MINISTERUL EDUCAȚIEI ȘI CERCETĂRII

UNIVERSITATEA TEHNICĂ
DIN CLUJ-NAPOCA

FACULTY OF AUTOMATION AND COMPUTER SCIENCE
COMPUTER SCIENCE DEPARTMENT

StackOverflow Web App (A3)

Student: Chichișan Ștefan

Group 30444

1. PROJECT SPECIFICATION

The StackOverflow web app is a simple version of the well-known Stack Overflow web app (see https://stackoverflow.com/). The system consists of an online platform where programmers may ask and answer questions. It contains questions and answers on a wide range of computer programming subjects. The website provides as a forum for users to ask and answer questions, as well as to vote questions and answers up or down.

A. BACK-END

2. ARCHITECTURAL DESIGN

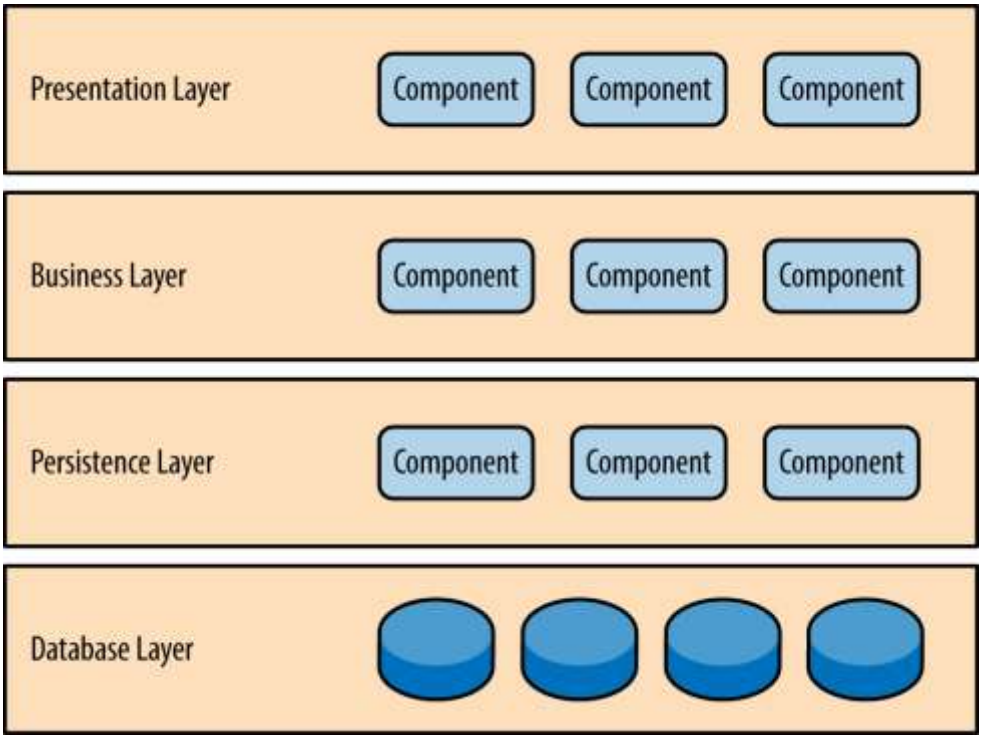


Figure.1

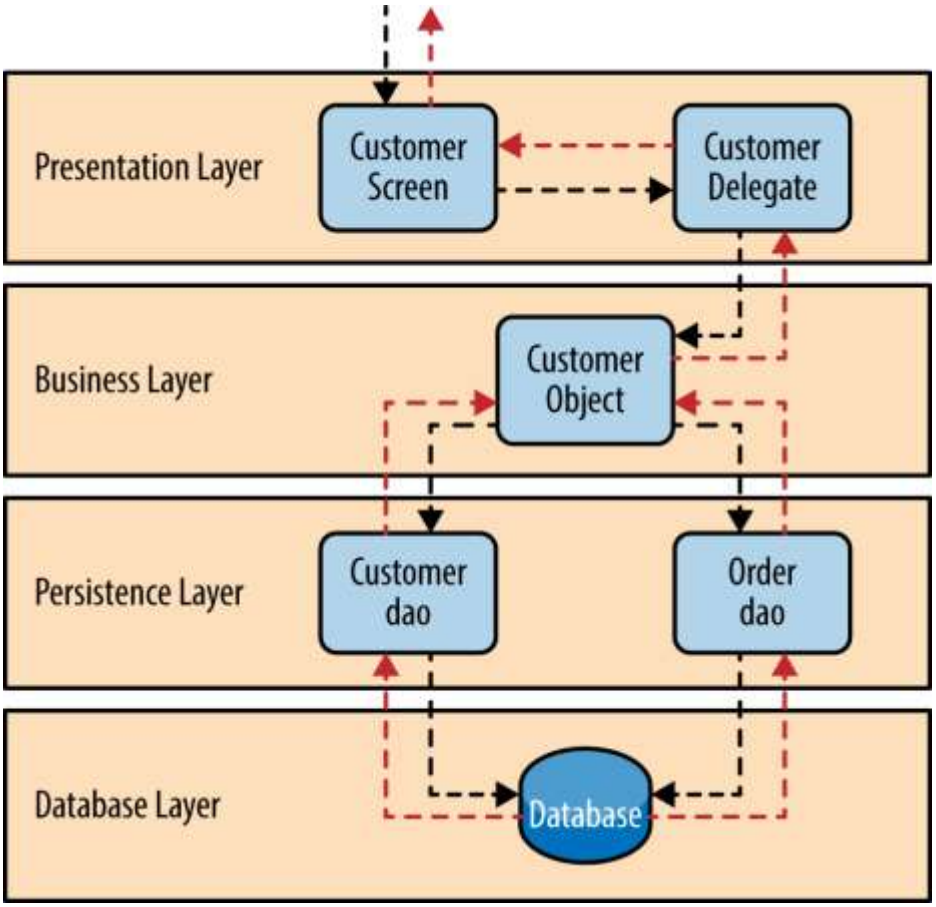
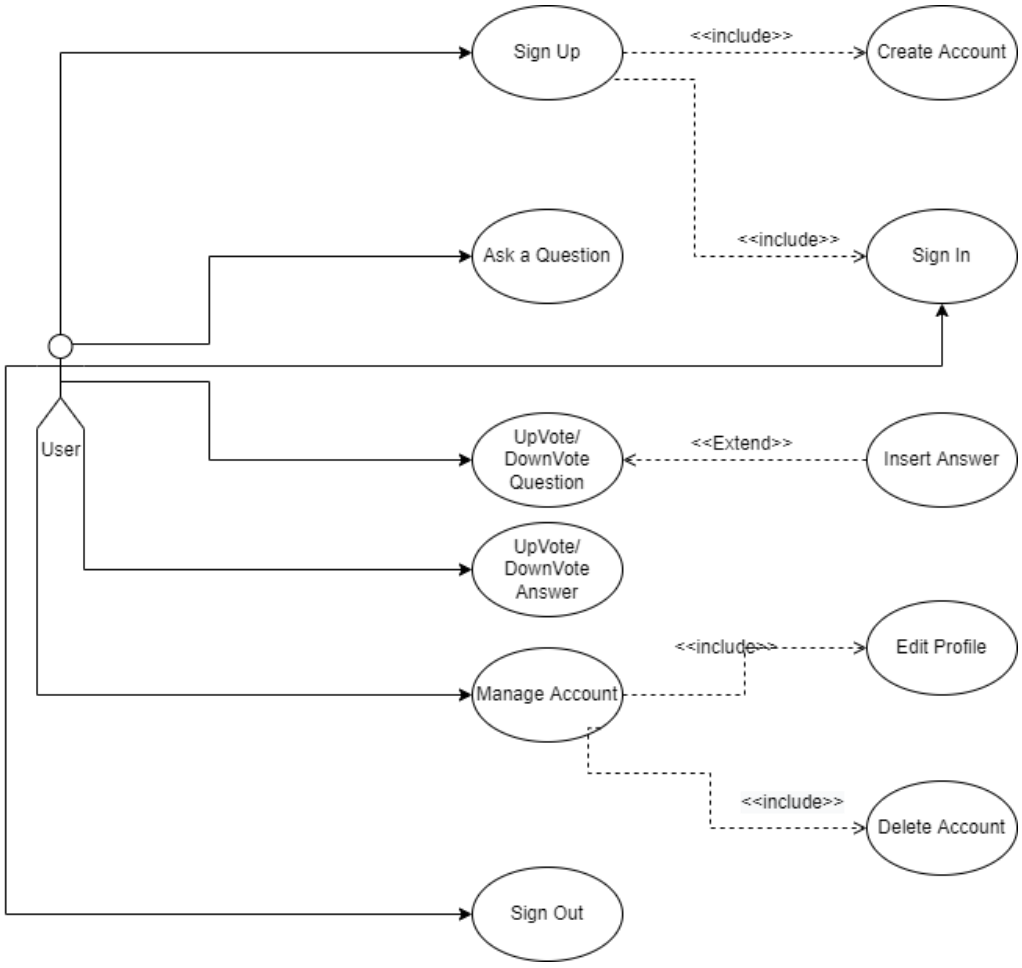


Figure.2

- The **PRESENTATION LAYER** is the program's user interface and communication layer, where the end user interacts with the application. Its primary function is to present information to and gather data from the user. This top-level layer, for example, can execute in a web browser, as a desktop program, or as a graphical user interface (GUI).
- The **BUSINESS LAYER** contains all of the business/domain logic, i.e. rules specific to the problem that the program was designed to solve. In short, it is where you tackle the problems your program was created to solve.

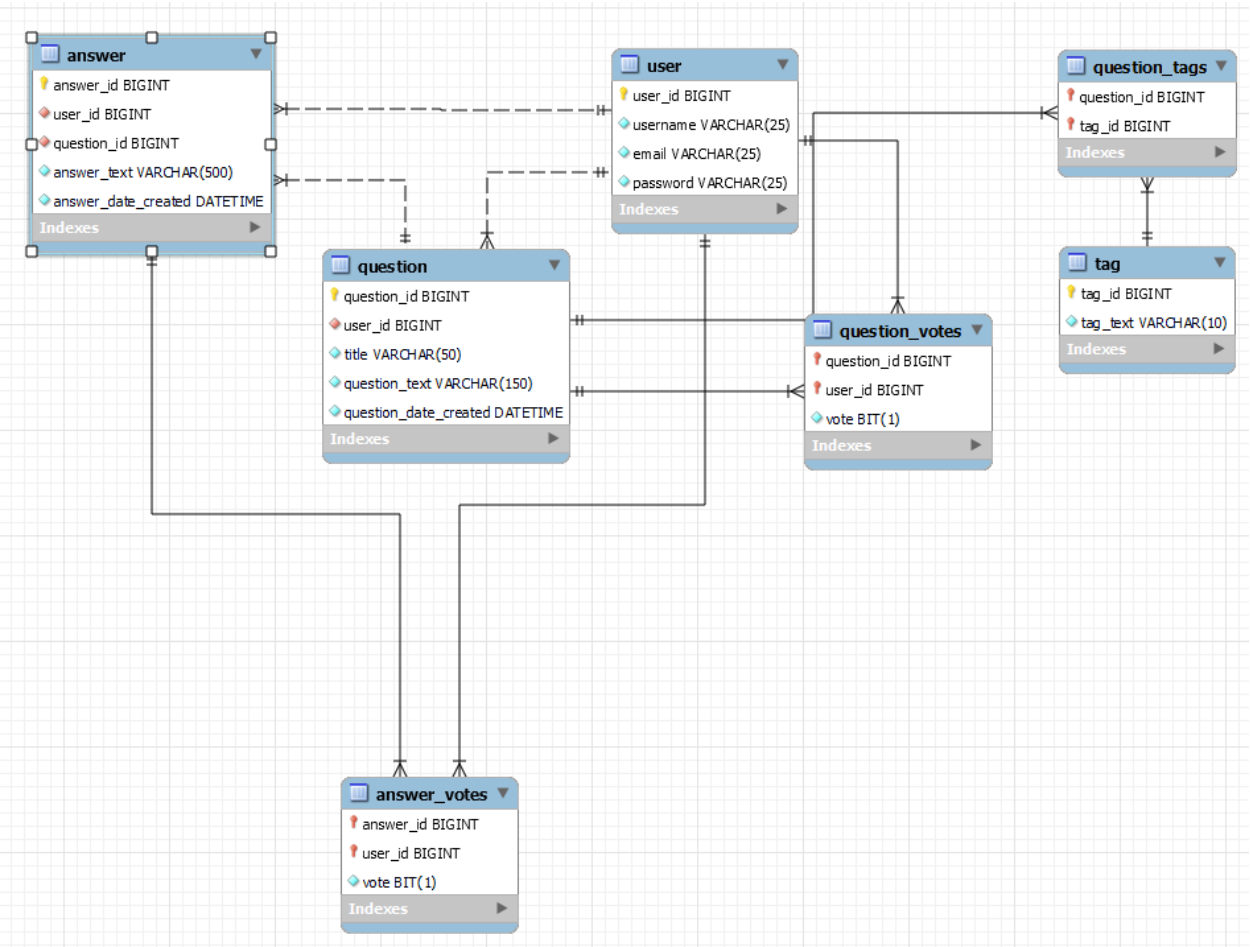
- The **PERSISTANCE LAYER** acts as a bridge between the application's business activities and the data it stores in a relational database. Because it maps objects to relational data, this persistence layer function is also known as object-relational mapping (ORM).
- The **DATABASE LAYER** is an application programming interface that connects a computer application to databases such as MySQL. Database abstraction layers decrease the amount of effort for the developer by providing a standard API and hiding the database intricacies behind this interface as much as feasible.

3. USE CASE

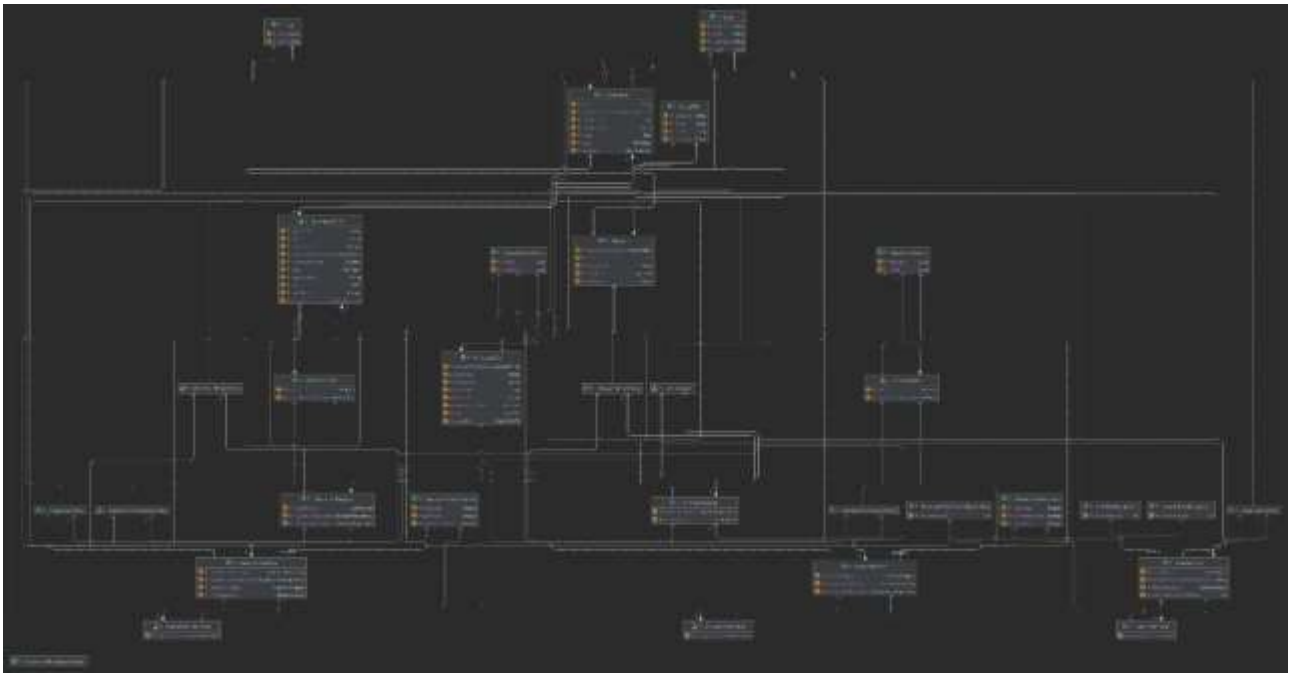


A user can sign up/log in into the web application. After logging in, he can scroll up or down throughout of the forum and search questions related to a specific problem. Then the user, can upvote/downvote a question or answer only once and add a comment to the question. Users will be able to log out whenever they want.

4. DATABASE DIAGRAM



5. CLASS DIAGRAM



6. ENDPOINTS

The application has several endpoints implemented, but only the most important ones were tested in Postman.

- *localhost:{port}/user/addUser (POST)*

```
{
    "username": "horhemicalut",
    "email": "george@com",
    "password": "risingevil99!"
}
```

- *localhost:{port}/users/getAllUsers (GET)*

- *localhost:{port}/questions/addQuestion/{userId} (POST)*

```
{
    "title": "Spring Boot crashes!",
    "questionText": "My Spring Boot application crashes when I run it
!",
    "user": {
        "userId": 2,
        "username": "horhemicalut",
        "email": "george@com",
        "password": "risingevil99!"
    },
    "tags": [
        {
            "text": "spring"
        }
    ]
}
```

- *localhost:{port}/questions/getQuestion/{id}/{userId} (GET)*

- *localhost:{port}/answers/addAnswer/{userId} (POST)*

```
{
    "answerText": "It crashes because you have a typo!",
    "user": {
        "userId": 3,
        "username": "bogdandavid",
        "email": "bogdi@com",
        "password": "djmogdicelmare99!"
    },
    "question": {
        "questionId": 3,
        "title": "Spring Boot crashes!",
        "questionText": "My Spring Boot application crashes when I ru
n it!",
        "questionDateCreated": "2022-03-14T22:06:02",
        "user": {
            "userId": 2,
            "username": "horhemicalut",
            "email": "george@com",
            "password": "risingevil99!"
        },
```

```
        "votesDTO": {
            "posVotes": 0,
            "negVotes": 0,
            "currentUserVote": null
        },
        "answers": [],
        "tags": [
            {
                "tagId": 5,
                "text": "spring"
            }
        ]
    }
}
```


- *localhost:{port}/answers/setAnswerVotes/{id}/{userId}?vote=<?> (POST)*
  e.g. ?vote=true

- *localhost:{port}/questions/setQuestionVotes/{id}/{userId}?vote=<?> (POST)*
  e.g. ?vote=true

All these requests are found and tested in Postman.

B. *FRONT-END*

7. FRONT-END ARCHITECTURE

Angular is a platform and framework that allows you to create single-page client apps in HTML and TypeScript. TypeScript is used to write Angular. It provides essential and optional functionality as a collection of TypeScript libraries that you can incorporate into your apps.

An Angular application's architecture is based on a few key ideas. The Angular framework's fundamental building pieces are Angular components arranged into NgModules. NgModules group together similar code to form functional sets; an Angular application is defined by a collection of NgModules. An application always contains at least one root module that facilitates bootstrapping, and many more feature modules are common.

- Components define views, which are collections of screen components that Angular may choose and alter based on the logic and data in your application.
- Components make use of services, which provide functionality that is not directly connected to views. Service providers may be injected as dependencies into components, making your code modular, reusable, and efficient.

Decorators are used by classes such as modules, components, and services. These decorators indicate their kind and include metadata that instructs Angular on how to utilize them.

- A component class's metadata ties it with a template that specifies a view. A template is a combination of standard HTML plus Angular directives and binding syntax that allow Angular to alter the HTML before producing it for display.
- The metadata for a service class contains the information required by Angular to make it available to components via dependency injection (DI).

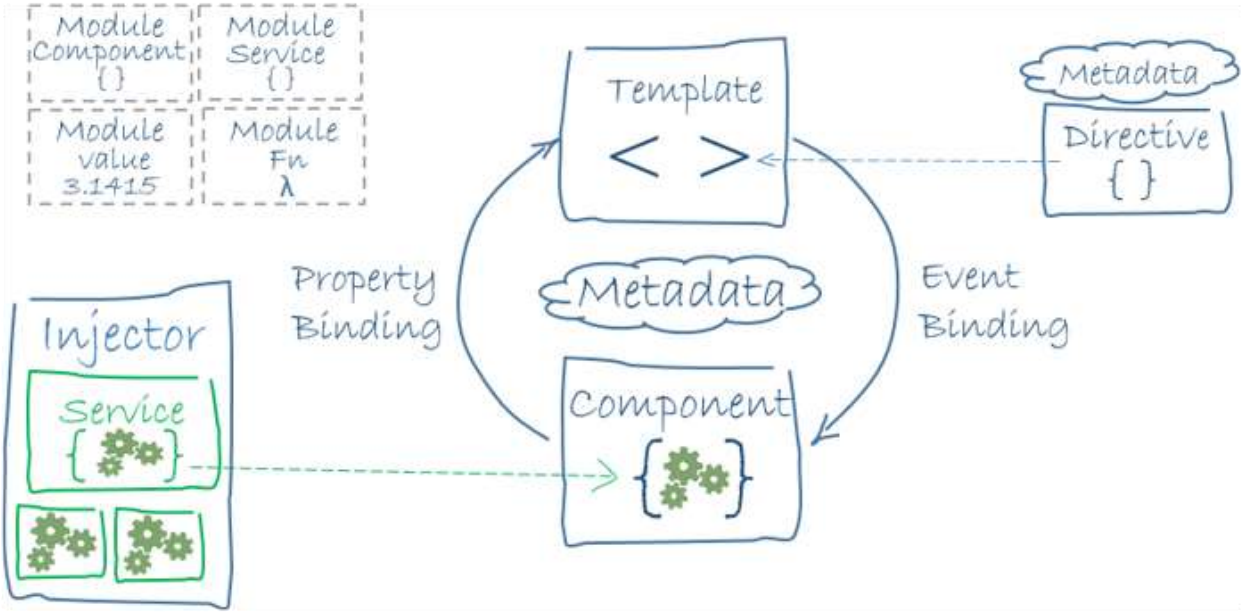


Figure.3

My front-end app in based on a 3-layer architectural style: Entity, Component and Service.

My entity decorator is based on the DTOs from the back-end. In my component decorator I created methods and properties which are used to bind with an UI (HTML in our case) of our application. In addition, the design part (SCSS files) is done here. In my service decorator I provide functionality that is not related directly to an UI. Basically, it is implemented the communication between back-end and the front-end. The controller classes from the back-end are replicated in the front-end, also calling the back-end endpoint which is *localhost:8080.* Moreover, the authentication and registration functionality are done here.

In simple words the behavior of the system: front-end server is on *localhost:4200* and back-end server is on *localhost:8080.* The front-end is calling the *localhost:8080* URLs, process the data coming from the back-end and it displays on the front-end.

8. DESIGN CONSTRAINTS

The application is not operating system dependent. It is necessary to have Java SDK and an SQL Platform to develop. For accessing it on world wide web, you must have a web browser installed. To connect to the database, we need to use HIBERNATE

and Java Persistence API (JPA). For backend Java Spring Boot is used. The database is designed using MySQL WorkBench. The application is designed in IntelliJ. For testing JUnit and Mockito. JUnit is the Java library used to write tests (offers support for running tests and different extra helpers - like setup and teardown methods, test sets etc.). Mockito is a library that enables writing tests using the mocking approach. For the front-end part it is used Angular based on Typescript.

9. BIBLIOGRAPHY

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

[2] https://towardsdatascience.com/software-architecture-patterns-98043af8028

[3] https://www.baeldung.com/spring-boot

[4] https://spring.io/projects/spring-boot

[5] https://spring.io/projects/spring-data-jpa

[6] https://www.baeldung.com/the-persistence-layer-with-spring-and-jpa

[7] https://projectlombok.org/

[8] https://www.vogella.com/tutorials/Mockito/article.html

[9] https://www.baeldung.com/mockito-annotations

[10] https://medium.com/@gara.mohamed/java-idioms-of-the-mapper-pattern-f7527827ac98

[11] https://stackoverflow.com/questions/1051182/what-is-a-data-transfer-object-dto

[12] https://www.baeldung.com/entity-to-and-from-dto-for-a-java-spring-application

[13] https://www.javatpoint.com/types-of-relationship-in-database-table

[14] https://www.baeldung.com/spring-data-jpa-query