Exercise 6 – Mock Testing

Goal:

The goal of this exercise is to practice unit and mock testing techniques. Based on the available code, you will analyze the implementation, derive test scenarios, and implement the missing parts to successfully run the tests.

Background:

FocusFlow follows the structure of a 3-tier architecture - a widely adopted pattern that separates an application into three distinct layers: the presentation layer, the business logic layer, and the data access layer. This separation of concerns allows for better maintainability, testability, and scalability of the application.

The presentation layer is responsible for handling user interactions and displaying information. The business logic layer, often implemented through services, contains the core functionality and rules of the application. The data access layer (repository), implemented through repositories, manages the interaction with the data storage.

The business logic layer, also called service layer, acts as an intermediary between the presentation layer and the data access layer. It implements the business logic and orchestrates the flow of data between different components. The repository layer abstracts the data storage implementation details from the rest of the application. It provides a collection-like interface for accessing domain objects.

Testing services and repositories requires different approaches due to their distinct responsibilities and dependencies. The testing strategy should focus on:

- Unit testing services in isolation (focus of this exercise)
- Integration testing repositories with the database (focus of this exercise)
- End-to-end testing of the complete flow (focus NOT in this exercise)

Exercise 6.1 (10 Points): Unit Testing of Models

Please review carefully the unit test implementations of *your* FocusFlow code repository using your gained knowledge about test design techniques from exercise five (e.g., equivalence class partitioning, boundary value analysis, decision table, etc.) Do you implement all test cases thoroughly? Implement missing test if you identified any.

Please describe your approach how you analyzed your current test implementations and document it in your repository.

Exercise 6.2 (10 Points): Service Testing

In order to store actual data (e.g., Tasks, User, Teams, etc.), a service layer is required to interact with the repository layer. This exercise focuses on testing the service layer by mocking the functionality of the actual repository layer. This approach allows testing the business logic in isolation from the actual data storage. Implement a **UserService** realizing the following functions:

- User registration with valid credentials
- User registration with invalid credentials
- User login with correct credentials
- User login with incorrect credentials
- Password validation rules
- User role assignment
- Team membership management

Next, create a test suite for the **UserService** that verifies the behavior of the described scenarios. For each scenario, you should consider to:

- Set up the necessary mock objects
- Configure the mock behavior
- Execute the service method
- Verify the expected outcomes
- Verify the interaction with the repository layer

Hints:

- You can use one of the following frameworks to implement mocks (e.g.; Java Mockito, Python Mock, etc.)
- Examples are available in https://github.com/dgrewe-hse/xyzTesting

Exercise 6.3 (10 Points): Repository Testing

The next exercise focuses on testing the repository layer using an in-memory database. This approach provides a more realistic testing environment while maintaining test isolation. Implement a UserRepository realizing the following functionality:

- Creating a new user
- Finding a user by ID
- Finding a user by email
- Updating user information
- Deleting a user
- Querying users by role
- Querying users by team membership

Next, create a test suite for the **UserRepository** that verifies the behavior of the described scenario, you should consider to:

- Set up the in-memory database
- Initialize the test data
- Execute the repository method
- Verify the database state
- Clean up the test data

Exercise 6.4 (10 Points): REST API Controller

The fourth exercise focuses on implementing REST API components as part of the controller layer, which serves as the entry point for HTTP requests in the application. This exercise serves as the basis for the upcoming week when we focus on API testing using different tools and techniques. Please consider implementing controllers for the following scenarios:

- User registration endpoint
- User login endpoint
- User profile retrieval and update endpoint
- Team management endpoints
- Role management endpoints
- Error handling and response codes
- Input validation