



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 : 8th March, 2021

Surname : Chadha

First Name : Chetan

Matriculation Number : 1324510

Course of Studies : M.E. in Information
Technology

Guidance of : Prof. Ulrich Trick

Submitted on : 8th March, 2021

Table of Contents

Abstract.....	4
Principles of Project	4
Terminologies Used	4
User	4
Victim.....	4
Admin	5
Rescuer	5
Technologies Used	5
Angular Framework.....	5
Hypertext Markup Language (HTML).....	5
Cascading Style Sheets (CSS)	5
Bootstrap	6
TypeScript.....	6
JavaScript Object Notation (JSON)	6
SpringBoot	7
Java	7
H2 Database	7
RESTful Web Services	7
Protocols Description	7
Hypertext Transfer Protocol (HTTP)	7
Project Description	8
User Page	9
Admin Page	9
Rescuer Page	9
Project Architecture	10
Front-End	10
Back-End	11
Protocol Functionalities	12
Project Implementation	14
Java Code Details	14

Angular Code Details	16
Package the Application in Docker Container.....	17
Steps to run the Application	18
Problems Faced and Solutions.....	21
Using Offline Maps	21
Notification Centre.....	22
Accessing APIs after Containerization.....	22
Testing APIs	22
Possible Project Extension	23
Extracting Live Location	23
Displaying the Rescuers reaching to Users on Map.....	23
Chat Functionality between the User and Admin	23
Timer should be set for every victim individually	23
Security	24
Caching the Login details	24
Storing the Data	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.

2. 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.

3. Terminologies Used

3.1. User

Users are ones connecting to the application. In a disastrous situation, when any person connect to the application to seek help, they are termed as users or victims

3.2. Victim

Another term that is interchangeably used with user is victim. As the name suggest victims are the people who are hit by a disaster and opening the application to ask for help. Users and victims are used to address the same people

3.3. Admin

Admin or administrator refers to the person who is controlling the application or the flow between the user and rescuer

3.4. Rescuer

The term is used in this project to address the people who saves someone from a dangerous or difficult situation. Rescuers will try to reach out to the users/victims to protect them and relocate them to a safe location.

4. Technologies Used

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

4.1. 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.

4.2. 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.

4.3. 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.

4.4. 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.

4.5. 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. We are making asynchronous typescript for backend calls. Asynchronous typescript is not a programming language rather it is a technique for accessing web servers from a web page without waiting for the servers to respond back.

4.6. 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 faster for data communication [4].

The following example shows the rescuer data in JSON data format,

```
[
  {
    "rescuerName": "Rescuer1",
    "latitude": "50.3201004",
    "longitude": "8.8521",
    "status": "Active"
  }
]
```

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

4.7. 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.

4.8. 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.

4.9. 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.

4.10. 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.

5. Protocols Description

5.1. 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].

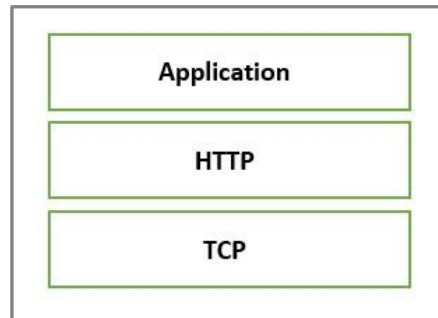


Figure 1. HTTP based on TCP

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.

6. 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.

6.1. 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 leaflet 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.

6.2. 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 all the rescuers available at that time.

6.3. 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 that a rescuer has been activated. 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.

7. Project Architecture

The below diagram shows the architecture of the project:

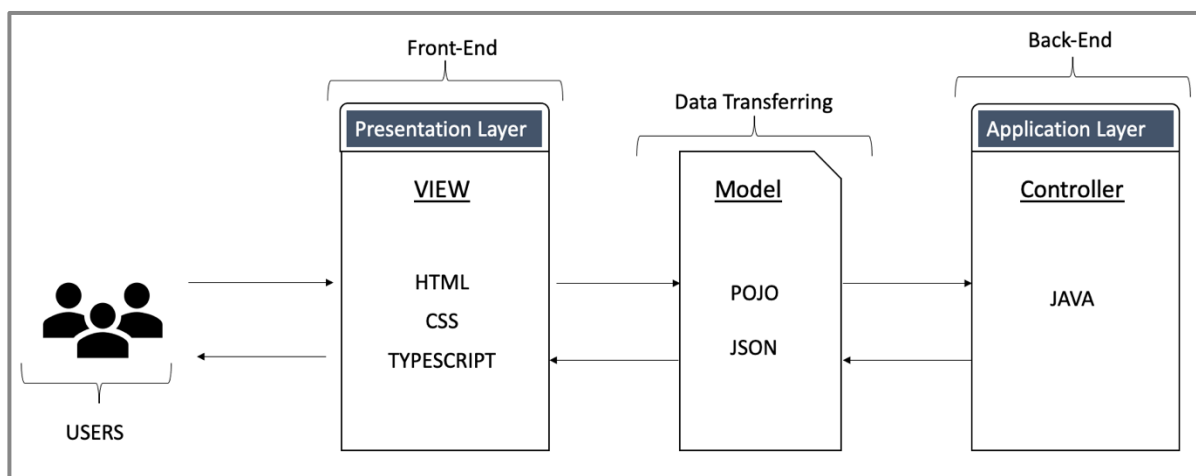


Figure 2. Project Architecture

As we can see the application has been divided into two main parts. That is Front-end and back-end. We have used MVC model in which we have a controller for backend logic and front-end to view the front-end. To render the data we are using POJOs(Plain old java object). Now let's discuss each aspect step by step.

7.1. Front-end

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

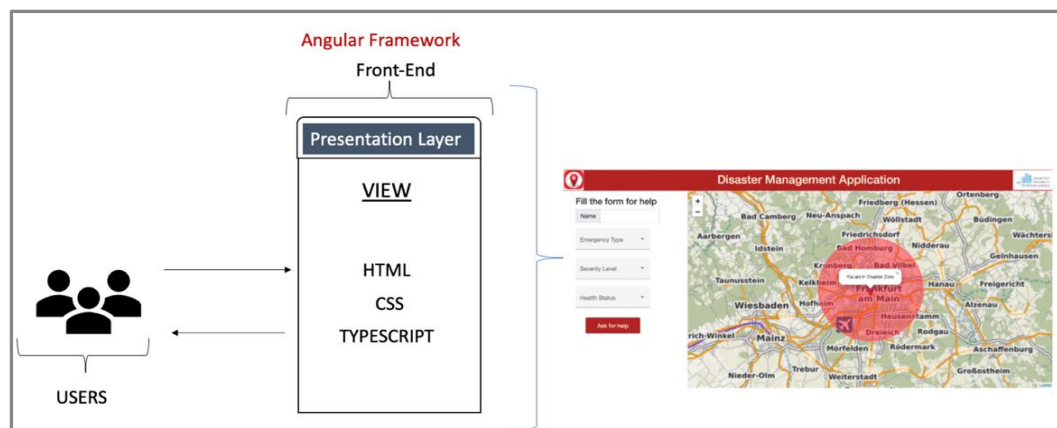


Figure 3. Front-End Architecture

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.

7.2. Back-end

The following diagram shows the backend architecture of the project:

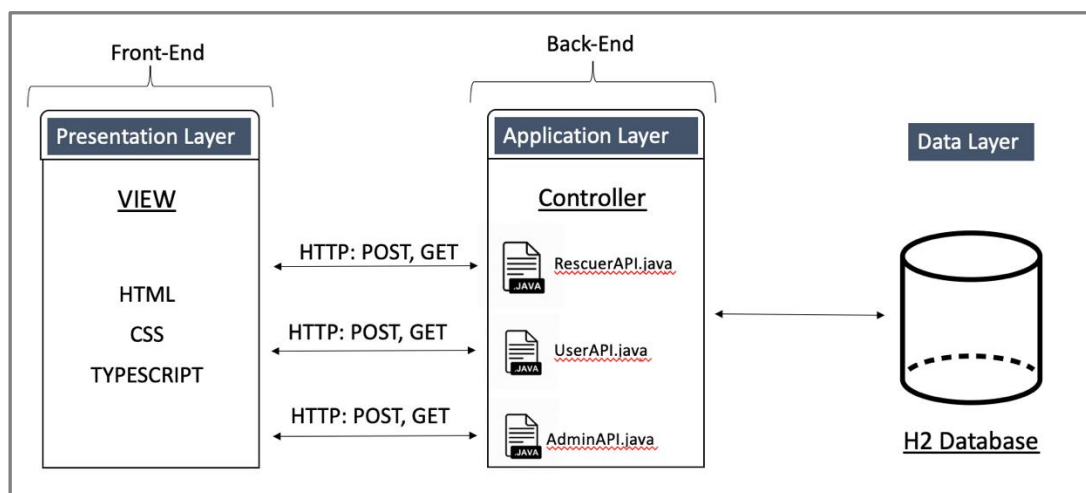


Figure 4. Backend Architecture

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.

7.3. Protocol Functionalities

7.3.1. HTTP

The HTTP protocol is used in between Backend application and Web application, where Backend application acts as HTTP Server and Web application acts as HTTP Client. Also, the socket for the Backend application (HTTP Server) is localhost:9200. Whereas the socket for the web application (HTTP Client) is localhost: 4200. The following shows the Message Sequence Chart for this communication:

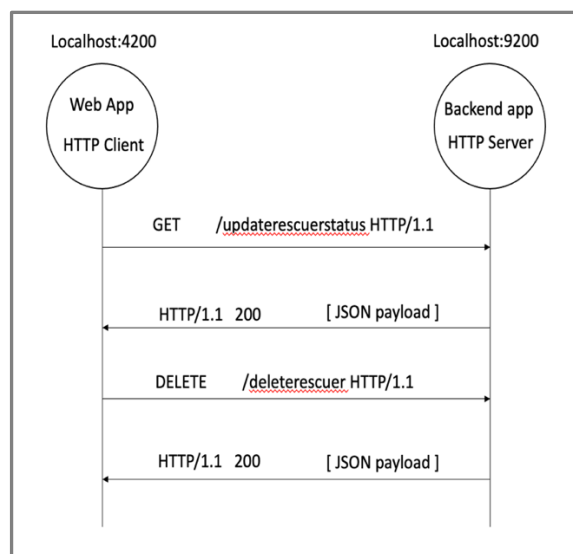


Figure 5. HTTP GET and DELETE methods

Here, we can see that web browser is sending GET and DELETE request to Backend service and Backend service is also sending the corresponding responses. Also, the payload is in the JSON format.

The following table shows HTTP URLs used in the project:

Function	Method	Description
	POST	Sign-up Rescuer
	POST	Login Rescuer

RescuerAPI.java	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

NOTE: Apart from these two Admin can access the user and rescuer APIs as well

7.3.2. Complete Flow

The following shows the complete flow for this communication

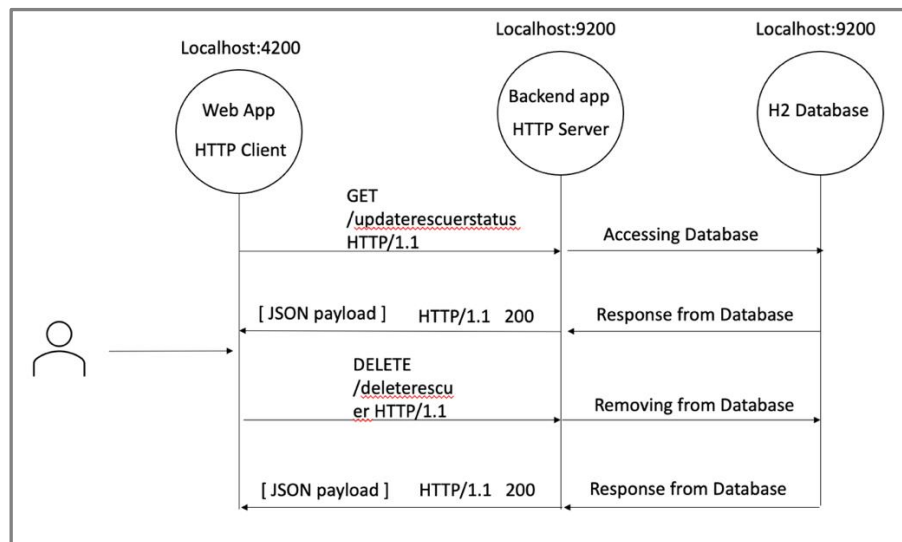


Figure 6. Complete flow including HTTP protocols

Here, the complete flow of data is shown i.e web app, backend and H2 database. Similar to this all other communications are taking place.

7.3.3. 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	Front end web application, http client
9902	HTTP	Backend application, http server

8. Project Implementation

This project is implemented in Java programming language using Spring Boot framework and IntelliJ IDE. Along with Java, HTML 5, CSS 3, Bootstrap 4 and Typescript are also used. Other used technologies are already discussed in Technologies Used section.

8.1. 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 six packages and two starter classes :

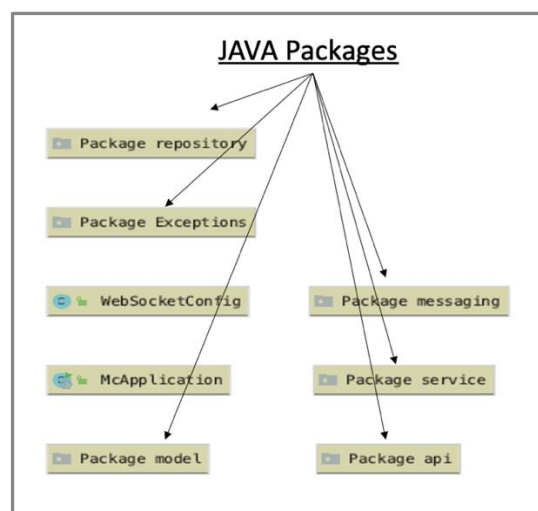


Figure 7. Java packages

- In first package, we have following classes. The class McApplication basically starts the Spring Boot Application. Whereas WebSocketConfig class is used for configuring socket programming:

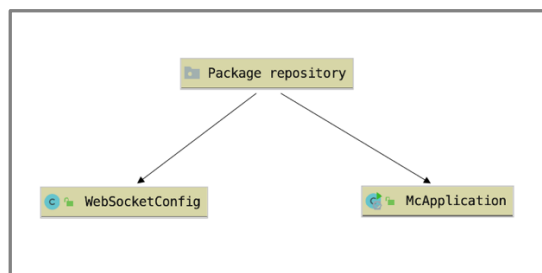


Figure 8. Package repository

- In second package which is known as package exceptions, we have following classes, these classes basically gives appropriate exceptions :

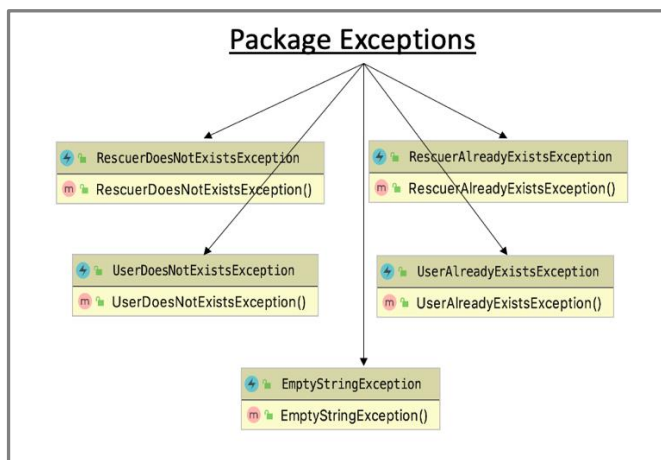


Figure 9. Exception package

- In the third package which is known as package api, we have the API classes.

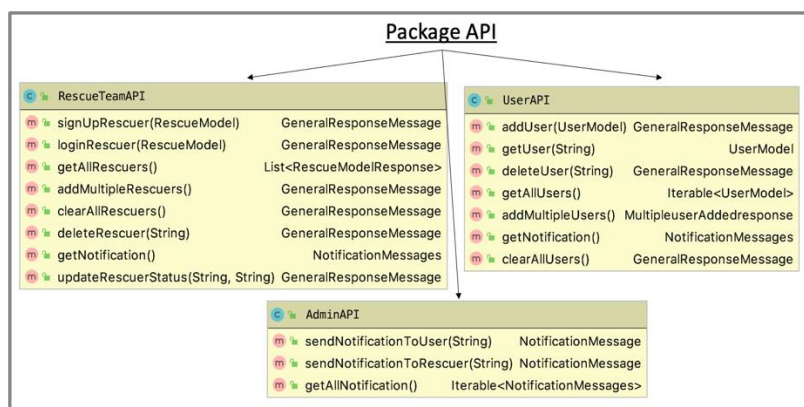


Figure 10. API package

- In the fourth package which is known as Model package we have model classes:

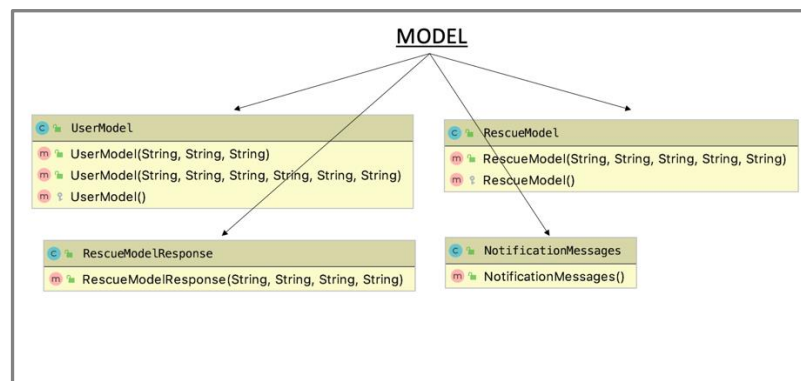


Figure 11. Model package

- At last we have service package, in which we have service classes, which basically is used to add a backend logic and to connect with the database.

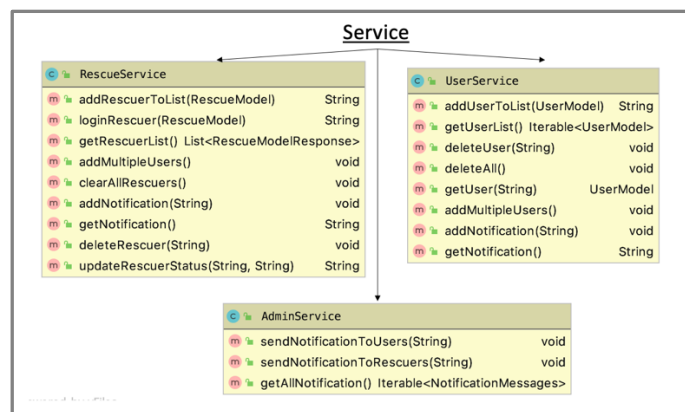


Figure 12. Service package

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

8.2. Angular Code Details

In Front end we have multiple components that basically helps to navigate and perform different task. The major idea to decouple the application into components is to make the code easy to maintain and debug. There are three different packages which are being discussed below:

- Firstly we have admin component. Which basically takes care of the admin pages and contains all the front end logic for the admin.

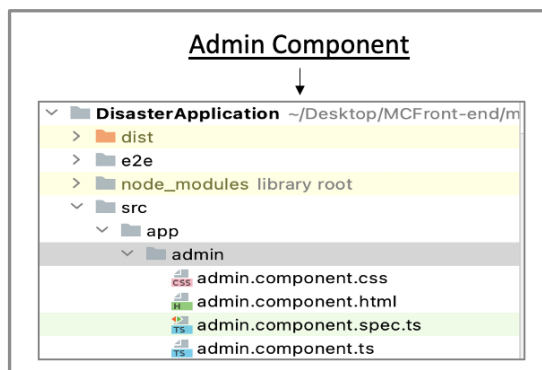


Figure 13. Front-end admin component

- Secondly we have User component. It manages the user actions and logics.

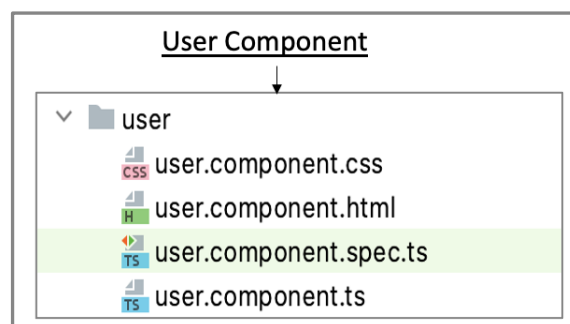


Figure 14. Front-end user component

- Thirdly we have Rescuer component that manages the rescuer pages.

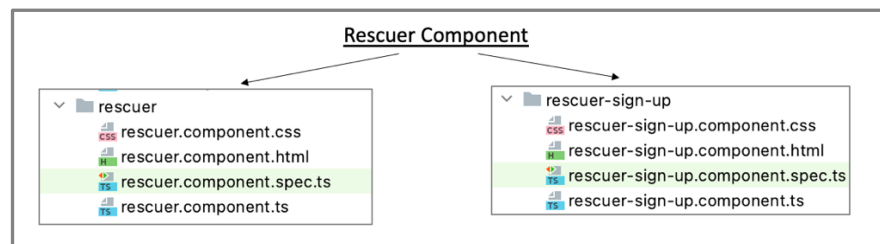


Figure 15. Front-end rescuer component

8.3. Package the Application in a Docker Container

We have two containers that are required to run the application. One container is required to run the front end [Angular 8.0] and the other one is needed to run the backend application [Spring boot]. Let's see the step-by-step procedure for both of them.

8.3.1. Back-End:

The following steps are required to package the backend application in the docker container:

- Firstly, make sure docker cli is up and running in the local.
- Open terminal /command line in the folder where the docker file is present.
General location for docker file ~ *MCPProject/Dockerfile*
- Now on the terminal run this command:
docker build . --tag mobilecomputing:latest
- After the successful execution of the above command the docker container will be created.

8.3.2. Front-End:

The following steps are required to package the frontend application in the docker container:

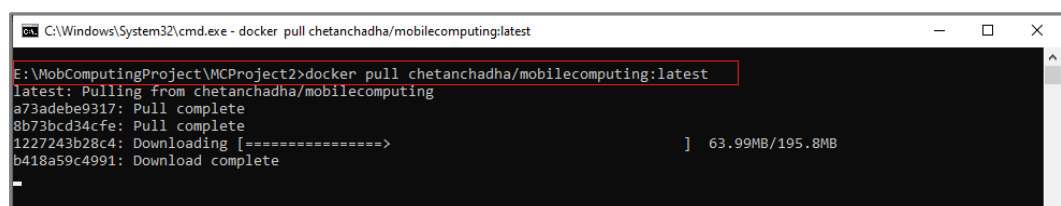
- Firstly, make sure docker cli is up and running in the local.
- Open terminal /command line in the folder where the docker file is present.
General location for docker file ~ *DisasterApplication/Dockerfile*
- Now on the terminal run this command:
docker build . --tag mobile-computing-front-end:latest
- After the successful execution of the above command the docker container will be created.

8.4. Steps to run the application

8.4.1. 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>”



```
C:\Windows\System32\cmd.exe - docker pull chetanchadha/mobilecomputing:latest
E:\MobComputingProject\MCPProject2>docker pull chetanchadha/mobilecomputing:latest
latest: Pulling from chetanchadha/mobilecomputing
a73adebe9317: Pull complete
8b73bcd34cfe: Pull complete
1227243b28c4: Downloading [=====>] 63.99MB/195.8MB
b418a59c4991: Download complete
```

Figure 16. Pulling docker image for backend application

- Once the image is downloaded

```
C:\Windows\System32\cmd.exe

E:\MobComputingProject\McProject2>docker pull chetanchadha/mobilecomputing:latest
latest: Pulling from chetanchadha/mobilecomputing
a73adebe9317: Pull complete
8b73bcd34cfe: Pull complete
1227243b28c4: Pull complete
b418a59c4991: Pull complete
Digest: sha256:da50c373d1f6f0b793d25964b2ee7fd2a493ab7f88d8db56baa7d681c43394f3
Status: Downloaded newer image for chetanchadha/mobilecomputing:latest
docker.io/chetanchadha/mobilecomputing:latest
```

Figure 17. Docker image downloading

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

```
C:\Windows\System32\cmd.exe

E:\MobComputingProject\McProject2>docker images
REPOSITORY                                TAG      IMAGE ID      CREATED        SIZE
chetanchadha/mobile-computing-front-end   latest   6c9e460a0c06  26 hours ago  1.14GB
chetanchadha/mobilecomputing              latest   6675e8c23c19  2 days ago    542MB
mycloudproject                            latest   a5702786db02  3 months ago  221MB
exercise2consoleapp1.azurecr.io/mycloudproject latest   a5702786db02  3 months ago  221MB
<none>                                     <none>   92e60ded3c6d  3 months ago  1.23GB
mycloudproject                             dev      13f53d289421  3 months ago  190MB
maxminvalueexp                             latest   b1b9a53fc054  4 months ago  216MB
```

Figure 18. Docker images

- Now to run the docker container, write the command “*docker run -p 9902:9902 <IMAGE ID>*” like shown below

```
C:\Windows\System32\cmd.exe - docker run -p 9902:9902 6675e8c23c19

E:\MobComputingProject\McProject2>docker run -p 9902:9902 6675e8c23c19
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/app.jar!/BOOT-INF/lib/logback-classic-1.2.3.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/app.jar!/BOOT-INF/lib/log4j-slf4j-impl-2.13.3.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation.
SLF4J: Actual binding is of type [ch.qos.logback.classic.util.ContextSelectorStaticBinder]

:: Spring Boot ::
(v2.4.1)

2021-02-02 23:05:37.549 INFO 1 --- [main] com.networking.mc.McApplication : Starting McApplication v1.0.0-SNAPSHOT
Using Java 15.0.2 on c365a61bdebd with PID 1 (/app.jar started by root in /)
2021-02-02 23:05:37.554 INFO 1 --- [main] com.networking.mc.McApplication : No active profile set, falling back to
default profiles: default
2021-02-02 23:05:39.088 INFO 1 --- [main] .s.d.r.c.RepositoryConfigurationDelegate : Bootstrapping Spring Data JPA repository
ies in DEFAULT mode.
2021-02-02 23:05:39.340 INFO 1 --- [main] .s.d.r.c.RepositoryConfigurationDelegate : Finished Spring Data repository scanning
g in 241 ms. Found 3 JPA repository interfaces.
2021-02-02 23:05:39.950 INFO 1 --- [main] trationDelegate$BeanPostProcessorChecker : Bean 'org.springframework.hateoas.conf
```

Figure 19. Running docker image

```
C:\Windows\System32\cmd.exe - docker run -p 9902:9902 6675e8c23c19

WARNING: Use --illegal-access=warn to enable warnings of further illegal reflective access operations
WARNING: All illegal access operations will be denied in a future release
2021-02-02 23:05:41.407 INFO 1 --- [main] org.apache.jasper.servlet.TldScanner : At least one JAR was scanned for TLDs yet contained no TLDs. En
nable debug logging for this logger for a complete list of JARs that were scanned but no TLDs were found in them. Skipping unneeded JARs during scanning can im
prove startup time and JSP compilation time.
2021-02-02 23:05:41.721 INFO 1 --- [main] o.a.c.c.C.[Tomcat].[localhost].[/] : Initializing Spring embedded WebApplicationContext
2021-02-02 23:05:41.724 INFO 1 --- [main] w.s.c.ServletWebServerApplicationContext : Root WebApplicationContext: initialization completed in 4021 m
s
2021-02-02 23:05:41.892 INFO 1 --- [main] com.zaxxer.hikari.HikariDataSource : HikariPool-1 - Starting...
2021-02-02 23:05:42.381 INFO 1 --- [main] com.zaxxer.hikari.HikariDataSource : HikariPool-1 - Start completed.
2021-02-02 23:05:42.392 INFO 1 --- [main] o.s.b.a.h2.H2ConsoleAutoConfiguration : H2 console available at '/h2-console'. Database available at
jdbc:h2:mem:testdb
2021-02-02 23:05:42.817 INFO 1 --- [main] o.hibernate.jpa.internal.util.LogHelper : HH0000204: Processing PersistenceUnitInfo [name: default]
2021-02-02 23:05:42.881 INFO 1 --- [main] org.hibernate.Version : HH0000412: Hibernate ORM core version 5.4.25.Final
2021-02-02 23:05:43.802 INFO 1 --- [main] o.hibernate.annotations.common.Version : HCANN000001: Hibernate Commons Annotations {5.1.2.Final}
2021-02-02 23:05:43.207 INFO 1 --- [main] org.hibernate.dialect.Dialect : HH0000400: Using dialect: org.hibernate.dialect.H2Dialect
2021-02-02 23:05:44.024 INFO 1 --- [main] o.h.e.t.j.p.i.JtaPlatformInitiator : HH0000490: Using JtaPlatform implementation: [org.hibernate.en
gine.transaction.jta.platform.internal.NoJtaPlatform]
2021-02-02 23:05:44.036 INFO 1 --- [main] j.LocalContainerEntityManagerFactoryBean : Initialized JPA EntityManagerFactory for persistence unit 'def
ault'
2021-02-02 23:05:44.719 WARN 1 --- [main] jpaBaseConfiguration$JpaWebConfiguration : spring.jpa.open-in-view is enabled by default. Therefore, data
base queries may be performed during view rendering. Explicitly configure spring.jpa.open-in-view to disable this warning
2021-02-02 23:05:45.173 INFO 1 --- [main] o.s.s.concurrent.ThreadPoolTaskExecutor : Initializing ExecutorService 'applicationTaskExecutor'
2021-02-02 23:05:46.282 INFO 1 --- [main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat started on port(s): 9902 (http) with context path ''
2021-02-02 23:05:46.297 INFO 1 --- [main] com.networking.mc.McApplication : Started McApplication in 9.584 seconds (JVM running for 10.369
s)
```

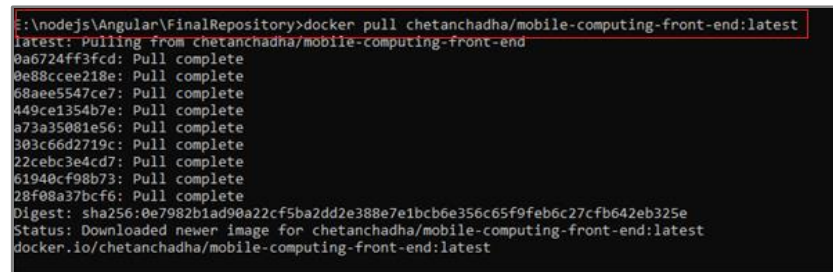
Figure 19. Application started

- Here we can see that the Spring Boot has started, and the server Tomcat is running on port 9902.

8.4.2. Front-End

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

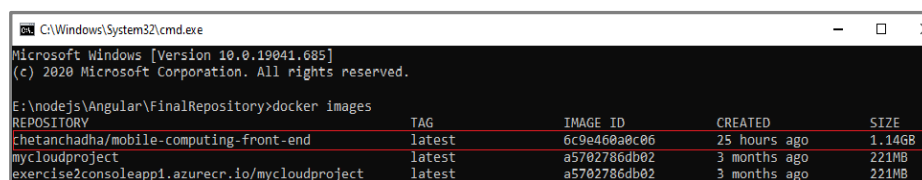
- Open the terminal in linux or cmd in Windows.
- First pull the docker image to your local machine using the command “docker pull <name of the docker image>”



```
E:\nodejs\Angular\FinalRepository>docker pull chetanchadha/mobile-computing-front-end:latest
latest: Pulling from chetanchadha/mobile-computing-front-end
0a6724ff3fcd: Pull complete
0e88ccee218e: Pull complete
68aee5547ce7: Pull complete
449ce1354b7e: Pull complete
a73a35081e56: Pull complete
303c66d2719c: Pull complete
22cebc3e4cd7: Pull complete
61940cf98b73: Pull complete
20f08a37bcf6: Pull complete
Digest: sha256:0e7982b1ad90a22cf5ba2dd2e388e7e1bcb6e356c65f9feb6c27cfb642eb325e
Status: Downloaded newer image for chetanchadha/mobile-computing-front-end:latest
docker.io/chetanchadha/mobile-computing-front-end:latest
```

Figure 20. Pulling docker image for frontend application

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

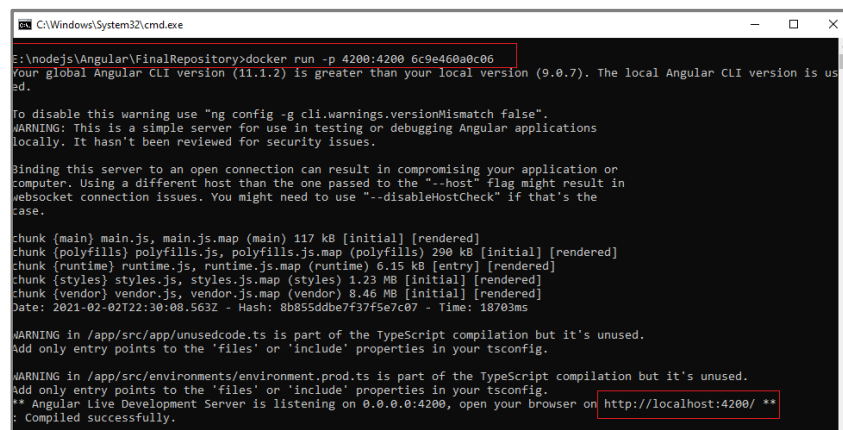


```
C:\Windows\System32\cmd.exe
Microsoft Windows [Version 10.0.19041.685]
(c) 2020 Microsoft Corporation. All rights reserved.

E:\nodejs\Angular\FinalRepository>docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
chetanchadha/mobile-computing-front-end   latest             6c9e460a0c06       25 hours ago       1.14GB
mycloudproject       latest             a5702786db02       3 months ago       221MB
exercise2consoleapp1.azurecr.io/mycloudproject latest             a5702786db02       3 months ago       221MB
```

Figure 21. Docker images

- Now to run the docker file, write the command “docker run -p 4200:4200 <IMAGE ID>” like shown below



```
C:\Windows\System32\cmd.exe
E:\nodejs\Angular\FinalRepository>docker run -p 4200:4200 6c9e460a0c06
Your global Angular CLI version (11.1.2) is greater than your local version (9.0.7). The local Angular CLI version is used.

To disable this warning use "ng config -g cli.warnings.versionMismatch false".
WARNING: This is a simple server for use in testing or debugging Angular applications locally. It hasn't been reviewed for security issues.

Binding this server to an open connection can result in compromising your application or computer. Using a different host than the one passed to the "--host" flag might result in websocket connection issues. You might need to use "--disableHostcheck" if that's the case.

chunk (main) main.js, main.js.map (main) 117 kB [initial] [rendered]
chunk (polyfills) polyfills.js, polyfills.js.map (polyfills) 290 kB [initial] [rendered]
chunk (runtime) runtime.js, runtime.js.map (runtime) 6.15 kB [entry] [rendered]
chunk (styles) styles.js, styles.js.map (styles) 1.23 MB [initial] [rendered]
chunk (vendor) vendor.js, vendor.js.map (vendor) 8.46 MB [initial] [rendered]
Date: 2021-02-02T22:30:08.563Z - Hash: 8b855ddb7f37f5e7c07 - Time: 18703ms

WARNING in /app/src/app/unusedcode.ts is part of the TypeScript compilation but it's unused.
Add only entry points to the 'files' or 'include' properties in your tsconfig.

WARNING in /app/src/environments/environment.prod.ts is part of the TypeScript compilation but it's unused.
Add only entry points to the 'files' or 'include' properties in your tsconfig.
** Angular Live Development Server is listening on 0.0.0.0:4200, open your browser on http://localhost:4200/ **
: Compiled successfully.
```

Figure 22. Starting docker front-end image

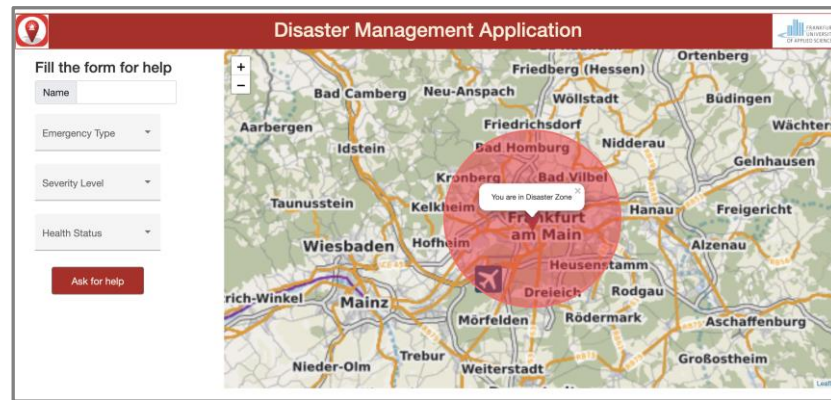


Figure 23. Front-end view

- 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

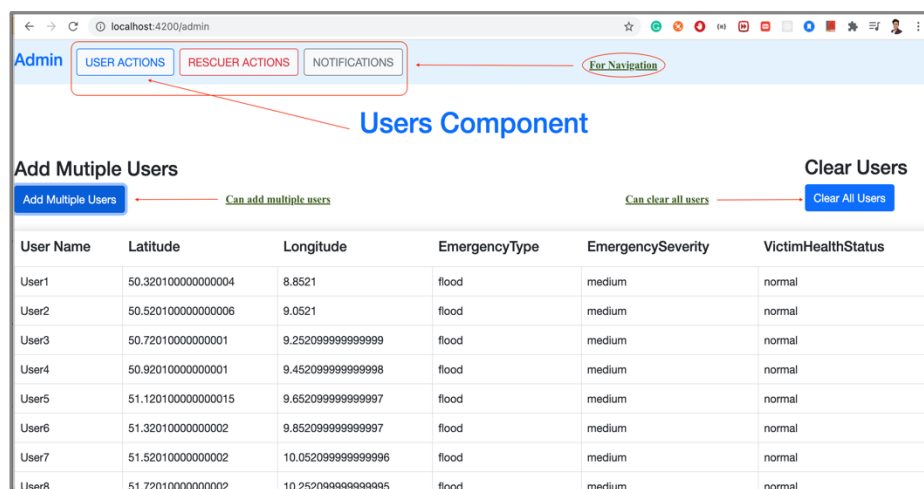


Figure 23. Front-end admin page view

9. 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.

9.1. 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 not 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.

9.1.1. 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 and other online map providers.

9.2. Notification Center

Notification center is core of any user interaction application. It itself is a complete project and requires lot of work. Keeping time constraint in mind we didn't build another application for notification center, but we do give a small work around for that. We added a few lines of code where we are storing the notifications from the admin. On the front-end we are polling notifications after every 5 seconds. Disadvantage to this is that every user will be notified a personal notification for a specific user cannot be triggered. On the other hand, admin has to manually remove the notification from the backend as it will not be auto removed.

9.3. Accessing APIs after Containerization

So, once we containerized the application and ran it in the docker, unfortunately we were unbale to access it from our web browsers. The reason behind it was that the application running in docker are using the docker ports and we need to map those ports to our local system ports. For this we use the mapping docker command and were able to resolve this issue.

9.4. Testing APIs

Testing is very important especially when backend is considered. In the beginning it was consuming lot of our time to test the APIs. We were first creating a front end and then using it testing the backend functionality. Since for testing there was a dependency on the front end. To solve this problem, we used postman application. A

full fledge learning curve was there but in a very small span of time we learnt about the application and tested our backend.

10. 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.

10.1. 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 through 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.

10.2. Displaying the Rescuers reaching to users on Map

It would be more convenient 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 show the distance between the rescuers and the users for further extension.

10.3. 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.

10.4. Timer should be set for every 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 demonstrate the concept. But this value should be setup by the rescuers.

10.5. Security

In the backend a security check can be included using spring boot security dependency. In that way anyone accessing the APIs will be required to attach a token while making an API call.

Secondly at this moment anyone can access the admin page. In the next step we need to add a login page on the admin side and that password needs to be stored in the encrypted form.

Thirdly the rescuer login information is stored in the data base without using any encryption technique. In the next step this information needs to be encrypted.

10.6. Caching the login details

As of now when a rescuer login we are not caching or storing its login details in the browser. In the next phase this information should be cached so once rescuers login he/she should not be required to login again,

10.7. Storing the chat data

As of now the chat data is not being stored anywhere. So a new rescuer joining the chat will not be able to see the old chat. In the next step we need to store this information in the backend so a new rescuer can access the previous chat history as well.

--

11. Acknowledgements

I would like to thank Prof Ulrich Trick 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 leaflet, Postman and spring boot community. We are thankful to them.

12. 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/>.