the writing to the connection output stream
		con.setDoOutput(true);
		// Creating the Data in the request body and writing it to output stream
		String data = "{\r\n" + "    \"userName\": \"" + user.getUserName() + 
"\",\r\n" + "    \"password\": \""
				+ user.getPassword() + "\",\r\n" + "    \"email\": \"" + 
user.getEmail() + "\",\r\n"
				+ "    \"mobileNumber\": \"" + user.getMobileNumber() + 
"\"\r\n" + "}";
		OutputStream os = con.getOutputStream();
		byte[] input = data.getBytes("utf-8");
		os.write(input, 0, input.length);

		// getting the response status code
		int responseCode = con.getResponseCode();
		// Reading the response
		Reader br = null;
		// checking the response is a success or an error
		if (responseCode >= 200 && responseCode <= 299) {
			br = new BufferedReader(new 
InputStreamReader(con.getInputStream(), "utf-8"));
		} else {
			br = new BufferedReader(new 
InputStreamReader(con.getErrorStream(), "utf-8"));
		}

		// parsing the JSON response to a Java Object
		Gson gson = new Gson();
		UserDTO userDTO = gson.fromJson(br, UserDTO.class);
		if (userDTO != null) {
			return userDTO;
		}
		return null;
	}


	//This method is used to login a client
	//When success it returns a user DTO with the success status,message and the 
authorization token
	//When failure it returns a user DTO with the success status and a message
	@Override
	public UserDTO login(UserDTO user) throws RemoteException, IOException {
		// Setting the REST API URL to login
		URL url = new URL("http://localhost:5000/api/users/login");
		// Opening a Connection
		HttpURLConnection con = (HttpURLConnection) url.openConnection();
		// Setting the Request Method
		con.setRequestMethod("POST");
		// Setting the Request Content Type
		con.setRequestProperty("Content-Type", "application/json; charset=UTF-
8");
		// Setting the Request header to accept response in JSON Format
		con.setRequestProperty("Accept", "application/json");
		// Enabling the writing to the connection output stream
		con.setDoOutput(true);
		// Creating the Data in the request body and writing it to output stream
		String data = "{\r\n" +
				"    \"userName\": \""+user.getUserName()+"\",\r\n" +
				"    \"password\": \""+user.getPassword()+"\"\r\n" +
				"}";
		OutputStream os = con.getOutputStream();
		byte[] input = data.getBytes("utf-8");
		os.write(input, 0, input.length);

		// getting the response status code
		int responseCode = con.getResponseCode();
		// Reading the response
		Reader br = null;
		// checking the response is a success or an error
		if (responseCode >= 200 && responseCode <= 299) {
			br = new BufferedReader(new 
InputStreamReader(con.getInputStream(), "utf-8"));
		} else {
			br = new BufferedReader(new 
InputStreamReader(con.getErrorStream(), "utf-8"));
		}

		// parsing the JSON response to a Java Object
		Gson gson = new Gson();
		UserDTO userDTO = gson.fromJson(br, UserDTO.class);
		if (userDTO != null) {
			return userDTO;
		}

		return null;
	}
```

```
}

chckbxStatus.addActionListener(new ActionListener() {
                    public void actionPerformed(ActionEvent e) {
                          boolean checked = chckbxStatus.isSelected();
                          String status = null;
                          if (checked) {
                                status = "Active";
                          } else {
                                status = "Not Active";
                          }
                          if (comboBox.getSelectedItem() != null) {
                                String id = (String) comboBox.getSelectedItem();
                                try {
                                      FireAlarmDTO fad =
fireAlarm.updateFireAlarmData(id, status);
                                      if (fad.isSuccess()) {
                                            JOptionPane.showMessageDialog(frame,
"Status Updated", "Updated",

      JOptionPane.INFORMATION_MESSAGE);
                                      } else {

                                            JOptionPane.showMessageDialog(frame,
"Error Updating Data.", "Error",

      JOptionPane.ERROR_MESSAGE);

                                      }
                                } catch (IOException e1) {

                                      JOptionPane.showMessageDialog(frame, "Error
Updating Data.", "Error",
                                                  JOptionPane.ERROR_MESSAGE);
                                      e1.printStackTrace();
                                } catch (Exception e2) {
                                      JOptionPane.showMessageDialog(frame, "Error
Updating Data.", "Error",
                                                  JOptionPane.ERROR_MESSAGE);
                                      e2.printStackTrace();
                                }
                          }

                    }
              });

              btnConnect.addActionListener(new ActionListener() {
                    public void actionPerformed(ActionEvent e) {
                          comboBox.removeAllItems();
                          try {
                                List<String> IdList = fireAlarm.getFireAlarmIDs();
                                for (String id : IdList) {
                                      comboBox.addItem(id);
                                }
```

```
                        } catch (IOException e1) {
                              JOptionPane.showMessageDialog(frame, "Error Getting
Data.", "Error", JOptionPane.ERROR_MESSAGE);
                              e1.printStackTrace();
                        } catch (Exception e2) {
                              JOptionPane.showMessageDialog(frame, "Error Getting
Data.", "Error", JOptionPane.ERROR_MESSAGE);
                              e2.printStackTrace();
                        }

                  }
            });

            comboBox.addActionListener(new ActionListener() {
                  public void actionPerformed(ActionEvent e) {
                        if (comboBox.getSelectedItem() != null) {
                              String id = (String) comboBox.getSelectedItem();
                              try {
                                    FireAlarmDTO fiDto =
fireAlarm.getFireAlarmData(id);
                                    if (fiDto != null) {

      sliderSmoke.setValue(fiDto.getSmokeLevel());

      sliderCo2.setValue(fiDto.getCo2Level());
                                          lblFloorNo.setText("Floor No\t:\t" +
fiDto.getFloorNo());
                                          lblRoomNo.setText("Room No\t:\t" +
fiDto.getRoomNo());

      chckbxStatus.setSelected(fiDto.getStatus().equals("Active"));
                                    } else {
                                          JOptionPane.showMessageDialog(frame,
"Error Getting Data.", "Error",

      JOptionPane.ERROR_MESSAGE);
                                    }
                              } catch (IOException e1) {
                                    JOptionPane.showMessageDialog(frame, "Error
Getting Data.", "Error", JOptionPane.ERROR_MESSAGE);
                                    e1.printStackTrace();
                              } catch (Exception e2) {
                                    JOptionPane.showMessageDialog(frame, "Error
Getting Data.", "Error", JOptionPane.ERROR_MESSAGE);
                                    e2.printStackTrace();
                              }
                        }

                  }
            });

            btnChangeData.addActionListener(new ActionListener() {
                  public void actionPerformed(ActionEvent e) {
                        if (comboBox.getSelectedItem() != null) {
                              String id = (String) comboBox.getSelectedItem();
```

```
                                try {
                                        FireAlarmDTO fiDto =
fireAlarm.updateFireAlarmData(id, sliderCo2.getValue(),
                                                        sliderSmoke.getValue());
                                        if (fiDto.isSuccess()) {
                                                JOptionPane.showMessageDialog(frame,
"Data Updated.", "Updated",

        JOptionPane.INFORMATION_MESSAGE);
                                        } else {
                                                JOptionPane.showMessageDialog(frame,
"Error Updating Data.", "Error",

        JOptionPane.ERROR_MESSAGE);
                                        }
                                } catch (IOException e1) {
                                        JOptionPane.showMessageDialog(frame, "Error
Updating Data.", "Error",
                                                        JOptionPane.ERROR_MESSAGE);
                                        e1.printStackTrace();
                                } catch (Exception e2) {
                                        JOptionPane.showMessageDialog(frame, "Error
Updating Data.", "Error",
                                                        JOptionPane.ERROR_MESSAGE);
                                        e2.printStackTrace();
                                }
                        }

                }
        });
    }

    private int resetTime = 10;

    private int time = resetTime;

    // Sending the sensor details in specified time intervals

    public void updateDataInIntervals() {
        Thread t = new Thread(new Runnable() {

                @Override
                public void run() {
                        for (;;) {
                                try {
                                        Thread.sleep(1000);
                                        lblTime.setText("Update in : " + time);
                                        if (time <= 0) {
                                                if (comboBox.getSelectedItem() != null)
{
                                                        if (sliderCo2 != null &&
sliderSmoke != null) {
                                                                String id = (String)
comboBox.getSelectedItem();
                                                                try {
```

```
                                                                FireAlarmDTO fiDto
= fireAlarm.updateFireAlarmData(id, sliderCo2.getValue(),

	sliderSmoke.getValue());
                                                                if
(fiDto.isSuccess()) {
//
	JOptionPane.showMessageDialog(frame, "Data Updated.", "Updated",
//
	JOptionPane.INFORMATION_MESSAGE);
                                                                } else {

	JOptionPane.showMessageDialog(frame, "Error Updating Data.", "Error",

	JOptionPane.ERROR_MESSAGE);
                                                                }
                                                        } catch (IOException e1) {

	JOptionPane.showMessageDialog(frame, "Error Updating Data.", "Error",

	JOptionPane.ERROR_MESSAGE);

	e1.printStackTrace();
                                                        } catch (Exception e2) {

	JOptionPane.showMessageDialog(frame, "Error Updating Data.", "Error",

	JOptionPane.ERROR_MESSAGE);

	e2.printStackTrace();
                                                        } finally {
                                                                time = resetTime;
                                                        }

                                                }
                                        } else {
                                                time = 10;
                                        }
                                }
                                time--;
                        } catch (InterruptedException e) {
                                e.printStackTrace();
                        } catch (Exception e2) {
                                e2.printStackTrace();
                        }

                }
        }
    });

    t.start();
  }

}
```

```
public class FireAlarmImpl implements FireAlarm {

	// This method is used to set the fire alarm details for the fire alarm sensor
	// dummy GUI
	// It gets all the fire alarm sensor details first and then select the
relevant
	// fire alarm sensor
	// details from them and returns it using a Fire Alarm DTO
	@Override
	public FireAlarmDTO getFireAlarmData(String id) throws IOException {
		// getting all fireAlarmDetails
		URL url = new URL("http://localhost:5000/api/firealarm/");
		HttpURLConnection con = (HttpURLConnection) url.openConnection();
		con.setRequestMethod("GET");
		// Setting the Request header to accept response in JSON Format
		con.setRequestProperty("Accept", "application/json");

		// getting the response status code
		int responseCode = con.getResponseCode();
		// Reading the response
		Reader br = null;
		// checking the response is a success or an error
		if (responseCode >= 200 && responseCode <= 299) {
			br = new BufferedReader(new
InputStreamReader(con.getInputStream(), "utf-8"));
		} else {
			br = new BufferedReader(new
InputStreamReader(con.getErrorStream(), "utf-8"));
		}

		// parsing the JSON response to a Java Object
		Gson gson = new Gson();
		try {
			FireAlarmDTO fireAlarmDTO = gson.fromJson(br,
FireAlarmDTO.class);
			if (fireAlarmDTO != null) {
				List<FireAlarmDTO> list = fireAlarmDTO.getData();
				for (FireAlarmDTO iterator : list) {
					if (iterator.getId().equals(id)) {
						return iterator;
					}
				}
			}
		} catch (Exception e) {
			System.out.println(e);
		}
		return null;
	}

	// This method is used to set the co2 level and smoke level of a fire alarm
	// sensor
	// it returns the success status and the message
	@Override
	public FireAlarmDTO updateFireAlarmData(String id, int co2Level, int
smokeLevel) throws IOException {
		// Setting the REST API URL to PATCH Request
		URL url = new URL("http://localhost:5000/api/firealarm/" + id);
		// Opening a Connection
		HttpURLConnection con = (HttpURLConnection) url.openConnection();
		// Setting the Request Method
		// Since PATCH is not available in HttpURLConnection this is a 
workaround
		allowMethods("PATCH");
		con.setRequestMethod("PATCH");
		// Setting the Request Content Type
		con.setRequestProperty("Content-Type", "application/json; charset=UTF-
8");
		// Setting the Request header to accept response in JSON Format
		con.setRequestProperty("Accept", "application/json");
		// Enabling the writing to the connection output stream
		con.setDoOutput(true);
		// Creating the Data in the request body and writing it to output stream
		String data = "{\r\n" + "    \"co2Level\":\"" + co2Level + "\",\r\n" + "    
\"smokeLevel\": \"" + smokeLevel
				+ "\"\r\n" + "}";
		OutputStream os = con.getOutputStream();
		byte[] input = data.getBytes("utf-8");
		os.write(input, 0, input.length);

		// getting the response status code
		int responseCode = con.getResponseCode();
		// Reading the response
		Reader br = null;
		// checking the response is a success or an error
		if (responseCode >= 200 && responseCode <= 299) {
			br = new BufferedReader(new 
InputStreamReader(con.getInputStream(), "utf-8"));
		} else {
			br = new BufferedReader(new 
InputStreamReader(con.getErrorStream(), "utf-8"));
		}

		// parsing the JSON response to a Java Object
		Gson gson = new Gson();
		FireAlarmDTO fireAlarmDTO = gson.fromJson(br, FireAlarmDTO.class);
		if (fireAlarmDTO != null) {
			return fireAlarmDTO;
		}

		return null;
	}

	// This method is used to set the status of a fire alarm
	// sensor
	// it returns the success status and the message
	@Override
	public FireAlarmDTO updateFireAlarmData(String id, String status) throws 
IOException {
		// Setting the REST API URL to PATCH Request
		URL url = new URL("http://localhost:5000/api/firealarm/" + id);
```

```
			// Opening a Connection
			HttpURLConnection con = (HttpURLConnection) url.openConnection();
			// Setting the Request Method
			// Since PATCH is not available in HttpURLConnection this is a 
workaround
			allowMethods("PATCH");
			con.setRequestMethod("PATCH");
			// Setting the Request Content Type
			con.setRequestProperty("Content-Type", "application/json; charset=UTF-
8");
			// Setting the Request header to accept response in JSON Format
			con.setRequestProperty("Accept", "application/json");
			// Enabling the writing to the connection output stream
			con.setDoOutput(true);
			// Creating the Data in the request body and writing it to output stream
			String data = "{\r\n" + "    \"status\":\"" + status + "\"\r\n" + "}";
			OutputStream os = con.getOutputStream();
			byte[] input = data.getBytes("utf-8");
			os.write(input, 0, input.length);

			// getting the response status code
			int responseCode = con.getResponseCode();
			// Reading the response
			Reader br = null;
			// checking the response is a success or an error
			if (responseCode >= 200 && responseCode <= 299) {
				br = new BufferedReader(new 
InputStreamReader(con.getInputStream(), "utf-8"));
			} else {
				br = new BufferedReader(new 
InputStreamReader(con.getErrorStream(), "utf-8"));
			}

			// parsing the JSON response to a Java Object
			Gson gson = new Gson();
			try {
				FireAlarmDTO fireAlarmDTO = gson.fromJson(br, 
FireAlarmDTO.class);
				if (fireAlarmDTO != null) {
					return fireAlarmDTO;
				}
			} catch (Exception e) {
				System.out.println(e.getMessage());
			}

			return null;
		}

		//This method is used to get the all the FireAlarm Ids
		//This fire alarm ids displayed in the combo box
		//Using the id's the user can get sensor details of the fire alarms
		@Override
		public ArrayList<String> getFireAlarmIDs() throws IOException {
			// getting all fireAlarmDetails
			URL url = new URL("http://localhost:5000/api/firealarm/");
```

```
            HttpURLConnection con = (HttpURLConnection) url.openConnection();
            con.setRequestMethod("GET");
            // Setting the Request header to accept response in JSON Format
            con.setRequestProperty("Accept", "application/json");

            // getting the response status code
            int responseCode = con.getResponseCode();
            // Reading the response
            Reader br = null;
            // checking the response is a success or an error
            if (responseCode >= 200 && responseCode <= 299) {
                  br = new BufferedReader(new
InputStreamReader(con.getInputStream(), "utf-8"));
            } else {
                  br = new BufferedReader(new
InputStreamReader(con.getErrorStream(), "utf-8"));
            }

            // parsing the JSON response to a Java Object
            Gson gson = new Gson();
            try {
                  FireAlarmDTO fireAlarmDTO = gson.fromJson(br,
FireAlarmDTO.class);
                  if (fireAlarmDTO != null) {
                        List<String> ids = new ArrayList<String>();
                        List<FireAlarmDTO> list = fireAlarmDTO.getData();
                        for (FireAlarmDTO string : list) {
                              ids.add(string.getId());
                        }
                        return (ArrayList<String>) ids;

                  }
            } catch (Exception e) {
                  System.out.println(e);
            }
            return null;
      }
}
```