**SpanacOverflow**

-Web Application-

Nemes Mihnea-Andrei

Group 30434

**Description**

This project will be a simple implementation of the StackOverflow website. The websites provides users the functionality of asking and answering questions. Users will be able to create their own account after which they will be able to ask multiple questions and provide answers to questions. They will also be able to vote on questions and answers based on how useful they perceive the answer/question in question to be. At the moment, the CRUD operations have been implemented, meaning users can be created and manipulated, questions can be created using one or more tags, answers can be created and linked to a question. Even more, the functionality of voting on questions and answers has also been added. When trying to add a new question with a tag that doesn’t exist so far, the application will add the newly used tags in the database.

**Use Case Model**

1. Use case: Ask a question
   * Primary actor: User
   * Main scenario: The user will open the application and will be redirected to the login page. In the case that the user doesn’t already have an account, a button will be present which when clicked will take the user to the register page. After a successful login, the user will be redirected to the questions page, where a button will be provided which when pressed will present a form where the user can fill in the title, body and tags of the questions that they want to ask. After they post it, they will be able to see who posts an answer to their question.
2. Use case: Answer a question
   * Primary actor: User
   * Main scenario: As with the “Ask a question” use case, the user will go through the login process and after which will arrive on the questions page. Here they can filter the questions based on a tag or by a title filter, or just choose a question from the main list. After they have chosen a question, they will be redirected to a page containing the chosen question, the answers that have already been posted for that question, and a textbox that will allow the user to write their answer to the question. After they post the answer, the other users will be able to see their response and vote on it.

**Class Diagram**

**O imagine care conține text, negru, interior

Descriere generată automat**

Fig1. Model Package Class Diagram

O imagine care conține text, placă, captură de ecran, tabelă de marcaj

Descriere generată automat

Fig2. Controller Package Class Diagram

O imagine care conține text, metal, captură de ecran, argintiu

Descriere generată automat

Fig3. Repository Package Class Diagram

O imagine care conține text, negru, metal, placă

Descriere generată automat

Fig4. Service Package Class Diagram

**Database design**

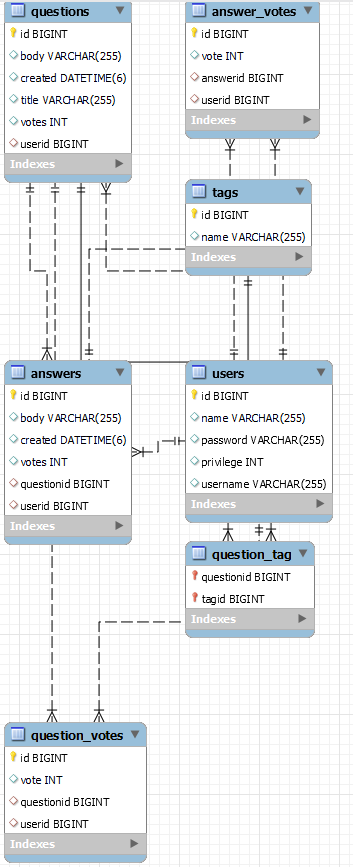


Fig5. Database Design

**Architecture**

In my application I will be using the Layered Architecture, which includes: the Presentation Layer, the Application Layer, the Business Layer and the Database Layer. The Presentation Layer is the UI layer, the view of the application that the user will see when operating the application. The Business Layer handles the logic that is used to serve different actions in the application. The Application Layer is the general controller of the application and serves as the pattern for the application that ensures that the fundamental principles of OOP are followed, like low coupling and high cohesion. Finally, the Database Layer handles the access to the repository, in this case the access to the mySQL database and includes all the services that are needed to support this connection for the business layer.

The fundamental idea of the Layered Architecture is that important parts of the application are separated. This means that the presentation, application processing and data management functionalities are separate from one another. This means that a request from one layer must traverse all the layers below it in order to reach their end point. A concrete example would be a request to create a new movie which is made in the presentation layer. It will pass through the application and business layer before finally reaching the database layer where the data will be processed. The Layered Architecture is the preferred architecture when building web applications that use CRUD operations on the data.

**Endpoints**

1. User(“**/users**”)
   * “**/getAll**” – No body, returns all users in the database.
   * “**/save**” – Body is a **user**, containing **name, username** and **password.**
   * “**/update**” – Query Params **userid** and JSON containing the new **name, username** and **password.**
   * “**/delete**” – Query Params has the **userid**, the id of the user to be deleted.
2. Question(“**/questions**”)
   * “**/getAll**”– No body, returns all questions in the database.
   * “**/save**” – Query Params is **tags**, the list of tags for the question, **userid**, and a body **question** as JSON which contains the **title** and **body.**
   * “**/filterTag**” – Query Params is a **tag**, will return the list of questions with that tag.
   * “**/filterTitle**” - Query Params is a **title**, will return the list of questions whose title contains that word.
   * “**/vote**” - Query Params is **questionid**, the id of the questions that is to be voted on, **userid**, the id of the user that performs the vote, and **vote** either 1/-1 depending on the vote being a like or a dislike.
   * “**/delete**” – Query Params is **questionid**, the id of the question that is to be deleted.
3. Answer(“**/answers**”)
   * “**/getAll**” – No body, returns all answers in the database.
   * “**/save**” – Query Params contains **questionid**, the id of the question that is answered, **userid**, the id of the user answering, and a text in **body** that represents the body of the answer.
   * “**/update**” – Query Params contains **answerid**, the id of the answer that is to be updated, and in **body**, the text that will replace the current body of the answer.
   * “**/delete**” – Query Params contains **answerid**, the id of the answer that is to be deleted.
   * “**/vote**” – Query Params contains **answerid**, the id of the answer that is to be voted on, **userid**, the id of the user that performs the vote, and **vote** a value either 1/-1 depending on the vote being a like or dislike.
4. Tag(“**/tags**”)
   * “**/getAll**” – No body, returns all tags in the database.
   * “**/save**” – Body is a **tag**, containing **name.**
   * “**/update**” – Query Params **tagid** and the body contains the new **name** of tag.
   * “**/delete**” – Query Params has the **tagid**, the id of the tag to be deleted.