

# Friendstor

A Microblogging Social Media Platform

---



# CONTENTS

Introduction to Our Team

Project Name & Overview

Project Motivation &  
Objectives

Assignment Requirements

System Architecture  
Overview

Technology Stack Overview

Backend Framework:  
Spring Boot

Spring Boot Layered  
Architecture

Backend Technologies  
(Detailed)

Web Frontend (Desktop  
Application)

Mobile Frontend

Architecture & Design  
Patterns (Overview)

Observer Pattern  
(Notifications)

Repository & MVC Patterns

Singleton Pattern

SOLID Principles Applied

Development Tools

Core Features (Users &  
Posts)

Core Features (Feed &  
Notifications)

Core Features (User  
Interface)

Backend API Endpoints

Challenges Faced

Solutions Implemented

Conclusion



# 1. Introduction to Our Team

## Team Lovelacers

We are Team Lovelacers, a group of five students working together on the CS613 CA02 Group Project. Our team collaborated on the design, development, testing, and presentation of a full-stack social media platform, applying advanced object-oriented programming concepts and design principles.



## Team Members

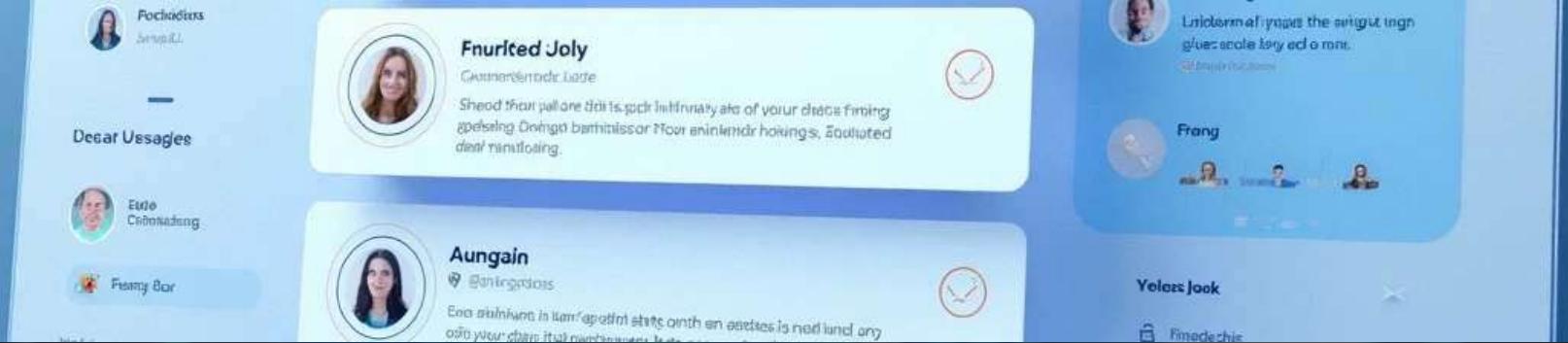
Chennuru Rahul

Varun Kodi

Hardik Negi

Prajjwal Thakur

Anandha Harini  
Selvaraj Kavitha



## 2. Project Name & Overview

### Friendstor: A Microblogging Social Media Platform

Friendstor is a Twitter-style microblogging social media platform that allows users to post short text messages, follow and unfollow other users, and receive notifications when other users follows them and when followed users post new content.

The project is developed as part of CS613 Advanced Concepts in Object-Oriented Programming, with a strong focus on clean architecture, reusability, and design patterns.

### 3. Project Motivation & Objectives

The main objectives of this project are to:

Build a lightweight alternative to Twitter

Allow users to create short text posts (maximum characters 250)

Enable follow and unfollow functionality

Display a personalized feed for each user

Provide instant notifications

Apply OOP concepts, SOLID principles, and design patterns

Support multiple client applications using a single backend

## 4. Assignment Requirements

The project satisfies all assignment requirements:

1

At least three applications:

- Service Application (Backend)
- Web/Desktop Application
- Mobile Application / Emulation

2

Users can follow and unfollow other users

3

Followers receive notification when other users follow them.

4

Followers receive notification when users they follow creates a post

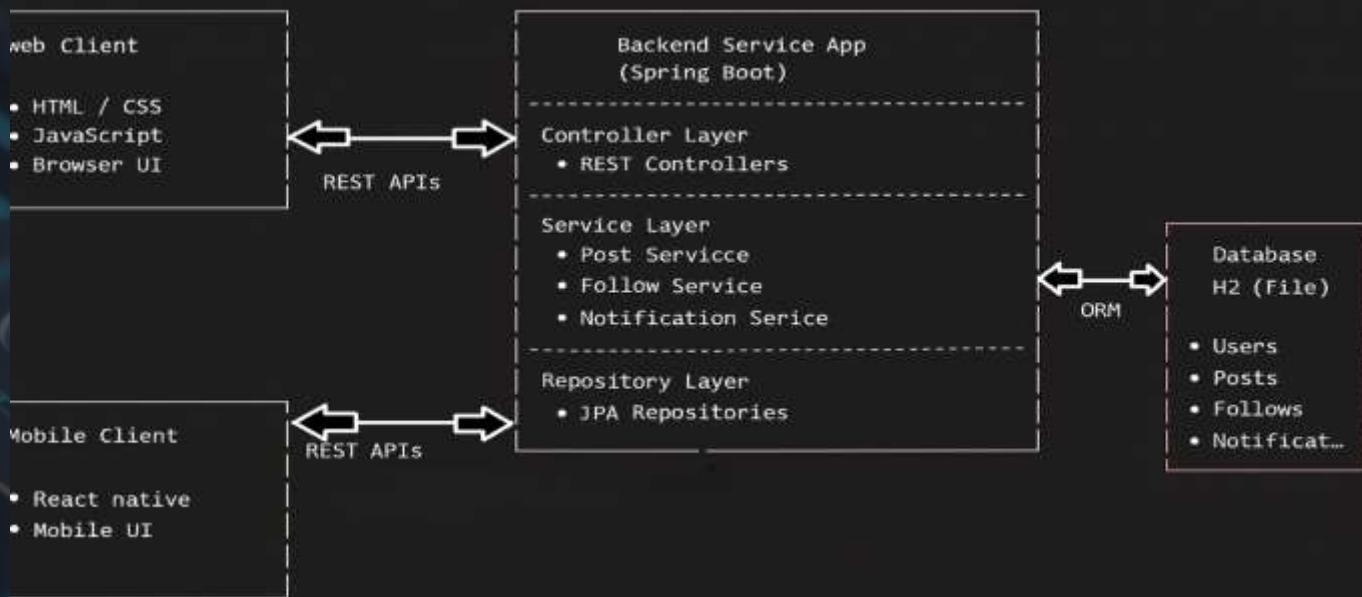
5

User is not be notified when another user unfollows them

6

Used H2 DB as Persistence provider

# 5. System Architecture Overview



Friendstor follows a multi-client architecture:

Backend Service Application\

Handles follow/unfollow services, feed generation, follow and posts notifications and data persistence

Web/Desktop Application\

Browser-based interface for interacting with the platform

Mobile Application\

Mobile client consuming the same backend APIs

All clients communicate with the backend using RESTful APIs.

# 6. Technology Stack Overview

1

Backend

Java, Spring Boot

2

Database

H2 (file-based persistence)

3

Web Frontend

HTML , CSS ,  
JavaScript

4

Mobile Frontend

React Native

5

Communication

RESTful APIs

6

Build Tool

Maven

7

IDE

VSCode, IntelliJ

8

Api Testing

Postman

9

Version Control

Git



# 7. Web Application Frontend Technologies

The web application is built using:

1

HTML for structure

2

CSS for styling and responsive design

3

Vanilla JavaScript for interaction

4

Python HTTP Server for serving static files

5

Runs on Port 3000

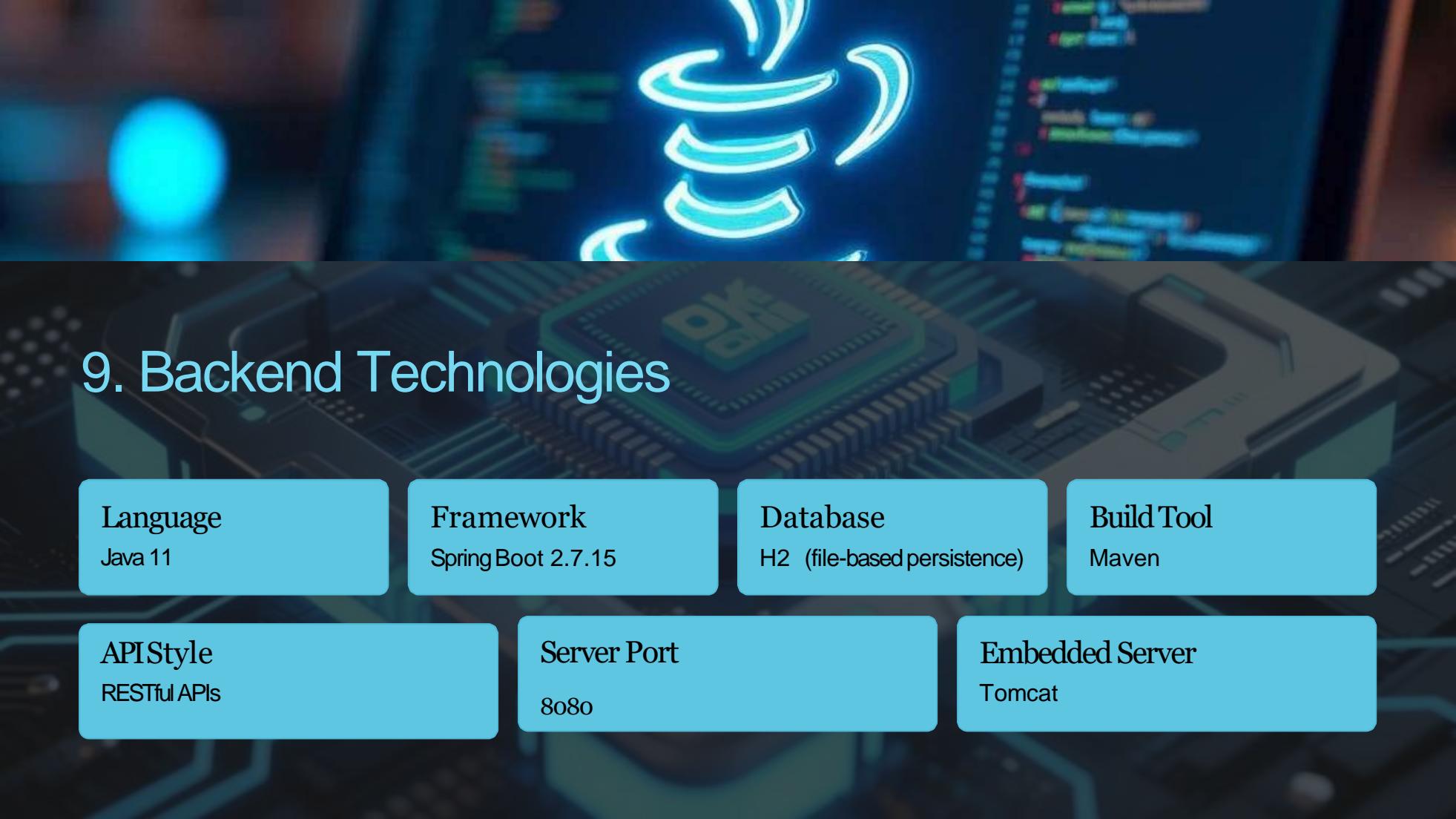
The UI focuses on simplicity and usability.





## 8. Mobile Application Frontend

The mobile application is implemented using React Native / Flutter and consumes the same REST APIs as the web application. This demonstrates backend reuse and platform independence.



## 9. Backend Technologies

Language

Java 11

Framework

Spring Boot 2.7.15

Database

H2 (file-based persistence)

Build Tool

Maven

API Style

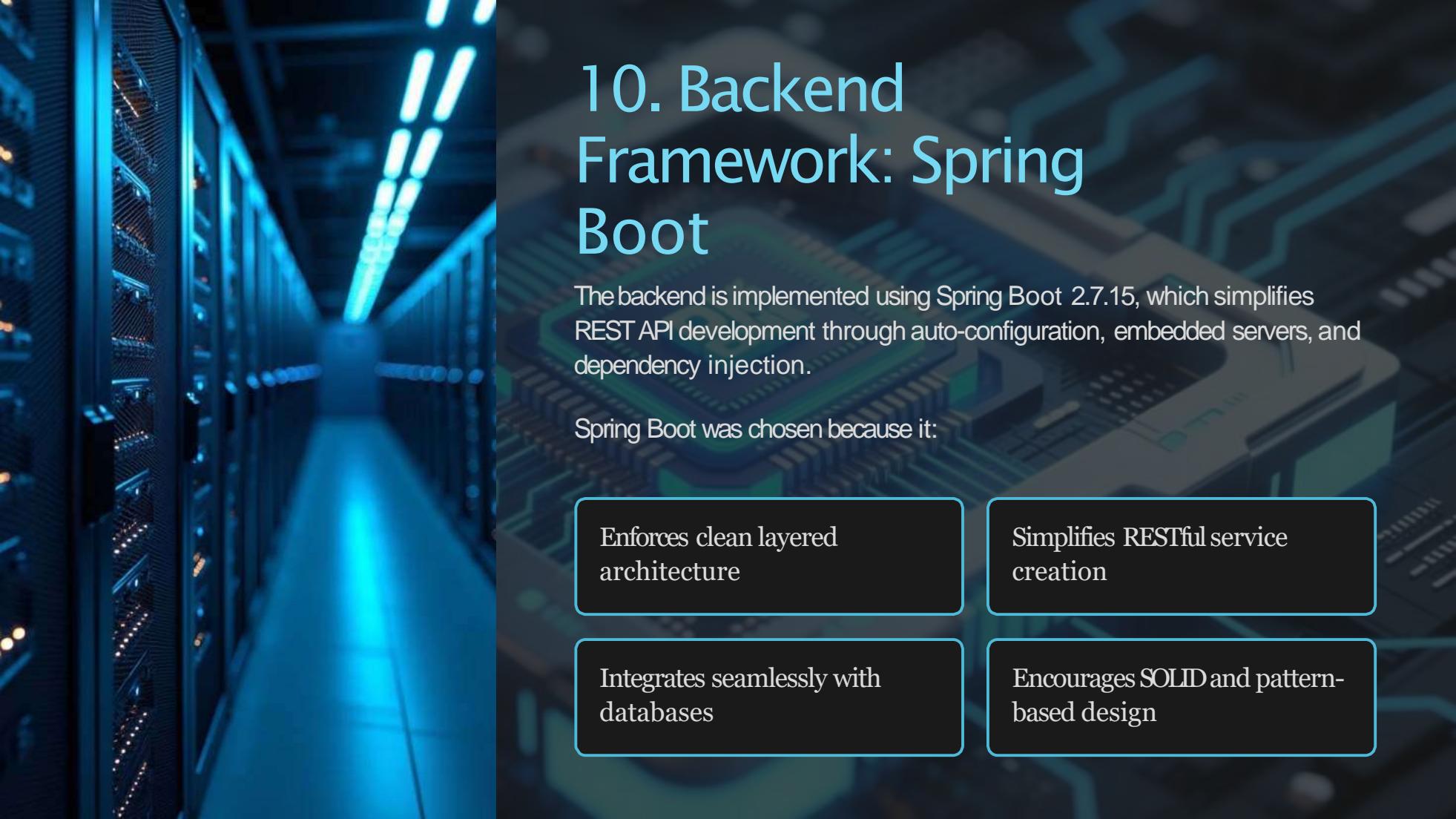
RESTful APIs

Server Port

8080

Embedded Server

Tomcat



# 10. Backend Framework: Spring Boot

The backend is implemented using Spring Boot 2.7.15, which simplifies REST API development through auto-configuration, embedded servers, and dependency injection.

Spring Boot was chosen because it:

Enforces clean layered architecture

Simplifies RESTful service creation

Integrates seamlessly with databases

Encourages SOLID and pattern-based design



# 11. Spring Boot Layered Architecture

The backend follows Spring Boot's layered architecture:

1 Controller Layer handles HTTP requests and API endpoints

2 Service Layer contains core business logic

3 Repository Layer manages data persistence using Spring Data JPA

This structure improves maintainability, testability, and scalability.

# 12. Singleton Pattern

Singleton behavior is achieved through Spring-managed Beans:

One instance per service class

Managed by Spring IoC container

Ensures consistency and efficient resource usage





# 13. Observer Pattern (Notifications)

The Observer Pattern is implemented in the notification system

Used in :

PostService.java

FollowService.java

When a user creates a post, all followers are automatically notified. Followers act as Observers, while the posting user acts as the Subject. This allows real-time notifications without tightly coupling users to their followers.

# 14. Repository & MVC Patterns

## Repository Pattern

Implemented using Spring Data JPA to abstract database access

## MVC Pattern

Model: User, Post, Follow, Notification

View: Web & Mobile Frontends

Controller: REST Controllers

This architecture improves readability, testing, and extensibility.

# 15.1 SOLID Principles Applied

## Single Responsibility Principle (SRP):

- PostService.java → Handles post operations only
- FollowService.java → Handles follow/unfollow logic
- NotificationService.java → Handles notifications
- PostRepository.java → Handles post database access
- PostController.java → Handles post HTTP endpoints

Each class has one responsibility, making the system maintainable and testable.

## Interface Segregation Principle (ISP):

- PostRepository → Post-related methods only
- FollowRepository → Follow-related methods only
- NotificationRepository → Notification-related methods only

Instead of one large repository, the system uses focused interfaces so services depend only on what they need.

# 15.2 SOLID Principles Applied

## Dependency Inversion Principle (DIP):

- Controller depends on Service. Service depends on Repository

### EXAMPLE:

- **PostController.java** → Depends on PostService, not implementation logic
- **PostService.java** → Depends on PostRepository interface

All dependencies are injected using Spring's IoC container, ensuring loose coupling between high-level modules and low-level implementations.  
This makes the system flexible, testable, and easy to maintain.



## 16. Core Features (Users & Posts)

1

User management (View users)

2

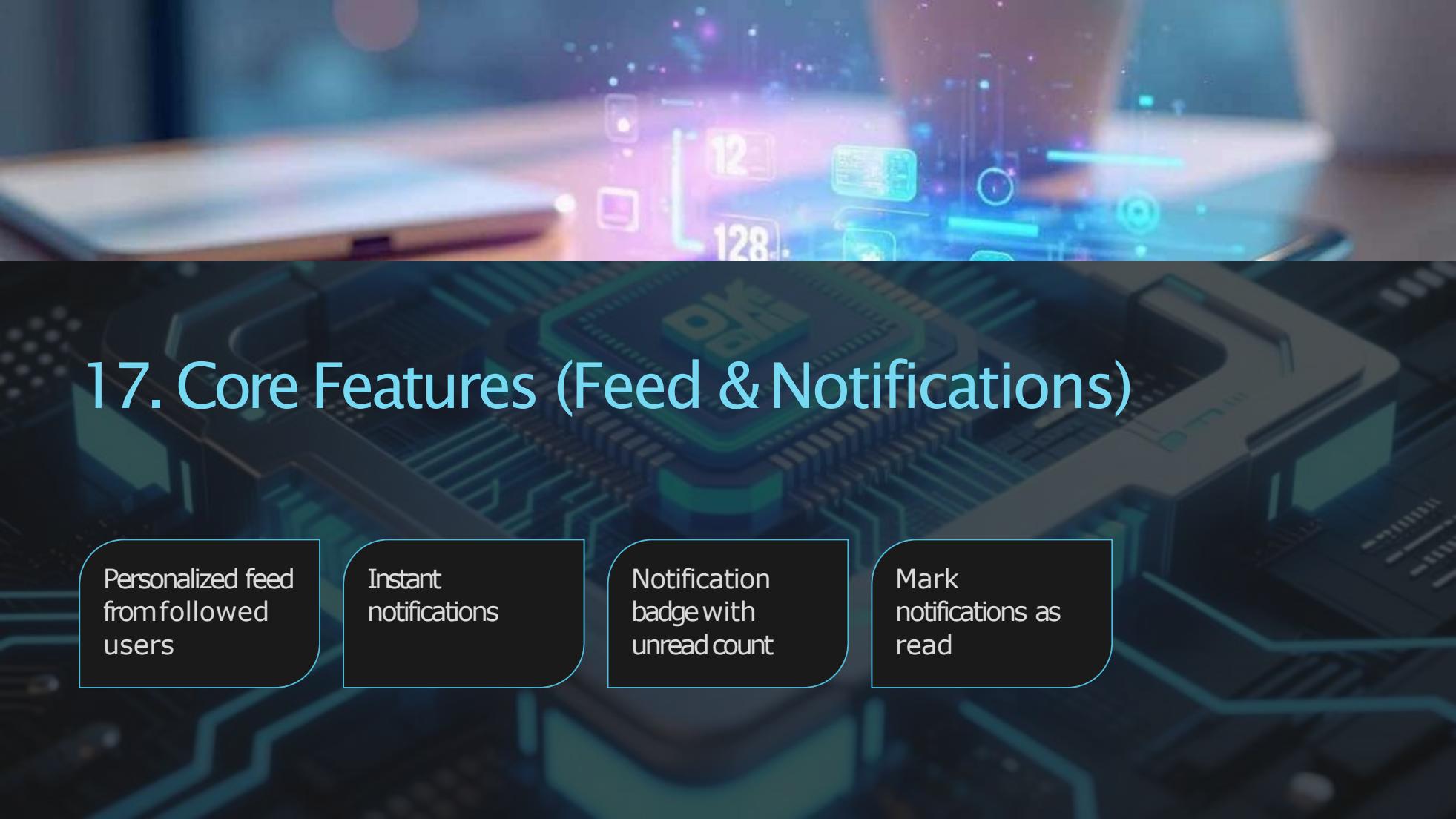
Post creation (maximum 250 characters)

3

Follow and unfollow functionality

4

Duplicate follow prevention



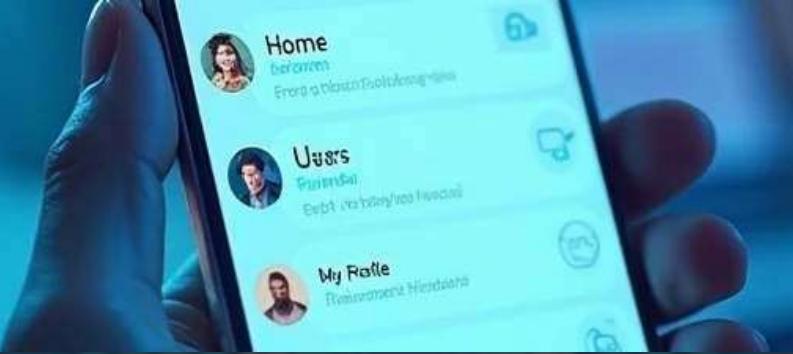
## 17. Core Features (Feed & Notifications)

Personalized feed  
from followed  
users

Instant  
notifications

Notification  
badge with  
unread count

Mark  
notifications as  
read



## 18. Core Features (User Interface)

1

Three-tab interface:

- Home
- Users
- My Profile

2

Human-readable timestamps (e.g., 30 minutes ago)

# 19. Backend API Endpoints

## Users

- GET  
`/api/users` : *Get all users*
- GET  
`/api/users/{id}`:  
*Get user by ID*

## Posts

- POST  
`/api/posts`: *creates new post*
- GET  
`/api/posts`: *gets all posts*
- GET  
`/api/posts/user/{userId}` : *Get posts by specific user*
- GET  
`/api/posts/feed/{userId}`  
*Get feed for a user (posts from users they follow + their own posts)*



# 20. Backend API Endpoints

## Follows

- POST  
`/api/follows/follow`: Follow a user
- POST  
`/api/follows/unfollow`: Unfollow a user
- GET  
`/api/follows/{userId}/following`: Get list of users this person is following
- GET  
`/api/follows/{userId}/follower`: Get followers of this user

## Notifications

- GET  
`/api/notifications/{userId}`: Get all notifications for a user
- GET  
`/api/notifications/{userId}/unread-count`: Get unread notifications count for a user
- GET  
`/api/notifications/{userId}/unread`: Get only unread notifications
- POST  
`/api/notifications/{id}/read`: Mark a notification as read
- DELETE  
`/api/notifications/{id}`: Deletes a notification



# 21. Solutions Implemented

Configured Spring CORS mappings

Switched to file-based H2 database

Timestamp conversion utilities

Implemented Observer Pattern

Polling-based UI refresh



## 22. Challenges Faced

1

CORS configuration issues

2

Database persistence handling

3

Preventing duplicate follows

4

Notification system implementation

5

User Feed logic

6

Timestamp formatting

7

UI state synchronization

## 23. Lessons Learned

1

Practical application of OOP and SOLID principles

2

Importance of clean layered architecture

3

Real-world REST API integration

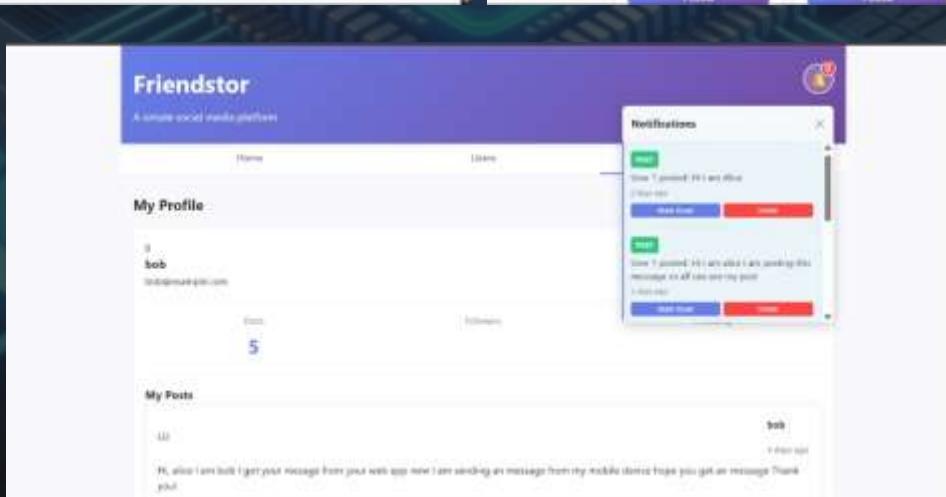
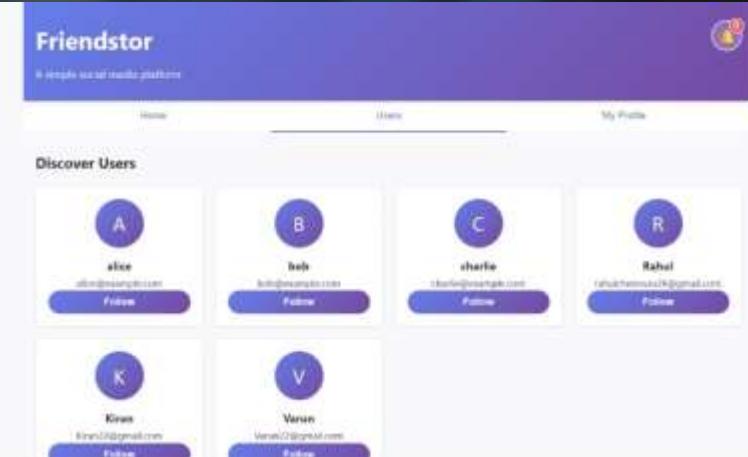
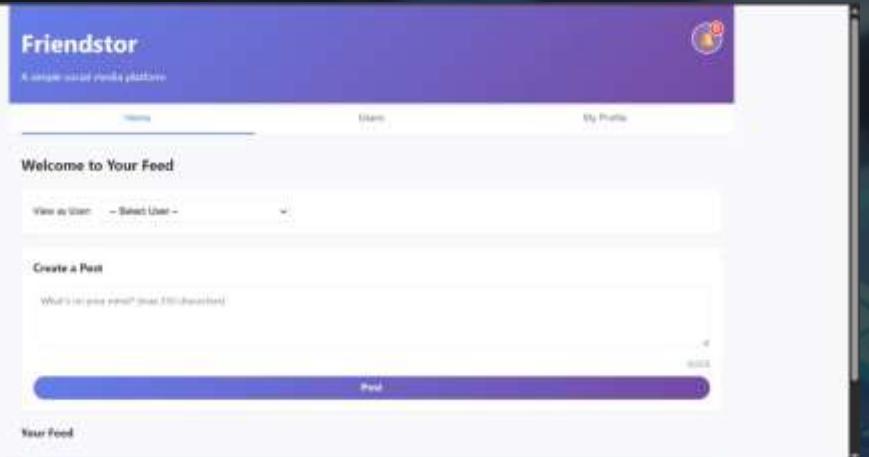
4

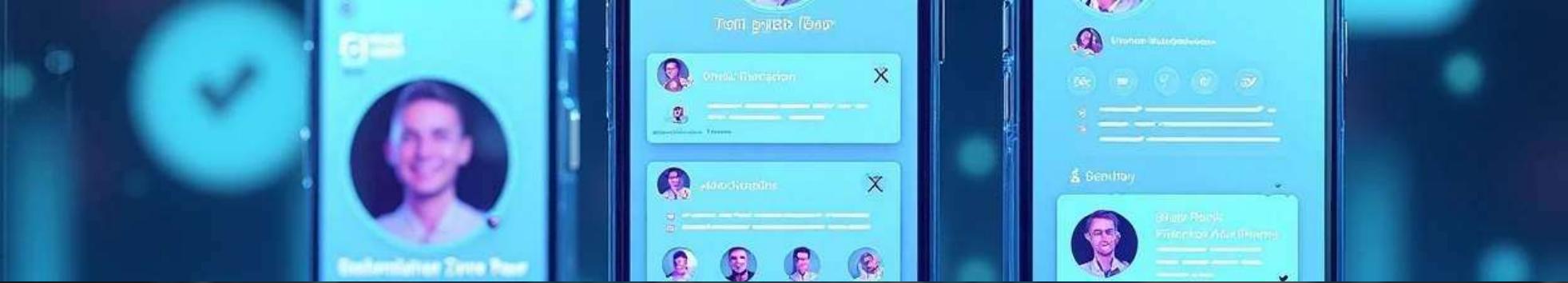
Benefits of design patterns in scalability

5

Team collaboration and version control experience

# 24. Application Screenshots





## 25. Conclusion

Friendstor successfully meets all assignment requirements. The project demonstrates strong use of Spring Boot, object-oriented principles, SOLID design, and architectural patterns. It provided valuable hands-on experience in full-stack development and team collaboration.



Thank You