**Problem Statement**

**Project Brief**

In this project, you will work on developing REST API endpoints of various functionalities required for a website (similar to Quora) from scratch. In order to observe the functionality of the endpoints, you will use the **Swagger** user interface and store the data in the **PostgreSQL** database. Also, the project has to be implemented using **Java Persistence API (JPA)**.

This is a group project, you would be working in a group of 3 or 4 students and there would be one final submission. Use Git and GitHub to conduct version control of your assignment code throughout your assignment development.

* As you have learnt in the version control module, it is a good software engineering practice to use version control while developing software.
* In your submission, include a link to your GitHub repo that contains the course 5 project code.

Download the stub file provided below and work on the same file to implement the required API endpoints as per the given problem statement.

**Github Collaboration Instructions**

One of the team members should act as the project lead, create a master repository for the project, and push the initial code stub to the master repository. After which, the project lead would create different branches for different functionalities to be developed, and share the repository URL with other team members.

The other team members should then fork and clone the master repository to their own repository on GitHub, so they can work on a specific branch and make updates on the project via pull requests. Also, it would be the project leader's responsibility to merge the pull requests into the master repository. It is always a good practice for each member of the team to review a pull request before it is merged into the master repository and give your comments on the pull request to help the project leader.

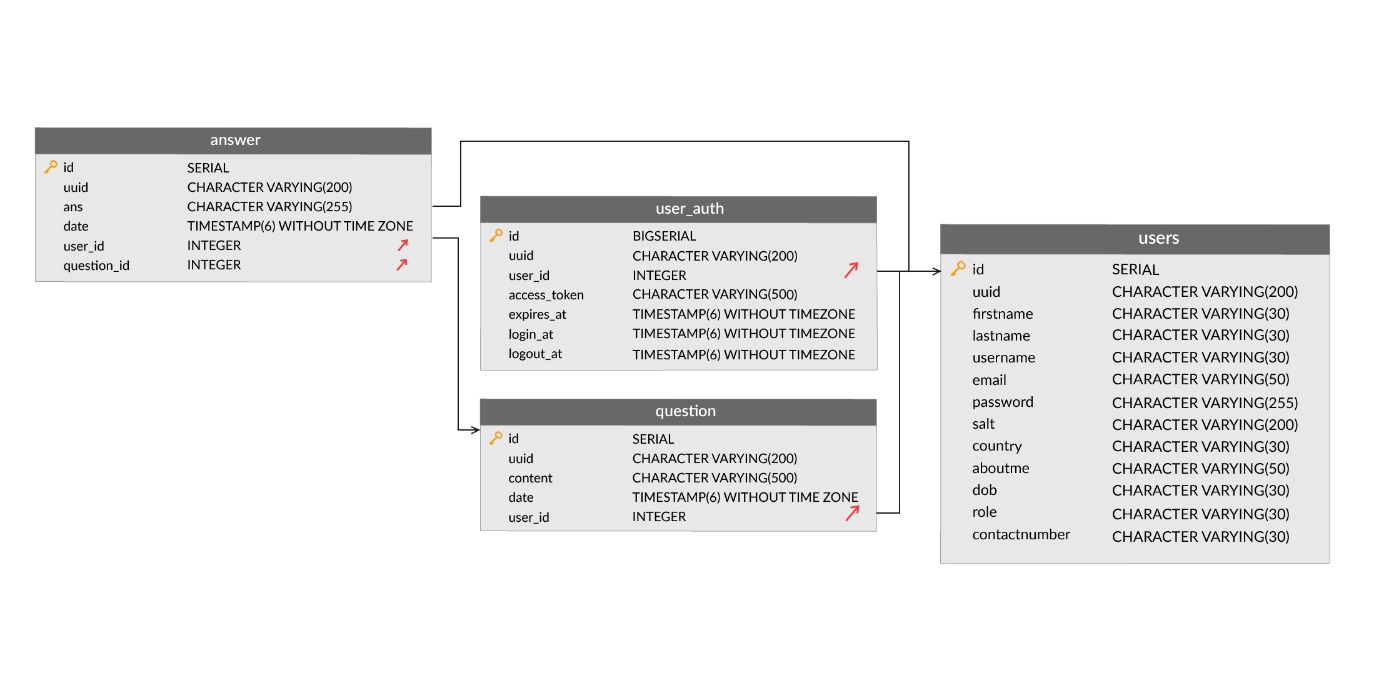
Lastly, if you are working off a fork, don't forget to[**fetch from the upstream repository**](https://help.github.com/articles/syncing-a-fork/) often, so you can get the latest commits and updates of the various branches in the upstream repository. Also, once all the required code implementation is done on a specific branch and is working fine, then the project lead can go ahead and merge the branch with the master repository.

A few additional notes to help your collaboration between teammates:

1. Use Github to [**track issues and bugs**](https://guides.github.com/features/issues/) for the project.
2. Use Github to [**conduct code reviews**](https://github.com/features/code-review), so each push request or commits are reviewed by another teammate before the code changes are merged into the main repository.

**Database Schema**

The database schema required for the project is designed and provided, as shown in the following image.



* You need to manually create a database named "quora" in your PostgreSQL and run the SQL queries against the database provided in the stub file by activating the profile setup.
* You need to update the environment variables such as server port, database name, database password in localhost.properties file in the folder located in quora-db/src/main/resources/config to integrate the database into your system with the project.

Let us recall the concept of the foreign key when a column in a table references the primary key of some other table for its reference. The table containing the primary key is a parent table and the child table contains a foreign key. When a table is related by some other table in the database and you try to delete a record from the parent table, what will happen? PostgreSQL gives the following option:  
**DELETE CASCADE** - In this case, all the referenced records in the child table will be deleted first and then the parent record will be deleted.

In the Quora project, we have used DELETE CASCADE option to delete all the referenced records in the child table first and then the record in the parent table. You can use **@OnDelete(action = OnDeleteAction.CASCADE)** annotation in JPA to specify the foreign key attribute in the Java class for DELETE CASCADE option.

**Project Structure**

The project must follow a definite structure in order to help the co-developers and reviewers for easy understanding. Also, the better project structure makes your code modular and it becomes easier to implement any new features on the existing application. Follow the directory structure given in the project stub file. The main module is divided into three sub-modules —  **quora-api**, **quora-db**, and **quora-service**.

**1. quora-api**

* **config** - This directory must consist of all the required configuration files of the project (if any). We have already provided swagger config file in the stub.
* **controller** - This directory must consist of all the controller classes required for the project (the list of required controllers along with the API endpoints are listed in the next segment).
* **exception**- This directory must consist of the exception handlers for all the exceptions. You have to implement the code for exception handler for all the exceptions to be implemented in the project.
* **endpoints** - This directory consists of the JSON files which are used to generate the Request and Response models.
* **test** - This directory consists of tests for all the controller classes. You need to uncomment all the given test cases to run these test cases after implementing the project.

**2. quora-db**

* **config**- This directory consists of the database properties and environment properties for local development.
* **sql**- This directory consists of all the SQL queries to create database schema tables.

**3. quora-service**

* **business** - This directory must consist of all the implementations of the business logic of the application.
* **dao** - This directory allows us to isolate the application/business layer from the persistence layer and must consist of the implementation of all the data access object classes.
* **entity** - This directory must consist of all the entity classes related to the project to map these class objects with the database. You need to observe the database schema and all the constraints given in SQL files carefully to map Java objects with the database.
* **exception**- This directory consists of all the exceptions related to the project. All the exceptions required for the project have been implemented in the stub file.

**Authentication and Authorization**

The authentication functionality should be implemented in such a way that the API endpoints are accessible only when a user successfully logs in to the endpoint '/user/signin'. In such a case, the user should be able to subsequently access the endpoints by providing only the access token to each endpoint. Also, the user will have access to the endpoints defined in the 'AdminController' class based on the role information provided in the database, as defined below:

* If the role of a user is 'admin', the user will be able to access all the API endpoints in the web application.
* If the role of a user is 'nonadmin', the user cannot access the API endpoints defined in the AdminController class; he/she can, however, access the rest of the API endpoints defined in the other controller classes.

**REST API endpoints - 1**

**Instructions:**

1. JSON files have been given in the stub file. You need to run these JSON files on Swagger UI and observe the details related to each endpoint. Observe the HTTP methods to make a call to that endpoint, request URL pattern, input and output parameters of the endpoint, and the status codes in each endpoint to handle the exceptions related to that endpoint.
2. All the exceptions have been implemented. You need to throw the same exception as described below in the explanation of each endpoint along with the same error code and same message. Since the test cases designed depends on the error code, so the test case would not pass if you use different error code.
3. Build the project in the main directory of the project using "mvn clean install -DskipTests". In order to activate the profile setup, move to quora-db folder using "cd quora-db" command in the terminal and then run "mvn clean install -Psetup" command to activate the profile setup.
4. Since the database is not mocked, "quora\_test.sql" file is given in the stub to create the records in the database to pass all the test cases. All the test cases would only pass if you have these records in the database. Therefore, **before running each test case** you need to ensure that the database contains all the records given in "quora\_test.sql" file. As otherwise, the test case may not pass even if the code implementation is correct.
5. In the Quora project, we will always refer to the resource's **uuid** whenever **id** of the resource is mentioned.

**UserController**

The following API endpoints must be implemented in 'UserController' class:

**1. signup - "/user/signup"**

 This endpoint is used to register a new user in the Quora Application.

* It should be a POST request
* This endpoint requests for all the attributes in 'SignupUserRequest' about the user.
* If the username provided already exists in the current database, throw ‘SignUpRestrictedException’ with the message code -**'SGR-001'** and message -**'Try any other Username, this Username has already been taken'**.
* If the email Id provided by the user already exists in the current database, throw ‘SignUpRestrictedException’ with the message code -**'SGR-002'** and message -**'This user has already been registered, try with any other emailId'**.
* If the information is provided by a non-existing user, then save the user information in the database and return the 'uuid' of the registered user and message 'USER SUCCESSFULLY REGISTERED' in the JSON response with the corresponding HTTP status. Also, make sure to save the password after encrypting it using 'PasswordCryptographyProvider' class given in the stub file.
* Also, when a user signs up using this endpoint then the role of the person will be 'nonadmin' by default. You can add users with 'admin' role only by executing database queries or with pgAdmin.

2. **signin - "/user/signin"**

This endpoint is used for user authentication. The user authenticates in the application and after successful authentication, JWT token is given to a user.

* It should be a POST request
* This endpoint requests for the User credentials to be passed in the authorization field of header as part of Basic authentication. You need to pass "**Basic username:password**" (where username:password of the String is encoded to Base64 format) in the authorization header.
* If the username provided by the user does not exist, throw "AuthenticationFailedException" with the message code -**'ATH-001'** and message-**'This username does not exist'**.
* If the password provided by the user does not match the password in the existing database, throw 'AuthenticationFailedException' with the message code -**'ATH-002'** and message -**'Password failed'**.
* If the credentials provided by the user match the details in the database, save the user login information in the database and return the 'uuid' of the authenticated user from 'users' table and message 'SIGNED IN SUCCESSFULLY' in the JSON response with the corresponding HTTP status. Note that 'JwtAccessToken' class has been given in the stub file to generate an access token.
* Also, return the access token in the access\_token field of the Response Header, which will be used by the user for any further operation in the Quora Application.

**3. signout - "/user/signout"**

This endpoint is used to sign out from the Quora Application. The user cannot access any other endpoint once he is signed out of the application.

* It should be a POST request.
* This endpoint must request the access token of the signed in user in the authorization field of the Request Header.
* If the access token provided by the user does not exist in the database, throw 'SignOutRestrictedException' with the message code -**'SGR-001'** and message - **'User is not Signed in'**.
* If the access token provided by the user is valid, update the LogoutAt time of the user in the database and return the 'uuid' of the signed out user from 'users' table and message 'SIGNED OUT SUCCESSFULLY' in the JSON response with the corresponding HTTP status.

**CommonController**

The following API endpoints must be implemented in 'CommonController' class:

**1. userProfile - "/userprofile/{userId}"**

This endpoint is used to get the details of any user in the Quora Application. This endpoint can be accessed by any user in the application.

* It should be a GET request
* This endpoint must request the path variable 'userId' as a string for the corresponding user profile to be retrieved and access token of the signed in user as a string in authorization Request Header.
* If the access token provided by the user does not exist in the database throw 'AuthorizationFailedException' with the message code - **'ATHR-001'** and message - **'User has not signed in'**.
* If the user has signed out, throw "AuthorizationFailedException" with the message code -**'ATHR-002'** and message -**'User is signed out.Sign in first to get user details'** .
* If the user with uuid whose profile is to be retrieved does not exist in the database, throw 'UserNotFoundException' with the message code -**'USR-001'** and message -**'User with entered uuid does not exist'**.
* Else, return all the details of the user from the database in the JSON response with the corresponding HTTP status.

**AdminController**

The following API endpoints must be implemented in 'AdminController' class:

**1. userDelete - "/admin/user/{userId}"**

This endpoint is used to delete a user from the Quora Application. Only an admin is authorized to access this endpoint.

* It should be a DELETE request.
* This endpoint requests the path variable 'userId' as a string for the corresponding user which is to be deleted from the database and access token of the signed in user as a string in authorization Request Header.
* If the access token provided by the user does not exist in the database throw 'AuthorizationFailedException' with the message code-**'ATHR-001'** and message -**'User has not signed in'**.
* If the user has signed out, throw 'AuthorizationFailedException' with the message code- **'ATHR-002'** and message -**'User is signed out'**.
* If the role of the user is 'nonadmin',  throw 'AuthorizationFailedException' with the message code-**'ATHR-003'** and message -**'Unauthorized Access, Entered user is not an admin'**.
* If the user with uuid whose profile is to be deleted does not exist in the database, throw 'UserNotFoundException' with the message code -**'USR-001'** and message -**'User with entered uuid to be deleted does not exist'**.
* Else, delete the records from all the tables related to that user and return 'uuid' of the deleted user from 'users' table and message 'USER SUCCESSFULLY DELETED' in the JSON response with the corresponding HTTP status.

**REST API endpoints - 2**

**QuestionController**

The following API endpoints must be implemented in 'QuestionController' class:

**1. createQuestion - "/question/create"**

This endpoint is used to create a question in the Quora Application which will be shown to all the users. Any user can access this endpoint.

* It should be a POST request.
* This endpoint requests for all the attributes in 'QuestionRequest' about the question and access token of the signed in user as a string in the authorization field of the Request Header.
* If the access token provided by the user does not exist in the database throw "AuthorizationFailedException" with the message code - **'ATHR-001'** and message - **'User has not signed in'**.
* If the user has signed out, throw 'AuthorizationFailedException' with the message code- **'ATHR-002'** and message -**'User is signed out.Sign in first to post a question'**.
* Else, save the question information in the database and return the 'uuid' of the question and message 'QUESTION CREATED' in the JSON response with the corresponding HTTP status.

**2. getAllQuestions - "/question/all"**

This endpoint is used to fetch all the questions that have been posted in the application by any user. Any user can access this endpoint.

* It should be a GET request.
* This endpoint requests for access token of the signed in user as a string in authorization Request Header.
* If the access token provided by the user does not exist in the database throw 'AuthorizationFailedException' with the message code - **'ATHR-001'** and message - **'User has not signed in'**.
* If the user has signed out, throw 'AuthorizationFailedException' with the message code-**'ATHR-002'** and message-**'User is signed out.Sign in first to get all questions'**.
* Else, return 'uuid' and 'content' of **all the questions** from the database in the JSON response with the corresponding HTTP status.

**3. editQuestionContent - "/question/edit/{questionId}"**

This endpoint is used to edit a question that has been posted by a user. Note, only the owner of the question can edit the question.

* It should be a PUT request.
* This endpoint requests for all the attributes in 'QuestionEditRequest', the path variable 'questionId' as a string for the corresponding question which is to be edited in the database and access token of the signed in user as a string in the authorization field of the Request Header.
* If the access token provided by the user does not exist in the database throw 'AuthorizationFailedException' with the message code-**'ATHR-001'** and message-**'User has not signed in'**.
* If the user has signed out, throw 'AuthorizationFailedException' with the message code-**'ATHR-002'** and message-**'User is signed out.Sign in first to edit the question'**.
* Only the question owner can edit the question. Therefore, if the user who is not the owner of the question tries to edit the question throw "AuthorizationFailedException" with the message code-**'ATHR-003'** and message-**'Only the question owner can edit the question'**.
* If the question with uuid which is to be edited does not exist in the database, throw 'InvalidQuestionException' with the message code -**'QUES-001'** and message -**'Entered question uuid does not exist'**.
* Else, edit the question in the database and return 'uuid' of the edited question and message 'QUESTION EDITED' in the JSON response with the corresponding HTTP status.

**4. deleteQuestion - "/question/delete/{questionId}"**

This endpoint is used to delete a question that has been posted by a user. Note, only the question owner of the question or admin can delete a question.

* It should be a DELETE request.
* This endpoint requests for the path variable 'questionId' as a string for the corresponding question which is to be deleted from the database and access token of the signed in user as a string in the authorization field of the Request Header.
* If the access token provided by the user does not exist in the database throw 'AuthorizationFailedException' with the message code - **'ATHR-001'** and message - **'User has not signed in'**.
* If the user has signed out, throw 'AuthorizationFailedException' with the message code- **'ATHR-002'** and message -**'User is signed out.Sign in first to delete a question'**.
* Only the question owner or admin can delete the question. Therefore, if the user who is not the owner of the question or the role of the user is ‘nonadmin’ and tries to delete the question, throw 'AuthorizationFailedException' with the message code-**'ATHR-003'** and message -**'Only the question owner or admin can delete the question'**.
* If the question with uuid which is to be deleted does not exist in the database, throw 'InvalidQuestionException' with the message code-**'QUES-001'** and message-**'Entered question uuid does not exist'**.
* Else, delete the question from the database and return 'uuid' of the deleted question and message -'QUESTION DELETED' in the JSON response with the corresponding HTTP status.

**5. getAllQuestionsByUser - "question/all/{userId}"**

This endpoint is used to fetch all the questions posed by a specific user. Any user can access this endpoint.

* It should be a GET request.
* This endpoint requests the path variable 'userId' as a string for the corresponding user whose questions are to be retrieved from the database and access token of the signed in user as a string in authorization Request Header.
* If the access token provided by the user does not exist in the database throw 'AuthorizationFailedException' with the message code-**'ATHR-001'** and message -**'User has not signed in'**.
* If the user has signed out, throw 'AuthorizationFailedException' with the message code-**'ATHR-002'** and message-**'User is signed out.Sign in first to get all questions posted by a specific user'**.
* If the user with uuid whose questions are to be retrieved from the database does not exist in the database, throw 'UserNotFoundException' with the message code -**'USR-001'** and message -**'User with entered uuid whose question details are to be seen does not exist'**.
* Else, return 'uuid' and 'content' of **all the questions** posed by the corresponding user from the database in the JSON response with the corresponding HTTP status.

**REST API endpoints - 3**

**AnswerController**

The following API endpoints must be implemented in 'AnswerController' class:

**1. createAnswer - "/question/{questionId}/answer/create"**

This endpoint is used to create an answer to a particular question. Any user can access this endpoint.

* It should be a POST request.
* This endpoint requests for the attribute in "Answer Request", the path variable 'questionId ' as a string for the corresponding question which is to be answered in the database and access token of the signed in user as a string in authorization Request Header.
* If the question uuid entered by the user whose answer is to be posted does not exist in the database, throw "InvalidQuestionException" with the message code - **'QUES-001'** and message - **'The question entered is invalid'**.
* If the access token provided by the user does not exist in the database throw "AuthorizationFailedException" with the message code -**'ATHR-001'** and message - **'User has not signed in'**.
* If the user has signed out, throw "AuthorizationFailedException" with the message code -**'ATHR-002'** and message - **'User is signed out.Sign in first to post an answer'**.
* Else, save the answer information in the database and return the "uuid" of the answer and message "ANSWER CREATED" in the JSON response with the corresponding HTTP status.

**2. editAnswerContent - "/answer/edit/{answerId}"**

This endpoint is used to edit an answer. Only the owner of the answer can edit the answer.

* It should be a PUT request.
* This endpoint requests for all the attributes in "AnswerEditRequest", the path variable 'answerId' as a string for the corresponding answer which is to be edited in the database and access token of the signed in user as a string in authorization Request Header.
* If the access token provided by the user does not exist in the database throw "AuthorizationFailedException" with the message code - **'ATHR-001'** and message - **'User has not signed in'**.
* If the user has signed out, throw "AuthorizationFailedException" with the message code - **'ATHR-002'** and message **'User is signed out.Sign in first to edit an answer'**.
* Only the answer owner can edit the answer. Therefore, if the user who is not the owner of the answer tries to edit the answer throw "AuthorizationFailedException" with the message code - **'ATHR-003'** and message - **'Only the answer owner can edit the answer'**.
* If the answer with uuid which is to be edited does not exist in the database, throw "AnswerNotFoundException" with the message code - **'ANS-001'** and message - **'Entered answer uuid does not exist'**.
* Else, edit the answer in the database and return "uuid" of the edited answer and message "ANSWER EDITED" in the JSON response with the corresponding HTTP status.

**3. deleteAnswer - "/answer/delete/{answerId}"**

This endpoint is used to delete an answer. Only the owner of the answer or admin can delete an answer.

* It should be a DELETE request.
* This endpoint requests for the path variable 'answerId' as a string for the corresponding answer which is to be deleted from the database and access token of the signed in user as a string in authorization Request Header.
* If the access token provided by the user does not exist in the database throw "AuthorizationFailedException" with the message code - **'ATHR-001'** and message - **'User has not signed in'**.
* If the user has signed out, throw "AuthorizationFailedException" with the message code - **'ATHR-002'** and message - **'User is signed out.Sign in first to delete an answer'**.
* Only the answer owner or admin can delete the answer. Therefore, if the user who is not the owner of the answer or the role of the user is ‘nonadmin’ and tries to delete the answer throw "AuthorizationFailedException" with the message code -**'ATHR-003'** and message -**'Only the answer owner or admin can delete the answer'**.
* If the answer with uuid which is to be deleted does not exist in the database, throw "AnswerNotFoundException" with the message code - **'ANS-001'** and message - **'Entered answer uuid does not exist'**.
* Else, delete the answer from the database and return "uuid" of the deleted answer and message "ANSWER DELETED" in the JSON response with the corresponding HTTP status.

**4. getAllAnswersToQuestion - "answer/all/{questionId}"**

This endpoint is used to get all answers to a particular question. Any user can access this endpoint.

* It should be a GET request.
* This endpoint requests the path variable 'questionId' as a string for the corresponding question whose answers are to be retrieved from the database and access token of the signed in user as a string in authorization Request Header.
* If the access token provided by the user does not exist in the database throw "AuthorizationFailedException" with the message code - **'ATHR-001'** and message - **'User has not signed in'**.
* If the user has signed out, throw "AuthorizationFailedException" with the message code - **'ATHR-002'** and message - **'User is signed out.Sign in first to get the answers'**.
* If the question with uuid whose answers are to be retrieved from the database does not exist in the database, throw "InvalidQuestionException" with the message code - **'QUES-001'** and message - **'The question with entered uuid whose details are to be seen does not exist'**.
* Else, return "uuid" of the answer, "content" of the question and "content" **of all the answers** posted for that particular question from the database in the JSON response with the corresponding HTTP status.