StackOverlow

Roman Alisa-Dariana

May 21, 2023

Contents

1 Introduction

My project is a simplified version of StackOverflow, focusing on key features such as asking questions with tags, sorting questions by creation date, editing and deleting questions, filtering options, answering questions, voting on questions and answers, and sorting answers based on votes. Additional bonus features include a user scoring system based on votes received and moderator privileges for content management and user banning.

2 Technologies

For the development of my StackOverlow project, I utilized the following technologies:

MySQL: I chose MySQL as the relational database management system (RDBMS) to store and manage user data, questions, answers, votes, and other relevant information.

Java Spring: I used the Java Spring framework to develop the backend logic and functionality of the system. Spring provided me with a range of features for handling HTTP requests, routing, and data management, making it ideal for building a robust web application.

Angular: I opted for Angular to develop the frontend of my application. Angular is a popular frontend framework that offers a comprehensive set of tools and features for creating dynamic and responsive user interfaces.

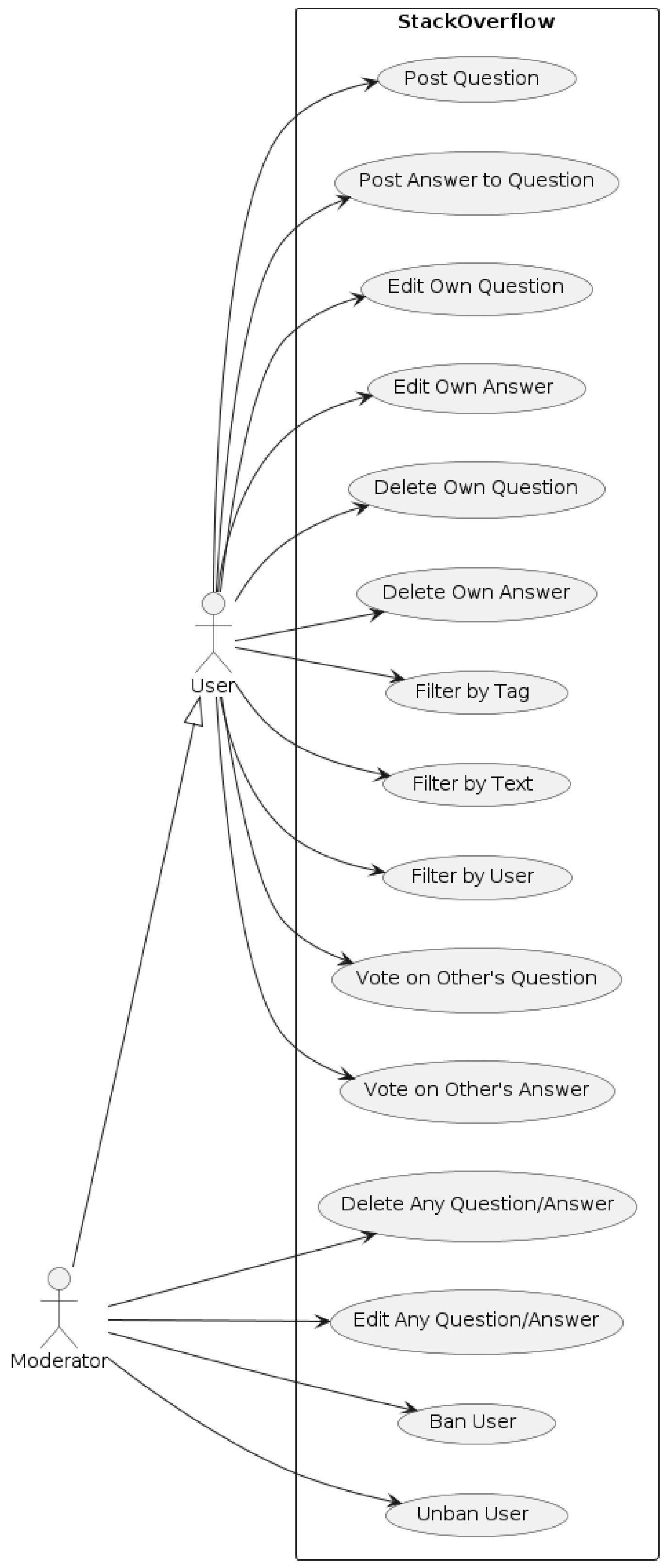
Git: I relied on Git as the version control system to track changes and manage my codebase. Git allowed me to create branches, merge code, and maintain a history of revisions during the development process.

IntelliJ Ultimate: I chose to work with IntelliJ IDEA Ultimate edition as my integrated development environment (IDE). With its powerful features, code assistance, and debugging capabilities, IntelliJ IDEA Ultimate made my development process smoother and more productive.

These technologies, including MySQL, Java Spring, Angular, Git, and IntelliJ Ultimate, were instrumental in building my StackOverlow project. They helped me manage the database effectively, develop the backend functionality, design the frontend, track changes with version control, and create a productive development environment.

3 Use Case Diagram

A user-case diagram is a formal visual representation of the user's interactions with the application. It depicts the the functionality of a system by illustrating the specific actions that users can perform.



4 Architecture

The layers that make up the architecture of my StackOverlow project are as follows:

Presentation Layer (Angular):

- Responsible for the user interface and user interactions.
- Implements Angular framework to build interactive and user-friendly web interfaces.
- Utilizes Angular components and templates to render views and handle user interactions.
- Implements route guards to control access to different routes based on user authentication and authorization.

Application Layer (Java Spring):

- Contains the business logic of your StackOverflow application.
- Implements Java Spring framework for building enterprise-grade applications.
- Handles user actions, validations, and coordination of data flow.
- Utilizes Spring components such as controllers and services to process user requests and orchestrate interactions between the presentation layer and the data layer.

Data Layer (MySQL and Spring Data JPA):

- Manages the persistence of data and communication with the MySQL database.
- Utilizes Spring Data JPA, an abstraction layer on top of JPA, to interact with the database.
- Implements data access objects (DAOs) or repositories to encapsulate database operations, such as retrieval, creation, updating, and deletion of data.
- Configures Hibernate as the JPA provider, which maps Java objects to database tables and manages the object-relational mapping.

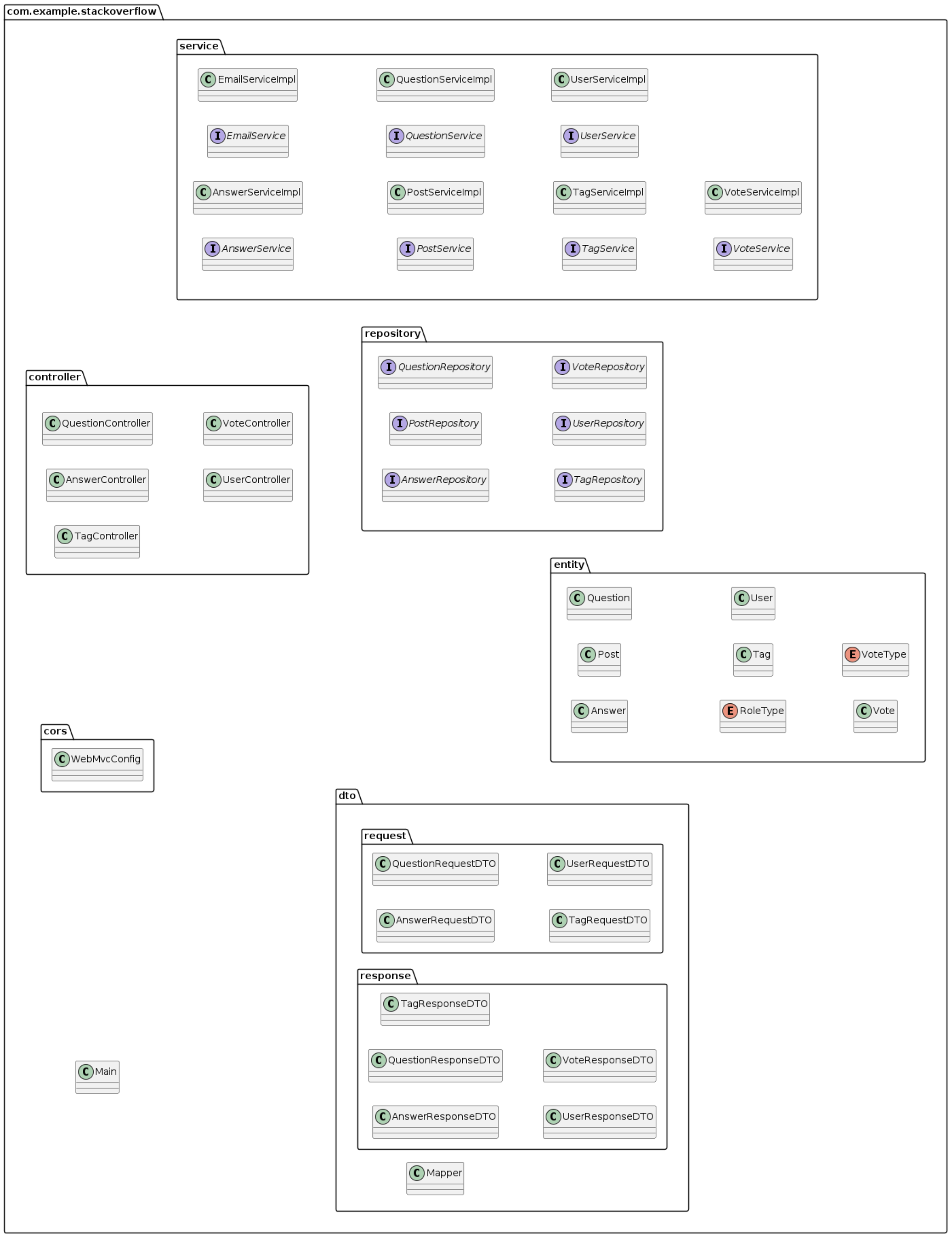
Security Layer (Angular Route Guards):

- Implements security measures within the presentation layer using Angular route guards.
- Controls access to specific routes based on user authentication and authorization.
- Ensures that only authenticated users can access certain parts of the application.
- Provides an additional layer of protection for sensitive data and enhances the overall security of the application.

5 Package Diagram

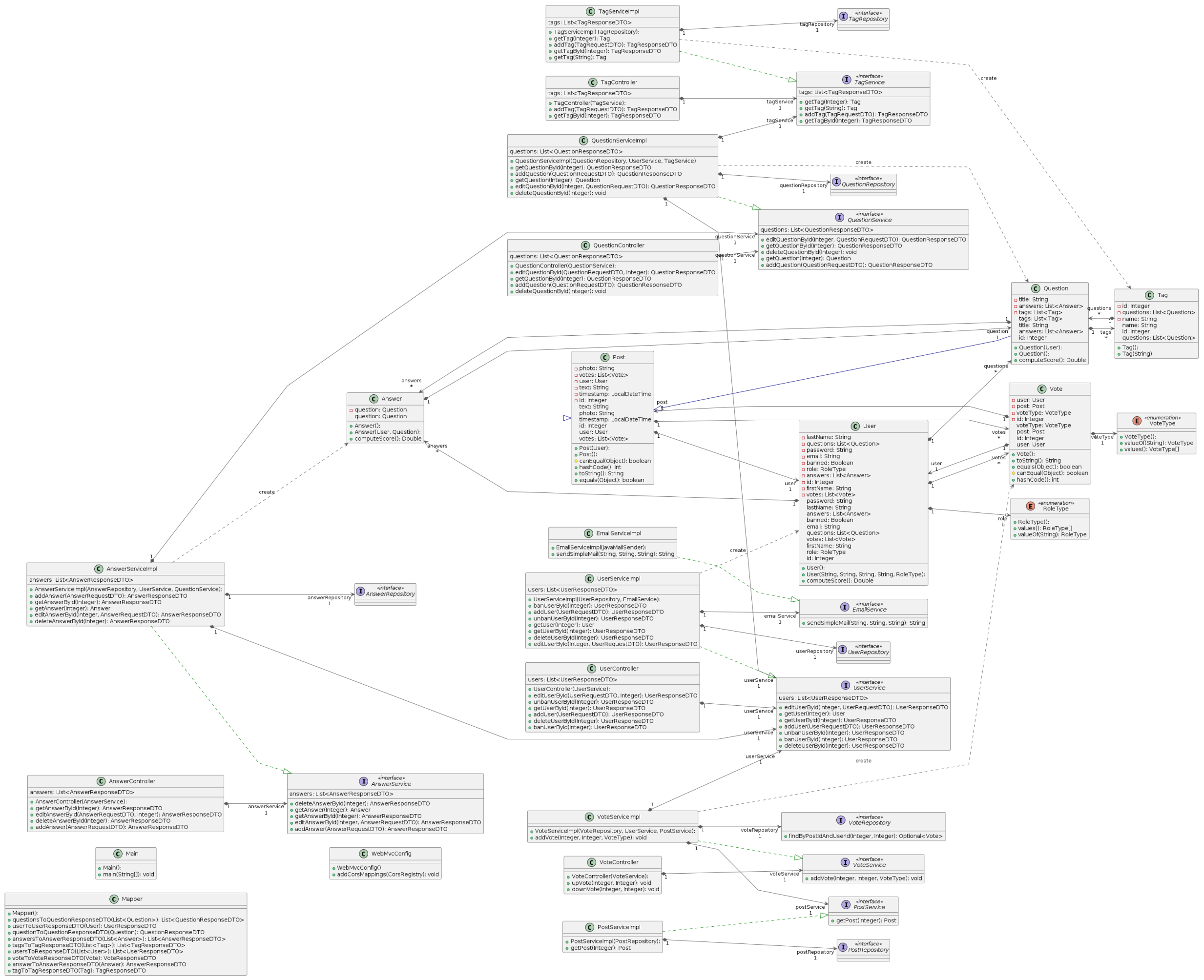
A package diagram is a formal visual representation of the system's structure. It depicts the organization of the system's components and their dependencies.

- com.example.stackoverflow: The root package of the application.
- controller: Contains the controllers responsible for handling incoming requests and coordinating the application's logic.
- cors: Contains the configuration for Cross-Origin Resource Sharing (CORS) to allow client-side requests from different origins.
- dto: Contains Data Transfer Objects (DTOs) used for exchanging data between the client and server.
- entity: Contains the entity classes that represent the application's domain model.
- repository: Contains interfaces defining the contracts for data access and manipulation.
- service: Contains interfaces and their corresponding implementations for business logic and application services.
- Main: Represents the main class of the application, which serves as the entry point for execution.



6 Class Diagram

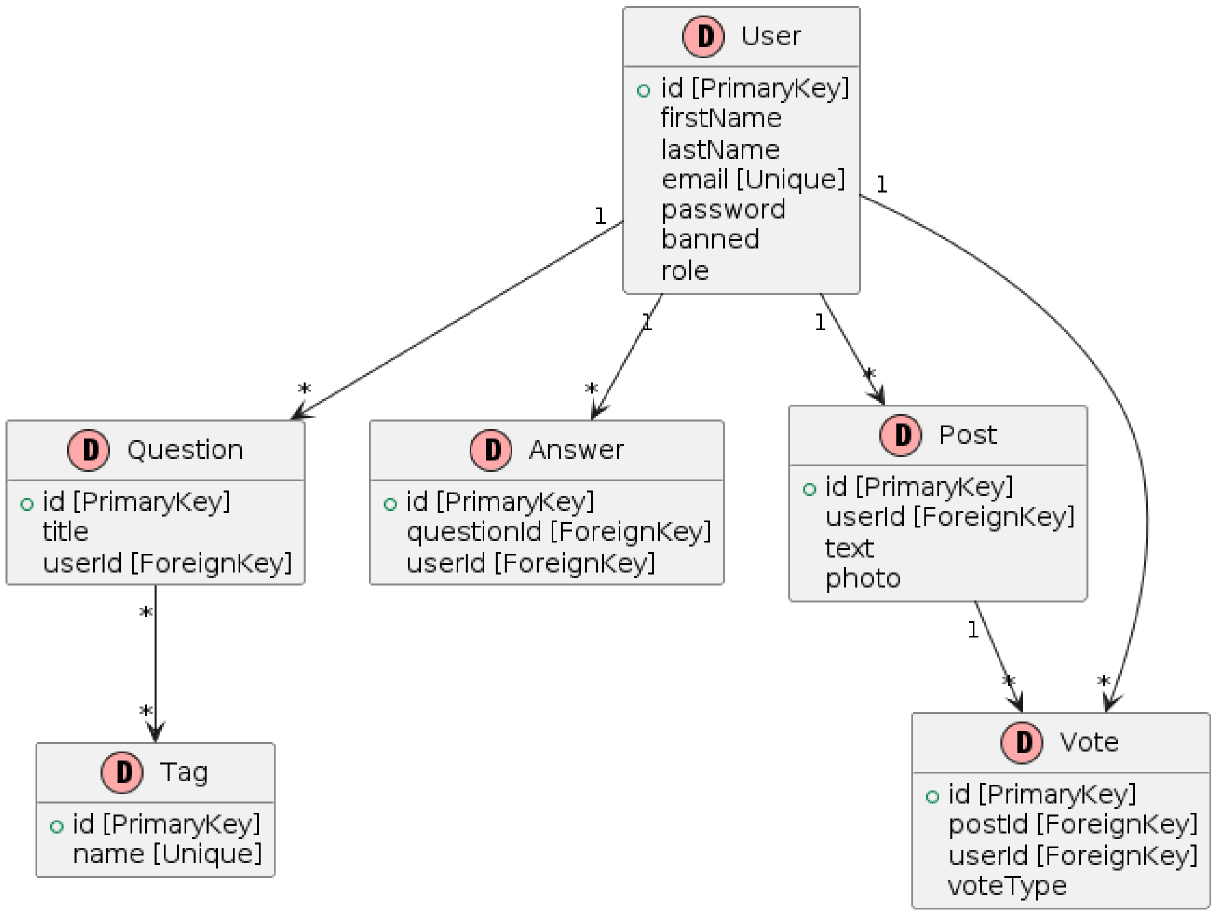
A class diagram is a formal visual representation of the system's structure. It depicts the classes, interfaces and their relationships within the system.



7 Database Diagram

A database diagram is a formal visual representation of the system's data structure. It depicts the tables, columns, and their relationships within the database.

- Post : Answer = One to one relationship
- Post : Question = One to one relationship
- User : Answer = One to many relationship
- User : Question = One to many relationship
- Post : Vote = One to many relationship
- User : Vote = One to many relationship
- Question : Tag = Many to many relationship



8 Endpoint Requests

Endpoint requests are a way to interact with a web application or API by sending specific HTTP requests to predefined URLs.

Endpoint: POST /answer/add

- Description: Add a new answer.
- Input: Request body containing an AnswerRequestDTO object that represents the details of the answer to be added.
- Output: Returns an AnswerResponseDTO object in the response body, which represents the newly added answer.

Endpoint: GET /answer/get

- Description: Get all answers.
- Input: None.
- Output: Returns a list of AnswerResponseDTO objects in the response body, representing all the answers available.

Endpoint: GET /answer/get/id

- Description: Get an answer by its ID.
- Input: Path parameter id specifying the ID of the answer to be retrieved.
- Output: Returns an AnswerResponseDTO object in the response body, representing the answer with the specified ID.

Endpoint: GET /answer/delete/id

- Description: Delete an answer by its ID.
- Input: Path parameter id specifying the ID of the answer to be deleted.
- Output: Returns an AnswerResponseDTO object in the response body, representing the deleted answer.

Endpoint: POST /answer/edit/id

- Description: Edit an answer by its ID.
- Input: Request body containing an AnswerRequestDTO object that represents the updated details of the answer, along with the path parameter id specifying the ID of the answer to be edited.
- Output: Returns an AnswerResponseDTO object in the response body, representing the edited answer.

9 Front-End Architecture

The project follows the typical Angular architecture, which is based on components, services, and modules.

Modules

- answers module: This module is responsible for managing the functionality related to answers. It includes components such as answer-add, answer-detail, and answer-list, which handle adding, viewing, and listing answers, respectively.
- authentication module: This module handles the authentication-related features of your application. It includes components like login for user login functionality and banned for handling banned users.
- author module: The author module likely represents a section related to authors or users. It includes components such as author.component that handle author-specific functionality.
- questions module: This module is responsible for managing the functionality related to questions. It includes components such as question-add, question-detail, and question-list, which handle adding, viewing, and listing questions, respectively.
- tags module: The tags module manages functionality related to tags. It includes components like tag-list that handle listing and displaying tags.
- users module: This module deals with user-related functionality. It includes components such as user-add, user-detail, and user-list, which handle adding, viewing, and listing users, respectively.
- vote module: The vote module is likely responsible for handling voting functionality within your application. It includes components like vote.component that handle the voting process.

Services

- answer.service.ts: This service is responsible for handling operations related to answers, such as adding answers, retrieving answers, editing answers, and deleting answers.
- authentication.service.ts: The authentication service handles user authentication-related functionality, such as user login, logout, and session management.
- question.service.ts: This service deals with operations related to questions, such as adding questions, retrieving questions, editing questions, and deleting questions.
- questions-tags.service.ts: The questions-tags service is likely responsible for handling operations related to the relationship between questions and tags. It might include functionalities like associating tags with questions, retrieving tags for a question, etc.
- tags.service.ts: This service manages operations related to tags, such as retrieving tags, adding tags, and updating tags.
- user.service.ts: The user service handles user-related operations, including adding users, retrieving user details, updating user information, etc.

- vote.service.ts: This service is responsible for managing voting functionality. It likely includes operations for voting on answers or questions, retrieving vote counts, and updating vote values.

Components

- **Authentication Module:**
  - `LoginComponent`: Responsible for user login functionality.
  - `BannedComponent`: Renders a message or page for banned users.
- **Author Module:**
  - `AuthorComponent`: Represents the author component responsible for author-related functionality.
- **Questions Module:**
  - `QuestionAddComponent`: Handles the addition of new questions.
  - `QuestionDetailComponent`: Displays detailed information about a specific question.
  - `QuestionListComponent`: Renders a list of questions.
- **Answers Module:**
  - `AnswerAddComponent`: Allows users to add new answers.
  - `AnswerDetailComponent`: Displays detailed information about a specific answer.
  - `AnswerListComponent`: Renders a list of answers.
- **Tags Module:**
  - `TagListComponent`: Renders a list of tags.
- **Users Module:**
  - `UserAddComponent`: Handles the addition of new users.
  - `UserDetailComponent`: Displays detailed information about a specific user.
  - `UserListComponent`: Renders a list of users.
- **Vote Module:**
  - `VoteComponent`: Represents the voting component responsible for handling votes.

10 Front-End Routing

AppModule routing

- **/users** - Loads the UsersModule, which has its own set of child routes. It requires authentication and bans guard.
- **/questions** - Loads the QuestionsModule, which has its own set of child routes. It requires authentication and bans guard.
- **/answers** - Loads the AnswersModule, which has its own set of child routes. It requires authentication and bans guard.
- **/login** - Loads the AuthenticationModule, which handles the login functionality.
- **/**** - For any other path, it redirects to **/questions**.

UsersModule routing

- **/add** - Loads the UserAddComponent, which allows the user to add a new user.

- **/:id** - Loads the UserDetailComponent with the user ID as a parameter.
- **/** - Loads the UserListComponent, which displays a list of all users.

QuestionsModule routing

- **/add** - Loads the QuestionAddComponent, which allows the user to add a new question.
- **/:id** - Loads the QuestionDetailComponent, which displays the details of a specific question based on the ID parameter in the URL.
- **/** - Loads the QuestionListComponent, which displays a list of all questions.

AnswersModule routing

- **/add/:qid** - Loads the AnswerAddComponent, which allows the user to add a new answer to the question with the ID specified by the "qid" parameter in the URL.
- **/:id** - Loads the AnswerDetailComponent, which displays the details of a specific answer based on the ID parameter in the URL.

AuthenticationModule routing

- **/banned** - Loads the BannedComponent, which displays a message to the banned users.
- **/** - Loads the LoginComponent, which handles the login functionality.