# FRANKFURT UNIVERSITY OF APPLIED SCIENCES

**Faculty 2: Computer Science and Engineering**



**Subject:** Mobile Computing – WS 2020/21 (Prof. Ulrich Trick)

### **Topic: Disaster Management Application**

**Author:** Group 2

Lakshmi Mounika Kolisetty (1324400)

Treesa Maria Thomas (1324484)

Raghavendra Yarlagadda (1324413)

Date of submission: 8th March 2021

**Table of Contents**

Table of Contents

1.Introduction………………………………………………………………………………...4

2.Architecture Overview…………………………………………………………………….5

2.1 Programming Languages …………………………………………………………...6

2.2 Middleware Systems ………………………………………………………………...6

3.UI Design…………………………………………………………………………………..7

3.1 Rescuer Access………………………………………………………………………7

3.2 Ticket Creation………………………………………………………………………..9

3.3 Admin Access.…………….…………………………………………………………10

4.Application Design………………………………………………………………………..11

4.1 Rescuer/Admin Login……………………………………………………………….11

4.2 Victim Helpline……………………………………………………………………….12

4.3 Ticket Allocation……………………………………………………………………..12

5.Role of Database in Web Application…………………………………………………..14

5.1 Rescuer & Admin Table……………………………………………………………..14

5.2 Victim Mandatory Items Table………………………………………………………15

6.Code Snippets ……………………………………………………………………………15

7.Docker Containerization…………………………………………………………………16

7.1 Docker File creation for Front-end…………………………………………………17

7.2 Docker File creation for Back-end………………………………………………….18

8.Installation Guide…………………………………………………………………………19

Conclusion…………………………………………………………………………………..21

Bibliography…………………………………………………………………………………22

Declaration…………………………………………………………………………………..23

# Appendix

**Authorship:**

Front-end: Raghavendra Yarlagadda (1324413)

Back-end: Lakshmi Mounika Kolisetty (1324400)

Database setup and Docker Containerization: Treesa Maria Thomas (1324484)

**ABSTRACT**

Disasters (Natural and Man-Made) have posed a threat to humanity since the dawn of time. Many countries are vulnerable to disasters as a result of their geographic location, changing climate and human actions. In addition, the countries lack an adequate emergency preparedness mechanism for coping with disasters. In case of a disaster there is a high possibility to existing communication infrastructure getting destroyed or heavily damaged. Therefore, an emergency communication infrastructure is constructed from battery-supplied multi-radio wireless outdoor-routers which are deployed by first responders in the affected region. The outdoor-routers are connecting to each other and are building a wireless mesh network (WMN)- based disaster network, which provides the basis for an IP-based communication. This paper highlights the design and development of a disaster management application which will be provided to the end-user groups of the WMN-based disaster network.

# 1. Introduction

Natural disasters such as Tsunami, Earthquakes, volcanic eruptions etc. and man-made disasters are associated with power failures in industries. These disasters affect the ecosystems and the people living in it. Disasters often cause havoc on existing communication infrastructure. Due to this it is impossible to coordinate the rescuers and volunteers in order to find the victims. As a consequence, it is concluded that a functioning communication infrastructure is essential for the rescue of victims and the management of rescue teams. The disaster network is constructed from battery-supplied wireless outdoor routers, which are deployed in the disaster area and together establish a wireless mesh network (WMN) see Figure 1.

Diagram

Description automatically generated

Figure 1 Network Infrastructure of a WMN based network [1]

The WMN-based disaster network makes use of the NFV (Network Function Virtualization) concept to offer multiple network functionalities and services to end-users. The WMN-based disaster network is intended to be utilized by various end-user groups. The groups include official government groups and also ordinary citizens.

The process of disaster evacuation must be quick, accurate, reliable, efficient, integrated, and accountable and the rescue of victims will be much easier if they are able to communicate through a medium. There comes the disaster management application. This application will be provided to end-user groups of the WMN-based disaster network and it helps in facilitating the communication. The application is built using HTML, CSS and ReactJS programming languages, using IntelliJ software, using NGINX as a web server, and also using PostgreSQL as a database.

This application helps in collecting and sharing disaster information collected by users in real time and ensuring the safety of victims. Furthermore, Disaster Management rescuers can monitor the position and can search for victims faster.

# 2. Architecture Overview

The web application architecture describes the relationships between the applications, the database and the middleware systems. It ensures that many applications can run at the same time. Any standard Web application consists of client-side and server-side.

Diagram

Description automatically generated

Figure 2 General Architecture of the web application

The front-end, often referred to as the client-side, responds to user input and interacts within the browser. The main goal of the client is to collect data from users. The front-end part is generally written in HTML, CSS and JavaScript languages. The back-end of an app is often referred to as the server-side. A server can run any application that can respond to HTTP requests. The server-side is in charge of generating the requested page as well as storing various types of data (text, images, files, etc.), such as user profiles and user feedback. The end-user is never aware of it. Back-end can be done in any of the following languages: Python, Java, Ruby, PHP.

There are at least two sections to the server component: app logic and database (MySQL, SQL, PostgreSQL, Oracle). The former serves as the main control center for the web application, while the latter holds all of the application's permanent data. The architecture implemented in our disaster management application is given in Figure 2.

**2.1 Programming Languages**

The languages implemented in our application are HTML (Hypertext Markup Language), CSS (Cascading Style Sheets), ReactJS and Spring Boot. HTML is a standard markup Language for creating structures and views on webpages. For this application the latest HTML5 version is used. CSS is a language that can describe elements of the HTML language displayed on the screen. CSS provides visual design in HTML language so that web pages can look beautiful CSS is designed to separate presentation and content. The separation results in layout, color and font aspects.CSS is designed to separate presentation and content. The separation results in layout, color and font aspects. This separation can increase the flexibility and control of presentation characteristics and allow HTML pages to access CSS pages separately. In addition, JavaScript is a programming language that can make a website more interactive. ReactJS is a JavaScript library that can be used to build a high-performance UILayer. VSCode is used as the text editor to perform coding for front-end.

Spring Boot is a microservice-based Java web platform that is open source. With its prebuilt code, the Spring Boot system provides a fully production-ready environment that is completely configurable. Developers can create a completely enclosed framework using the microservice architecture, which includes embedded application servers. IntelliJ is a text editor for back-end development.

**2.2** **Middleware Systems**

Tomcat acknowledges client requests, dynamically compiles a container-managed Java class to handle the request as specified in the Context specific application, and returns the result to the client using Java Servlet and JSP APIs. This method of dynamic content generation allows for extremely fast, threaded, and platform-independent request processing.

The key purpose of a web server is to accept and fulfill client requests for static content from a web site.  The client is almost always a browser or a mobile device, and the request is a Hypertext Transfer Protocol (HTTP) message, much like the web server's response. When deployed in front of web and application servers, NGINX is an extremely powerful reverse proxy and load balancer, with translation modules for a variety of application server types.

It's a remarkably robust database management system, with more than two decades of group growth to thank for its high levels of resiliency, honesty, and correctness. Many web, mobile, geospatial, and analytics applications use PostgreSQL as their primary data store or data warehouse.

# 3. UI Design

# The user interface of our web application provides a login page for respective roles such as admin and rescuer. After logging in with their respective credentials they have access to the system's functions. Authentication services are required in order to login and display the victim’s information. The UI also consists of a victim helpline page which a victim can seek help by filling a form.

# 3.1 Rescuer Access

# Each assigned rescuer will be given with a username and password. After login with their respective credentials, rescuer get access to the information like details of victims, InProgress tickets (which are assigned to the respective rescuer who logged in) and Closed tickets - tickets resolved by rescuer who logged in (which can also be closed by admin). The User Interface for rescuer login is shown in Figure 3.

# Once the rescuer is logged in, the login page is directed to the dashboard. The dashboard displays the ticket information such as Ticket Id, Ticket Status (InProgress and Closed), Ticket allocated to which rescuer, date and time of ticket created and the required items.

# To get details of the victim, click on desired Ticket Id displayed on dashboard. The Ticket details page is opened which contains Ticket Id, Ticket Status, Ticket allocated to which rescuer, date and time of ticket created, download attachments (uploaded by victim), Emergency contact details and Victim details.

# Emergency contact details has:

# Emergency Address

# Emergency Contact Age

# Emergency Phone Number

# Lat, Long (Latitude, Longitude)

# Victim details has:

# Victim Name

# Victim Age

# Victims Around – Number of victims in the immediate surroundings.

# Additional Information – Requested as an emergency need by victim such as water, food, medical kit.

# Graphical user interface, application Description automatically generated

# *Figure 3 User Interface for Rescuer Login*

# Left side bar has Open Tickets, Closed Tickets and InProgress Tickets. Open Tickets are only visible to admins. Closed Tickets displays the tickets which are closed by the admin/rescuer has the basic information about the ticket like Ticket id, Ticket Status (Close), Ticket allocated to which rescuer, date and time of ticket created and the Action.

# InProgress Tickets contains the tickets that are in process. It also displays the details of the ticket such as Ticket id, Ticket Status (InProgress), Ticket allocated to which rescuer, date and time of ticket created and the Action. Action can be performed to the InProgress ticket is Resolve. Once the ticket is resolved, rescuer can use resolve button and the whole ticket is moved to closed tickets. After the session is completed, the rescuer can log out using Logout button.

# 3.2 Ticket Creation

# As shown in Figure 3, Victim can create a ticket by clicking on the + button. It displays the following page (Figure 4) where victim should fill a form.

# Graphical user interface, application Description automatically generated

# *Figure 4 Webpage to create a ticket*

# Victim Details:

# Victim Name – Name of the Victim should be written in text.

# Victim Age – Age of the Victim should be a number.

# Required Items – Things which are needed in the affected area such as medical kit, water or food.

# Emergency Details:

# Contact Age – It should be in number.

# Contact Name

# Contact Address

# Phone Number

# Additional Information:

# Additional Information – Victim can include a description of the surroundings and any details that can help to rescue them.

# Victims Around – Number of persons in the immediate surroundings.

# Upload Attachment – Victim can be able to attach an image or video of the affected area.

# After admin/rescuer logged in, the dashboard displays all the tickets. By clicking on any one of the tickets, tickets details page is displayed as in Figure 5

# 

# *Figure 5 Ticket Details Page*

# 3.3 Admin Access

# Using the same page shown in Figure 3, Admin can also login. Admin has the privilege to view the tickets in detail, status of the tickets and to add another Admins/Rescuer. After logging in, Dashboard is displayed. Dashboard contains same information as the Dashboard of Rescuers access page, but it contains all the types of tickets (Open, Closed and InProgress). The Dashboard of Admin is shown in Figure 5.

# Graphical user interface Description automatically generated

# *Figure 6 UI of Admins Dashboard*

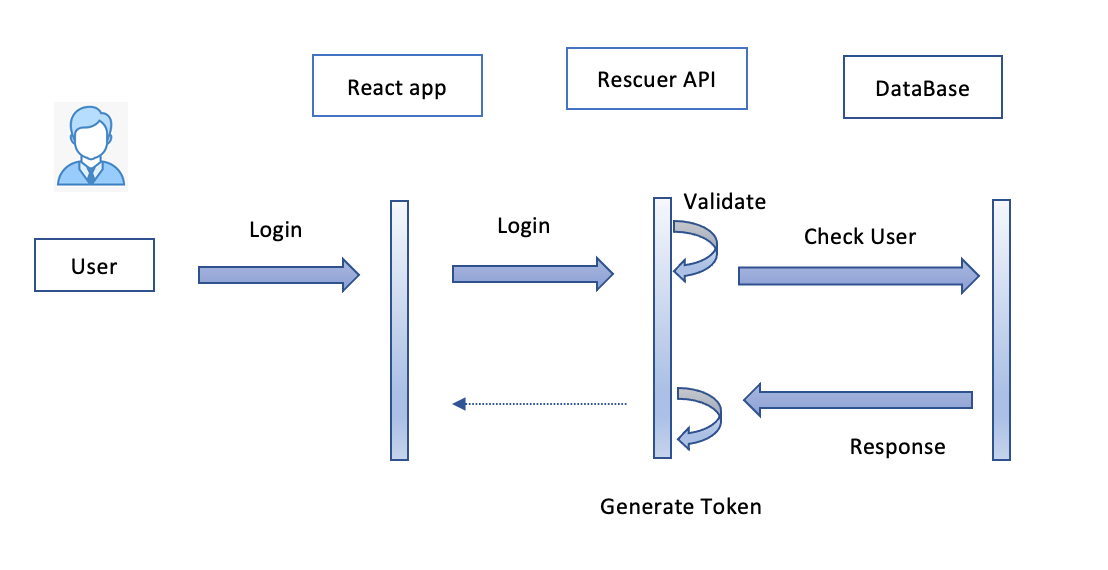
# The left-side panel contains Open Tickets, Closed Tickets, InProgress Tickets, Add Admin and Add Rescuer. Open Tickets has all the tickets which are still not assigned to any rescuer. It views all the basic information along with action (Assign). The assign button will allocate the ticket to random rescuer. Another way of assigning tickets is doing automatically for every 15 mins while the first ticket is assigned in 5 mins after executing the application.

# Closed and InProgress Tickets contains same information as the respective tabs of Rescuers. Action (Reopen) in Closed Tickets is assigned to same rescuer who closed the ticket. Action (Resolve) in InProgress Tickets, works in such a way that after resolving the whole ticket it is moved to Closed Tickets. Admin has special role to add new Admin/Rescuer by setting credentials to them.

# 4. Application Design

**4.1 Rescuer/Admin Login**

The client-side of the application shows a login page for rescuers/admins and a helpline page for victims. When a rescuer/admin tries to login with the required credentials a request will be sent to the application from the browser. The application forwards the request to the rescuer API to validate the credentials given by the rescuer. Next, the rescuer API approaches the database with a request to authenticate the credentials. Then the database server examines these credentials and if it matches to the one which are in the server it responds with an OK response to the API. The API then generates a token to start the session for the rescuer/admin and this session will be active for few minutes. The sequence flow is represented in the Figure 6.



*Figure 7 Sequence flow of the Rescuer/Admin login*

**4.2 Victim Helpline**

As mentioned, the client-side also contains a helpline page for victims. In this page we setup a helpline page where a victim in need can reach out for help by filling in the details. The details in which the victim can fill are the name, age, current location, number of other victims around and the details of emergency contact. After filling all these details, the browser sends the details to the API as a request to create a ticket. The API processes this request and after storing the details in the database a ticket will be created. The API logs the ticket ID in the allocation service and sends the response to the browser as success. The flow is described in Figure 7.

Diagram

Description automatically generated

*Figure 8 Flow diagram for ticket generation*

**4.3 Ticket Allocation**

The status of the tickets is defined as open, In-Progress, close and re-open. As soon as the ticket is created, the status is automatically assigned as open. If the status of the ticket is open for 15 minutes, then a rescuer will be automatically assigned to it and the status will be assigned as In-Progress. If a rescuer finishes their task by identifying the victim, he/she can close the ticket manually. By any chance is the same ticket is re-opened it will be automatically assigned to the rescuer who previously closed the ticket. The sequence is shown in Figure 8.

Diagram

Description automatically generated

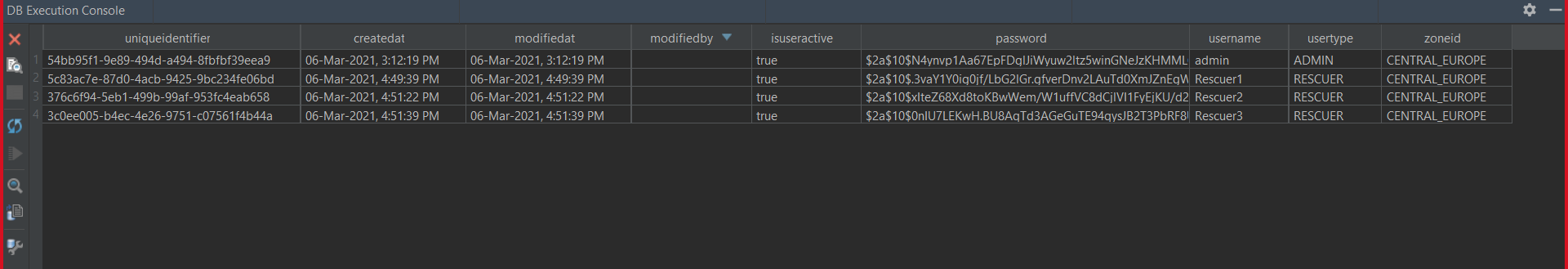
*Figure 9. Ticket allocation service*

**5. Role of Database in Web Application**

The role of database in the web application is to store the application’s permanent data. The data is stored in the form of tables. PostgreSQL is one of the most popular relational database management systems (RDBMSs) on the market. It’s also free and open source. It gives the users the ability to manage data regardless of how large the database is. Considering all these advantages we have used PostgreSQL for our database.

**5.1 Rescuer and Admin Table:**

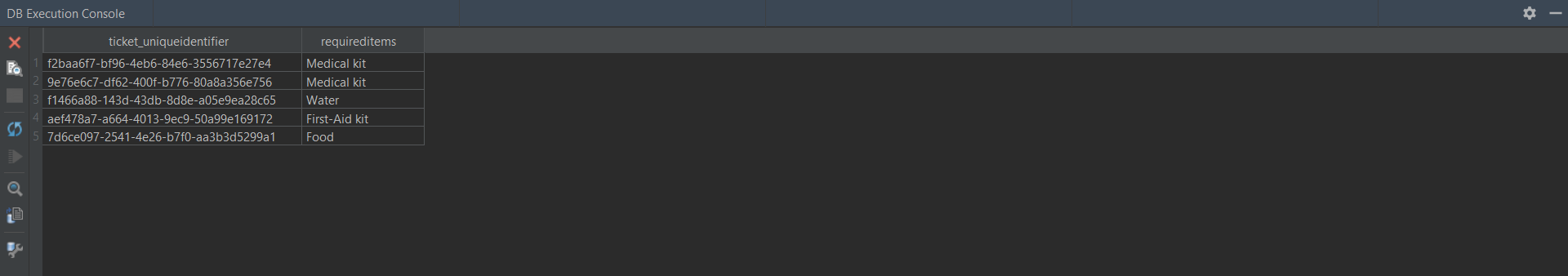
The table is shown in Figure 9. As already mentioned in the admin access page the admin has a special privilege to add another admin and rescuer. The added user types are stored in the table named rescuer. This rescuer table stores the random unique identifier which is unique for each user type and the password in encrypted format. It also stores the date and time (CENTRAL\_EUROPE) at which the user is added.



*Figure 10. Rescuer Table*

**5.2 Victim Mandatory Items Table:**

Figure 10 shows the victim mandatory items table in which it stores the required items which are inserted by victims while filling the form. For each required item a unique identifier is created and stored.

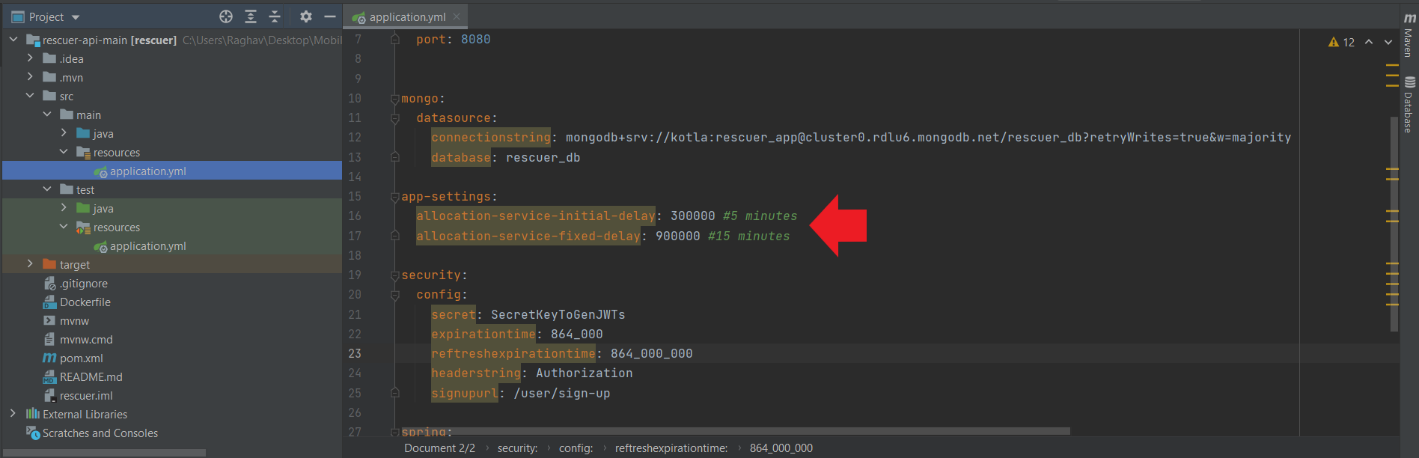


*Figure 11. Victim Mandatory Items Table*

Also, there are 3 more tables which are named as ticket, user ticket stats, victim attachments in the database. The ticket table shows all the details which are entered by the victim using the helpline page. Along with these details a unique identifier for each ticket and date and time at which it is created are also stored. Coming to the user ticket stats table it stores the ticket id and no of closed, opened, reopened tickets are currently there in the application. In the helpline page a victim can take a snap of the surroundings or record a video and upload it like an attachment with other details. The details of the attachment like attachment name, attachment type and the file are stored in the database table.

**6. Code Snippets**

As mentioned earlier, If the status of the ticket is open for 15 minutes, then a rescuer will be automatically assigned and for the first ticket the rescuer is assigned in 5 mins after executing the application. To change the time stamps of these ticket allocations, it is recommended to open the path (MobileComputing-Project-main\rescuer-api-main\src\main\resources\application.yml) as in below figure and change the time accordingly. Figure11 shows the actual implementation of the timestamps.

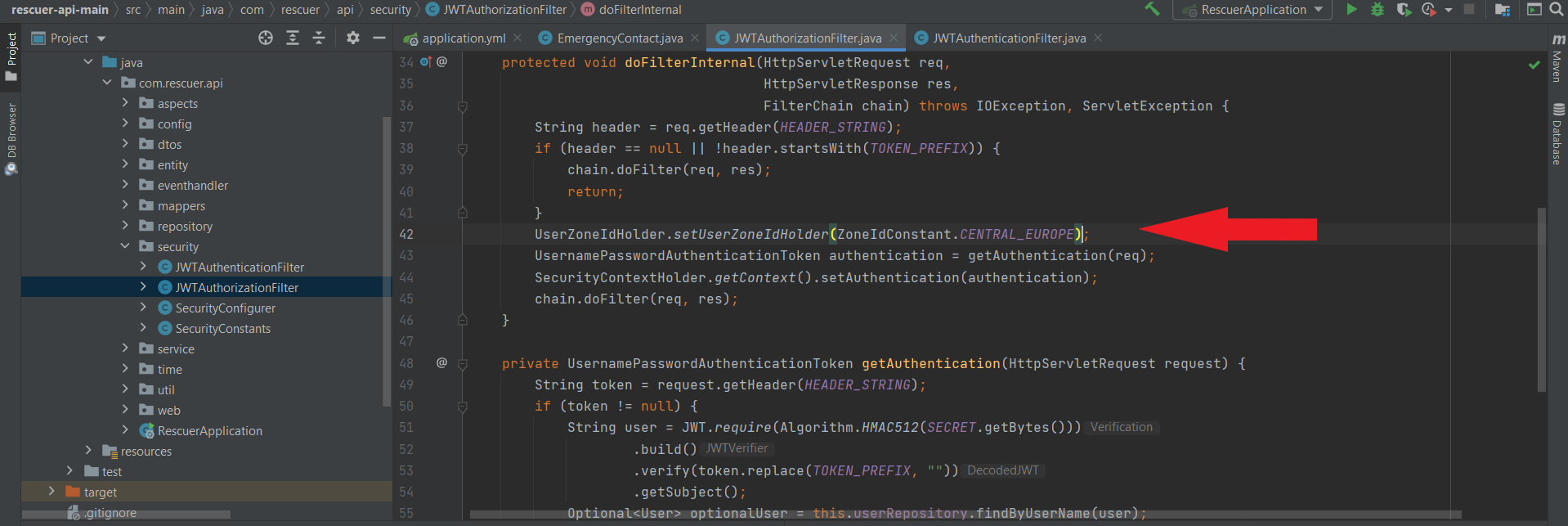


*Figure 12. Timestamps*

For every ticket creation the application creates a unique identifier and the date and time at which the ticket is created. The date and time are set as CENTRAL\_EUROPE standard. We can change this timing to our desired zone. To change the time zone, go to the path which is given below and change the desired time zone as shown in figure 12.

Path: MobileComputing-Project-main\rescuer-api-main\src\main\resources

Open the JWTAuthorizationFilter file and change the time zone.



*Figure 13. Time Zone*

**7.Docker Containerization**

In conventional software development, codes developed in one computing environment often comes across with bugs and errors when deployed in a different environment. This issue can be solved by deploying the Software through container in the cloud. Containerization helps development teams to work rapidly, deploy applications efficiently and perform on an enhanced rate. Containers provide a logical packaging mechanism in which applications can be segregated from the environment in which they currently work. This decoupling enables quick and consistent deployment of container-based applications, regardless of whether the target environment is a private data center, public cloud, or even the personal laptop of a developer. Containerization offers a clean separation of concerns, with developers focusing on the application logic and dependencies while, IT operations teams can focus on deployment and management without bothering with application details such as specific software versions and configurations specific to the app. All the above requirements can be best served by Docker. Docker enables developers to easily pack, ship, and run any application as a lightweight, portable, self-sufficient container, which can run virtually anywhere. The app first needs to be package up as Docker image, which can be test locally by using Docker for Windows. [2] [3]

**7.1 Docker File creation for Front-end:**

Create a file and name it as "Dockerfile".

The following ENVs are used for backend apis:

ENV REACT\_APP\_CREATE\_TICKET=http://localhost:8080/ticket/create-ticket

ENV REACT\_APP\_LOGIN\_URL=http://localhost:8080/login

ENV REACT\_APP\_GET\_ALL\_TICKETS=http://localhost:8080/ticket/all-tickets

ENV REACT\_APP\_GET\_OPEN\_TICKETS=http://localhost:8080/ticket/open-tickets

ENV REACT\_APP\_TICKET\_DETAILS=http://localhost:8080/ticket/

ENV REACT\_APP\_GET\_CLOSE\_TICKETS=http://localhost:8080/ticket/closed-tickets

ENV REACT\_APP\_GET\_IN\_PROGRESS\_TICKETS=http://localhost:8080/ticket/inProgress-tickets

ENV REACT\_APP\_ADD\_RESCUER=http://localhost:8080/admin/add-rescuer

ENV REACT\_APP\_ADD\_ADMIN=http://localhost:8080/admin/add-admin

ENV REACT\_APP\_ASSIGN\_TICKET=http://localhost:8080/admin/assign-ticket/

ENV REACT\_APP\_RESOLVE\_TICKET=http://localhost:8080/ticket/resolve/

ENV REACT\_APP\_REASSIGN\_TICKET=http://localhost:8080/ticket/reopen/

ENV REACT\_APP\_DOWNLOAD\_FILES=http://localhost:8080/file/download/

To build and run the frontend application use:

WORKDIR /app

COPY . .

RUN npm i && npm run build

NGINX open source dockerfile is created using:

FROM nginx:1.16.0-alpine

COPY --from=build /app/build /usr/share/nginx/html

RUN rm /etc/nginx/conf.d/default.conf

COPY nginx/nginx.conf /etc/nginx/conf.d

CMD ["nginx", "-g", "daemon off;"]

Following commands are used to create a docker image and run the front-end application:

* sudo docker build -t rescue-app-nginx .
* sudo docker run --init --rm -d -p 80:80 rescue-app-nginx

**7.2 Docker File creation for Back-end:**

A Docker file is a file that contains the guidelines for creating a Docker image. Create a file and name it as "Dockerfile".

Use following commands to install maven dependency as a plug in:

FROM maven:3.6.3-jdk-11 AS MAVEN\_BUILD

COPY pom.xml /build/

COPY src /build/src/

WORKDIR /build/

RUN mvn clean install -Dmaven.test.skip=true

Use the following commands to install JRE (The Java Runtime Environment, or JRE, is a layer of software that runs in the background of the operating system software on a computer and includes the class libraries and other resources that a Java program requires to run):

FROM openjdk:11.0.4-jre-slim

WORKDIR /app

COPY --from=MAVEN\_BUILD /build/target/rescuer-0.0.1-SNAPSHOT.jar /app/

ENTRYPOINT ["java", "-jar", "rescuer-0.0.1-SNAPSHOT.jar"]

Following commands are used to create a docker image and run the application:

* sudo docker build -t rescuer-app-service .
* docker run -e "SPRING\_APPLICATION\_NAME=rescuer-app-service" -e "server.port=8080" -p 8080:8080 --init --rm -d --name rescuer-service rescuer-app-service

**8. Installation Guide**

Below are the following steps to test the application.

1. Clone the following repository from GitHub.

<https://github.com/MounikaKolisetty/MobileComputing-Project>

1. Download and Install IntelliJ IDEA from the below link

<https://www.jetbrains.com/idea/>

1. Now open the folder rescuer-api-main in IntelliJ IDEA. Automatically all the required packages will be downloaded and installed in the IDE. Also download Java version 11 from the plugins
2. Open the command prompt and change the directory to MobileComputing-Project/rescuer-api-main and execute the following commands consecutively.
3. sudo docker build -t rescuer-app-service .
4. docker run -e “SPRING\_APPLICATION\_NAME=rescuer-app-service” -e ”server.port=8080” -p 8080:8080 --init --rm -d --name rescuer-service rescuer-app-service
5. Open a new cmd terminal and change the directory to MobileComputing-Project/rescuer-app-ui-main and run the following commands.
6. sudo docker build -t rescue-app-nginx .
7. sudo docker run --init --rm -d -p 80:80 rescue-app-nginx
8. Open a browser and search for localhost in the URL bar. Now the application is ready and can test it by creating a ticket by entering the required details.
9. To check the tables in database, download the database navigator plugin from the following link and install to IDE.

<https://plugins.jetbrains.com/plugin/1800-database-navigator>

1. Go to <https://id.heroku.com/login> and enter the below credentials

Email Address: r.dhawan27@hotmail.co.u

Password: RescuerApp@123

1. After logging in, click on the rescuer-app-project then go to the resources tab and click on Heroku Postgres add-ons. Now go to the Settings tab and click on view credentials.
2. On the left panel of IntelliJ IDEA, open the DB Browser tab and open a new PostgreSQL connection. Now enter the values of Host, Database, User, Password from the Heroku website.

Note: These details frequently change as it is a free account. If the application come across any error, make sure to update the values from the Heroku website in PostgreSQL connection.

1. After entering the details click apply and now the connection to the database has been established.
2. To see the tables, Expand the schemas and then expand the public and now the Tables are visible.

# Conclusion

In this paper the design of a web application which is useful for victims to communicate with the official government groups during a disaster has been successfully presented. As we know that we can’t predict when a natural disaster will occur and when a man-made event can take place. So, the need for disaster preparedness is extremely important in the global context. Without a systematic approach it is hard to tackle these kinds of disasters. In this situation an emergency communication infrastructure can be constructed from battery-supplied multi-radio wireless outdoor-routers which can be deployed by first responders in the affected region. The routers together form a WMN network and a web application can be provided to the end usergroups of WMN based disaster network.

# Bibliography

# G. Frick, A. Paguem Tchinda, U. Trick and A. Lehmann (2018). Distributed NFV Orchestration in a WMN-based Disaster Network. Centre for Security, Communications and Network Research

# "What is docker and why it is so darn popular,"

# [Online]. Available:

# https://www.zdnet.com/article/what-is-docker-andwhy-is-it-so-darn-popular/.

# 3) "CONTAINERS AT GOOGLE," [Online].

# Available: https://cloud.google.com/containers.

# Declaration

We confirm that the work for the following paper with the title: "Disaster Management Application " was solely undertaken by us and that no help was provided from other sources as those allowed. All sections of the paper that use quotes or describe an argument or concept developed by another author have been referenced, including all secondary literature used, to show that this material has been adopted to support our thesis.

Lakshmi Mounika Kolisetty (1324400)

Treesa Maria Thomas (1324484)

Raghavendra Yarlagadda (1324413)

Signature

Date: 08.03.2020