Refactoring the Quackstagram Codebase

Table of contents

1 Refactoring and Code Smells

In this lab, you'll look at the Quackstagram codebase with a critical eye, identifying opportunities for improvement through refactoring. Your goal is to enhance the code's structure, maintainability, and scalability without altering its external behavior.

You will have to consider class responsibilities, potential new classes, and the removal of obsolete ones.

Lab Duration: Approximately 2 hours

Learning Objectives:

- Understand the principles and benefits of code refactoring.
- Analyze the existing Quackstagram codebase to identify areas for improvement.
- Apply refactoring techniques to Quackstagrams codebase.
- Think critically about class responsibilities and the overall architecture of the application.

Preparation:

1. Review the principles of code refactoring and familiarize yourself with common refactoring techniques (Revisit the refactoring lecture).
2. Revisit the Quackstagram source code with a focus on understanding its current structure and functionality.

Lab Tasks:

1. **Codebase Review and Analysis**:

- Start with a thorough review of the Quackstagram codebase. Understand the current classes, their responsibilities, and how they interact.
- Identify any code smells or areas where the code might not adhere to principles of good design.

2. **Envisioning Improvements**:

   - Discuss as a team how the code could be improved. Consider new class responsibilities, the removal of unnecessary classes, and potential for new classes.
   - Think about how these changes could make the code cleaner, more modular, and easier to maintain.

3. **Refactoring in Action**:

   - Begin implementing refactoring changes. Focus on one area or class at a time to ensure that the external behavior of the application remains unchanged.
   - Continually test the application to ensure that your refactoring isn't introducing any new bugs or issues.

4. **Optional: Addressing Code Smells**:

   - If you're making good progress, start looking for and addressing any code smells. These might include long methods, large classes, duplicate code, or unnecessary complexity.

Post-Lab Reflection:

- Consider how the refactored code might better accommodate the requirements you identified in the previous lab.

End of Lab:

- Ensure that all refactoring changes are well-documented (before and after code snippets) and tested. Your code should be in a stable state, ready for further enhancements.
- Be prepared to justify the changes you've made to the codebase to your peers or instructors.