

Deliverable 2 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.....	5
1 Introduction.....	6
1.1 Background.....	6
1.2 Key features	6
2 Technology	7
2.1 Spring Framework	7
2.2 Spring Data JPA.....	7
2.3 Spring Test Framework.....	7
2.4 Smaller libraries	8
2.5 Express.js	8
2.6 Embedded JavaScript templates.....	8
3 Requirements	9
3.1 Non-Functional Requirements	9
3.2 Functional Requirements	9
4 Design Diagrams.....	10
4.1 Use-Case Diagram	10
5 Architecture.....	15
5.1 Back-End Architecture.....	15
5.1.1 Model	15
5.1.2 DAO	15
5.1.3 Service.....	15
5.1.4 Controller	15
5.2 Front-End Architecture	16
5.2.1 Routes	16
5.2.2 Views	16
6 UML Diagrams	17
6.1 UML Package Diagram	17
6.2 UML Class Diagrams	17
6.3 UML Database Diagram	20
6.4 UML Deployment Diagram	21
6.5 UML Component Diagram	22
6.6 UML Domain Model	23
7 Endpoint Requests	24

8	The Application	27
9	Testing.....	32
9.1	Unit Testing	32
10	Conclusions and Further Development.....	33
11	Bibliography	34

Table of Figures

Figure 1 Use Case Diagram for User	10
Figure 2 Use Case Diagram for Moderator.....	11
Figure 3 Package Diagram	17
Figure 4 Class Diagram of controller package.....	18
Figure 5 Class Diagram of dao package	18
Figure 6 Class Diagram of model package	18
Figure 7 Class Diagram of service package.....	19
Figure 8 Database Diagram	20
Figure 9 UML Deployment Diagram.....	21
Figure 10 UML Component Diagram.....	22
Figure 11 UML Domain Model	23
Figure 12 Welcome Page	27
Figure 13 Login Page.....	28
Figure 14 Registration Page	28
Figure 15 Main Page	29
Figure 16 Filtering Questions	29
Figure 17 Edit a Question	30
Figure 18 Answer Page.....	30
Figure 19 Answer a Question	31
Figure 20 Edit an Answer	31
Figure 21 Unban a User	31
Figure 22 UserController test case example	32

Abstract

This web application aims to be a simpler version of the well-known application 'Stackoverflow'. 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 prototype 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 moderator. The key features are presented below for each kind of user.

The login system is the first window that a user/moderator 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 their phone number. All these inputs get validated. After that, the user is logged 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 edit or delete them. The user can post new questions by filling out a form with a question title, some content, a picture(optional), and by adding tags(optional).

If the user clicks on the 'View' button, they can view all the answers posted for that question. The answers 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 add their own answer. They can also react to the comments posted by others with a like or a dislike.

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, 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.

The moderators can perform any of the operations mentioned above. Additionally, they can edit or delete any user's question or answer. They can ban and unban users for using the application inappropriately.

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 JPA Repository, 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 [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 in 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` and `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.

2.5 Express.js

Express.js is a popular and widely used web framework for building web applications with Node.js. It provides a robust set of features and utilities for building web applications with HTTP methods, middleware, and routing. With its minimal and flexible design, developers can quickly and easily create server-side applications that handle a wide range of client requests and responses. Overall, Express.js is a powerful and reliable tool for building scalable, fast, and efficient web applications.

2.6 Embedded JavaScript templates

EJS (Embedded JavaScript) is a simple templating language that enables developers to generate HTML markup with plain JavaScript. EJS templates are easy to read and write and can be used to create reusable code snippets for different parts of a web application. EJS templates are similar to other templating engines, but with the added benefit of being able to execute JavaScript code within the template. This allows developers to create dynamic templates that can display data from different sources and respond to user input. EJS is a popular choice for building Node.js web applications and is widely used in combination with the Express.js web framework.

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 at graphically capturing 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.

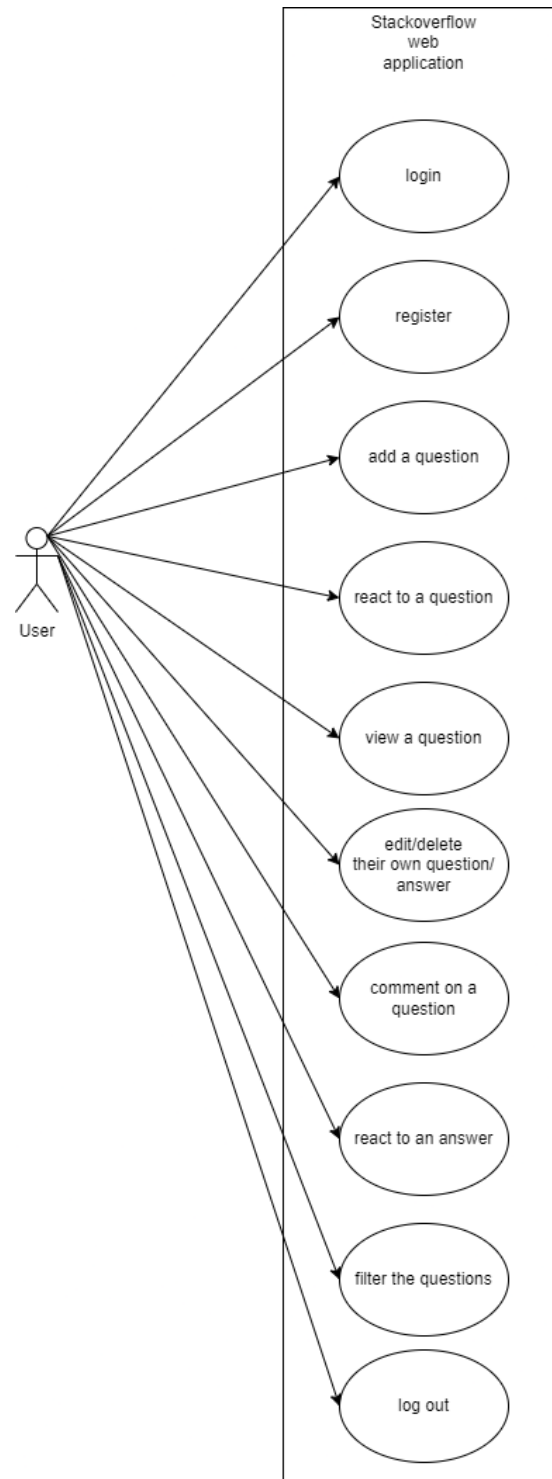


Figure 1 Use Case Diagram for User

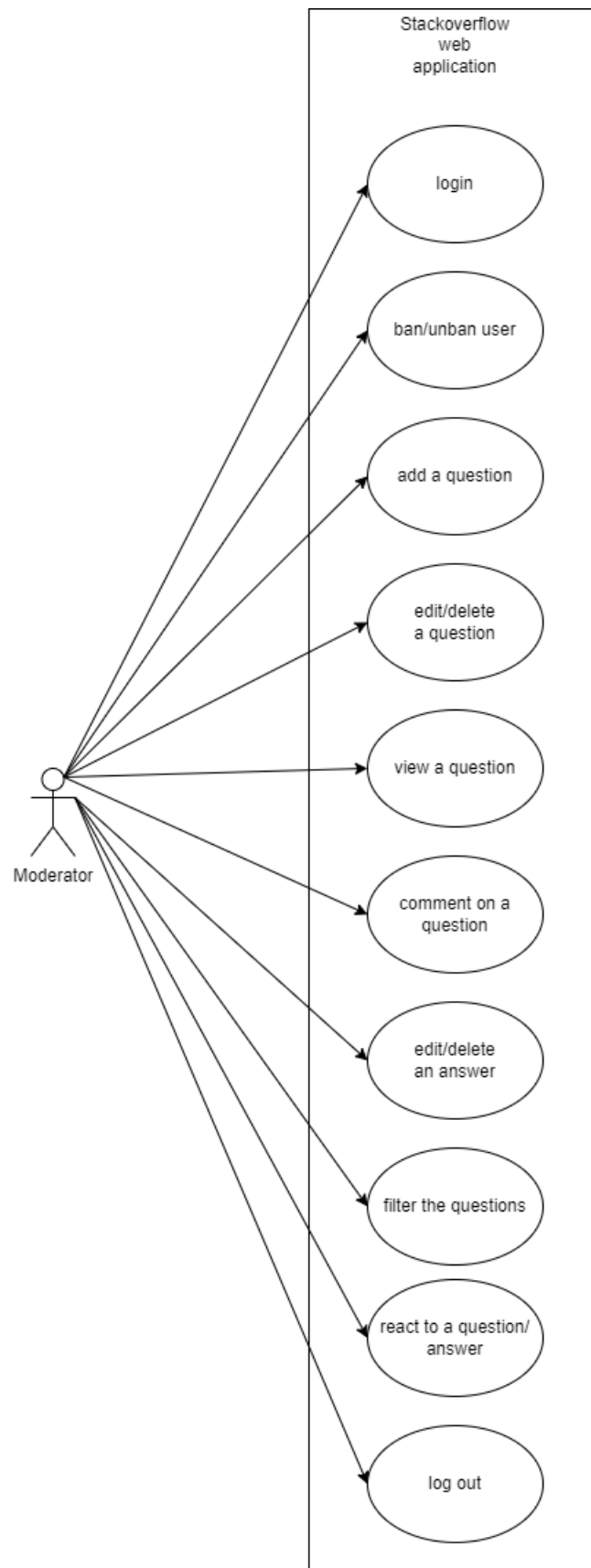


Figure 2 Use Case Diagram for Moderator

Use Case 1: login

Primary Actor: user/moderator

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

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 an 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 3: add a question

Primary Actor: user/moderator

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 4: react to a question

Primary Actor: user/moderator

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 5: view a question

Primary Actor: user/moderator

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 6: edit/delete a question/answer

Primary Actor: user/moderator

Main Success Scenario:

1. The user is on the view question page of the application.
2. They click on the Edit button under the question or under the answer they wish to edit.
3. A new page appears where they must write the part of the content they wish to change (or upload a new photo).
4. After clicking on the Edit button again, their changes will be saved and they will be redirected to the previous page.

Alternative Sequence

1. The user can only edit or delete their own question or answer. However, a moderator can edit or delete any question or answer.

Use Case 7: answer a question

Primary Actor: user/moderator

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 8: react to an answer

Primary Actor: user/moderator

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 9: filter the questions

Primary Actor: user/moderator

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 10: ban/unban users

Primary Actor: moderator

Main Success Scenario:

1. The moderator is on the questions page of the application.
2. If they see inappropriate content on the page, they can click on the ban button that appears under the vote count of each question.
3. Then the user gets an email and a text message that informs them that they have been banned.
4. If the moderator wants to then unban them, they can click on the Unban users button in the top right corner.
5. A new page appears with a list of names containing the currently banned users. The moderator can choose a user and unban them.

Use Case 11: log out

Primary Actor: user/moderator

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

5.1 Back-End 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.

5.1.1 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 ids for all the other classes. This class is the superclass of User, Question, Answer, Tag, QuestionVote and AnswerVote.

5.1.2 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.

5.1.3 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.

5.1.4 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 outdtos 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.

5.2 Front-End Architecture

The architecture is represented on a directory level. The 3 main directories being: routes, static and views.

5.2.1 Routes

This directory contains all the JavaScript files that route the clients' requests coming to the back-end by performing fetch-es. There is a main router (express app) which routes the requests from the root ("/"). Furthermore, there are 5 routers that route the requests by a given keyword: answers.js ("/answers/..."), login.js ("/login"), questions.js ("/questions/..."), register ("/register"), and users.js ("/users/...").

5.2.2 Views

This directory contains the ejs files that contain the html code for each page of the application. The partials directory contains the navbar.ejs, the common navbar that all the other pages import. The ejs files in this directory are: welcome.ejs, login.ejs, register.ejs, questions.ejs, answers.ejs, updateAnswer.ejs, updateQuestion.ejs, unban.ejs, astronaut.ejs. The source of the code for the error page, the astronaut.ejs is mentioned in the references.

5.2.3 Static

This directory contains the style sheets for the webpages that get rendered. There is a directory with the pictures the users upload for their questions or answers.

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.

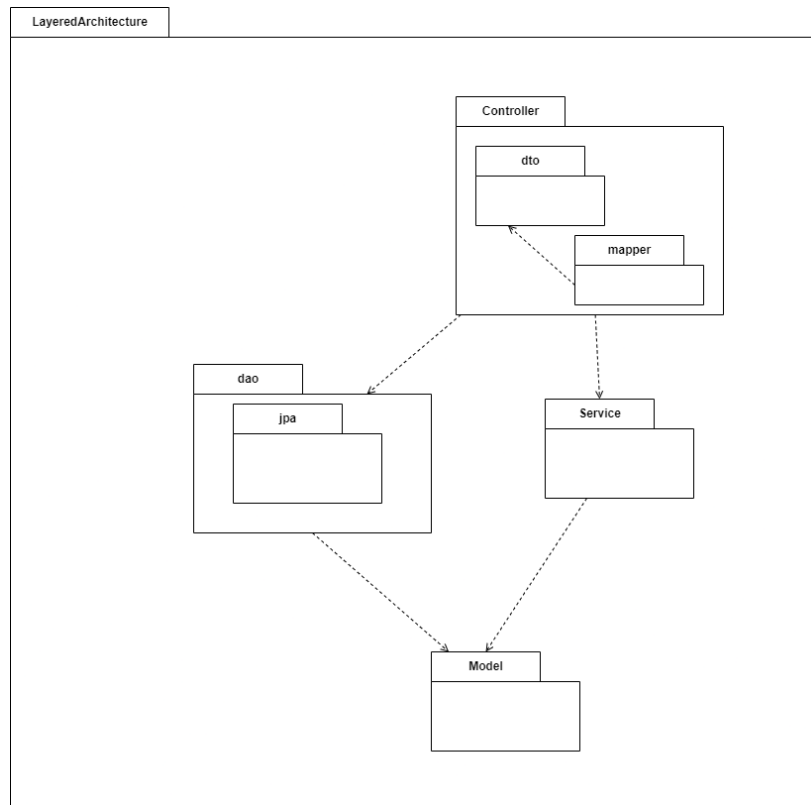


Figure 3 Package Diagram

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.

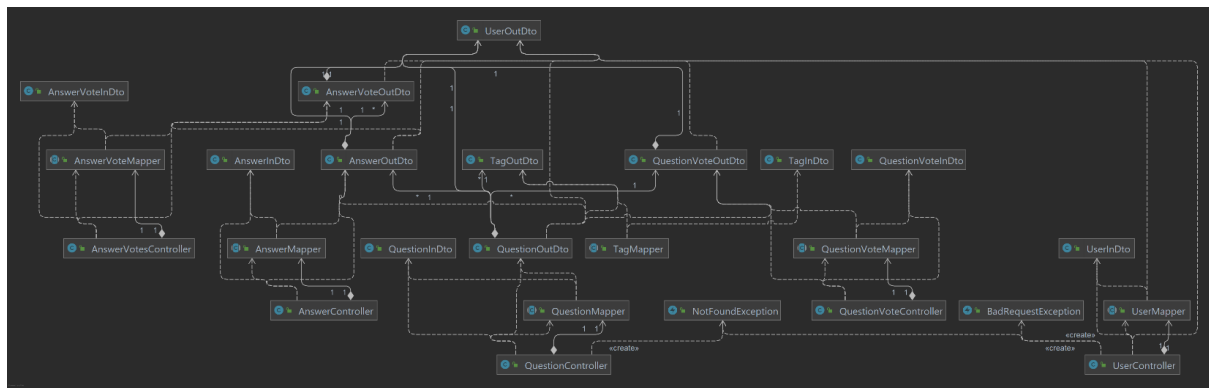


Figure 4 Class Diagram of controller package

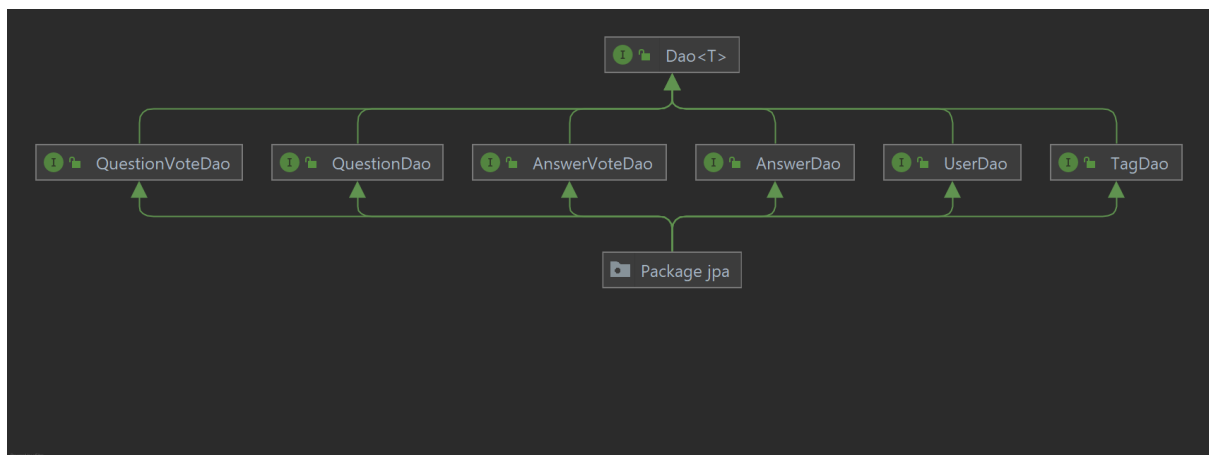


Figure 5 Class Diagram of dao package

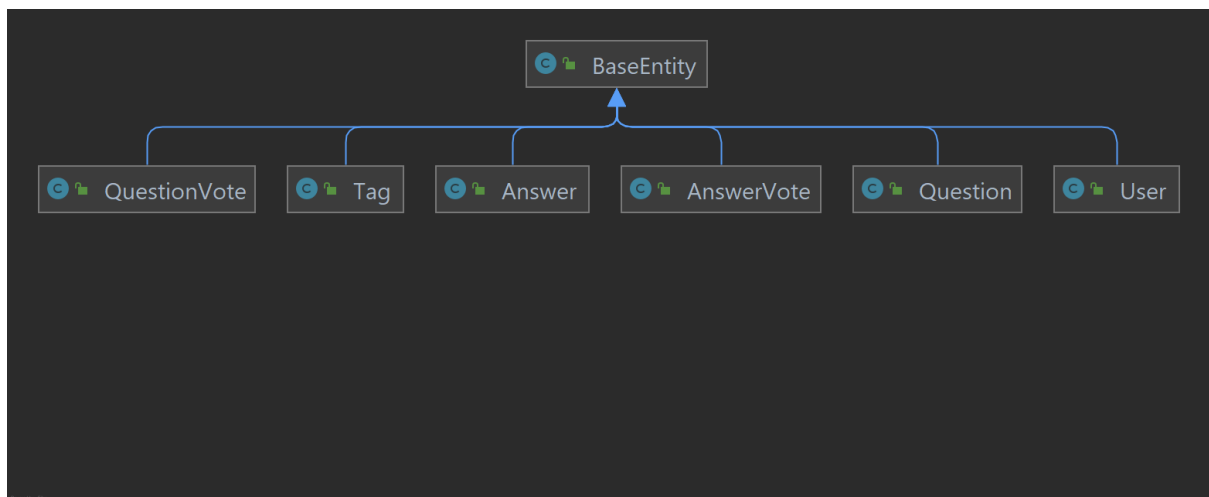


Figure 6 Class Diagram of model package

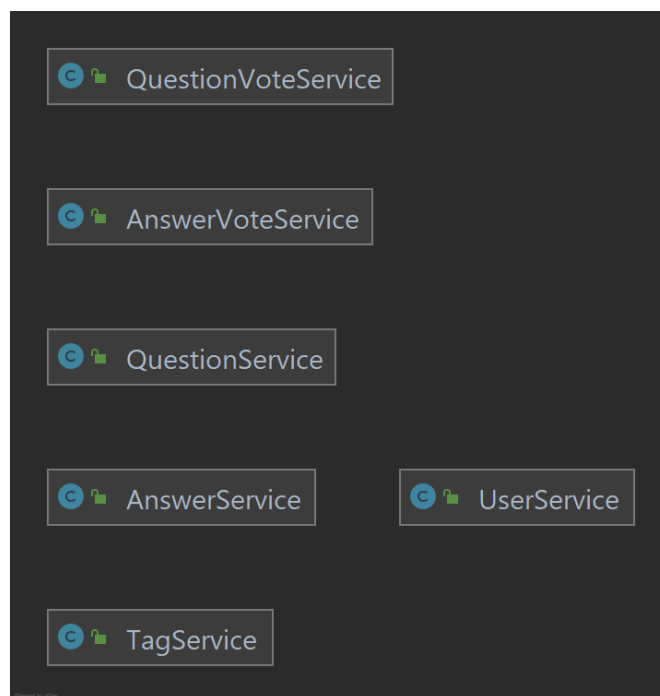


Figure 7 Class Diagram of service package

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.

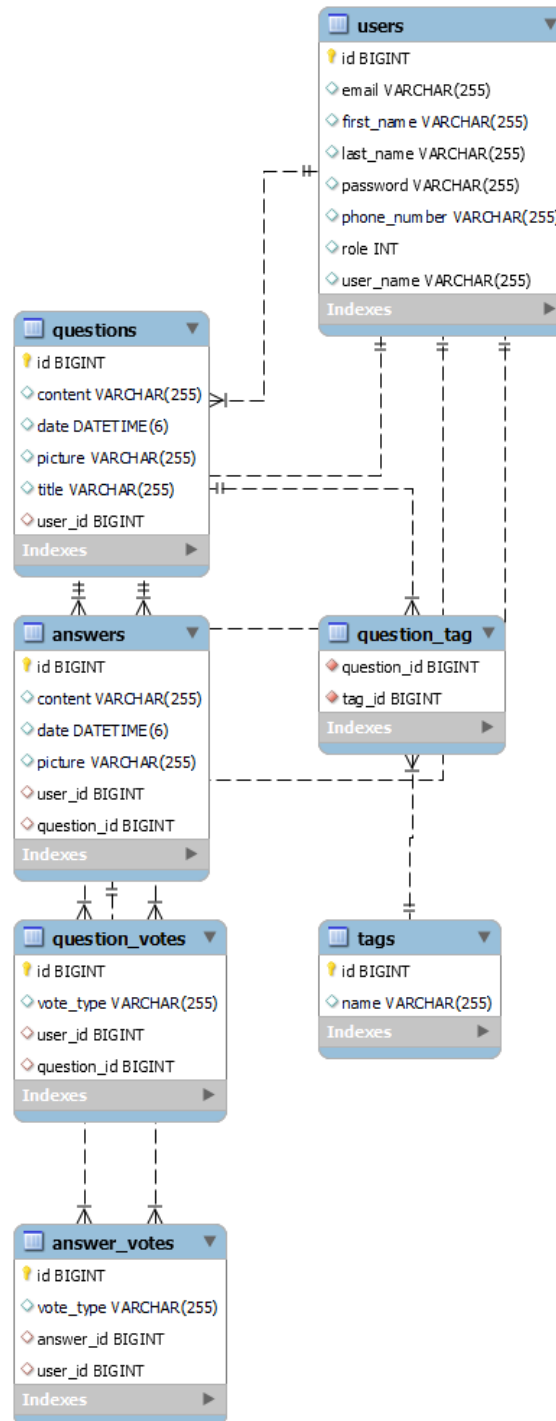


Figure 8 Database Diagram

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

6.4 UML Deployment Diagram

A deployment diagram is a type of UML diagram that depicts the physical deployment of software components within a system, including hardware and network infrastructure. In the case of this web application, the deployment diagram includes nodes for the web server, application server, database server, and any other servers or resources required to support the application. Each of these nodes are connected by communication paths that show how they communicate with each other.

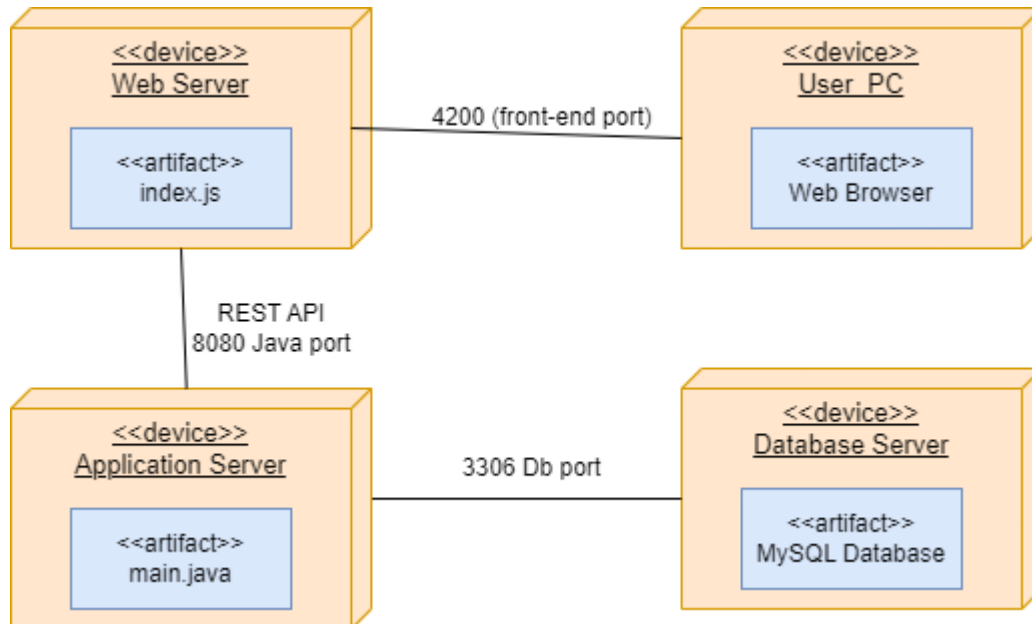


Figure 9 UML Deployment Diagram

6.5 UML Component Diagram

A component diagram is a type of UML diagram that shows the components, interfaces, and dependencies of a software system. In the case of a web application, a component diagram shows the various components of the application, such as web servers, application servers, databases, and other software components. Each component is connected to other components by dependencies, which indicate how the components depend on each other to function.

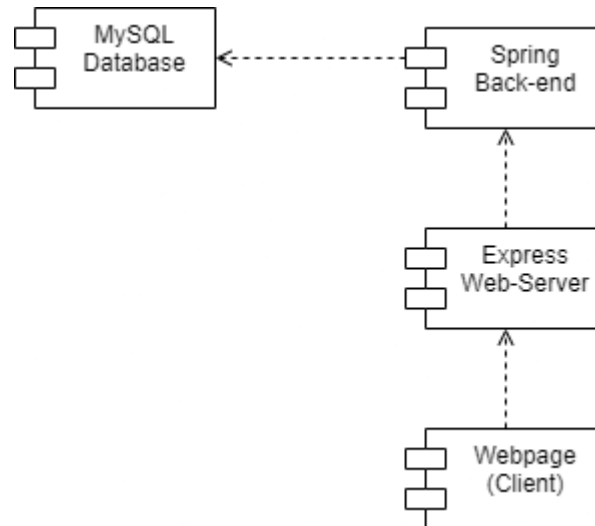


Figure 10 UML Component Diagram

6.6 UML Domain Model

A domain model is a type of UML diagram that shows the conceptual classes and relationships that make up a system. In the case of this web application, a domain model shows the key entities and relationships that are relevant to the application's domain, such as users, questions, answers, and other business objects.

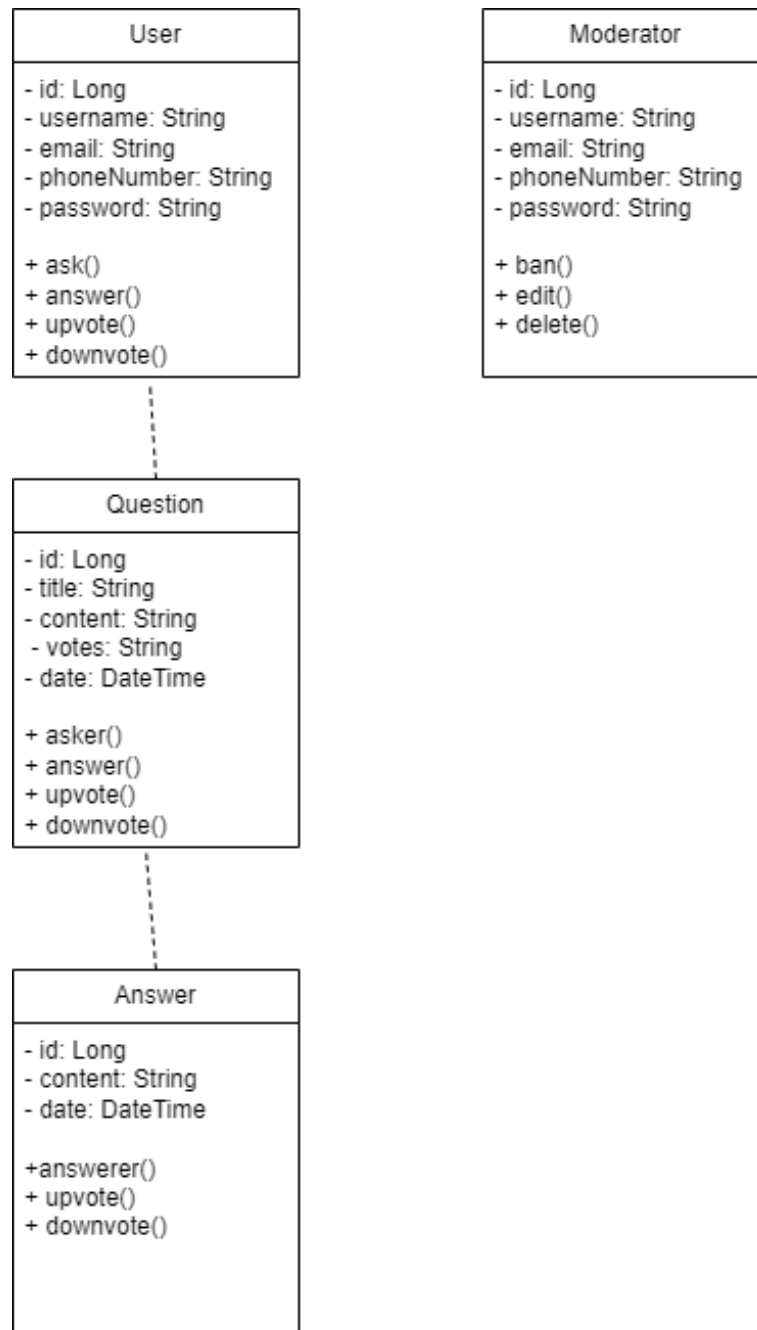


Figure 11 UML Domain Model

7 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 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 asnwrs 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 answer.

```
@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.

8 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 laptop or computer that has MySQL installed.

To show how to interact with the user interface of the system there are some steps that show all the necessary information for a user.

1. Welcome page

When a user searches for the web application, they will end up on this page. If they press Login, then the button will redirect them to the Login page.

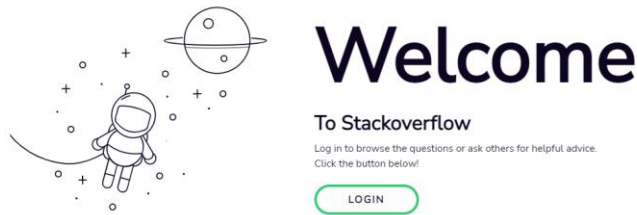
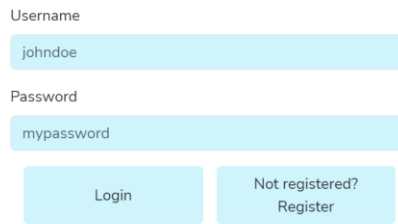


Figure 12 Welcome Page

2. Login page

The Login pages allows a user to enter their credentials (username and password), then the Login button redirects them to the Main page. If they do not have an account, then the Register button will redirect them to the Register page.



Username
johndoe

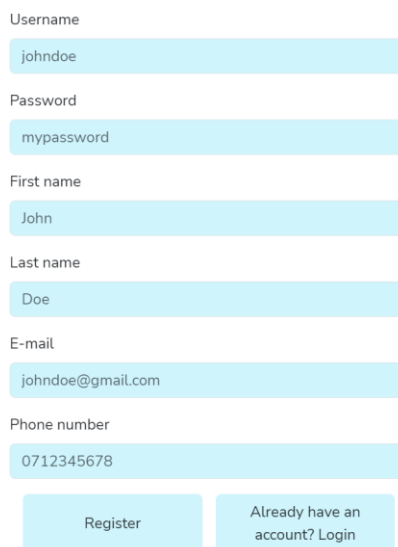
Password
mypassword

Login Not registered?
Register

Figure 13 Login Page

3. Registration page

This page allows the user to enter some personal information to make an account on the website. After that the Register button will redirect them to the Login page.



Username
johndoe

Password
mypassword

First name
John

Last name
Doe

E-mail
johndoe@gmail.com

Phone number
0712345678

Register Already have an
account? Login

Figure 14 Registration Page

4. Main page

stackoverflow Tag Filter value jgysarolta Logout

Unban users

All questions

-1 votecount **What is agile methodology?**

0 answers What is agile methodology and how is it used in software development? What are some benefits and drawbacks of using agile methodology?

Ban agile software development 2023.04.02 22:36

View Edit Delete michaelwright [-1.5]

0 votecount **What is open source software?**

1 answers What is open source software and how does it differ from proprietary software? What are some popular open source programs?

coding opensource software 2023.04.02 22:34

charlotteturner

Title

Title

Content

Content

Tags (separated by ;)

tag1;tag2;tag3...

Picture

Choose File No file chosen

Post question

Figure 15 Main Page

stackoverflow Tag coding Reset filter jgysarolta Logout

Unban users

All questions

0 votecount **What is open source software?**

1 answers What is open source software and how does it differ from proprietary software? What are some popular open source programs?

coding opensource software 2023.04.02 22:34

View Like Dislike Edit Delete charlotteturner [-1.5]

0 votecount **What is coding?**

1 answers What is coding and why is it important in the IT industry? What are some common programming languages and their uses?

Ban coding programming 2023.04.02 22:24

Title

Title

Content

Content

Tags (separated by ;)

tag1;tag2;tag3...

Picture

Choose File No file chosen

Post question

Figure 16 Filtering Questions

Title

What is agile methodology?

Content

What is agile methodology and how is it used in software development? What are sor

Tags

agile;software;development

Change to new picture (leave blank if no change)

Choose File No file chosen

Update Cancel

Figure 17 Edit a Question

5. Answer page

stackoverflow Tag Filter value jgysarotta Logout

What is open source software?

Asked 2023-04-02 22:34

What is open source software and how does it differ from proprietary software? What are some popular open source programs?

0

coding opensource software

Back to questions

charlotteturner [-1.5]

Content

Picture

Choose File No file chosen

Answer

1 answer

Open source basically means that anyone can contribute to the development. It is improved by the users and those who invest the time in it.

Figure 18 Answer Page

Content

Picture

Choose File

No file chosen

Answer

Figure 19 Answer a Question

Content

Picture

Choose File

No file chosen

Update answer

Cancel

Figure 20 Edit an Answer

6. Unban page

isabellachen

▼

Unban

Figure 21 Unban a User

9 Testing

9.1 Unit Testing

For unit testing I 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.

```
@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 22 UserController test case example

10 Conclusions and Further Development

11 Bibliography

- [1] Astronaut code: <https://codepen.io/kdbkapsere/pen/oNXLbqQ>