Chatbot Report

Elective Project 1

Supervisor:

Prof. Dr.-Ing. Karin Landenfeld

Authors:

Alhajras Algdairy, Saleh Gamil, Atif Saeed

Content

1 Introduction.

During the rapid growth of online shopping, E-commerce business, and human-internet interaction many problems have been rising, one of the problems is the customer service.
Human support is needed to answer questions and to provide the right support for their service or products.

This project aims to help solve this problem, and more specifically for, Hochschule für Angewandte Wissenschaften (HAW) website, where the main goal is to provide a Chatbot or smart widget to answer the frequently asked question and to redirect the students to links based on their search on the Chatbot.

Before getting into any further details a brief explanation of the Chatbot must be introduced.

A Chatbot or an Artificial Conversational Entity is a software which communicates with the human by using a text or speech. The intention behind this software is to simulate how the human behaves by providing the right answers to the users, whether by introducing the services or redirecting them to links.

This Chatbot is going to be used as a prototype inside the HAW website the English version, and proper testing cases must be conducted.

2 Requirements.

For the sake of building the Chatbot, some preconditions must be met, and some goals and milestones must be set, the essential requirements for this software to function are:

- Should support the English and German languages.
- Scalable and easy to setup.
- Distributed and independent of the HAW website.
- Smart algorithm for searching inside the HAW website articles.
- Customizable and can be adjusted based on the user desire.
- Good infrastructure for further improvements, enhanced with the newest technologies.

3 Implementation and Setup.

To achieve those requirements, the following technologies have been used in this project:

Backend

- Java 8.
- Hibernate (MySQL).
- Spring boot framework.
- Flyway (MySQL).
- Gradle.
- Elasticsearch.
- Lombok project.

Frontend

- HTML.
- JavaScript.
- jQuery.
- Vue framework.
- Thymleaf.

3.1 Setup

To setup the project and to be able to start, the main IDE that is going to be used is Eclipse, and in this project Eclipse (Photon) has been used. Moreover, Java 8 has been chosen since it provides some features like function interfaces, lambda expressions, and a different approach to null handling Java SE Development Kit 8.

The next step, a building tool is needed and in this project, Gradle was chosen, in order to use Gradle a new Gradle project inside Eclipse must be selected as shown in *Figure 1*
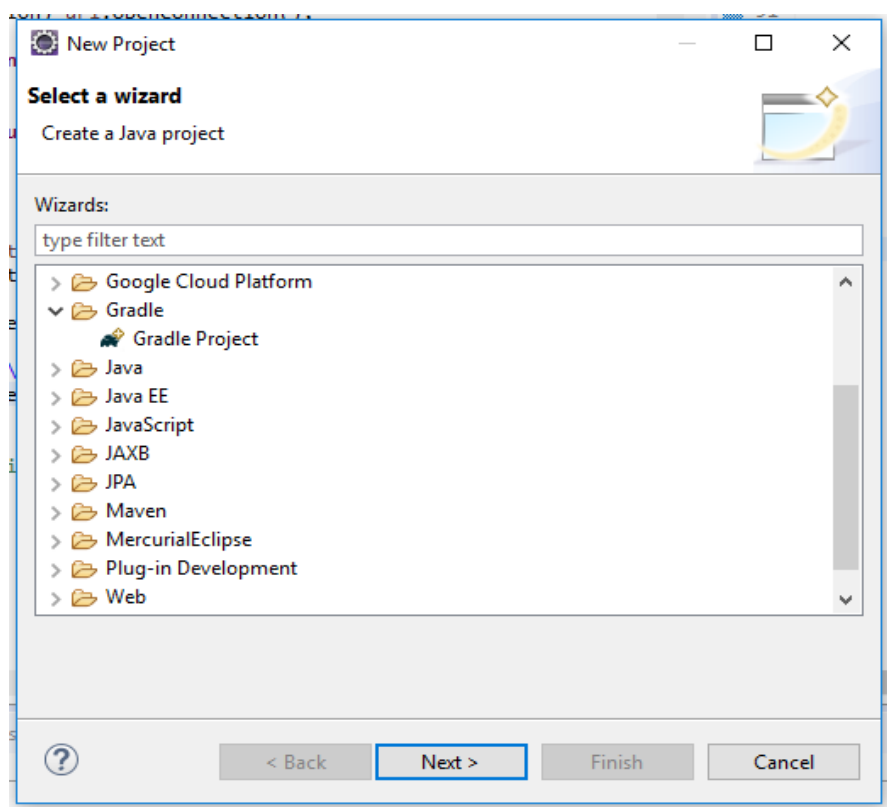

Figure 1 Creating Gradle project in Eclipse

Gradle is going to be the tool to download the Jars for the rest technologies by setting up the *build.gradle* file, the right dependencies for the project can be chosen including the versions, as shown in *Figure 2*.

```
48 dependencies {
49     // This dependency is exported to consumers, that is to say found on their compile classpath.
50     api 'org.apache.commons:commons-math3:3.6.1'
51
52     // This dependency is used internally, and not exposed to consumers on their own compile classpath.
53     implementation 'com.google.guava:guava:23.0'
54
55     // Use JUnit test framework
56     testImplementation 'junit:junit:4.12'
57     compile('org.projectlombok:lombok:1.16.20')
58     compile 'org.flywaydb:flyway-core:4.2.0'
59     compile('org.springframework.boot:spring-boot-starter-web',
60     'org.apache.tomcat.embed:tomcat-embed-jasper',
61     'javax.servlet:jstl')
62     compile('org.hibernate:hibernate-search-orm:5.6.1.Final',
63     'org.hibernate:hibernate-entitymanager:5.2.3.Final')
64     compile('org.hibernate:hibernate-core:5.2.10.Final',
65     'org.hibernate:hibernate-entitymanager')
66     compile('javax.xml.bind:jaxb-api:2.3.0')
67     compile('org.springframework.boot:spring-boot-starter-data-jpa')
68     compile('mysql:mysql-connector-java:8.0.12')
69     compile('org.springframework.boot:spring-boot-starter-thymeleaf')
70     testCompile('org.springframework.boot:spring-boot-starter-test')
71 }
```

Figure 2 Build.gradle File with the dependencies

Figure 2, illustrates how simple it is to include the Jars in the project, and as it can be noticed, adding Compile('dependency's name:Version') will fetch the jars related to the given decencies from the Maven repository, note that sometimes it is better not to mention the version and make Gradle choose the compatible version automatically.

The next table shows each line and the corresponding technology associated with:

Line number	Technology
57	Lombok project
58	Flyway
59,67	Spring boot
64	Hibernate
68	MySQL connector
69	Thymleaf

Table 1 Dependencies associated with the lines in Figure 2

To monitor the database for the project, MySQL Workbench is used, additionally, Workbench Server is needed, which runs the database locally. All these programs can be found here.

It worth noticing that spring is quite picky regarding the construction of the project, therefore the project should be ordered as illustrated in *Figure 3.*
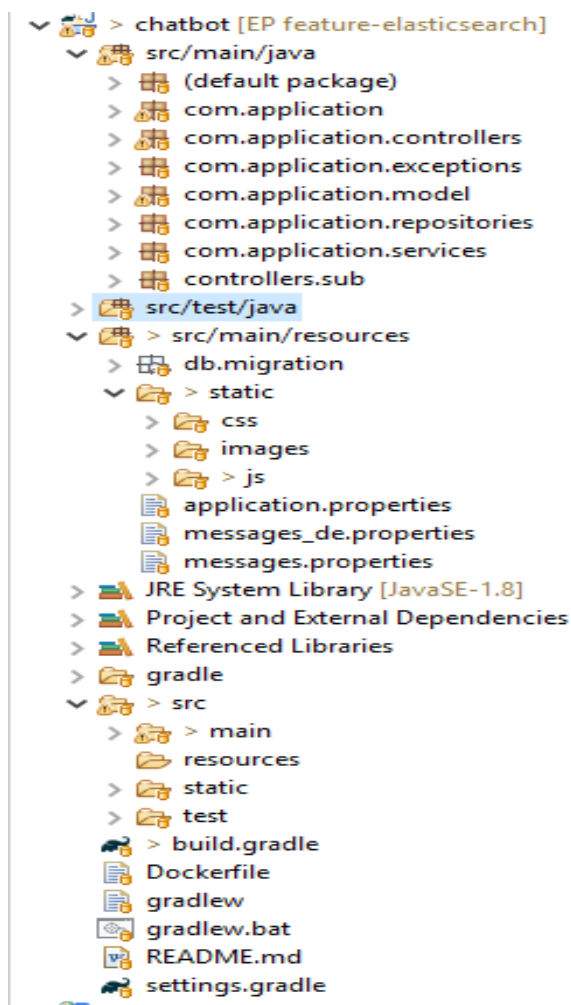
Figure 3 The hierarchy of spring boot application

Helpful tools

Some important tools can be used for the project are:

1. Postman:
   Useful tool to make HTTP requests and to test API's, it can be downloaded from here.
2. Notepad++.
   Great editor for a verity of programming languages, it can be downloaded from here.
3. Vue plugin in chrome.
   Makes it easier to debug Vue's objects in the browser's console, it can be downloaded from here.

3.2 Implementation

Frontend

To work in the Frontend HTML, Vue, Thymleaf, and JavaScript are used. The whole HTML code is written in the *widget.jsp* file.

The HTML code is straightforward but the Vue and Thymleaf markers shall be explained.

```
<div v-for="(email, index) in robotMessages">
    <div v-if="!email.robotResponse"
```

Figure 4 Vue markers inside widget.jsp file

Figure 4 Shows the usage of the Vue inside HTML or JSP file, this code renders the E-mails from the JavaScript file *fe-chatbot.js* and iterate them inside the JSP file, this makes JSP or HTML file more dynamic.
Moreover, the second line makes a decision based on a variable called *robotResponse* inside the *email* object, and if the condition is true the next div is included otherwise it is going to be hidden. In this case, it is decided whether the message is from the robot or from the user and based on this condition Vue decide which div to show.

To declare the Vue variables and use them, an instance must be created in the JavaScript file. *Figure 5*, gives a small glance at how to declare them. robotMessages, as it can be seen, is an object with variables such as robotResponse. On the other hand, the iconMode is a Boolean variable used to show and hide the chat window when the ✕ symbol is clicked.

```
1 var vue_det = new Vue({
2     el : '#chatbot-div',
3     data : {
4         robotMessages : [ {
5             message : 'Welcome to our website, how can I help?',
6             time : '',
7             robotResponse : true,
8             url : '',
9         } ],
10        iconMode : true
11    },
```

Figure 5 Variables of the Vue instance in fe-chatbot.js

Figure 6, explains the usage of the Thymleaf inside the *widget.jsp* file, where the frame is included in the website as a demo, if the *demo* variable is true, and excludes it from rendering otherwise.
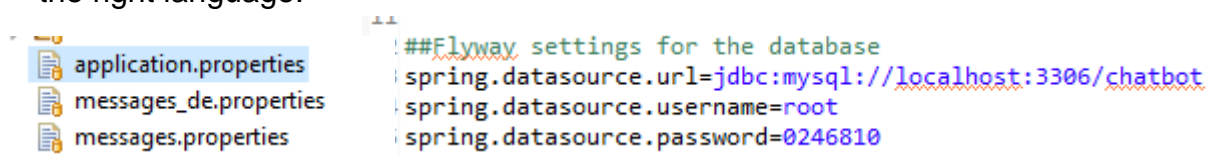
```
<iframe th:if="${demo == true}"
        src="https://www.haw-hamburg.de/startseite.html"
        style="width: 1800px; height: 1500px;"></iframe>
```

Figure 6 Thymleaf usage inside widget.jsp

To include the demo variable, it needs to be injected by using the properties file in the Backend.

Backend

Properties files are files used heavily in java to store some configurable variables that can be injected into the project. Furthermore, it is going to be used for internationalization, where keywords can be declared in the file and used in the frontend and based on the local language spring-Thymleaf is taking care of fetching the right language.

```
application.properties          ##Flyway settings for the database
messages_de.properties          spring.datasource.url=jdbc:mysql://localhost:3306/chatbot
messages.properties             spring.datasource.username=root
                                spring.datasource.password=0246810
```

Figure 7 Application.properties file with the global variables

For the database configuration, the *application.properties* file is used, where the domain, username, and password have been initialized.

The Java Persistence API (JPA) is the standard way of persisting Java objects into relational databases. In this project, it has been used hand by hand with Hibernate to establish the saving and loading from the database. To create a repository a spring framework annotation @Repository is needed as shown in the next figure.

```
@Repository
public interface ArticleRepository extends JpaRepository<Article, Long> {

    Article findById(Long id);

}
```

Figure 8 ArticleRepository to save and fetch the articles from the database.

By extending the JpaRepository<Article, Long>, automatic methods are already provided, such as: *save, findById*.

SearchController is a @RestController, which provides the results from the Elasticsearch engine, the usage of this is very important especially if the Elasticsearch is used for searching, due to the cross-region error.
If an HTTP request directly made by JavaScript a cross-region error my occur and that's why this method must be ignored and the JSON parsing for the Elasticsearch must be done in the backend.

Overview

For a better understanding of how the software is working, two diagrams are shown in *Figure 9* and *10.* The user opens the chat window from the browser and inserts the question, then the question is forwarded to the JavaScript by Vue, then sending a GET request to the backend of the project where the spring boot is in, afterward Java application sends a request to the Elasticsearch, a search algorithm is applied and a result with a percentage of accuracy is returned. Finally, Vue renders back the answer in the *widget.html* file and show as a message.
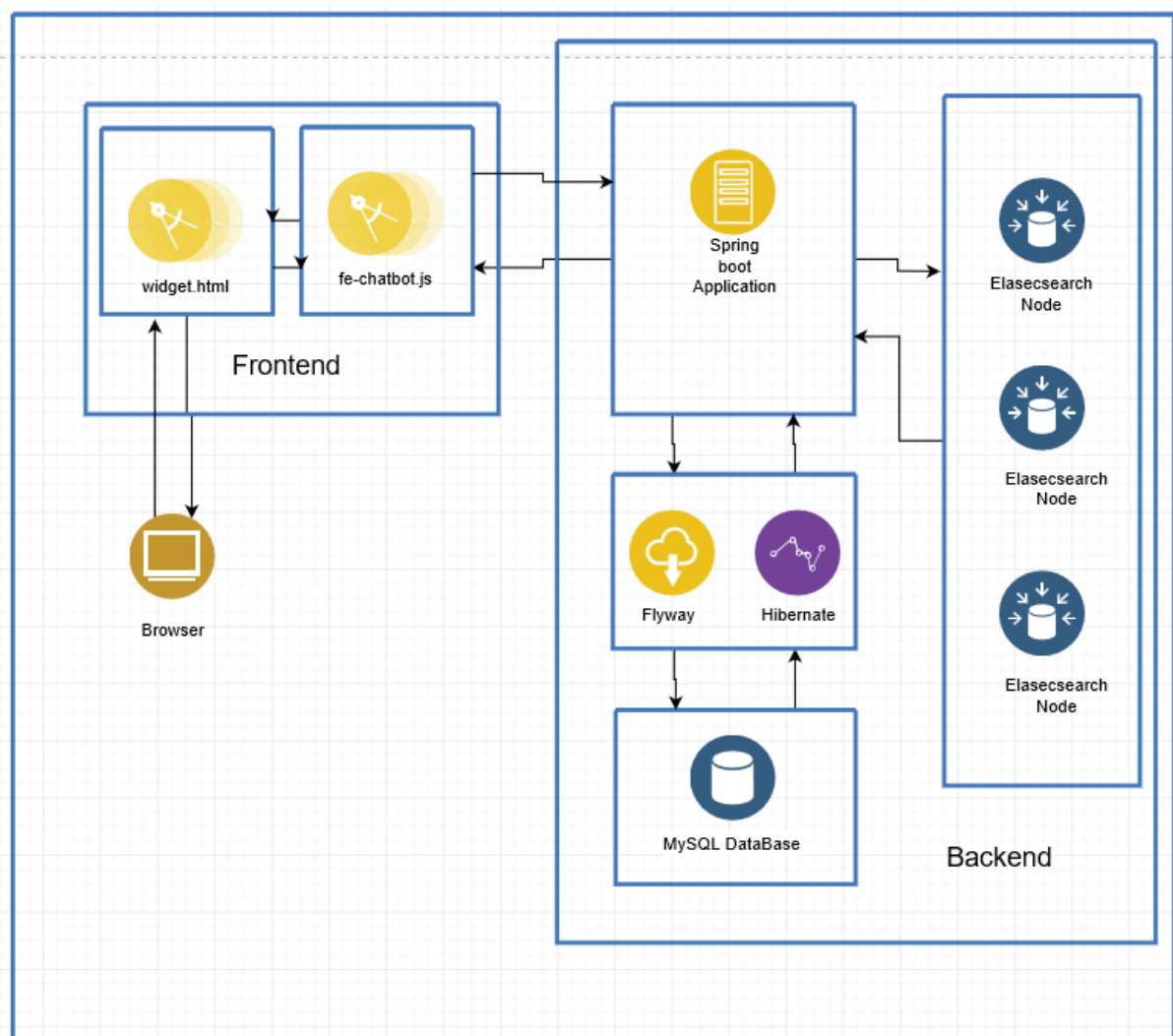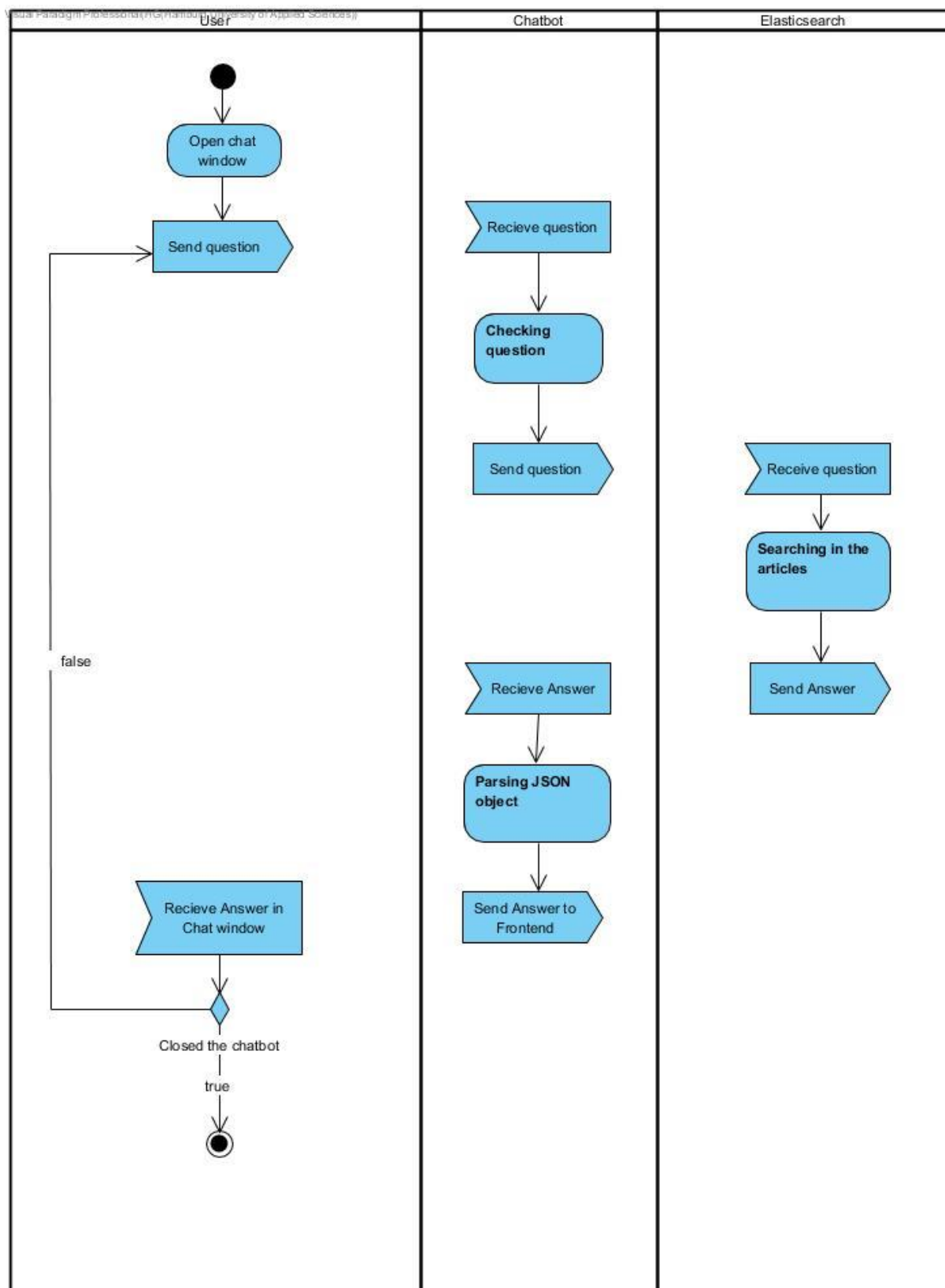

Figure 9 Overview of the chatbot system

Figure 10 Activity diagram of the project

The search subsystem

The search sub system is a system which will have to purposes, first it will save the data and second it will provide the search capabilities.
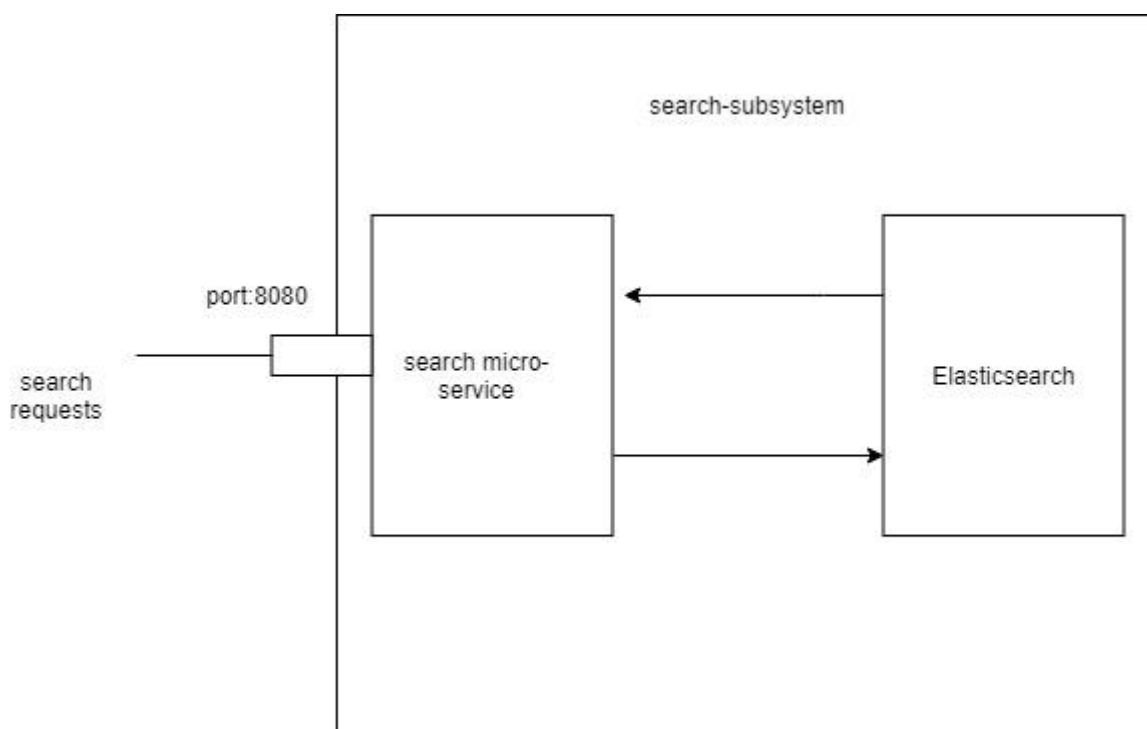

Figure 11: general view of the search-subsystem

The system consists of two main parts:

1- *Elasticsearch*, which is used to store and search the data. It provides a distributed, multitenant-capable full-text search engine with an HTTP web interface and schema-free JSON documents. Elasticsearch is developed in Java and is released as open source under the terms of the Apache License. [1]

The data that is to be saved and searched using Elasticsearch must be structured using indexes, an index is basically a collection of data which have something in common an entry inside the index is referred to as a document. In our project there is one index which is the article index and all the articles found on the HAW website should be saved under this index.

2- search microservice, search microservice is the application that is used to save the data to Elasticsearch, search for the data, and communicate with the chatbot.

The configurations of Elasticsearch and the data structure are all written in the search microservice.

If for example a new index is to be added or the sorting of the search result needs to be changed, this is to be done in the search microservice.

Technologies used in the search microservice:

- spring boot: which is a framework that provides inversion of control, dependency injection, and some other APIs.
- Gradle: build and dependency management tool

How the whole system works:

The search subsystem receives an Http request from the chatbot, it could be a search request, delete request or even insert request. The search subsystem then receives the request and process it. The processing of such requests require the search subsystem to find something in the data or maybe manipulate it. Querying the data is achieved through HTTP requests to Elasticsearch.

4 Testing and Results.

Since the whole Chatbot system is made of two main subsystems, Testing was done in two different steps:

Step one, unit testing, Each Unit was tested on its own to ensure the correct operation of each of the sub systems:

Search sub-system test:
To test the search sub-system postman -an Http client for testing web services- was used. An Example is shown in *Figure 12*.
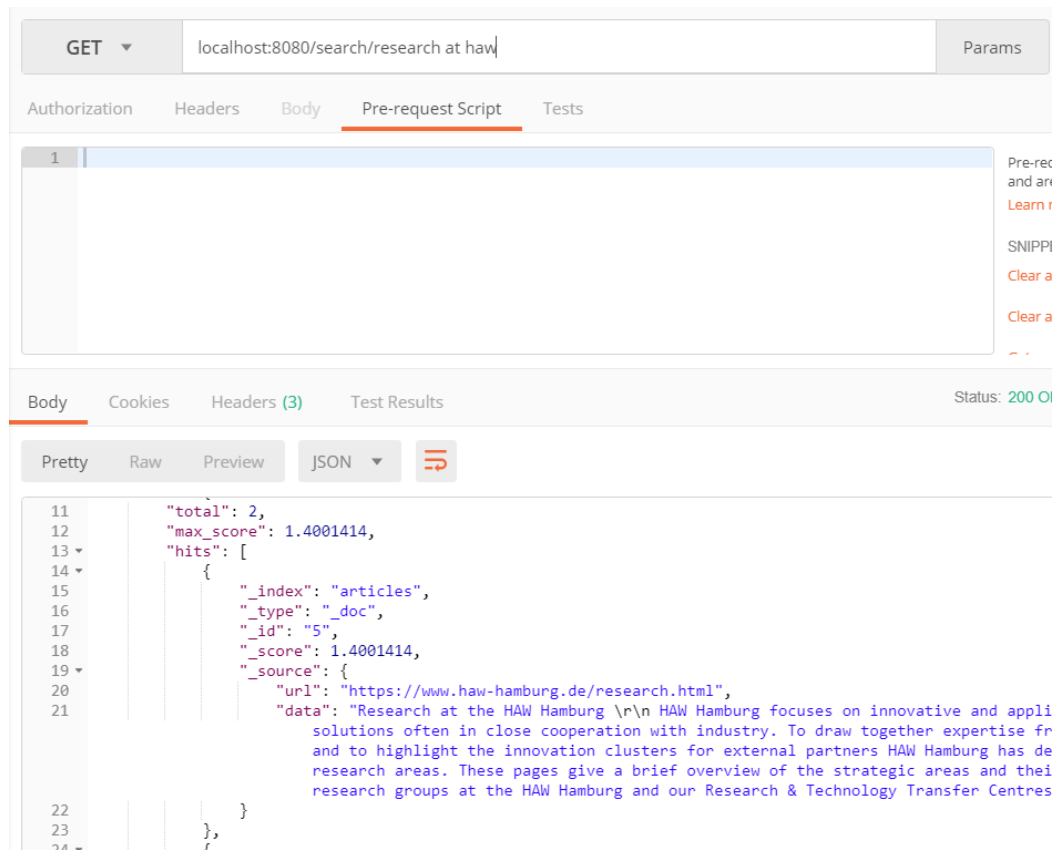
Figure 12: An Example of a search request to The search sub-system

Step two, integration testing, after making sure that every sub-system works properly on its own, it was necessary to ensure the system works properly as a whole too, because a lot of things can go wrong when connecting the different parts of the systems together. Therefore, the system was tested in its entirety.

This was done be asking a question in the chatbot widget at the frontend side, and then observing the correct operation of the whole system as the request gets sent to the chatbot subsystem and from there to the search subsystem and all the way back to the chatbot widget.

Frontend Test:

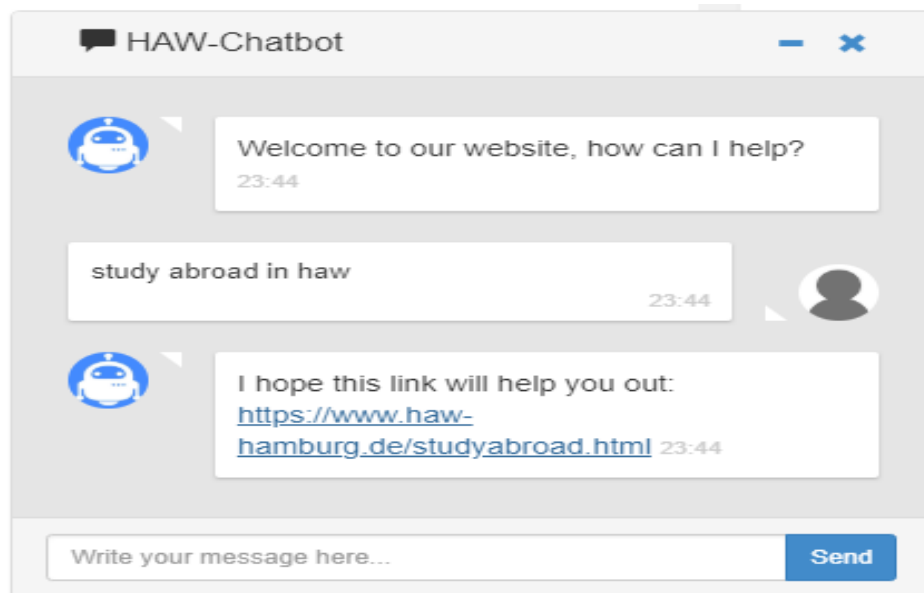An overall testing has been achieved and some cases are shown in the next figures:

Figure 13 Study abroad request

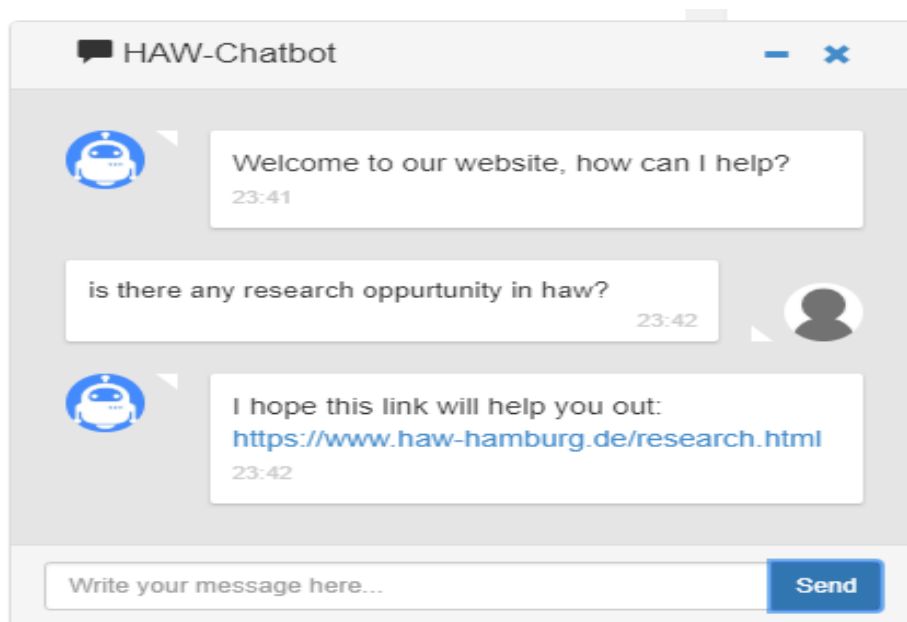*Figure 13,* shows a successful request for a user interested in studying abroad.

Figure 14 Research request

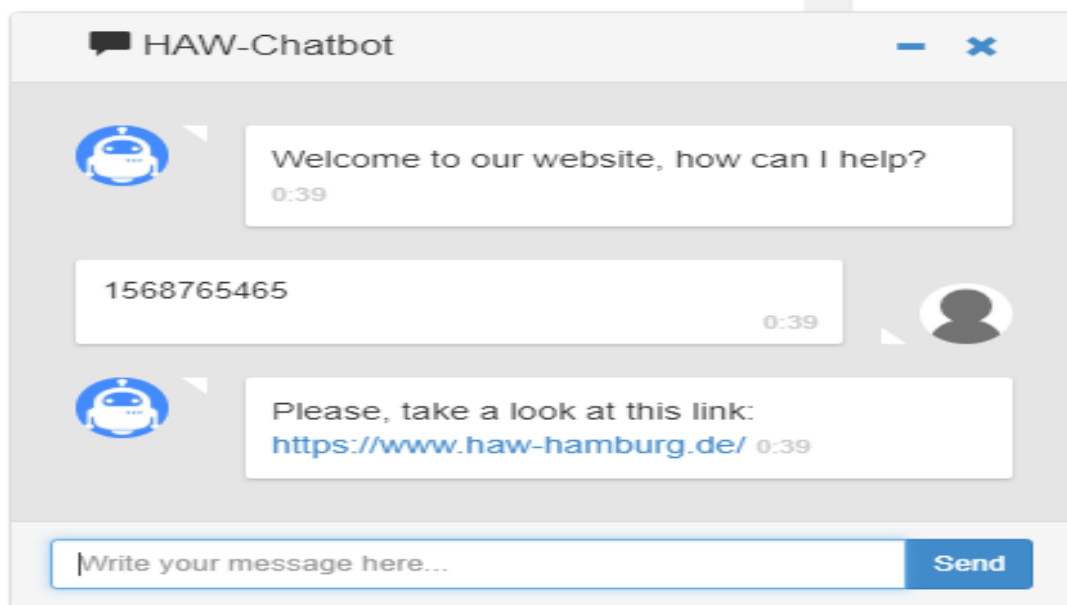*Figure 14,* shows a successful request for a user interested in researches.

Figure 15 Invalid input

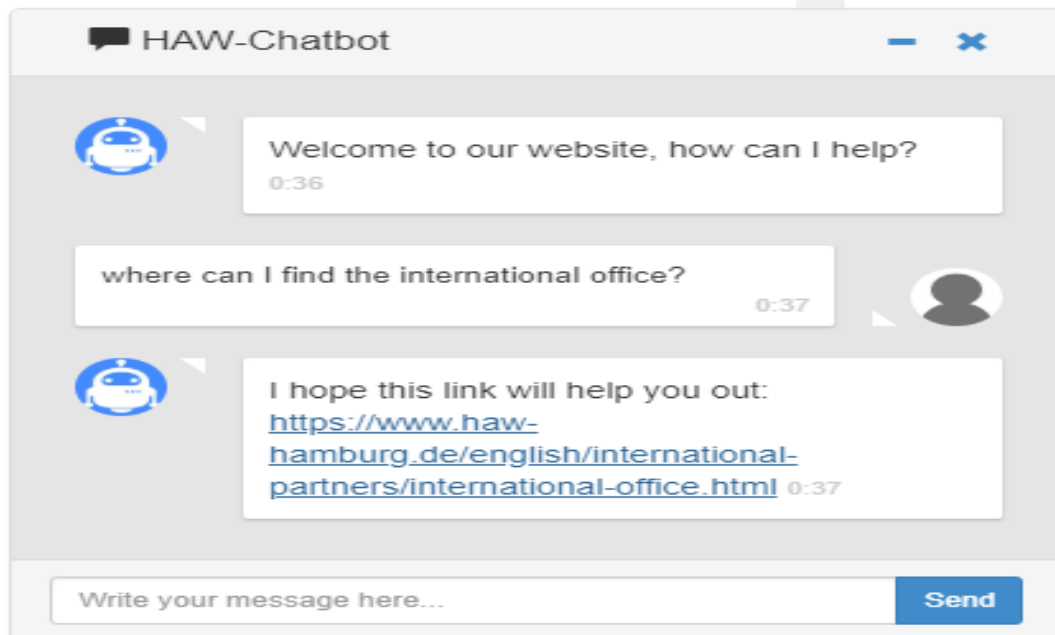*Figure 15,* shows an unsuccessful request for a user entered a random request.

Figure 16 International office request

Figure 16, shows a successful request for a student who is searching for the international office.

5 Conclusion and future improvements.

5.1 Future improvements

Some ideas and future improvements can be achieved by extending the current version of the Chatbot and adding more features to it.

1 – Adding an attachment icon to allow the user to upload some documents if it is necessary.
2 – Adding a voice recorder to send voices.
3 – Improving the AI of the Chatbot in the Backend, by adding a neural network to understand the people's questions.
4 – Adding more nodes in the Elasticsearch to optimize the searching.
5 – Adding an HTML page or XML file to customize the Chatbot.

5.2 Conclusion

Chatbot applications streamline interactions between people and services, enhancing customer experience. At the same time, they offer companies new opportunities to improve the customer's engagement process and operational efficiency by reducing the typical cost of customer service.

To be successful, a chatbot solution should be able to effectively perform both tasks. Human support plays a key role here: Regardless of the kind of approach and the platform, human intervention is crucial in configuring, training and optimizing the chatbot system.

Chatbot has been implemented in this project for the HAW website, and it has been proven that it can provide a reasonable result for the student search. Chatbot still has some long way to go in regard to the AI, however, the infrastructure has been set and a distributed system has been established to make it easier to develop the Chatbot and to make it more sophisticated.

6 References

Thymleaf:
https://www.thymeleaf.org/doc/tutorials/2.1/thymeleafspring.html

Vue Guide:
https://vuejs.org/v2/guide/

Spring boot Tutorial:
https://www.tutorialspoint.com/spring_boot/index.htm

Elasticsearch Tutorial:
https://www.elastic.co/guide/en/elasticsearch/reference/current/getting-started.html

Documentation:
https://github.com/Alhajras/chatbot-/blob/feature-elasticsearch/Chatbot%20Documentation.pdf

GitHub Repository:
https://github.com/Alhajras/chatbot-/tree/feature-elasticsearch