A Deliverable 1 Documentation on

Stackoverflow web application

Software Design

Computer Science

2022-2023

Submitted by:

Jakab-Gyik Sarolta

Technical University of Cluj-Napoca

Romania

Table of contents

Abstract

This web application aims to be a simpler version of the well-known application ‘Stakoverflow’. This app is widely used by programmers all around the world to ask questions related to programming or the computer science domain in general, and these questions are answered by other programmers who use the application. Every user has an account after registering and after that he can post his own questions or answer to the questions posted by others. This is the general idea of the app, however, this porotype later could be extended with many other features.

1. Introduction

1.1 Background

This project is my first web application that tries to follow the idea behind ‘Stackoverflow’. Hence the name. For this first deliverable the back-end part of the application will be presented.

1.2 Key features

The application will provide 2 types of users: the regular user and the administrator. The administrator is not yet implemented. Therefore, all the features presented below are features that any user can take advantage of.

The login system is the first window that a user would see when they search for the website. Any other endpoint redirects them to the login page if they have not logged in yet. There, the user must provide their username and password to proceed further. If the user is not registered, they can choose to register by clicking on a button. If the user has already logged in, then the system will store that information in a cookie and no login is required.

For registration the user must provide their first and last name starting with capital letters, a username that is not yet taken by other users, a password, an email address and his phone number. All of these inputs get validated. After that, the user can log in.

When logged in, the user can read the questions posted by themselves or other users. The latest questions are posted first. The user can react to the questions of others by clicking on like or dislike buttons.

The user cannot react to their own questions but can update them or delete them. The user can post new questions as well by filling out a form with a question title, some content and by adding tags if they prefer to. Multiple tags can be added to any question.

If the user clicks on the ‘View’ button, he can view all the comments posted for that question. The comments appear by vote count in descending order. The answer that proved to be the most useful (with the highest vote count) will be the first on the list and so on. The user can also add their own comments regardless of the author of the question (it’s their own question or somebody else’s). They can also react to the comments posted by others with a like or a dislike, thus contributing to the vote count.

The user can filter questions based on different criteria. Any user can filter their own questions, questions based on a keyword that they type in. They can filter based on tags that they type in or by username, providing an existing username in the system.

In case of switching users, any user can log out and they will be redirected to the login page.

2.Technology

2.1 Spring Framework

The Spring Framework is an open-source Java-based application development framework that helps programmers building light-weight web application. I chose this framework because it supports object-oriented programming, and it provides front-end and back-end technologies in one place. It provides a lot of features like dependency injection, data access, and transaction management. Spring integrates with other popular Java frameworks and technologies like Hibernate, also used in this project. It has a modular and extensible architecture that allows for maintainability, scalability and flexibility. In the future, if this application were to publish on the market, the underlying Spring technologies would enable the developers to easily adapt to the clients' needs.

2.2 Spring Data JPA

The Spring Data JPA is a module of the Spring Framework that provides a simplified way for working with Java Persistence API in the data access layer. It comes with a set of generic interfaces and implementations that can be extended and customized to fit the programmer's needs. For the CRUD operations (create, read, update, delete) the programmer doesn't have to write all the code just use a generic JPARepository, like in this project. This repository enables these CRUD operations to be generated automatically just by specifying the method's signature. It creates DAOs (data access objects) automatically at compile time. It uses ORMs (object related mappings) in these DAOs. This allows developers to focus on writing business logic rather than code for data access.

2.3 Spring Test Framework

This module of the Spring Framework provides annotations and many utilities for testing in a Spring-based application. It offers support for testing different Spring components, such as controllers (in the example below at 8.Testing), services, repositories and others. It allows integrations tests, unit tests, end-to-end tests and tests on different levels of the application stack. The Spring Test Framework provides powerful testing features, such as dependency injection for the test classes, transaction management, and web testing support. It enables for reliable tests and robust code.

I have used JUnit as an open-source testing framework for my application. It can be easily integrated with Gradle and provides simple ways to write and run different type of tests. I wrote some unit tests to verify individual methods and classes, such as the Controller classes and through them the Service classes and their methods and the DAOs and the Mappers. JUnit is a standard tool for testing application is Java, in general.

Next to JUnit, I have used the Mockito open-source Java testing framework that provides tools to create mock objects for testing. It allows developers to simulate the behavior of dependencies without having to set up a real instance of those dependencies. With Mockito, it is easy to isolate parts of the code under test and focus on specific scenarios.

For the dependencies I have used the Spring Boot Starter Test module of the Spring Boot framework. It provides dependencies and annotations to simplify testing. It includes testing frameworks like JUnit and Mockito, also used in this project. It includes utilities for testing web applications. Programmers can write efficient and comprehensive tests using this module.

2.4 Smaller libraries

The *Spring Validation API* is a module of the Java Spring Framework that provide a flexible way to perform validation of Java objects and attributes. It is based on the Java Bean Validation API and extends it with additional features and functionalities. It allows developers to define custom validation rules and constraints. It provides a set of in-built validators for common use cases, like @NotNull or @Pattern for regex validation. This module allows validation to be seamlessly integrated with web applications. Therefore, it ensures the quality and reliability of the application when it comes to user input.

Another library that I have used is *Mapstruct*, an open-source Java based library that simplifies the mapping between Java beans. It automates the creation of mapping code and reduces the code necessary for converting between Java beans. It uses annotations and templates to generate mapping code at compile time. It allows for a customized mapping process. Mapstruct reduces the coding process with a considerable amount when it comes to mapping and improving performance.

Lombok, another open-source library for Java, provides a set of annotations to reduce the amount of repetitive code required for common tasks, such as constructor methods, getters and setters or toString ans hashCode methods. It provides annotations for logging, null checking and many others. This library helps developers to save time and increase productivity being easily integrated in both Maven and Gradle build tools.

Gradle Linter is another tool that I used for enforcing coding standards in this Gradle project. It provides automated checks for various aspects such as syntax errors, deprecated methods, and best practices. It helps to improve the quality of the code, reduce errors, and increase readability in the whole project. This tool is easy to integrate into the Gradle build process. Gradle Linter is a great too to maintain consistency and quality throughout a Gradle project.

3. Requirements

For this web application there are two types of requirements: non-functional requirements, and functional requirements.

3.1 Non-Functional Requirements

Non-functional requirements contain demands that concern the conceptual properties of a product. They do not say what to do, but what properties the web application needs to have while doing the actions stated by functional requirements. The application should be:

1. *Intuitive and easy to use:* The GUI of the application should not be a burden for the administrator or the user to use. It should be clean and intuitive.
2. *Reliable, deal with incorrect user inputs:* The inputs and outputs should be validated, password encrypted and interfaces separated. Therefore, the application should display accurate data and only data available to that type of user that is logged in.
3. *Responsive:* the time for the web application to load and switching in between pages should happen in a very short time period. User shouldn't wait for the application's response.
4. *Display messages that help the user:* The application should always validate the input of the users and in case of issues the details of the problem should also be displayed. For example, the incorrect registration data should be displayed with an error message like incorrect first and last name or username is already taken, choose another one.
5. *Efficient*: As every application, efficiency is a key requirement that can manifest itself in different forms throughout the development and maintenance process.
6. *Entertaining, instructive*: in the case of such an application, the user should learn from the questions and answers they find on the page, it should help them resolve their issues. It should also be entertaining, since they use it to interact with other users, to create a community around the users with the same fields of interests.

3.2 Functional Requirements

Functional requirements say what the application should be able to do, but without mentioning how that should be achieved. They should be compliant with the SMART requirement's properties; they should be specific, measurable, attainable, realistic, and traceable. Our functional requirements are presented at 1.2 Key features.

4. Design Diagrams

4.1 Use-Case Diagram

The use-case diagram is aimed to graphically capture the system's actors, i.e. the user, and the actions which can be performed. This diagram provides a structure for our application as it helps to identify components in the design phase. It also helps to capture the requirements which were presented in detail.

Use Case 1: login user
Primary Actor: user
Main Success Scenario:

1. User searches for the web application in their browser.
2. They get to the login page of the app.
3. They introduce their username and password.
4. They are redirected to the main page with all the questions listed.

Alternative Sequence

1. If they have never logged into the system, they will click on the register button and will be redirected to the register page.
2. If one of the inputs (username or password) is incorrect, they would get an error message informing them that the username or password is incorrect. They would have to retry logging in.

Use Case 2: register user
Primary Actor: user
Main Success Scenario:

1. User is on the registration page of the app.
2. They would write their first and last names, with capital letters to be valid names.
3. They would write a username and a password, the username being unique in the system
4. They would provide their email and phone number.
5. They click on the register button to get registered.
6. They would be redirected to the login page.

Alternative Sequence

1. The user writes first or last name with lower case first letter, then get an error message when clicking on the register button that names must start with upper case letters.
2. The user writes a username already present in the system. They would get an error message informing them that the username is already taken and they should choose another username.
3. The user writes and incorrect email address that does not have a correct format. They would get an error message informing them that the email address is incorrect.
4. The user writes a phone number that is not valid or it contains letters or other special characters. They would get an error message informing them that the phone number is incorrect.
5. If any of the fields were left blank an error message appears that no fields can be left empty.
6. The user already has an account. Then the message will appear that "you already have an account. Try logging in!".

Use Case 1: user adds a question
Primary Actor: user
Main Success Scenario:

1. User is on the questions page of the application.
2. At the top they should fill out the form with the required fields for a new question
3. User should write a title, a content and optionally some tags, separated with ;
4. After clicking in the add question button the user should see their new question added at the top of the questions posted previously.

Alternative Sequence

1. If the user doesn't provide title or content to their question, they should get an error message that some of the required fields were left blank.
2. If the user doesn't insert the tags in the correct way, they should get an error message that tags were not written in the correct input form.

Use Case 1: user reacts to a question
Primary Actor: user
Main Success Scenario:

1. The user is on the questions page of the application.
2. User clicks on Like or Dislike buttons positioned under a question.
3. The page refreshes with the reaction and the vote count changes according to the button the user pressed (+1 for a Like, -1 for a Dislike).
4. The like and dislike buttons disappear from that question.

Alternative Sequence

1. If the user tries to react to their own question, they will find no way to do that since there will be no reaction buttons under one's own question.

Use Case 1: user views a question
Primary Actor: user
Main Success Scenario:

2. The user is on the questions page of the application
3. The user clicks on the View button under the question they wish to see.
4. A new page pops up with the question and all the answers listed according to their vote count in descending order.

Use Case 1: user comments on a question
Primary Actor: user
Main Success Scenario:

1. The user is on the view question page of the application.
2. User writes a text in the comment text field.
3. User clicks on the add comment button.
4. The page refreshes with the comment appearing under the question, as the last among the already posted comments .

Alternative Sequence

1. The user doesn't insert any text in the comment field. An error appears that a comment must have some text in it.

Use Case 1: user reacts to a comment on a question
Primary Actor: user
Main Success Scenario:

1. The user is on the view question page of the application.

2. User clicks on the Like/Dislike button under one of the comments.
3. The page refreshes with the vote count changes for that comment.
4. The reaction buttons disappear from that comment.

Alternative Sequence

1. If the user tries to react to their own comment, they will find no way to do that since there will be no reaction buttons under one's own comment.

Use Case 1: user filters the questions
Primary Actor: user
Main Success Scenario:

2. The user is on the questions page of the application.
3. On the top they select from the drop-down list the option by which they want to filter.
4. If it is indicated, they write in the text field underneath the tag, username or keyword they want to filter by.
5. They click on the filter button.
6. The page refreshes with only the questions that are selected by that filtering.

Alternative Sequence

1. The text field was left blank even though the filter button has been clicked on. An error message appears that filtering needs a text input**.**

Use Case 1: user logs out
Primary Actor: user
Main Success Scenario:

1. The user is on the questions page of the application.
2. User clicks on the logout button.
3. The login page appears again and the session cookie is cleared.

5.Architecture

The architectural pattern followed in this project is the Layered Architectural Pattern. It suggests that there are several layers that are indicated by the separate packages. I have worked with 4 layers; Model, DAO, Service and Controller layers.

5.1 Model

This layer contains all the entities used in the project. They are: User, Question, Answer, QuestionVote, AnswerVote, Tag and BaseEntity. The BaseEntity class provides the unique id-s for all the other classes. This class is the superclass of User, Question, Answer, Tag, QuestionVote and AnswerVote.

5.2 DAO

This package contains the data access objects and the JPADAOs. The generic DAO interface is extended by the other dao-s: AnswerDao, AnswerVoteDao, QuestionDao, QuestionVoteDao, TagDao, UserDao.

Inside the Dao package there is a Jpa package that contains the JpaDaos. These extend the Dao interfaces and the generic JpaRepository. They generate the methods for the various CRUD operations automatically. No methods are instantiated in them.

5.3 Service

This package contains the service classes that call the methods from the Daos. These service classes are: AnswerService, AnswerVoteService, QuestionService, QuestionVoteService, TagService, and UserService. For each model class there is a separate service class that implements the CRUD methods of these objects.

5.4 Controller

This package contains the controller classes, one for each model: AnswerController, AnswerVoteController, QuestionController, QuestionVoteController, and UserController. These classes class the methods from the service classes and they are each associated to an endpoint.

In this package there is a dto package that contains all the in and out dtos used for each model. They make it easier to get input from the user or send an instance to the user without having to include all attributes of that specific instance. They are mapped to the model classes and the model classes mapped to oudtos inside the mapper package. There are mapper classes for each model. They use Mapstruct to shorten and optimize the mapping. Therefore, the methods are not written by the developer, only the signature. The method is automatically generated. This class contains the connection between back-end and front-end.

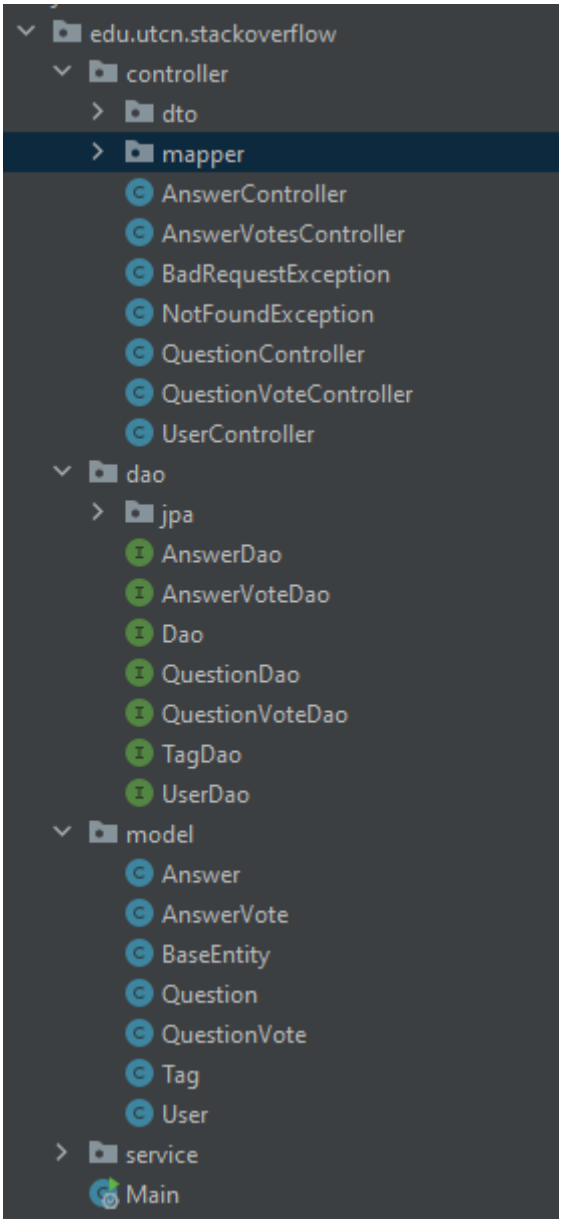


Figure 1

Architecture Hierarchy

6. UML Diagrams

6.1 UML Package Diagram

A package diagram is a type of UML diagram that shows how various components are organized in packages. A package is a container that holds related components together. By using a package diagram, one can visualize the structure of a software project and understand how different components interact with each other. The package diagram shows all packages included in the project.

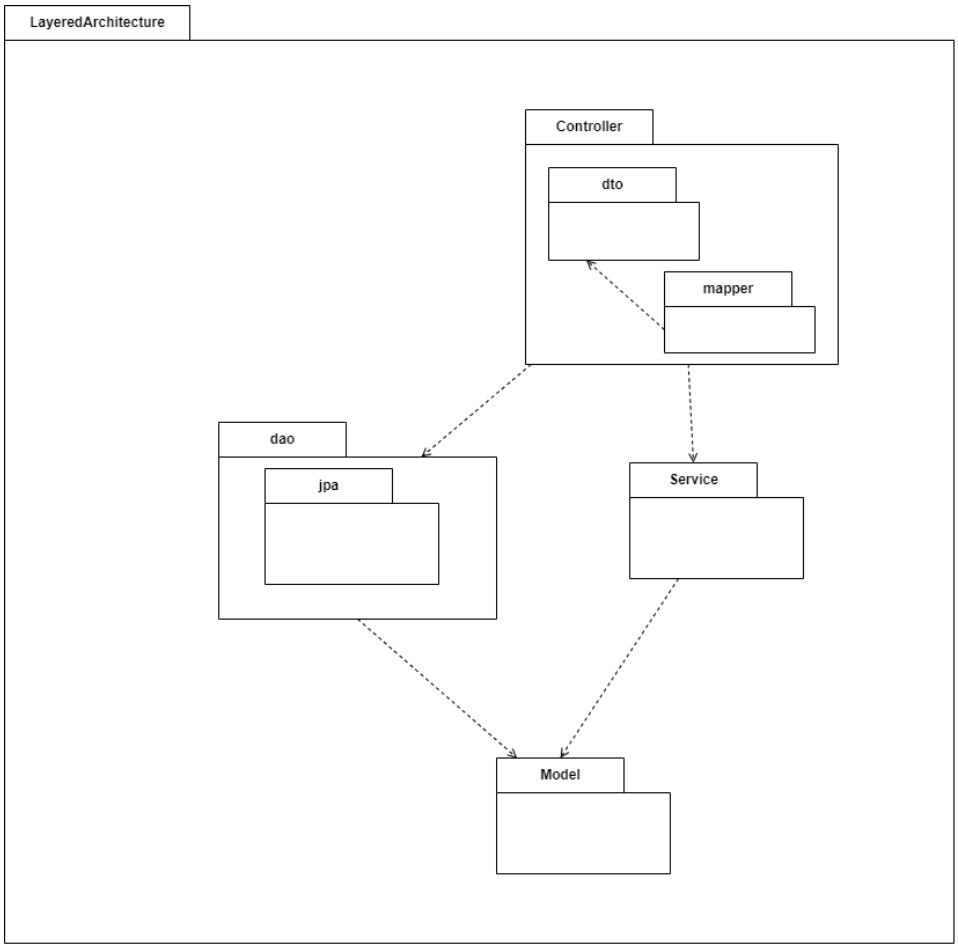


Figure 2 Package Diagram

6.2 UML Class Diagrams

The class diagram is a type of UML diagram that shows the structure of a software project by presenting the classes, interfaces, and relationships between them. A class diagram helps to design and understand a software project's structure and helps to ensure that different parts of the software fit together properly.

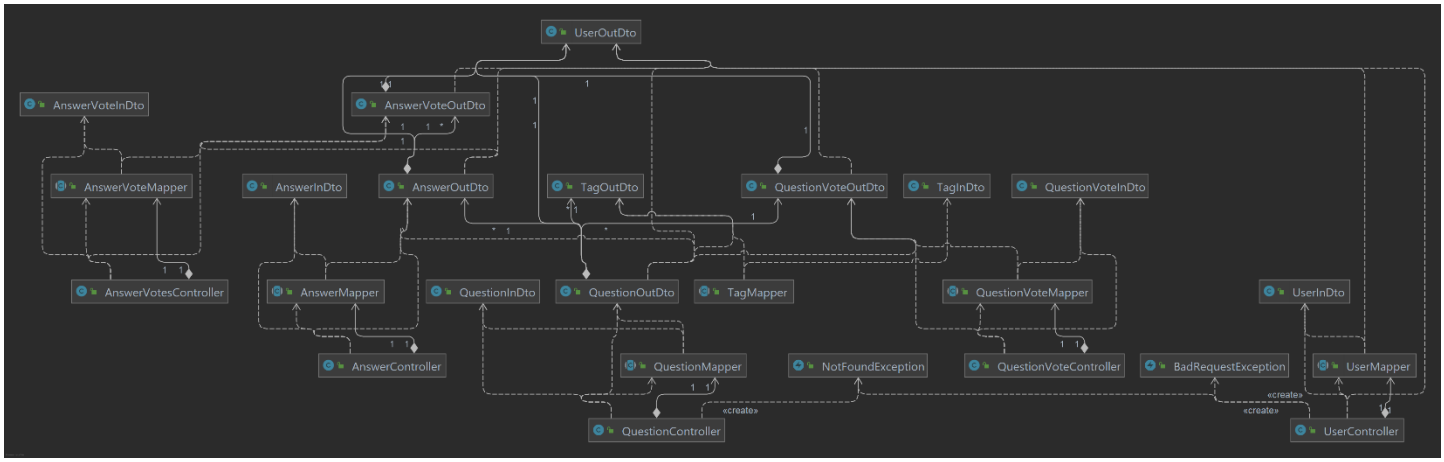


Figure 3 Class Diagram of controller package

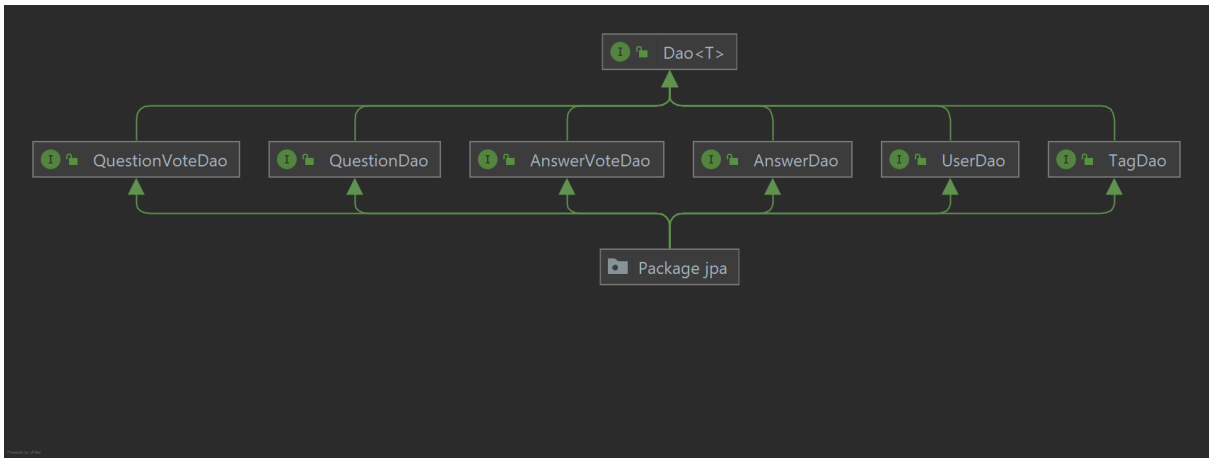


Figure 4 Class Diagram of dao package

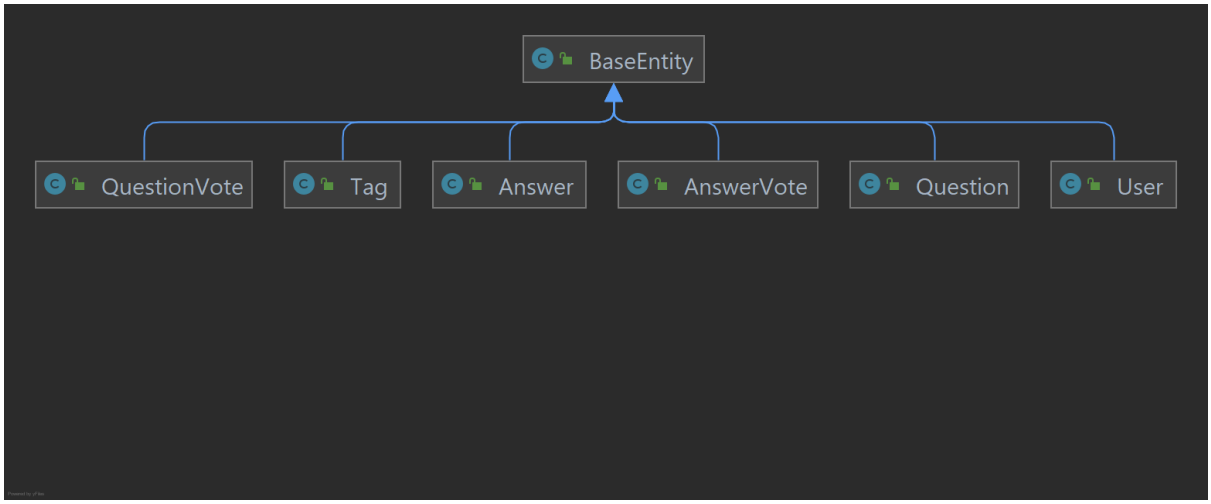
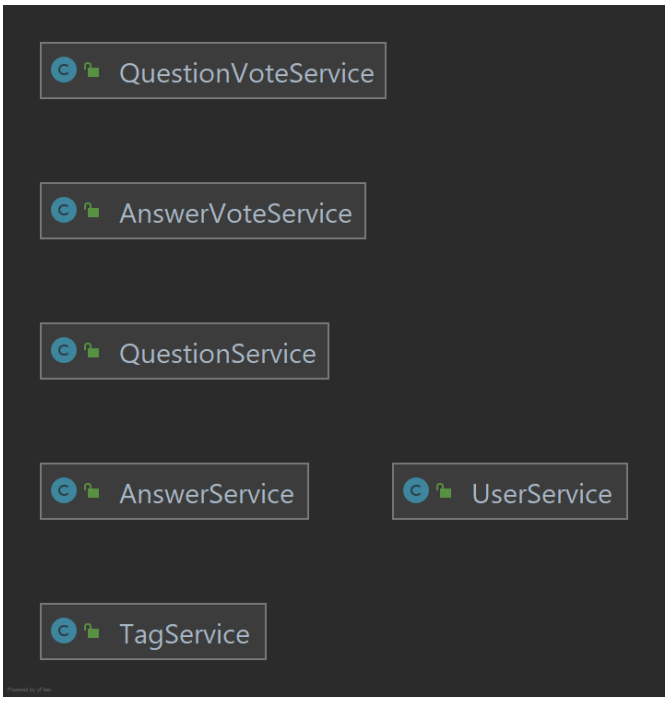


Figure 5 Class Diagram of model package

Figure 6 Class Diagram of service package



6.3 UML Database Diagram

A UML Database Diagram is a type of UML diagram that illustrates the structure of a database. It shows the tables in the database, the fields in each table, and the relationship between the tables. It helps to design and understand the database structure, and it allows developers to plan the database structure. A database diagram is a map of the database that shows how data is organized and related to each other.

In this project a MySQL database is used, 'Stackoverflow'.

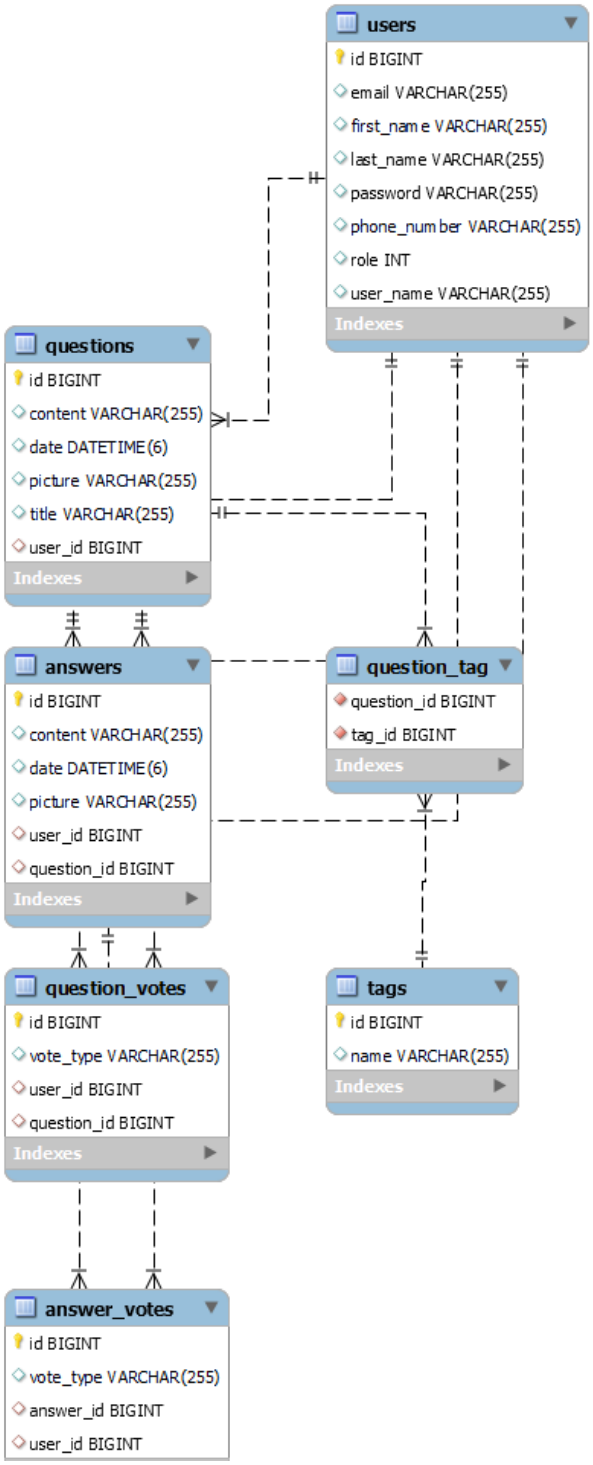


Figure 7 Database Diagram

6.4 Endpoint Requests

There are 5 different endpoints of the application. There are the 5 controller classes in the controller package. Each of them will be presented in more details below.

/users

The UserController class handles the HTTP requests that target the /users endpoint. The controller works according to REST conventions. The handles GET, POST, PUT and DELETE requests.

@GetMapping

This method takes no parameters. It returns all the users registered in the database as UserOutDtos. Uses the getAllUsers() method of the service class.

@GetMapping(“/{id}”)

This method takes the id of a user as path variable and calls the getUserById() method of the service class. If it finds the user, it returns a UserOutDto. Otherwise, it throws a NotFoundException.

@GetMapping(“/username/{userName}”)

This method takes the username of a user as path variable and calls the findUserById() method of the service class. It is finds the user, it return a UserOutDto. Otherwise, it throws a BadRequestException.

@PostMapping

This method takes a UserInDto parameter in the http request body. It validates the input, it checks if the user is present in the database. If yes, it throws a BadRequestException. Otherwise, it saves the new user in the database. It returns the UserOutDto of the new user.

@PutMapping(“/{id}”)

This method takes the id of a user as path variable and a UserInDto in the request body. It checks if the user is present in the database. If no, then it throws a NotFoundException. Otherwise, it updates the user in the database. It returns the UserOutDto of the updated user.

@DeleteMapping(“/{id}”)

This method takes the id of a user as path variable. It checks if the user is present in the database. If no, then it throws a NotFoundException. Otherwise, it deletes the user from the database. It does not return anything.

/questions

The QuestionController class handles the HTTP requests that target the /questions endpoint. The controller works according to REST conventions. The handles GET, POST, PUT and DELETE requests.

@GetMapping

This method takes no parameters. It returns all the questions registered in the database as QuestionOutDtos. Uses the getAllQuestions() method of the service class.

@GetMapping(“/{questionId}”)

This method takes the id of a question as path variable and calls the getQuestionById() method of the service class. If it finds the question, it returns a QuestionOutDto. Otherwise, it throws a NotFoundException.

@GetMapping(“/filter/tag/{tagName}”)

@GetMapping(“/filter/author/{authorName}”)

@GetMapping(“/filter/title/{keyword}”)

@PostMapping

This method takes a QuestionInDto parameter in the http request body. It validates the input, it checks if the question is present in the database. It saves the new question in the database. It returns the QuestionOutDto of the new question.

@PutMapping(“/{questionId}”)

This method takes the id of a question as path variable and a QuestionInDto in the request body. It checks if the question is present in the database. If no, then it throws a NotFoundException. Otherwise, it updates the question in the database. It returns the QuestionOutDto of the updated question.

@DeleteMapping(“/{questionId }”)

This method takes the id of a question as path variable. It checks if the question is present in the database. If no, then it throws a NotFoundException. Otherwise, it deletes the question from the database. It does not return anything.

/answers

The AnswerController class handles the HTTP requests that target the /answers endpoint. The controller works according to REST conventions. The handles GET, POST, PUT and DELETE requests.

@GetMapping

This method takes no parameters. It returns all the asnwers registered in the database as AnswerOutDtos. Uses the getAllAnswers() method of the service class.

@GetMapping(“/{id}”)

This method takes the id of an answer as path variable and calls the getAnswerById() method of the service class. If it finds the asnwer, it returns a AnswerOutDto. Otherwise, it throws a NotFoundException.

@PostMapping

This method takes a AnswerInDto parameter in the http request body. It validates the input, it checks if the answer is present in the database. It saves the new answer in the database. It returns the AnswerOutDto of the new answer.

@PutMapping(“/{id}”)

This method takes the id of an answer as path variable and an AnswerInDto in the request body. It checks if the answer is present in the database. If no, then it throws a NotFoundException. Otherwise, it updates the answer in the database. It returns the AnswerOutDto of the updated asnwer.

@DeleteMapping(“/{id}”)

This method takes the id of an answer as path variable. It checks if the answer is present in the database. If no, then it throws a NotFoundException. Otherwise, it deletes the answer from the database. It does not return anything.

/questionvotes

The QuestionVoteController class handles the HTTP requests that target the /questionvotes endpoint. The controller works according to REST conventions. The handles POST requests. There is no need for the other types of requests to be handled because a vote can only be created. It cannot be updated, deleted or retrieved.

@PostMapping

This method takes a QuestionVoteInDto parameter in the http request body. It validates the input. It saves the new question vote in the database. It returns the QuestionVoteOutDto of the new question vote.

/answervotes

The AnswerVoteController class handles the HTTP requests that target the /answervotes endpoint. The controller works according to REST conventions. The handles POST requests. There is no need for the other types of requests to be handled because a vote can only be created. It cannot be updated, deleted or retrieved.

@PostMapping

This method takes a AnswerVoteInDto parameter in the http request body. It validates the input. It saves the new answer vote in the database. It returns the AnswerVoteOutDto of the new answer vote.

7. The Application

The link to the github repository where the source code is:

https://github.com/JakabSarolta/Software-Design-Assignment

There is a database .sql dump file that can be imported and then the application will run on any latop or computer that has MySQL installed.

8. Testing

8.1 White-box Testing

For white box testing we have tested separate methods and classes on their own. Since the most important package that contains the logic behind the application is the controller package, the test cases are targeting the classes mostly from this package.

```java
@RunWith(SpringRunner.class)
@SpringBootTest
public class UserControllerTest {
    @Autowired
    private UserController userController;

    @Test
    void allUsersShouldNotBeNull() { Assertions.assertNotNull(userController.getAllUsers()); }

    @Test
    void unknownUsernameUserShouldThrowException() {
        Assertions.assertThrowsExactly(BadRequestException.class, () -> userController.getUserByUserName("cocacola"));
    }
```

Figure 8 UserController test case example

9. Conclusions