**Documentation assignment 3**

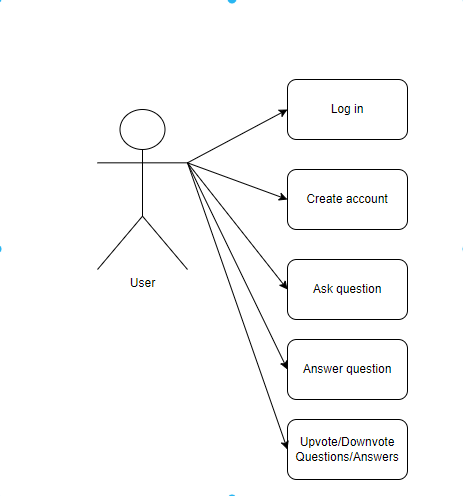
Name: Marcu Horea-Radu

Group: 30433

**1. Introduction**

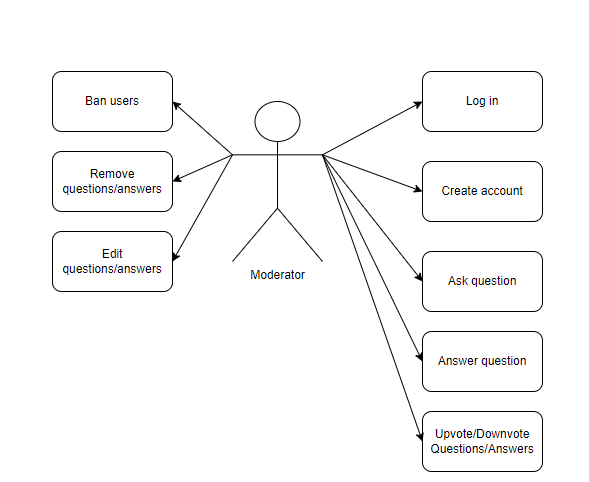
This assignment will be an implementation of a simple version of Stack Overflow. This application will have two types of users, regarding their roles: the normal user and the moderators. For a person to access the application they will have to be connected to their account, not being able to do anything without being logged in. The users will be able to ask questions, answer them, and upvote or downvote both the answers and the questions. The moderators will be able to do all the things that a normal user can do, but also remove questions and answers, edit them or even ban different users if they have bad behavior.

**2. Technology**

The IDE which I am going to use for this assignment is IntelliJ. As for the technologies, I am going to use Java Swing, SQL. Java Swing provides lightweight and platform-independent components, making it suitable and efficient in designing and developing desktop-based applications (systems).

**3. Use case diagrams**

A use case diagram shows what can be done in the application by a type of user.

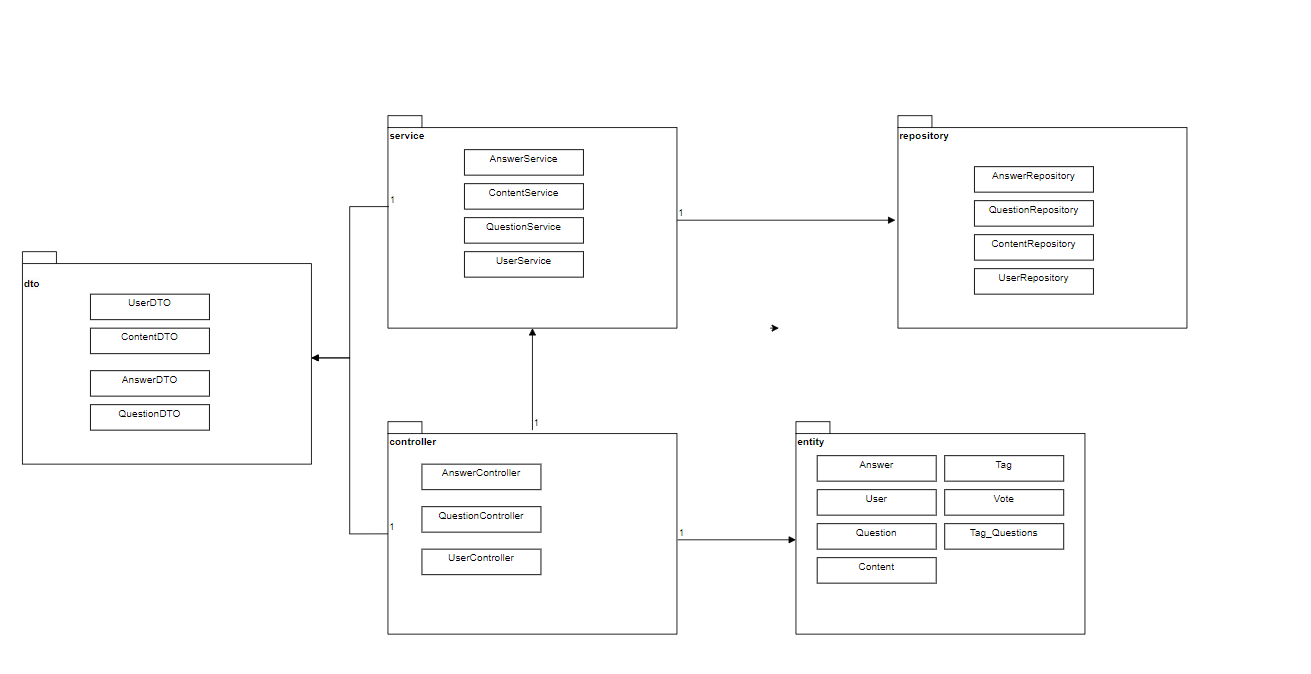


**4. Architecture**

This applications uses a layered architecture which is the most comment and widely used architectural framework is software development. The layers of this application are: controller, service, repository, model, logic, graphical user interface. In the controller there are the requests towards the database. In the service, there are all the operations needed, such as save, delete, insert, for all the tables. The role of the repository layer is to minimize scattering and duplication of the query code. The model layer is the one which retrieve data and transforms it into useful concepts for the application. The logic layer is responsible for implementing the business logic of the application. The graphical user interface layer is responsible for creating an environment in which the user can easily interact with the application and perform all the things that they are allowed to.

**5. Package diagram**

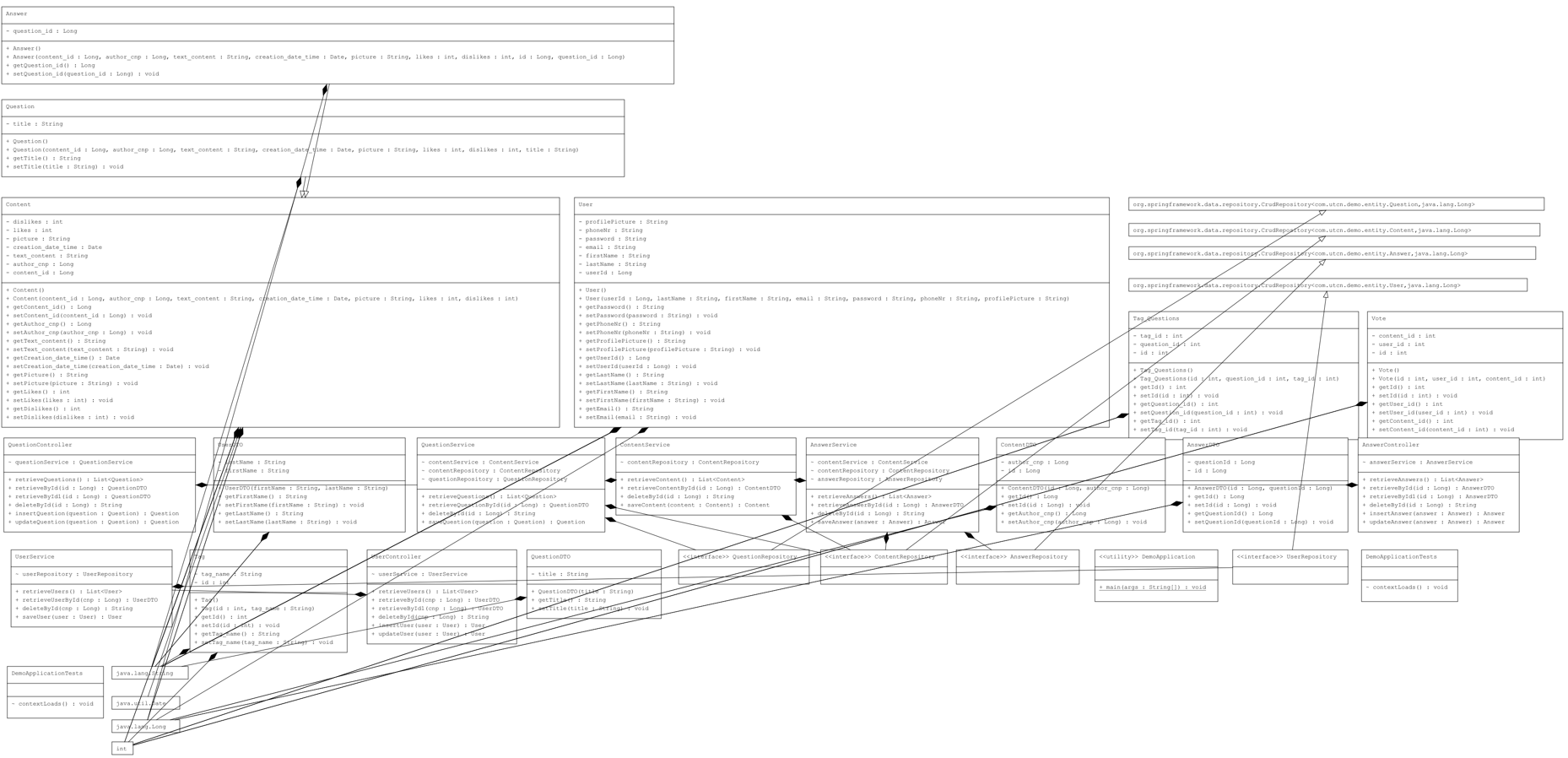
A package diagram is a type of UML diagram which provides an overview of the structure of the application by showing how it components are organized into packages.



The packages of the application take care of the layers from the architecture, such as the controller package does the controller layer, the service package does the service layer ,the entity package does the model layer, the repository package does the repository layer and the dto package does the logic layer.

**6. Class diagram**

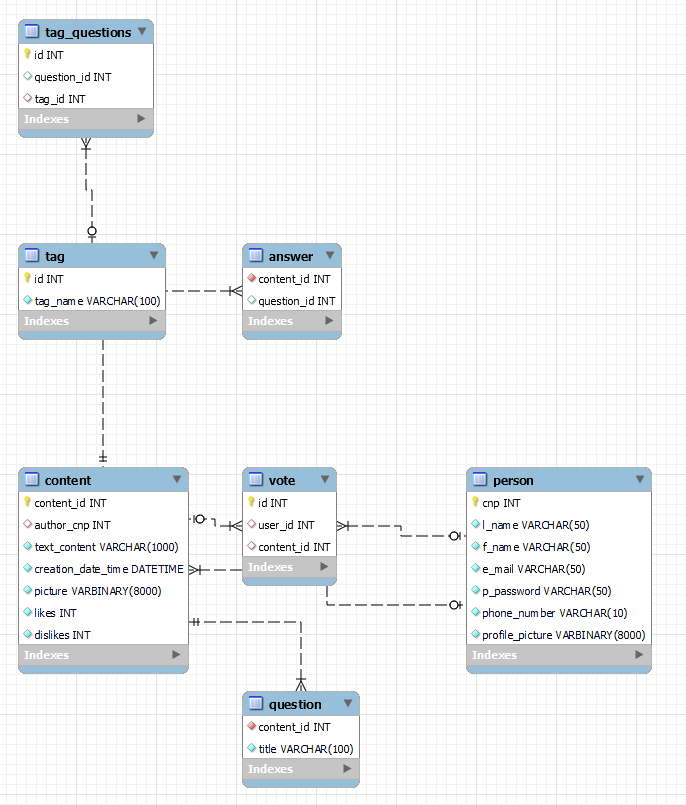
A class diagram is a type of UML diagram that represents the structure of a system by showing the classes, interfaces, associations, and other relationships between them. It provides a view of the system by modeling the classes, their attributes, methods and relationships between them.



Each class that appears in this design has its own specific purpose, which can be easily known by the name of the class. For example, the AnswerController class contains all the controller operations for the Answer, while the AnswerService contains the service side of methods for the Answer, etc.

**7.Database diagram**

A database diagram is a graphical representation of a database that shoes the tables, their columns, and the relationships between the tables. It provides a visual overview of the database schema and helps to visualize the structure of the database.



In this database diagram there can be seen all the tables, and their connections. The tag\_questions table has the role of connecting the questions to the tags, so that they are easily accessed and seen together. The content table represents the common parts of the answer and the questions (the fields that both entities should have, so that they are not repeated). The vote table holds the values of the user and the content they have voted, so that a user cannot vote on the same content twice.

**8. Endpoints requests**

The endpoints of the application can be either at answers, questions or user. They can be either a creation of an entity, an update, a deletion or a get. For instance, with the url “/answers/getAll”, all the answers which have been given are shown. If we change the last part of the url with “getById/{id}”, there is needed an input, an id, and the output is going to be the answerDTO which corresponds to that given id. For the “/answers/deleteById/{id}” url, the input is an id, which if it is valid the mentioned answer gets deleted and the output is a String, which tells if the operation has been successful or not. For the “/answers/insertAnswer” url, the input is an answer, and the output is also that answer, if it has been inserted in the database. For the “/answers/updateAnswer” url, the input is also an answer, and the output is the modified answer if there was already and answer with that same id in the database. The same operations can be done on questions and users as well, having similar results to the ones presented above, but for questions or users respectively.

**9. Architecture**

For the development of this part of the assignment, I have used the Angular framework. Angular uses a declarative approach to building user interfaces, which means developers can specify what they want the interface to look like, and Angular will take care of the underlying code needed to create it. It uses two-way data binding, which means that any changes made to the data on the user interface are immediately reflected in the model and vice versa. Angular also includes a powerful set of features for creating reusable components, managing application state, handling forms, routing, and more. It uses TypeScript, a superset of JavaScript, which adds features such as static typing and class-based object-oriented programming to the language.

For creating the actual parts of the frontend, such as buttons and divs, I have used HTML and CSS. In order to display actual questions on the screen, I have generated some mock ones in typescript, which will be replaced with the real ones by using the http.get<> method, which I have only tested to see that it works as intended. The same will be changed for the answers of each question, them needing to be taken with a similar request. The main component of the application is the app.component page, which creates the navigation part and the content part. The content is then taken with the help of router-outlet, which allows us to show just the wanted content, which has been associated with the link on which we are currently on (either by typing it in or by going through different buttons).

As for the structure of the project, I have arranged the components in different packages, so that the parts which are connected are close to one another and a bit separated from the rest.

**10. Routing**

The actual routing of the application has been done in the app.module file, where all the components are declared and all the paths are defined for each component. In an Angular app, routing is managed by the Angular Router module. The router reads the URL in the browser's address bar and matches it to a defined route in the application's route configuration. When a user navigates to a particular URL, the corresponding component or view is loaded and displayed.

As for my application the current routes are the following:

const appRoute: Routes = [  
 {path: '', component: HomePageComponent},  
 {path: 'profile', component: UserPageComponent},  
 {path: 'questions', component: QuestionPageComponent},  
 {path: 'answers', component: AnswerPageComponent}  
]

So, if the link does not have anything after it, the home page is displayed, while if it has the defined words after it, the user accesses the different components which have been specified.