Technical University of Cluj-Napoca

Faculty of Automation and Computer Science

2nd Semester 2022-2023

Software Design

- Laboratory Assignment-

StackOverflow Application

Student: Maria Gliga-Hambeț

Group: 30433

Supervising teacher: Bogdan Bindea

1. Introduction

This documentation provides a detailed overview of the design and implementation of a StackOverflow-inspired application developed using Angular and Spring. The aim of this project is to create a robust platform that facilitates the exchange of knowledge and fosters a collaborative environment for users.

The application is designed to cater to two types of users and ensures that all actions are accessible only to authenticated users. User passwords are stored securely in an encrypted format within the database to ensure data privacy. Features of the application:

- Feature 1: Asking Questions

One of the key features of the application is the ability for users to ask questions. Each question consists of essential details such as an author, title, text, creation date and time, and optional picture. Additionally, questions can be tagged with relevant categories, and users can create new tags if necessary. The list of questions is displayed in chronological order, with the most recent questions appearing first. Users have the capability to edit or delete their own questions. The application also provides filtering options based on tags, text search, users, and personal questions.

- Feature 2: Answering Questions

Users can provide answers to questions posted by others, including the original question author. Each answer includes an author, text, picture and creation date and time. Similar to questions, answers can be edited or deleted by their respective authors. When viewing an individual question, the associated answers are displayed, sorted by their vote count.

- Feature 3: Voting on Questions and Answers

To facilitate community participation, users have the ability to vote on both questions and answers. Votes can be either upvotes or downvotes, equivalent to likes and dislikes. Each user is limited to voting once on each question or answer and cannot vote on their own content. The vote count, calculated as the difference between upvotes/likes and downvotes/dislikes, is displayed alongside each question or answer. The system accommodates negative vote counts, if applicable. Answers are sorted based on their vote counts, with the highest-rated answers appearing first.

- Feature 4: User Scoring

An additional feature of the application is the computation of a user score based on the accumulated votes on their content. Users start with a score of 0 and gain points when their questions or answers receive upvotes. Conversely, points are deducted if their content receives downvotes or if they downvote answers provided by other users. The user score is displayed alongside the author's name for each question and answer. It is important to note that the score can be negative.

- Feature 5: Moderator Privileges

The application incorporates moderator privileges to ensure the maintenance of a healthy and respectful community. Moderators have the authority to remove inappropriate questions or answers, edit any content on the site, and indefinitely ban users in cases of misconduct. Banned users receive a message indicating their ban status upon login and are prohibited from accessing the application via the URL. They also receive an email and SMS notification regarding their ban. Moderators can lift the ban for previously banned users.

2. Technology Stack

The application is built on a powerful technology stack to provide a smooth experience. The technologies used were chosen whilst having in mind the scalability, maintainability and security of the application.

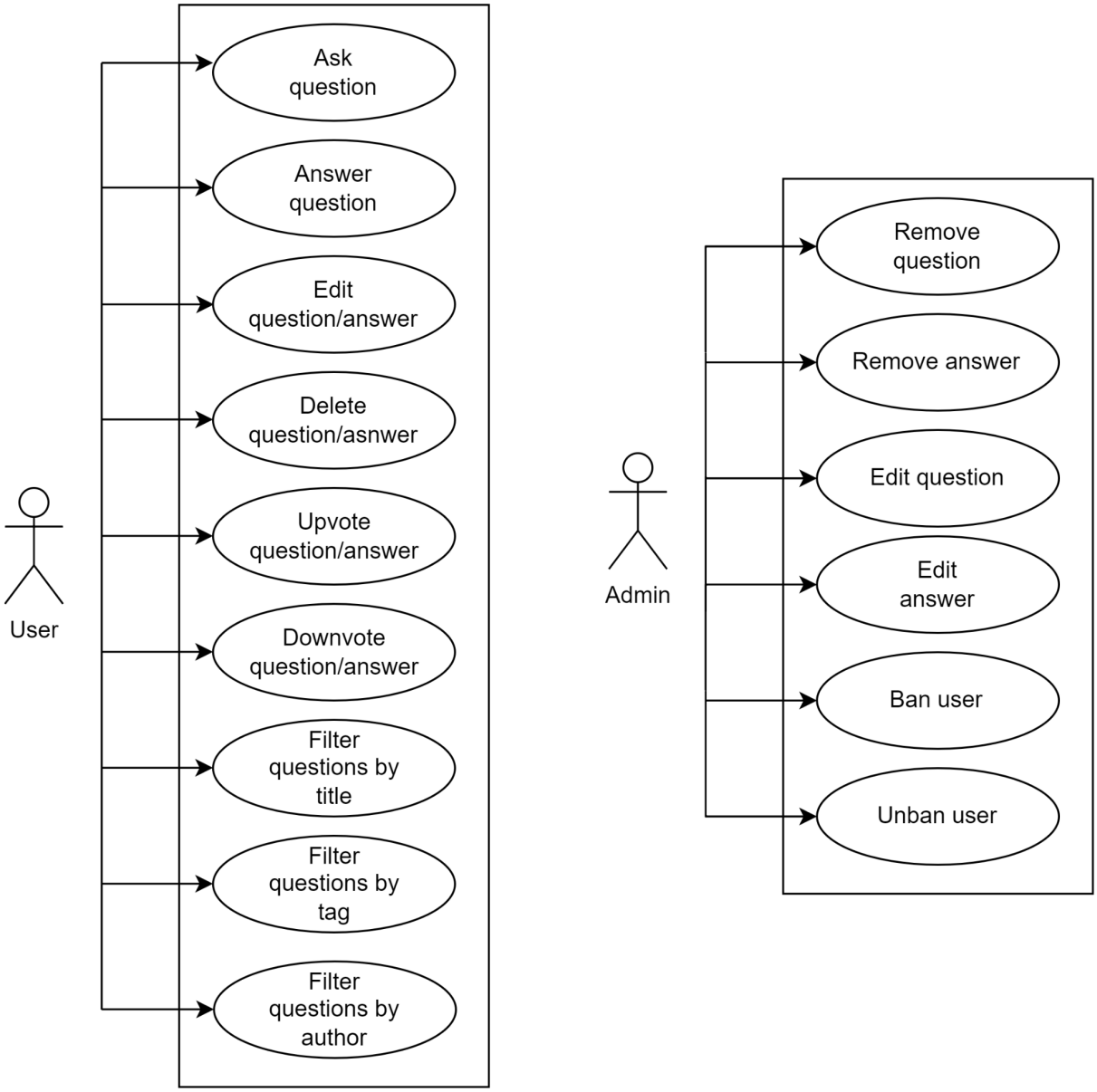
The frontend is developed using Angular, a renowned JavaScript framework. Angular's component-based architecture enables me to manage the application efficiently, even as its complexity grows.

For the backend I have chosen Java Spring, a comprehensive framework for enterprise Java. It provides a range of features that streamline the development process, from dependency injection to transaction management.

In terms of database management, I utilise MySQL, a reliable and robust open-source relational database management system. MySQL's wide range of operations, including querying and indexing, proves essential for the smooth operation.

3. Use-Case Diagram:

A use case diagram is a visual representation that illustrates the interactions between actors (users, external systems, or other entities) and a system. It provides an overview of the system's functionalities from a user's perspective, capturing the various use cases or actions that actors can perform and the relationships between them.



4. Software Architecture:

The application employs a structured and layered software architecture to effectively organise the application into distinct components based on their responsibilities. This approach ensures a clear separation of concerns, enhancing the development process, system management, and scalability of the application.

At the foundation of the architecture lies the Database Layer, where MySQL, a reliable relational database system, is utilised for storing and retrieving all application data. This layer plays a critical role in managing various types of data, including artwork details, exhibition information, user profiles, and bookings.

Above the Database Layer is the Persistence Layer, which acts as an intermediary between the database and the business layer. This layer primarily consists of Data Access Objects (DAOs) and Repository interfaces. Its purpose is to abstract the underlying database operations and provide a more object-oriented representation of the data.

The Business Layer serves as the core of the application, encompassing the business logic necessary for processing data and enforcing business rules. Within ArtisticAvenues, this layer encompasses operations such as managing bookings, handling user access, and facilitating guided tours.

Lastly, the Presentation Layer is responsible for the application's user interface (UI) that users interact with. Developed using Angular, this layer comprises UI components that handle user inputs and outputs, as well as displaying data to users. It manages the visual representation of information and facilitates user interactions, such as button clicks and form submissions.

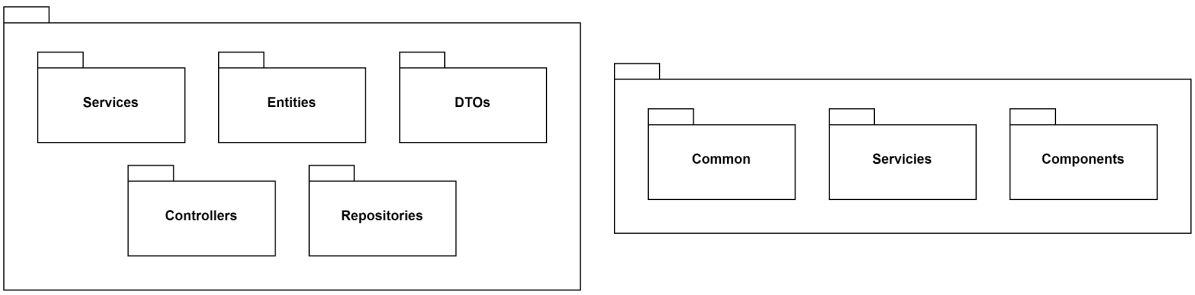
5. Package Diagram:

A package diagram is a type of UML diagram that provides an overview of the organisation and structure of a software system or application. It depicts the various packages, modules, or namespaces within the system and shows the relationships and dependencies between them.

The application is organised into several packages:

- Controllers: This package contains the controllers responsible for handling incoming requests and directing them to the appropriate business logic. The controllers interact with the UI components in the Presentation Layer and delegate the processing to the corresponding services.
- Services: The services package encapsulates the business logic and acts as a bridge between the controllers and the repositories. It implements the necessary operations for handling user actions, such as asking questions, answering queries, and managing voting.
- Repositories: This package encompasses the repository interfaces provided by Spring Data JPA. These interfaces define the methods for data access and manipulation, allowing seamless interaction with the underlying database. The repositories facilitate efficient and consistent data retrieval and storage.
- Entities: The entities package contains the data model classes that represent the application's domain entities. These classes define the structure and relationships of the data stored in the database, including user details, questions, answers, and voting information.

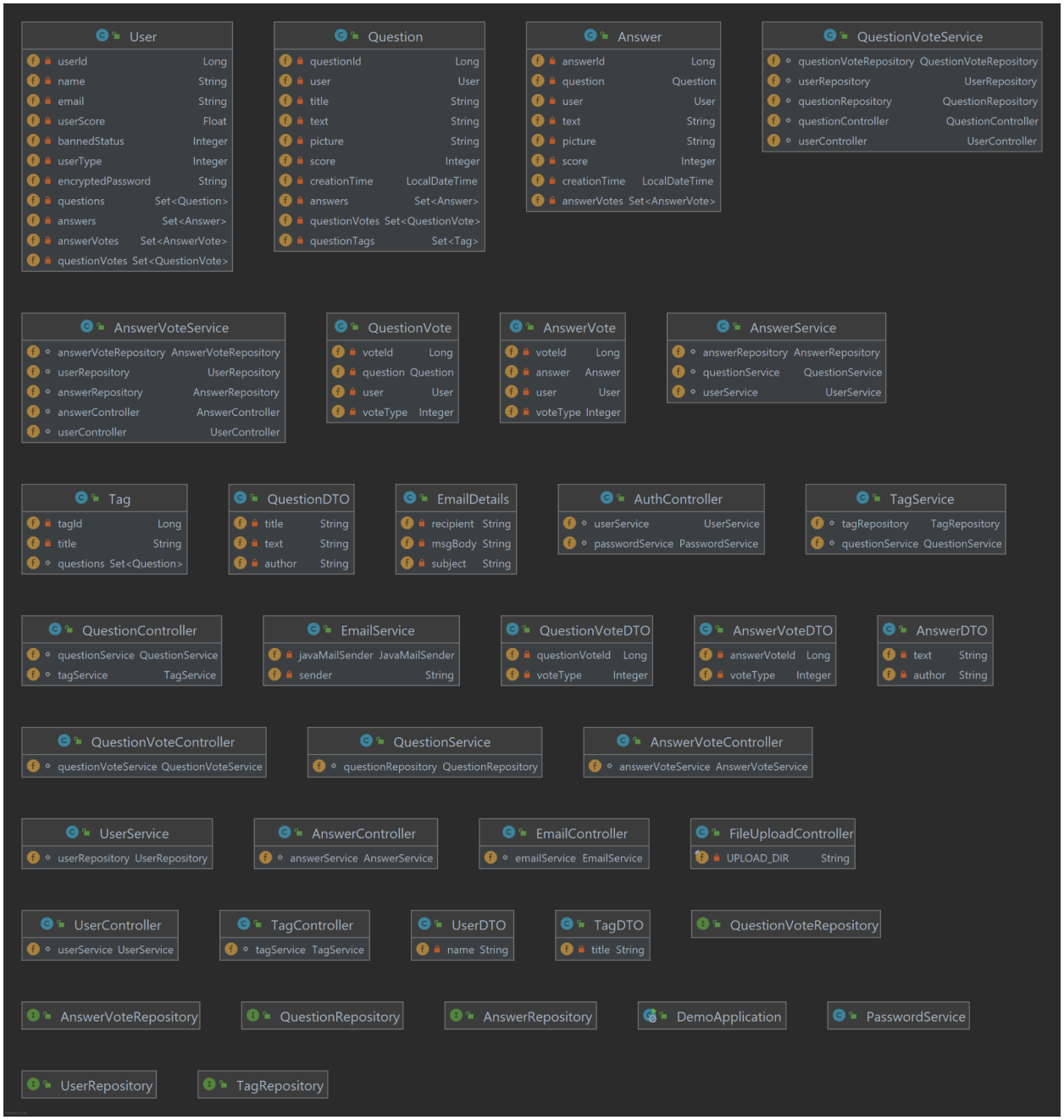
- DTOs (Data Transfer Objects): The DTOs package contains the Data Transfer Objects used for data exchange between different layers of the application. DTOs serve as standardised data structures that encapsulate specific data required for communication between the backend and frontend or between different services within the application.



6. Class Diagram:

A class diagram is a type of UML (Unified Modeling Language) diagram that provides a visual representation of the structure and relationships of classes within a system. It depicts the classes, their attributes, methods, and the associations between them.

Class diagrams are widely used to model the static structure of object-oriented systems and are instrumental in understanding the overall system architecture and design. They provide a visual overview of the classes, their properties, and the relationships between them, helping to communicate the system's structure and behaviour.

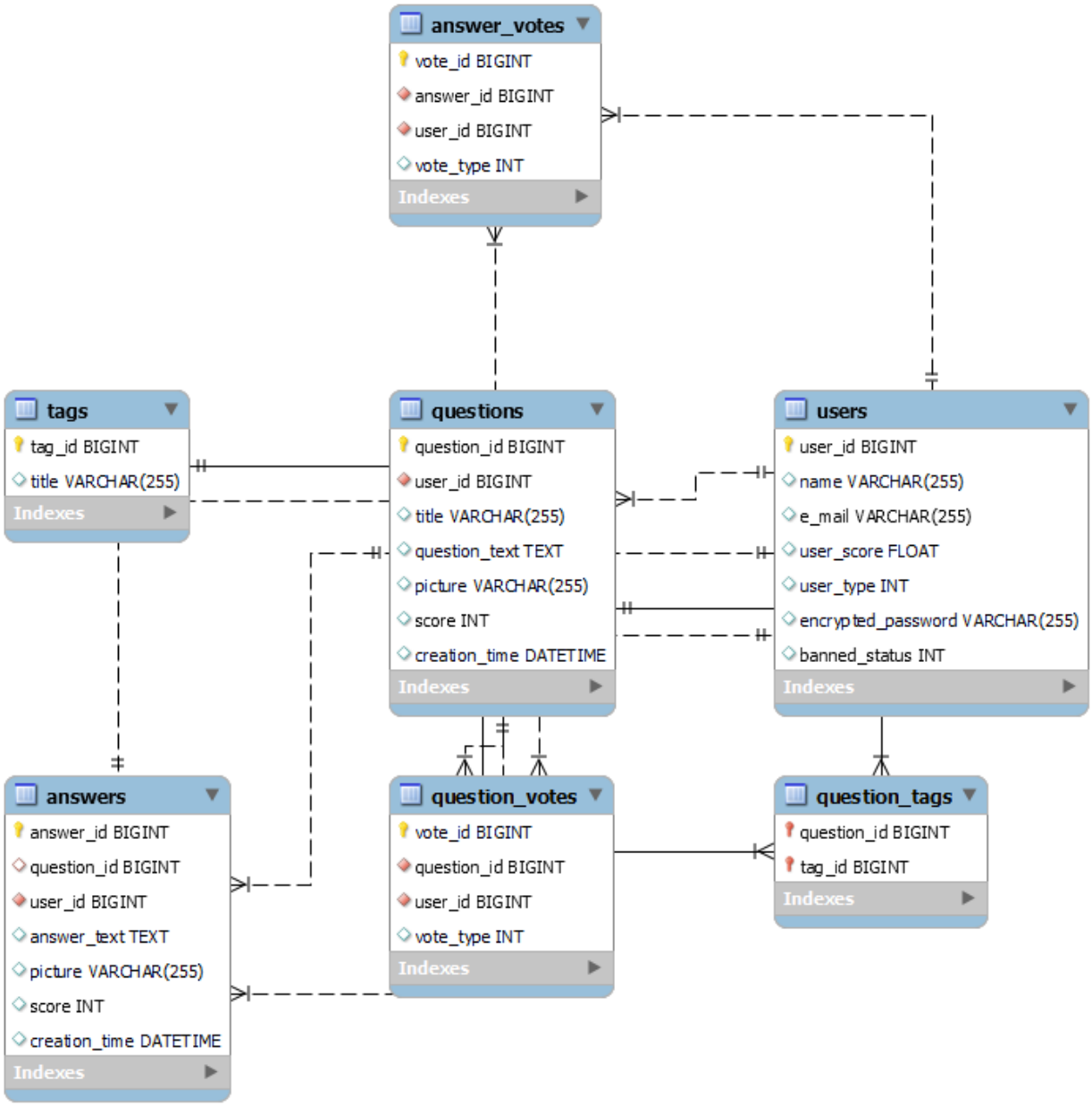


7. Database Diagram:

A database diagram, also known as an entity-relationship (ER) diagram, is a visual representation of the structure and relationships of a database system. It illustrates the tables, their attributes, and the relationships between them, providing a high-level overview of the database schema.

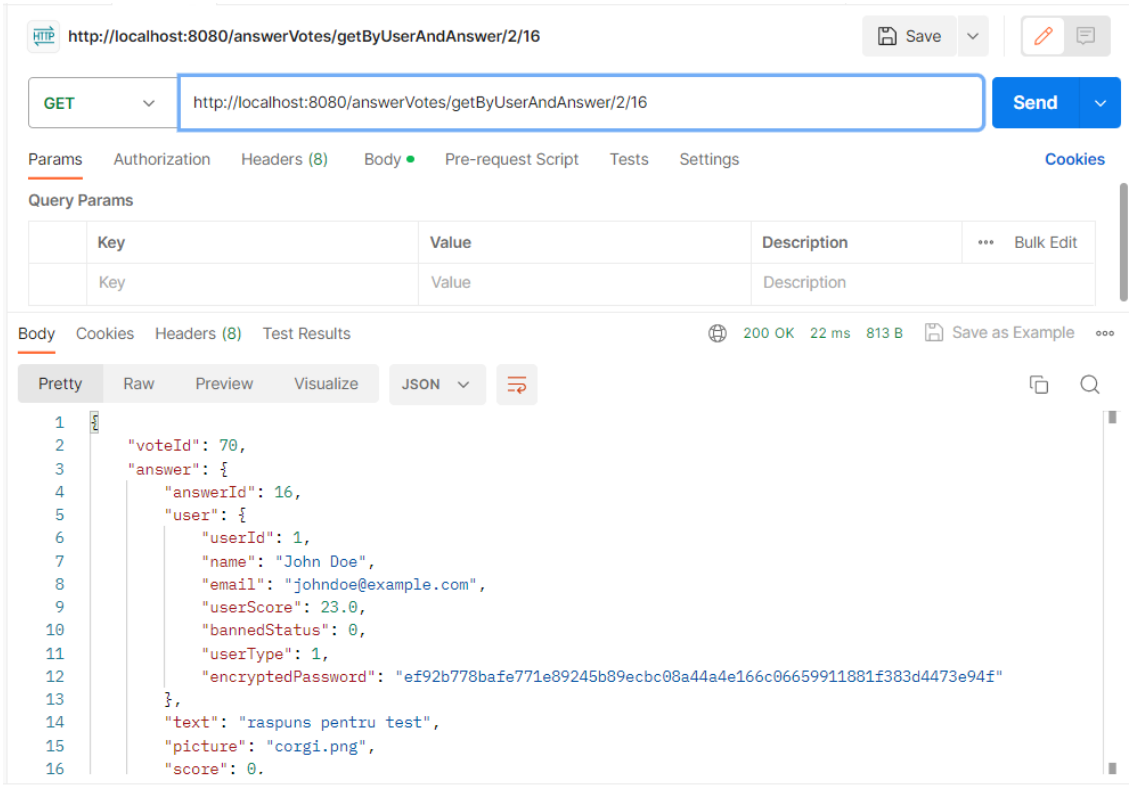
Database diagrams are instrumental in understanding the logical structure of a database system and help in visualising the relationships and dependencies between tables. They provide a visual

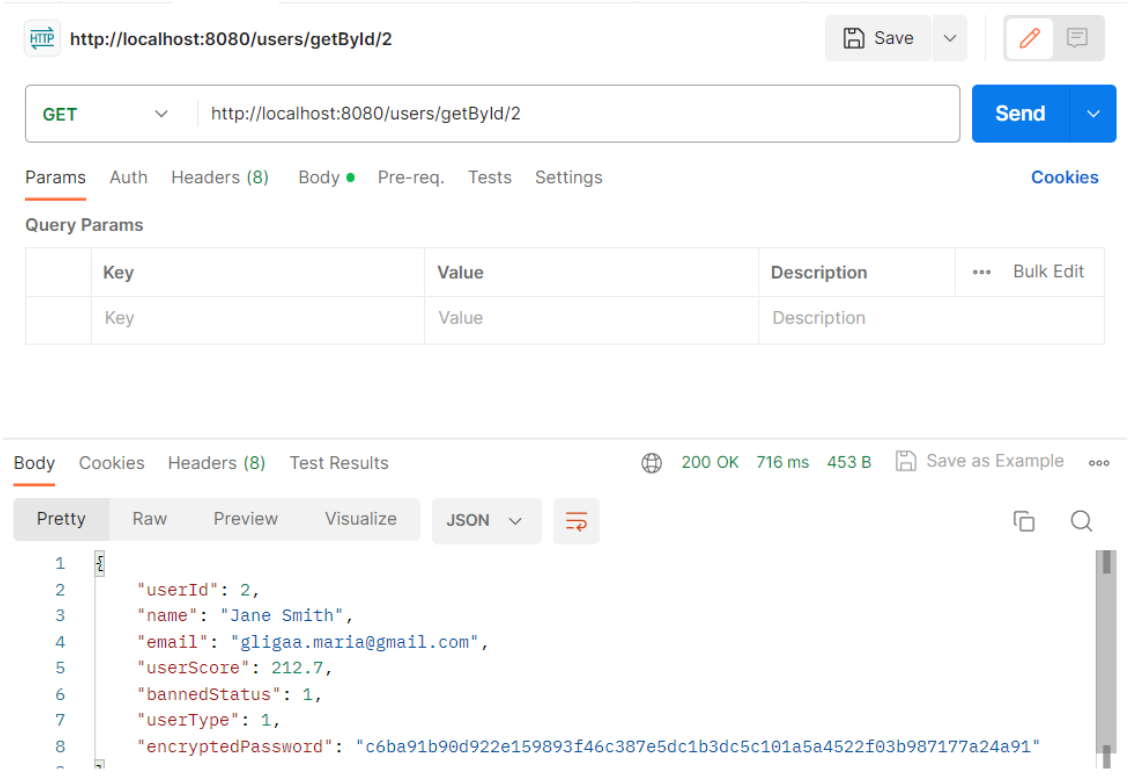
representation of the database schema, aiding in communication, documentation, and database design.



8. Endpoint requests:

Endpoint requests, also known as API requests or HTTP requests, are the means by which clients interact with a web server or an application programming interface (API) to request or manipulate data. These requests are typically made using the HTTP protocol and allow clients to communicate with a server and perform specific actions or retrieve information.


http://localhost:8080/answerVotes/getByUserAndAnswer/2/16
Save
GET
http://localhost:8080/answerVotes/getByUserAndAnswer/2/16
Send
Params
Authorization
Headers (8)
Body
Pre-request Script
Tests
Settings
Cookies
Query Params
Key
Value
Description
Bulk Edit
Key
Value
Description
Body
Cookies
Headers (8)
Test Results
200 OK
22 ms
813 B
Save as Example
Pretty
Raw
Preview
Visualize
JSON
{
"voteId": 70,
"answer": {
"answerId": 16,
"user": {
"userId": 1,
"name": "John Doe",
"email": "johndoe@example.com",
"userScore": 23.0,
"bannedStatus": 0,
"userType": 1,
"encryptedPassword": "ef92b778bafe771e89245b89ecbc08a44a4e166c06659911881f383d4473e94f"
},
"text": "raspuns pentru test",
"picture": "corgi.png",
"score": 0.


http://localhost:8080/users/getById/2
Save
GET
http://localhost:8080/users/getById/2
Send
Params
Auth
Headers (8)
Body
Pre-req.
Tests
Settings
Cookies
Query Params
Key
Value
Description
Bulk Edit
Key
Value
Description
Body
Cookies
Headers (8)
Test Results
200 OK
716 ms
453 B
Save as Example
Pretty
Raw
Preview
Visualize
JSON
{
"userId": 2,
"name": "Jane Smith",
"email": "gligaa.maria@gmail.com",
"userScore": 212.7,
"bannedStatus": 1,
"userType": 1,
"encryptedPassword": "c6ba91b90d922e159893f46c387e5dc1b3dc5c101a5a4522f03b987177a24a91"

http://localhost:8080/answers/getByQuestionId/8

GET http://localhost:8080/answers/getByQuestionId/8

Params Authorization Headers (8) Body Pre-request Script Tests Settings

Query Params

	Key	Value
	Key	Value

Body Cookies Headers (8) Test Results

Pretty Raw Preview Visualize JSON


```
[
    {
        "answerId": 22,
        "user": {
            "userId": 1,
            "name": "John Doe",
            "email": "johndoe@example.com",
            "userScore": 23.0,
            "bannedStatus": 0,
            "userType": 1,
            "encryptedPassword": "ef92b778bafe771e89245b89ecbc08a44a4e166c06659911881f383d4473e
        },
        "text": "raspund si aici",
        "picture": "corgi.png",
        "score": 0,
        "creationTime": "2023-05-28T14:41:08"
    },
    {
        "answerId": 16,
        "user": {
            "userId": 1,
            "name": "John Doe",
```


e Find and replace Console

http://localhost:8080/answerVotes/deleteById/70

Save

DELETE http://localhost:8080/answerVotes/deleteById/70 Send

Params Authorization Headers (8) Body Pre-request Script Tests Settings Cookies

Query Params

Key	Value	Description	Bulk Edit
Key	Value	Description	

Body Cookies Headers (8) Test Results 200 OK 52 ms 259 B Save as Example

Pretty Raw Preview Visualize Text

```
1 Success
```

http://localhost:8080/answerVotes/insertVote

Save

POST http://localhost:8080/answerVotes/insertVote Send

Params Authorization Headers (8) Body Pre-request Script Tests Settings Cookies

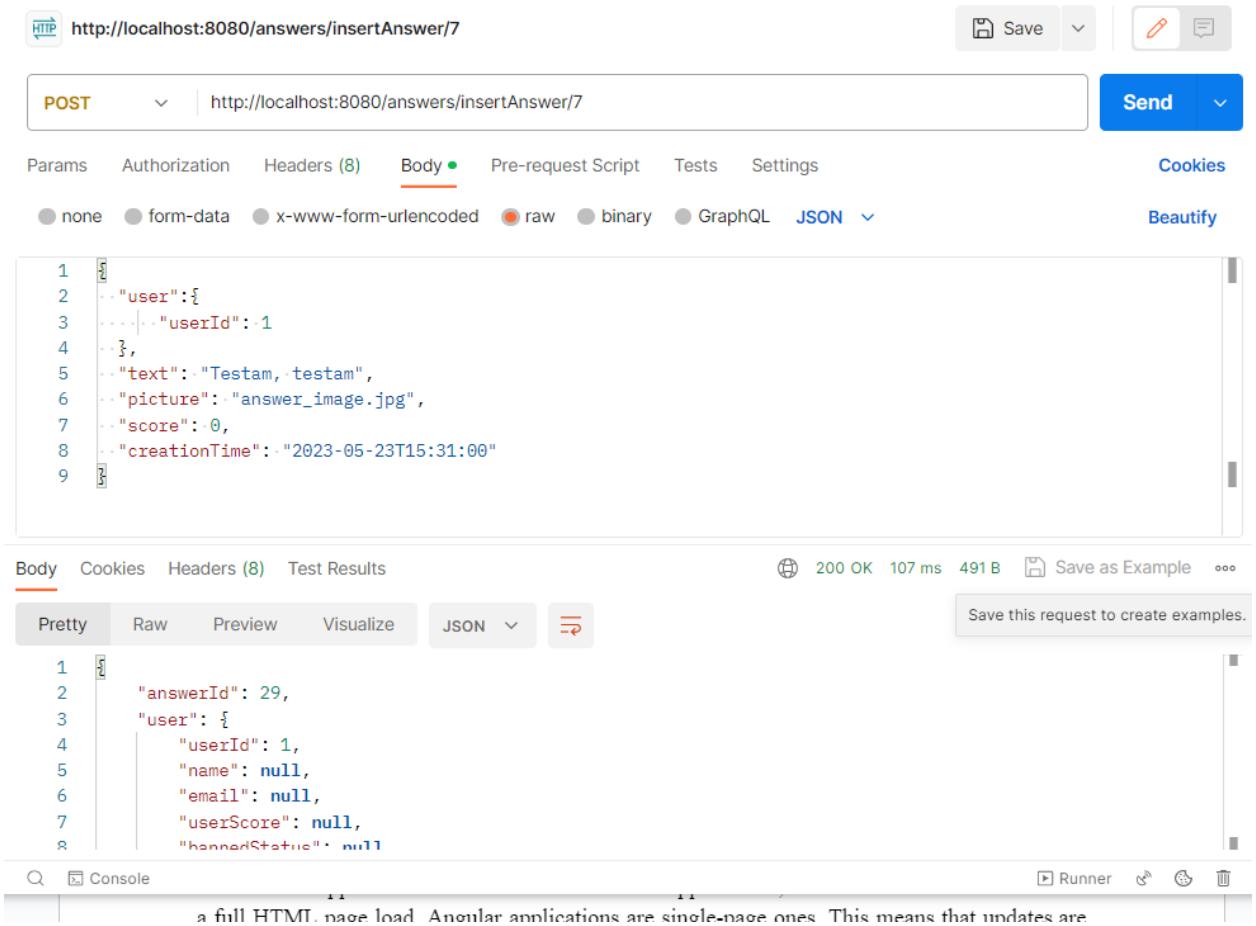
none form-data x-www-form-urlencoded raw binary GraphQL JSON Beautify

```
{
    "answer":{
        "answerId":28
    },
    "user":{
        "userId":2
    },
    "voteType":1
}
```

Body Cookies Headers (8) Test Results 200 OK 106 ms 809 B Save as Example

Pretty Raw Preview Visualize JSON

```
{
    "voteId": 74,
    "answer": {
        "answerId": 28,
        "user": {
            "userId": 1,
            "name": "John Doe",
            "email": "johndoe@example.com",
            "userScore": 28.0,
```



Front-end development is a vital aspect of web application development that focuses on user interfaces and user experiences. This section will discuss key concepts in front-end development, particularly focusing on the architecture of Angular applications.

Angular is a popular front-end framework that makes it easier to build high-quality, maintainable and scalable applications. In contrast to traditional applications, where each user action results in a full HTML page load, Angular applications are single-page ones. This means that updates are partial, rather than full page ones. The framework allows developers to make use of both static and dynamic elements, using a combination of HTML, CSS and TypeScript.

Angular is a component-based framework. In this context, a component is a core element that encapsulates the logic, view and styling of a specific part of a web application. It is a self-contained and reusable unit that represents a part of the user interface and handles the associated behaviour. Their modularity promotes clean and maintainable code. Each component consists of:

- An HTML template that defines what will be rendered and displayed to the user;
- A TypeScript class which contains the behaviour of the component. It defines methods that are used to handle user interactions and respond to events. The component class holds information which helps with the interaction with the rest of the application. With the help of the selector keyword, we can invoke the said component and its properties;
- A CSS selector which holds information about the elements displayed and their style.

Angular applications are organised into modules, which are logical groupings of related components, directives and services. Modules encapsulate related code, allowing for better organisation and separation of concerns within an application. By encapsulating components, services, and other related entities, modules help to keep the codebase modular, maintainable, and scalable.

Services are classes that encapsulate related logic and can be shared across multiple components. They are used to share data, perform operations, and interact with external resources, such as APIs or databases. Dependency injection is used to inject services into components or other services. This ensures that the necessary dependencies are provided when needed, making it easier to manage and test the application's dependencies.

Directives are also a powerful feature in Angular which allows for the extension of the HTML syntax and addition of custom behaviour to elements or components. Their purpose is to manoeuvre the DOM (Document Object Model), be it by adding or removing elements and even changing their appearance.
Data binding establishes a connection between the data in the component and the user interface (UI) elements, ensuring that changes in one automatically reflect in the other. It enables the developer to dynamically update and synchronise data between the two, providing a seamless and interactive user experience.

Since Angular is a single-page framework where content is dynamically loaded and updated, there is the need for the multiple views and pages to interact. This is done with the help of routing. Routes are used to define navigation paths within an application and determine which components should be displayed for each specific route. The Router is the main routing service. It enables navigation between views based on user actions. RouterLink is a link to specific routes in the application. In the application's main template file, the router outlet acts as a placeholder where the corresponding component for the current route will be rendered.

Here is an examples of routes, based on the application that I have developed:

```
const routes: Routes= [
 {path: 'login', component: LoginComponent},
 {path: 'questions/:id', component: QuestionDetailsComponent},
 {path: 'questions/user/:user', component: QuestionListComponent},
 {path: 'search/:keyword', component: QuestionListComponent},
 {path: 'questions', component: QuestionListComponent},
 {path: 'new/question', component: NewQuestionComponent},
 {path: 'tags', component: TagListComponent},
 {path: '', redirectTo: '/questions', pathMatch: 'full'},
 {path: '**', redirectTo: '/questions', pathMatch: 'full'}
]
```

As it can be seen, based on what path is pattern matched, there is a component which should be accessed. Another useful feature is the use of parameters (for example id, user and keyword), in order to also get specific behaviours in the component. The last two paths are used for redirecting back to the main page in case there is nothing added or nothing is matched.

Depending on the route which led to a given component, elements can be displayed or not, by checking the origin of the path. This is particularly useful for managing elements which are placed outside of the router outlet.

In my application, I have some elements which are constantly displayed (the navigation bar and the sidebar), as they are significant in all the pages. The main content is displayed with the help of the router outlet. The only view which is not concerted with the two elements mentioned above is the login component. For this scenario, I have used the technique previously described of checking the path which led to the component and showing the navigation bar and sidebar or not.

The sidebar helps with the filtering of questions. If the “Tag” option is selected, the main content that will be displayed is a list of all the existing tags. By clicking either of them, the user is led to the questions which contain that tag. The “User” option refers to filtering questions based on their author, whilst “My question” returns the questions asked by the user that is logged in.

The navigation bar contains a search field. After entering a word and pressing the “Search” button or the “Enter” key, the user is presented with the questions which contain that word in their title.