OmaNews

Software Design Report

Table of contents

Introduction

Project Overview

OmaNews is a Sentiment Analysis News Feed Application that enables users to explore and filter news articles based on sentiment (positive, neutral, or negative) and visualize trends over time. By using sentiment analysis, the application helps users understand the general mood of news coverage on various topics, offering personalized insights based on user preferences.

The backend of the application is built with Spring Boot and MongoDB to manage news retrieval, sentiment analysis, and user preferences. The frontend, built with React, provides users with an interactive interface to filter news by sentiment and visualize trends.

Objectives and Scope

The primary objective of this project is to provide users with a news feed that can be filtered based on sentiment, allowing them to:

- View news sentiment trends.
- Filter news based on specific topics.
- Visualize trends over time for selected topics.

Use of AI Tools

OpenAI’s GPT was utilized to clarify and refine the project's final implementation ideas. The model was used to generate summaries and propose structured designs for various components of the system.

Functional and Non-Functional Requirements

Course requirements

Data retrieval

API 1: News data from NEWS API
API 2: Sentiment analysis from Twinword Sentiment Analysis API

Data combination and visualization

Retrieved news are analyzed for sentiment score and then the score is added to articles. News in the feed have a mark showing if it is positive, neutral or negative. Charts with overall trends in news or trends for specific topics are shown to users.

User preferences

Users can switch news sentiment in the main feed. Users as well can see charts for specific topics and can set timeframes. These settings will be saved.

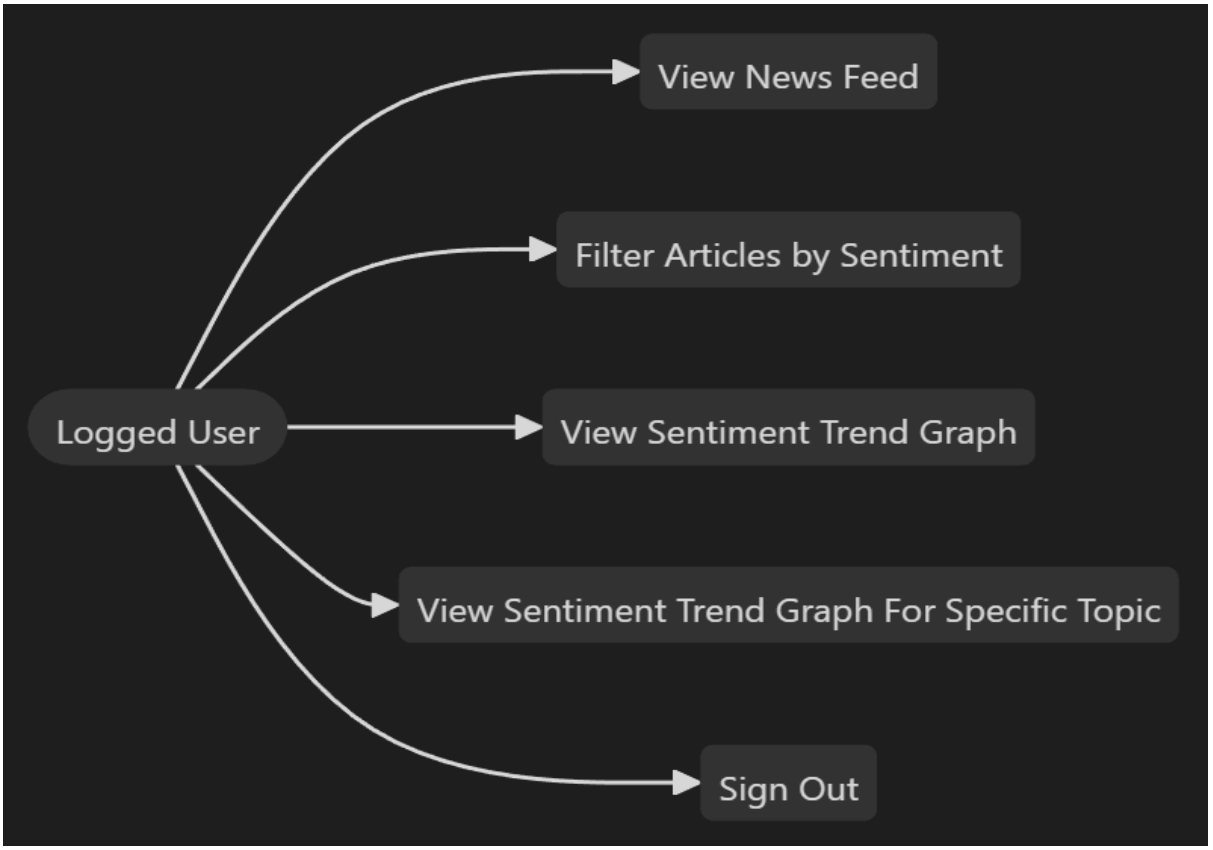
Use cases

Use Case: News Filtering

- **Actors**: User
- **Description**: The user selects a sentiment (positive/negative/neutral) to filter news articles in the feed. The system fetches articles with the matching sentiment and displays them.

Use Case: Trend Visualization

- **Actors**: User
- **Description**: The user selects a topic and time period (daily, weekly, monthly) to visualize sentiment trends. The system generates a chart showing sentiment fluctuations over time.



Functional Requirements

- **News Retrieval**: Fetch news articles from the **NewsAPI**.
- **Sentiment Analysis**: Analyze the sentiment of each article using the **Twinword Sentiment Analysis API** and tag it as positive, neutral, or negative.
- **User Preferences**: Allow users to save their favorite topics and sentiment preferences.
- **Visualization**: Display trends over time for user-selected topics, allowing users to filter news based on sentiment and timeframes.

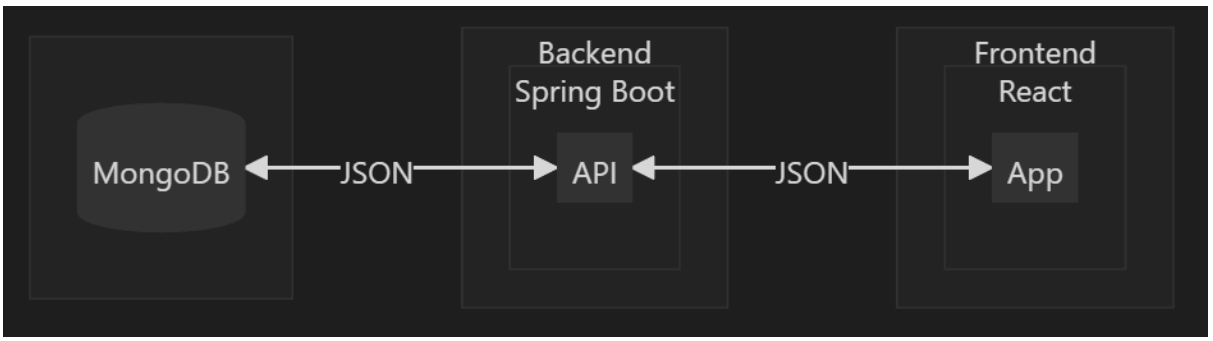
Non-Functional Requirements

- **Performance**: The system should process and display results within 1-2 seconds after a user request.
- **Scalability**: The system must handle increased traffic by efficiently managing API requests and database operations.
- **Security**: User data, including preferences and login information, should be securely stored and transmitted.
- **Usability**: The interface must be intuitive and easy to navigate, with clear options for filtering and visualization.

System Overview

The system overview diagram describes the high-level architecture of the sentiment analysis application, showcasing the interaction between the backend, frontend, and database components.

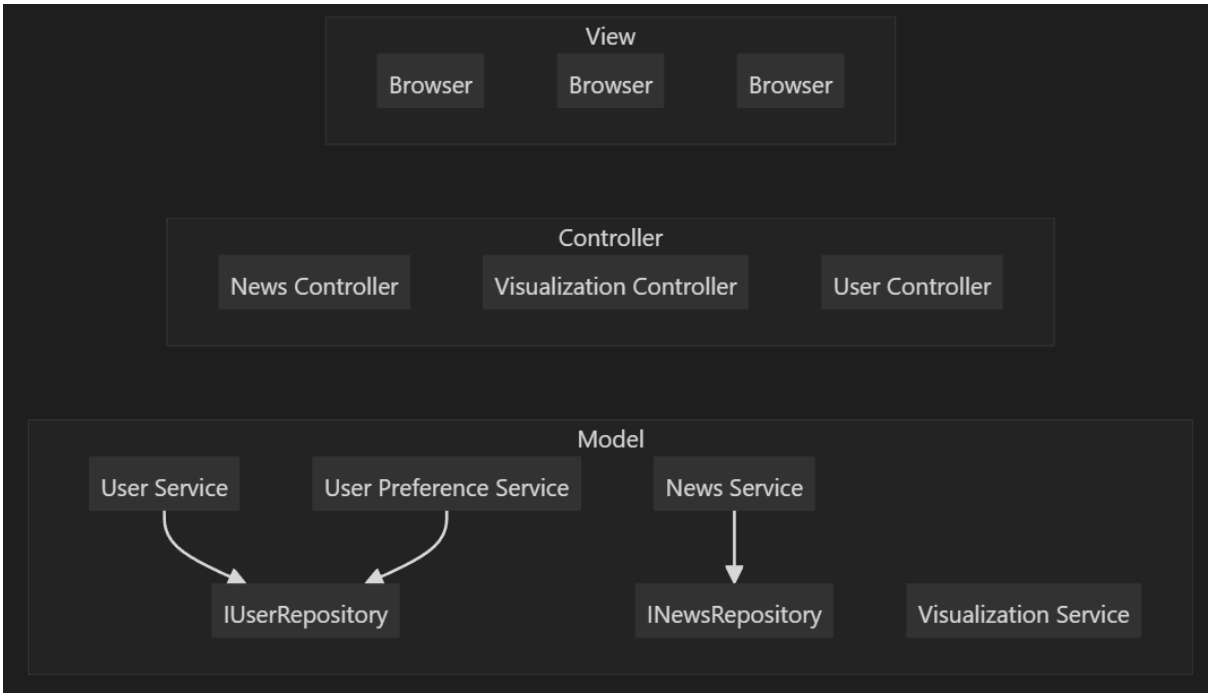
- **MongoDB**: Acts as the database for storing news articles and user preferences.
- **Backend (Spring Boot)**: Implements the application's core logic and APIs, fetching data from external sources, processing it, and interacting with MongoDB.
- **Frontend (React)**: The user interface that enables users to interact with the application, visualize trends, and filter articles based on sentiment.
- **JSON**: Represents the data format exchanged between the frontend, backend, and database.



System Components

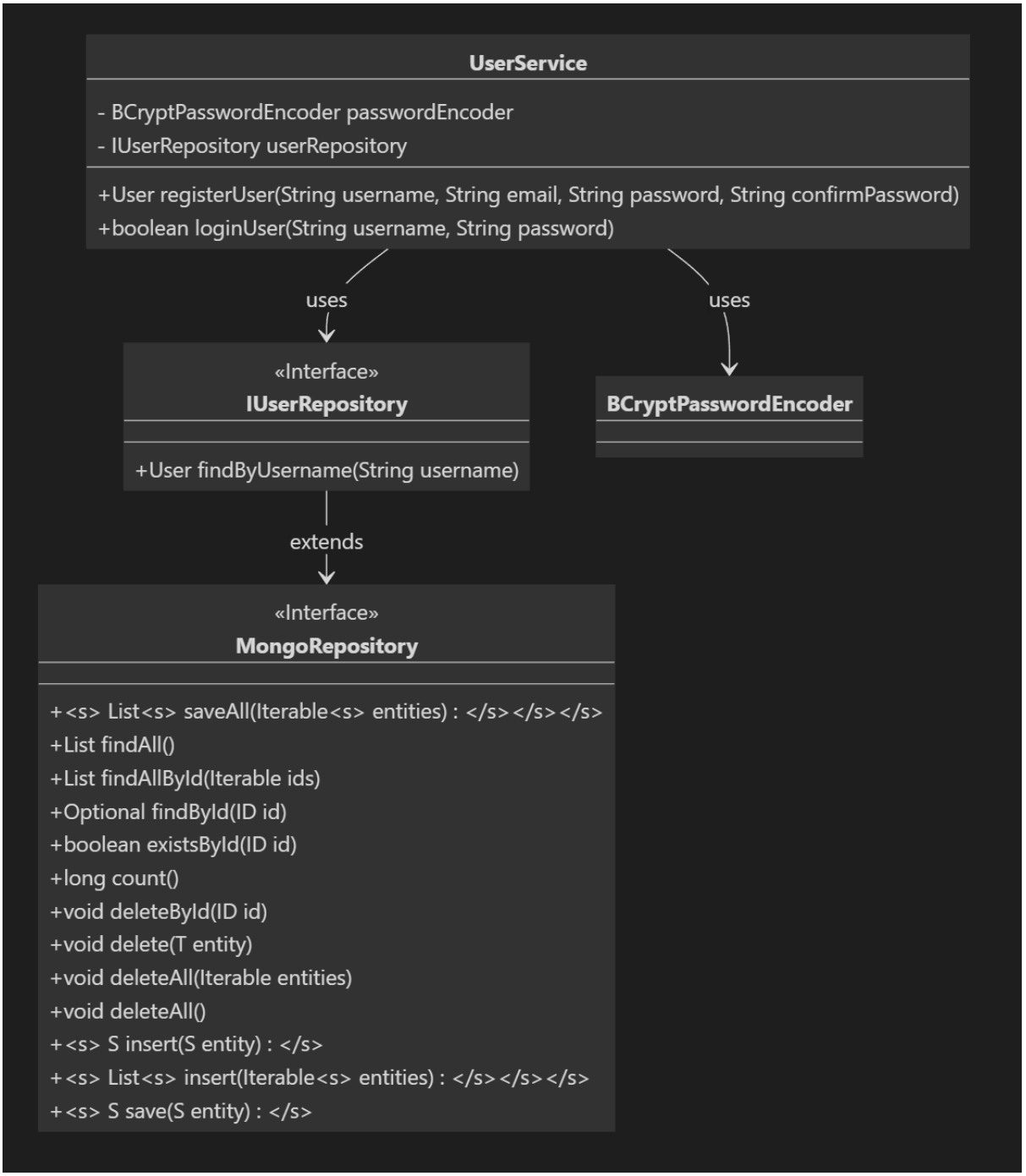
Backend

This section provides an overview of the backend structure, organized using the MVC (Model-View-Controller) pattern. The **View** represents the frontend browsers that communicate with the backend controllers. In the **Controller** layer, the **NewsController** manages requests related to fetching and displaying news, the **VisualizationController** handles requests for generating sentiment visualizations, and the **UserController** manages user-related requests such as authentication and preferences. The **Model** layer contains the core business logic and data handling services, including the **UserService**, which manages user operations such as registration and login, and the **NewsService**, which is responsible for fetching and processing news articles from external sources. The **VisualizationService** generates data for visualizations by providing sentiment trends, while the **PreferencesService** handles user preferences for news filtering and visualization options. The **IUserRepository** and **INewsRepository** serve as interfaces for accessing and managing user and news article data within the database, respectively.



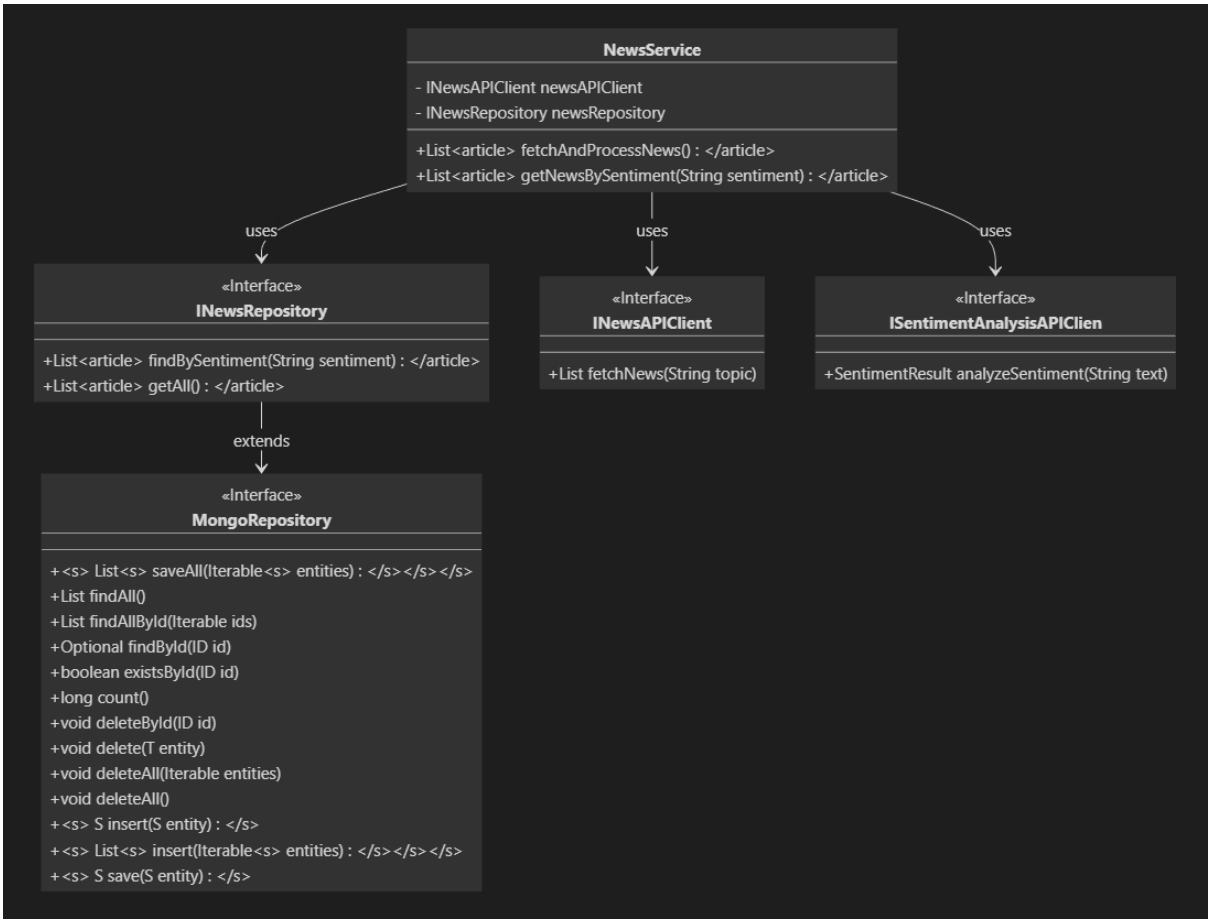
UserService

The UserService class is responsible for user management tasks, including registration, login, and retrieving user information.



NewsService

The NewsService class is responsible for fetching news articles from an external API, processing them for sentiment analysis, and storing them in MongoDB.



VisualizationService

The UserService class is responsible for generating and combining data for charts.

UserPreferenceService

The UserPreferenceService class is responsible for storing users favorite topics, sentiment preference and data periods.

Design Patterns and Principles

The design of this application follows several established design patterns:

The **Repository Pattern** is used in the **INewsRepository** and **IUserRepository** interfaces, providing a clear separation between the data access layer and the business logic. This

pattern allows for flexibility in changing the data source without affecting other parts of the application, making it easier to switch between different databases or storage mechanisms if needed.

The **Factory Pattern** is implicitly used by the Spring framework to create instances of controllers, services, and repositories. This ensures that objects are created with all their dependencies injected, following the Inversion of Control (IoC) principle.

Moreover, **OmaNews's** design is based on the **SOLID** designing principles, which offer an easy maintenance and scalable system architecture.

The **Single Responsibility** Principle encourages the division of the application logic into distinct services like NewsService and UserService, where each service does not take up more than its share of the application's logic.

The **Open/Closed Principle** is applicable to the system because it is built in a way that encourages expansion without interference with the internal structure of the system, as in the case of adding new APIs. It can also be seen to also resolve the dependency inversion principle.

In other words, this system is designed in such a mechanism that it uses interfaces such as INewsRepository and IUserRepository but not the objects. Therefore, it will be very convenient to replace certain parts, like a database, for instance, without disturbing the whole system.

Prototype

The design prototype for the OmaNews project is available on Figma. This prototype includes all the UI/UX designs, user flows, and interactions that are planned for the project. It serves as a visual guide and reference for developers, designers, and stakeholders.

Accessing the Prototype

To view the prototype, please use the following Figma link:

[OmaNews Figma Prototype]

Security and Error Handling

User Authentication

We prioritize the security of user login information by employing JSON Web Tokens (JWT) for session management. This ensures that user credentials are securely stored and transmitted, providing a robust layer of protection against unauthorized access.

Error Handling

Our system features centralized error handling to effectively manage and log issues such as API timeouts, database failures, or invalid data formats. This approach ensures that any problems are promptly identified and addressed, while users receive appropriate and informative responses.

Testing Strategy

Unit Testing

We rigorously test individual services, such as the NewsService, using mock data to ensure they function correctly. This granular level of testing helps us catch and fix issues early in the development process.

Integration Testing

We conduct end-to-end tests to verify that data flows seamlessly between the frontend and backend. This comprehensive testing ensures that all components of the system work together as intended.

User Acceptance Testing (UAT)

We engage with users to validate that the application meets their requirements for features like filtering, sentiment visualization, and preference management. This user-centric approach ensures that the final product aligns with user expectations and needs.

Deployment Strategy

CI/CD

We leverage a CI/CD pipeline to automate updates and scaling, ensuring that our application remains up-to-date and can handle varying levels of demand with ease. By focusing on these strategies, we aim to deliver a secure, reliable, and user-friendly application that meets the highest standards of quality and performance.

Conclusion

OmaNews provides a platform for users to filter news based on sentiment and visualize trends over time. The project uses advanced AI tools for sentiment analysis and scalable architecture to handle user preferences and data retrieval efficiently. This project demonstrates a robust integration of backend, frontend, and external APIs to deliver a sentiment-based news experience.