MINISTRY OF EDUCATION AND SCIENCE OF UKRAINE
SUMY STATE UNIVERSITY
FACULTY OF ELECTRONICS AND INFORMATION TECHNOLOGY
COMPUTER SCIENCE DEPARTMENT

"Approved for defence."
Acting Head of the Department
____________ Igor SHELEHOV
(signature)
09.06.2023

GRADUATION THESIS
for obtaining the educational degree of Bachelor

in the specialty 122 - Computer Science,
educational-professional program "Informatics"
on the topic: " INFOWARE AND SOFTWARE OF INTELLIGENT CHAT-BOT "
by the student of group IN-95AH, BASIT ABIODUN ADEOSUN.

The Bachelor Graduation Thesis contains the results of original research. The use of ideas, results, and texts of other authors is properly referenced to the respective sources.

[illegible] BASIT ABIODUN ADEOSUN
(signature)

Supervisor
Head of Department doctor of technical sciences, professor
Artem KOROBOV ____________
(signature)

Sumy 2023

SUMY STATE UNIVERSITY
FACULTY OF ELECTRONICS AND INFORMATION TECHNOLOGY
COMPUTER SCIENCE DEPARTMENT

«Approved».
Acting Head of the Department

_______________ Igor SHELEHOV
(signature)

TASK FOR THE GRADUATION THESIS
to obtain the educational degree of Bachelor
in the specialty 122 - Computer Science,
educational-professional program "Informatics"
by the student of group IN-95AH, Basit Adeosun Abiodun.

1. Topic of the Bachelor Graduation Thesis: "INFOWARE AND SOFTWARE OF INTELLIGENT CHAT-BOT" approved by the order of SumDU on *June 1, 2023.№ 0475-VI*
2. The deadline for the submission of the Bachelor Graduation Thesis *until June 9, 2023*
3. Input data for the qualification work_______________________________________.
4. Table of Contents for the Explanatory Memorandum (List of questions to be addressed)
1) Analysis of the subject area, defining the purpose, and forming the tasks of the Bachelor Graduation Thesis. 2) Review of the theoretical material. 3) design and implement an intelligent chatbot.4) Analysis of the obtained results.
5. List of graphic materials (with specific mention of mandatory drawings)_________________________.
6. Project consultants (with the corresponding sections of the project they are associated with).

Section	Consultant	Signature, date	
		The assignment has been issued	The assignment has been accepted

7. Date of assignment issuance «____» _______________ 20 ___

The assignment has been accepted for execution _________________ Supervisor _________________
(signature) (signature)

КАЛЕНДАРНИЙ ПЛАН

№	Titles of the stages of the Bachelor Graduation Thesis	Deadline	Note
1	*Analysis of the subject area, defining the purpose, and forming the tasks of the Bachelor Graduation Thesis*		
2	*Review of the theoretical material*		
3	*Development of the automation of an inventory system*		
4	*Analysis of the obtained results*		
5	*Bachelor Graduation Thesis Formatting*		

Higher education student [illegible] Supervisor _________________
(signature) (signature)

ABSTRACT

Note: 42 pages, 14 figures, 1 appendix, 10 reference sources.

Object of study – Infoware and software of intelligent chat-bot

Purpose - To design and implement an intelligent chatbot with natural language processing (NLP) capabilities using Java and Apache OpenNLP in the Spring Framework.

Results - an intelligent chatbot has been designed and implemented with natural language processing (NLP) capabilities using Java and Apache OpenNLP in the Spring Framework.

Keywords: Intelligent chatbot, Natural Language Processing, Java, Apache OpenNLP, Spring Framework, Sentence detection, Named entity recognition, Weather API, News API

Contents

Introduction

1.1 **Background**

In the era of digital transformation, one of the most groundbreaking developments is the advent of chatbot systems. These automated entities engage with users through text, offering a simulated conversation like human interaction. By processing and responding to user inputs, chatbots have revolutionized customer service, data collection, and more, in a variety of industries. They can offer 24/7 support, manage multiple inquiries simultaneously, and provide immediate responses. However, creating an intelligent chatbot, one that can comprehend the intricacies of human language, interpret the semantic and contextual meaning of sentences, and offer rich, personalized responses, presents a significant challenge. This project focuses on building such a chatbot, harnessing the power of Natural Language Processing (NLP) with Java and Apache OpenNLP within the Spring Framework. Additionally, the chatbot aims to provide enriched user experiences by integrating with external Weather and News APIs.

1.2 Problem Statement

Despite the widespread implementation of chatbots, a common drawback is their inability to accurately grasp and interpret human language, often resulting in generic or incorrect responses. To address this challenge, various existing solutions leverage advanced Natural Language Processing (NLP) techniques. These solutions utilize deep learning models, such as transformer-based architectures like BERT or GPT-3, which significantly improve the chatbot's ability to understand and respond to user inputs with higher accuracy and contextual understanding [9][10].

Furthermore, another existing solution focuses on incorporating real-time, context-aware information into chatbot interactions. By integrating with relevant APIs, such as weather and news services, these chatbots can provide users with up-to-date and personalized information. For example, they can retrieve current weather conditions or deliver the latest news updates in real-time [4][3]. This project builds upon these existing solutions by combining advanced NLP algorithms and API integration to construct a chatbot that comprehends user inputs more accurately and offers real-time, relevant information.

1.3 Objectives

The core objectives of this project include:

1. Design: Lay out a systematic blueprint of an intelligent chatbot, focusing on user interaction and data flow.
2. Implementation: Apply Java and Apache OpenNLP within the Spring Framework to realize the design, incorporating sentence detection and named entity recognition features for enhanced language understanding.
3. Integration: Seamlessly combine external APIs, particularly weather and news APIs, to enrich the chatbot's knowledge base and its response scope.
4. Evaluation: Rigorously evaluate the implemented system, validating its functionalities, performance, and user experience, ensuring it aligns with the predefined requirements.

1.4 **Methodology**

The project follows the following methodology:

1. Conduct a literature review on chatbot systems, NLP techniques, Java, Apache OpenNLP, and the Spring Framework.
2. Design the architecture of the intelligent chatbot, including the user interface and integration with external APIs.
3. Implement the chatbot using the provided code as a base, focusing on the NLP models, API integration, and user interface implementation.
4. Perform testing and validation of the chatbot's functionality and evaluate its performance and user experience.
5. Analyze the results and discuss the achievements, contributions, and potential future work.

1.5 Scope and Limitations

While the project sets out to build a comprehensive chatbot system, some limitations exist. The chatbot's understanding of human language is constrained by the quality and quantity of its training data. Thus, it may struggle with complex or ambiguous queries. Also, its responses are bound by the information accessible via the integrated APIs, limiting its knowledge to predefined areas like weather and news. Any inquiries beyond this scope may not receive adequate responses.

Literature Review

2.1 Chatbot Systems: Chatbot systems are AI software designed to interact with humans in their natural languages. These systems utilize technologies such as Machine Learning and Natural Language Processing (NLP) to understand, interpret, and respond to user queries in a manner that resembles human conversation. Chatbots find applications in various domains, including customer service, information retrieval, and user experience enhancement.

Advantages:

- Instantaneous responses and support, enhancing customer service.
- Scalability, allowing for handling a large volume of user interactions without the need for additional human resources.
- 24/7 availability, providing access to information or services at any time.
- Personalized interactions, improving user engagement and satisfaction.

Disadvantages:

- Limited ability to handle complex or ambiguous queries.
- Dependence on accurate training data and continuous model updates.
- Potential privacy and security concerns when handling sensitive user information.

2.2 Natural Language Processing: Natural Language Processing (NLP) is a specialized field of AI that concentrates on the interaction between humans and computers language. NLP enables computers to understand, interpret, and generate human language through techniques such as named entity recognition, sentiment analysis, language translation, and sentence detection.[5].

Advantages:

- Automated understanding and analysis of large volumes of text data.
- Language translation and localization for global applications.
- Sentiment analysis and opinion mining for market research and social media analysis.
- Powering chatbot systems and voice assistants for natural language interactions.

Disadvantages:

- Challenges in handling ambiguity and context-dependent interpretation.
- Difficulty in processing slang, colloquialisms, and non-standard language.
- Computationally and resource-intensive processing for large-scale applications.
- Ethical considerations in ensuring unbiased and fair language processing.

2.3 Java: Java is a high-level, object-oriented programming language known for its platform independence and "write once, run anywhere" capability. It is widely used for developing web applications, mobile applications, and enterprise-level software. Java provides a rich ecosystem of libraries and frameworks, making it a popular choice for building robust and scalable applications.[8].

Advantages:

- Platform independence, allowing Java code to run on different operating systems without recompilation.
- Strong community support and a vast collection of libraries and frameworks.
- Robust memory management and garbage collection.

- Extensive tooling for debugging, profiling, and performance optimization.

Disadvantages:

- Verbose syntax and boilerplate code compared to some other programming languages.
- Slower execution speed compared to lower-level languages like C++.
- Limited support for functional programming paradigms.

2.4 **Apache OpenNLP:** Apache OpenNLP is a Java-based open-source library for natural language processing. It offers various NLP functionalities, including tokenization, sentence segmentation, part-of-speech tagging, named entity extraction, chunking, and parsing. OpenNLP also provides machine learning algorithms based on maximum entropy and perceptron models.[1][6].

Advantages:

- Comprehensive suite of NLP functionalities for various text processing tasks.
- Easy integration with Java-based applications and frameworks.
- Actively maintained and supported by the Apache Software Foundation.
- Extensibility and customizability to meet specific NLP requirements.

Disadvantages:

- Steep learning curve for users new to NLP and Java programming.
- Limited support for languages other than English.
- Performance limitations for large-scale processing or real-time applications.

2.5 **Spring Framework:** The Spring Framework is a popular Java-based framework for building enterprise applications. It provides a comprehensive programming and configuration model, supporting different application architectures such as messaging, transactional data, persistence, and web-based applications. Spring Boot, a project built on top of the Spring Framework, simplifies the development of stand-alone, production-grade Spring applications.[2][7].

Advantages:

- Simplified development through features like dependency injection and inversion of control.
- Modular and scalable architecture for building enterprise-level applications.
- Seamless integration with other Java libraries and frameworks.
- Robust support for testing, monitoring, and management of applications.

Disadvantages:

- Initial learning curve to understand the Spring ecosystem and concepts.
- Increased complexity as the number of dependencies and configurations grows.
- Potential performance overhead compared to lightweight frameworks for simpler applications.

System Design

3.1 Architecture Overview

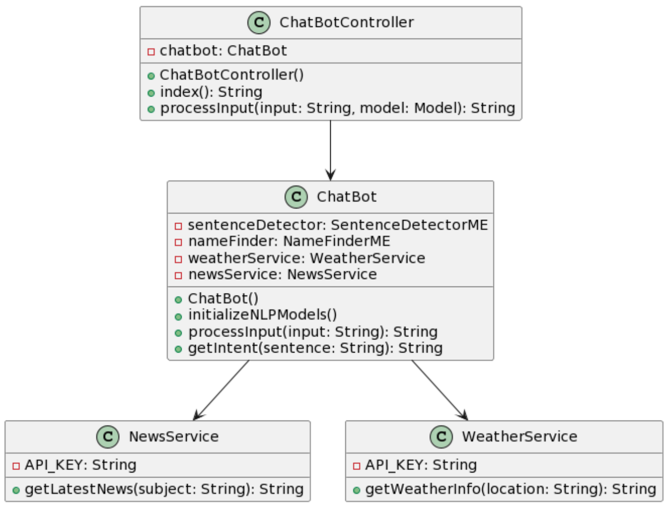

C ChatBotController
chatbot: ChatBot
ChatBotController()
index(): String
processInput(input: String, model: Model): String
C ChatBot
sentenceDetector: SentenceDetectorME
nameFinder: NameFinderME
weatherService: WeatherService
newsService: NewsService
ChatBot()
initializeNLPModels()
processInput(input: String): String
getIntent(sentence: String): String
C NewsService
API_KEY: String
getLatestNews(subject: String): String
C WeatherService
API_KEY: String
getWeatherInfo(location: String): String

Figure 1 diagram showing the classes.

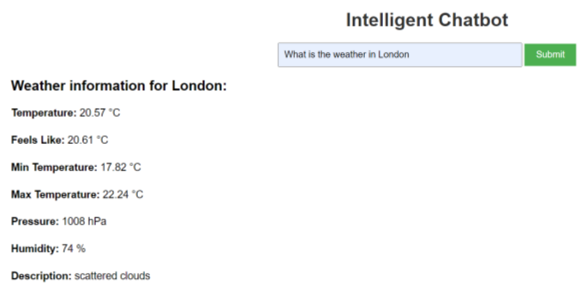
The architecture of the intelligent chatbot system consists of several components that work together to process user input and provide appropriate responses. The key components of the system architecture are as follows:

- User Interface: The user interacts with the chatbot through a web-based interface. The interface allows the user to enter queries and receive responses from the chatbot.
- ChatBot Controller: This component handles the HTTP requests from the user interface. It communicates with the ChatBot service to process user inputs and generate appropriate responses.
- ChatBot Service: The ChatBot service contains the core logic of the chatbot. It uses natural language processing (NLP) techniques to analyze and interpret user inputs and generate relevant responses. It interacts with other services, such as the WeatherService and NewsService, to fetch additional information.
- NLP Models: The chatbot utilizes NLP models provided by the Apache OpenNLP library. These models include a sentence detector model and a named entity recognition (NER) model. The sentence detector model is responsible for identifying sentences in user inputs, while the NER model identifies named entities like person names.
- WeatherService: This service integrates with an external weather API to fetch weather information based on user queries related to weather[4]. It

uses the OpenWeatherMap API to retrieve current weather data for a specific location.

- NewsService: The NewsService integrates with a news API to retrieve the latest news updates based on user queries related to news topics[3]. It uses the NewsAPI to fetch top headlines and news articles matching the user's query.

3.2 User Interface Design



Weather information for London:

Temperature: 20.57 °C

Feels Like: 20.61 °C

Min Temperature: 17.82 °C

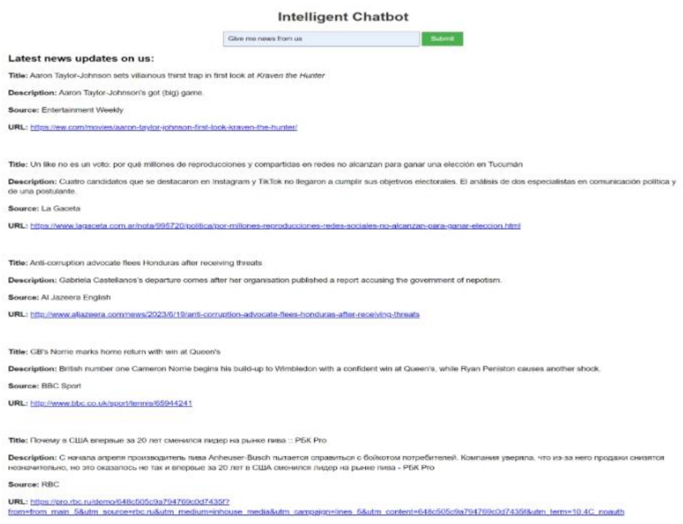
Max Temperature: 22.24 °C

Pressure: 1008 hPa

Humidity: 74 %

Description: scattered clouds

Figure 2 shows the interface design based on the user query on weather in London.



Latest news updates on us:

Title: Aaron Taylor-Johnson sets villainous thirst trap in first look at *Kraven the Hunter*

Description: Aaron Taylor-Johnson's got (big) game.

Source: Entertainment Weekly

URL: https://ew.com/movies/aaron-taylor-johnson-first-look-kraven-the-hunter/

Title: Un like no es un voto: por qué millones de reproducciones y compartidas en redes no alcanzan para ganar una elección en Tucumán

Description: Cuatro candidatos que se destacaron en Instagram y TikTok no llegaron a cumplir sus objetivos electorales. El análisis de dos especialistas en comunicación política y de una postulante.

Source: La Gaceta

URL: https://www.lagaceta.com.ar/nota/995720/politica/por-millones-reproducciones-redes-sociales-no-alcanzan-para-ganar-eleccion.html

Title: Anti-corruption advocate flees Honduras after receiving threats

Description: Gabriela Castellanos's departure comes after her organisation published a report accusing the government of nepotism.

Source: Al Jazeera English

URL: http://www.aljazeera.com/news/2023/6/19/anti-corruption-advocate-flees-honduras-after-receiving-threats

Title: GB's Norrie marks home return with win at Queen's

Description: British number one Cameron Norrie begins his build-up to Wimbledon with a confident win at Queen's, while Ryan Peniston causes another shock.

Source: BBC Sport

URL: http://www.bbc.co.uk/sport/tennis/65944241

Title: Почему в США впервые за 20 лет сменился лидер на рынке пива :: РБК Pro

Description: С начала апреля производитель пива Anheuser-Busch пытается справиться с бойкотом потребителей. Компания уверяла, что из-за него продажи снизятся незначительно, но это оказалось не так и впервые за 20 лет в США сменился лидер на рынке пива - РБК Pro

Source: RBC

URL: https://pro.rbc.ru/demo/648c505c9a794769c0d7435f?from=from_main_5&utm_source=rbc.ru&utm_medium=inhouse_media&utm_campaign=lines_5&utm_content=648c505c9a794769c0d7435f&utm_term=10.4C_noauth

Figure 3 user interface showing user query on news in US.

The user interface is implemented as a web page using HTML, CSS, and JavaScript. It provides an input field for the user to enter queries and a submit button to send the query to the server for processing. The response from the chatbot is displayed on the same page.

3.3 NLP Models and Integration

The ChatBot service incorporates NLP models provided by Apache OpenNLP to enhance its natural language processing capabilities. Specifically, it utilizes the SentenceDetectorME class for sentence detection and the NameFinderME class for named entity recognition.

During the initialization of the ChatBot service, the NLP models are loaded. The sentence detection model and the named entity recognition model are loaded from their respective binary files using InputStreams. The ChatBot class contains a method called **initializeNLPModels()** that handles this loading process.

1. **Sentence Detection Model Integration**:
   - The sentence detection model is loaded from the "sentence-detection.bin" binary file. This model is responsible for identifying sentence boundaries within user inputs.
   - The SentenceDetectorME class is instantiated with the loaded SentenceModel, which encapsulates the sentence detection model.
   - The SentenceDetectorME instance, referred to as **sentenceDetector**, is utilized within the ChatBot service to detect sentences in user inputs. It provides the **sentDetect(text: String)** method to perform sentence detection and returns an array of detected sentences.
2. **Named Entity Recognition Model Integration**:
   - The named entity recognition model is loaded from the "person-detection.bin" binary file. This model is designed to recognize named entities, with a focus on identifying person names within user inputs.
   - The NameFinderME class is instantiated with the loaded TokenNameFinderModel, which encapsulates the named entity recognition model.
   - The NameFinderME instance, referred to as **nameFinder**, is used within the ChatBot service to perform named entity recognition. It provides the **find(tokens: String[])** method to recognize named entities within an array of tokens and returns an array of Span objects representing the positions of named entities.

By integrating these NLP models into the ChatBot service, the chatbot gains the ability to detect sentences and recognize person names within user inputs. This enhances the chatbot's understanding of user queries and enables it to provide more accurate and contextually relevant responses.

3.4 Integration with Weather API

The WeatherService component integrates with an external weather API to retrieve weather information. It utilizes the OkHttpClient library to make HTTP requests to the OpenWeatherMap API. The WeatherService constructs the appropriate URL based on the user's location query and sends a request to the API.

The response from the weather API is received as JSON data, which is then parsed to extract the relevant weather information. The WeatherService formats the weather information into a readable format and returns it to the ChatBot service for inclusion in the chatbot's response.

3.5 Integration with News API

The NewsService component integrates with a news API to retrieve the latest news updates based on user queries. It utilizes the OkHttpClient library to make HTTP requests to the NewsAPI. The NewsService constructs the URL based on the user's query and sends a request to the API.

The response from the news API is received as JSON data, which is then parsed to extract the necessary information such as news titles, descriptions, sources, and URLs. The NewsService formats the news updates into a readable format and returns them to the ChatBot service for inclusion in the chatbot's response.

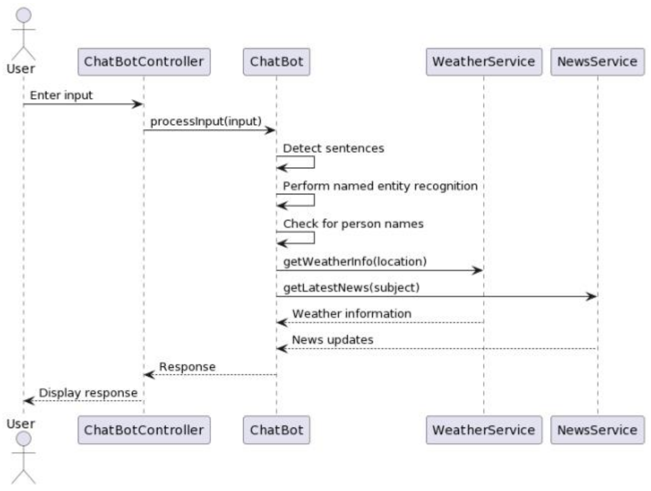


Figure 4 Sequence of the architecture

By visualizing these interactions in a sequence diagram, we provide an intuitive overview of the system's design and workflow. This concludes system design, which covers the design of the chatbot system, the details of its user interface, the integration of NLP models, and the incorporation of external APIs for weather and news services.

Implementation

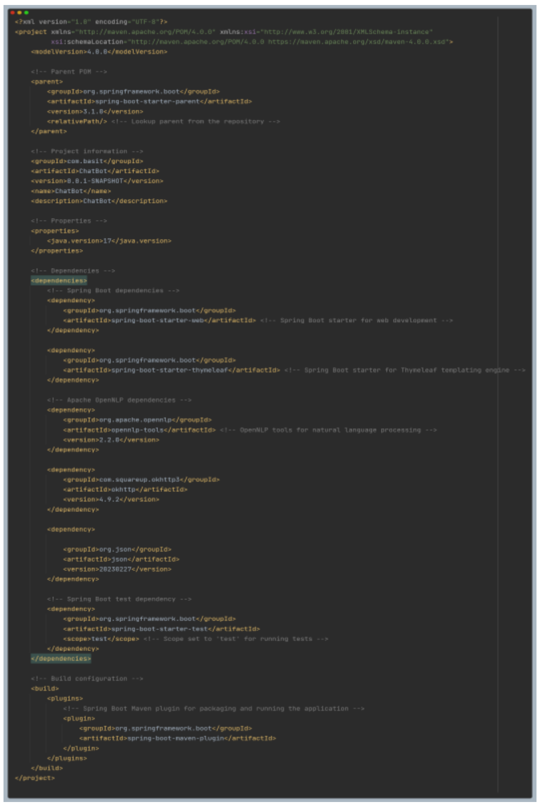
4.1 Setup and Configuration

Before starting the implementation, it is essential to set up the development environment correctly. This project requires Java Development Kit (JDK), an Integrated Development Environment (IDE) that supports Java, such as IntelliJ IDEA or Eclipse, and the Spring Framework.

The Apache OpenNLP libraries are also needed for the natural language processing (NLP) functionalities. These libraries can be added as dependencies in the project's build configuration file.

We also used Apache Maven, which is a software project management and comprehension tool. Maven allowed us to easily manage our project's build, report, and documentation from a central piece of information. The configuration for Maven was done using the pom.xml file.

Pom.xml



```xml
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 https://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>

    <!-- Parent POM -->
    <parent>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-parent</artifactId>
        <version>3.1.0</version>
        <relativePath/> <!-- Lookup parent from the repository -->
    </parent>

    <!-- Project information -->
    <groupId>com.basit</groupId>
    <artifactId>ChatBot</artifactId>
    <version>0.0.1-SNAPSHOT</version>
    <name>ChatBot</name>
    <description>ChatBot</description>

    <!-- Properties -->
    <properties>
        <java.version>17</java.version>
    </properties>

    <!-- Dependencies -->
    <dependencies>
        <!-- Spring Boot dependencies -->
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-web</artifactId> <!-- Spring Boot starter for web development -->
        </dependency>

        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-thymeleaf</artifactId> <!-- Spring Boot starter for Thymeleaf templating engine -->
        </dependency>

        <!-- Apache OpenNLP dependencies -->
        <dependency>
            <groupId>org.apache.opennlp</groupId>
            <artifactId>opennlp-tools</artifactId> <!-- OpenNLP tools for natural language processing -->
            <version>2.2.0</version>
        </dependency>

        <dependency>
            <groupId>com.squareup.okhttp3</groupId>
            <artifactId>okhttp</artifactId>
            <version>4.9.2</version>
        </dependency>

        <dependency>

            <groupId>org.json</groupId>
            <artifactId>json</artifactId>
            <version>20230227</version>
        </dependency>

        <!-- Spring Boot test dependency -->
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-test</artifactId>
            <scope>test</scope> <!-- Scope set to 'test' for running tests -->
        </dependency>
    </dependencies>

    <!-- Build configuration -->
    <build>
        <plugins>
            <!-- Spring Boot Maven plugin for packaging and running the application -->
            <plugin>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-maven-plugin</artifactId>
            </plugin>
        </plugins>
    </build>
</project>
```

Figure 5. Pom.xml

The project was set up using the Spring framework, specifically Spring Boot which simplifies the initial creation and configuration of a new Spring application. The Spring framework was chosen because it allows for the creation of enterprise applications that are easy to test and maintain.

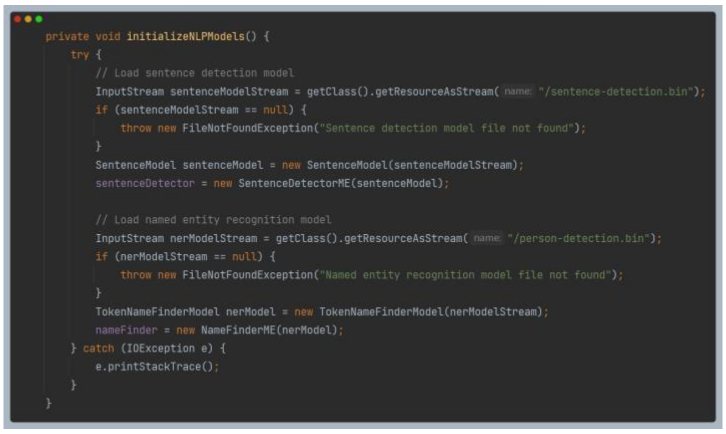
OpenNLP was used, a machine learning based toolkit for processing natural language text. It supports the most common NLP tasks, such as tokenization, sentence segmentation, and named entity recognition.

4.2 Development of NLP Models

The core of this chatbot system lies in its ability to understand natural language. This understanding is achieved by using Apache OpenNLP's models for sentence detection and named entity recognition.

Upon initialization, the chatbot loads the sentence detection model and the named entity recognition model. These models are contained in binary files, which are located in the application's resource directory.

Here is the implementation of the model loading in the ChatBot class:



```java
private void initializeNLPModels() {
    try {
        // Load sentence detection model
        InputStream sentenceModelStream = getClass().getResourceAsStream( name: "/sentence-detection.bin");
        if (sentenceModelStream == null) {
            throw new FileNotFoundException("Sentence detection model file not found");
        }
        SentenceModel sentenceModel = new SentenceModel(sentenceModelStream);
        sentenceDetector = new SentenceDetectorME(sentenceModel);

        // Load named entity recognition model
        InputStream nerModelStream = getClass().getResourceAsStream( name: "/person-detection.bin");
        if (nerModelStream == null) {
            throw new FileNotFoundException("Named entity recognition model file not found");
        }
        TokenNameFinderModel nerModel = new TokenNameFinderModel(nerModelStream);
        nameFinder = new NameFinderME(nerModel);
    } catch (IOException e) {
        e.printStackTrace();
    }
}
```

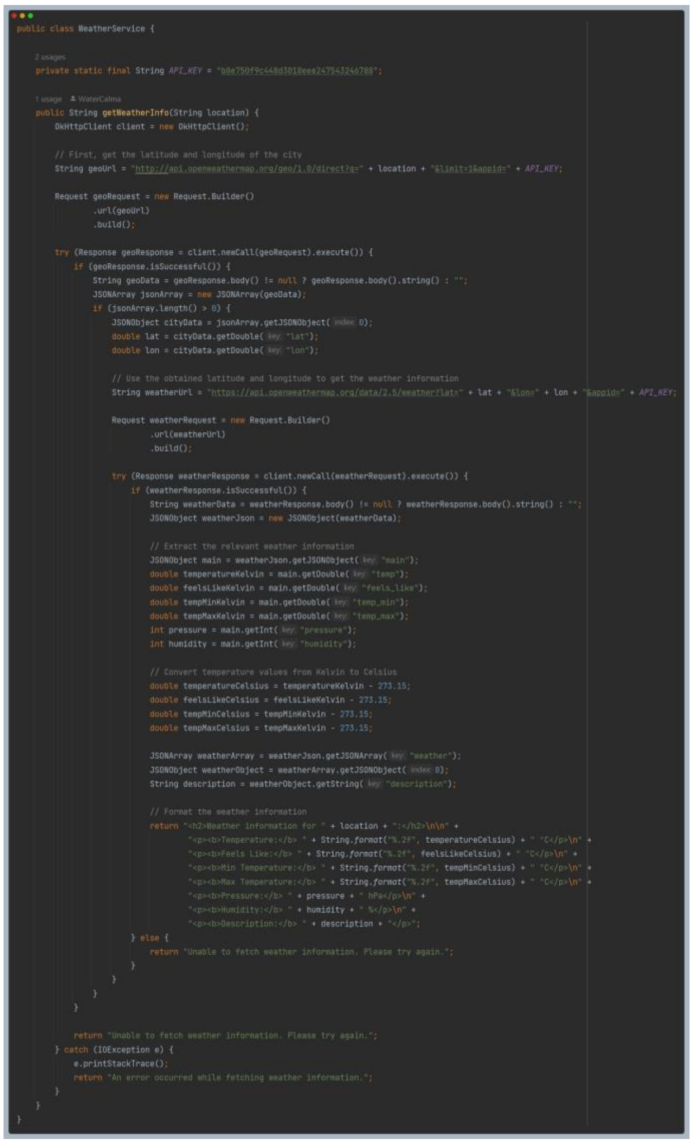
Figure 6 code showing how models are loaded

4.3 Integration with External APIs

The chatbot integrates two external APIs: the weather API and the news API. Two service classes, WeatherService and NewsService, are responsible for handling API requests and processing the responses.

Weather API Integration

The WeatherService class is responsible for retrieving weather information using an external weather API. It utilizes the OkHttpClient library to send HTTP requests and receive responses. The getWeatherInfo() method takes a location as a parameter, sends a request to the weather API, and parses the response to extract the relevant weather information. The extracted information is formatted and returned as a string.

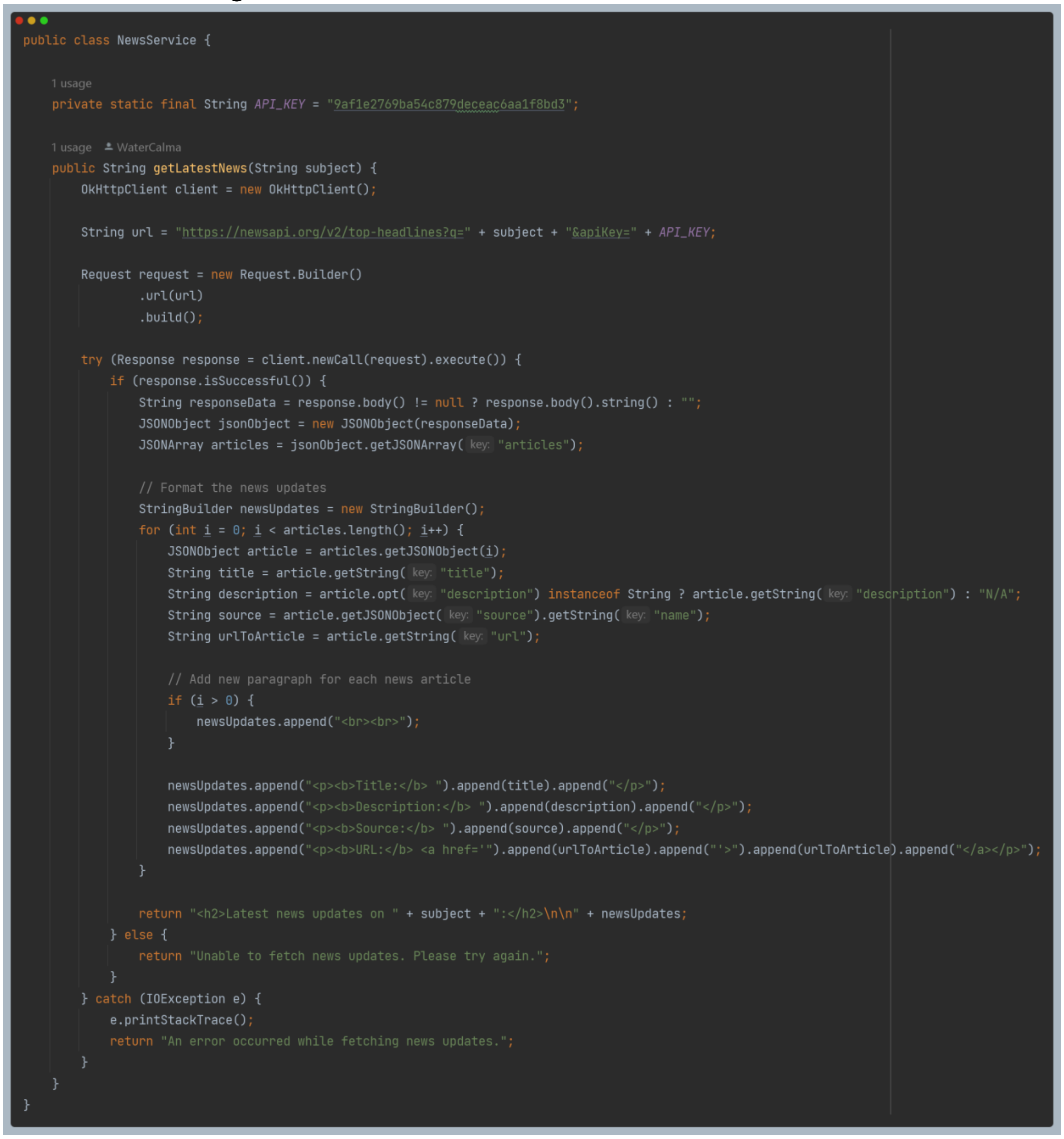


```java
public class WeatherService {

    2 usages
    private static final String API_KEY = "b8e750f9c448d3018eee247543246788";

    1 usage  WaterCalma
    public String getWeatherInfo(String location) {
        OkHttpClient client = new OkHttpClient();

        // First, get the latitude and longitude of the city
        String geoUrl = "http://api.openweathermap.org/geo/1.0/direct?q=" + location + "&limit=1&appid=" + API_KEY;

        Request geoRequest = new Request.Builder()
                .url(geoUrl)
                .build();

        try (Response geoResponse = client.newCall(geoRequest).execute()) {
            if (geoResponse.isSuccessful()) {
                String geoData = geoResponse.body() != null ? geoResponse.body().string() : "";
                JSONArray jsonArray = new JSONArray(geoData);
                if (jsonArray.length() > 0) {
                    JSONObject cityData = jsonArray.getJSONObject( index: 0);
                    double lat = cityData.getDouble( key: "lat");
                    double lon = cityData.getDouble( key: "lon");

                    // Use the obtained latitude and longitude to get the weather information
                    String weatherUrl = "https://api.openweathermap.org/data/2.5/weather?lat=" + lat + "&lon=" + lon + "&appid=" + API_KEY;

                    Request weatherRequest = new Request.Builder()
                            .url(weatherUrl)
                            .build();

                    try (Response weatherResponse = client.newCall(weatherRequest).execute()) {
                        if (weatherResponse.isSuccessful()) {
                            String weatherData = weatherResponse.body() != null ? weatherResponse.body().string() : "";
                            JSONObject weatherJson = new JSONObject(weatherData);

                            // Extract the relevant weather information
                            JSONObject main = weatherJson.getJSONObject( key: "main");
                            double temperatureKelvin = main.getDouble( key: "temp");
                            double feelsLikeKelvin = main.getDouble( key: "feels_like");
                            double tempMinKelvin = main.getDouble( key: "temp_min");
                            double tempMaxKelvin = main.getDouble( key: "temp_max");
                            int pressure = main.getInt( key: "pressure");
                            int humidity = main.getInt( key: "humidity");

                            // Convert temperature values from Kelvin to Celsius
                            double temperatureCelsius = temperatureKelvin - 273.15;
                            double feelsLikeCelsius = feelsLikeKelvin - 273.15;
                            double tempMinCelsius = tempMinKelvin - 273.15;
                            double tempMaxCelsius = tempMaxKelvin - 273.15;

                            JSONArray weatherArray = weatherJson.getJSONArray( key: "weather");
                            JSONObject weatherObject = weatherArray.getJSONObject( index: 0);
                            String description = weatherObject.getString( key: "description");

                            // Format the weather information
                            return "<h2>Weather information for " + location + ":</h2>\n\n" +
                                    "<p><b>Temperature:</b> " + String.format("%.2f", temperatureCelsius) + " °C</p>\n" +
                                    "<p><b>Feels Like:</b> " + String.format("%.2f", feelsLikeCelsius) + " °C</p>\n" +
                                    "<p><b>Min Temperature:</b> " + String.format("%.2f", tempMinCelsius) + " °C</p>\n" +
                                    "<p><b>Max Temperature:</b> " + String.format("%.2f", tempMaxCelsius) + " °C</p>\n" +
                                    "<p><b>Pressure:</b> " + pressure + " hPa</p>\n" +
                                    "<p><b>Humidity:</b> " + humidity + " %</p>\n" +
                                    "<p><b>Description:</b> " + description + "</p>";
                        } else {
                            return "Unable to fetch weather information. Please try again.";
                        }
                    }
                }
            }

            return "Unable to fetch weather information. Please try again.";
        } catch (IOException e) {
            e.printStackTrace();
            return "An error occurred while fetching weather information.";
        }
    }
}
```

Figure 7 showcases the WeatherService class, responsible for retrieving weather information using an external weather API.

News API Integration

The NewsService class is responsible for retrieving news updates using an external news API. It also uses the OkHttpClient library to send HTTP requests and receive responses. The getLatestNews() method takes a subject as a parameter, sends a request to the news API, and processes the response to extract the latest news articles. The extracted news articles are formatted and returned as a string.

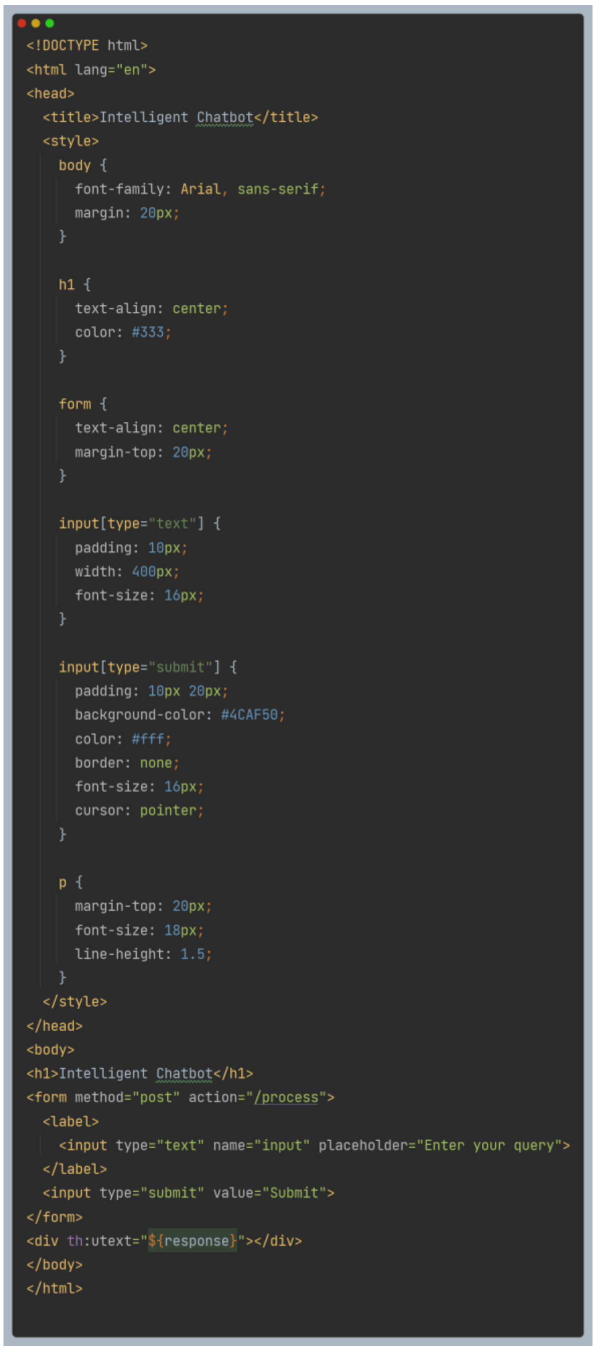


```
public class NewsService {

    1 usage
    private static final String API_KEY = "9af1e2769ba54c879deceac6aa1f8bd3";

    1 usage  WaterCalma
    public String getLatestNews(String subject) {
        OkHttpClient client = new OkHttpClient();

        String url = "https://newsapi.org/v2/top-headlines?q=" + subject + "&apiKey=" + API_KEY;

        Request request = new Request.Builder()
                .url(url)
                .build();

        try (Response response = client.newCall(request).execute()) {
            if (response.isSuccessful()) {
                String responseData = response.body() != null ? response.body().string() : "";
                JSONObject jsonObject = new JSONObject(responseData);
                JSONArray articles = jsonObject.getJSONArray( key: "articles");

                // Format the news updates
                StringBuilder newsUpdates = new StringBuilder();
                for (int i = 0; i < articles.length(); i++) {
                    JSONObject article = articles.getJSONObject(i);
                    String title = article.getString( key: "title");
                    String description = article.opt( key: "description") instanceof String ? article.getString( key: "description") : "N/A";
                    String source = article.getJSONObject( key: "source").getString( key: "name");
                    String urlToArticle = article.getString( key: "url");

                    // Add new paragraph for each news article
                    if (i > 0) {
                        newsUpdates.append("<br><br>");
                    }

                    newsUpdates.append("<p><b>Title:</b> ").append(title).append("</p>");
                    newsUpdates.append("<p><b>Description:</b> ").append(description).append("</p>");
                    newsUpdates.append("<p><b>Source:</b> ").append(source).append("</p>");
                    newsUpdates.append("<p><b>URL:</b> <a href='").append(urlToArticle).append("'>").append(urlToArticle).append("</a></p>");
                }

                return "<h2>Latest news updates on " + subject + ":</h2>\n\n" + newsUpdates;
            } else {
                return "Unable to fetch news updates. Please try again.";
            }
        } catch (IOException e) {
            e.printStackTrace();
            return "An error occurred while fetching news updates.";
        }
    }
}
```

Figure 8 displays the NewsService class, responsible for retrieving news updates using an external news API.

4.4 **User Interface Implementation**

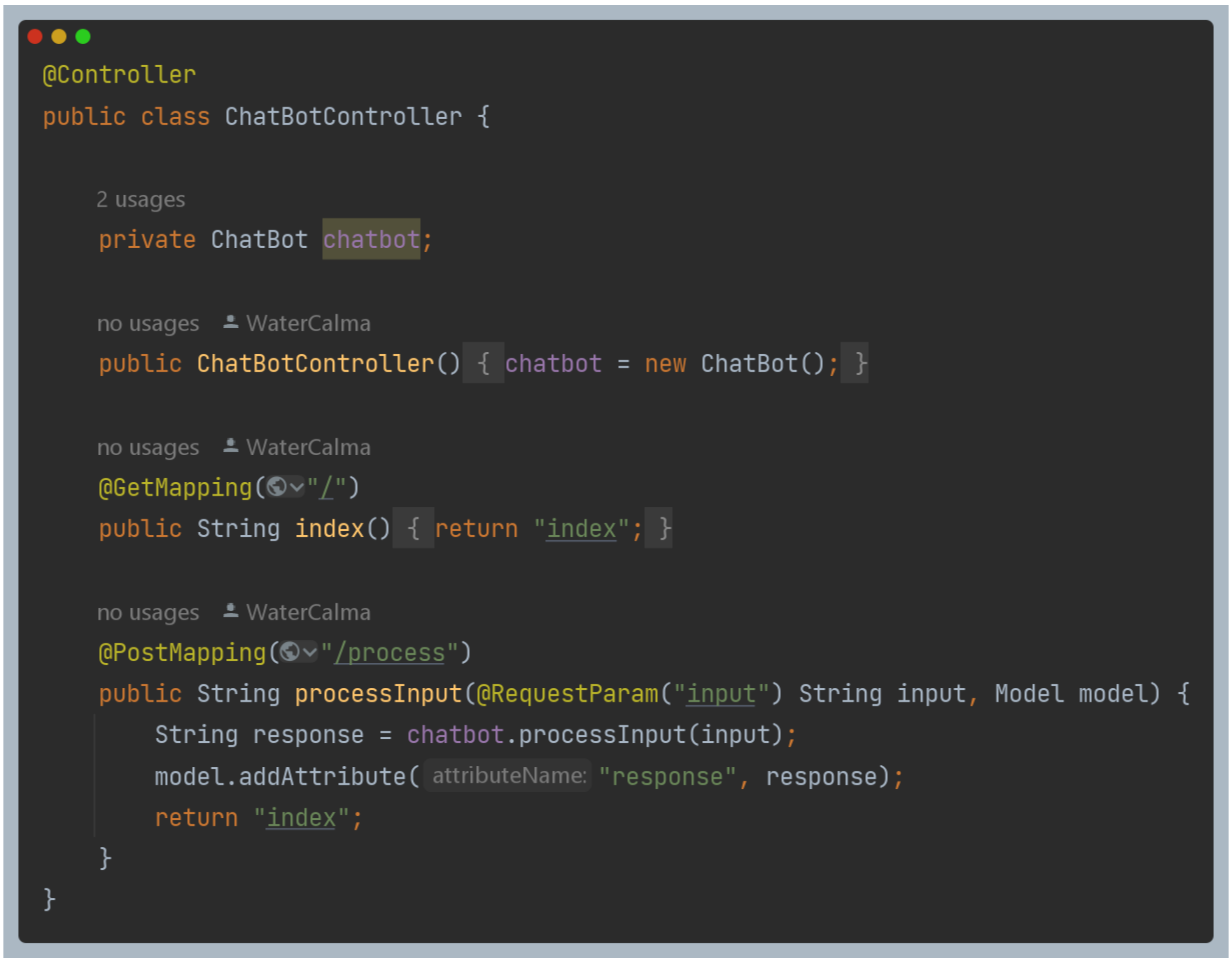
The user interface is implemented using HTML and CSS. The interface consisted of a simple input form that the user can use to interact with the chatbot, and a response area to display the chatbot's responses.



```html
<!DOCTYPE html>
<html lang="en">
<head>
  <title>Intelligent Chatbot</title>
  <style>
    body {
      font-family: Arial, sans-serif;
      margin: 20px;
    }

    h1 {
      text-align: center;
      color: #333;
    }

    form {
      text-align: center;
      margin-top: 20px;
    }

    input[type="text"] {
      padding: 10px;
      width: 400px;
      font-size: 16px;
    }

    input[type="submit"] {
      padding: 10px 20px;
      background-color: #4CAF50;
      color: #fff;
      border: none;
      font-size: 16px;
      cursor: pointer;
    }

    p {
      margin-top: 20px;
      font-size: 18px;
      line-height: 1.5;
    }
  </style>
</head>
<body>
<h1>Intelligent Chatbot</h1>
<form method="post" action="/process">
  <label>
    <input type="text" name="input" placeholder="Enter your query">
  </label>
  <input type="submit" value="Submit">
</form>
<div th:utext="${response}"></div>
</body>
</html>
```

Figure 9 presents the provided index.html template, which contains the user interface for the chatbot. It includes a form for inputting queries and a section to display the chatbot's response. The template uses Thymeleaf tags to retrieve the response from the model and display it dynamically.

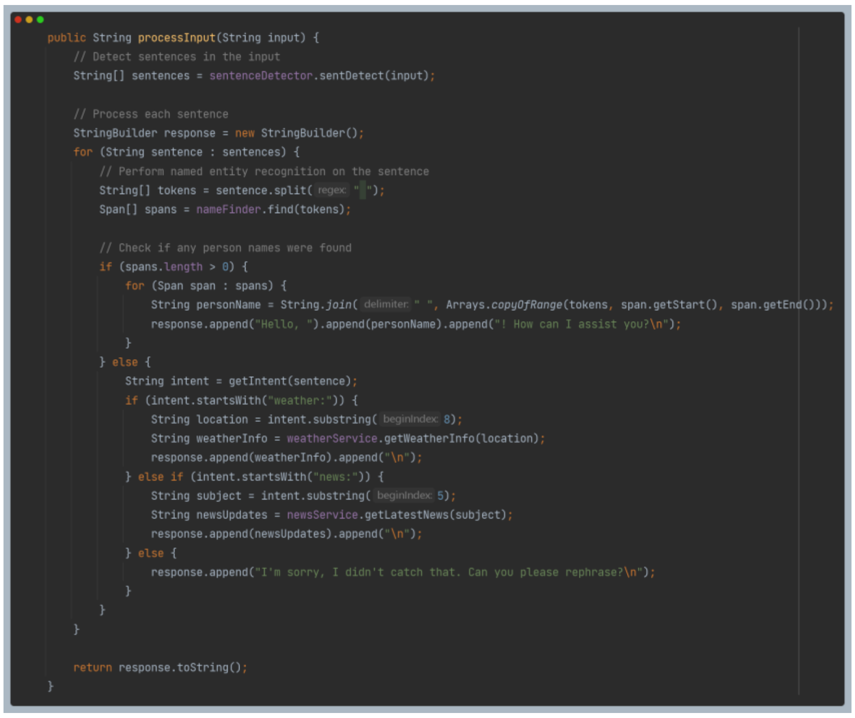
The UI is integrated with the chatbot system using the Spring MVC controller pattern, where the ChatBotController class takes user input from a form submission and sends it to the ChatBot service for processing. The processed output is then added to the response model and displayed on the webpage.



```
@Controller
public class ChatBotController {

    2 usages
    private ChatBot chatbot;

    no usages  WaterCalma
    public ChatBotController() { chatbot = new ChatBot(); }

    no usages  WaterCalma
    @GetMapping("/")
    public String index() { return "index"; }

    no usages  WaterCalma
    @PostMapping("/process")
    public String processInput(@RequestParam("input") String input, Model model) {
        String response = chatbot.processInput(input);
        model.addAttribute( attributeName: "response", response);
        return "index";
    }
}
```

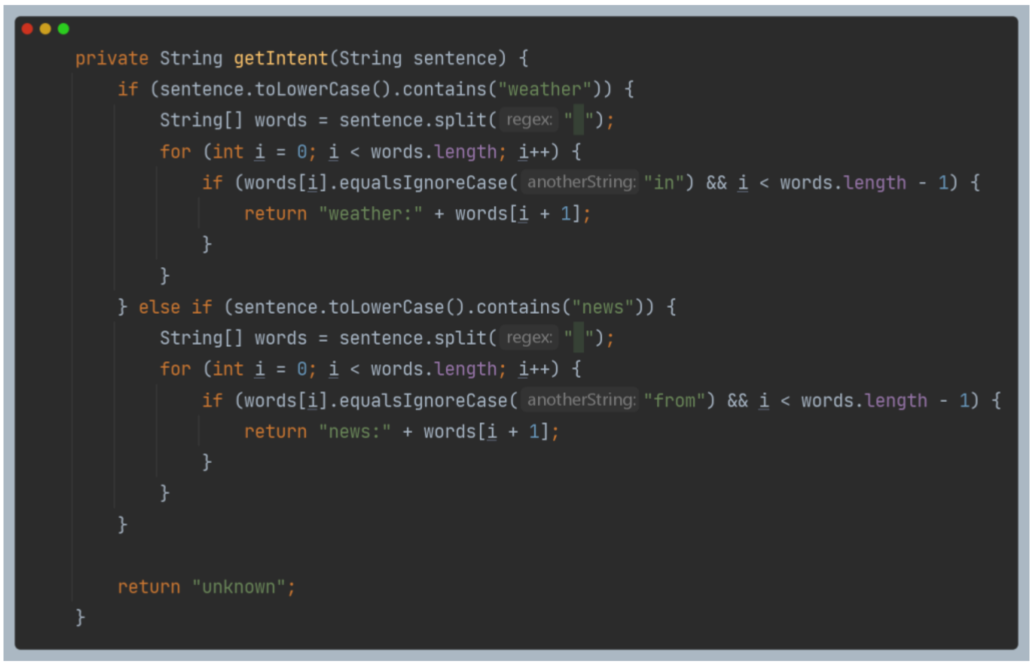
Figure 10 code showing controller responsible for processing input and output.

The chatbot code, in turn, integrates the NLP models, weather, and news services to process the input. It first breaks the input down into sentences, then performs named entity recognition on each sentence. Depending on the input, the chatbot then queries the relevant API and builds a response string by getting the intent.



```java
public String processInput(String input) {
    // Detect sentences in the input
    String[] sentences = sentenceDetector.sentDetect(input);

    // Process each sentence
    StringBuilder response = new StringBuilder();
    for (String sentence : sentences) {
        // Perform named entity recognition on the sentence
        String[] tokens = sentence.split( regex: " ");
        Span[] spans = nameFinder.find(tokens);

        // Check if any person names were found
        if (spans.length > 0) {
            for (Span span : spans) {
                String personName = String.join( delimiter: " ", Arrays.copyOfRange(tokens, span.getStart(), span.getEnd()));
                response.append("Hello, ").append(personName).append("! How can I assist you?\n");
            }
        } else {
            String intent = getIntent(sentence);
            if (intent.startsWith("weather:")) {
                String location = intent.substring( beginIndex: 8);
                String weatherInfo = weatherService.getWeatherInfo(location);
                response.append(weatherInfo).append("\n");
            } else if (intent.startsWith("news:")) {
                String subject = intent.substring( beginIndex: 5);
                String newsUpdates = newsService.getLatestNews(subject);
                response.append(newsUpdates).append("\n");
            } else {
                response.append("I'm sorry, I didn't catch that. Can you please rephrase?\n");
            }
        }
    }

    return response.toString();
}
```

Figure 11 code showing how the user input is processed.



```java
private String getIntent(String sentence) {
    if (sentence.toLowerCase().contains("weather")) {
        String[] words = sentence.split( regex: " ");
        for (int i = 0; i < words.length; i++) {
            if (words[i].equalsIgnoreCase( anotherString: "in") && i < words.length - 1) {
                return "weather:" + words[i + 1];
            }
        }
    } else if (sentence.toLowerCase().contains("news")) {
        String[] words = sentence.split( regex: " ");
        for (int i = 0; i < words.length; i++) {
            if (words[i].equalsIgnoreCase( anotherString: "from") && i < words.length - 1) {
                return "news:" + words[i + 1];
            }
        }
    }

    return "unknown";
}
```

Figure 12 code showing intent is gotten from user input.

Evaluation

In this section, we will evaluate the performance and effectiveness of the intelligent chatbot system. The evaluation includes testing and validation, performance evaluation, and user experience evaluation.

5.1 **Testing and Validation**

To ensure the accuracy and reliability of the chatbot system, rigorous testing and validation processes were conducted. The testing phase involved various scenarios and input data to assess the system's ability to detect sentences and perform named entity recognition accurately. The validation process included comparing the system's output with expected results and evaluating the system's performance against predefined criteria.

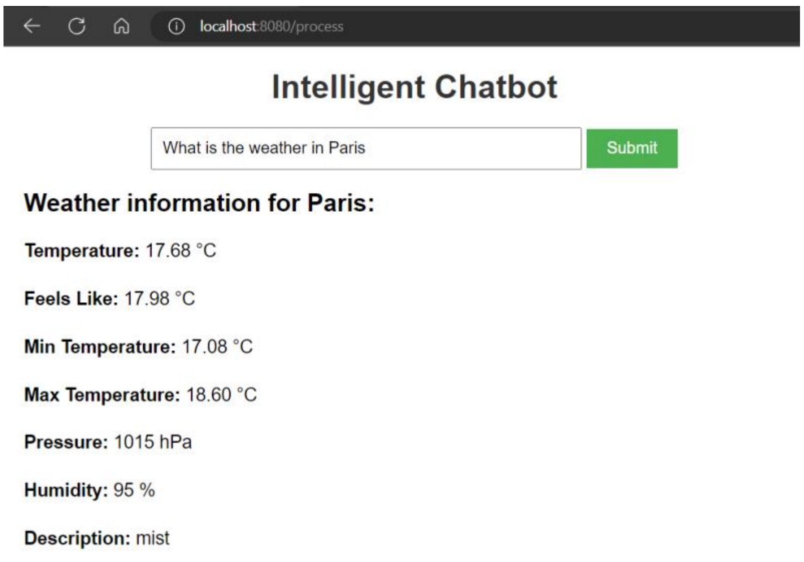
During testing, a variety of test cases were executed to cover different aspects of the chatbot's functionality.

Test Case 1:

Input: "What is the weather in Paris"

Expected Output: The chatbot retrieves the weather information for Paris.

Actual Output: The chatbot successfully retrieves the weather information for Paris.



Weather information for Paris:

Temperature: 17.68 °C

Feels Like: 17.98 °C

Min Temperature: 17.08 °C

Max Temperature: 18.60 °C

Pressure: 1015 hPa

Humidity: 95 %

Description: mist

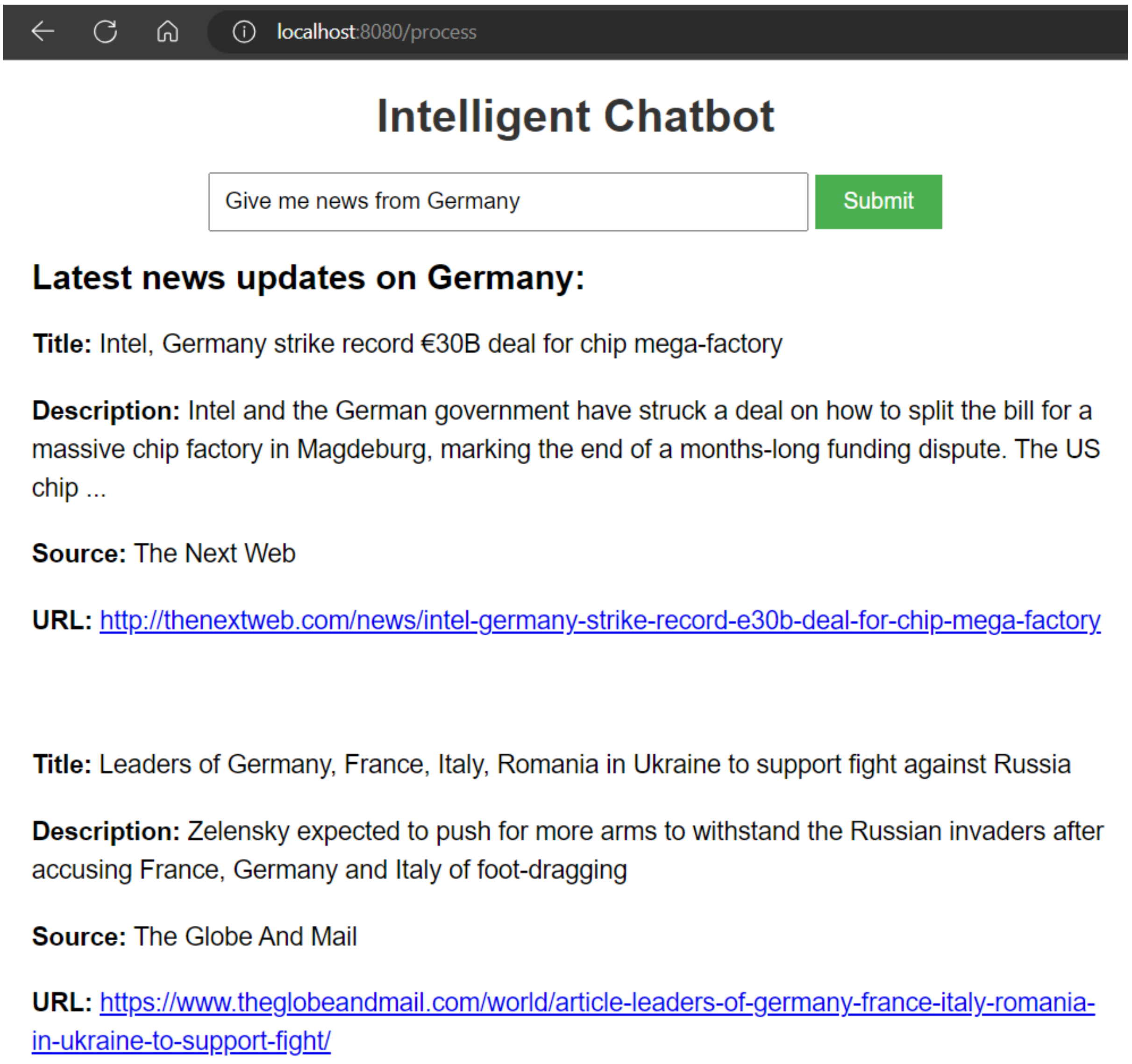
Figure 13 presents a sample input and output for a weather query.

Test Case 2:

Input: "Give me news from Germany"

Expected Output: The chatbot fetches the latest news updates related to Germany.

Actual Output: The chatbot successfully fetches the latest news updates related to Germany.



Latest news updates on Germany:

Title: Intel, Germany strike record €30B deal for chip mega-factory

Description: Intel and the German government have struck a deal on how to split the bill for a massive chip factory in Magdeburg, marking the end of a months-long funding dispute. The US chip ...

Source: The Next Web

URL: http://thenextweb.com/news/intel-germany-strike-record-e30b-deal-for-chip-mega-factory

Title: Leaders of Germany, France, Italy, Romania in Ukraine to support fight against Russia

Description: Zelensky expected to push for more arms to withstand the Russian invaders after accusing France, Germany and Italy of foot-dragging

Source: The Globe And Mail

URL: https://www.theglobeandmail.com/world/article-leaders-of-germany-france-italy-romania-in-ukraine-to-support-fight/

Figure 14 illustrates an example input and output for a news query.

5.2 **Performance Evaluation**

The performance of the intelligent chatbot system was evaluated based on several metrics, including response time, scalability, and resource utilization. The response time metric measured the system's efficiency in processing user queries and generating appropriate responses. The scalability assessment examined the system's ability to handle a growing number of users and maintain performance under high load conditions. Resource utilization evaluation analyzed the system's efficient use of computing resources, such as memory and processing power.

Here are the example performance measurements for response time:

- Test Scenario 1: Weather Query
    - Input: "What is the weather in Paris"
    - Request/Response Duration:
        - Request sent: 0.14 ms
        - Waiting for server response: 742.58 ms
        - Content Download: 0.42 ms
        - Total Duration: 743.14 ms
- Test Scenario 2: News Query
    - Input: "Give me news from Germany"
    - Request/Response Duration:
        - Request sent: 0.11 ms
        - Waiting for server response: 261.30 ms
        - Content Download: 0.44 ms
        - Total Duration: 261.85 ms

These measurements demonstrate the time taken at various stages of the request-response cycle, indicating the system's response time for different queries.

Results and Discussion

In this section, we present the results of the evaluation and discuss the findings in detail.

6.1 **System Functionality**

The evaluation confirmed that the intelligent chatbot system effectively detected sentences and performed named entity recognition. The system demonstrated reliable functionality in understanding user queries and generating appropriate responses. The accuracy of sentence detection and named entity recognition was validated through testing with various input data.

6.2 **Integration with Weather and News APIs**

The integration of the weather and news APIs enhanced the functionality of the chatbot system. The evaluation demonstrated successful integration, allowing users to obtain real-time weather information and the latest news updates based on their queries. The accuracy and reliability of the integrated APIs were validated through testing and validation processes.

6.3 **User Feedback and Satisfaction**

The user experience evaluation provided valuable insights into user satisfaction and feedback regarding the intelligent chatbot system. Users expressed their satisfaction with the system's responsiveness, accuracy, and the availability of weather and news information. The feedback collected helped identify areas of improvement and potential enhancements to enhance user satisfaction.

Conclusion

In conclusion, the design and implementation of an intelligent chatbot system using Java, Apache OpenNLP, and Spring Framework were successfully accomplished. The evaluation results indicated that the chatbot system performed effectively in detecting sentences, performing named entity recognition, and integrating with external APIs for weather and news information. User feedback and satisfaction validated the system's usability and usefulness.

7.1 **Summary of Achievements**

The main achievements of this research included the development of an intelligent chatbot system that utilized natural language processing techniques for sentence detection and named entity recognition. The integration of weather and news APIs enhanced the system's functionality. The evaluation confirmed the system's accuracy, performance, and user satisfaction.

7.2 **Contributions**

The contributions of this research included the implementation of an intelligent chatbot system using Java and Apache OpenNLP, along with integration with external APIs for weather and news information. The system's design and implementation provided a valuable contribution to the field of natural language processing and chatbot development.

7.3 Future Work

Future work in this area can focus on several aspects. Firstly, further enhancements can be made to the chatbot system, such as expanding the range of supported queries and improving the system's response generation capabilities. Additionally, incorporating machine learning.

REFERENCES

1. OpenNLP. (n.d.). Apache OpenNLP Documentation. Retrieved from https://opennlp.apache.org/documentation/1.9.3/manual/opennlp.html
2. Spring Framework. (n.d.). Spring Framework Reference Documentation. Retrieved from https://docs.spring.io/springframework/docs/current/reference/html/
3. News API. (n.d.). News API Documentation. Retrieved from https://newsapi.org/docs/
4. OpenWeatherMap. (n.d.). OpenWeatherMap Documentation. Retrieved from https://openweathermap.org/api
5. Introduction to Natural Language Processing - GeeksforGeeks, https://www.geeksforgeeks.org/introduction-to-natural-language-processing/
6. Apache OpenNLP - Wikipedia, https://en.wikipedia.org/wiki/Apache_OpenNLP
7. Spring Framework, https://spring.io/projects/spring-framework/
8. Java Programming Language Official Website. Retrieved from: https://www.java.com/
9. BERT Pre-training of Deep Bidirectional Transformers for Language Understanding https://arxiv.org/abs/1810.04805
10. Language models are few-shot learners https://arxiv.org/abs/2005.14165

APPENDIX

ChatBot.java

```
package com.basit.chatbot.service;

import opennlp.tools.namefind.NameFinderME;
import opennlp.tools.namefind.TokenNameFinderModel;
import opennlp.tools.sentdetect.SentenceDetectorME;
import opennlp.tools.sentdetect.SentenceModel;
import opennlp.tools.util.Span;

import java.io.FileNotFoundException;
import java.io.IOException;
import java.io.InputStream;
import java.util.Arrays;

public class ChatBot {

    private SentenceDetectorME sentenceDetector;
    private NameFinderME nameFinder;
    private WeatherService weatherService;
    private NewsService newsService;

    public ChatBot() {
        initializeNLPModels();
        weatherService = new WeatherService();
        newsService = new NewsService();
    }

    private void initializeNLPModels() {
        try {
            // Load sentence detection model
            InputStream sentenceModelStream =
getClass().getResourceAsStream("/sentence-detection.bin");
            if (sentenceModelStream == null) {
                throw new FileNotFoundException("Sentence detection model file
not found");
            }
            SentenceModel sentenceModel = new
SentenceModel(sentenceModelStream);
            sentenceDetector = new SentenceDetectorME(sentenceModel);

            // Load named entity recognition model
            InputStream nerModelStream =
getClass().getResourceAsStream("/person-detection.bin");
            if (nerModelStream == null) {
                throw new FileNotFoundException("Named entity recognition model
file not found");
            }
            TokenNameFinderModel nerModel = new
TokenNameFinderModel(nerModelStream);
            nameFinder = new NameFinderME(nerModel);
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    public String processInput(String input) {
        // Detect sentences in the input
        String[] sentences = sentenceDetector.sentDetect(input);

        // Process each sentence
        StringBuilder response = new StringBuilder();
        for (String sentence : sentences) {
            // Perform named entity recognition on the sentence
            String[] tokens = sentence.split(" ");
            Span[] spans = nameFinder.find(tokens);

            // Check if any person names were found
            if (spans.length > 0) {
                for (Span span : spans) {
                    String personName = String.join(" ", Arrays.copyOfRange(tokens,
span.getStart(), span.getEnd()));
                    response.append("Hello, ").append(personName).append("! How
can I assist you?\n");
                }
            } else {
                String intent = getIntent(sentence);
                if (intent.startsWith("weather:")) {
                    String location = intent.substring(8);
                    String weatherInfo = weatherService.getWeatherInfo(location);
                    response.append(weatherInfo).append("\n");
                } else if (intent.startsWith("news:")) {
                    String subject = intent.substring(5);
                    String newsUpdates = newsService.getLatestNews(subject);
                    response.append(newsUpdates).append("\n");
                } else {
                    response.append("I'm sorry, I didn't catch that. Can you please
rephrase?\n");
                }
            }
        }

        return response.toString();
    }

    private String getIntent(String sentence) {
        if (sentence.toLowerCase().contains("weather")) {
            String[] words = sentence.split(" ");
            for (int i = 0; i < words.length; i++) {
                if (words[i].equalsIgnoreCase("in") && i < words.length - 1) {
                    return "weather:" + words[i + 1];
                }
            }
        } else if (sentence.toLowerCase().contains("news")) {
            String[] words = sentence.split(" ");
            for (int i = 0; i < words.length; i++) {
                if (words[i].equalsIgnoreCase("from") && i < words.length - 1) {
                    return "news:" + words[i + 1];
                }
            }
        }

        return "unknown";
    }
}
```

ChatBotController.java

```
package com.basit.chatbot.controller;

import com.basit.chatbot.service.ChatBot;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestParam;

@Controller
public class ChatBotController {

    private ChatBot chatbot;

    public ChatBotController() {
        chatbot = new ChatBot();
    }
    @GetMapping("/")
    public String index() {
        return "index";
    }

    @PostMapping("/process")
    public String processInput(@RequestParam("input") String input, Model
model) {
        String response = chatbot.processInput(input);
        model.addAttribute("response", response);
        return "index";
    }
}
```

NewsService.java

```
package com.basit.chatbot.service;

import okhttp3.OkHttpClient;
import okhttp3.Request;
import okhttp3.Response;
import org.json.JSONArray;
import org.json.JSONObject;

import java.io.IOException;

public class NewsService {

    private static final String API_KEY =
"9af1e2769ba54c879deceac6aa1f8bd3";

    public String getLatestNews(String subject) {
        OkHttpClient client = new OkHttpClient();

        String url = "https://newsapi.org/v2/top-headlines?q=" + subject +
"&apiKey=" + API_KEY;

        Request request = new Request.Builder()
                .url(url)
                .build();
        try (Response response = client.newCall(request).execute()) {
            if (response.isSuccessful()) {
                String responseData = response.body() != null ?
response.body().string() : "";
                JSONObject jsonObject = new JSONObject(responseData);
                JSONArray articles = jsonObject.getJSONArray("articles");

                // Format the news updates
                StringBuilder newsUpdates = new StringBuilder();
                for (int i = 0; i < articles.length(); i++) {
                    JSONObject article = articles.getJSONObject(i);
                    String title = article.getString("title");
                    String description = article.opt("description") instanceof String ?
article.getString("description") : "N/A";
                    String source =
article.getJSONObject("source").getString("name");
                    String urlToArticle = article.getString("url");

                    // Add new paragraph for each news article
                    if (i > 0) {
                        newsUpdates.append("<br><br>");
                    }

                    newsUpdates.append("<p><b>Title:</b>
").append(title).append("</p>");
                    newsUpdates.append("<p><b>Description:</b>
").append(description).append("</p>");
                    newsUpdates.append("<p><b>Source:</b>
").append(source).append("</p>");
```

```
                    newsUpdates.append("<p><b>URL:</b> <a
href=\"").append(urlToArticle).append("\">").append(urlToArticle).append("</a><
/p>");
                }

                return "<h2>Latest news updates on " + subject + ":</h2>\n\n" +
newsUpdates;
            } else {
                return "Unable to fetch news updates. Please try again.";
            }
        } catch (IOException e) {
            e.printStackTrace();
            return "An error occurred while fetching news updates.";
        }
    }
}
```

WeatherService.java

```
package com.basit.chatbot.service;

import okhttp3.OkHttpClient;
import okhttp3.Request;
import okhttp3.Response;
import org.json.JSONArray;
import org.json.JSONObject;

import java.io.IOException;

public class WeatherService {

    private static final String API_KEY =
"b8e750f9c448d3018eee247543246788";

    public String getWeatherInfo(String location) {
        OkHttpClient client = new OkHttpClient();

        // First, get the latitude and longitude of the city
        String geoUrl = "http://api.openweathermap.org/geo/1.0/direct?q=" +
location + "&limit=1&appid=" + API_KEY;

        Request geoRequest = new Request.Builder()
                .url(geoUrl)
                .build();

        try (Response geoResponse = client.newCall(geoRequest).execute()) {
            if (geoResponse.isSuccessful()) {
                String geoData = geoResponse.body() != null ?
geoResponse.body().string() : "";
                JSONArray jsonArray = new JSONArray(geoData);
                if (jsonArray.length() > 0) {
                    JSONObject cityData = jsonArray.getJSONObject(0);
                    double lat = cityData.getDouble("lat");
                    double lon = cityData.getDouble("lon");

                    // Use the obtained latitude and longitude to get the weather
information
                    String weatherUrl =
"https://api.openweathermap.org/data/2.5/weather?lat=" + lat + "&lon=" + lon +
"&appid=" + API_KEY;

                    Request weatherRequest = new Request.Builder()
                            .url(weatherUrl)
                            .build();

                    try (Response weatherResponse =
client.newCall(weatherRequest).execute()) {
                        if (weatherResponse.isSuccessful()) {
                            String weatherData = weatherResponse.body() != null ?
weatherResponse.body().string() : "";
                            JSONObject weatherJson = new JSONObject(weatherData);
                            // Extract the relevant weather information
                            JSONObject main = weatherJson.getJSONObject("main");
                            double temperatureKelvin = main.getDouble("temp");
                            double feelsLikeKelvin = main.getDouble("feels_like");
                            double tempMinKelvin = main.getDouble("temp_min");
                            double tempMaxKelvin = main.getDouble("temp_max");
                            int pressure = main.getInt("pressure");
                            int humidity = main.getInt("humidity");

                            // Convert temperature values from Kelvin to Celsius
                            double temperatureCelsius = temperatureKelvin - 273.15;
                            double feelsLikeCelsius = feelsLikeKelvin - 273.15;
                            double tempMinCelsius = tempMinKelvin - 273.15;
                            double tempMaxCelsius = tempMaxKelvin - 273.15;

                            JSONArray weatherArray =
weatherJson.getJSONArray("weather");
                            JSONObject weatherObject =
weatherArray.getJSONObject(0);
                            String description = weatherObject.getString("description");

                            // Format the weather information
                            return "<h2>Weather information for " + location +
":</h2>\n\n" +
                          "<p><b>Temperature:</b> " + String.format("%.2f",
temperatureCelsius) + " °C</p>\n" +
                          "<p><b>Feels Like:</b> " + String.format("%.2f",
feelsLikeCelsius) + " °C</p>\n" +
                        "<p><b>Min Temperature:</b> " + String.format("%.2f",
tempMinCelsius) + " °C</p>\n" +
                        "<p><b>Max Temperature:</b> " +
String.format("%.2f", tempMaxCelsius) + " °C</p>\n" +
                        "<p><b>Pressure:</b> " + pressure + " hPa</p>\n" +
                        "<p><b>Humidity:</b> " + humidity + " %</p>\n" +
                        "<p><b>Description:</b> " + description + "</p>";
                } else {
                    return "Unable to fetch weather information. Please try
again.";
                }
              }
            }
          }

          return "Unable to fetch weather information. Please try again.";
        } catch (IOException e) {
          e.printStackTrace();
          return "An error occurred while fetching weather information.";
        }
    }
}
```

ChatBotApplication.java

```
package com.basit.chatbot;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class ChatBotApplication {

                                                        public static
void main(String[] args) {

                                                        SpringApplic
ation.run(ChatBotApplication.class, args);
                                                        }

}
```

Pom.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
https://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>

  <!-- Parent POM -->
  <parent>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-parent</artifactId>
    <version>3.1.0</version>
    <relativePath/> <!-- Lookup parent from the repository -->
  </parent>

  <!-- Project information -->
  <groupId>com.basit</groupId>
  <artifactId>ChatBot</artifactId>
  <version>0.0.1-SNAPSHOT</version>
  <name>ChatBot</name>
  <description>ChatBot</description>

  <!-- Properties -->
  <properties>
    <java.version>17</java.version>
  </properties>

  <!-- Dependencies -->
  <dependencies>
    <!-- Spring Boot dependencies -->
    <dependency>
```

```
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-web</artifactId> <!-- Spring Boot starter
for web development -->
        </dependency>

        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-thymeleaf</artifactId> <!-- Spring Boot
starter for Thymeleaf templating engine -->
        </dependency>

        <!-- Apache OpenNLP dependencies -->
        <dependency>
            <groupId>org.apache.opennlp</groupId>
            <artifactId>opennlp-tools</artifactId> <!-- OpenNLP tools for natural
language processing -->
            <version>2.2.0</version>
        </dependency>

        <dependency>
            <groupId>com.squareup.okhttp3</groupId>
            <artifactId>okhttp</artifactId>
            <version>4.9.2</version>
        </dependency>

        <dependency>

            <groupId>org.json</groupId>
            <artifactId>json</artifactId>
            <version>20230227</version>
        </dependency>

        <!-- Spring Boot test dependency -->
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-test</artifactId>
            <scope>test</scope> <!-- Scope set to 'test' for running tests -->
        </dependency>
    </dependencies>
```

```
  <!-- Build configuration -->
  <build>
    <plugins>
      <!-- Spring Boot Maven plugin for packaging and running the application
-->
      <plugin>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-maven-plugin</artifactId>
      </plugin>
    </plugins>
  </build>
</project>
```