StackOverflow

1. Technologies

For the Back-End of this assignment I have chosen to use Spring-Boot 3.1.4. Along Spring-Boot I have used the following dependencies:

* spring-boot-starter-data-jpa
* spring-boot-starter-security
* spring-boot-starter-web
* mysql-connector-j
* spring-boot-starter-jdbc
* spring-boot-starter-test
* spring-security-test
* jjwt-api
* jjwt-impl
* jjwt-jackson

For login and security, I chose to use JWT token because I have worked with it in the past and provide better security than simple authentication. For JWT I have chosen a validity time of 1 hour which seems appropriate for this application. For security reasons for most endpoints, I also request the user to submit the username together with the token. Even though the username can be extracted from the token, if a malicious actor gets the token from somewhere, he cannot use it because he does not expect to also submit the username in a field. This would allow requests only from trusted front ends which know about this trick. Also for obvious security reasons the passwords are stored in the database using the bcrypt algorithm, which is on purpose very slow (about 1sec/passwordHash), to discourage brute force attempts.

For the database I have used MySQL Server 8.0.28

2. OOP and Architecture

I have decided to use a layered architecture because, presentation, application processing and data management functions are physically separated. In my case I decided to use three-tier architecture:

* Controller
* Service
* Repository

Each class or interface is part in a package which has the layer’s name. We also have two extra packages named: Model in which classes for entities are described and Configuration in which JWT token and app security configurations are made.

The controller layer handles all the requests to the server. It is responsible to deny unauthenticated or unauthorized users access to resources and to ensure that the request format complies with some rules.

The service layer does all the preprocessing needed before querying the database. Mathematical operations for example can be done here.

The repository layer is where all the queries to the database are performed. Here we ask and receive answers. The answers are then sent back to the service layer and further to the controller which in the end sends them to the client.

For this project I needed the following entities:

* Answer
* AuthRequest
* Question
* QuestionAnswers
* User
* UserInfoDetails

Interesting OOP concepts that I have implemented would be the use of extended interfaces, constructor overloading and overriding functions.

3. Features and use cases

The user firstly can create an account (/user/register).

After the account is created, the user can log in (/user/login) using the credentials and he will receive a JWT Token available for 1 hour.

Only after the user is logged in, he can access the following endpoints by submitting the token preceded by “Bearer “ in the Authorization header of the HTTP Request. The endpoints accept either GET or POST HTTP Requests depending on the type of action.

The user is able to ask questions (/user/postQuestion). He can also list all of his questions (/user/getQuestionsOfUser) or search through all questions by more filters (/user/searchQuestions). The user can also edit (/user/updateQuestion) or delete (/user/deleteQuestion) his own questions. There can be no filter selected, in this way he will be prompted with all questions, or more filters and in this way he will receive questions which pass all the filters. All questions are received descending by their posted date, so newer questions will be first. The user can upvote questions (/user/upvoteQuestion) or downvote questions (/user/downvoteQuestion), only once, but can change his vote.

The user is also able to answer to questions (/user/answerQuestion), get all answers of a question(including the question) (/user/getQuestionDetails). The user can upvote answers (/user/upvoteAnswer) or downvote answers (/user/downvoteAnswer), only once, but can change his vote. The user can also edit the contents of an answer (/user/updateAnswer) or delete it. (/user/deleteAnswer).

Each question and answer have a score that modifies with each upvote or downvote of the user. Each user also has a score that modifies when one of his questions or answers gets upvoted or downvoted, or when a user downvotes another question. The users cannot vote their own question or answer.

4. Spring specific

Throughout the project I have used annotations because they enhance code readability. Here is a brief explanation on the most important annotations that I have used.

**1. @Autowired:**

I've applied the @Autowired annotation to inject dependencies automatically, reducing manual configuration and promoting loose coupling between components. This simplifies the wiring of beans, making the codebase more maintainable and scalable.

**2. @Repository:**

The @Repository annotation has been utilized to designate classes as data repositories, facilitating seamless interaction with the underlying database. By incorporating this annotation, I've abstracted away the complexities of data access, allowing for clean and concise repository implementations.

**3. @Service:**

For encapsulating business logic and service-oriented functionalities, I've annotated relevant classes with @Service. This annotation aids in categorizing components within the service layer, enabling clear separation of concerns and promoting modularity in the application architecture.

**4. @Controller:**

To handle incoming HTTP requests and orchestrate the flow of web interactions, I've annotated controller classes with @Controller. These annotated classes serve as entry points for request handling, enabling me to define request mappings and implement business logic to fulfill client requests effectively.

**5. @Entity:**

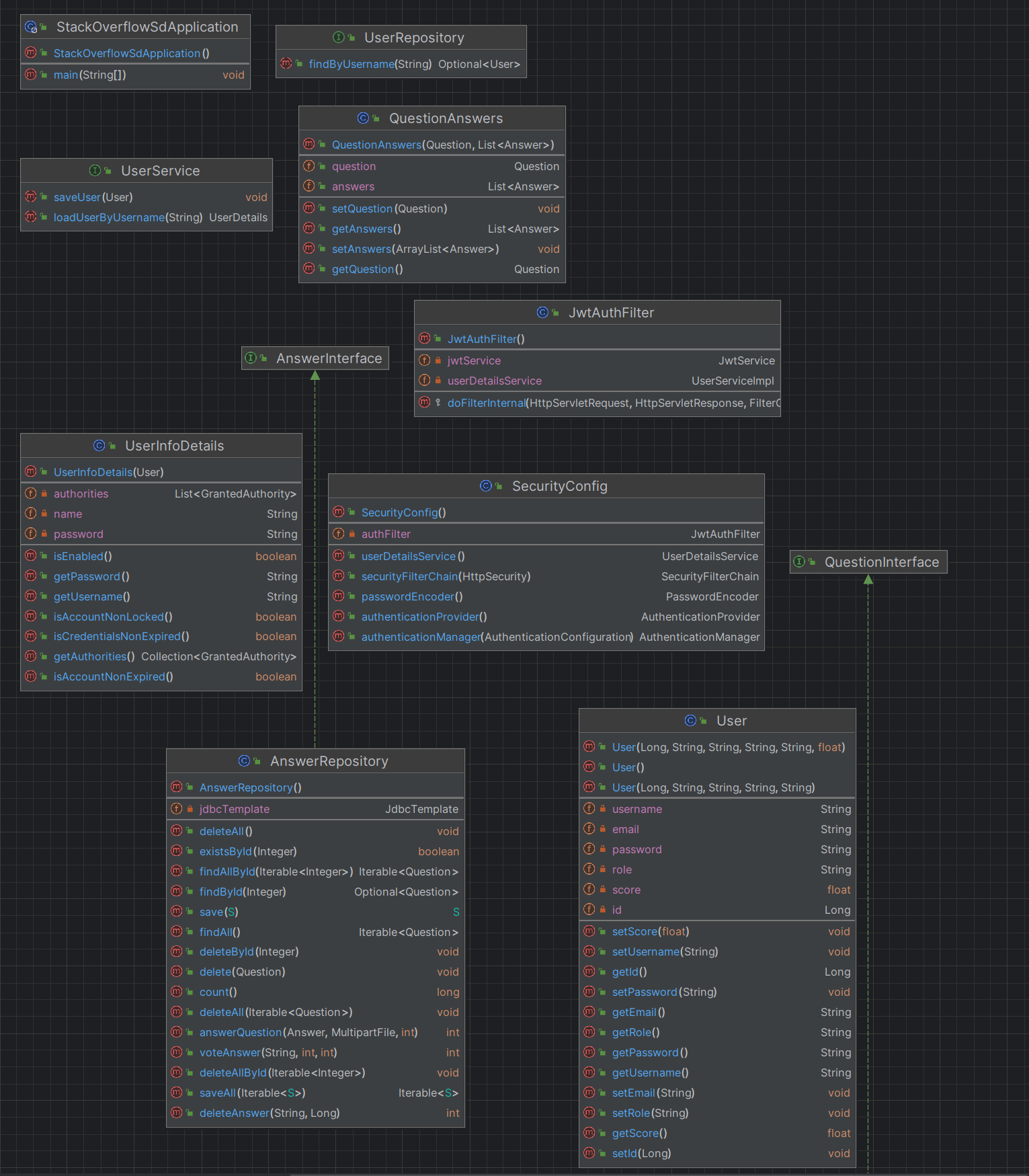
By marking domain classes with the @Entity annotation, I've established them as JPA entities, representing persistent data stored in a relational database. This annotation facilitates object-relational mapping, allowing for seamless integration between Java objects and database tables, thereby simplifying data persistence operations.

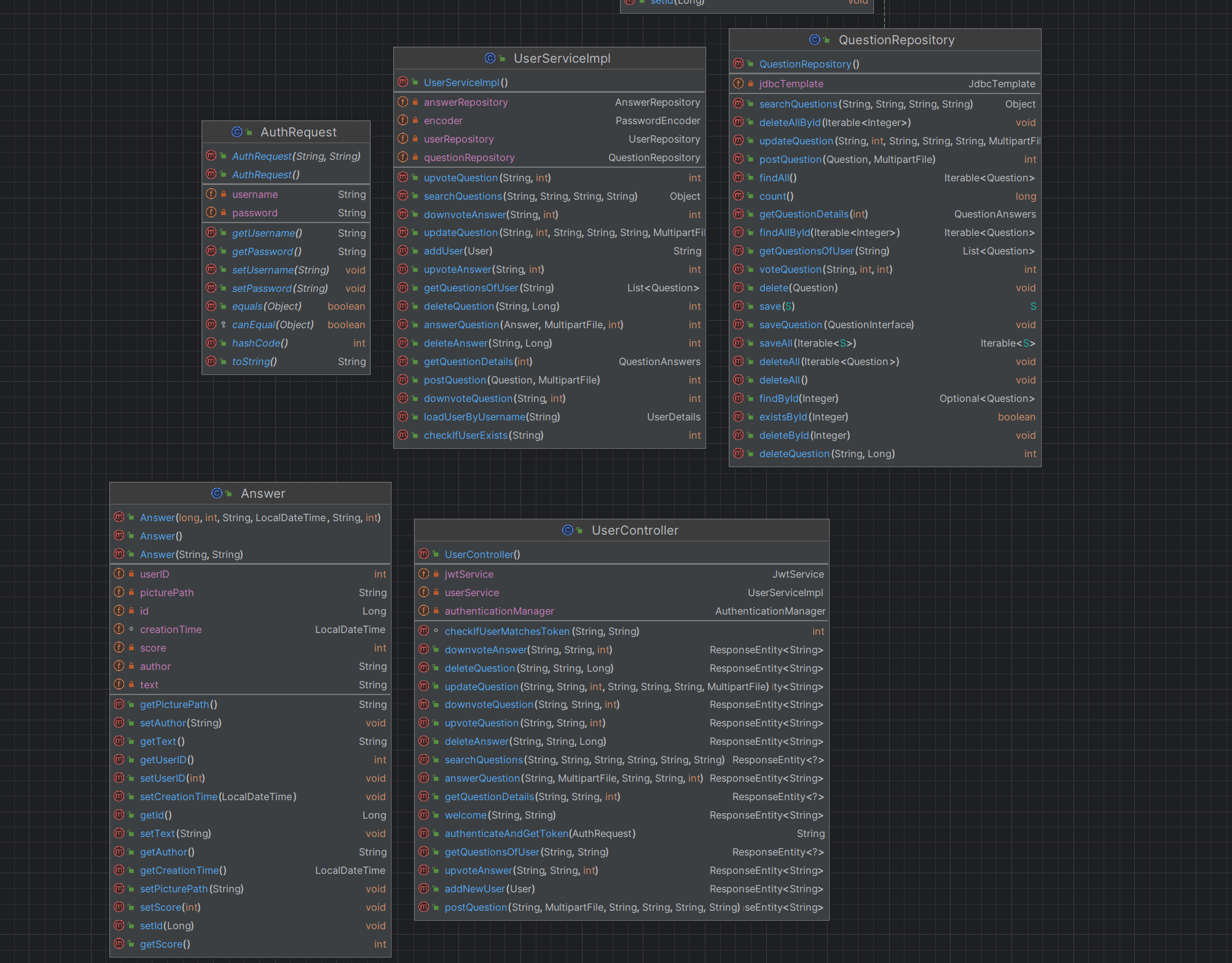
**6. @Test:**

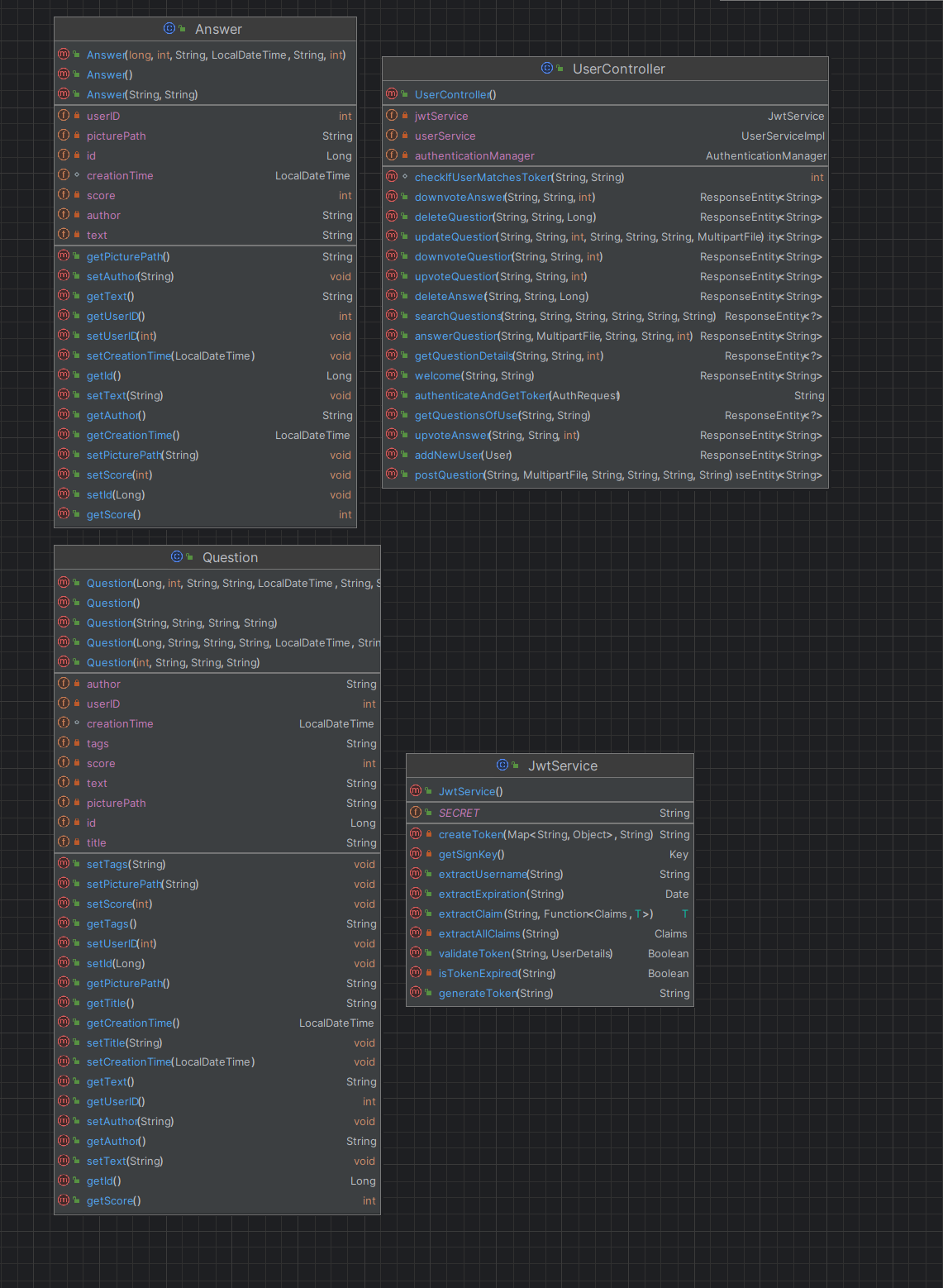
To ensure the reliability and correctness of my codebase, I've annotated test methods with @Test. These annotations signal the intent of methods to serve as test cases, enabling automated testing frameworks like JUnit to execute them during the testing phase. By incorporating unit tests, I've validated the functionality of individual components, promoting code quality and robustness.

5. Diagrams

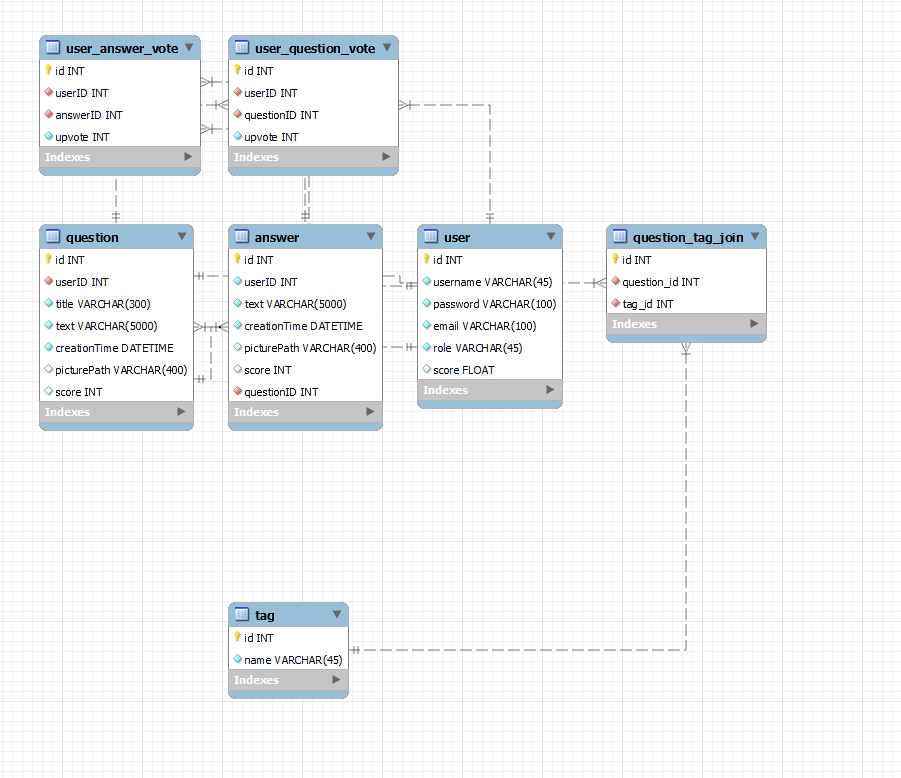
Class diagrams:







Database diagram



Package diagram

