Joseph Gallant

# QAP 1 - DevOps and SDAT

## 1. Explain how your code meets clean code practices by using at least 3 examples of your own code.

**Use of meaningful variable names**

In the following example, you can see that clear variable names have been used to describe exactly what the function does. It can almost be read as a full sentence.

A screen shot of a computer

AI-generated content may be incorrect.

**Avoid magic numbers**

Use of constants for menu options instead of numbers, for more clarity. Adding more features in the future will also be easier, as we can easily add more constants and keep track of everything.

A black screen with white text

AI-generated content may be incorrect.

**Single responsibility principle**

Functions and classes are only assigned one purpose. For example, my UserService class manages everything associated with the User. This login function only has a single responsibility, to handle User logins.

A screen shot of a computer program

AI-generated content may be incorrect.

**Only comment what's necessary**

In the following example, it makes sense to comment what the code does, because it is not so obvious at first glance. Explaining the logic behind it can save the next developer that will look at code a lot of headaches in the future.

A computer screen with white text

AI-generated content may be incorrect.

## 2. Explain your project. What it does, how it works. Explain the test cases you used.

### 2.1 Application Details

This Java Fitness Tracker application allows users to register a profile and log in. They can track cardio exercises and set up goals. At launch, the user is prompted to a login and registration menu. After registering, they can log in with their credentials. They will then be redirected to a personalized menu showing their name. From there, they can track cardio exercises, track their goals, or view and modify their profile.

The main program can be executed by running the **Main** class. A new instance of the **Menu** class is then created, which handles the menu navigation and creates a new instance of **UserService** which manages the user login and registration, their profile, their cardio exercises and their goals.

Classes for the **User**, **Exercise**, and **CardioExercise** entities are in the **Model** package.

User registration, login and logout, as well as other functions such as verifying if a username is already taken, are all part of the **UserService** class in the **UserManagement** package. This class is also responsible for managing the cardio sessions for the logged user.

#### 2.1.1 Screenshots

|  |  |
| --- | --- |
| **Unauthenticated Menu**  A screen shot of a black screen  AI-generated content may be incorrect. | **User Login Screen**  A screen shot of a computer screen  AI-generated content may be incorrect. |

|  |  |
| --- | --- |
| **Authenticated User Menu**  A screen shot of a computer program  AI-generated content may be incorrect. | **User Registration Screen**  A screenshot of a computer program  AI-generated content may be incorrect. |

|  |  |
| --- | --- |
| **Track Cardio Menu**  A screen shot of a black screen  AI-generated content may be incorrect.  **Log Cardio Session**  **A screenshot of a computer program  AI-generated content may be incorrect.**  **View Cardio History**  A screen shot of a computer program  AI-generated content may be incorrect.  **View All Cardio Sessions**  A screen shot of a black screen  AI-generated content may be incorrect. | **User Profile Screen**  A screenshot of a computer  AI-generated content may be incorrect.  **Profile Update**  A screenshot of a computer  AI-generated content may be incorrect. |
| **Track Goals**  A screenshot of a computer screen  AI-generated content may be incorrect. | **Update Goals**  A screenshot of a computer program  AI-generated content may be incorrect. |

### 2.2 Unit Tests used

**UserTest:**

The **testForSetAndVerifyPassword** unit test ensures that the password is hashed and decrypted properly using *assertTrue*, and *assertFalse* assertions. The plain password is passed to the *verifyPassword* method, which decrypts it and returns true if they match.

A computer screen shot of a computer code

AI-generated content may be incorrect.

The **testForGetAge** unit test validates that the age calculation is done correctly using the *getAge* method, with the *assertEquals* assertion. The user's birthday and current date are used for the calculation.

A computer screen shot of a computer code

AI-generated content may be incorrect.

**UserServiceTest:**

The **testForRegister** unit test verifies that the registration process is working properly. When a user is registered successfully, the *register* method returns true. By using an *assertTrue* assertion, we can be assured that it is functional.

A screen shot of a computer program

AI-generated content may be incorrect.

The **testForRegisterNull** unit test is similar to the previous test, but it takes a null value as a parameter and uses the *assertFalse* assertion.

A screen shot of a computer program

AI-generated content may be incorrect.

The **testForLogout** unit test validates that the *logout* method returns a null value. The *assertNull* assertion is perfect for this scenario.

A screenshot of a computer program

AI-generated content may be incorrect.

## 3. Outline the needed dependencies. Where did you get them from?

I found all the Maven dependencies on the MVN Repository website: [https://mvnrepository.com](https://mvnrepository.com/)

**Jupiter JUnit (v5.13.4)**: is a testing framework. Used for the various unit tests using Assertions.

Link: <https://mvnrepository.com/artifact/org.junit.jupiter/junit-jupiter/5.13.4>

**BCrypt (v0.10.2)**: used for hashing and decrypting passwords.

Link: <https://mvnrepository.com/artifact/at.favre.lib/bcrypt>

## 4. If you had any problems the QAP please explain what happened.

It was a bit challenging finding the right project structure and keeping the code clean and organized. With more planning, I was able to overcome that challenge.