StackOverflow web application

Student: Bucur Alexandra

Group: 30434

2021-2022

# Short description

This web application will be a copy of the official StackOverflow website. The app be used in order to ask and answer questions. Customers will be able to create accounts and ask or answer several question. Each user can add tags to the question, in order the questions to be somehow “divided into sections”. The questions can later be voted and the one with the most votes will be displayed first. Each question can have several answers from users, or the author himself. In order to see the most relevant answers, they may be voted also. Questions and answers can be updated (changed) by the author. At this point the CRUD operations with the database were implemented. The web app will later contain a user interface that will facilitate the usage.

The technologies that were used: MySQL Workbench for the database management; Java and Spring for implementation and Angular for the frontend. The testing was done with Postman.

# Use-case model

*Use case: Create a question*

*Primary actor: Client*

*Main success scenario: The client will see on the main page of the application a button to get into the account. If the client has no account, he shall create one. After getting into the account, a special button will be displayed. The button will transfer the user to the page where he will be able to create a new question. After inserting all the necessary data, such as title and the body of the desired question, the user will be able to post it in order to the community to see it.*

*Extensions: Of the user is not logged in, or the login fails due to invalid credentials, the client will not be able to create a question.*

*Use case: Create an answer*

*Primary actor: Client*

*Main success scenario: The client will see on the main page of the application a button to get into the account. If the client has no account, he shall create one. After getting into the account, a special button will be displayed. The page with al the questions will be displayed. After selecting one, the user will be able to respond to the desired question.*

*Extensions: Of the user is not logged in, or the login fails due to invalid credentials, the client will not be able to create an answer.*

*Use case: Update a question*

*Primary actor: Client*

*Main success scenario: The client will see on the main page of the application a button to get into the account. If the client has no account, he shall create one. After getting into the account, a special button will be displayed. The button will transfer the user to the page where he will be able to update a question. After inserting all the necessary data, such as title and the body of the desired question, the user will be able to post it in order to the community to see it modified.*

*Extensions: Of the user is not logged in, or the login fails due to invalid credentials, the client will not be able to create a question.*

*Use case: Vote a question*

*Primary actor: Client*

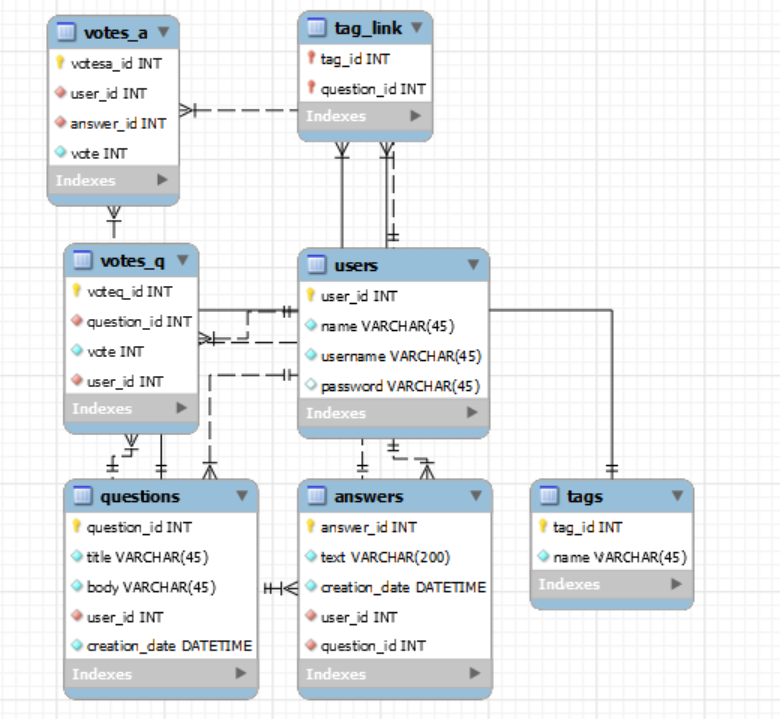
*Main success scenario: The client will see on the main page of the application a button to get into the account. If the client has no account, he shall create one. After getting into the account, a special button will be displayed. The page with al the questions will be displayed. After selecting one, the user will be able to vote the selected question, either with a like or dislike. The vote will be taken into consideration having in mind the final vote.*

*Extensions: Of the user is not logged in, or the login fails due to invalid credentials, the client will not be able to create an answer.*

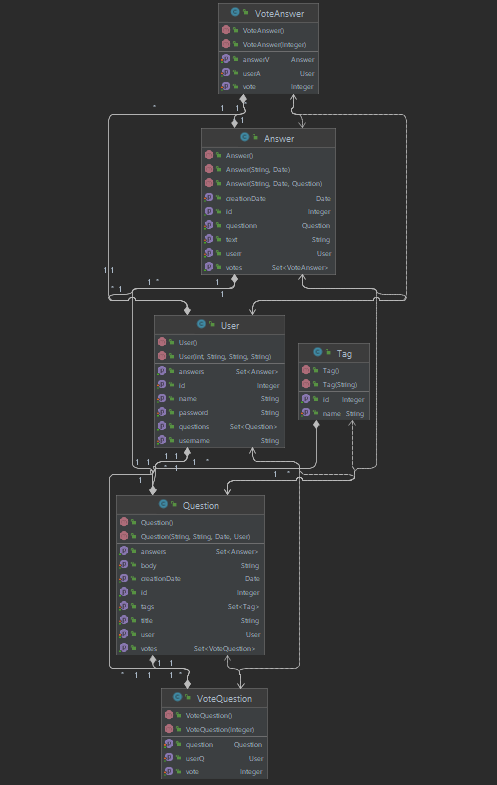
*Extensions: Of the user is not logged in, or the login fails due to invalid credentials, the client will not be able to create a question.*

# Database design

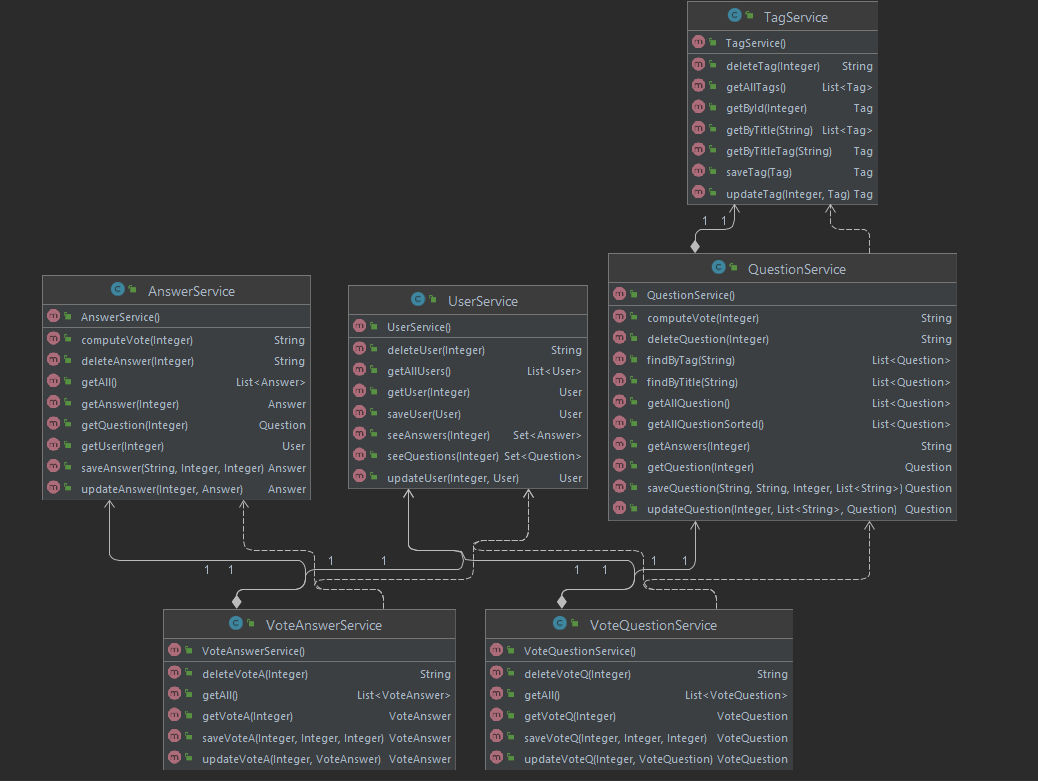
The database of the application is presented below. The database will pe updated when the bonus features will be implemented.



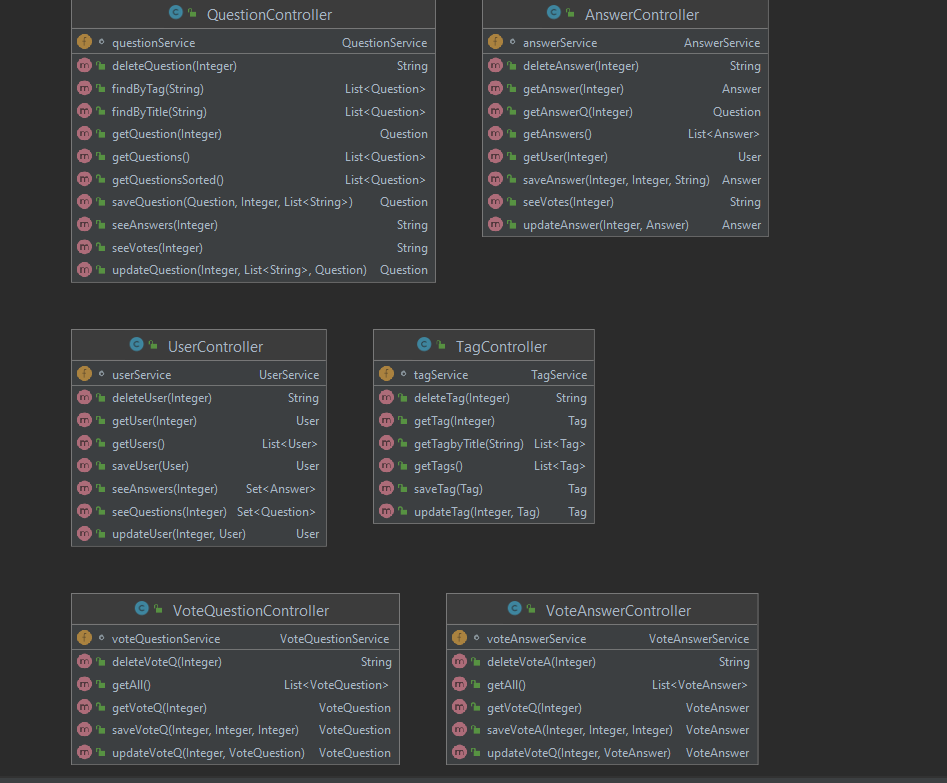
# Class diagram

Model package

Service package



Controller package



# Code explained

In order to have a better understanding of the code, I explained the methods oh how to be called in postman -> the methods with a requestBody

**User**

saveUser – requires a body of type user (id,name,username,password)

the rest are based on a parameter (id)

**Question**

saveQuestion requires a body(title, body)

the rest are based on parameters(id or title(string))

**Answer**

saveAnswer requires a body of type answer

**Vote**

saveVote requires a body of type vote

# Architecture

## Layered Architecture

This type of architecture has the following main parts:

* The web layer is the uppermost layer of a web application. It is responsible of processing user’s input and returning the correct response back to the user. The web layer must also handle the exceptions thrown by the other layers. Because the web layer is the entry point of our application, it must take care of authentication and act as a first line of defense against unauthorized users.
* The service layer resides below the web layer. It acts as a transaction boundary and contains both application and infrastructure services. The application services provides the public API of the service layer. They also act as a transaction boundary and are responsible of authorization. The infrastructure services contain the "plumbing code" that communicates with external resources such as file systems, databases, or email servers. Often these methods are used by more than a one application service.
* The repository layer is the lowest layer of a web application. It is responsible of communicating with the used data storage

|  |  |
| --- | --- |
|  | Layered architecture uses the idea of a closed layer, which means that as a request moves from layer to layer, it must go through the layer right below it to get to the next layer below that one. For example, a request originating from the presentation layer must first go through the business layer and then to the persistence layer before finally hitting the database layer.  In order to have a better understanding, let us compare the controller in an MVC model and in a layered arch. The mvc controller have all the logics it needs in order to perform certain operations, for example if we want to perform the sum of 2 numbers, the mvc controller will contain all the logics needed to perform such an operation. |

As opposed to the layered architecture controller which will only contain an entry point into the application. The endpoints are filtered and they get to the service where the logics is implemented. This controller will only call the need to perform the sum of 2 numbers and the service will actually perform the operation. One of the most important aspect of this type of architectures is the fact that one change does not change all the layers.

# Bibliography

* <https://medium.com/@udith.indrakantha/issue-related-with-infinite-recursive-fetching-of-data-from-relationships-between-entity-classes-ffc5fac6c816>
* <https://site.mockito.org/>
* <https://thorben-janssen.com/ultimate-guide-association-mappings-jpa-hibernate/>
* <https://www.tutorialspoint.com/jpa/jpa_entity_managers.htm>
* <https://www.tutorialspoint.com/design_pattern/data_access_object_pattern.htm>
* <https://www.tutorialspoint.com/hibernate/hibernate_configuration.htm>
* https://www.jetbrains.com/help/idea/class-diagram.html