# TEAM 1 PROJECT REPORT

## Title

**A Microblogging Platform**

### Team Details

| S.no | Name | USN | Roll no. |
|---|---|---|---|
| 1 | Rishi P Kulkarni | 01FE23BCI088 | 103 |
| 2 | Amaanali D | 01FE23BCI091 | 120 |
| 3 | Sneha Baragi | 01FE23BCI054 | 125 |
| 4 | Chirag S H | 01FE23BCI005 | 102 |

## Executive Summary

Wakanda Social is a microblogging platform inspired by the technologically advanced nation of Wakanda from the Marvel universe. Built using **Spring Boot**, it empowers users to express themselves through short posts, follow other users, engage in real-time messaging, and interact socially via comments and likes. The application demonstrates fundamental **Object-Oriented Programming (OOP)** principles and leverages multiple software design patterns. It features a dark UI theme with Wakandan aesthetics, a secure **JWT-based authentication system**, and robust back-end logic supported by an **H2 in-memory database**.

## Objectives

- **Domain Modeling**: Identify and model core entities like `User`, `Post`, `Comment`, `Like`, and `DirectMessage`.
- **OOP Principles**: Apply encapsulation, inheritance (via `BaseEntity`), abstraction (via service and repository interfaces), and polymorphism (interface-based service implementations).
- **Design Patterns**:
  - Singleton – Service classes

    - Strategy – Authentication mechanism
    - Observer – Notification and event systems
    - DTO – Safe API communication
    - MVC – Structured layering
    - Builder – Flexible DTO creation
    - Factory – Service instantiation logic
    - Repository – Abstract data access
    - Dependency Injection – Spring IoC container
- **Spring Boot**: RESTful API design using Spring MVC, authentication via Spring Security with JWT, and persistent in-memory H2 database.
- **Security**: JWT token-based authentication and role-based authorization.
- **Testing & Robustness**: Global exception handling with meaningful HTTP responses; modular and testable service-layer logic.

---

## System Architecture

- **Controllers**: Handle HTTP requests, validate inputs, and delegate to services.
- **Services**: Implement business logic and transaction management for each entity.
- **Repositories**: Spring Data JPA interfaces providing CRUD operations for entities.
- **DTOs**: Facilitate secure and efficient data exchange between backend and frontend.
- **Security Layer**: Manages token-based authentication using `JwtTokenProvider` and configures access control via `SecurityConfig`.
- **Database**: In-memory H2 database, auto-configured and schema-generated via Hibernate ORM.

---

# Feature Overview

| Feature | Endpoint / Description |
|---|---|
| **User Registration/Login** | `/auth/register`, `/auth/login` – JWT-based security system |
| **Create/View Posts** | `/posts/create`, `/posts/all`, `/posts/user/{id}` – Post microblogs (up to 280 chars) |
| **Like and Comment** | `/posts/{id}/like`, `/comments/create` – Interact on posts |
| **Follow/Unfollow** | `/users/{id}/follow`, `/users/{id}/unfollow` – Manage connections |
| **Direct Messaging** | `/messages/send`, `/messages/conversations` – Private chats between users |
| **User Feed** | `/posts/feed` – Personalized timeline |
| **Profile Updates** | `/users/updateProfile` – Edit display name, bio, etc. |
| **Search** | `/users/search`, `/posts/search` – Search for users and posts |
| **Exception Handling** | `GlobalExceptionHandler` – Handles errors like unauthorized access, invalid inputs |

# Key Classes & Responsibilities

| Class | Responsibility |
|---|---|
| `User`, `Post`, etc. | Represent core domain entities with JPA annotations |
| `BaseEntity` | Abstract superclass with audit fields (`id`, `createdAt`, `updatedAt`) |
| `UserService` | Manages profile, followers, user search |
| `PostService` | Manages post creation, feed generation |
| `LikeService`, `CommentService` | Handle likes and comments on posts |
| `DirectMessageService` | Manages messaging between users |
| `JwtTokenProvider` | Generates and validates JWTs |
| `SecurityConfig` | Configures Spring Security filters, providers |
| `UserController`, etc. | Handle API requests and responses |

## Application Flow

1. **Startup**:
    - Spring Boot initializes context
    - H2 database schema created via Hibernate
    - Optional seed data loaded
2. **Authentication**:
    - `/auth/login` – Client sends credentials
    - JWT token issued on success
    - Used in headers for subsequent API calls
3. **User Interaction**:
    - Posting, liking, commenting, following, messaging via controllers
    - Feed generated based on followed users
    - DirectMessage flow uses message read/unread logic
4. **Exception Handling**:
    - `@ControllerAdvice` with `GlobalExceptionHandler`
    - Returns structured error response with status codes

---

## Demonstrated OOP Concepts

| Concept | Implementation |
| --- | --- |
| **Encapsulation** | Private fields with public getters/setters in all entities |
| **Inheritance** | `BaseEntity` superclass extended by all entity classes |
| **Polymorphism** | Interfaces for services (e.g., `UserService`, `PostService`) with implementations |
| **Abstraction** | Interfaces define logic contract, implemented in service classes |

## Conclusion

Wakanda Social effectively combines modern web development practices with solid software engineering principles. It provides a rich, interactive user experience while highlighting how Spring Boot, design patterns, and clean

architecture can be applied in a real-world microblogging system. The application serves as a comprehensive showcase for OOP, RESTful API design, JWT security, and scalable component-based system architecture.

---