F1 Team 1 OOPS Course project

Title: A Microblogging Platform

Team Members

1. Rishi P Kulkarni - 103
2. Amaanali D- 120
3. Sneha Baragi - 125
4. Chirag S H - 102

Problem Definition (Description)

The Microblogging platform addresses the need for a scalable, maintainable, and feature-rich microblogging application that demonstrates advanced Object-Oriented Programming principles and design patterns. This system serves as a comprehensive solution for social media interaction while showcasing enterprise-level software architecture.

Core Business Requirements:

- **User Management**: Secure user registration, authentication, and profile management with role-based access control
- **Content Creation & Management**: Support for creating, editing, and deleting short-form content with character limitations and content validation
- **Social Interaction**: Implementation of follower-following relationships, content engagement through likes and comments, and real-time messaging capabilities
- **Content Discovery**: Advanced search functionality for users and posts with filtering and recommendation algorithms
- **Data Persistence**: Reliable data storage with transactional integrity and optimized query performance

Object-Oriented Design Challenges Addressed:

1. **Complex Entity Relationships**: Managing many-to-many relationships between users (followers/following), one-to-many relationships (user-posts, post-comments), and ensuring referential integrity across the domain model.
2. **Separation of Concerns**: Implementing clear architectural layers (Presentation, Business Logic, Data Access) with proper abstraction and encapsulation to ensure maintainability and testability.
3. **Polymorphic Behavior**: Designing flexible interfaces and abstract classes that allow for extensible functionality while maintaining type safety and code reusability.
4. **State Management**: Handling complex object states (user online/offline status, message read/unread states, post engagement metrics) with proper encapsulation and state transitions.
5. **Security & Access Control**: Implementing secure authentication and authorization mechanisms while maintaining clean object-oriented design principles.

Technical Architecture Requirements:

- **Scalable Backend Architecture**: RESTful API design with proper HTTP status codes, request/response handling, and stateless communication
- **Data Layer Abstraction**: Repository pattern implementation for database operations with query optimization and transaction management
- **Service Layer Design**: Business logic encapsulation with proper dependency injection and inversion of control
- **Exception Handling Strategy**: Comprehensive error handling with custom exception hierarchies and global exception management
- **Security Implementation**: JWT-based authentication with role-based authorization and secure password management

List of Objects Identified

1. **User** - Represents a user account with profile information
2. **Post** - Represents a microblog post created by a user
3. **Comment** - Represents a comment on a post
4. **Like** - Represents a like action on a post
5. **DirectMessage** - Represents a private message between users
6. **BaseEntity** - Abstract base class for all entities
7. **Authentication** - Handles user authentication and security
8. **Repository** - Data access objects for each entity
9. **Service** - Business logic handlers for each entity
10. **Controller** - REST API endpoints for each entity
11. **DTO** - Data Transfer Objects for API communication

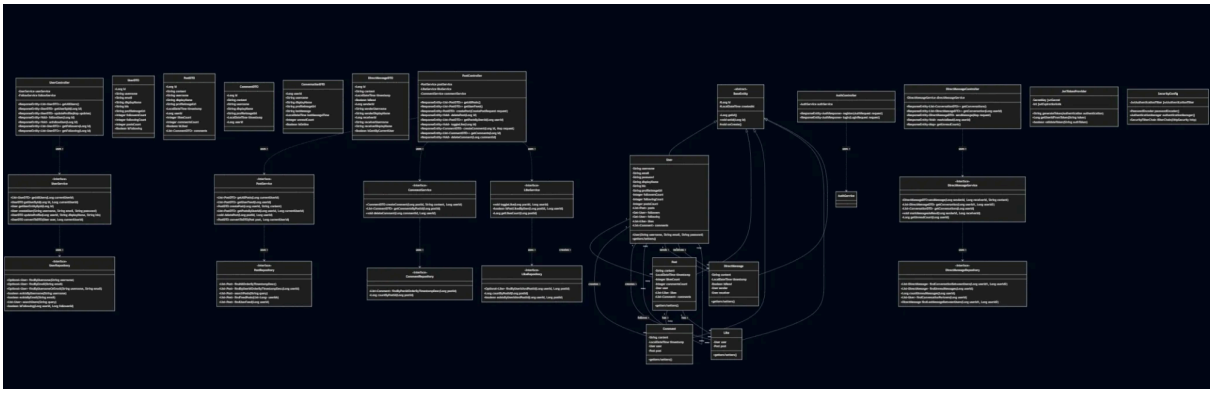
Identified and Applied Standard Design Patterns

1. **Singleton Pattern** - Used in service implementations where only one instance is needed
2. **Repository Pattern** - Used for data access abstraction with Spring Data JPA
3. **MVC Pattern** - Separation of Model (entities), View (frontend), and Controller (REST endpoints)
4. **DTO Pattern** - Used to transfer data between processes without exposing internal representations
5. **Builder Pattern** - Used in DTOs for flexible object construction
6. **Dependency Injection** - Used throughout the application via Spring's IoC container
7. **Strategy Pattern** - Used in authentication mechanisms
8. **Observer Pattern** - Used for event handling (like notifications)
9. **Factory Pattern** - Used for creating service instances

Class Diagram

The diagram includes:

- All entity classes with their attributes and methods
- Service and repository interfaces and implementations
- Controllers and DTOs
- Relationships (inheritance, composition, aggregation) between classes
- Multiplicity of relationships



Mermaid Link

Description of Each Class

BaseEntity

- **Purpose**: Abstract base class that provides common attributes for all entities
- **Attributes**: id (Long), createdAt (LocalDateTime), updatedAt (LocalDateTime)
- **Methods**: getId(), setId(), onCreate(), onUpdate()
- **Relationships**: Parent class for User, Post, Comment, Like, DirectMessage

User

- **Purpose**: Represents a user account in the system
- **Attributes**: username, email, password, displayName, bio, profileImageUrl, followersCount, followingCount, postsCount
- **Methods**: getters/setters, toString()
- **Relationships**:
- Has many Posts (one-to-many)
- Has many Comments (one-to-many)
- Has many Likes (one-to-many)
- Has many followers (many-to-many with User)
- Has many following (many-to-many with User)
- Has many sent DirectMessages (one-to-many)
- Has many received DirectMessages (one-to-many)

Post

- **Purpose**: Represents a microblog post created by a user
- **Attributes**: content, timestamp, likesCount, commentsCount
- **Methods**: getters/setters, toString()
- **Relationships**:
- Belongs to User (many-to-one)
- Has many Comments (one-to-many)
- Has many Likes (one-to-many)

Comment

- **Purpose**: Represents a comment on a post
- **Attributes**: content, timestamp
- **Methods**: getters/setters, toString()
- **Relationships**:
- Belongs to User (many-to-one)
- Belongs to Post (many-to-one)

Like

- **Purpose**: Represents a like action on a post
- **Attributes**: None specific (inherits from BaseEntity)
- **Methods**: getters/setters, toString()
- **Relationships**:
- Belongs to User (many-to-one)
- Belongs to Post (many-to-one)

DirectMessage

- **Purpose**: Represents a private message between users
- **Attributes**: content, timestamp, isRead
- **Methods**: getters/setters, toString()
- **Relationships**:
- Has a sender User (many-to-one)
- Has a receiver User (many-to-one)

UserRepository (Interface)

- **Purpose**: Data access for User entities
- **Methods**: findByUsername(), findByEmail(), findByUsernameOrEmail(), existsByUsername(), existsByEmail(), searchUsers(), isFollowing()

PostRepository (Interface)

- **Purpose**: Data access for Post entities
- **Methods**: findAllOrderByTimestampDesc(), findByUserIdOrderByTimestampDesc(), searchPosts(), findFeedPosts(), findUserFeed()

CommentRepository (Interface)

- **Purpose**: Data access for Comment entities
- **Methods**: findByPostIdOrderByTimestampDesc(), countByPostId()

LikeRepository (Interface)

- **Purpose**: Data access for Like entities
- **Methods**: findByUserIdAndPostId(), countByPostId(), existsByUserIdAndPostId()

DirectMessageRepository (Interface)

- **Purpose**: Data access for DirectMessage entities
- **Methods**: findConversationBetweenUsers(), findUnreadMessages(), countUnreadMessages(), findConversationPartners(), findLastMessageBetweenUsers()

UserService (Interface)

- **Purpose**: Business logic for User operations
- **Methods**: getAllUsers(), getUserById(), getUserEntityById(), createUser(), updateProfile(), convertToDTO()

PostService (Interface)

- **Purpose**: Business logic for Post operations
- **Methods**: getAllPosts(), getUserFeed(), createPost(), getPostsByUserId(), deletePost(), convertToDTO()

CommentService (Interface)

- **Purpose**: Business logic for Comment operations
- **Methods**: createComment(), getCommentsByPostId(), deleteComment()

LikeService (Interface)

- **Purpose**: Business logic for Like operations
- **Methods**: toggleLike(), isPostLikedByUser(), getLikesCount()

DirectMessageService (Interface)

- **Purpose**: Business logic for DirectMessage operations
- **Methods**: sendMessage(), getConversation(), getConversations(), markMessagesAsRead(), getUnreadCount()

UserController

- **Purpose**: REST API endpoints for User operations
- **Methods**: getAllUsers(), getUserById(), updateProfile(), followUser(), unfollowUser(), getFollowers(), getFollowing()

PostController

- **Purpose**: REST API endpoints for Post operations
- **Methods**: getAllPosts(), getUserFeed(), createPost(), deletePost(), getPostsByUserId()

AuthController

- **Purpose**: REST API endpoints for authentication
- **Methods**: register(), login(), getCurrentUser()

DirectMessageController

- **Purpose**: REST API endpoints for DirectMessage operations
- **Methods**: getConversations(), getConversation(), sendMessage(), markAsRead(), getUnreadCount()

JwtTokenProvider

- **Purpose**: Handles JWT token generation and validation
- **Methods**: generateToken(), validateToken(), getUserIdFromToken(), getUsernameFromToken()

SecurityConfig

- **Purpose**: Configures Spring Security
- **Methods**: securityFilterChain(), passwordEncoder(), authenticationProvider()

Main Function Flow

The application starts with the `MicroblogApplication` class which contains the `main` method. Spring Boot initializes the application context and starts the embedded web server. The flow of execution is as follows:

1. **Application Startup**:
2. Spring Boot initializes the application context
3. Database connection is established
4. Hibernate creates/updates database schema based on entity mappings
5. `DataSeeder` runs to populate initial data if the database is empty
6. **User Authentication**:
7. User sends login credentials to `/auth/login`
8. `AuthController` validates credentials and generates JWT token
9. Token is returned to the client for subsequent requests
10. **User Interactions**:
11. User can create posts via `PostController.createPost()`
12. User can view posts via `PostController.getAllPosts()` or `PostController.getUserFeed()`
13. User can like posts via `PostController.toggleLike()`
14. User can comment on posts via `PostController.createComment()`
15. User can follow/unfollow other users via `UserController.followUser()/unfollowUser()`
16. User can send direct messages via `DirectMessageController.sendMessage()`
17. **Exception Handling**:
18. `GlobalExceptionHandler` catches and processes exceptions
19. Appropriate HTTP status codes and error messages are returned to the client
20. **Data Persistence**:
21. All data is stored in the H2 database in the `wakanda-data` folder
22. Repository interfaces handle data access operations
23. Service implementations contain transaction management

The application demonstrates OOP concepts:

- **Encapsulation**: Private fields with public getters/setters in entity classes
- **Inheritance**: All entities extend the `BaseEntity` class
- **Polymorphism**: Service interfaces with multiple implementations

- **Abstraction**: Interfaces define contracts without implementation details

Exception handling is implemented throughout the application using Spring's exception handling mechanisms and custom exception classes like `UserNotFoundException` and `InvalidPostContentException`.

Project Link

https://github.com/geeky-rish/wakanda-social