Advanced Programming Course: Final Project

Online Furniture Store

***clarifications in red, given on 01/02/2025*

Project Overview

In this project, your team will develop an online furniture store. The goal is to simulate a full-fledged web-like application, managing an inventory of furniture items, user profiles, and an e-commerce experience with a shopping cart, checkout, and order management system.

You'll work in small teams (3-5 people), using GitHub for version control and collaboration. You'll be expected to apply advanced programming techniques you've learned in this course, such as efficient memory management, design patterns, object-oriented principles, and automated testing.

Key Components to Build

1. Furniture Store Classes (Domain Model)

- Create classes to represent at least **5 different types of furniture** (e.g., Chair, Sofa, Table, etc.). Each class should:
  - Have attributes like name, description, price, and dimensions.
  - Implement methods to calculate discounts, apply tax, and check availability.

  Make up the items that you want to have in your store. Design the classes hierarchy. Consider using design patterns where appropriate.

2. Store Inventory

- Design an **Inventory class** to manage the available furniture items.
  - The inventory should be able to add, remove, and update the quantity of furniture.

    - Implement methods for searching by attributes (name, category, price range).

3. User Management (Basic Authentication)

- Implement a **User class** with attributes like name, email, password, and address.
    - Users can register and log in to the store.
    - Consider implementing basic password hashing (e.g., using hashlib or bcrypt).
- Users should be able to manage their profiles and view order history.

4. Shopping Cart

- Create a **ShoppingCart class** to allow users to add/remove furniture items.
    - Users should be able to view their cart and the total price.
    - Implement a method to apply promotional discounts (e.g., percentage off for specific items or total cart).
    - Consider implementing a design pattern like **Composite** for handling different types of items in the cart (e.g., furniture and accessories).

5. Checkout Process

- Design a **Checkout system** that:
    - Collects user information (address, payment method).
    - Validates the cart and available inventory.
    - Processes the payment (for the sake of simplicity, mock payment processing can be implemented).
    - Finalizes the order and updates the inventory.

6. Order Management

- Implement an **Order class** that stores the details of each user’s purchase.
    - Each order should include user information, the list of items, total price, and order status (e.g., pending, shipped, delivered).
    - Allow users to view their past orders and their status.

7. API (RESTful)

- Create a **RESTful API** using Python (Flask/Django/FastAPI) that exposes:

- **GET** methods to fetch information about furniture items, orders, and user profiles.
- **POST** methods to allow users to create new orders, register new users, and update their profile information.
- **PUT** and **DELETE** methods to update or remove items from the cart and inventory.

The API should be well-documented and robust.

7. Additional - creative enhancements

Are you a group of 4-5 people? think of enhancements that you can add to your code (suggestions: additional class types, more functionality within the existing classes). **Be creative!**

Additional Requirements

1. **Test-Driven Development (TDD) and code standards**
    a. Cover the project with meaningful tests - unittests, integration tests, regression tests etc.
        i. Ensure at least **80% code coverage** of unittests for the backend logic and API.
        ii. One regression test is required (choose a process to cover with a test. You'll find that regression test to be useful - for example, when an order is made, all other components are updated.
        iii. Integration tests are required ONLY IF you're integrating with an external service (external API).
    b. Cover the project with code standards tests (lint, black, pyink, ruff - not all because they conflict but whatever you find suitable).
    c. Tests and lint workflows should run in GitHub workflows with coverage report. We'll check that the coverage report is above 80%.
    d. Make the workflows run on each PR merged to main/master (GitHub settings) - in alignment with bullet 3 below.
2. **GitHub Collaboration**
    a. All team members should use **Git** for version control. You must:
        i. Create **feature branches** for each new feature (e.g., feature/user-authentication, feature/shopping-cart).

        ii. Use **pull requests (PRs)** for code reviews.
        iii. Each PR should be **reviewed by at least one other member** before merging into the main branch.
        iv. Ensure that the **main branch** always holds the latest stable version of the project.

3. **GitHub Workflows**
    a. Set up **CI/CD pipelines** using GitHub Actions or another tool to automate:
        i. **Testing**: Run tests automatically when code is pushed to the repository.
        ii. **Linting**: Ensure code style consistency (e.g., using black or flake8).
4. **Code Documentation and type hinting**
    a. Comment your code and use **docstrings** for classes and functions. This will help reviewers understand your design decisions.
    b. Provide a **README** file with:
        i. Group members.
        ii. A description of the project.
        iii. Setup instructions (dependencies, how to run the application locally).
        iv. API documentation.

5. **Database Handling**

you may choose whatever technology you want for storage handling. data may be stores in pandas as pickles, mysql, or files like csv or json. You may also consider an external server. Whatever you choose, make sure to do the IO actions as clean as possible (read/write/copy/list). Design it appropriately!

****UI is not required!**

Evaluation Criteria

Your project will be evaluated based on the following:

1. **Code Quality and Design**
    a. Use of Object-Oriented Design (OOP) principles, design patterns, and clean code practices.
    b. Efficient memory usage and code optimization, particularly in parts that could potentially handle large inventories or multiple users.

2. **Functionality**
    a. The furniture store should have a working inventory system, shopping cart, and checkout process.
    b. The API should be fully functional and responsive.
3. **Testing**
    a. Comprehensive unit tests and other tests with high coverage and edge case handling.
4. **GitHub Collaboration**
    a. Proper use of version control, clear commit messages, meaningful pull requests, and effective collaboration with team members.
    b. Quality of code reviews and integration of feedback.
5. **Documentation**
    a. Well-documented code and clear instructions for setting up and running the project. Add your designed API calls for some use cases (choose 3-5 main actions).
    b. Explain at least 3 Design patterns that were used in your project.
6. **Creativity and Additional Features (for groups with 4-5 members)**
   Think of additional functionalities that could be implemented in an online store.

Submission Instructions

- **Deadline: Sunday 02.03.25 (instead of 20.2.25), 23:00**
- Submit your project as a **GitHub repository** link.
- Ensure that your repository includes:
    - A detailed README as explained above.
    - Project design (can be stored as a pdf).
        - Include a brief explanation of how the design changed over the course of the implementation. Keep track of the design changes!
    - Clear instructions on how to run the application and tests.
    - All code and tests.
    - GitHub Actions workflows for CI/CD.