Ain Shams University Faculty of Engineering

Project Report

CSE231 Advanced Computer Programming

Project Title: Social Media Platform

Prepared by :

Abdallah Belal Momen 22P0036
Ahmed Khaled Hamdy 22P0024
Malak Ossama Zaiter 22P0052
Mohamed Khaled Mohamed 22P0041
Yousef Sameh Shoman 22P0010

Prepared to :

Dr. Mahmoud Khalil
Eng. Mostafa Ashraf

Abstract

This document outlines the creation and major attributes of an extensive Social Media Platform, developed using Java and focusing on Object-Oriented Programming (OOP) techniques., and enhanced with Graphical User Interfaces (GUI), multithreading, and basic networking functionalities. The first milestone focused on establishing the core structure of the platform through the creation of user profiles, posts, and interaction systems. The second milestone extended the platform's capabilities by incorporating a dynamic GUI, real-time updates through multithreading, and network communication between users.

Additional features such as direct messaging and trending topics were also explored as part of a bonus challenge. This report presents the analysis, design, implementation details, and testing results of the social media platform, providing a thorough overview of the project from conception to completion.

Introduction

Today, many social media website platforms have become a crucial part for people interaction and information sharing between each other in all different parts of the world. This Social Media Platform is not just about giving users a simple and efficient way to interact with each other but also as an educational tool to apply and demonstrate the knowledge of Object-Oriented Programming (OOP) in Java.

Purpose

The main purpose is to apply and strengthen the OOP concepts learned in Java. The software is primarily based on principles like class inheritance, encapsulation, polymorphism, and interface implementation targeting its scalability and maintenance.

Moreover, we created this platform to maintain a credible destination point where members can offer their content, interact with comments, and handle individual profiles themselves. It is a digital hangout for meaningful discussions about ideas, experiences, and interests in an efficient, and secure online platform.

Platform interface and data management are performed with confidence to put forward the integrity of the product whilst also providing a key reference point for the effectiveness of OOP in software development.

Table of Contents

Abstract I

Introduction I

Purpose I

Description 1

Core Features 1

Detailed Feature Description 2

Beneficiaries of the Project 3

Detailed Analysis 3

Project Structure & Key Components 3

Functional Requirements 4

Non-Functional Requirements 4

UML Class Diagram 5

User Class 7

Network Class 9

Comment Class 10

Post Class 11

Task Breakdown 13

Time Plan 15

System Architecture & Design 19

Test Scenarios & Results 21

End-user guide 24

1.Getting Started 24

2.Main Interface 25

3.Navigation and Features 27

Conclusion 28

Appendix 28

References 29

Figure 1 User Class 7
Figure 2 Networking Class 9
Figure 3 Comment Class 10
Figure 4 Post Class 11
Figure 5 Gnatt Chart 18
Figure 6 Login Page 24
Figure 7 Sign Up Page 24
Figure 8 Creating Post 25
Figure 9 Interacting with a Post 25
Figure 10 View User Profile 26
Figure 11 Managing User Friends 26
Figure 12 Messaging Friends 27
Figure 13 Navigation Options 27
Figure 14 UML Class Diagram 28

Description

This project involves developing a comprehensive social media platform utilizing Java, focusing on Object-Oriented Programming (OOP) principles, Graphical User Interface (GUI) design, multithreading, and basic networking. The platform will facilitate user interaction through creating and sharing posts, searching for friends, viewing profiles, and engaging in real-time chat.

Core Features

1. Authentication

- **Login/Sign-Up Page:** Secure and user-friendly interface for new users to sign up and existing users to log in.

2. Home Page

- **Create Posts:** Users can create text or multimedia posts.
- **Search for Friend Profiles:** Users can search and view friend profiles.
- **Engage with Posts:** Users can like and comment on posts.
- **Profile Access:** Users can access their profile to view and edit certain details.

3. Explore Page

- **Discover Content:** Users can explore trending topics and posts.

4. Profile Page

- **View Profile:** Users can view their profile, which displays:
    - Number of followers and following.
    - History of posts.
- **Edit Profile:** Users can edit their bio and profile picture (username cannot be changed).
- **Followers List:** Users can view their followers and initiate chats.

5. Chat Functionality

- **ChatView:** When users tap on a follower, they are directed to a chat interface where they can send and receive messages with timestamps.

Detailed Feature Description

Authentication

- **Login/Sign-Up:** Secure sign-up with password encryption. Users can log in using their credentials.

Home Page

- **Create Posts:** Users can post text, posts are displayed on the home feed.
- **Search and View Profiles:** Search bar to find friends by username. Viewing profiles shows the user's posts, followers, and following.
- **Like and Comment:** Interactive features to like and comment on posts.
- **Profile Access:** Quick access to the user's profile to view their activity and manage settings.

Explore Page

- **Discover:** Explore trending posts based on like count.

Profile Page

- **Profile View:** Comprehensive view showing user information, post history, and social connections.
- **Edit Bio and Profile Picture:** Users can update their bio and change their profile picture to reflect their status.
- **Followers List:** View a list of followers and initiate direct chats by tapping on a follower's name.

View Chat

- **Messaging:** Direct messaging system, showing timestamps for all messages sent and received.

Beneficiaries of the Project

- **Users:** Gain a platform to share and interact with content, connect with friends, and communicate.

- **Developers:** Learn and implement advanced Java concepts, including OOP, GUI design, and networking.

Detailed Analysis

Project Structure & Key Components

1. Classes and Functionality

- **User and UserProfileView**: These classes manage user data and user profile display. This includes user attributes such as username, bio, profile picture, and friend list.

- **Post and Comment**: These classes handle posts and comments in the social media platform, including creating and interacting with posts and comments.

- **Message and UserMessagesView**: Manage user messages and display them, which is a part of the direct messaging feature mentioned in the bonus section of the project overview.

- **Networking and Database**: The Networking class may handle all network-related operations such as sending and receiving data to/from the server, whereas the Database class handles data storage and retrieval operations.

2. View and Controller Classes

- **SignUpFormBuilder and LoginFormBuilder**: These classes are responsible for creating the forms for user registration and login.

- **FriendProfileView and ExploreView**: These seem to be UI components that allow users to explore other profiles and the broader social network.

- **Home**: The main/home page view of your social media platform.

3. Additional Features

- **SocialMediaApp**: The main driver class that initializes and starts the application.

Functional Requirements

1. User Account Management

- **Registration**: Users can create accounts using their username, and password.
- **Login/Logout**: Users can log in and out of the system with valid credentials.
- **Profile Management**: Users can edit their profile information, including bio & profile picture.

2. Social Interactions

- **Friend list**: Users can add, remove, and manage their friend list.
- **News Feed**: Display a feed of friends' recent posts and updates.
- **Post Creation**: Users can create text-based posts.
- **Comments and Likes**: Users can comment on and like posts.
- **Direct Messaging**: Users can send private messages to other users.

3. Content Management

- **Search Functionality**: Users can search for other users.

Non-Functional Requirements

1. Usability

- **User-Friendly Interface**: The GUI should be intuitive and easy to navigate for users of varying technical proficiency.

2. Reliability

- **Data Integrity**: Ensure data is accurately stored and retrieved without errors.

3. Security

- **Authorization**: Ensure users can only perform actions when they are successfully logged in.

4. Maintainability

- **Modularity**: The system should be developed in modular components to facilitate easier maintenance and upgrades.

UML Class Diagram

Check the appendix to view the diagram.

Networking:

- Attributes: users (a map of users).
- Methods: manage user registration, login, retrieval, and display of users.

User:

- Attributes: username, bio, profilePictureUrl, password, friends, posts, messages, followersCount.
- Methods: manage user data, friends, posts, messages, and profile information.

Post:

- Attributes: author, content, comments, likes, time.
- Methods: manage post creation, validation, commenting, liking, and retrieving post details.

Comment:

- Attributes: commenter, text, time.
- Methods: manage comment creation, validation, and retrieval.

Message:

- Attributes: sender, receiver, text, timestamp.
- Methods: manage message creation, text setting, and retrieval.

SocialMediaApp:

Attributes: networking, currentUser, loginScene, postsScene, postsLayout.

Methods: manage app startup, user login, registration, logout, and display of posts and alerts.

Various Views:

(chat view, friend profile view, user profile view, followers view, explore view, home, login form builder, signup form builder)

- Methods: manage the display and interaction of different parts of the application interface.

Database:

- Methods: update stored objects.

Relationships:

- Networking manages multiple users.
- User contains multiple posts, friends, messages, and followers.
- Post contains multiple comments and likes.
- Comment is linked to a user.
- Message is linked between sender and receiver.
- Various views use different components to display information and interact with the user interface.

Associations:

- Posts are associated with their authors, comments, and users who like them.
- Users interact with the application through various views and actions like posting, commenting, and messaging.

User Class

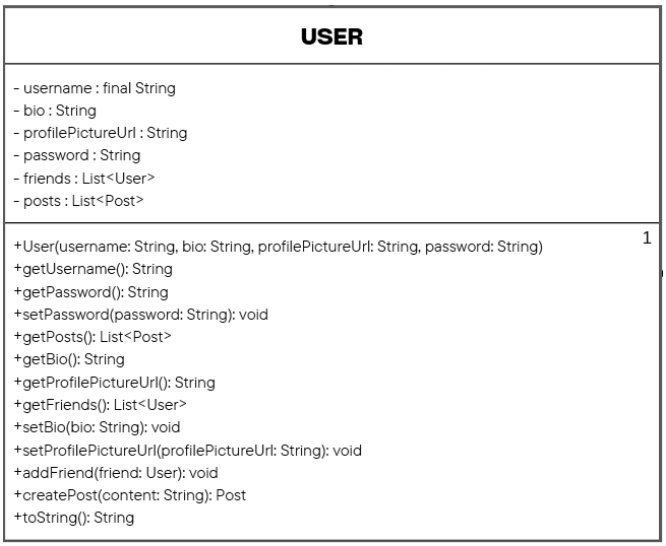


Figure 1 User Class

Attributes

username: String

- Represents the user's unique identifier.

bio: String

- A short biography or description of the user.

profilePictureUrl: String

- URL of the user's profile picture.

password: String

- User's password for authentication.

friends: List<User>

- A list of the user's friends, which are instances of the User class.

posts: List<Post>

- A list of posts made by the user, which are instances of the Post class.

messages: List<Message>

- A list of messages sent or received by the user, which are instances of the Message class.

Benefits of Each Method

Constructor (User)

- Initializes a new user with the required attributes, ensuring that each user has a unique username, bio, profile picture URL, and password. Essential for authentication, ensuring that the provided username and password match the stored values, thereby securing access.

getUsername, getPassword:

- These getter methods allow access to private attributes, facilitating encapsulation and controlled access to user information.

setPassword:

- Provides a way to update the user's password securely.

getPosts:

- Allows retrieval of all posts made by the user, useful for displaying the user's activity.

getBio, getProfilePictureUrl:

- Enables access to the user's bio and profile picture URL.

getFriends:

- Allows access to the user's friend list, enabling social connections within the application.

setBio, setProfilePictureUrl:

- These setter methods allow updating the user's bio and profile picture.

addFriend:

- Facilitates the addition of new friends, promoting social networking features within the application.

createPost:

- Allows users to create new posts, adding to their content and activity on the platform.

getMessages:

- Provides access to the user's messages, essential for communication features within the application.

toString:

- Offers a string representation of the user, useful for debugging and displaying user information in a readable format.

Network Class

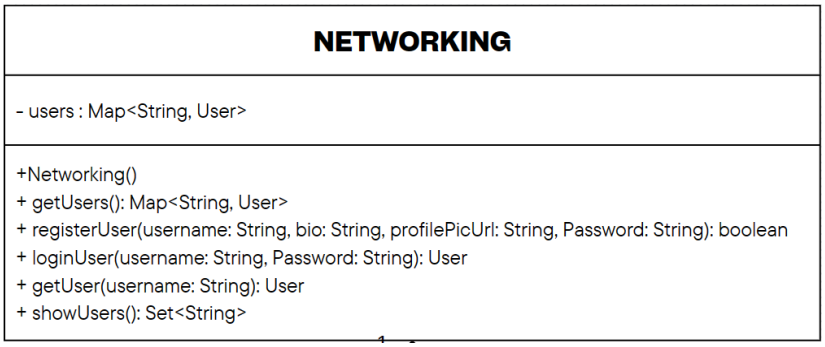


Figure 2 Networking Class

Attributes

users: Map<String, User>

- A map that stores user objects, with usernames as keys and User objects as values.

Benefits of Each Method

Constructor (Networking):

- Initializes the Networking instance, setting up the data structures for managing users.

getUsers:

- Provides access to the entire map of users, which can be useful for displaying all users.

registerUser:

- Facilitates the creation of new user accounts, ensuring that each user has a unique username and that their information is correctly stored. Returning a boolean helps handle cases where registration might fail (e.g., if the username is already taken).

loginUser:

- Handles user authentication by validating the provided username and password. Returns the User object upon successful login to allow access.

getUser:

- Enables retrieval of a specific user's details using their username for accessing user profiles.

showUsers:

- Returns a set of all usernames, which can be useful for features like search, listing users, or suggesting friends. This method ensures that only the usernames (not the sensitive details) are exposed.

Comment Class

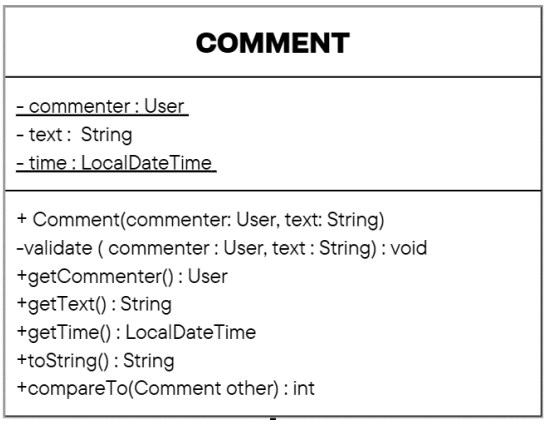


Figure 3 Comment Class

Attributes

commenter: User

text: String

- The actual content made by the user.

time: LocalDateTime

- The timestamp indicating when the comment was made.

Benefits of Each Method

Constructor (Comment):

- Initializes a new comment with the necessary attributes, ensuring that each comment has an associated user and content. Ensures that the commenter's identity is valid and that the comment content meets certain criteria (e.g., not empty, not offensive).

getCommenter:

- Provides access to the User object representing the commenter, allowing the application to retrieve additional information about the user.

getText:

- Returns the text content of the comment, which is necessary for displaying the comment in the application's user interface.

getTime:

- Returns the timestamp of when the comment was made.

toString:

- Provides a string representation of the comment, which can be useful for debugging, logging, or displaying the comment in a simple text format.

compareTo:

- Allows comments to be compared, typically based on their timestamps.

Post Class

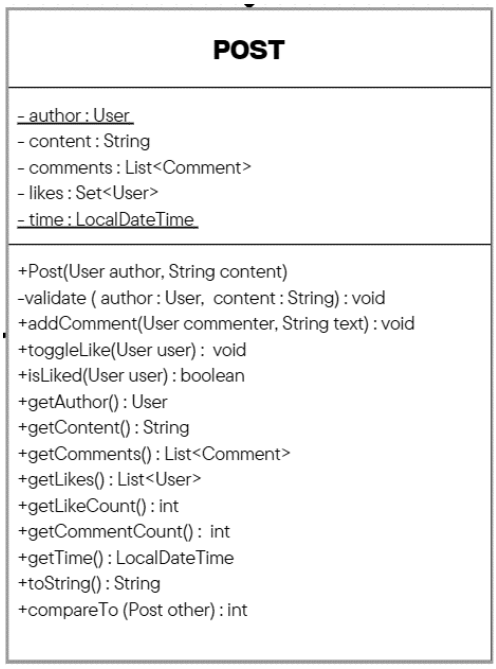


Figure 4 Post Class

Attributes

author: User

- The user who created the post.

content: String

- The main body of the post. This is a text string containing the content.

comments: List<Comment>

- A list of comments made on the post. Each item in the list is a comment object containing its own attributes like text and time.

likes: Set<User>

- A set of users who have liked the post. The use of a set ensures that each user is represented only once, no matter how many times they attempt to like the post.

time: LocalDateTime

- The timestamp when the post was created.

Benefits of Each Method

Constructor (Post):

- Initializes a new post with the necessary attributes, ensuring that each post has an associated author and content. Ensures that the author's identity is valid and that the post content meets certain criteria (e.g., not empty, not offensive).

addComment:

- Allows users to add comments to a post.

toggleLike:

- Enables users to like or unlike a post, facilitating engagement and feedback mechanisms.

isLiked:

- Checks if a specific user has liked the post, to display the like status to the user or for implementing features like showing who liked a post.

getAuthor:

- Provides access to the User object allowing the application to retrieve additional information about the author or display the author's profile alongside the post.

getContent:

- Returns the text content of the post, which is necessary for displaying the post.

getComments:

- Returns the list of comments associated with the post, enabling the application to display all comments and manage interactions on the post.

getLikes:

- Returns the list of users who have liked the post, which can be used to display like counts or show who has engaged with the post.

getLikeCount:

- Provides the total number of likes on the post, which is useful for displaying the popularity or engagement level of the post.

getCommentCount:

- Provides the total number of comments on the post.

getTime:

- Returns the timestamp of when the post was created.

toString:

- Provides a string representation of the post.

compareTo:

- Allows posts to be compared, typically based on their timestamps.

Task Breakdown

1. Project Planning and Requirements Analysis

- **Define Project Scope**: Identify the key features and boundaries of the project.
- **Gather Requirements**: Collect functional and non-functional requirements from stakeholders.
- **Resource Allocation**: Assign team members to various tasks based on their skills.

2. Design Phase

- **Architecture Design**: Define the overall system architecture, including technology stack and design patterns.
- **Database Design**: Design the schema for user data, posts, messages, and interactions.
- **Interface Design**: Sketch out the GUI layout and user flow for registration, login, news feed, etc.
- **Class Diagrams**: Create UML class diagrams detailing the relationships and dependencies.

3. Development Phase

- **Environment Setup**: Prepare development and testing environments.
- **Core Functionality Development**:
    - **User Management**: Implement registration, login, and profile management.
    - **Post Management**: Develop functionalities for creating, editing, deleting, and interacting with posts.
    - **Messaging System**: Build the infrastructure for direct messaging between users.
- **GUI Implementation**: Develop the front-end using JavaFX for interactive user interfaces.
- **Networking Implementation**: Set up basic client-server communication for data exchange and connect it to a database.
- **Integration**: Integrate various components (back-end and front-end) to ensure they work together seamlessly.

4. Testing Phase

- **Unit Testing**: Test individual components for expected functionality.
- **Integration Testing**: Ensure that integrated components work together as expected.
- **System Testing**: Conduct full system tests to validate compliance with requirements.
- **Performance Testing**: Evaluate the responsiveness, stability, and scalability of the application.
- **User Acceptance Testing (UAT)**: Allow potential users to test the system and provide feedback.

5. Maintenance and Upgrades

- **User Feedback Collection**: Gather feedback from users on functionality and usability.
- **Issue Resolution**: Address bugs and operational issues reported by users.
- **Feature Upgrades**: Implement new features and enhancements based on user feedback and new requirements.

6. Documentation and Reporting

- **Project Reporting**: Prepare final project reports and presentations detailing the development process, testing outcomes, and user feedback.

7. Project Closure

- **Review Project Outcomes**: Evaluate the success of the project against initial goals and objectives.
- **Final Deliverables Submission**: Submit all code, documentation, and reports.

Time Plan

Week 1: Project Initialization and Planning

Day 1-2: Project Kick-off

- Define project scope and objectives.
- Set up initial project documentation.
- Establish communication channels and tools.

Day 3-4: Requirement Analysis

- Conduct brainstorming sessions with the team.
- Gather functional and non-functional requirements.
- Document and prioritize requirements.

Day 5-6: Design

- Design UML diagrams and system architecture.
- Conduct design reviews with the team and incorporate feedback.
- Finalize the system architecture.

Day 7: Initial Project Setup

- Set up version control system (e.g., Git).
- Create initial project structure in the repository.
- Ensure all team members have access to the necessary tools and resources.

Week 2: Development Phase 1 - Core OOP Structure

Day 8-10: Core Class Implementation

- Implement core classes (User, Post, Comment, etc.) in Java.
- Develop essential methods and properties for these classes.
- Document class design and methods.

Day 11-12: Networking Infrastructure

- Set up basic client-server architecture for user communication.
- Implement initial network communication protocols.
- Document networking setup and protocols used.

Day 13-14: Initial Unit Testing

- Write unit tests for core classes.
- Execute unit tests and document results.
- Debug and refactor code based on test results.

Day 15: Code Review

- Conduct a code review session for the implemented classes and networking infrastructure.
- Address feedback and make necessary adjustments.

Week 3: Extended OOP Structure and Database Integration

Day 16-18: Advanced Class Implementation

- Implement additional classes and refine existing ones.
- Develop relationships between classes (e.g., associations, inheritances).

Day 19-20: Database Setup

- Design the database schema based on requirements.
- Set up the database and connect it to the project.
- Implement basic CRUD operations.

Day 21: Database Testing

- Write and execute tests for database operations.
- Ensure data integrity and correct functionality.

Week 4: GUI Development - Phase 1

Day 22-24: GUI Wireframing and Prototyping

- Develop wireframes and prototypes using tools like Figma or Sketch.
- Review prototypes with the team and iterate based on feedback.

Day 25-27: GUI Implementation - Basic Components

- Implement basic GUI components using JavaFX.
- Ensure navigation between different screens and views is functional.

Day 28: GUI Component Testing

- Write and execute tests for GUI components.
- Debug and refine GUI elements based on test results.

Week 5: GUI Development - Phase 2

Day 29-31: GUI Implementation - Advanced Components

- Implement advanced GUI components and interactions.
- Integrate multimedia elements if required.

Day 32-33: User Authentication

- Implement user authentication (login, registration).
- Ensure secure handling of user data.

Day 34-35: User Feedback Integration

- Conduct usability testing sessions with a small group of users.
- Gather feedback and make necessary improvements to the GUI.

Day 36: Mid-Project Review

- Review project progress with the team.
- Adjust timelines and goals if necessary based on feedback.

Week 6: Final Development and Testing

Day 37-39: Feature Completion

- Complete any remaining features.
- Ensure all functional requirements are met.

Day 40-42: Comprehensive Unit Testing

- Write and execute unit tests for all remaining components.
- Debug and refactor code based on test results.

Day 43-44: System Testing

- Conduct system-wide testing to ensure all components work together seamlessly.
- Identify and fix integration issues.

Week 7: Final Preparations

Day 45-47: User Acceptance Testing (UAT)

- Conduct UAT sessions with the team.
- Gather feedback and make final adjustments.

Day 48-49: Documentation Completion

- Finalize all project documentation, including user manuals and technical documentation.
- Ensure all documentation is up to date and accurate.

Day 50-52: Final Review and Sign-off

- Conduct a final project review with the team.
- Obtain project sign-off from all members.

Name	I.D	Contribution	Percentage
Abdallah Belal Momen	22P0036	Diagrams, Comment, Gui	20%
Ahmed Khaled Hamdy	22P0024	Database, Networking, Gui	20%
Malak Ossama Zaiter	22P0052	Messaging, Gui, Report	20%
Mohamed Khaled Mohamed	22P0041	Explore, Search, Manage Account, Gui, Report	20%
Yousef Sameh Shoman	22P0010	Manage Account, Add/Remove Friends, Gui, Report	20%

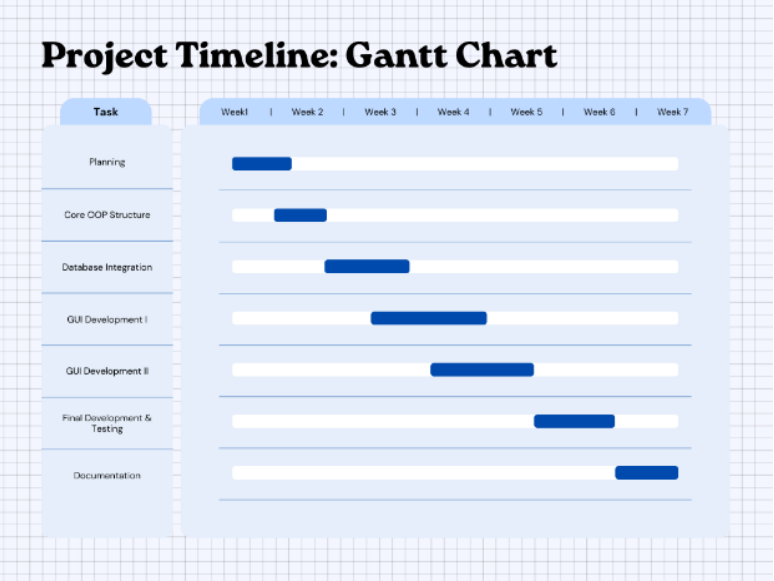


Figure 5 Gnatt Chart

System Architecture & Design

1. Presentation Layer (Client-Side GUI)

- **Technology**: JavaFX for building the graphical user interface.
- **Components**:
  1. **Login/Register View**: Handles user authentication and new user registration.
  2. **UserProfileView**: Allows users to view and edit their profile, including bio, profile picture, and friend list.
  3. **News Feed View**: Displays posts, comments, and likes from friends.
  4. **FriendProfileView**: Enables viewing of friends' profiles.
  5. **Messaging View**: For direct messaging between users, using a dedicated window.
  6. **ExploreView**: View trending posts according to like count.

2. Application Layer

- **Backend Framework**: Use Java for backend processing.
- **Components**:
  1. **User Management**: Handles business logic related to user activities like registration, profile management, and authentication.
  2. **Post Management**: Manages creation, modification, and deletion of posts and interactions like comments and likes.
  3. **Messaging System**: Handles sending, receiving, and storing messages.
  4. **Networking Manager**: Manages connections, data transfer, and synchronization between the client and server.
  5. **Session Management**: Maintains user session data for persistent state across different views.

3. Database Class

Purpose: Handles the persistence of data to a file using serialization.

- **Method:** update(Map object)
  - **Input:** A Map object to be serialized and saved to a file.
  - **Functionality:** Serializes the provided Map object and writes it to a file named data.txt. Uses FileOutputStream and ObjectOutputStream for file operations. Catches and handles IOException.

Design Details

Serialization

- **Purpose**: To save and load the state of the users' map.
- **Classes Used**:
    - **FileOutputStream**: Writes data to a file.
    - **ObjectOutputStream**: Serializes Java objects.
    - **FileInputStream**: Reads data from a file.
    - **ObjectInputStream**: Deserializes Java objects.

Exception Handling

- **Handled Exceptions**:
    - **IOException**: Caught during file operations to handle errors related to file access and I/O issues.
    - **ClassNotFoundException**: Caught during deserialization to handle errors related to missing class definitions.

User Management

- **Registration**:
    - Ensures unique usernames by checking the users map before adding a new user.
    - Updates the persistent storage after successful registration.
- **Login**:
    - Validates user credentials by comparing the provided password with the stored password.
- **Data Retrieval**:
    - Provides methods to retrieve user information and a list of all registered usernames.

Test Scenarios & Results

1. User Registration

Scenario: A new user registers with a unique username, email, and password.

- **Test Steps:**
    1. Open the application.
    2. Navigate to the registration page.
    3. Enter a unique username, valid email, and password.
    4. Submit the registration form.
- **Expected Result:** The user is successfully registered, and a new account is created in the system.
- **Actual Result:** The user registration was successful, and the account was created. All entered data were stored correctly.
- **Status:** Passed

2. User Login

Scenario: An existing user logs in with correct credentials.

- **Test Steps:**
    1. Open the application.
    2. Navigate to the login page.
    3. Enter the registered username and password.
    4. Submit the login form.
- **Expected Result:** The user is authenticated and redirected to the homepage.
- **Actual Result:** The user was authenticated and successfully redirected to the homepage.
- **Status:** Passed

3. Post Creation

Scenario: A logged-in user creates a new post.

- **Test Steps:**
    1. Log in to the application.
    2. Navigate to the post creation section.
    3. Enter text content for the post.
    4. Submit the post.
- **Expected Result:** The post is created and displayed in the user's feed.
- **Actual Result:** The post was created and displayed correctly in the user's feed.
- **Status:** Passed

4. Commenting on a Post

Scenario: A user comments on an existing post.

- **Test Steps:**
    1. Log in to the application.
    2. Navigate to an existing post.
    3. Enter a comment in the comment field.
    4. Submit the comment.
- **Expected Result:** The comment is added to the post and displayed under the post.
- **Actual Result:** The comment was successfully added and displayed under the post.
- **Status:** Passed

5. Adding a Friend

Scenario: A user adds a friend.

- **Test Steps:**
    1. Log in to the application.
    2. Search for another user by username.
    3. Click the "Add Friend" button on the user's profile.
- **Expected Result:** Update followers list of the added user.
- **Actual Result:** The list was updated successfully.
- **Status:** Passed

6. Direct Messaging

Scenario: A user sends a direct message to a friend.

- **Test Steps:**
    1. Log in to the application.
    2. Navigate to the friend's profile.
    3. Open the messaging interface.
    4. Enter a message and send it.
- **Expected Result:** The message is delivered to the friend and displayed in the chat.
- **Actual Result:** The message was delivered and displayed in the chat correctly.
- **Status:** Passed

7. User Profile Update

Scenario: A user updates their profile information.

- **Test Steps:**
    1. Log in to the application.
    2. Navigate to the profile settings.
    3. Update the bio and profile picture.
    4. Save the changes.
- **Expected Result:** The profile information is updated and displayed correctly.
- **Actual Result:** The profile was updated and displayed correctly.
- **Status:** Passed

End-user guide

1. Getting Started

Login

Steps:

1. Open the Social Media Platform application.
2. Enter your username and password.
3. Click on the "Login" button to access your account.

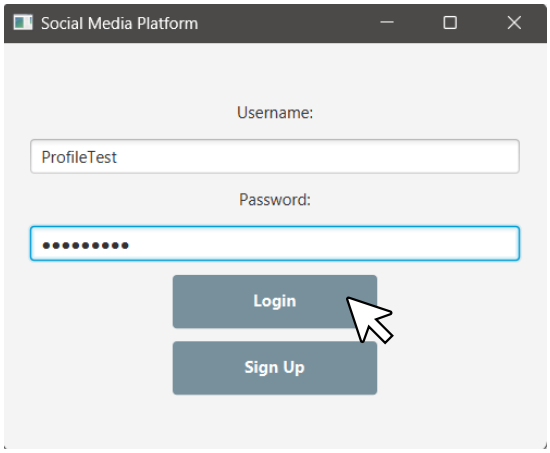
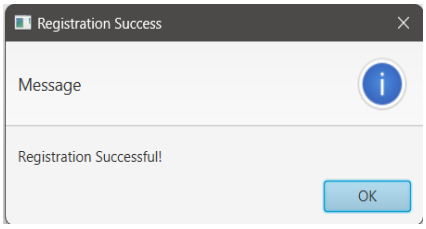


Figure 6 Login Page

Sign Up

Steps:

1. Click on the "Sign Up" button on the login page.
2. Fill in the required fields: Username, Password, Bio, and Profile Picture URL.
3. Click on the "Sign Up" button to create your account.



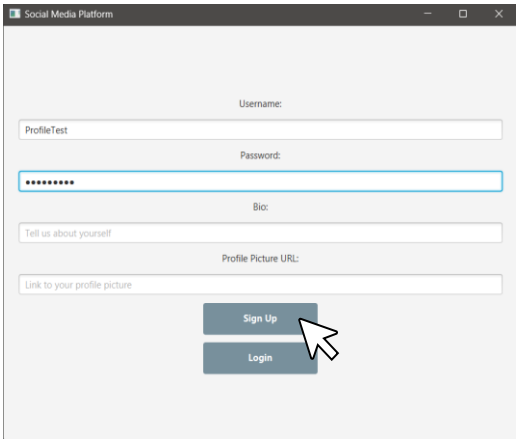


Figure 7 Sign Up Page

2. Main Interface

Features:

- Create a new post.

- Search for friends by username.

- Navigate using the "Explore", "Profile", and "Logout" buttons.

Creating a Post

Purpose: Share updates or thoughts with friends.

Steps:

1. Enter your post in the "What's on your mind?" field.
2. Click the "Create Post" button.

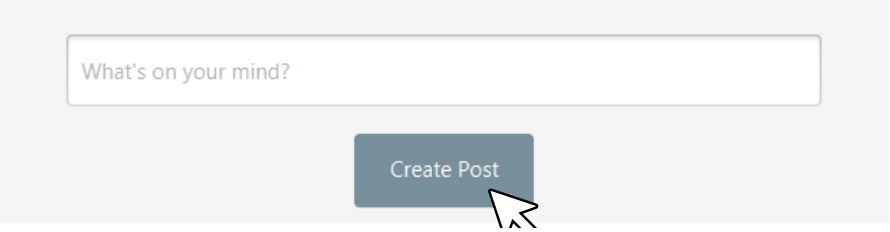


Figure 8 Creating Post

Interacting with Posts

Purpose: Like and comment on posts.

Steps:

1. To like a post, click on the "Like" button.
2. To comment, type your comment in the "Enter a comment" field and press "Comment".

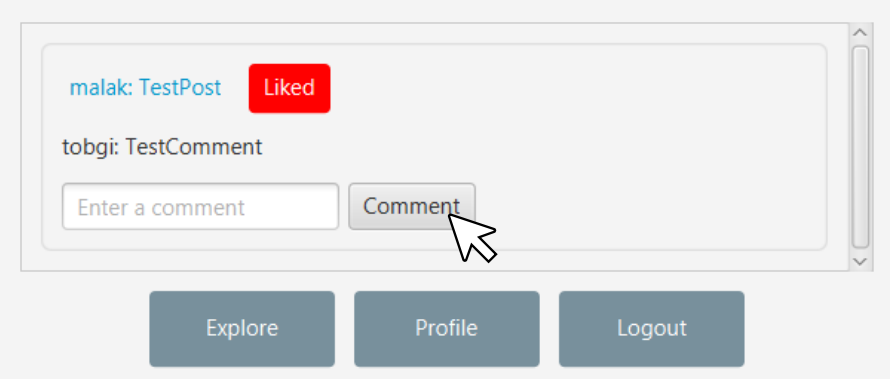


Figure 9 Interacting with a Post

View User Posts

Steps:

1. Press on the profile button.
2. Scroll to view your posts with their comments and number of likes.

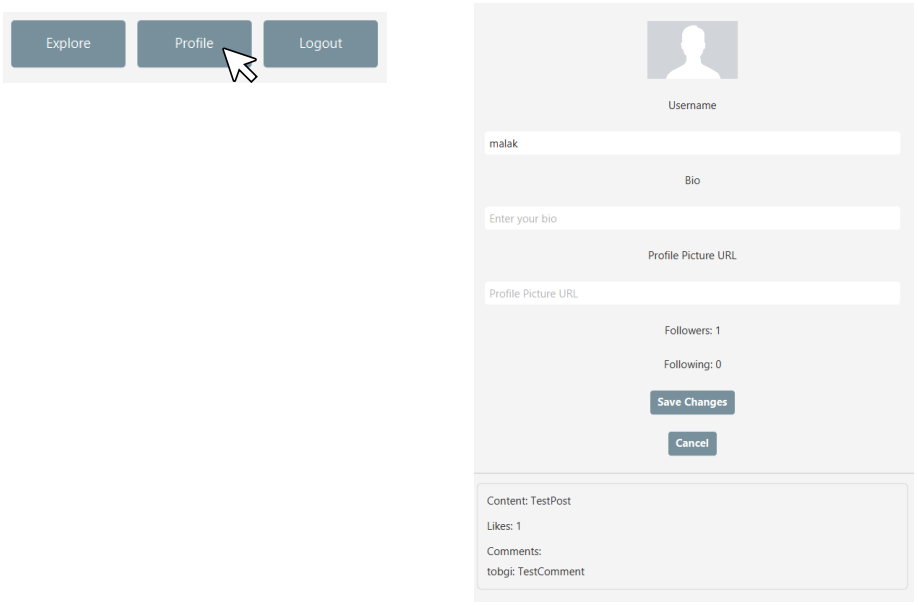


Figure 10 View User Profile

Managing Friends

Purpose: Add or remove friends from your network.

Steps:

1. Enter the username of the friend in the "Enter username to find friends" field.
2. Use the "Search" button to locate friends.
3. To add a friend, click on "Add Friend" on their profile.

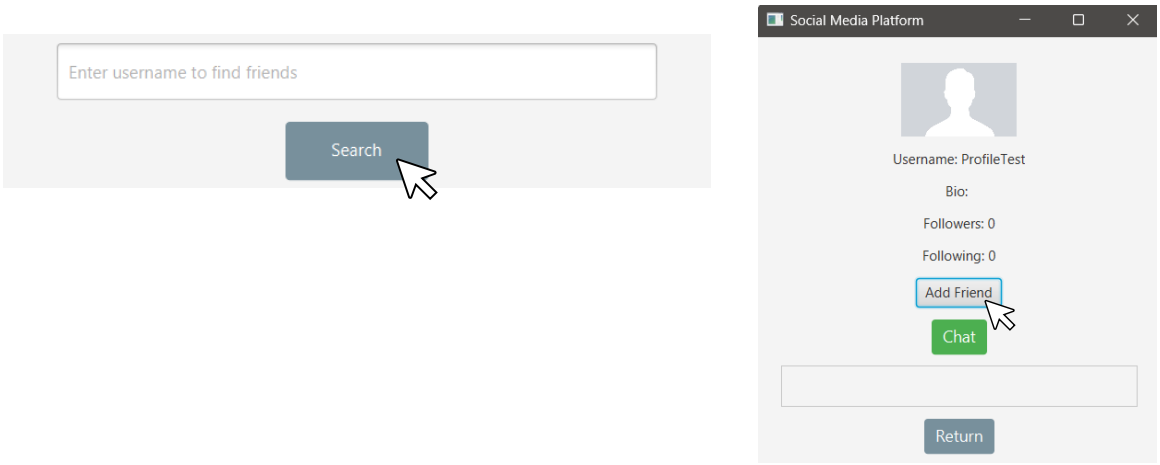


Figure 11 Managing User Friends

Messaging

Purpose: Send private messages to friends.

Steps:

1. Go to a friend's profile.
2. Click on the chat button.
3. Enter your message in the "Type your message here..." field.
4. Click on "Send Message" to deliver your message.

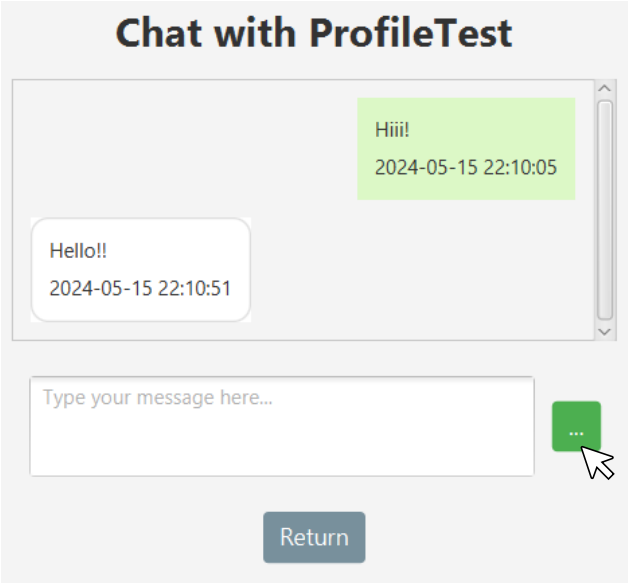


Figure 12 Messaging Friends

3. Navigation and Features

Explore: Discover new content and interact with posts from different users.

Profile: View your profile and update your bio or profile picture.

Logout: Securely exit your account.

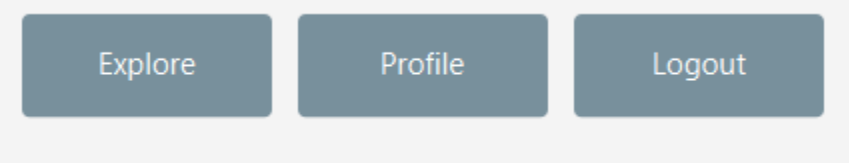


Figure 13 Navigation Options

Conclusion

The development of this social media platform has been a comprehensive exercise in applying advanced Java programming concepts and software engineering principles. By focusing on Object-Oriented Programming (OOP), graphical user interface (GUI) design, and basic networking, we have created a functional and user-friendly application.

This project not only offers users a platform to share content, connect with friends, and communicate but also provides developers with valuable experience in handling complex software design and implementation challenges. Through rigorous testing and user feedback, we ensured the platform's reliability, performance, and usability.

In summary, the project successfully integrates essential features such as user authentication, post creation, profile management, and a chat functionality, all within a well-structured and efficient system architecture. This foundation sets the stage for future enhancements, ensuring the platform can evolve and adapt to meet the growing needs of its user base.

Appendix

To view the whole UML class diagram please click the following link:
https://salmon-dulcie-51.tiiny.site

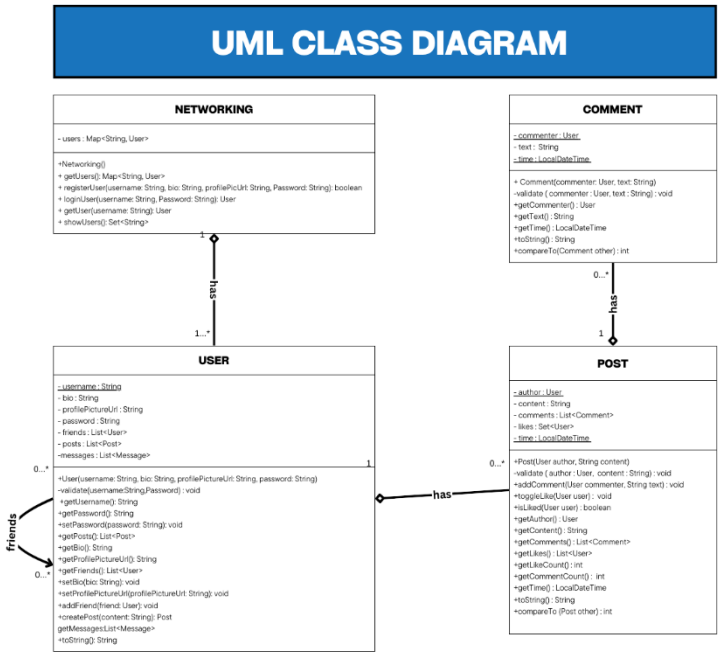


Figure 14 UML Class Diagram

References

1. JavaFX Official Documentation. (n.d.). Retrieved from https://openjfx.io/
2. Liang, D. (n.d.). *Introduction to Java Programming 10th Edition*.