Stack overflow application

Documentation

Manoliu Silvia

Group 30432

# Introduction

In this project, we aim to develop a simplified version of StackOverflow, a popular question-and-answer platform. The system will allow users to ask questions, provide answers, and vote on both questions and answers.

# Tech Stack

Backend: Java Spring Boot

Frontend: Angular

# Software Architecture

## Presentation Layer (Frontend - Angular):

Responsible for presenting the user interface to the users.

Handles user interactions and sends requests to the backend.

Receives and displays data from the backend.

## Application Layer (Backend - Java Spring Boot):

Contains the business logic of the application.

Handles requests from the frontend, processes them, and interacts with the database.

Manages user authentication, authorization, and data validation.

## Data Layer (Database):

Stores and retrieves data required by the application.

Consists of a relational database (e.g., MySQL) to store user information, questions, answers, tags, and votes.

# Functional Requirements

## Feature 1

Users shall be able to ask questions. Each question must have an author, title, text,

creation date & time, picture and one or more tags. If an appropriate tag does not exist,

the user must be able to create one.

The list of questions shall be displayed, sorted by creation date. The most recent

question should be displayed first.

Questions may be edited or deleted by their author.

The user must be able to filter questions by tag, via a text search, via users or for his

own questions. The text search should check the question title.

## Feature 2

Each question may be answered one or more times by any user (including the original

author).

Each answer must have an author, text, picture and creation date & time.

Answers may be edited or deleted by their author.

When displaying a question individually, the list of answers must also be displayed

## Feature 3

Users may vote questions and answers (upvote and downvote, like and dislike).

Each user may only vote once on each question or answer. Users cannot vote on their

own answers or questions (Like&Dislike).

On each voted question or answer, the vote count must be displayed

(vote count = upvote/like count - downvote/dislike count). The vote count can be

negative.

The answers for a question must be sorted by their vote count. Answers with the highest

vote count must be displayed first.

# Non-functional requirements

## Security:

User passwords must be stored securely in the database, encrypted using a strong hashing algorithm.

Actions must be restricted to logged-in users only.

## Performance:

The application should handle a large number of concurrent users and maintain responsiveness.

Database queries should be optimized for performance, especially when retrieving questions and answers.

## Scalability:

The architecture should support horizontal scalability to accommodate increasing user loads.

Components should be loosely coupled to facilitate scalability.

## Usability:

The user interface should be intuitive and easy to navigate.

Error messages should be clear and informative.

## Reliability:

The application should be reliable and resilient to failures.

Proper error handling and logging mechanisms should be implemented.

# Diagrams

## Class diagram:

A computer screen shot of a diagram

Description automatically generated

## Use case diagram

A screenshot of a computer screen

Description automatically generated

## Database diagram

A diagram of a question

Description automatically generated

# Implementation

## Entities:

Entity classes were created to represent the core domain objects of the application. The entities are Question, Answer, Tag, User, QuestionTag, QuestionVote & AnswerVote. These classes define the structure of the data stored in the database. The class QuestionTag was created to avoid the many to many relationship between Question and Tag and the class and the class AddQuestionRequest is a class which is not present in the database but it helps request the question and tags at the same time.

## Repository Interfaces:

Repository interfaces were defined to interact with the database and perform CRUD operations on the entity objects. Spring Data JPA was utilized to simplify data access code.

## Service Package:

Service classes were implemented to encapsulate the business logic of the application. These classes orchestrate interactions between the controller layer and the repository layer, handling tasks such as user authentication, question and answer management, voting, etc. For each main class there is a dedicated service layer which has the required functions implemented using the repositories

## Controller package:

The bridge between frontend and backend used for mapping the implemented functions.

# Frontend

For the frontend part I chose to create it using React since I consider it to be a more popular alternative. The dependencies used are Routing Dom and Material U

## Pages Overview

Our application comprises various pages designed to cater to different user interactions:

User Page: Displays a list of users and their details. Created using React components and fetched user data through the UserService.

Login Page: Provides a login interface for users to authenticate themselves. Implemented with React components and utilizes the LoginService for authentication.

Question Page: Showcases a collection of questions with options for filtering by various criteria like user, tag, or text. Leveraged React components and fetched question data using the QuestionService.

Add Question Page: Enables users to submit new questions along with relevant tags. Built with React components and utilizes AddQuestionService and TagService for adding questions and tags respectively.

Edit Question Page: Allows users to modify existing questions. Utilizes React components and QuestionService for updating question details.

Answer Page: Displays answers to a specific question and provides an interface for users to add new answers. Utilizes React components and AnswerService.

Add Answer Page: Enables users to submit answers to questions. Created with React components and utilizes AnswerService.

## Services Overview

To facilitate seamless communication with our backend, we've implemented several services:

UserService: Fetches user data from the backend, utilized in the User Page and Login Page and handles user authentication, used in the Login Page.

QuestionService: Manages fetching, filtering, and updating question data, employed in the Question Page, Add Question Page, and Edit Question Page.

AnswerService: Handles fetching and adding answers to questions, utilized in the Answer Page and Add Answer Page and facilitates the addition of new questions, used in the Add Question Page.

TagService: Manages fetching and adding tags to questions, employed in the Add Question Page.

For testing purposes, our frontend interacts seamlessly with the backend services provided by our project's backend, ensuring real-world functionality and integration.

FrontEnd & Backend  
The connection between the frontend and the backend was done using the services in the frontend and the controllers in the backend.

The features implemented are

feature1: users are able to ask questions, each having an author(the user), title, text, creation date & time (not visible in the frontend but are sorted by it) and one or mode tags. If the appropriate tag does not exist, the user can create one when adding a question or choose between the ones already available, also you can delete a question if it is yours.  
Question edits are not connected to the frontend  
A user can filter all the questions by the mentioned filters

feature2: each question can be answered one or more times by any connected user, the answer having an authors, text, image and creation date and time, moreover you can delete your own answer if you want.  
answers can be edited only in the backend  
for displaying the answers, the question is also available  
  
feature 3 was not implemented