

**Development of Disaster Management Application**

**Mobile Computing Winter Semester 20/21**

*Surname :* **Khan**

*First Name :* **Muhammad Mubashir Ali**

*Matriculation Number :* **1324099**

*Course of Studies :* **M.E. in Information Technology**

*Guidance of :* **Prof. Ulrich trick**

*Submitted on :*

*Surname :* **Chadha**

*First Name :* **Chetan**

*Matriculation Number :* **1324510**

*Course of Studies :* **M.E. in Information Technology**

*Guidance of :* **Prof. Ulrich trick**

*Submitted on :*

Table of Contents

**Abstract3**

**Principles of Project 3**

**Terminologies Used 3**

Internet of Things (IoT)3

IoT Environment3

IoT Gateway4

Smart Home5

**Technologies Used 5**

Hypertext Markup Language (HTML)5

Cascading Style Sheets (CSS)5

JavaScript 6

eXtensible Markup Language (XML)6

JavaScript Object Notation (JSON)6

Java 7

RESTful Web Services7

**Protocols Description 7**

Hypertext Transfer Protocol (HTTP) 7

Constrained Application Protocol (CoAP) 8

**Project Description 9**

Use Case Explanation 9

**Project Architecture 10**

Protocol Functionalities 11

**Project Implementation 14**

Java Code Details 14

Steps to run the Application17

Wireshark Captures 19

**Problems Faced and Solutions22**

Using Offline Maps 22

Steps to run the Application17

**Possible Project Extension 23**

Extracting Live Location 24

Displaying the Rescuers reaching to Users on Map24

Chat Functionality between the User and Admin 24

**Acknowledgements 25**

**References 25**

1. **Abstract :**

Disaster Management can be defined as the management of resources and responsibilities for dealing with all humanitarian aspects of emergencies, in particular preparedness, response and recovery in order to lessen the impact of disasters. This project is based on a web application that provide victims a simple platform to seek help in case of a natural disaster and also make it easier for the rescuers to respond and provide necessary help to the victims. To achieve the task, we have built a complete front to back-end application based on different technologies and protocols which are discussed in detail in the following subsequent topics.

1. **Principles of Project :**

This project aims to develop a complete front-end, back-end application supported by multiple APIs which connects both the frontend and backend together. In a disaster situation, the emergency communication infrastructure is constructed from battery-supplied multi-radio wireless outdoor-routers which are deployed by first responders in the affected region. The application consists of three pages, Landing page, Admin page and the Rescuer Page. Victims have the possibility to connect to the internet and directly open the application which will route them to the landing page, where the victim will be able to see his current location on the map, the area affected and also the victims around him who are accessing the application. By filling out simple information, victim can request help. The rescuers have access to the Rescuer page and they can see both the positions of the victims and the location of the rescuers. Then there is an admin page which will be accessed by the administrator only. The administrator can add or remove both the victims and rescuers from the database manually in case there is some issue and can send notifications to both the victims and the rescuers.

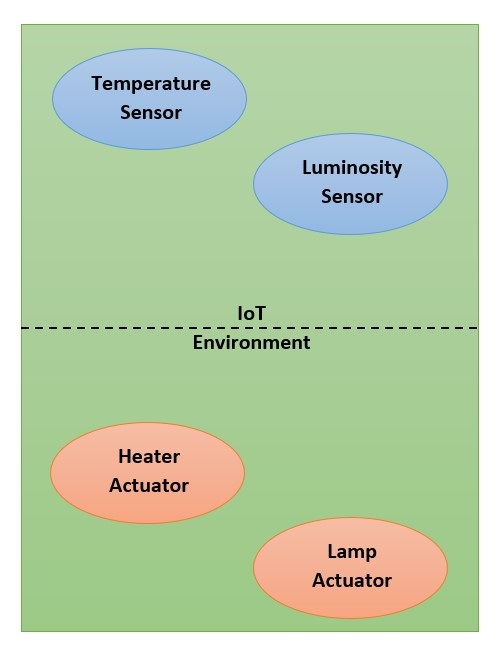
Web services will be based on HTTP protocol. Through web service, user can watch and control all these appliances on a web browser.

1. **Terminologies Used :**
   1. **Internet of Things (IoT) :**

The Internet of Things (IoT) consists of technologies that focuses on communication or basically data flow between different physical objects like electronic devices and virtual objects like emulators. These technologies provide interaction between humans and objects and also, among different objects. These objects work together to form a Network which is commonly termed as Internet of Things (IoT).

* 1. **IoT Environment :**

Different electronic devices or emulators forms the IoT Environment. Generally, these electronic devices are sensors and actuators. In this project, I have used an Emulated IoT Environment which consists different sensors and actuators.



**Figure 3.2.(a) IoT Environment with Sensors and Actuators**

* + 1. **Sensors :**

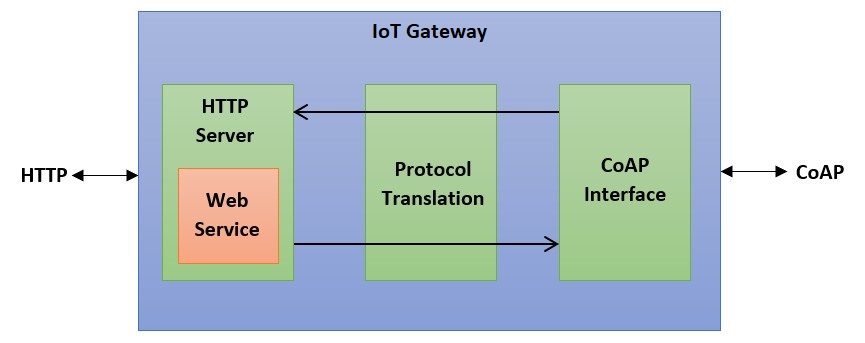
Sensors are the electronic devices which reads some environmental parameters like temperature, luminosity etc. For example, a temperature sensor can read temperature from the environment.

* + 1. **Actuators :**

Actuators are the electronic devices which controls the other electronic devices like heater, lamp etc. For example, a heater actuator can power on or power off the heater and can also sets the heating power of the heater.

* 1. **IoT Gateway :**

In order to provide communication between different protocols, an intermediate protocol translation must be done. This can be done by the use of a Gateway. For example, in this project, web service is using Hypertext Transfer Protocol (HTTP) [1] and IoT devices are based on Constrained Application Protocol (CoAP) [2], then, to provide communication between web service and IoT devices, an IoT Gateway is required.



**Figure 3.3.(a) IoT Gateway with Protocol Translation**

* 1. **Smart Home :**

In a Smart Home, there are different intelligent devices which can communicate with each other and perform their individual tasks with or without human intervention. These intelligent devices or smart home appliances are generally connected in a network. So, we can say that Smart Home is one of the applications of IoT. And, this project deals with the same.

1. **Technologies Used :**

In this project, to develop an application, we have used the following technologies:

* 1. **Angular Framework**

Angular is an application design framework and development platform for creating efficient and sophisticated web apps. It is a Typescript-based open-source web application framework led by the Angular Team at Google and by a community of individuals and corporations. The main advantage of this strongly typed language is that it helps developers to keep their code clean and understandable. This makes it quicker when it comes to debugging and also easier to maintain a large codebase as well as allowing the application to be scaled seamlessly. We have used Angular 8 for our project.

* 1. **Hypertext Markup Language (HTML) :**

The Hypertext Markup Language (HTML) is a standard markup language for creating Web Pages that are displayed in Web Browsers. It consists of different elements that are represented by tags. To display the content, these elements are used. In this project, we are using HTML 5 in angular framework. It is the latest version of HTML with some new elements.

* 1. **Cascading Style Sheets (CSS) :**

The Cascading Style Sheets (CSS) describes the style of the HTML document. In this project, we have used CSS 3 in angular framework.

* 1. **BootStrap:**

Bootstrap is HTML, CSS and JavaScript framework for developing responsive web pages. We have used different components of Bootstrap 4 and Bootstrap 5 in our application.

* 1. **TypeScript :**

Typescript is an open-source language which builds on JavaScript which is a scripting language which runs on the web browser. It is also called programming language of HTML. Along with TypeScript, we have used jQuery and AJAX in this project. jQuery is a JavaScript library which simplifies JavaScript programming. Asynchronous JavaScript And XML (AJAX) is not a programming language rather it is a technique for accessing web servers from a web page. Using AJAX, we can update a web page without reloading it.

* 1. **eXtensible Markup Language (XML) :**

The eXtensible Markup Language (XML) is a markup language like HTML. It is both human and machine readable data format which is used to store and transport data [3]. The following example shows the temperature and luminosity data in XML data format,

<data>

<temperature>20</temperature>

<luminosity>80</luminosity>

</data>

In this project, I have used com.fasterxml.jackson.dataformat library written in Java for creating and reading data in XML format.

* 1. **JavaScript Object Notation (JSON) :**

The JavaScript Object Notation (JSON) is a syntax for storing and exchanging data. It is basically a text which can be directly converted into JavaScript Object and vice versa. It is a data format which is lighter than XML and hence, is more faster for data communication [4]. The following example shows the temperature and luminosity data in JSON data format,

{

"data":{

"temperature":20,

"luminosity":80

}

}

In this project, I have used org.json library written in Java for creating and reading data in JSON format.

* 1. **SpringBoot:**

Spring Boot is an open source Java-based framework used to build stand-alone and production ready spring applications that you can just run. We have used Spring boot to create REST APIs. It provides a powerful batch processing and manages REST endpoints.

* 1. **Java :**

Java is an Object Oriented Programming (OOP) language which is used for creating Desktop Applications, Web Applications, Web Servers, Application Servers and much more. It is an open-source and free programming language. In this project, we have used Java and Spring Boot for developing the backend part of the application.

* 1. **H2 Database :**

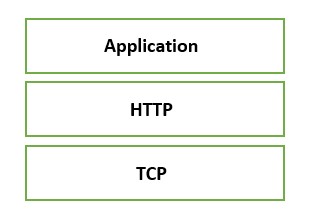
We have implemented our database using H2 database engine. H2 is a relational database management system written in Java. It can be embedded in Java applications or run in client-server mode. There is full intrinsic support for it in the Spring Boot ecosystem.

* 1. **RESTful Web Services :**

Web Services are open standard based web applications that are used for exchanging data between applications. The applications written in various programming languages and running on various platforms can also use web services to exchange data over network. RESTful Web Services are based on REST architecture. Representational State Transfer (REST) is a web standards based architecture where everything is a resource. In this architecture, REST Server provides access to resources and REST client accesses and modifies the resource. REST generally uses text, XML and JSON representation to represent a resource where JSON is the most popular one. It uses HTTP protocol. We have used three HTTP methods in REST based architecture: GET, POST and DELETE, where GET is used to read a resource, POST is used to create a resource and DELETE is used to remove a resource.

1. **Protocols Description :**
   1. **Hypertext Transfer Protocol (HTTP) :**

The Hypertext Transfer Protocol (HTTP) is an application-level protocol for distributed, collaborative, hypermedia information systems. It is a stateless protocol for the transmission of data on the application layer. It is based on Transmission Control Protocol (TCP). It is also used as a generic protocol for communication between user agents and proxies/gateways to other Internet systems [1].



**Figure 5.1.(a) HTTP based on TCP**

HTTP also provides request/response based on client/server model and has several methods such as HTTP GET, HTTP POST, HTTP PUT, HTTP DELETE, OPTIONS etc. But RESTful Web Services uses only four methods, i.e., GET, POST and DELETE.

1. **Project Description :**

**Scope of the Project :**

The disaster management project is derived from an idea to investigate the possibility to deploy an emergency communication infrastructure in case of a natural disaster destroyed or heavily damaged the existing communication infrastructure. The emergency communication infrastructure is constructed from battery-supplied multi-radio wireless outdoor-routers which are deployed by first responders in the affected region. The scope of the project is to design and develop a disaster management application which will be provided to the end-users in such unfortunate conditions.

For our project, we have divided the whole application into three pages namely User Page, Admin Page and Rescuer Page.

* 1. **User Page :**

This page is accessed by the victim. As soon as he connects to the application, the User’s page will be displayed. On the left side of the page there is a form to collect the user’s information and on the right side, the street view of the leaftlet map is displayed. Upon filling the information and pressing the “Ask for help button”, the user can see himself on the map as a red pointer based upon the longitude and latitude coordinates. Also, the area affected by the disaster can be seen with a pink circle. Now, in order to seek help, the victim needs to enter his name, select emergency type, select health status and disaster severity using the drop down boxes and just click the “Ask for help” button. All the data will be then posted through the API Gateway to the H2 database and later the Admin, based upon the information that the victim enters, can make suitable arrangements for his rescue. Furthermore, below the “Ask for help” button, there is also “Show Other Victims” button, which upon clicking displays the other victims in the same map and their positions, so the victim can see all the other nearby victims who have logged in to the application. Then there is timer, which will be set by the “admin” that will display how much time it will take for the rescuer to reach to the victim. The victim can also receive messages via notification models in order to get the important information in real time and does not need the web page to be refreshed.

* 1. **Admin Page :**

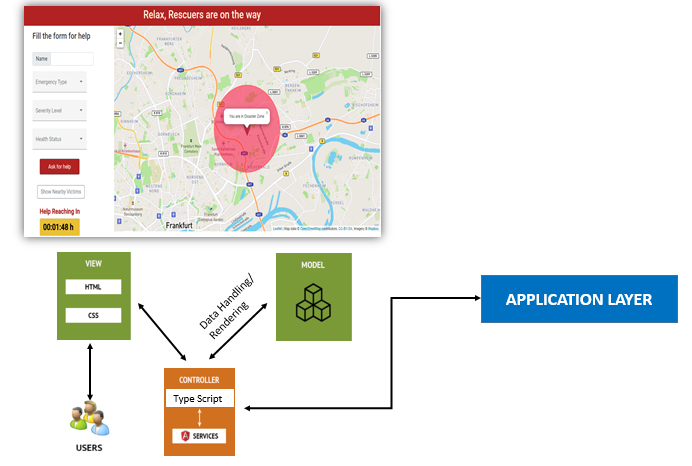
Admin Page will only be accessed by the administrator. On the admin page there are different options for the administrator to manage both the victims and the rescuers. Admin can load multiple users and rescuers from the database to display all the information including the locations of the rescuers and the victims on the web page. Also, administrator has the possibility to add or delete the users and rescuers manually if in case there is some problem with the application. The admin can send messages in the form of notifications to both the users and the rescuers from the admin page. Also, there is a chat functionality between the Admin and the Rescuers which enable the Admin chat with a particular rescuer

* 1. **Rescuer Page :**

Rescuers first need to prove their identity in order to login to the rescuer’s page by entering their username and password. The user name and password will be matched within the database and only the registered rescuers are routed to the rescuer’s page. For the new rescuers, they can sign up and later can login to the Rescuer’s page. This functionality is added so that every rescuer has his own identity and can be tracked individually. Upon logging, the similar map to that on the user’s page is displayed showing the locations of all the victims with red marker and other rescuers with green marker. As soon as the rescuer clicks the “Start Rescuing” button, his location will be displayed on the map and admin will be notified. The rescuer has also the option to take a break by clicking on “Take a break” button which will notify the administrator about the non-availability of that rescuer and his marker will also disappear and will not be shown on the map to the other rescuers. (The rescuer will set a timer according to the location of the victim about how much time will it take for the rescuer to reach that particular victim and a countdown timer will start on the user’s page)

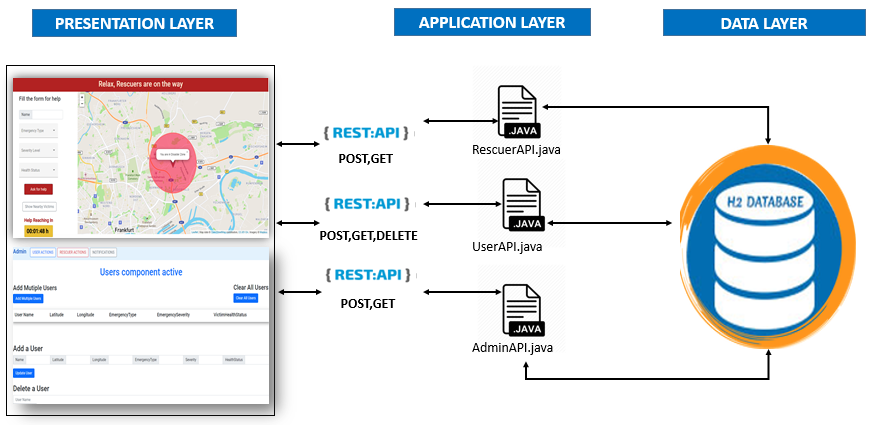
1. **Project Architecture :**
   1. **Front-end :**

The following diagram shows the front-end architecture of the project:

**Figure 7.(a) Front-End Architecture** 

The front-end part of the application has been completely built on Angular 8.0 framework. Angular framework is based on the Model View Controller (MVC) as shown in the picture above. The benefit of using a MVC is that the program logic is divided into three interconnected elements such that each section of your code has a purpose, and those purposes are different. Some of your code holds the data of your app, some of your code makes your application look nice, and some of your code controls how your app functions. The view part is the element that is shown to the user which consists of HTML and CSS. Here in Angular 8 version we have used HTML5 and CSS3. The controller part is based on type script which is the super script of JavaScript and is connected to the Application Layer. All the API calls are made in this part.

* 1. **Back-end :**

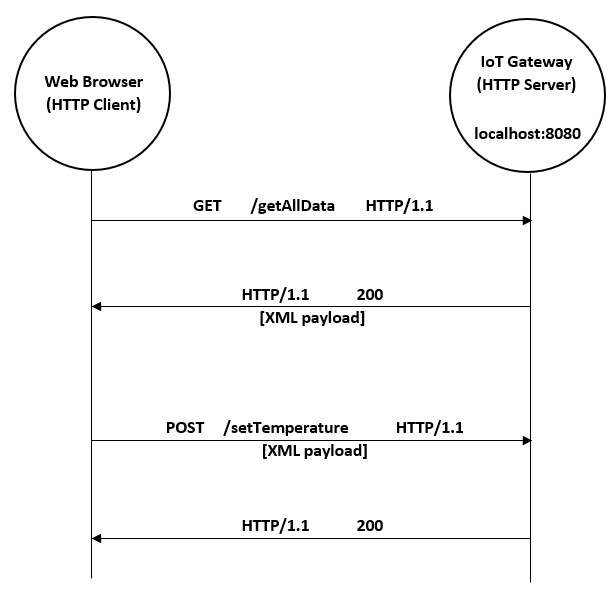
The following diagram shows the backend architecture of the project:

**Figure 7.(b) Back-End Architecture**

Here, we have divided the architecture in three layers namely presentation, application and data layer. The presentation layer consists of the front end part which is connected to the application layer via the REST API gateways. In the application layer, we have our main functions namely RescuerAPI, UserAPI and AdminAPI. The RescuerAPI function hold all the logics related to the Rescuer page and similarly all the program logics related to user page and admin page reside in the UserAPI and AdminAPI functions respectively. The description about all the REST API used in each functions can be found below.

* 1. **Protocol Functionalities :**
     1. **HTTP :**

This protocol is used in between IoT Gateway and Web Browser, where, IoT Gateway acts as HTTP Server and Web Browser acts as HTTP Server. Also, the socket for the IoT Gateway (HTTP Server) is localhost:8080. The following shows the Message Sequence Chart for this communication,



**Figure 7.1.1.(a) HTTP GET and POST method**

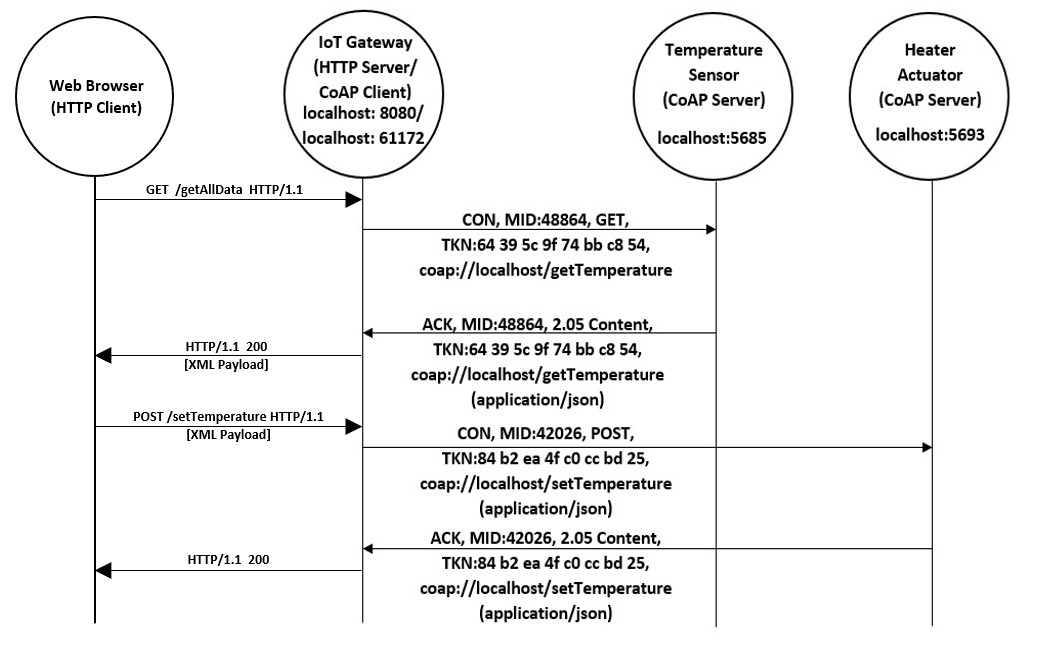
Here, we can see that web browser is sending GET and POST request to IoT Gateway and IoT Gateway is also sending the corresponding responses. Also, the payload is in the XML format.

The following table shows HTTP URLs used in the project:

|  |  |  |
| --- | --- | --- |
| **Function** | **Method** | **Description** |
| **RescuerAPI.java** | POST | Sign-up Rescuer |
| POST | Login Rescuer |
| GET | Load all rescuers to the page |
| GET | Add multiple rescuers |
| GET | To clear all rescuers from the page |
| DELETE | Delete a rescuer |
| GET | To send notification to the rescuer |
| GET | Update rescuer status on the map |
| **UserAPI.java** | POST | To add a user to the database |
| GET | To show user on the map |
| DELETE | Delete a user |
| GET | Show all users on the map |
| GET | Add multiple users |
| GET | To send notification to the rescuer |
| GET | To clear all users from the page |
| **AdminAPI.java** | POST | Send notification to users by admin |
| POST | Send notification to the rescuers by admin |
| GET | To show all notifications sent by admin |

* + 1. **Complete Flow :**

The following shows the complete flow for this communication,



**Figure 7.1.3.(a) Complete flow including both HTTP and CoAP protocols**

Here, the complete flow of data is shown for one sensor and one actuator case. Similarly, all other communications are taking place.

* + 1. **Ports Used in the Project :**

The following ports are used in project, please free them if they are already in use to run the application:

|  |  |  |
| --- | --- | --- |
| **Port** | **Protocol** | **Description** |
| 4200 | HTTP | IoT Gateway HTTP Server |
| 9902 | CoAP | Temperature Sensor CoAP Server |

1. **Project Implementation :**

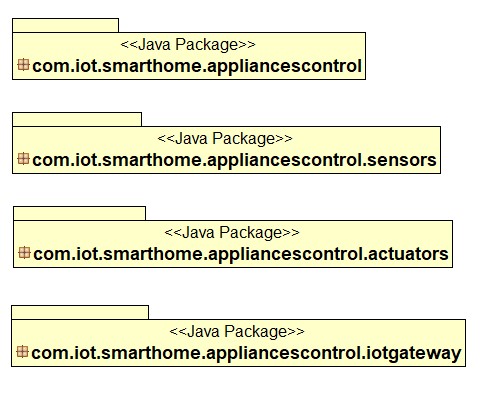
This project is implemented in Java programming language using Spring Boot framework and Eclipse IDE. Along with Java, HTML 5, CSS 3, Bootstrap 4, JavaScript, jQuery and AJAX are also used. For implementation of CoAP protocol, Californium (Cf) Core library is used. Other used technologies are already discussed in Technologies Used section.

* 1. **Java Code Details :**

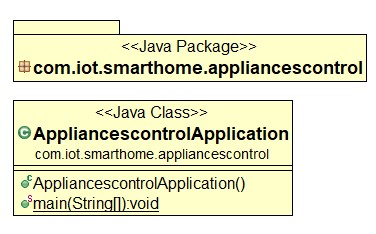
This project contains one Spring Boot Application Project named “appliancescontrol”.

The following points discuss this application in detail :

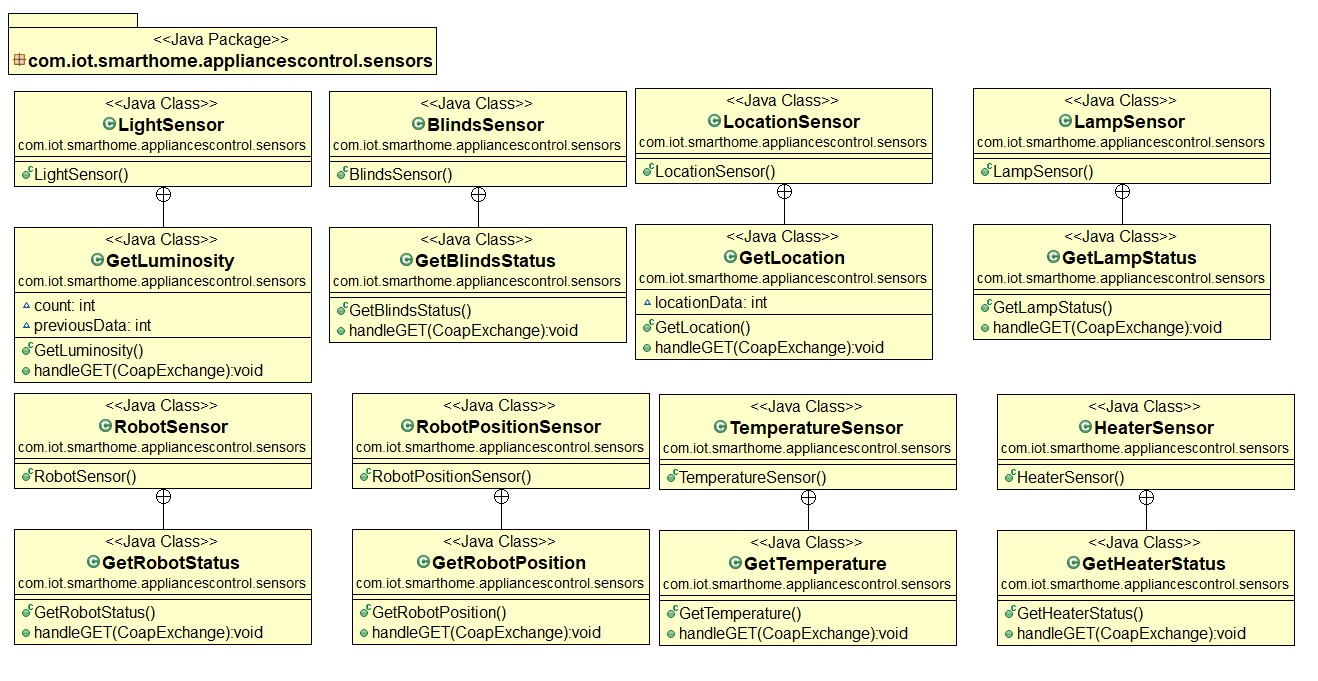
* In this project, we have in total 4 packages :



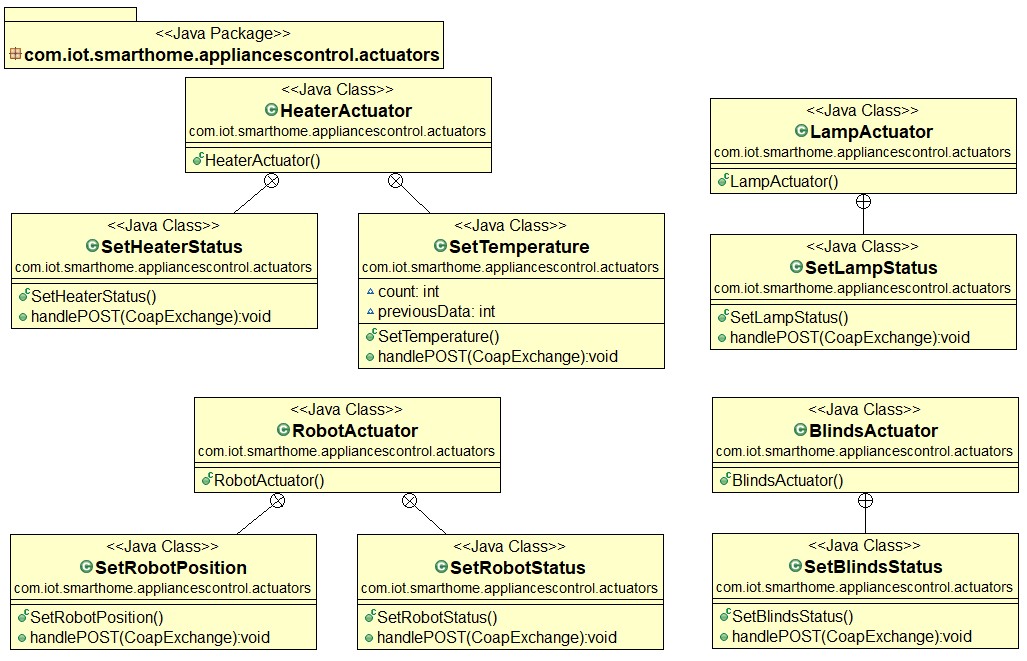
* In first package, we have following class, this class basically starts the Spring Application :



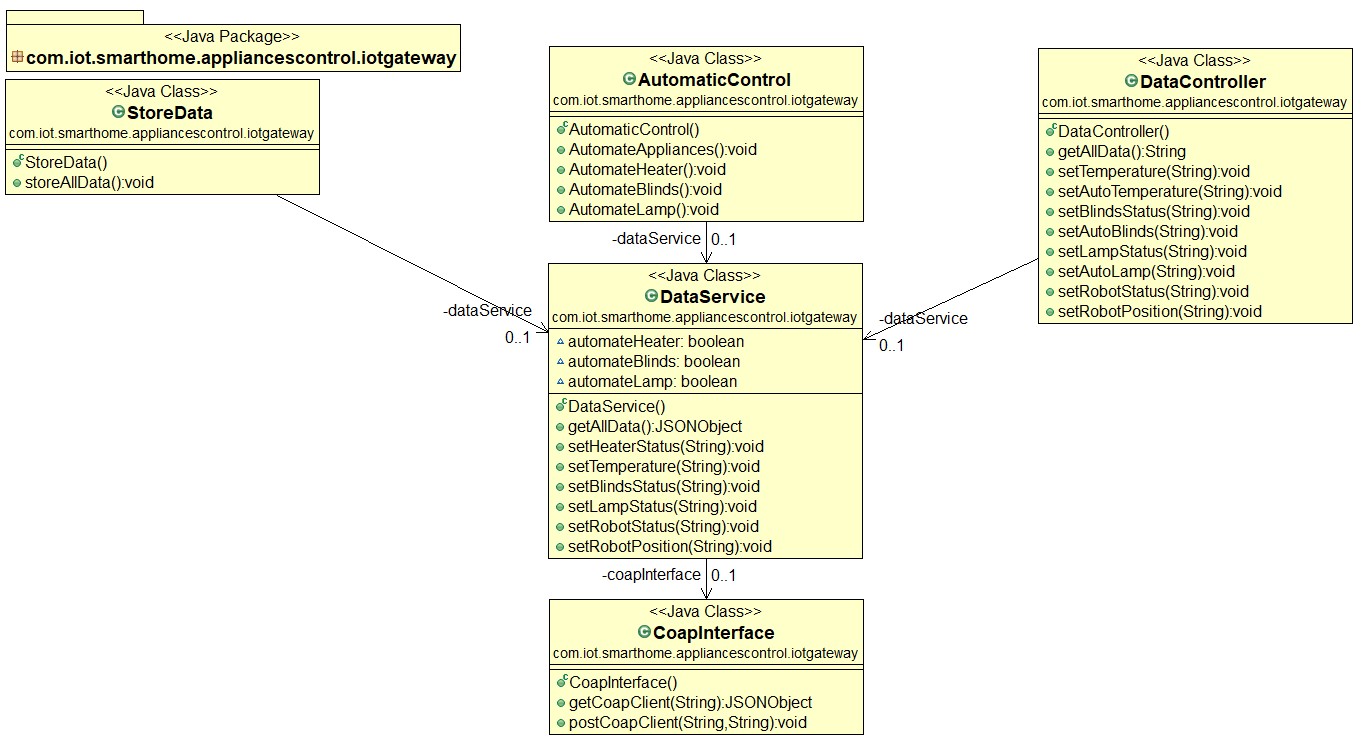
* In second package, we have following classes, these classes are basically implementation of sensors :



* In third package, we have following classes, these classes are basically implementation of actuators :



* In last package, we have 5 classes named StoreData, AutomaticControl, DataController, DataService and CoapInterface, these classes are implemented to form an IoT Gateway. StoreData class stores the data into LocalStorage.json file, AutomaticControl class implements automation logic of different appliances, DataController class handles the HTTP GET and POST requests from HTTP client, DataService class gets and sets the data, all classes requests this class to get and send data and at last CoapInterface class provides the interface for communication in CoAP protocol.

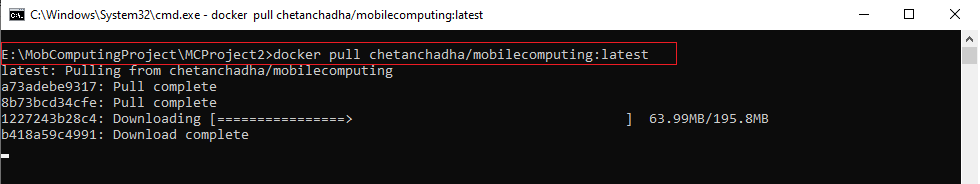


For more details of classes, methods and attributes, please refer to the documentation inside Source Code.

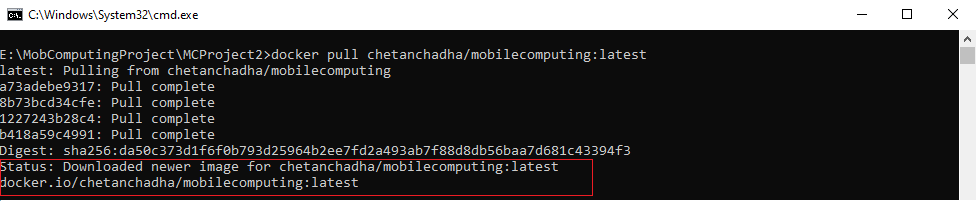
* 1. **Package the Application in a Docker Container**
  2. **Steps to run the application :**
     1. **Back-End:**

The following are the steps to run the application in a Docker container:

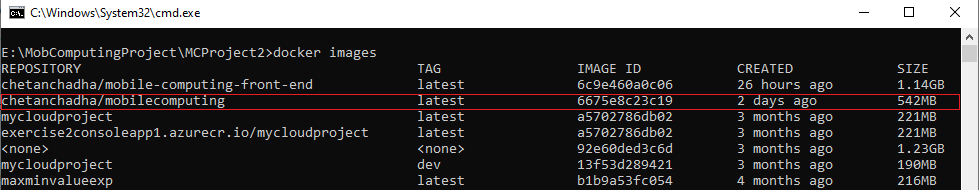
* Open the terminal in linux or cmd in Windows.
* First pull the docker image to your local machine using the command “docker pull <name of the docker image>”



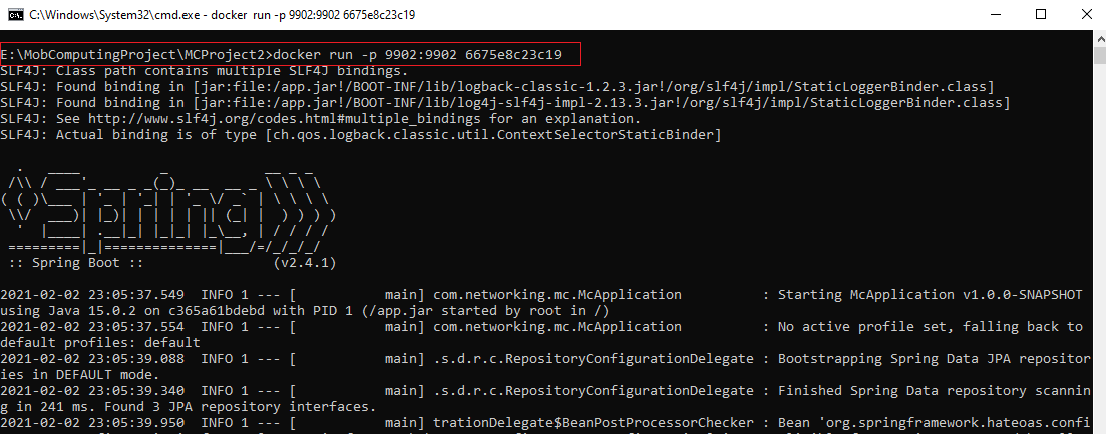
* Once the image is downloaded

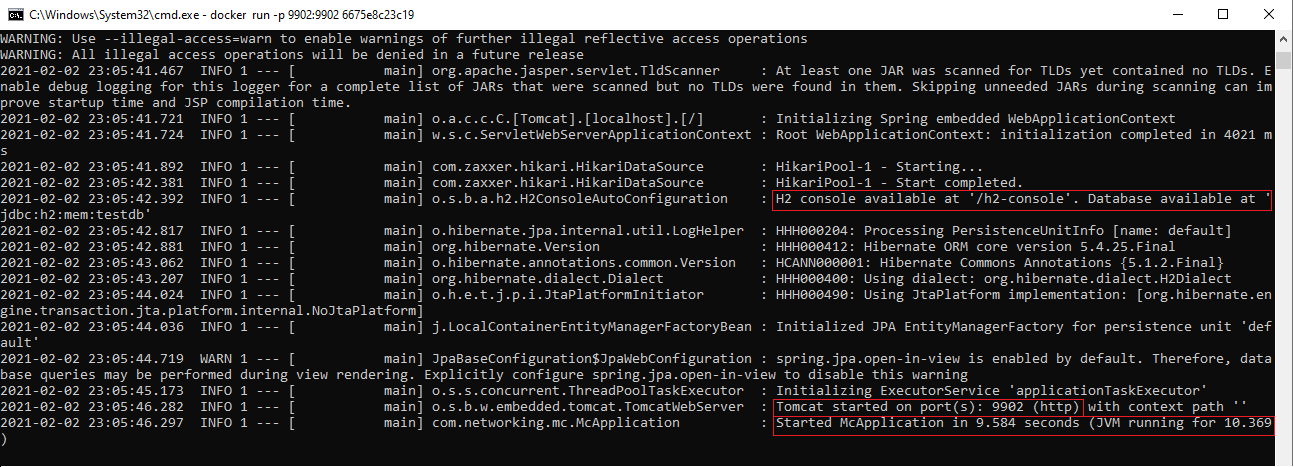


* Then write command “docker images” to see the Docker image pulled.

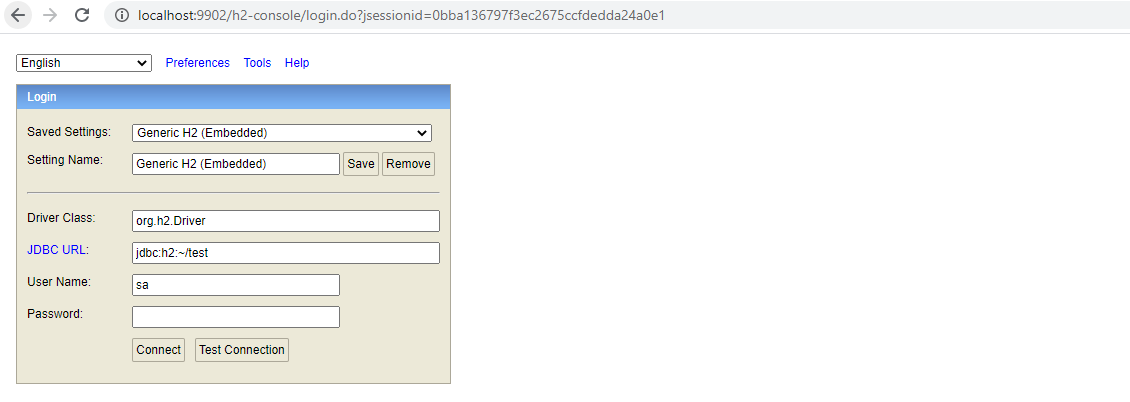


* Now to run the docker file, write the command “docker pull <image ID>” like shown below





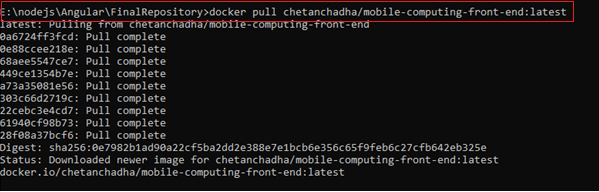
* Here we can see that the Spring Boot has started and the server Tomcat is running on port 9902. To access the database write “<http://localhost:9902/>h2-console“ in the web browser to view our H2 database.



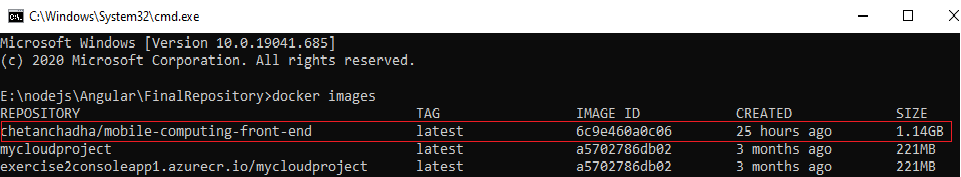
* Enter the username and password to access the data inside the database
  + 1. **Front-End:**

The following are the steps to run the application in a Docker container:

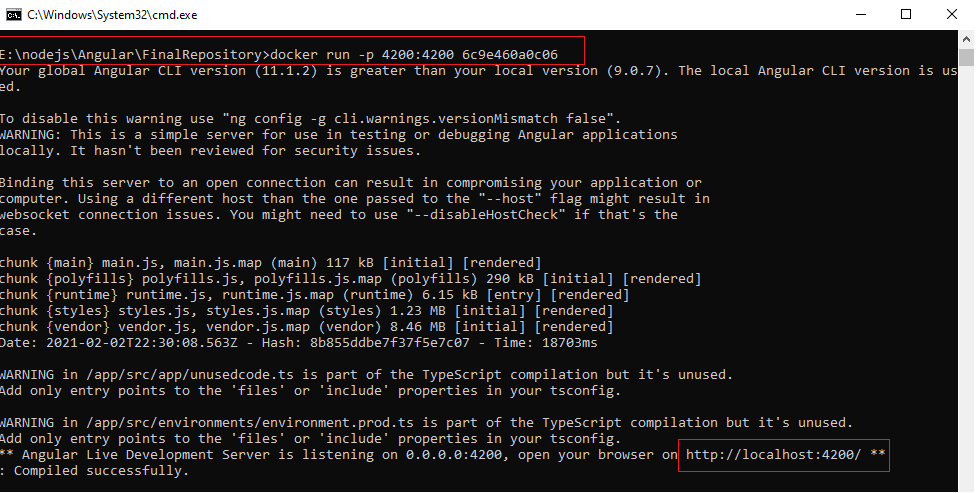
* Open the terminal in linux or cmd in Windows.
* First pull the docker image to your local machine using the command “docker pull <name of the docker image>”

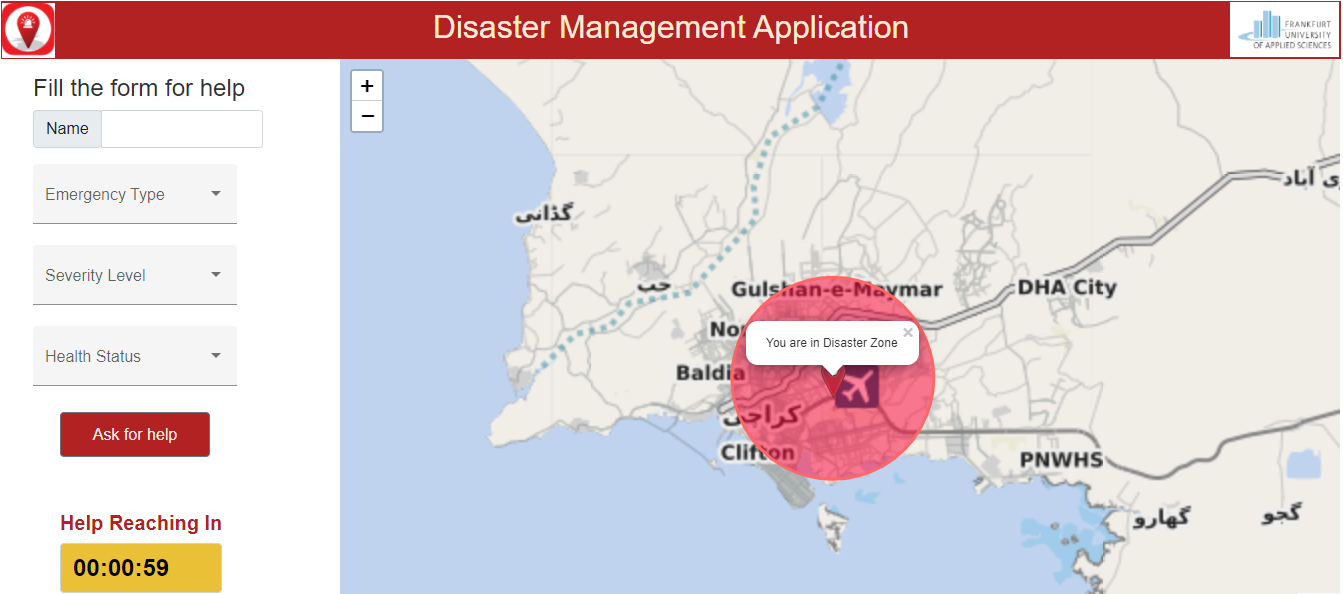


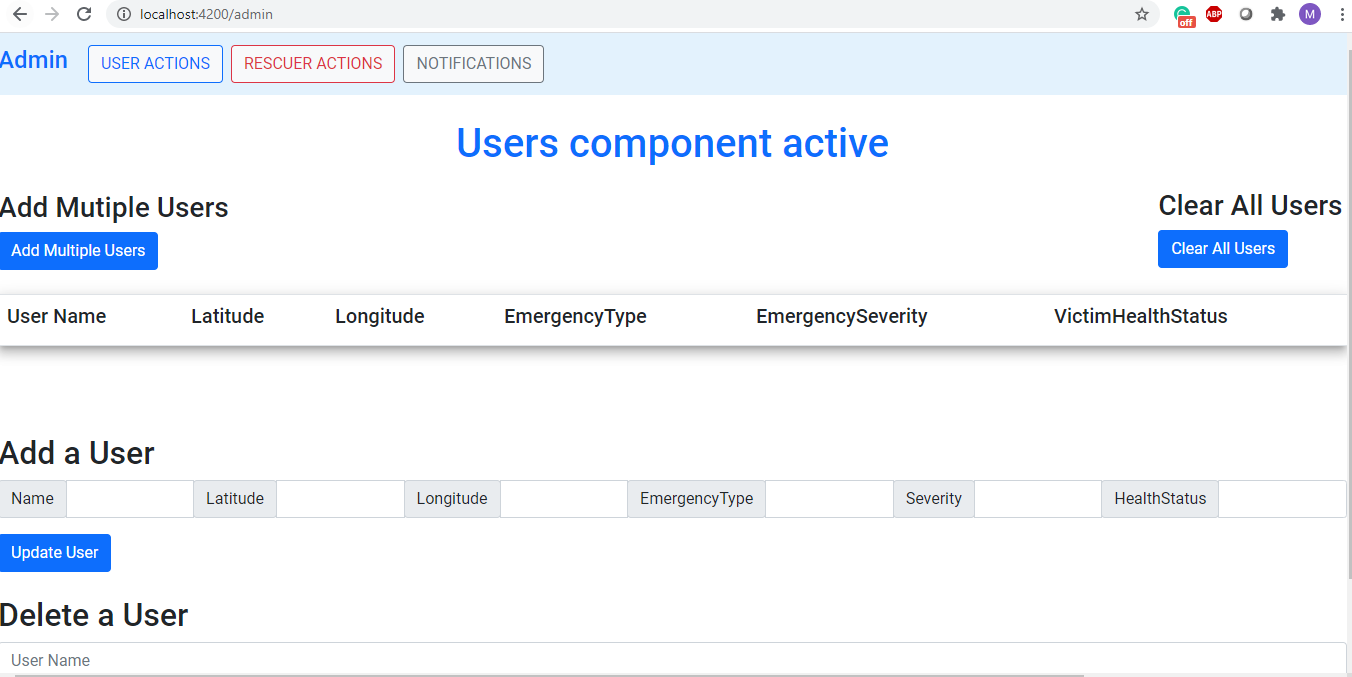
* Then write command “docker images” to see the Docker image pulled.



* Now to run the docker file, write the command “docker pull <image ID>” like shown below



* Here we are using port 4200 to open the web page. Angular use port 4200 by default. Now copy the local host “ <http://localhost:4200/> “ in the web browser to open the application.
* When you open the application, it will by default be routed to the User Page. To open the admin page just write in the browser <http://localhost:4200/admin> and similarly for rescuer page to be displayed write <http://localhost:4200/rescuer>



1. **Problems Faced and Solution :**

There was several challenges that we faced during the implementation in order to fulfill the project requirements. Some of the major issues and how we tackled them are discussed below.

* 1. **Using Offline Maps :**

In our application we have used leaflet maps to show the location of the user which gives a better visualization to the user and allows the user to also see the nearby victims. For this we used Google Maps API to fetch the the live map tiles to be displayed. However, as a major project requirement, we were not supposed to make an external API call as it uses the internet connection which was now allowed. So after doing a lot of research, we figured it out that map tiles can be downladed locally and the application should point out to the location where the map tiles are stored and hence can be displayed on the front-end without making an external API call.

* + 1. **Mobile Atlas Creator (MOBAC):**

Mobile Atlas Creator (MOBAC) version 2.4.1 was used to download map tiles and create atlas. Mobile Atlas Creator (formerly known as TrekBuddy Atlas Creator) is an open source (GPL) program which creates offline atlases. As source for an offline atlas Mobile Atlas Creator can use a large number of different online maps such as [OpenStreetMap](http://www.openstreetmap.org/) and other online map providers.

* 1. **Notifications Without refreshing the page :**
  2. **Create Docker Image and containerize the application:**
     1. **Docker Image for Angular:**
     2. **Docker image for SpringBoot:**
  3. **Creating APIs to access the Database:**
     1. **Testing APIs using Postman:**

1. **Possible Project Extension :**

Since the project is a Proof of Concept (PoC) and has been developed under a limited time constraint. There is a great room for improvement to this application. Apart from different functionalities, there are some of the ideas that we have and can be added to the application to make it more comprehensive and user friendly.

* 1. **Extracting Live Location :**

Due to the limited amount of time we were able to implement the concept of offline maps. Also, we have written the complete code to how to take the user’s live location(latitude and longitude) and store it in the database and then retrieve it from the database and display it on the map. However, at this time we have not worked on to fetch the live location thorugh the GPS in “offline mode” (without internet connectivity). At this time we have stored some users data including their location in the database and to demonstrate our concept, we are fetching their location(latitude and longitude) from the database and displaying them on the map.

* 1. **Displaying the Rescuers reaching to users on Map :**

It would be more convinient for the users to see the live location of the rescuers on the map, where are they and how will they be reaching to the victims. We have already written the code to the show the distance between the rescuers and the users for further extension.

* 1. **Chat functionality between the User and Admin :**

For now we have incorporated the chat functionality between the Admin and the rescuers. We have a particular login for every rescuer so that every rescuer has his own identity. This enables the Admin to know which rescuer is sending the message and the reply could be sent to that particular rescuer. For now we did not want the user to sign up first in order to access the application so we only kept the notification functionality for the user. Users can only receive the notification messages but cannot have a one on one chat, so this could be another functionality that can be added.

* 1. **Timer should be set for ever victim individually:**

We have displayed a timer on the front-end which displays the count down time so that a user knows when the rescuer will reach him. For now we are just using a value to be displayed on the counter so demonstratrate the concept. But this value should be setup by the rescuers.

1. **Acknowledgements :**

I would like to thank Prof. Armin Lehmann for providing such an interesting and useful project and also, for providing proper guidance for it. I would also like to thank Mr. Frick and Mr. Shala for providing advice and guidance during the course of this project. I also appreciate the work of Open Source Community for providing various libraries especially for CoAP protocol and thankful to them.

1. **References :**

|  |  |
| --- | --- |
| [1] | Internet Engineering Task Force (IETF), "RFC 2616 - Hypertext Transfer Protocol -- HTTP/1.1". |
| [2] | Internet Engineering Task Force (IETF), "RFC 7252 - The Constrained Application Protocol (CoAP)". |
| [3] | w3schools.com, "XML Tutorial," 16 January 2020. [Online]. Available: https://www.w3schools.com/xml/. |
| [4] | w3schools.com, "JSON - Introduction," 16 January 2020. [Online]. Available: https://www.w3schools.com/js/js\_json\_intro.asp. |
| [5] | Eclipse Californium, "Californium (Cf) Core," [Online]. Available: https://www.eclipse.org/californium/. |