Software Engineering and Testing. BSC Year 2, 2024/2025
(Assignment 3 - 20%)

Assessment 3: Design and Draft Implementation

Submitted by: Danyil Tymchuk (B00167321) &

Artem Surzhenko (B00163362)

21/03/2025

Declaration

I herby certify that this material, which I now submit for assessment on the programme of study leading to the award of Ordinary Degree in Computing in the Institute of Technology Blanchardstown, is entirely my own work except where otherwise stated.

Author: Danyil Tymchuk Dated: 21/03/2025

Author: Artem Surzhenko Dated: 21/03/2025

Table of Contents

Title: QWERTY - Social Media

Abstract / Executive Summary

This document outlines the design for "QWERTY," a web-based social platform that allows users to share text posts and interact with others. Following Object-Oriented Analysis and Design principles, we present class diagrams and entity-relationship diagrams that form the foundation of our implementation. The design addresses all requirements from our previous assessment while providing a clear roadmap for development using PHP, MySQL, and HTML/CSS technologies.

Project Definitions

Purpose of document: This document translates the requirements for the "QWERTY" social network into a concrete design that will guide implementation.

What is the project? "QWERTY" is a simple web-based social platform where users can create accounts, write text posts, search and follow other users, view users profiles and edit their own profile data.

Functional Specifications: Account management, post creation, following system, and administrator crud panel. Login and registration for new users.

Main components: User Interface Layer (login forms, post display), Business Logic Layer (authentication, account management), and Data Access Layer (database operations).

Document Revision

Rev. 1.0 - 21/03/2025 - Initial version

Methodology

System models -- UML

For this project, we use Unified Modeling Language (UML) to visualize and document the software system structure. UML provides standardized diagrams that help communicate design concepts between team members.

Use of, and necessity of OOAD

Object-Oriented Analysis and Design (OOAD) is essential for this project as it:

- Allows modular development of components
- Promotes code reusability through encapsulation
- Provides clear organization through abstraction
- Simplifies maintenance through well-defined interfaces

Purpose of using classes / What is a class diagram?

Classes encapsulate data (attributes) and behavior (methods) of objects within our system. A class diagram shows the system's classes, their attributes, methods, and relationships between objects, serving as a blueprint for implementation.

Static Versus Dynamic Case Diagrams

Static diagrams (class diagrams, ERDs) show system structure at a specific point in time, while dynamic diagrams (sequence diagrams) show behavior over time. Our focus is on static diagrams to establish the system structure.

What is an ERD?

An Entity-Relationship Diagram (ERD) represents the database design, showing entities (tables), attributes (columns), and relationships between entities.

Volatile versus Persistent storage

Our design uses both:

- Object instances (volatile storage) for business logic in memory
- Database tables (persistent storage) for long-term data retention

User Interface template

We've chosen a minimalist HTML/CSS design that aligns with our non-functional requirements of simplicity and accessibility, ensuring core functions are accessible within two clicks.

Requirements

Use Cases

From Assessment 2, we identified these key use cases:

1. User Registration
2. User Login/Logout
3. Create Post
4. Search for users
5. Follow users
6. View Posts
7. Admin User Management

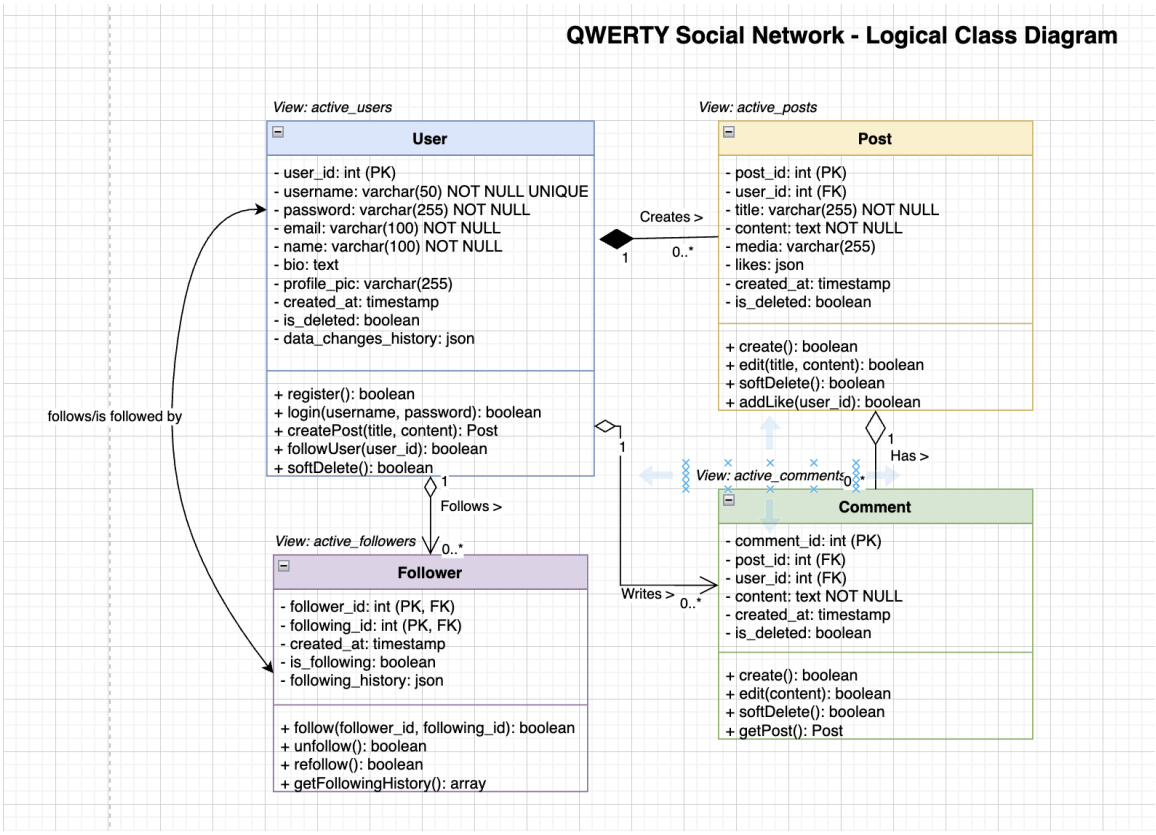
Use Case Specifications

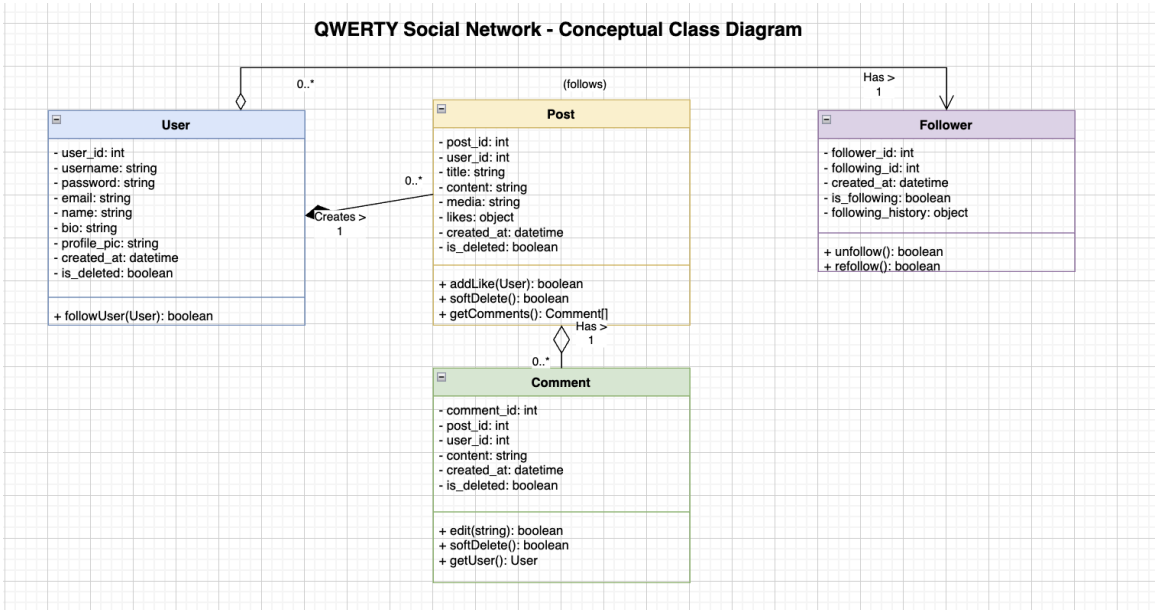
Use case specifications have guided our class and database design. For example:

Use Case: Create Post

- Actor: Registered User
- Description: User creates a new text post
- Flow: User selects "Create Post" → System displays form → User enters text → System validates → System saves → Success message
- Class Implication: User abstract, and UserCreator extend User, UserProfile for operation with user profile data, na dUserManager for admin panel

Class Diagrams





Our class diagram shows 4 main classes:

1. **User**: Core user functionality with attributes (username, password) and methods (login, register)
2. **Post**: Represents user posts with content and timestamp
3. **Comment**: Represents comments on posts
4. **Follower:** Represent following relationship between users

Class Relationships:

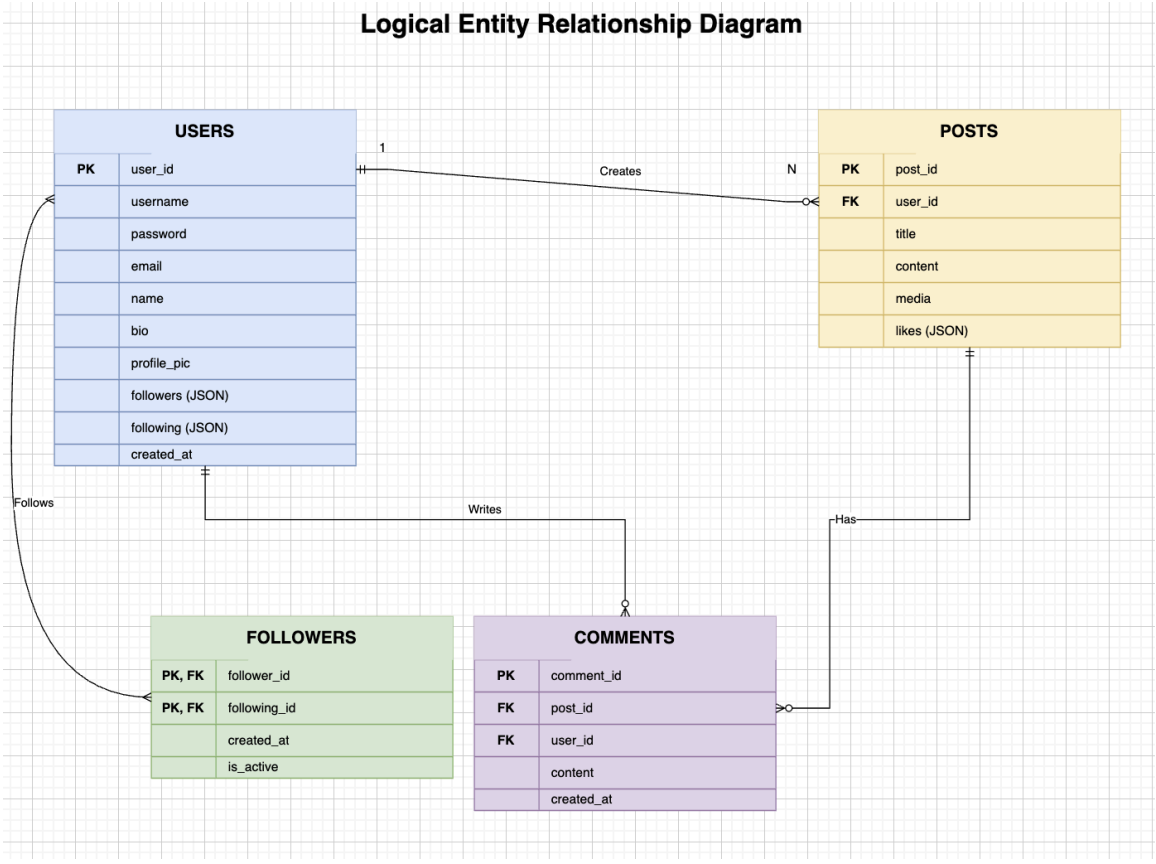
1. Composition: User owns Posts (strong relationship)

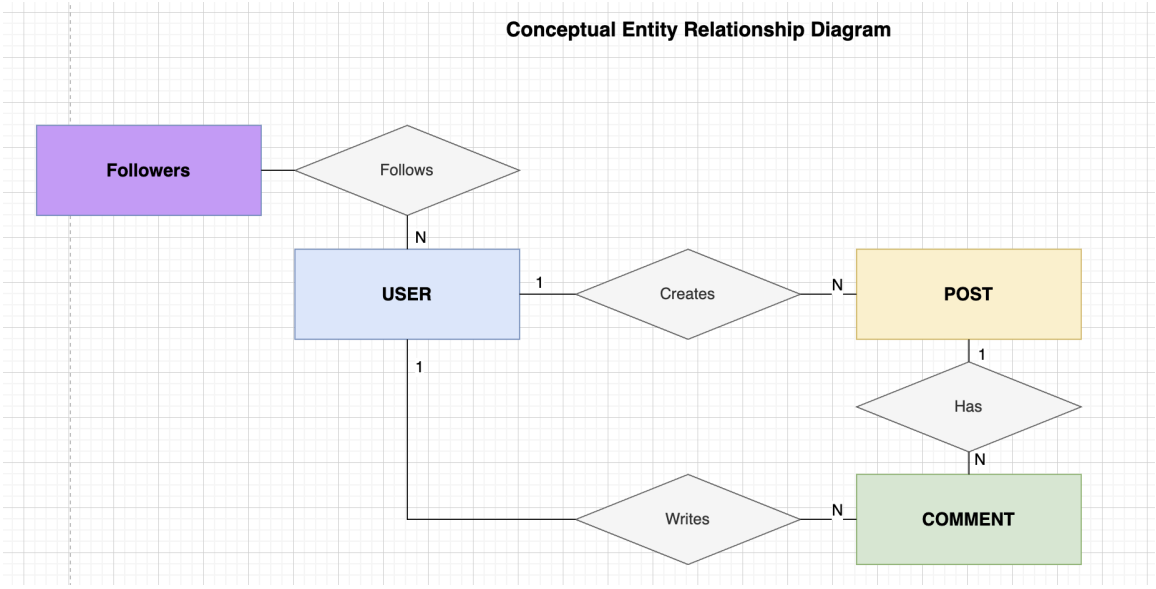
2. Aggregation: Post has Comments (weak relationship)

3. Association: User writes Comments

4. Association: User follows other Users via Follower

Entity Relationship Diagram





Our ERD consists of three main entities:

1. **USERS**: Stores account information (PK: user_id)
2. **POSTS**: Stores post content (PK: post_id, FK: user_id)
3. **COMMENTS**: Stores user comments (PK: comment_id, FK: user_id, post_id)
4. **FOLLOWERS:** Stores following relationship between users (PK: (FK: follower_id, FK: following_id))

Key relationships:

- One User creates many Posts (1:N)
- One User writes many Comments (1:N)
- One Post has many Comments (1:N)
- Many users can have many following (M:N)

Design decisions:

- Database normalized to third normal form
- Foreign keys implement referential integrity
- Appropriate data types chosen for performance

Conclusions

The design successfully translates our requirements into an implementable architecture. Through class diagrams and ERDs, we've created a modular system with clear separation of concerns.

Modifications from the original proposal:

1. Simplified authentication system
2. Enhanced post metadata
3. Focused admin functionality on essential moderation tasks

The current design provides a solid foundation for implementation while maintaining the scope outlined in Assessment 2.

GitHub repo: https://github.com/DanyilT/WebDev-Project

Last version: https://github.com/DanyilT/WebDev-Project/tree/83b1dc4c97385afbb3a8d0cd13731363d6d1c769

Checklist: Is your document complete and correct?

Content:

- Do the requirements state the customers' needs
- Are you satisfied with all parts of the document
- Do you believe all parts are possible to implement
- Is each part of the document in agreement with all other parts
- Do the requirements avoid specifying a solution
- Do the requirements avoid specifying a design

Completeness:

- Are all the necessary interfaces specified – this includes input and output
- Are the specifications precise enough
- Are all sections from the document template included – if changed, why?

Clarity:

- Are all requirements reasonable?
- Is the level of details for each requirements appropriate?
- Are the requirements written in a language appropriate to the reader?
- Are all items clear and unambiguous?