**Stack Overflow**

Student: Presecan Alexandru

Group: 30434

Teaching assistant: Bindea Bogdan

1. **Introduction**

Stack Overflow is a well known website among programmers, used mainly to ask questions and receive answers for whatever difficulty the programmer might encounter. This project is a simplified version of Stack Overflow, preserving just the main functionalities such as posting questions and answers, tagging the questions, and an up/down vote system that affects the users’ score.

1. **Tech Stack**

Backend: ASP.NET

Frontend: Angular

Database: SQL Server

1. **Endpoints**

**ApplicationUsers**

/api/ApplicationUsers - GET

/api/ApplicationUsers/{id} - GET

/api/ApplicationUsers – POST

{

    "email": "testuser@yahoo.com",

    "userName": "TestUser",

    "password": "Password123!"

}

/api/ApplicationUsers/{id} – PUT

{

    "email": "testusernew@yahoo.com",

    "userName": "TestUserNew",

    "password": "Password123!",

"newPassword": "Password123!New"

}

/api/ApplicationUsers/{id} - DELETE

**Questions**

/api/Questions - GET

/api/Questions/{id} - GET

/api/Questions – POST

{

    "authorId": "authorId",

    "title": "test",

    "text": "test",

    "tags": ["tag1", "tag2", “tag3”]

}

/api/Questions/{id} – PUT

{

    "title": "testNew",

    "text": "testNew"

}

/api/Questions/{id} - DELETE

**Answers**

/api/Answers - GET

/api/Answers/{id} - GET

/api/Answers – POST

{

    "authorId": "authorId",

"questionId": "questionId",

    "text": "test"

}

/api/Answers/{id} – PUT

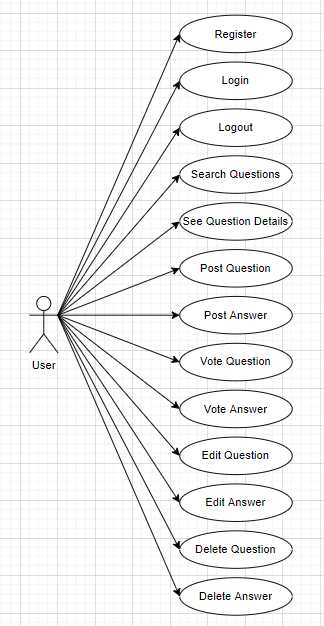
{

    "text": "testNew"

}

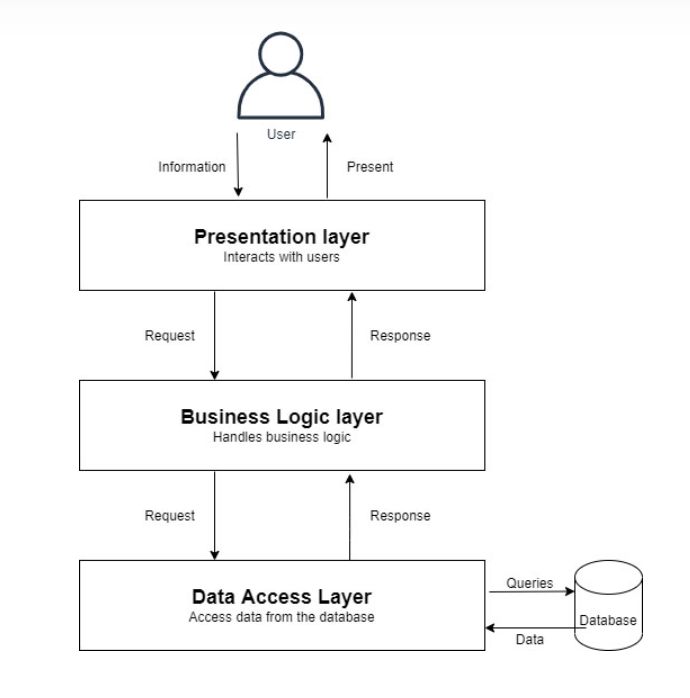
/api/Answers/{id} - DELETE

1. **Use cases diagrams**



User posts answer Case:

The user registers if not already registered, logs in, searches through questions and selects one that he/she is able to answer. He/She completes the answer and posts it. The user will gain or lose score points if other users vote his/her answer.

1. **Architecture**

Presentation Layer:

* Represents the client application (Angular SPA Application)
* Acts as an user interface
* Allows user interactions

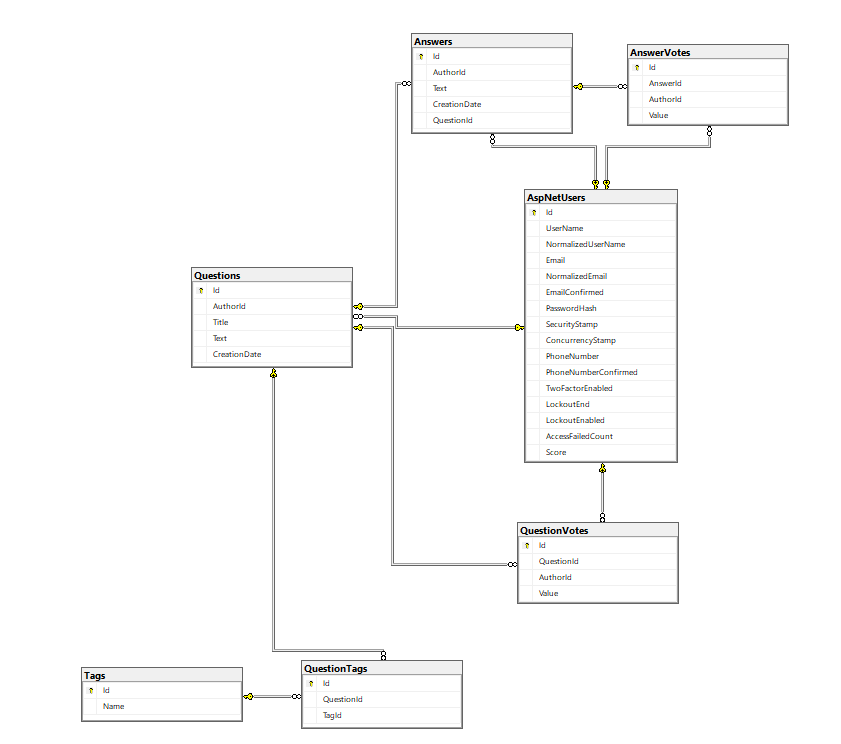
Business Logic Layer:

* Contains all the business logic of the application
* Composed out of API controllers and services

Data Access Layer:

* Used to access the database and perform transactions
* Most data access functionality is provided by the Entity Framework

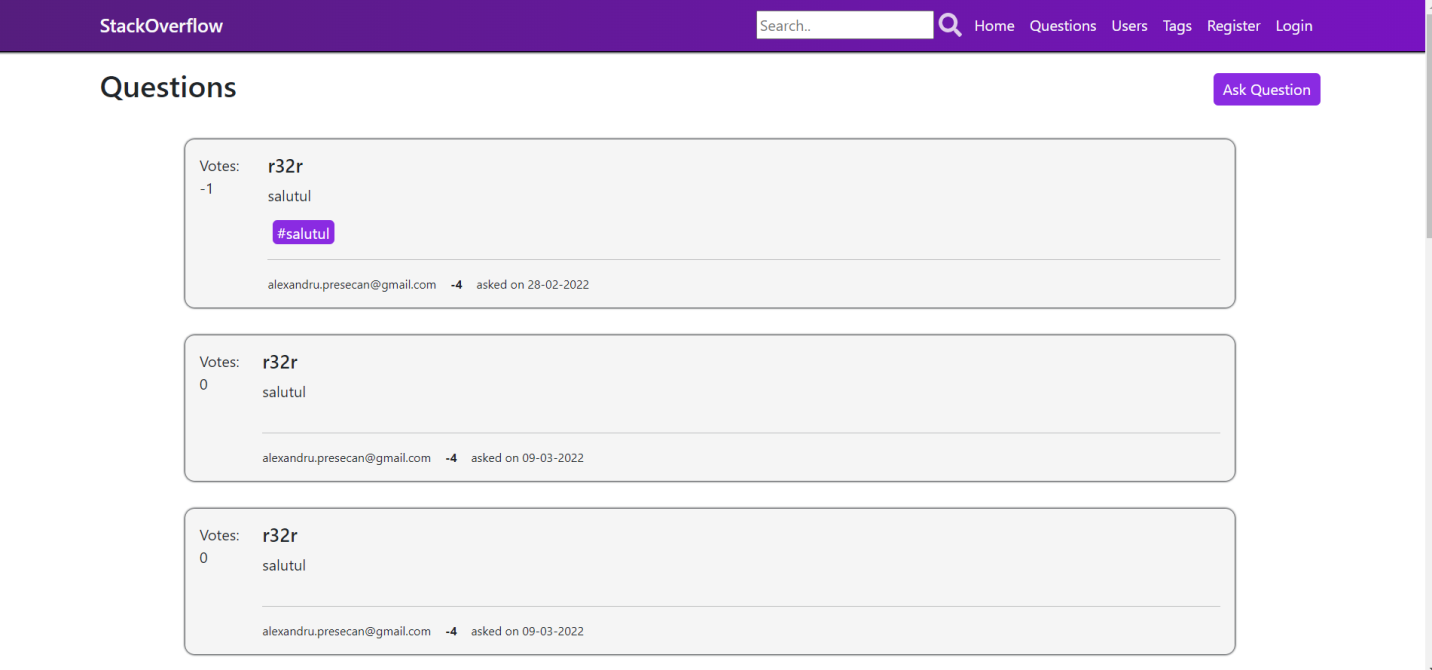
1. **Database Diagram**

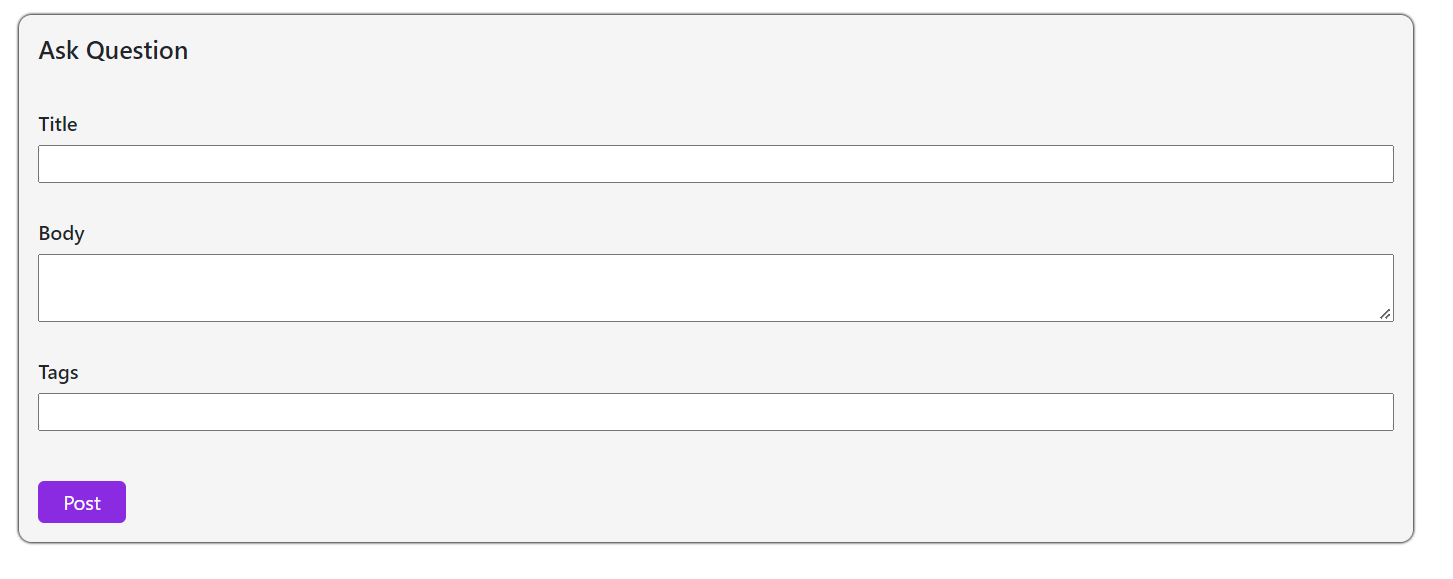
****

1. **Web Pages**

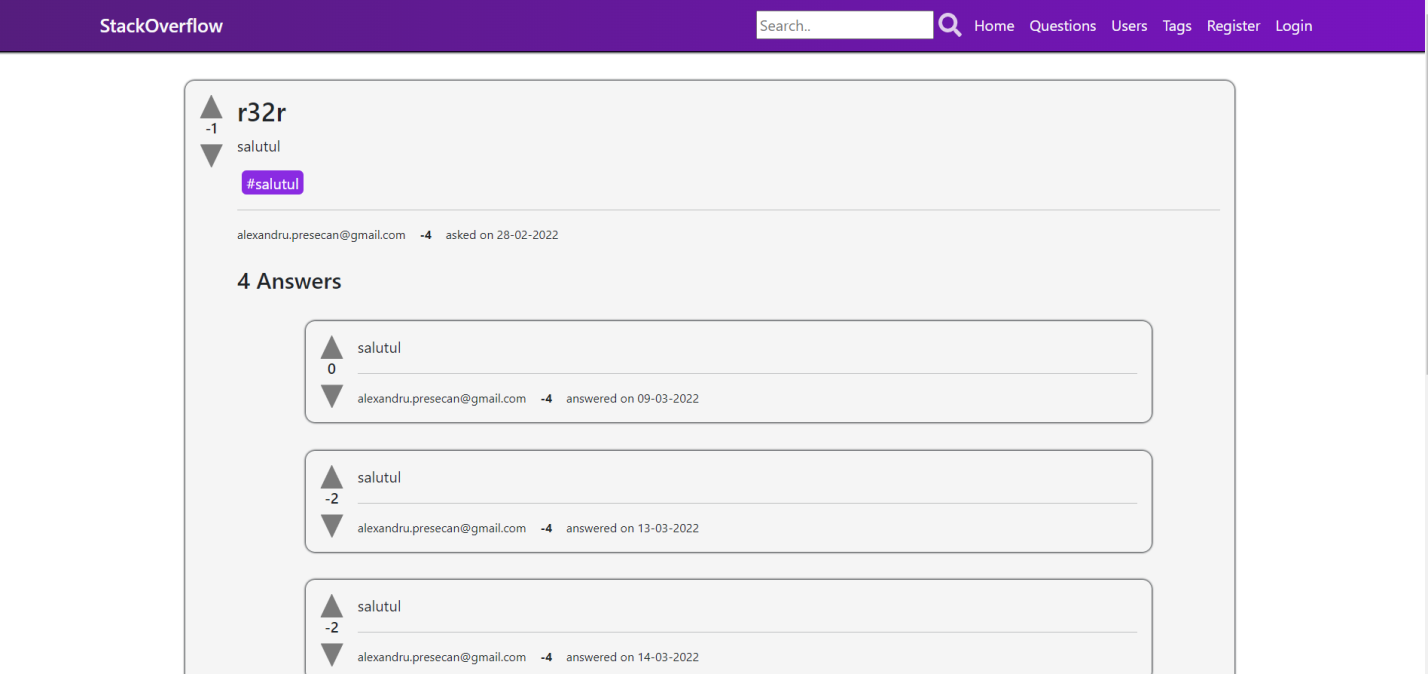
**Questions Page**

All the questions and some of their details (votes, title, text, tags, author and creation date) are displayed on this page. Clicking one question will redirect us to the selected question’s page. We also have the option to ask a new question.





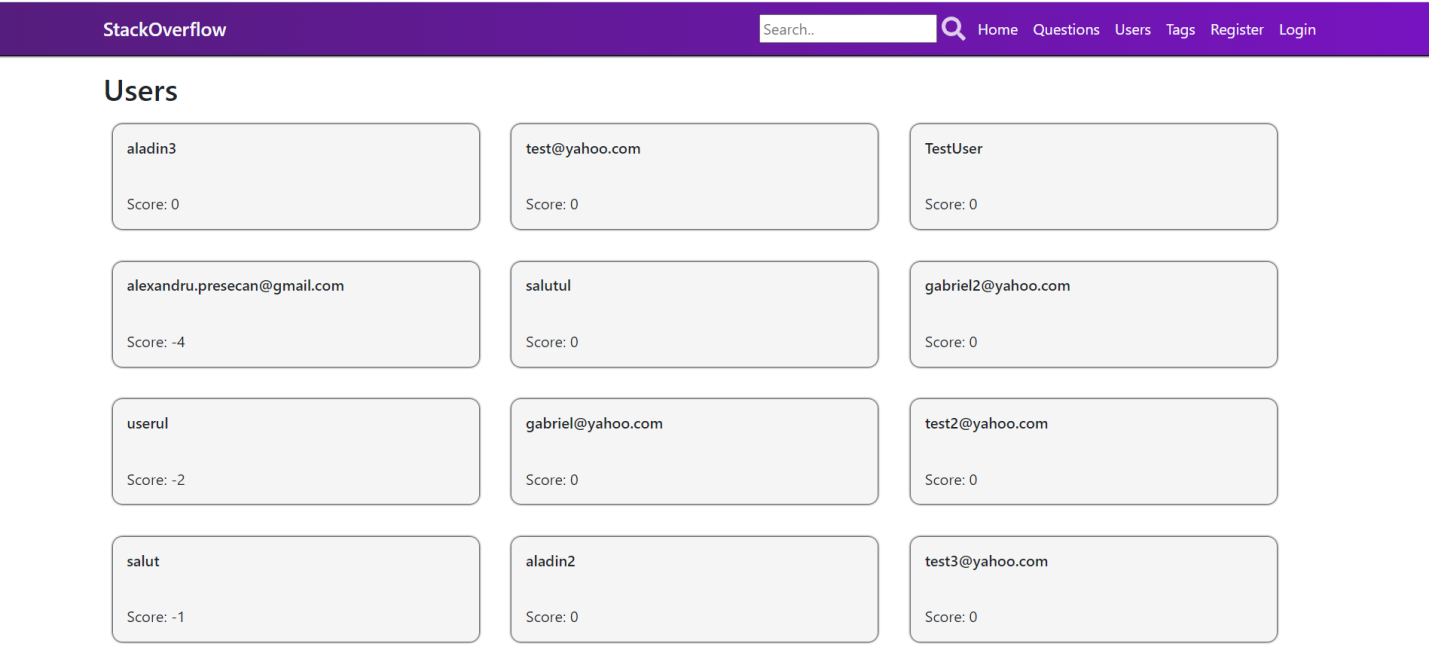
**Question Page**

The question page contains all the answers sorted by their vote count and allows the user to vote the question or the answers. An option to post an answer also exists.

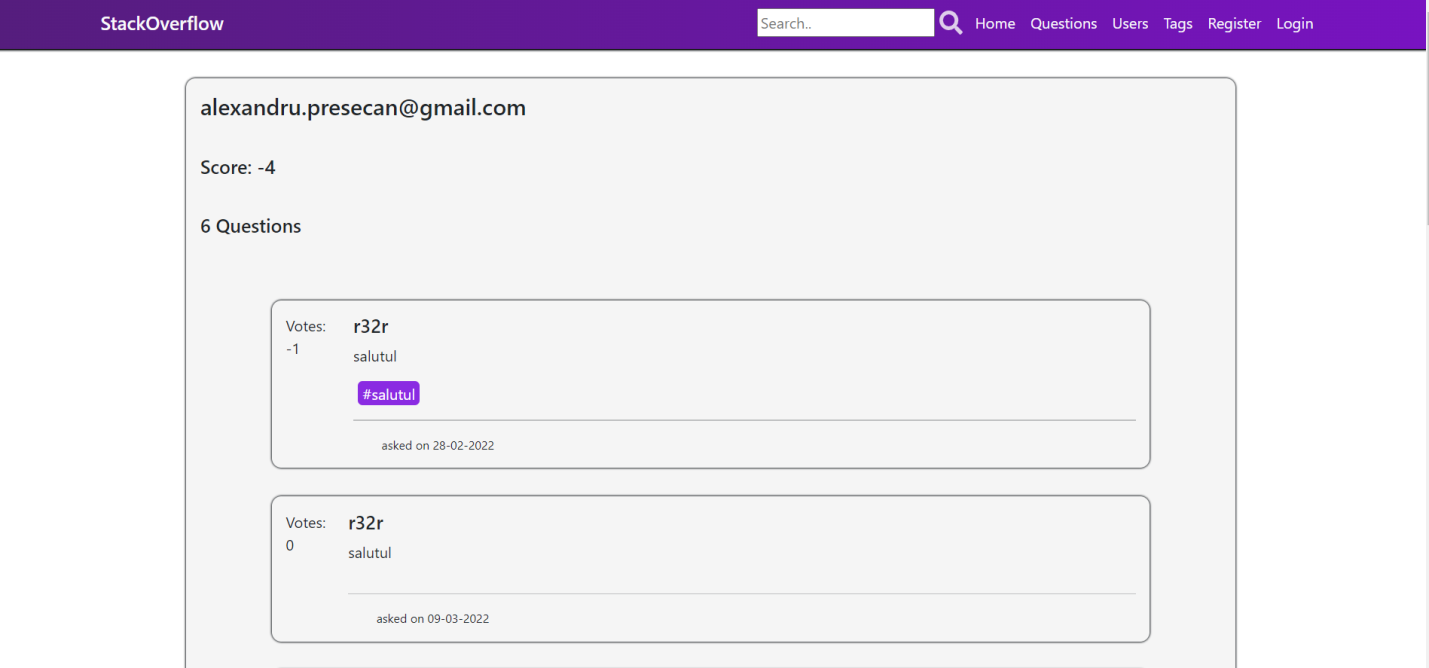


**Users Page**

All the users and their scores are displayed here. Clicking one user will redirect us to see more details about that user.

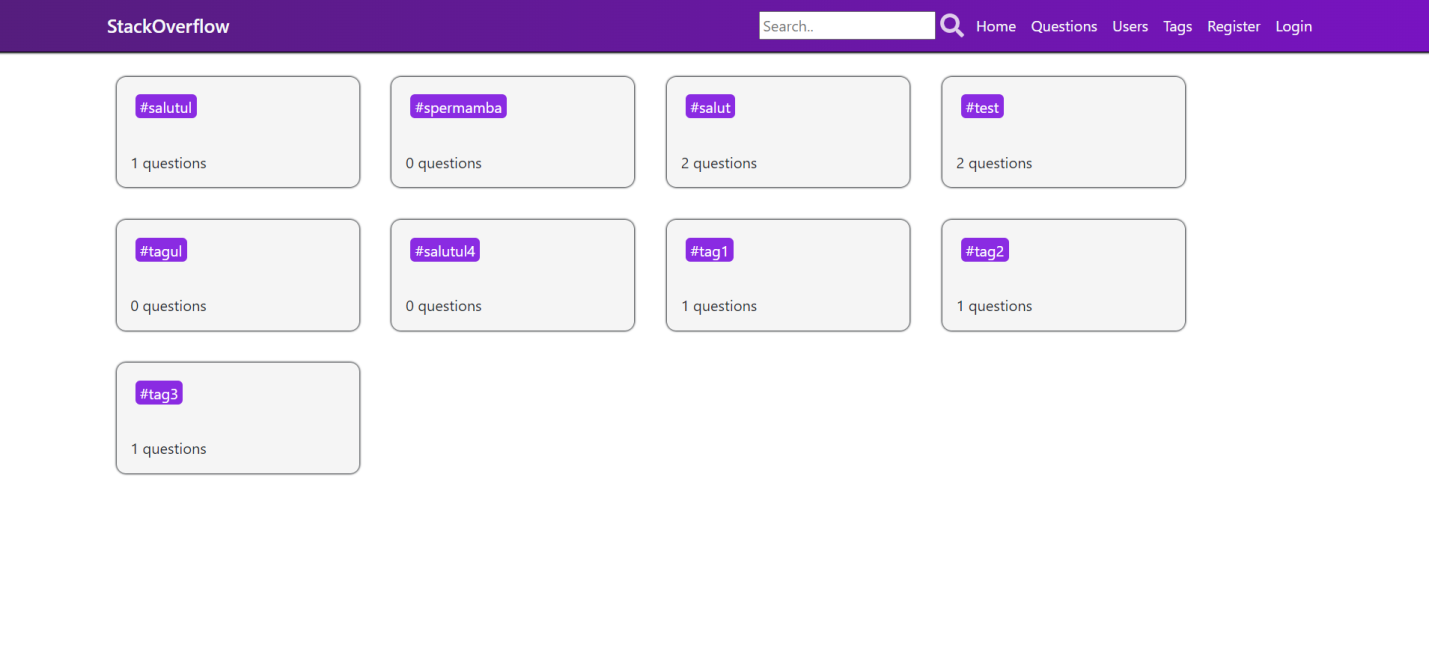


**User Page**

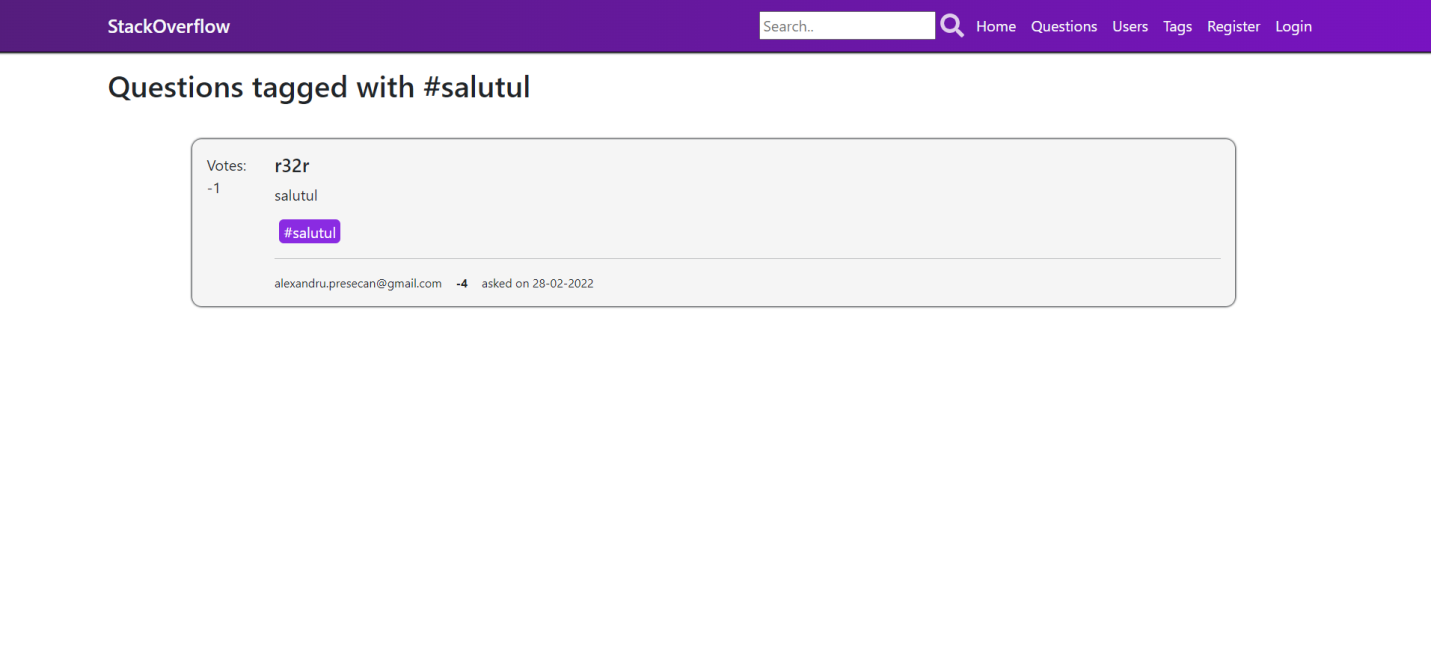
All the questions and the answers posted by the user are displayed here. We will get redirect to the question/answer on click.

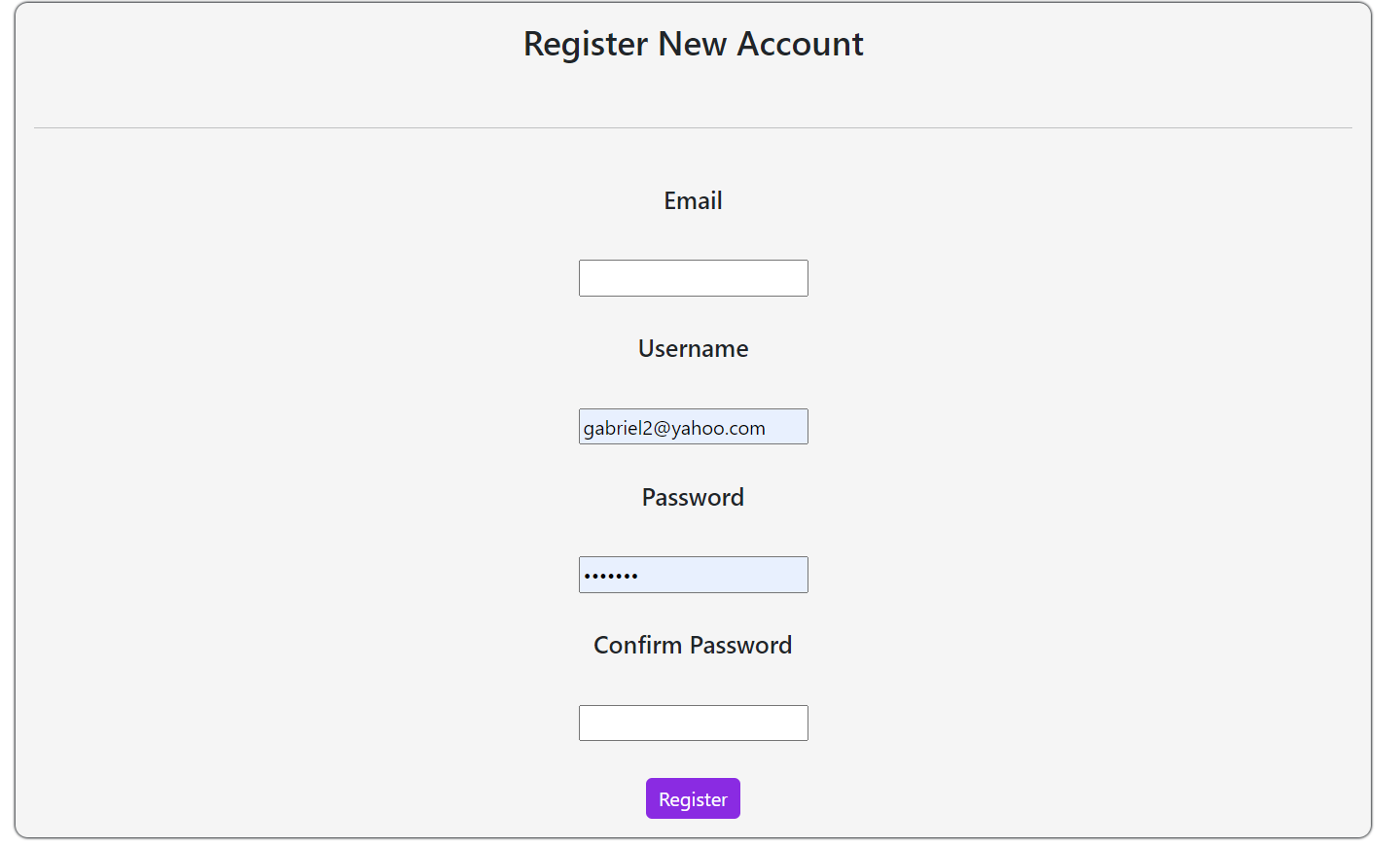
**Tags Page**

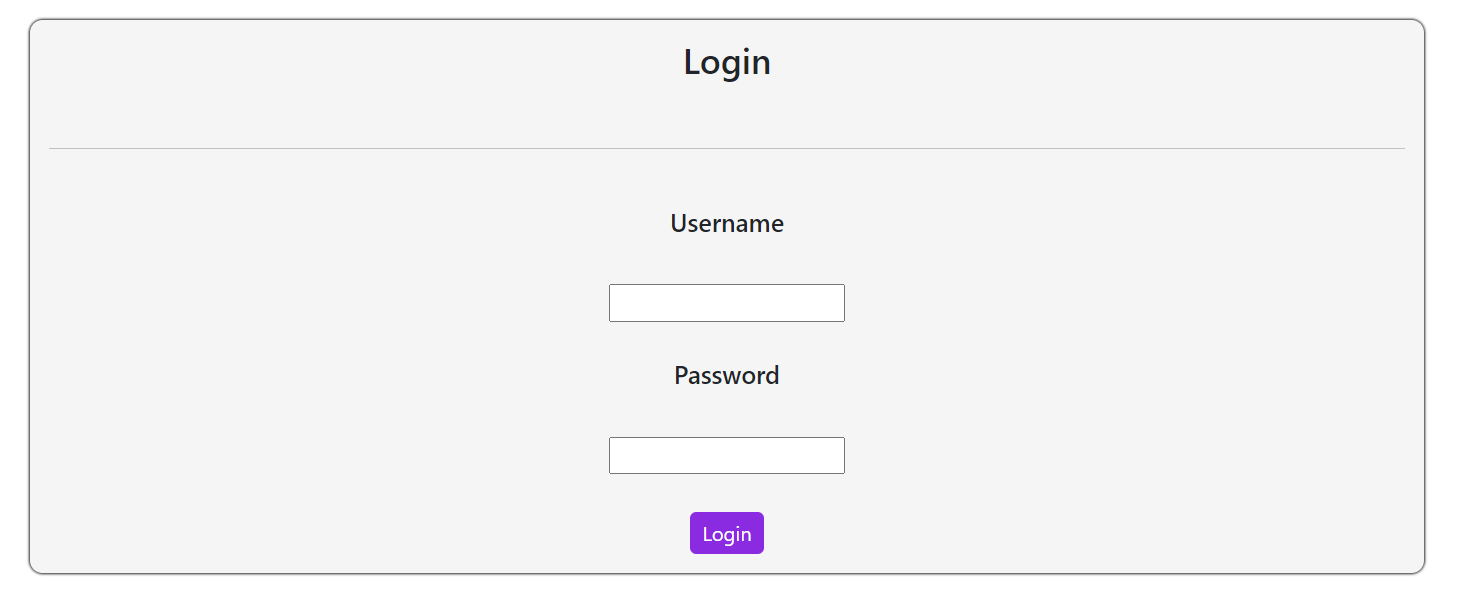
All the tags and the number of questions for each tag are displayed here. Clicking one tag will redirect to the page with questions having that tag.

**Tagged Questions Page**

Questions having a specific tag are displayed here. Clicking one question will let us see more details.



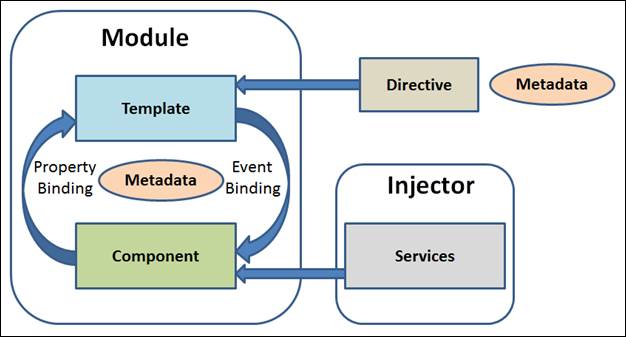
**Register Page**

****The required fields for registering a new account are email, username, password and the password confirmation.

**Login Page**

The required fields for logging in are the username and the password.

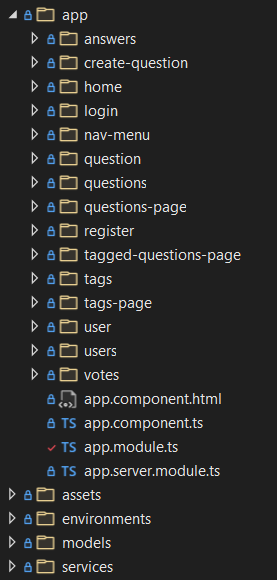
1. **Front-End Architecture**



Modules are composed out of templates (html for the web page/component), styling (css), and components (typescript files describing the behavior of the module). The modules use the property and event binding in order to communicate between the template and the component.

Services can be injected in the components, allowing them to communicate with the backend or perform other logic that the injected service provides.

1. **Front-End Implementation**



Angular allows reusability of the code by using components and also provides the developer with a clean project structure.

Structure of the project:

* app: contains the components
* assets: contains icons/images
* environments: contains constants
* models: class models used to map entities received from backend
* services: used to interact with the backend

1. **Front-End Testing**

Testing was done for the following services: Tags Service, Questions Service, Answers Service, Users Service. These tests check the retrieving of data from the backend.

1. **Bibliography**

<https://angular.io/>

<https://stackoverflow.com/>

<https://docs.microsoft.com/en-us/aspnet>

<https://single-spa.js.org/docs/ecosystem-angular>