Different layers of packages:

1. Controller handles HTTP requests and corresponding responses
2. Service handles business logics related to requests
3. Mapper handles crud operations with the database
4. Model defines the entity in the database

Springboot testing:

<https://medium.com/simform-engineering/testing-spring-boot-applications-best-practices-and-frameworks-6294e1068516>

**Register:**

1. User can type in username, password, and confirm password from the frontend UI
2. Validate username, password:
   1. Not null
   2. Username is at least 4 characters long
   3. Password is at least 8 characters long
   4. No username already exists
   5. Username does not include special characters
   6. Password and confirm password are the same
3. Encrypt password
4. Create user object and insert it into database

**Login:**

1. User can type in username and password from the frontend UI
2. Validate username and password:
   1. Not null
   2. Username is at least 4 characters long
   3. Password is at least 8 characters long
   4. Username does not include special characters
3. Compare input password with the one in the database
4. Record user session
5. Return non-sensitive user information

**JWT implementation:**

1. Implement a util for JWT encoding and decoding
2. Implement an interceptor to handle request authentication, it serves the purpose of classifying user type and validating username and ID
   1. Implement functionality to read the contents (username, user role and ID) in the JWT
   2. Each of the Admin API should receive a request and check from the JwtUtil and the repository to ensure correct user type
3. Implement a web configuration to decide which routes are intercepted
4. Add username and id into the JWT

**Encapsulate responses into an object:**

1. Implement BaseResponse class to encapsulate responses type, use generic type for its data. The class should follow the below format:  
   {

“code”: “xxx”,

“data”: “{user: username}”, // Json object

“message”: “ok”

}

1. Implement a utility class to encapsulate different types of responses defined by BaseResponse.
2. Implement an error code class to describe the details of the errors

Global error handling:

1. Implement business error handling
2. Implement global error handling class to capture all the errors. This is to ensure that the frontend will get a more detailed error description while keeping the server status and loggings private. It also helps with logging all the errors of the same type.

//TODO:

1. Refactor register and login methods
2. Re-structure repositories according to features
3. Create classes to filter sensitive user information