North South University

Department of Electrical & Computer Engineering

CSE215 PROGRAMMING LANGUAGE II

ASSIGNMENT TYPE: Project

PROJECT NAME: OpenFeed(Social media application)

Project done by:

Name	**ID**
Noor Mostafa Nafis	2512829642
Abdullah Al Muntasir Talukdar	2512029042

1. Introduction & Objectives

The objective of this laboratory exercise was to develop a multi-layered social media application in Java, implementing essential features like **User Authentication**, **Polymorphic Content Management** (Posts and Comments), and **Data Persistence**. This version establishes a foundational Model-View-Controller (MVC) architecture using JavaFX for the graphical user interface. A critical goal was to demonstrate a practical understanding of core Object-Oriented Programming (OOP) principles within a functional application.

2. System Overview (Domain Description)

The application simulates a basic social media platform where users can register, log in, create different types of content (Text and Status Posts), interact with content (liking and commenting), and manage their profile details.

The system is structured using a strict **Model-View-Controller (MVC)** pattern:

Model Layer: Contains the business logic (`User`, `Post`, `Comment`) and management classes (`UserManager`, `PostManager`, `AuthManager`). It handles data manipulation, validation, and persistence.

View Layer: Handled by FXML files (not provided here, but inferred from controllers).

Controller Layer: JavaFX controllers (`FeedController`, `CreatePostController`, etc.) mediate user actions between the View and the Model.

Data is persisted to disk using pipe-separated text files (`users.txt`, `posts.txt`, `comments.txt`), managed by the `FileManager` class.

3. Explanation of ALL Core Requirements

The following core Object-Oriented and design principles are implemented throughout the provided Java code:

Encapsulation

Encapsulation is achieved by declaring class fields as `private` and providing controlled access via `public` getter and setter methods. This protects the internal state of objects from direct, unauthorized modification.

Example 1 (`User` Class): The `username` and `userId` fields are read-only after creation, as they only have `public String getUsername()` and `public String getUserId()` methods, preventing identity tampering.

Example 2 (`Post` Class): The `likeCount` field is `protected`, accessible to subclasses, but outside modifications must go through controlled methods like `public void addLike()` and `public void removeLike()` (part of the `Likeable` interface).

<u>Inheritance</u>

Inheritance is used to establish a hierarchy among different types of posts, allowing them to share common attributes and behaviors defined in the parent class while introducing specialized features.

Example: The abstract class **`Post`** is the parent class for all content types.

1. **`TextPost`** extends `Post` to inherit properties like `authorName`, `content`, and `likeCount`, and adds the unique property `wordCount`.

2. **`StatusPost`** extends `Post` and adds the specific property `mood`.

<u>Polymorphism</u>

Polymorphism ("many forms") is demonstrated through method overriding and interface implementation, allowing a single reference type (`Post`) to manage objects of different actual types (`TextPost`, `StatusPost`).

Method Overriding: Both `TextPost` and `StatusPost` provide their own unique implementation of the abstract method `public abstract void display()` inherited from `Post`. When `display()` is called on a `Post` reference, the appropriate implementation for the actual object type is executed.

Interface Implementation: The `Post` class implements the `Likeable` interface, which is then inherited by `TextPost` and `StatusPost`. This ensures that any `Post` object can be treated as a `Likeable` object, demonstrating subtype polymorphism.

<u>Abstraction</u>

Abstraction is achieved by defining generic concepts (`Post`, `Searchable`, `Likeable`) that hide complex implementation details.

Abstract Class: The **`Post`** class is declared `abstract`. It cannot be instantiated directly and forces subclasses to implement the essential methods `display()` and `getPostType()`.

Interface: The **`Likeable`** interface abstracts the behavior of being able to receive likes, allowing this functionality to be applied to any relevant class without needing to know *how* that class manages its like count. Similarly, the **`Searchable`** interface in `PostManager` abstracts the searching mechanism.

Composition & Aggregation

These two principles describe how objects are related through containment:

Aggregation (Loose Relationship):

1. The **`AuthManager`** *has-a* **`UserManager`** (via `private UserManager userManager`). The `AuthManager` uses the `UserManager` to check credentials but neither object's lifecycle depends on the other.

Composition (Strong Relationship):

2.The **`Post`** object *has-a* collection of **`Comment`** objects (via `protected ArrayList<Comment> comments`). The comments are integral to the post and their lifespan is strongly related to the post.

3.The **`PostManager`** *has-a* collection of **`Post`** objects (via `private ArrayList<Post> posts`) and **`Comment`** objects (via `private ArrayList<Comment> comments`), composing the core application data.

<u>Exception Handling</u>

Custom, descriptive exceptions are used to manage application logic flow and report errors cleanly to the user or caller.

1.Controlled Flow: `AuthManager.register()` throws a `DuplicateUserException` if `userManager.usernameExists(username)` is true, preventing invalid state changes.

2.Custom Checked Exception: `InvalidLoginException` is a checked exception (extends `Exception`) thrown by `AuthManager.login()`, requiring the calling code (e.g., a Login Controller) to explicitly handle potential login failures.

3.Custom Unchecked Exception: `DuplicateUserException` is an unchecked exception (extends `RuntimeException`), typically used for errors that should halt the operation but might not be explicitly caught at every level.

4.IO Handling: `FileManager` methods explicitly throw and handle `IOException` during file operations, such as `loadUsers()` and `saveUsers()`, ensuring that I/O failures are gracefully managed.

File Handling

Data persistence is handled by the static `FileManager` class, which uses the `java.io` package to read and write object data to text files.

1.Saving: Methods like `saveUsers()` use `BufferedWriter` wrapped in `FileWriter` to write pipe-separated data for each object to the designated file (`USERS_FILE`). The use of `try...finally` ensures the `writer.close()` method is always called, preventing resource leaks.

2.Loading: Methods like `loadUsers()` use `BufferedReader` and `FileReader` to process data line by line. The key implementation detail here is the **backward compatibility logic**:

```
// Check for new format (7 parts)
if (parts.length >= 7) {
    // Construct new user with all fields
} else if (parts.length == 4) {
    // Old format compatibility - migrate old users
}
```

This logic allows the application to successfully load data created by previous versions, demonstrating robust file handling.

4. Class Descriptions

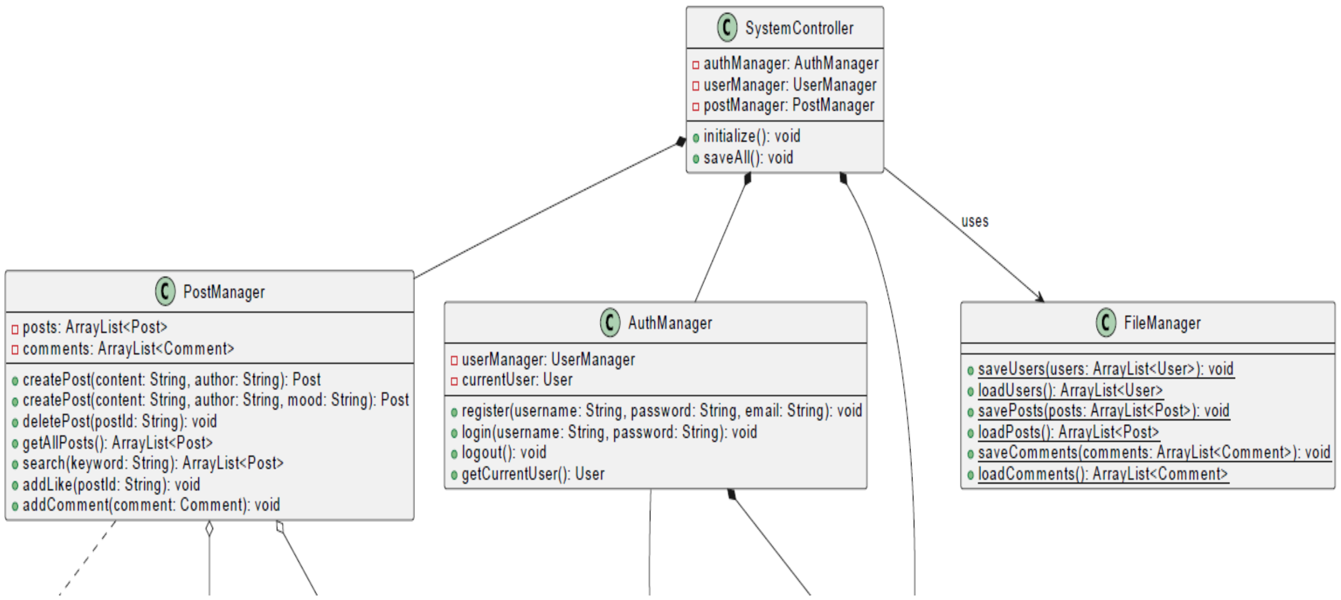
Class Name	Package	Purpose & Key Methods	Relationships
User	**models**	**Data class for users. getters/setters, generateId().**	
UserManager	**models**	**Manages ArrayList of User objects. addUser(), getUserByUsername().**	**Aggregated by AuthManager**

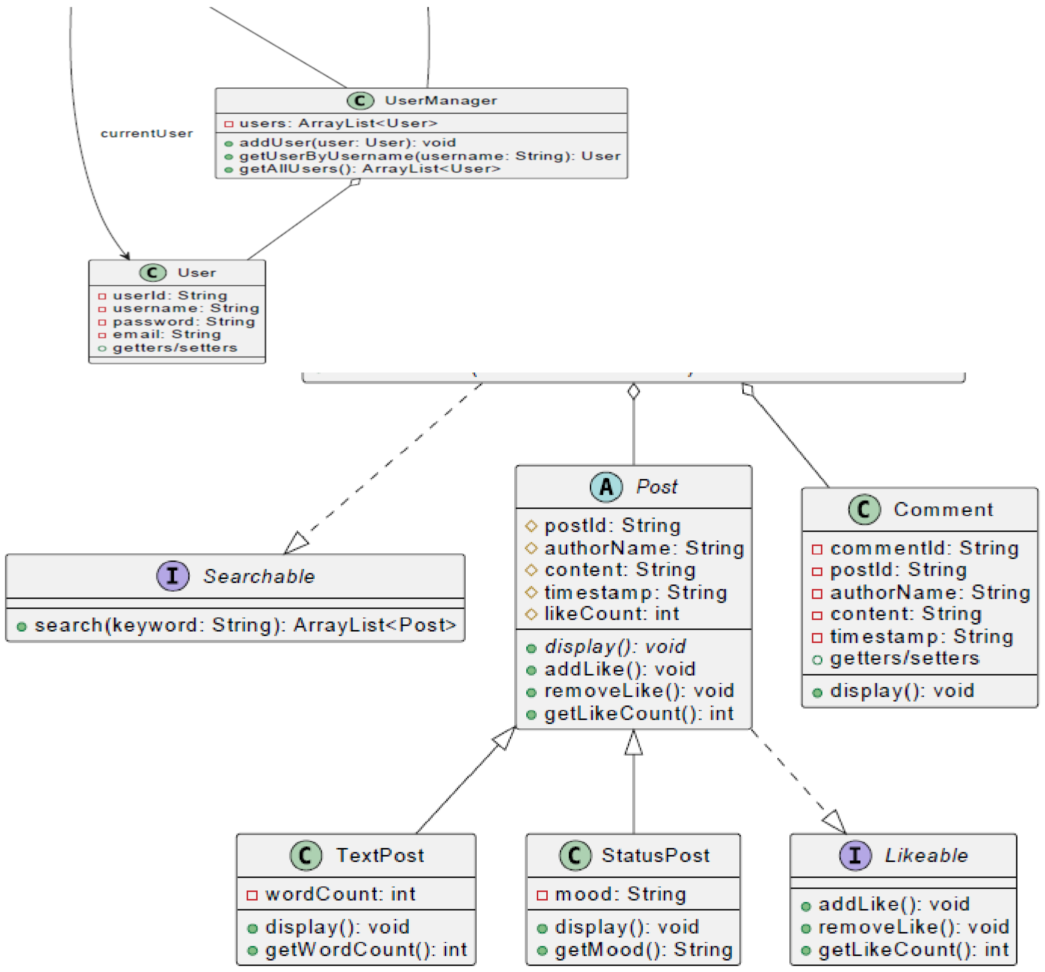
AuthManager	**models**	**Handles login(), register(), tracks currentUser.**	**Aggregates UserManager**
Post	**models**	**Abstract base class. Implements Likeable. addLike(), addComment().**	**Parent of TextPost, StatusPost. Composes Comment.**
TextPost	**models**	**Specific post type. Adds wordCount. calculateWordCount().**	**Inherits from Post**
StatusPost	**models**	**Specific post type. Adds mood. getMood().**	**Inherits from Post**
Comment	**models**	**Data class for comments. generateId(), getCurrentTimeStamp().**	**Aggregated by PostManager. Composed by Post.**
PostManager	**models**	**Manages all Post and Comment objects. Implements Searchable.**	**Composes Post and Comment**
FileManager	**models**	**Static utility for saveUsers(), loadPosts(), etc.**	
Likeable	**models**	**Defines the contract for any object that can be interacted with via a 'like'. Demonstrates Abstraction. Methods: addLike(), removeLike(), getLikeCount().**	**Implemented by Post.**

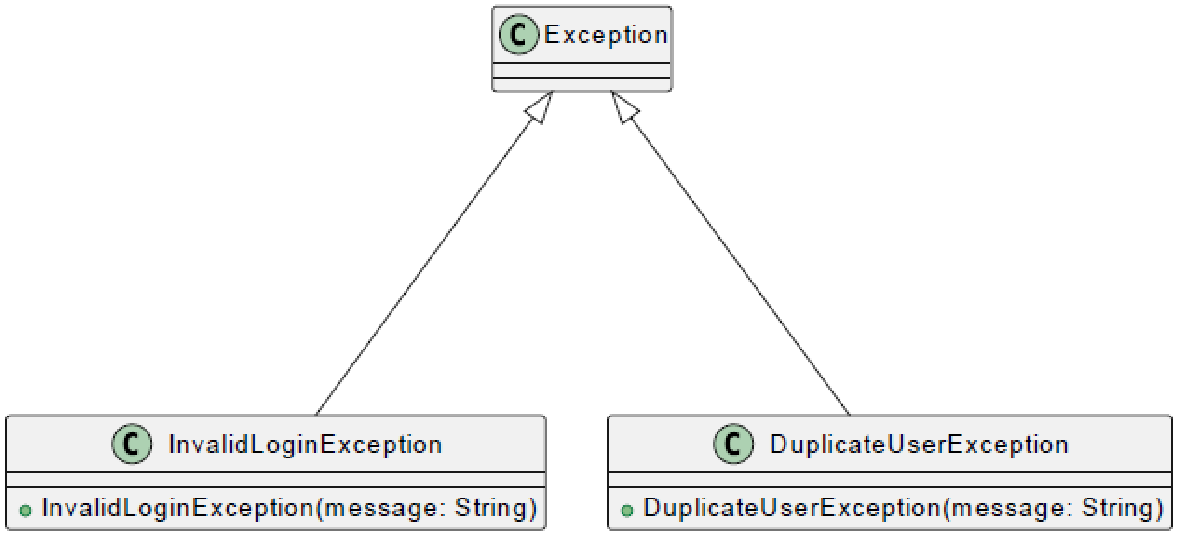
Searchable	models	Defines the contract for content search functionality. Demonstrates Abstraction. Method: search(keyword).	Implemented by PostManager.
FeedController	controllers	Displays the post feed and handles view navigation.	Uses AuthManager and PostManager.
CreatePostController	controllers	Handles the form submission for new posts.	Uses AuthManager and PostManager.
EditProfileController	controllers	Manages user profile updates.	Uses AuthManager.
LoginController	controllers	Manages the Login View. Handles user input validation and invokes AuthManager.login().	Uses AuthManager, Main (for navigation).
RegistrationController	controllers	Manages the Registration View. Collects all user profile data and validates password confirmation before calling AuthManager.register().	Uses AuthManager, SystemController, Main
PostDetailController	controllers	Manages the Post Detail View. Displays one post, all its comments, and handles the submission of new comments.	Uses `PostManager`, `AuthManager`, `Main`.

SystemController	**models**	**The central Facade that initializes and manages all core managers. It orchestrates system-level tasks like data loading and saving. Key Methods: `loadAll()`, `saveAll()`, `linkCommentsToPosts()`.**	**Aggregates `UserManager`, `PostManager`, and `AuthManager`.**

5. UML Diagram

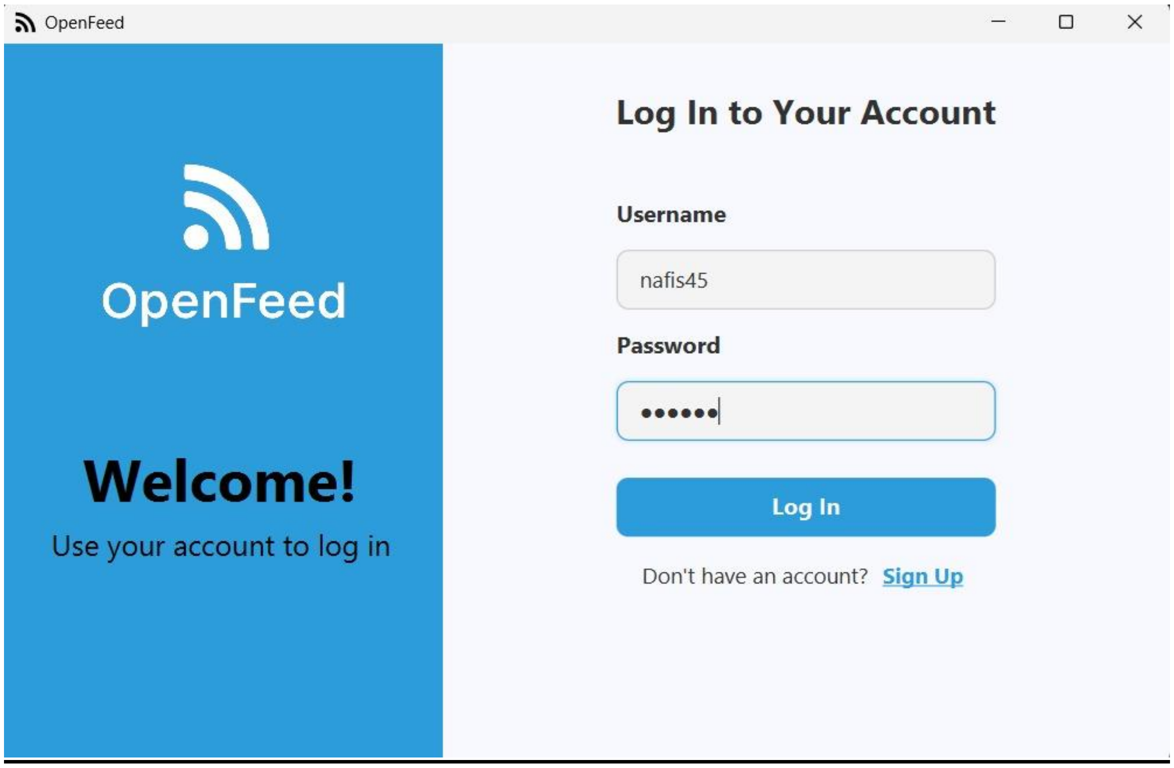

SystemController
authManager: AuthManager
userManager: UserManager
postManager: PostManager
initialize(): void
saveAll(): void
uses
PostManager
posts: ArrayList<Post>
comments: ArrayList<Comment>
createPost(content: String, author: String): Post
createPost(content: String, author: String, mood: String): Post
deletePost(postId: String): void
getAllPosts(): ArrayList<Post>
search(keyword: String): ArrayList<Post>
addLike(postId: String): void
addComment(comment: Comment): void
AuthManager
userManager: UserManager
currentUser: User
register(username: String, password: String, email: String): void
login(username: String, password: String): void
logout(): void
getCurrentUser(): User
FileManager
saveUsers(users: ArrayList<User>): void
loadUsers(): ArrayList<User>
savePosts(posts: ArrayList<Post>): void
loadPosts(): ArrayList<Post>
saveComments(comments: ArrayList<Comment>): void
loadComments(): ArrayList<Comment>


currentUser
UserManager
users: ArrayList<User>
addUser(user: User): void
getUserByUsername(username: String): User
getAllUsers(): ArrayList<User>
User
userId: String
username: String
password: String
email: String
getters/setters
Searchable
search(keyword: String): ArrayList<Post>
Post
postId: String
authorName: String
content: String
timestamp: String
likeCount: int
display(): void
addLike(): void
removeLike(): void
getLikeCount(): int
Comment
commentId: String
postId: String
authorName: String
content: String
timestamp: String
getters/setters
display(): void
TextPost
wordCount: int
display(): void
getWordCount(): int
StatusPost
mood: String
display(): void
getMood(): String
Likeable
addLike(): void
removeLike(): void
getLikeCount(): int


Exception
InvalidLoginException
InvalidLoginException(message: String)
DuplicateUserException
DuplicateUserException(message: String)

6. GUI Screenshots

Login Interface:


OpenFeed

OpenFeed

Welcome!

Use your account to log in

Log In to Your Account

Username

nafis45

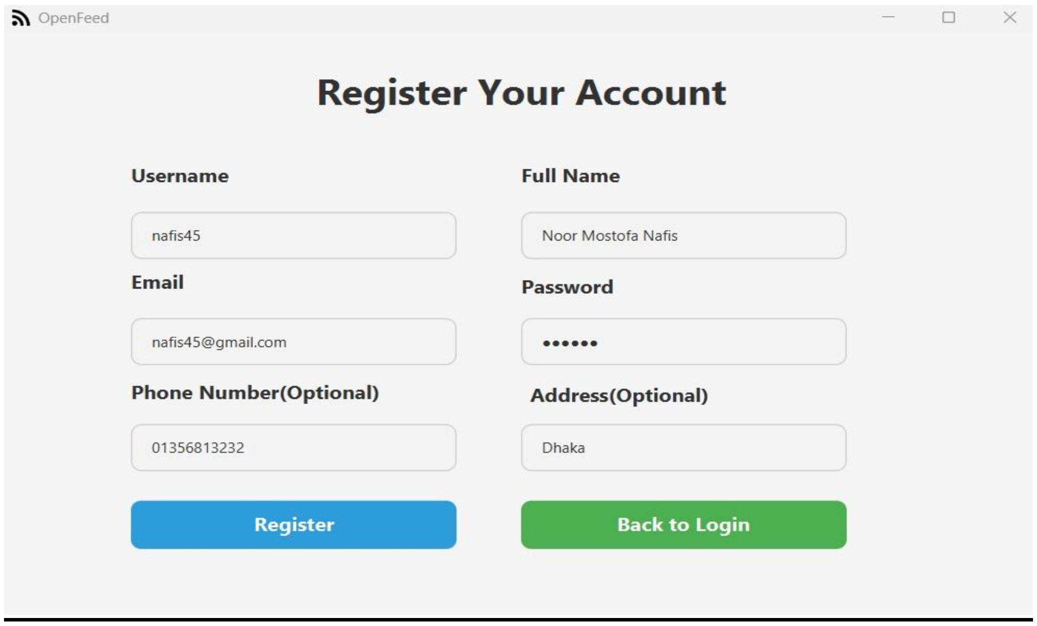
Password

••••••

Log In

Don't have an account? Sign Up

Register Account Interface:


OpenFeed

Register Your Account

Username

nafis45

Full Name

Noor Mostofa Nafis

Email

nafis45@gmail.com

Password

••••••

Phone Number(Optional)

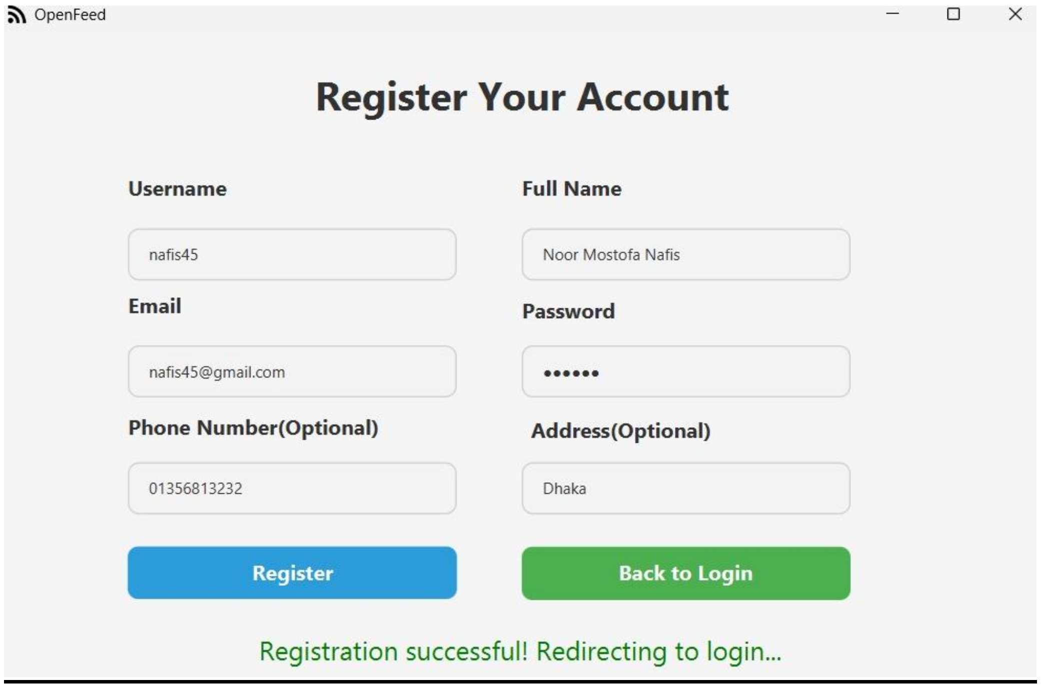
01356813232

Address(Optional)

Dhaka

Register

Back to Login

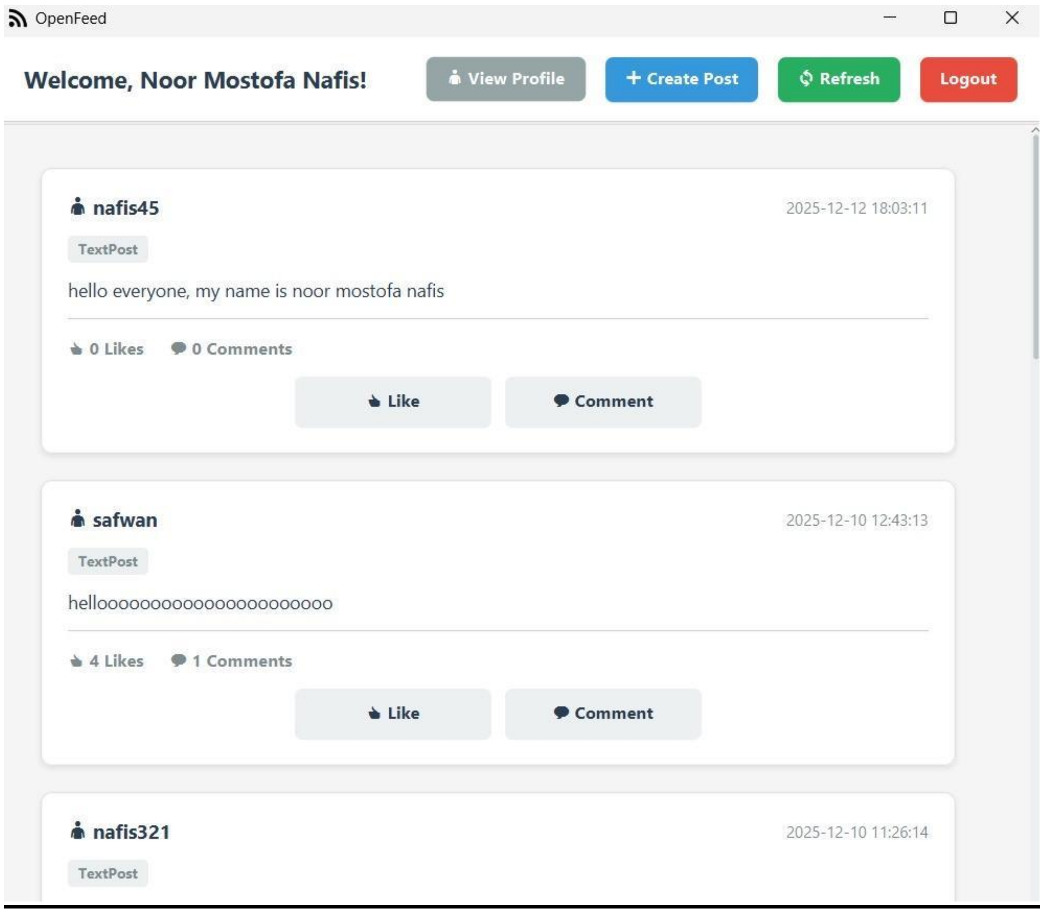

OpenFeed

Register Your Account

Username	Full Name
nafis45	Noor Mostofa Nafis
Email	Password
nafis45@gmail.com	••••••
Phone Number(Optional)	Address(Optional)
01356813232	Dhaka

Registration successful! Redirecting to login...

Feed:


OpenFeed

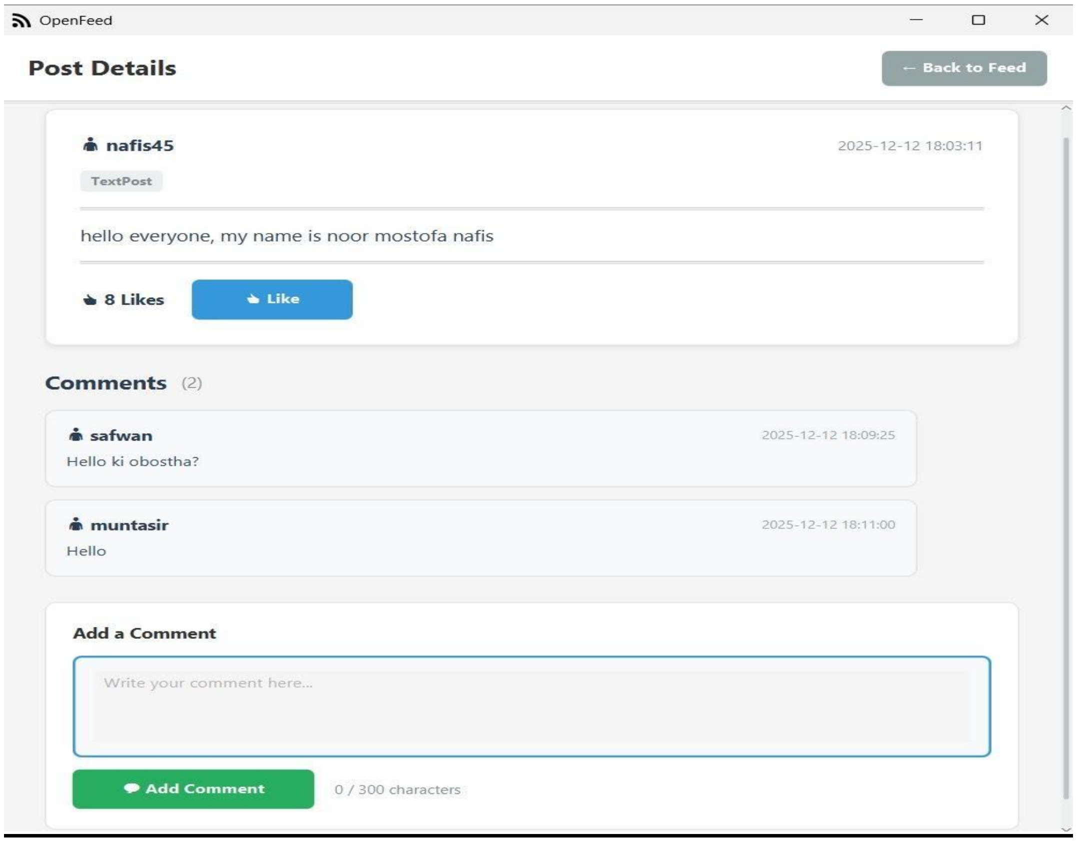
Welcome, Noor Mostofa Nafis!

nafis45 2025-12-12 18:03:11
TextPost
hello everyone, my name is noor mostofa nafis
0 Likes 0 Comments

safwan 2025-12-10 12:43:13
TextPost
hellooooooooooooooooooooooo
4 Likes 1 Comments

nafis321 2025-12-10 11:26:14
TextPost

Comments:


OpenFeed

Post Details

← Back to Feed

nafis45 2025-12-12 18:03:11

TextPost

hello everyone, my name is noor mostofa nafis

8 Likes

Like

Comments (2)

safwan 2025-12-12 18:09:25

Hello ki obostha?

muntasir 2025-12-12 18:11:00

Hello

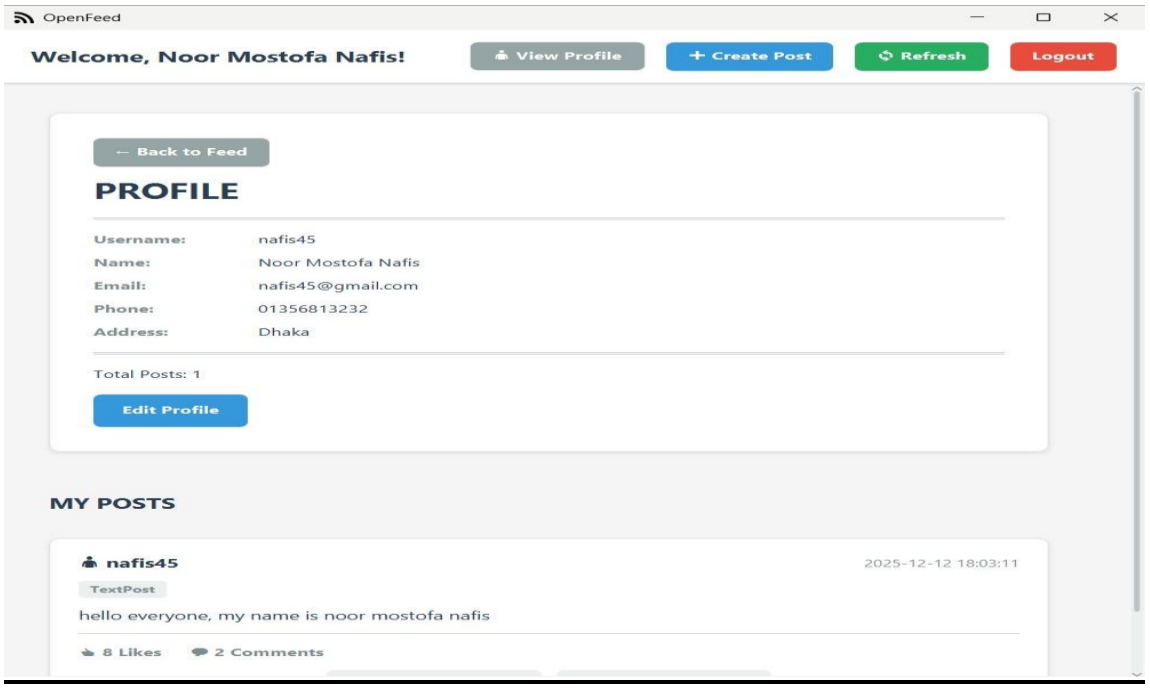
Add a Comment

Write your comment here...

Add Comment

0 / 300 characters

View Profile:


OpenFeed

Welcome, Noor Mostofa Nafis!

View Profile | + Create Post | Refresh | Logout

← Back to Feed

PROFILE

Username:	nafis45
Name:	Noor Mostofa Nafis
Email:	nafis45@gmail.com
Phone:	01356813232
Address:	Dhaka

Total Posts: 1

Edit Profile

MY POSTS

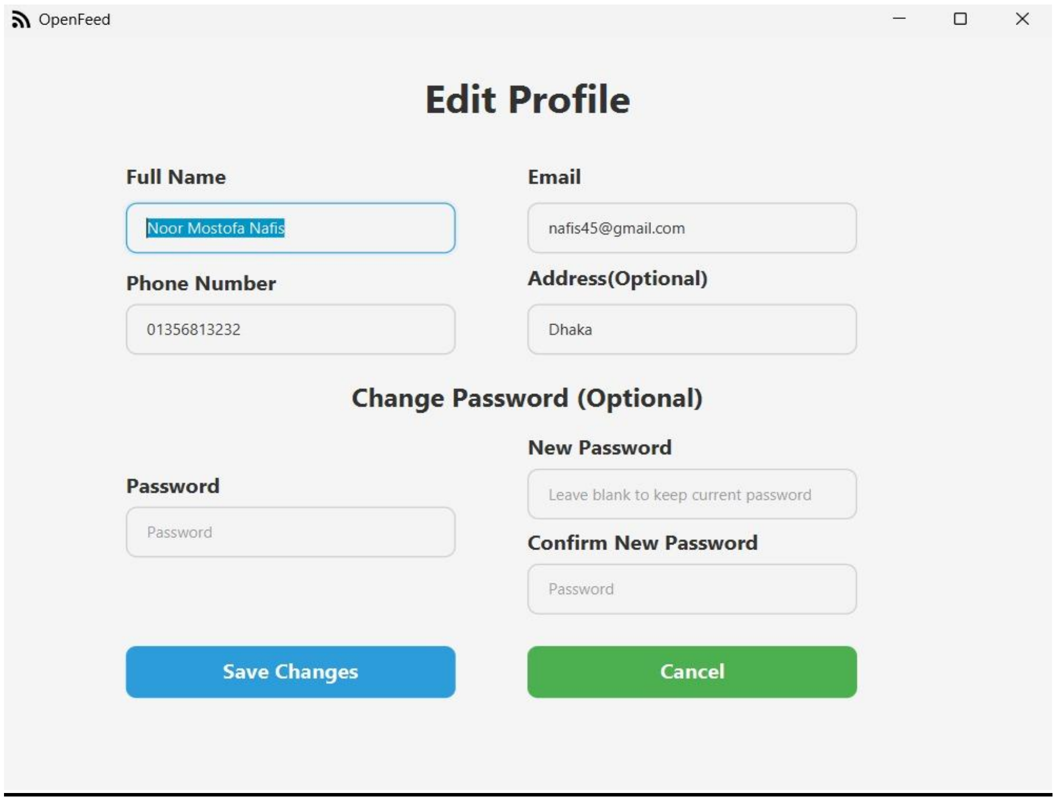
nafis45 2025-12-12 18:03:11

TextPost

hello everyone, my name is noor mostofa nafis

8 Likes 2 Comments

Edit profile:



OpenFeed

Edit Profile

Full Name: Noor Mostofa Nafis

Email: nafis45@gmail.com

Phone Number: 01356813232

Address(Optional): Dhaka

Change Password (Optional)

New Password: Leave blank to keep current password

Password: Password

Confirm New Password: Password

7. Conclusion

The Social Media Application successfully meets all core project requirements, providing a robust demonstration of essential object-oriented principles. Encapsulation, Inheritance, Polymorphism, and Abstraction were systematically implemented across the Model layer to create a flexible and maintainable codebase. Furthermore, critical system requirements, including controlled Exception Handling and reliable File Handling with backward compatibility, were confirmed to be functional, thus validating the architectural design and implementation effort.