Course Project: Cheapo Software Solutions (CSS.inc).

Thursday, March 28, 2024

Table of contents

1 Social Media App for DACS Opportunity (BCS1430 30%)

Cheapo Software Solutions is a company that specializes in market 'opportunities' in the software sector. They have just spotted one such opportunity. DACS (read as DAACKS or ducks) is looking for software to allow students to share images with each other, similar to Instagram. They want a system that has all the usual image sharing application functionalities (allow users to create profile, post images, like images and follow other users), and has a nice GUI feel to it (Java Swing or FX).

At the core of this app is the feature that allows users to set up and personalize their profiles. Users sign up, log in, and add a personal touch to their profiles with photos and bios, similar to what they would do on Instagram. Following profile setup, users can begin sharing snapshots of their lives by uploading images with captions and (optionally) tags. The system is designed to handle a variety of image formats, ensuring every shared moment is displayed in its best light.

Interaction is a crucial part of the experience. Users can explore new posts, connect with peers through following and messaging, and interact with posts through likes and comments. The Explore feature enables users to discover trending content and new profiles, encouraging them to engage with fresh and interesting posts. Real-time notifications keep users updated on new interactions, ensuring they're always in the loop.

But it's not all just socializing and sharing; the application should also prioritize a safe and orderly environment. Admin users play a crucial role here, with special privileges to moderate content and address user concerns. All user data, including posts, comments, and likes, are managed and stored with care, ensuring both integrity and a seamless user experience.

DACS has their own database where they securely store all the images, user information and application related metadata. This database is not part of the specification for this project. Instead, the application has to load up a number of "txt" files that is stores all the app related data.

The 'opportunity' that Cheapo Solutions perceives is that it has found some software in a university course material. This software is written in Java and is appropriately called "Quakstagram". It already has much of the functionality that the DACS want. Cheapo Solutions thinks that they will be able to adapt this existing system to their needs and reap the huge reward (€80,000) that the DACS is willing to offer.

You have just been taken on by Cheapo Solutions (being paid a miserable 30% of your BCS1430 module /annum) and they have tasked you with adapting the Quakstagram system to the needs of the DACS. You have over 7 weeks to achieve this. Specifically your job is to

1. In-depth Requirement Analysis:
    1. Conduct a thorough examination of the current Quackstagram codebase to extract all existing functionalities.
    2. Engage with potential users and stakeholders to gather additional requirements and expectations.
    3. Document a detailed requirement specification, including functional and non-functional requirements.
2. UML Modeling:
    1. Create comprehensive UML diagrams representing the entire [refactored] system, including structural, behavioral, and interaction diagrams [At least 2 diagrams of each type, in total 6 diagrams].
    2. Develop a series of detailed use cases to illustrate possible interaction users can have with the [refactored] system, including edge cases and error handling [At least 6 use cases].
3. Refactoring:
    1. Employ refactoring techniques to optimize code readability, performance, and maintainability.

2. Document each refactoring step with a clear justification and its impact on the overall system's performance and maintainability.
3. Establish a refactoring log to track changes and ensure that all modifications align with the initial design objectives. You will need to provide us with before and after code snippts of re-factoring [1] .

4. Design Patterns:

   1. Analyze the system's needs and apply multiple design patterns where applicable.
   2. Provide a detailed explanation and diagram for each design pattern used, including how it fits into the larger system architecture. You would have to show us before and after UML diagrams for the [part of] code that has changed with the design pattern.

Senior Cheapo Technicians (Spriha and your TAs) will be available to help you for 2 hour every week in your labs, but you will probably need to work after hours to achieve this target (Hey – it is "Cheapo Software Solutions"!).

2 Product Delivery

You have to do this in teams of 4 but I will expect all team members to be able to explain the code-base to me and reserve the right to hold interviews in the week following the final submission to make sure that each student can explain their/their team's submitted code.

2.1 Phase 1: Plan for Refactoring + New functionalities

In the first phase you have to understand the requirements and the existing code. In the first four weeks you will work on a requirements document, UML diagrams for the current code, plan for refactoring the code and a proposal for a new functionality. Fill in the Phase 1 Assignment Document provided on Canvas to submit your deliverables on Canvas by the Wednesday of Week 4 (06-03).

2.2 Phase 2: Design Patterns

The second phase is about implementing all the proposed changes from phase 1 and design patterns. You should submit the fully refactored source code with extensive comments and documentation, ready to compile/execute, to Canvas by Thursday of Week 7 (28-03). This

[1]There is no minimum number of refactoring that you need to do, we have un-intentionally put in many chunks of code that need refactoring (hey, this a direct consequence of operating with a budget that could rival the thickness of a single-ply tissue paper—a hallmark of the economical ingenuity for which Cheapo Software Solutions is renowned).

should include the instructions to build the executable and run it in a readme file. Along with the source code, you will also submit a UML diagram for your code reflecting all the changes you implemented from phase 1. Moreover, you will submit a 5-minute video going over the refactored code and the diagram to show the design pattern you implemented and give a demo of the functionalities of the application.

2.3 Grading Breakdown:

- Requirement Documentation and UML Models (8%): Your ability to analyze the Quackstagram system, document its functionalities, and create comprehensive UML diagrams reflecting the system's architecture.
- Code Refactoring and Object-Oriented Design (8%): The effectiveness of your refactoring efforts to optimize code, adherence to object-oriented design principles, and overall system maintainability.
- Design Pattern Implementation (8%): The appropriateness and justification of design pattern choices and their integration into the larger system architecture.
- Application Functionality and User Experience (6%): The robustness, performance, and user experience of the final application.

3 Deliverables

- Phase 1: Requirements + UML + Refactoring (16%)
  - Report:
    - Finalized list of requirements and their priorities
    - UML Diagrams
    - Code Refactoring Snippets and justification
    - OO Design change code snippets and justification
  - Code:
    - Refactored code to account for bad design practices and OO design
- Phase 2: Design Pattern + New Functionality (8%)
  - Report:
    - Description of design pattern implementation
    - Overview of new functionality (if possible with updated UML diagrams)

  - Code:
    * Well documented code with design patterns implementation
  - Video Demo:
    * Provide a link to the 5-minute video going over the refactored code, UML diagram, design pattern implementation, and a demo of the application functionalities.
- Demo Day (Week 7 Lab): New Functionality Demonstration (6%)
  - Demo:
    * You give a live demo to your examiners (Ashish and Spriha) of your added functionalities.

4 Some tips

- Start early! There are a lot of deliverables for the project. They are all doable if you start early and are consistent. But probably not if you wait up till week 3 to start :)
- Read all the relevant documents carefully. Make sure you understand what the problem is and what is being asked of you. If anything is unclear, make sure you reach out to your teachers or TA's in the lab and get it clarified from the get go. Be proactive.
- Make a plan! To manage all tasks and your time make a schedule for your group and define milestones and indicate all important deadlines. This will help you be on track and ensure all deliverables are met at the end. You may make a gantt chart and use the issue boards just like the semester project!
- Communication is key in group work. Make sure all group members are aligned and know their tasks. Meet regularly for example weekly to meet up and work or discuss any open issues. If you're stuck with a problem don't wait up for the next meeting in a week - that will just cause a delay in work! Reach out, communicate, be proactive!
- Keep up with the labs! They are designed to help you with the project and give you time to work on it while having the support of TA's. Take advantage of this!
- Don't forget to have fun! While you have a lot of deliverables for this project which can make it look like a daunting task, don't forget it was designed so you can learn about the theoretical concepts of good and efficient programming - in a fun way. So have fun with it and don't shy away from showing off your creativity to us! We will love it and are here for it