STACKOVERFLOW

- Assignment 1 –

Bogdan Maria

CTI – English

Group 30434, semigroup 1

Contents

1. **Introduction** 3
2. **Tech Stack** 3
3. **Use Case Description** 3
4. **Architecture** 4
5. **Database diagram** 5
6. **CRUD endpoints** 6
7. **Package diagram** 9
8. **Bibliography** 10

1. Introduction

The assignment consists in the backend part of the StackOverflow web-site, which is a simpler version of the well known site, where users can ask technical questions related to their work and other users can reply/post answers.

The users are also able to react to other users' questions or answers – each question or answer has a vote that will appear next to them and users can see this vote. The vote is computed based on all of the reacts that a question or answer has.

2. Tech Stack

- Database:
  - **MySQL**
- Backend framework & programming language:
  - **SPRING**, **JAVA**
- Frontend framework:
  - **ANGULAR**

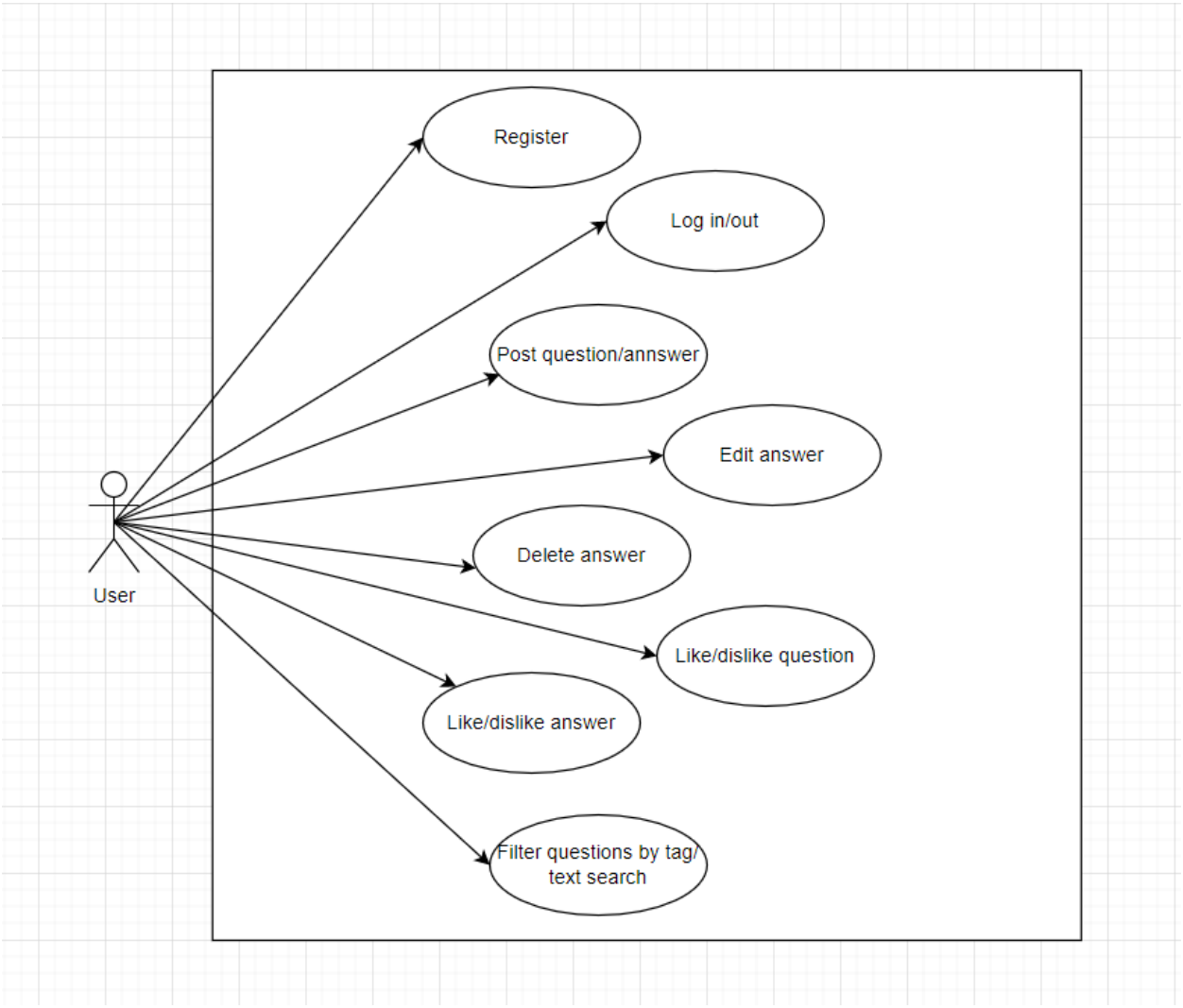
3. Use Case Description

Use Case: ***Answer a question***

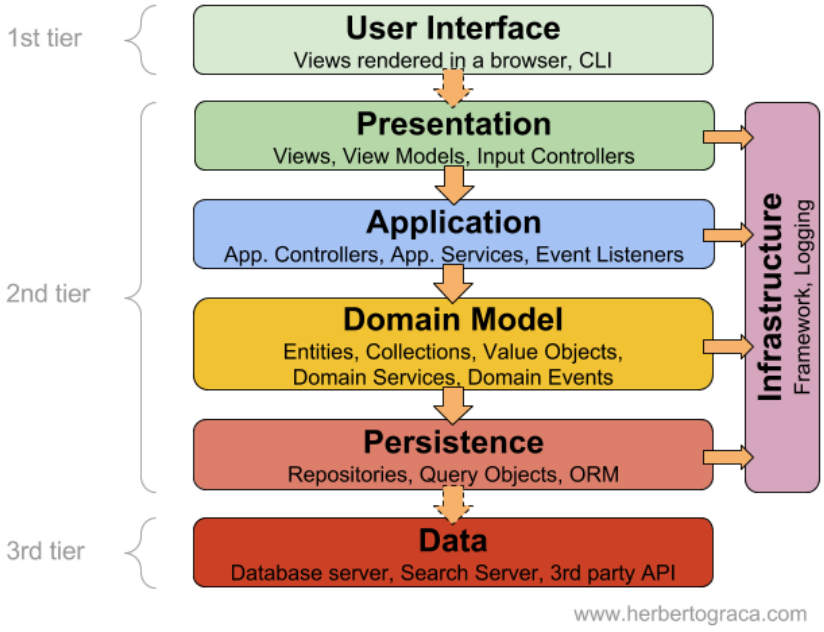
Actor: user

Main success scenario: The user has to have an account and to already be logged into that account, when posting an answer. He chooses the question he wants to answer, insert the text representing the content of his answer and posts it, by clicking *Post* button.

Exception scenario: The user tries to insert an empty text answer, so an error will be thrown and a message will be displayed, suggesting that he should insert some content/text, before clicking the *Post* button.



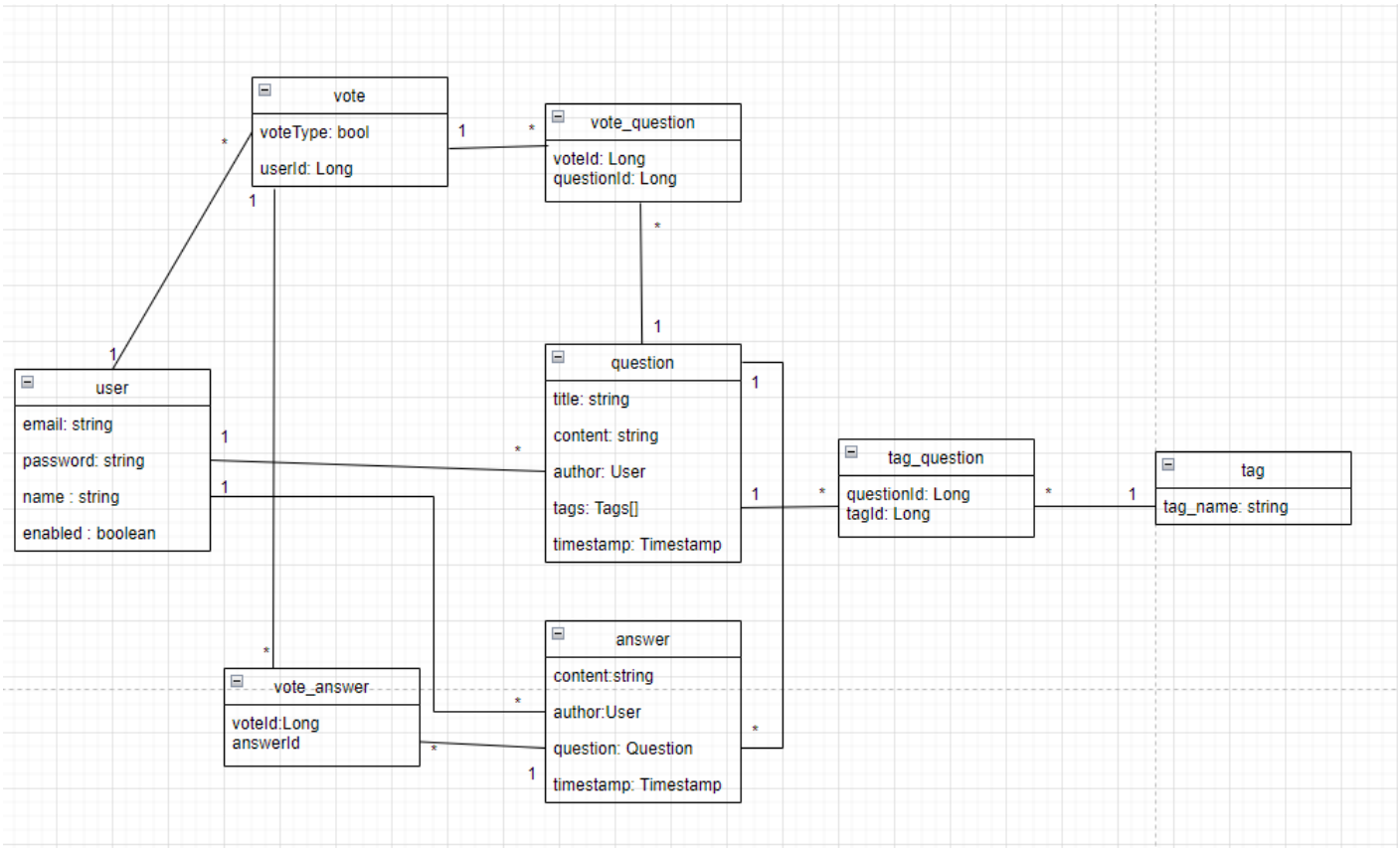
4. Architecture



This web application is based on the *Layered Architecture*, which consists in organizing modules or components that have similar functionalities, in horizontal layers – each layer will perform a specific role. This style promotes the concept of *separation of concerns*, by abstracting the view of the system. Each layer communicates with the layer below it.

The *User Interface Layer* consists in all that a client sees when enters the application. In the *Presentation Layer*, the client sends requests and waits the responses from the server. In the *Application Layer* there are the controller classes, that do not contain business logic – just redirect the requests to the service classes, where the logic actually stands – in the *Domain Model Layer*. The *Persistence Layer* consists of the DAOs (Data Access Objects) – these are used in order to separate what the user needs to see, from all the fields that are in the database. The *Data Layer* stores the entire database – tables and relations between them.

5. Database diagram



6. CRUD endpoints

- **CRUD user**
  - Log in
    - endpoint: **http://localhost:8080/api/authenticate**
    - body:

```
{
  "username":"maria3",
  "password":"maria3"
}
```

  - Log out
    - endpoint: **http://localhost:8080/logout**
  - Register
    - endpoint: **http://localhost:8080/api/register**
    - body:

```
{
  "email":"maria_bogdan@yahoo.com",
  "username":"maria4",
  "password":"maria4"
}
```

  - Delete user
    - endpoint: **http://localhost:8080/api/4**
  - Edit user
    - endpoint: **http://localhost:8080/api/4**
    - body:

```
{
  "email":"maria_bogdan@yahoo.com",
  "username":"maria24",
  "password":"maria2"
}
```

  - Get all users - endpoint: **http://localhost:8080/api/users**

- **CRUD question**
  - Filter
    - endpoint: **http://localhost:8080/api/question/filter**
    - body:

```
{
  "text":"PI",
  "tags": ["math"]
}
```

  - Delete question
    - endpoint: **http://localhost:8080/api/question/2**
  - Post question
    - endpoint: **http://localhost:8080/api/question**
    - body:

```
{
  "title":"Real number",
  "content":"Is PI a real nb?",
  "timestamp":1646489999600,
  "authorId":1,
  "tags": ["math", "number"]
}
```

  - Edit question
    - endpoint: **http://localhost:8080/api/question/2**
    - body:

```
  {
    "title": "What is the nth digit of PI",
    "content": "How can I calculate the nth
            digit of PI in Python language",
    "timestamp": "2022-03-05T14:11:37.000+00:00"
  }
```

- Get all question
  - endpoint: **http://localhost:8080/api/question**
- Like question
  - endpoint: **http://localhost:8080/api/vote/likeQuestion/3/2**
- Dislike question
  - endpoint: **http://localhost:8080/api/vote/dislikeQuestion/5/2**
- Get all tags for question
  - endpoint: **http://localhost:8080/api/tag/forQuestion/3**

- **CRUD answer**
  - Delete answer
    - endpoint: **http://localhost:8080/api/answer/4**
  - Post answer
    - endpoint: **http://localhost:8080/api/answer**
    - body:

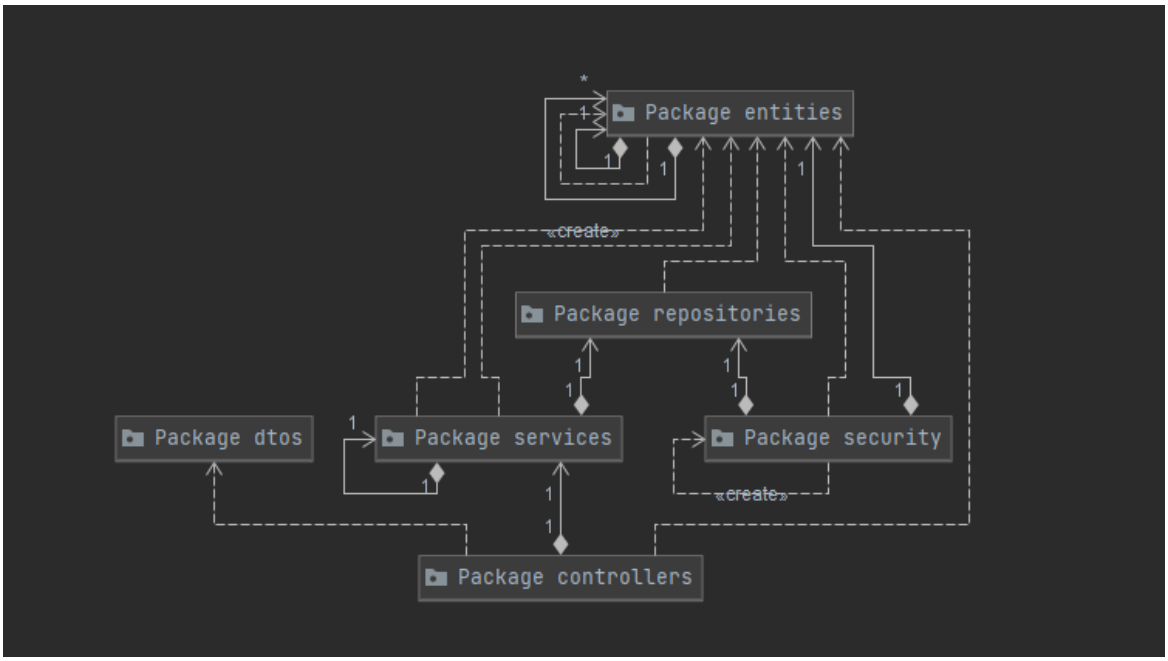
```
{
  "content": "The answer is quite
              obvious, you need to
              compute the sum!",
  "authorId": 2,
  "questionId": 1
}
```

  - Edit answer
    - endpoint: **http://localhost:8080/api/answer/4**
    - body:

```
{
  "content": "The answer is not
              easy to be found..."
}
```

- Get all answers
  - endpoint: **http://localhost:8080/api/answer**
- Like answer
  - endpoint: **http://localhost:8080/api/vote/likeAnswer/3/2**
- Dislike answer
  - endpoint: **http://localhost:8080/api/vote/dislikeAnswer/5/2**

7. Package diagram



The package diagram illustrates the concept of layered architecture, especially the *separation of concerns* – we can see that the packages are split based on their functionalities and that they communicate with each other and use each other's logic.

The **controllers** package does not contain business logic, it just redirects the requests to the service classes, where the logic actually stands and that is in the **services** package – this package contains the methods that deal with actual backend logic and that call the queries that are in **repositories** package. The **entities** package defines each class needed and provides the attributes, getters and setters for each one. The **dtos** are used in order to communicate with the fronted and these are used in order to separate what the user needs to see, from all the fields that are in the database.

Classes that are in each package:

- **controllers**:
  - `UserController,`
  - `QuestionController,`
  - `AnswerController,`
  - `TagController,`
  - `VoteController`
- **services**:
  - `UserService,`
  - `QuestionService,`
  - `AnswerService,`
  - `TagService,`
  - `VoteService`
- **repositories**:
  - `UserRepository,`
  - `QuestionRepository,`
  - `AnswerRepository,`
  - `TagRepository,`
  - `VoteRepository`
- **entities**:
  - `User,`
  - `Question,`
  - `Answer,`
  - `Tag,`
  - `TagQuestion,`
  - `Vote,`
  - `VoteAnswer,`
  - `VoteQuestion`
- **dtos**:
  - `UserBuilder,`
  - `QuestionBuilder,`
  - `AnswerBuilder,`
  - `AnswerDTO,`
  - `QuestionDTO,`
  - `UserDTO,`
  - `FilterDTO`

8. Bibliography

https://www.oreilly.com/library/view/software-architecture-patterns/9781491971437/ch01.html

https://thebadoc.com/ba-techniques/f/use-case-description-basics

https://www.google.com/search?q=layer+architecture&rlz=1C1CHBD_enRO868RO868&oq=laye&aqs=chrome.0.69i59j0i512j69i57j0i67j0i20i263i512j0i67j46i67j69i61.1302j0j7&sourceid=chrome&ie=UTF-8