Project Report

SYNAPSE: Social Media Web Application

Organization: Lovely Professional University

Submitted To: Dr. Siddardha Dasari

Prepared By:

1) Rajveer Singh (12317715)
2) Nashit Shahi (12313603)
3) Samad (12319149)
4) Harisvardhan Singh Naghyal (12319805)

Date: 20-Feb-2026

1. Executive Summary

In today's rapidly evolving digital ecosystem, the demand for social communication platforms that fundamentally prioritize user privacy, data security, and systemic resilience has never been more critical. As global users become increasingly aware of data vulnerabilities and digital footprints, the industry requires a paradigm shift away from traditional, data-mining-centric social networks. This project addresses this critical market gap by delivering a comprehensive, production-ready foundational blueprint for a "private-first" digital communication platform. The primary objective is to engineer a trusted digital environment that redefines secure social interactions, ensuring that user data sovereignty remains paramount without compromising on modern user experience or platform responsiveness.

At the core of this innovative solution is a highly robust, scalable backend infrastructure engineered utilizing the Python programming language and the Django web framework. Moving beyond standard, out-of-the-box implementations, the architecture employs a thoughtfully modified Model-View-Template (MVT) design pattern. This tailored architectural approach allows for a stringent separation of concerns, ensuring that the complex business logic and rigorous privacy controls are cleanly decoupled from the data presentation layer. This design not only fortifies the application's core but also streamlines future feature development and significantly reduces long-term technical debt.

Recognizing that platform integrity begins at the point of user entry, the system integrates the industry-standard OAuth 2.0 authorization framework. This strategic implementation serves a dual purpose: it fortifies the application against unauthorized access through rigorous token-based security, while simultaneously providing a frictionless, federated identity management experience for the end-user. By leveraging OAuth 2.0, the architecture minimizes reliance on traditional, vulnerable password-based authentication models, thereby mitigating common security vectors and fostering immediate user trust upon onboarding.

Furthermore, to satisfy the modern user's baseline expectation for seamless, instantaneous communication, the application completely bypasses traditional, latency-prone HTTP request-response cycles for its messaging capabilities. Instead, the architecture features a state-of-the-art, bi-directional real-time communication engine powered by WebSockets. This asynchronous functionality is facilitated through the Asynchronous Server Gateway Interface (ASGI) and powerfully orchestrated by Django Channels. This sophisticated, event-driven architecture maintains persistent, low-latency connections, enabling true real-time chat, instant systemic notifications, and live user updates without overwhelming underlying server resources.

Ultimately, this architectural blueprint is engineered with foresight. By strategically decoupling the asynchronous real-time components from the synchronous core application, the system is inherently primed for dynamic horizontal scalability. Whether deployed in a highly containerized cloud ecosystem or across a distributed edge network, the platform is fully equipped to maintain high-performance, real-time interactions, guaranteeing a flawless, secure, and infinitely scalable communication experience for a globally distributed user base.

2. Project Overview

Business Problem Statement

Existing social networking platforms often struggle to guarantee the complete privacy and security of one-to-one communications, resulting in low user confidence in data integrity and control. The technical challenge was to design and implement a digital communication platform where user identity is cryptographically secured, and all private messaging occurs over a dedicated, low-latency, real-time channel, ensuring resilience against common communication-level exploits and data interception.

Project Objectives:

Objective ID	Description	Key Deliverable
O-01	**Secure Identity Management**	Implement the core authentication system using Django's framework, hardened with OAuth 2.0 social sign-in capabilities, ensuring all credentials are cryptographically hashed and salted.
O-02	**Real-Time Messaging**	Develop a persistent, bi-directional communication layer utilizing WebSockets via Django Channels and an in-memory broker (Redis) to facilitate instantaneous, low-latency private messaging streams.
O-03	**Scalable Social Graph**	Design a relational database schema (models.py) capable of efficiently modeling and querying complex, asymmetrical follower relationships and hierarchical content (posts/comments) for high-performance retrieval.

Scope:

Category	Description	Status
Authentication	User registration, login, profile management, and OAuth 2.0 social sign-in.	**In Scope**
Messaging System	Persistent, real-time private messaging feature leveraging the ASGI architecture.	**In Scope**
Content/Posts	Core application functionality for content posting and feed generation.	**In Scope**
AI-driven Moderation	Implementation of recommendation engines or machine learning-driven content ranking/moderation algorithms.	**Out-of-Scope (Phase 1)**

Key Deliverables & Success Criteria

The project success was measured across three primary vectors, completed within the 9-day intensive sprint:

- **Functional Codebase:** A production-ready, fully unit-tested Python/Django application implementing all in-scope features, including a secure authentication system, a performant social graph, and a low-latency real-time messaging module.
- **Technical Documentation:** A comprehensive Solution Architecture report detailing the system's logic, data flow, and security posture (this document).
- **Performance Benchmarks:** Demonstration of millisecond-level message delivery latency and mitigation of the N+1 query problem across core feed and comment retrieval endpoints.

3. Solution Architecture & Design

3.1 System Overview

The foundational architecture adheres to a **Decoupled Multi-tier Pattern**, evolving from the traditional monolithic **Model-View-Template (MVT)** configuration to accommodate real-time, event-driven requirements. The system runs dual server processes:

- **Synchronous WSGI:** Handles standard, request-response HTTP cycles (e.g., registration, profile updates).
- **Asynchronous ASGI:** Handles the persistent, bi-directional communication channels required for live chat functionalities. The **ASGI Server** utilizes Django Channels to manage **WebSocket** connections, maintaining persistent Transmission Control Protocol (TCP) connections to mitigate the latency of continuous polling. This system uses a Redis-backed message broker layer as the channel layer, enabling the Publisher/Subscriber paradigm for instantaneous, real-time message delivery across distributed application instances.

3.2 Technology Stack

Component Category	Technology
Backend Framework	Python 3.x, Django, Django REST Framework
Primary Database	PostgreSQL (Relational)
In-Memory Channel Layer	Redis (High-throughput for Real-Time)
Real-Time Protocol	WebSockets via Django Channels
Authentication Module	social-auth-app-django (OAuth 2.0)
Asynchronous Processing	Celery (Background Notifications/Tasks)

3.3 Integration Points

- **Cloud Media Storage:** User-generated multimedia (avatars, images) are not stored in the relational database. The integration layer delegates persistence to highly available cloud object storage protocols (e.g., S3-compatible services). Only the public access URL is committed to the database. A Content Delivery Network (CDN) is used to cache these assets at edge locations, optimizing global content delivery speeds.
- **Background Task Processing:** The system uses a distributed task queue (Celery) to offload intensive, long-running computational processes—such as email dispatch and notification fan-out—from the main user-facing request-response cycle. This ensures the web servers remain responsive, delegating slow network operations to dedicated background worker nodes.

3.4 Security

The security posture is built upon a defense-in-depth strategy:

- **PBKDF2 Password Hashing:** For direct platform authentication, the application leverages the highly audited **Password-Based Key Derivation Function 2 (PBKDF2)** utilizing a Secure Hash Algorithm 256-bit (SHA256) pseudorandom function. This process includes salting and a high iteration count to computationally frustrate brute-force attacks.
- **OAuth 2.0 Integration:** The system implements the industry-standard **OAuth 2.0 Authorization Code Grant Flow** for external social sign-in. The backend rigorously verifies a cryptographic state parameter during the multi-step handshake to prevent **Cross-Site Request Forgery (CSRF)** attacks.
- **Framework Middleware Defenses:** Django's middleware stack is leveraged for automated defenses. **CSRF Middleware** enforces a randomized, secure token for all state-changing requests. **Cross-Site Scripting (XSS)** mitigation is achieved through the automatic escaping of user-provided data during the rendering process, preventing the execution of injected malicious JavaScript payloads.

3.5 Scalability

The architecture relies entirely on dynamic **Horizontal Scaling** of its containerized synchronous and asynchronous application instances.

- **Multi-Tiered Caching:** To mitigate the relational database as a primary performance bottleneck, an aggressive, **multi-tiered caching mechanism** is used. **Redis** serves a critical dual purpose as the high-throughput channel layer and an in-memory caching tier. The system employs a Cache-Aside strategy to retrieve data with sub-millisecond latency on a cache hit, drastically reducing database Input/Output (I/O) load.
- **Database Optimization:** Views strictly utilize advanced pre-fetching methods to fetch all necessary relational data in a singular, massively optimized transaction, effectively

solving the classic **N+1 Query Mitigation** problem that plague naive social graph implementations.

4. Implementation Plan (9-Day Timeline)

The project was executed within a highly compressed, 9-day intensive sprint, broken down into four distinct phases:

Phase/Days	Focus Area	Key Activities	Risk Assessment & Mitigation
Phase 1: Days 1-2	Requirements, Architecture Design, and Authentication Setup	Finalize architectural blueprint (ASGI, MVT). Implement core Django User Model and custom Profile model. Integrate and test OAuth 2.0 social sign-in flow. Setup PostgreSQL and Redis.	**Risk: Database Migration** - *Mitigation:* Ensure strict, version-controlled migrations (`makemigrations`/`migrate`) are run after each models.py change.
Phase 2: Days 3-5	Core Development: Posts, Media, Social Graph, and Feed Logic	Implement Content Model. Develop Social Graph (Follower) junction table with unique composite constraints. Implement **Modified Preorder Tree Traversal** for hierarchical comment fetching to ensure **N+1 query mitigation**. Integrate cloud media storage.	**Risk: N+1 Query Problem** - *Mitigation:* Mandatory code review of all views.py logic to enforce `select_related()` and `prefetch_related()` for all feed endpoints.

Phase 3: Days 6-7	Real-time Messaging and Notification Integration	Implement distinct Thread and Message models. Configure Django Channels and Redis channel layer. Implement asynchronous consumer classes. Test **Publisher/Subscriber Paradigm** for real-time message delivery. Integrate background task queue (Celery) for notifications.	**Risk: Concurrency Handling in Chat** - *Mitigation:* Utilize atomic transactions when saving new messages to prevent race conditions and ensure data integrity.
Phase 4: Days 8-9	UI Refinement, Testing, and Quality Assurance	Finalize all API endpoints with Django REST Framework serializers. Conduct extensive unit and integration testing across all modules. Performance testing of real-time latency. Finalize and review all technical documentation.	**Risk: Security Vulnerabilities** - *Mitigation:* Final security audit (CSRF/XSS) and comprehensive checks of PBKDF2/OAuth 2.0 implementations.

4.2 Resource Allocation

The 9-day intensive sprint required a carefully selected set of software and infrastructure resources to maximize development velocity and ensure the quality of the final product.

- **Software and Development Tools:**
  - **Backend Framework:** Python 3.x, Django 4.x
  - **Database & Broker:** PostgreSQL, Redis
  - **Source Control:** GitHub
  - **Development Environment:** VS Code
- **Infrastructure:**
  - **Development:** Local development servers for rapid iteration and testing.
  - **Testing/Staging:** AWS/Heroku testing environments for validation of cloud-native behaviors, such as media storage and background task processing.

5. Development Team Introduction

The 9-day project was executed by a specialized team of four, with roles designed for maximum focus and efficiency across the key architectural domains:

Role	Name	Experience	Responsibilities
Full Stack Developer	Nashit	Student	Authentication System (Auth) logic.
Backend Developer	Harisvardhan	Student	Content Management (Posts) and Media Storage.
Frontend Developer	Samad	Student	Social Graph
Full Stack Developer	Rajveer Singh	Student	Real-Time Messaging System, Search, Feed and Admin Interface.

6. Financial & Business Impact Analysis

6.1 Development Cost Estimation

This project was a 9-day intensive sprint executed by a team of four developers working a standard 8-hour day. This effort is calculated to provide a conceptual foundation for project budgeting and demonstrates an understanding of the resource-to-time constraint inherent in academic projects.

- **Total Project Effort Calculation:**
  $$\text{4 Developers} \times \text{9 Days} \times \text{8 Hours/Day} = \mathbf{288 \text{ Total Man-Hours}}$$
- **Conceptual Budget Estimate (Based on Indian Junior Developer Rate):**
  $$\text{288 Man-Hours} \times \text{₹500/Hour (Conceptual Rate)} = \mathbf{₹1,44,000 \text{ Total Development Cost (Conceptual)}}$$

Note: This is a theoretical exercise to demonstrate project budgeting based on a conceptual Indian junior developer rate and does not represent actual, incurred project costs.

6.2 Estimated Infrastructure Costs (Conceptual for 1,000 Users)

The infrastructure for the initial 1,000 users would be cost-optimized around high-availability cloud services, utilizing the containerized approach for efficiency.

- **Compute (AWS EC2/Containers):** Approximately 2-3 small application instances running the dual synchronous/asynchronous server processes. Cost would scale linearly with horizontal growth.
- **Database (AWS RDS/PostgreSQL):** A managed relational database instance configured with a small primary write node and at least one read-replica for offloading read-heavy operations, mitigating load.
- **In-Memory Broker (AWS ElastiCache/Redis):** A single, high-memory Redis instance to serve the critical channel layer backbone and the multi-tiered caching system.
- **Storage/CDN (AWS S3/CloudFront):** Minimal ongoing storage costs for media assets, with distribution costs managed by the CDN for global user access.

The initial estimated monthly cost for an optimized, resilient infrastructure for 1,000 active users would be in the low-to-mid hundreds of USD, with the most significant scaling factor being the compute layer (application instances) and the Redis instance as user concurrency increases.

6.3 Return on Investment (ROI) of a Private, Secure Ecosystem

The ROI of the platform is not measured purely in immediate financial returns but in strategic market positioning and user trust. By building a private, secure ecosystem:

- **Premium Positioning:** The platform is positioned as a secure alternative, attracting a niche user base highly concerned with data privacy, leading to high user lifetime value (LTV).
- **Mitigated Liability:** Robust security (PBKDF2, OAuth 2.0, CSRF/XSS middleware) significantly reduces the risk of data breaches, which carry catastrophic financial and legal costs.
- **Feature Differentiation:** The integrated, low-latency, real-time communication via WebSockets is a critical feature differentiator against less architecturally advanced competitors, driving organic adoption.

7. User Manual

7.1 Introduction

This User Manual provides step-by-step instructions for using the **Social Media Web Application**. The system allows users to create accounts, manage profiles, search other users, follow/unfollow profiles, and communicate through a real-time messaging system.

The manual is intended for first-time users with basic knowledge of using web applications.

7.2 System Requirements

Hardware Requirements

- Processor: Intel i3 or above
- RAM: Minimum 4 GB
- Storage: 500 MB free space

Software Requirements

- Operating System: Windows 10 / Linux / macOS
- Python 3.x installed
- Web Browser (Google Chrome recommended)
- Internet connection
- Django Framework
- SQLite Database (default)

7.3 Installation Guide

Follow these steps to run the application:

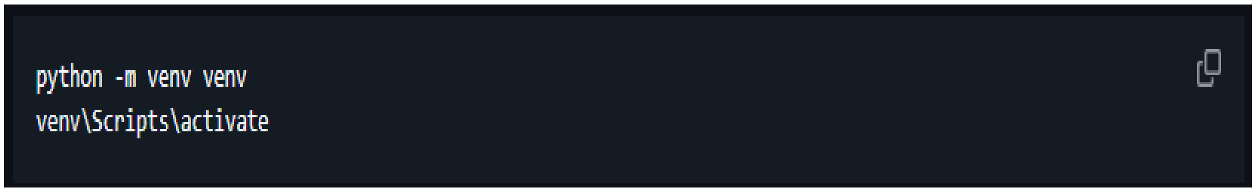
Step 1: Download Project

Download or extract the project folder to your system.

Step 2: Open Terminal

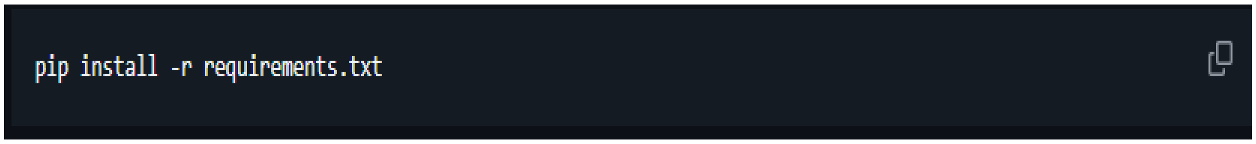
Open Command Prompt or Terminal inside the project directory.

Step 3: Create Virtual Environment



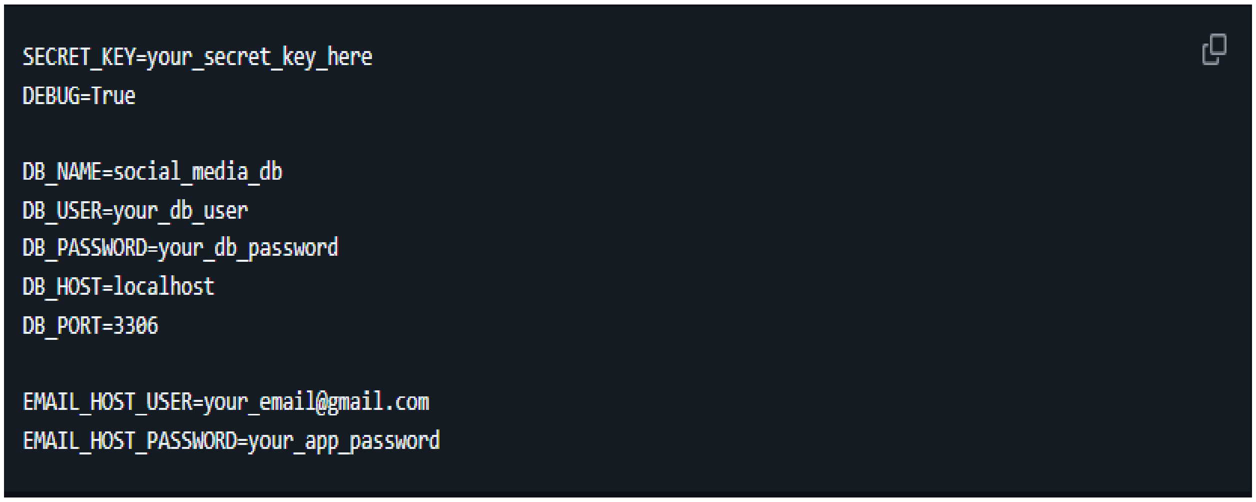
```
python -m venv venv
venv\Scripts\activate
```

Step 4: Installing Required Libraries



```
pip install -r requirements.txt
```

Step 5: Create .env File



```
SECRET_KEY=your_secret_key_here
DEBUG=True

DB_NAME=social_media_db
DB_USER=your_db_user
DB_PASSWORD=your_db_password
DB_HOST=localhost
DB_PORT=3306

EMAIL_HOST_USER=your_email@gmail.com
EMAIL_HOST_PASSWORD=your_app_password
```

Step 5: Configure MYSQL Database

```
CREATE DATABASE social_media_db;
```

Step 6: Apply Migrations

```
python manage.py makemigrations
python manage.py migrate
```

Step 7: Create Superuser (Optional)

```
python manage.py createsuperuser
```

Step 8: Run Deployment Server

```
python manage.py runserver
```

Step 9: Open in Browser

```
http://127.0.0.1:8000/
```

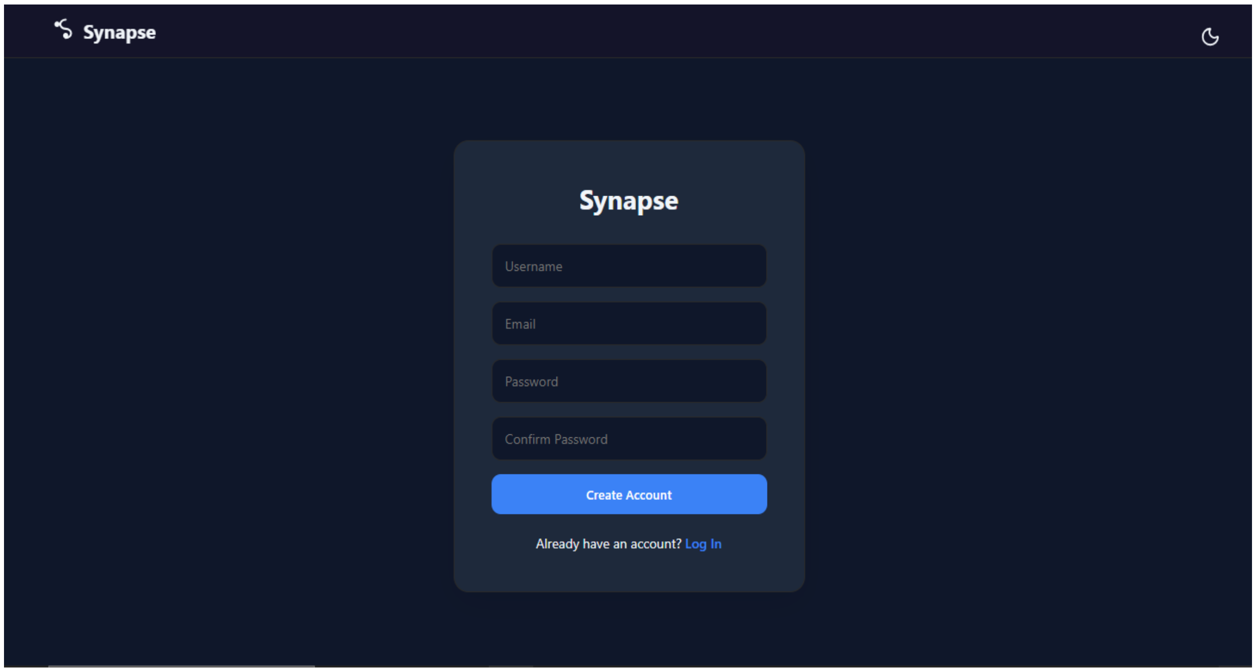
7.4 User Registration

New users must create an account before accessing features.

Steps:

1. Open the homepage.
2. Click **Sign Up / Register**.
3. Enter:
   - Username
   - Email address
   - Password
4. Click **Register**.

Account is created successfully and user can log in.

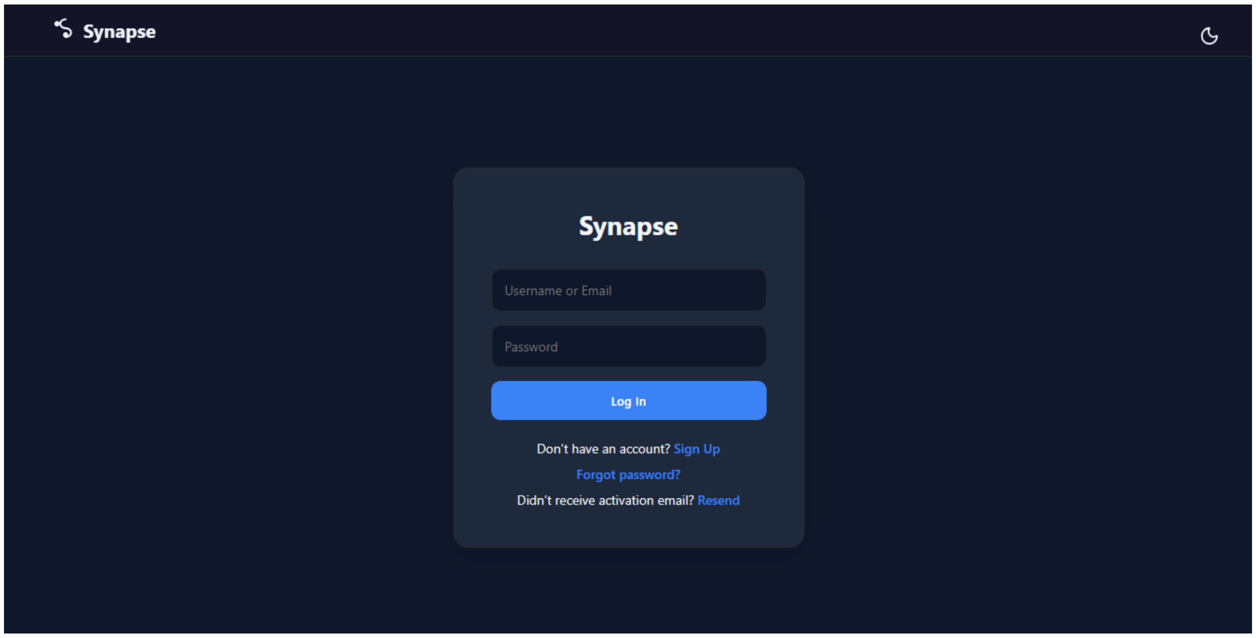


7.5 User Login

Steps:

1. Click **Login**.
2. Enter username and password.
3. Click **Login Button**.

After successful login, the user is redirected to the home/dashboard page.

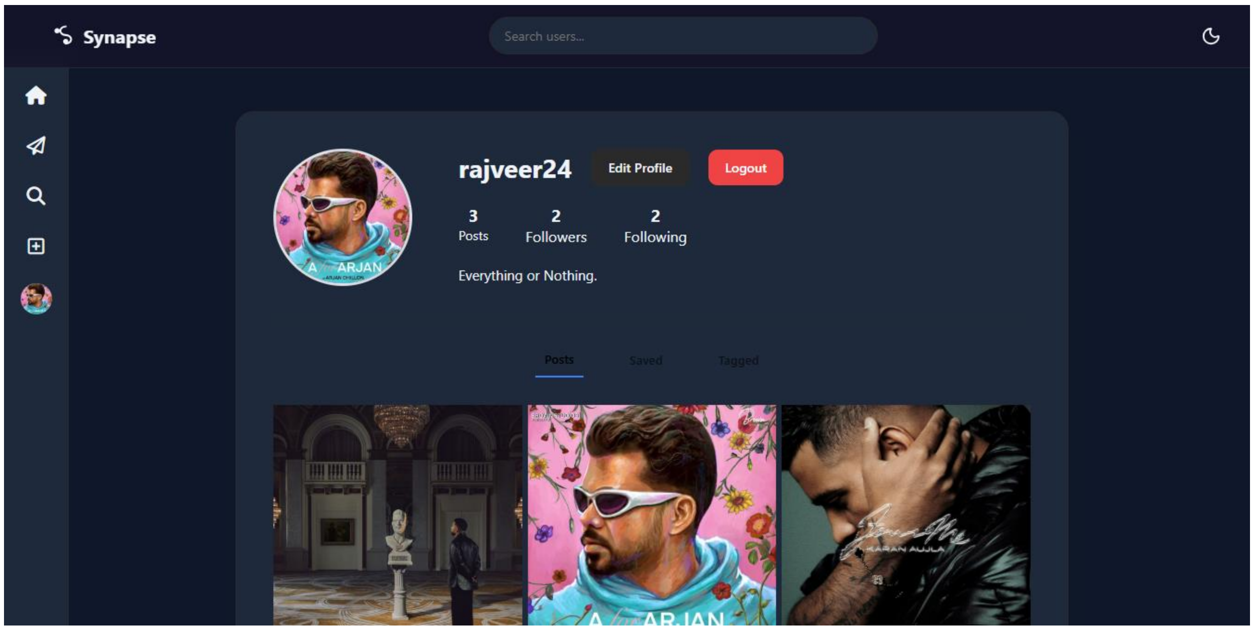

Synapse
Synapse
Username or Email
Password
Log In
Don't have an account? Sign Up
Forgot password?
Didn't receive activation email? Resend

7.6 User Profile

After login, users can:

- View posts or user activity
- Access search feature
- Open messages
- Visit profiles
- Logout

The sidebar provides quick access to all features.



7.7 Profile Management

Users can manage their personal profile.

Features:

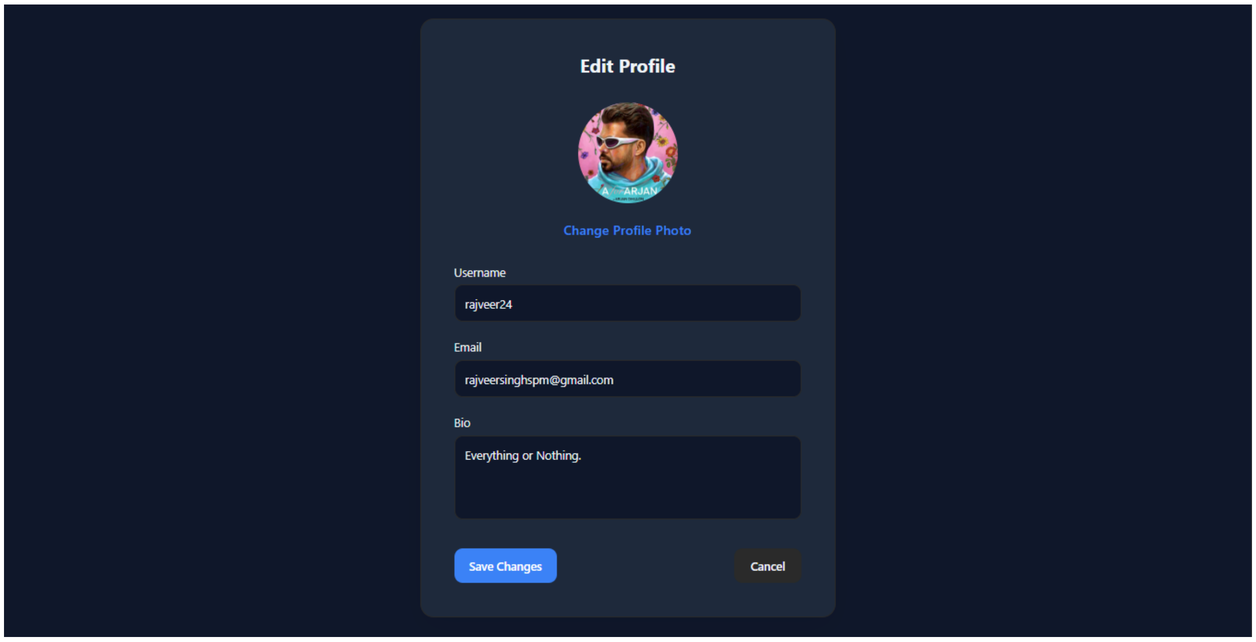
- Upload profile picture
- Edit bio information
- Update personal details

Steps:

1. Click profile icon.
2. Select **Edit Profile**.
3. Update information.
4. Click **Save**.

Changes are updated instantly

.


Edit Profile

Change Profile Photo

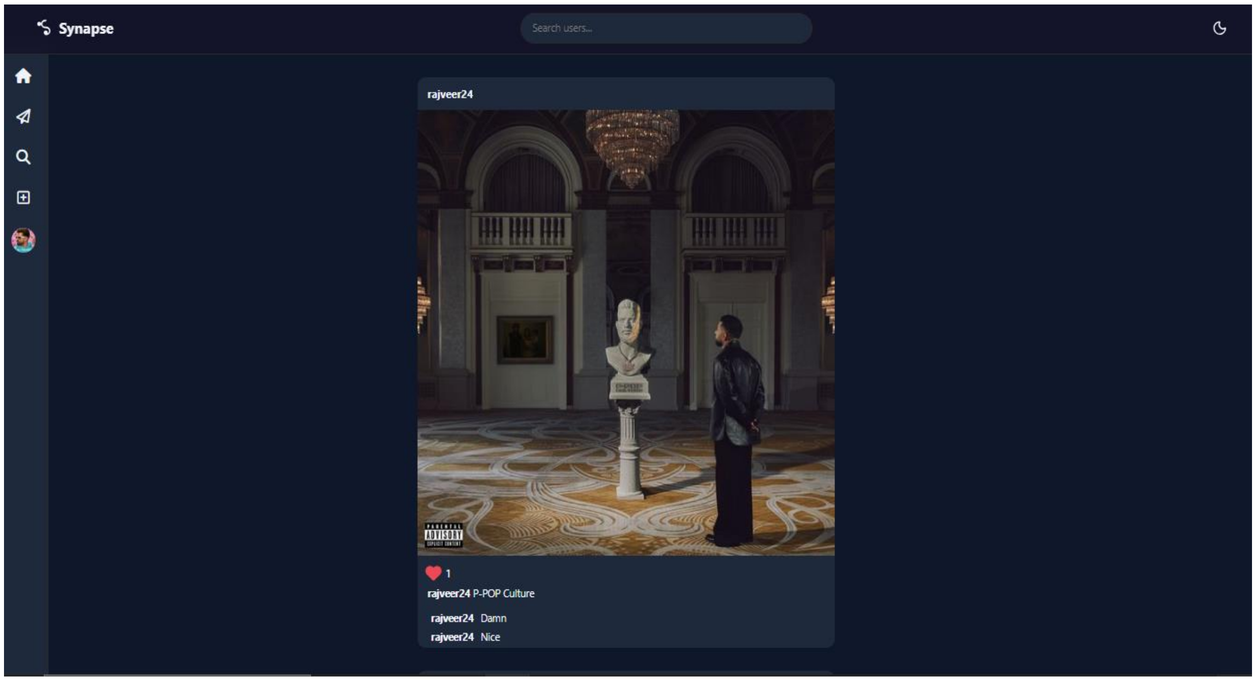
Username
rajveer24

Email
rajveersinghspm@gmail.com

Bio
Everything or Nothing.

7.8 Home Feed

- Home feed contains the recent posts of all the users.
- You can like the posts.
- You can also visit the user profile.
- You can click the post to see the post details
- You can also see like count and comments



rajveer24 P-POP Culture
rajveer24 Damn
rajveer24 Nice

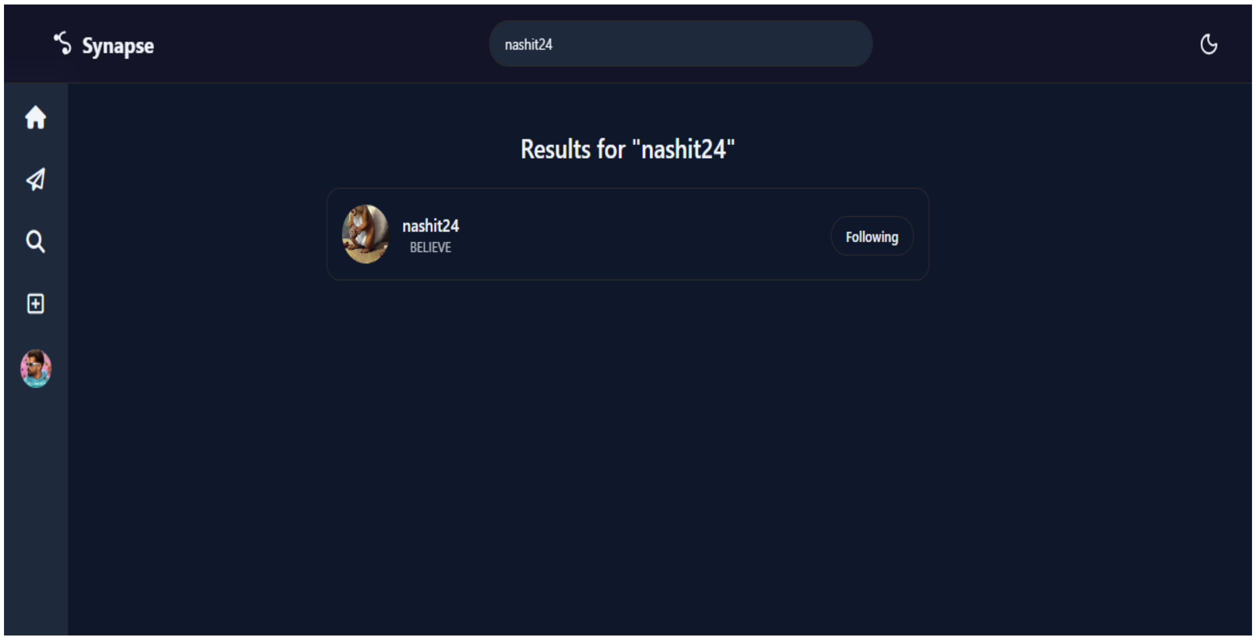
7.9 Searching User

The system allows users to search for other registered users.

Steps:

1. Locate the search bar.
2. Enter username.
3. Press search.

Search results display matching profiles.



7.10 Messaging System

The application provides private messaging between users.

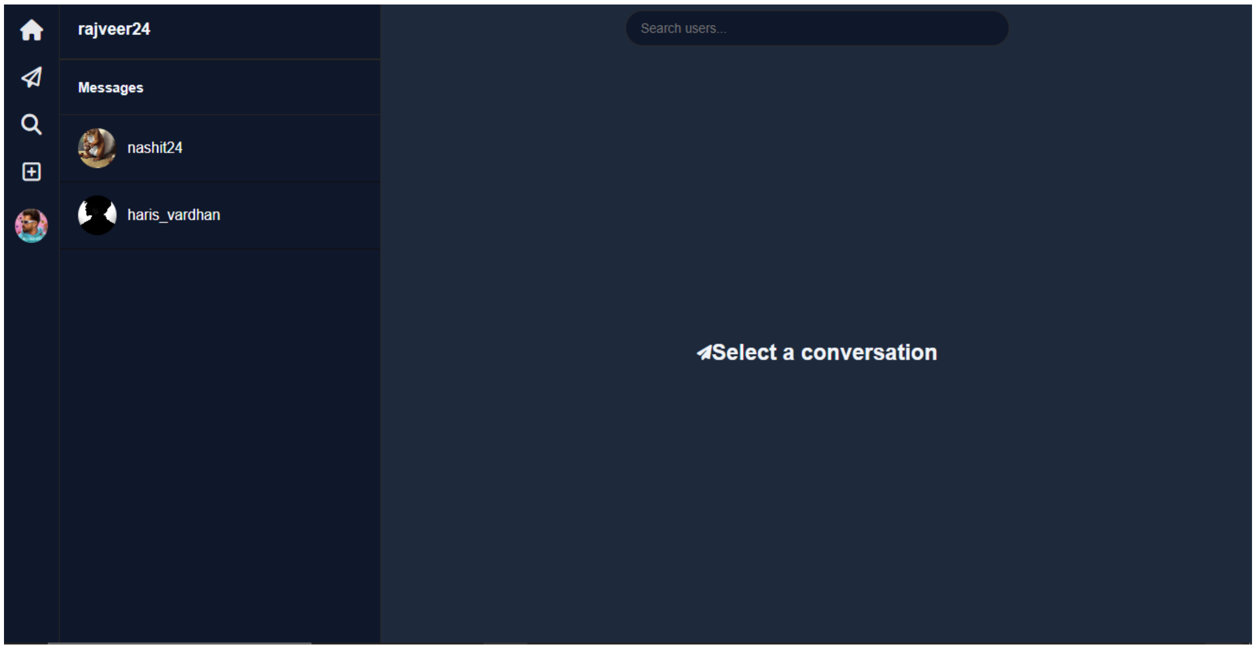
Sending Messages:

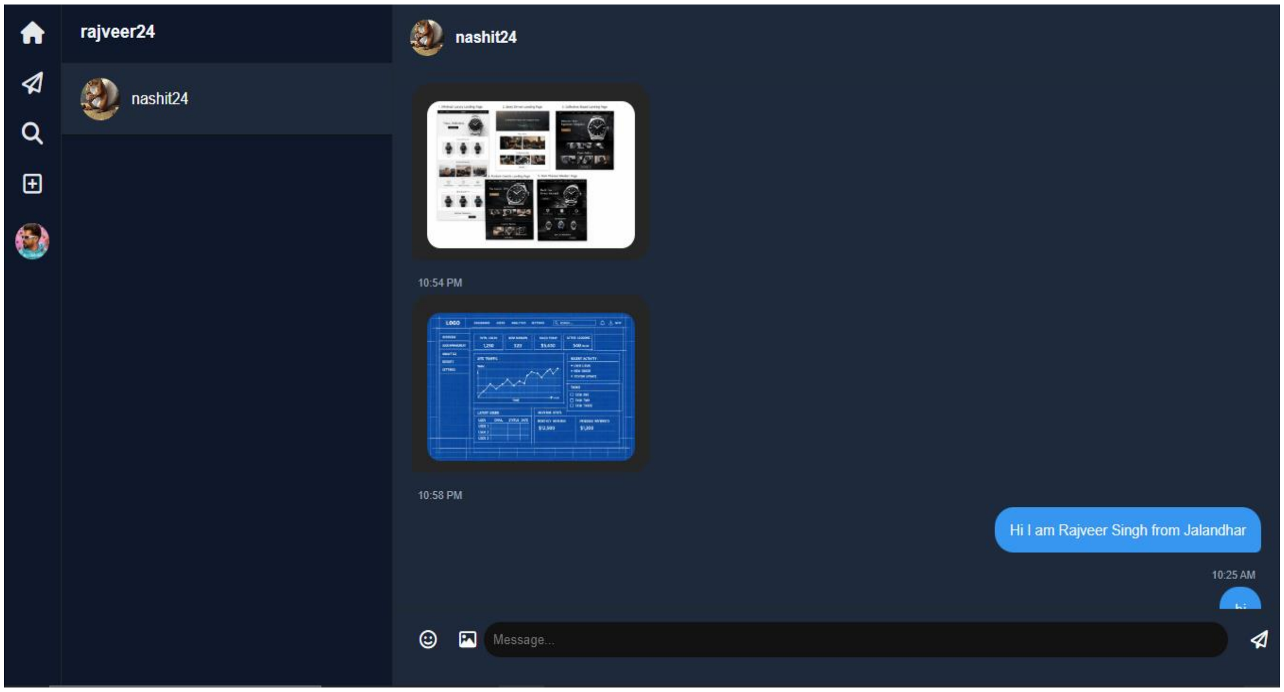
1. Open **Messages** section.
2. Select a user.
3. Type message in chat box.
4. Press Send.

Messages appear instantly in conversation window.

Features:

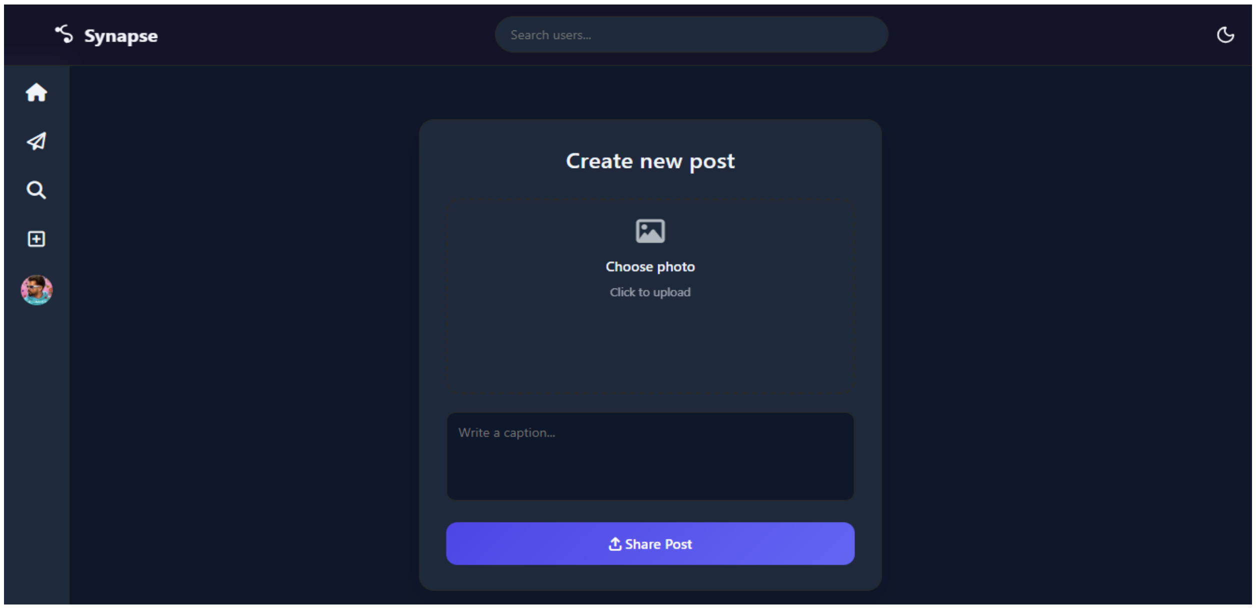
- Real-time chat interface
- User profile picture display
- Conversation history


rajveer24
Search users...
Messages
nashit24
haris_vardhan
Select a conversation


rajveer24
nashit24
nashit24
10:54 PM
10:58 PM
Hi I am Rajveer Singh from Jalandhar
10:25 AM
Message...

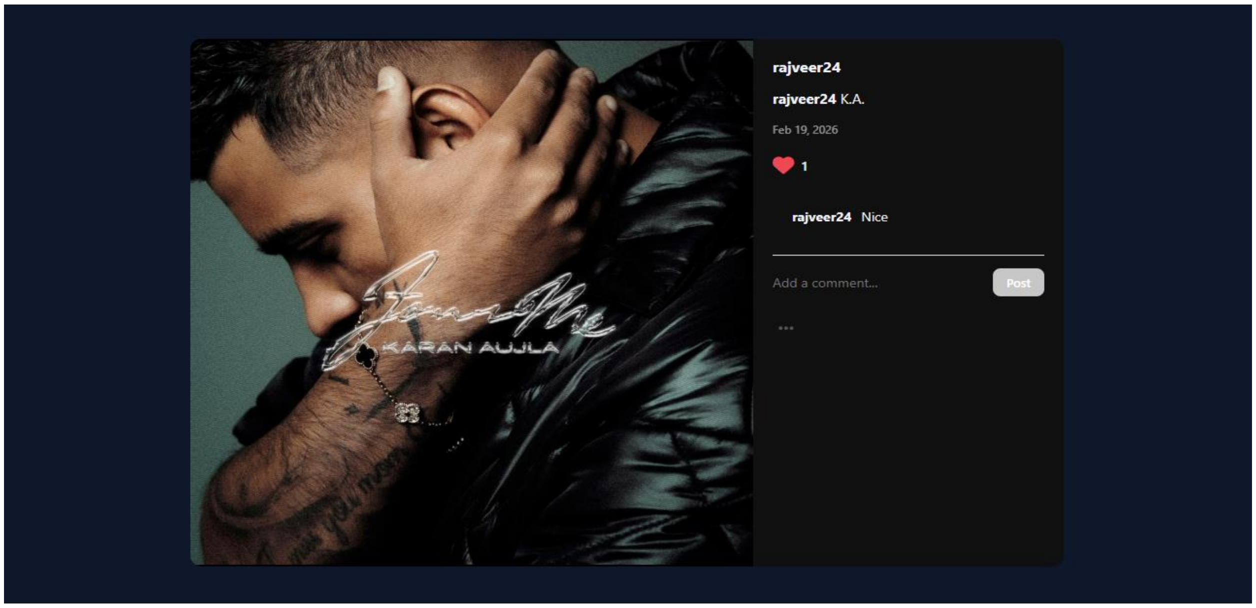
7.11 Create Post

- Users can create a post and add it to their profile



7.12 Post details

- Users can view a post by clicking on it.
- It shows the like count and the comments on the post too



7.13 Logout Process

To safely exit:

1. Click **Logout** button in navigation bar.
2. User session ends.
3. Redirected to login page.

7.14 Error Handling

The system handles common errors:

Situation	System Response
Wrong password	Error message displayed
User not found	No results shown
Empty search	Prompt message shown
Unauthorized access	Redirect to login page

7.15 Security Features

- Password authentication system
- Secure user sessions
- Login required for private features
- Protected messaging access

7.16 Tips for Users

- Use strong passwords.
- Logout after using shared computers.
- Keep profile updated for better interaction.
- Refresh page if chat delays occur.

7.17 Troubleshooting

Problem	Solution
Server not running	Run `python manage.py runserver`
Page not loading	Check internet or refresh
Login issue	Verify username/password
Database error	Run migrations again

7.18 Conclusion

The Social Media Web Application provides a user-friendly platform for communication and social interaction. The system enables profile management, user discovery, and real-time messaging in an organized and secure environment.

8. Conclusion & Next Steps

The architectural design of the Django Social Media Application represents a highly sophisticated integration of standard, deeply reliable monolithic stability with modern, asynchronous, event-driven requirements. The database schema enforces **Referential Integrity** while remaining heavily optimized for massive read loads. The identity management system provides an exceptionally secure dual-layered approach, and the architectural evolution to an asynchronous WebSockets infrastructure, backed by a robust in-memory broker, bridges the latency gap for instantaneous real-time communication streams.

Roadmap for Future Development:

- **Phase 2: Mobile Application Integration (Stateless API):** Develop native iOS and Android clients, transitioning the JSON Web Token (JWT) stateless authentication to be the primary interface.
- **Phase 3: Cloud-Native Deployment:** Transition the current containerized deployment to a fully managed Kubernetes cluster for dynamic, zero-touch provisioning and de-provisioning of application instances (Horizontal Pod Autoscaler).
- **Phase 4: Advanced Search and Analytics:** Implement a dedicated search engine (e.g., Elasticsearch) to enable high-speed, full-text searching and deploy real-time analytics monitoring.