



DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.

Experiment No: 3

Student Name: Aditya Singh
Branch: BE CSE
Semester: 6th
Subject Name: System Design

UID: 23BCS11827
Section/Group: 23BCS_KRG-2_B
Date of Performance: 28/01/2026
Subject Code: 23CSH-314

1- Aim - To design a social media platform similar to Facebook or Instagram

2- Requirements: Functional & Non-Functional

A- Functional Requirement

- Client should be able to register and login to the application.
- Client should be able to create post (text / image / videos).
- Client should be able to follow each other (or send friend requests).
- Client should be able to like or comment on the post.
- Client should be able to view the feed of post from users they follow.

B- Non-Functional Requirement

- **Scalability:** The system should support 500 million daily active users.
- **Consistency & Availability:** For a social media application, high availability is prioritized over strict consistency, since temporary delays in data propagation are acceptable, but downtime is not.
- **Latency** (Uploading speed for publishing a post): The target latency is 500 ms for uploading a post.

3- Core-entities of System

- Users
- Posts
- Followers
- Like and comments
- Feed

4- API endpoint creation

A. User Onboarding APIs

1. **User Registration** POST /api/users/register
2. **User Login** POST /api/users/login
3. **Display User Data** GET /api/users/{user_id}/profile
4. **Update User Data** PUT /api/users/{user_id}/profile

B. User Posts APIs

1. **Create a Post** POST /api/users/{user_id}/posts
2. **Get a Post by ID** GET /api/posts/{post_id}
3. **Update a Post** PUT /api/posts/{post_id}
4. **Delete a Post** DELETE /api/posts/{post_id}



DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.

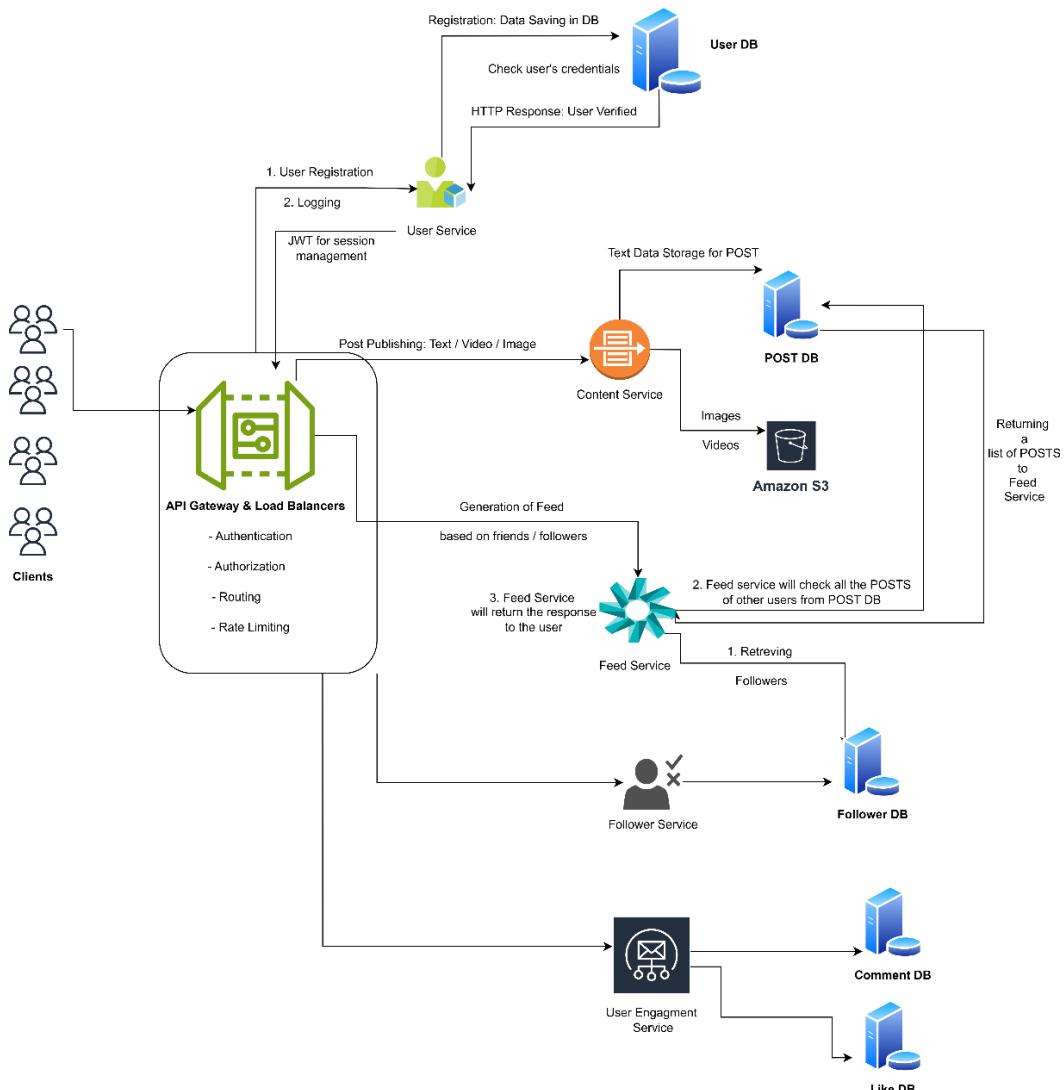
5. **Get Feed (Pagination)** GET /api/posts/feed?limit={limit}&offset={offset}
 6. **Get Posts of a Specific User (Pagination)** GET /api/users/{user_id}/posts

C. User Interactions

1. POST/api/posts/ {post_id} / like
 2. DELETE / api/posts/ {post_id} / unlike
 3. POST/api/posts/ {post_id} / comments
 4. GET/api/posts/ {post_id} / comments
 5. PUT/api/ comments / {comment_id}
 6. DELETE / api/comments/{post_id}/ {comment_id}
 7. POST/api/ users / {user_id} / follow
 8. DELETE / api / users / {user_id} / unfollow

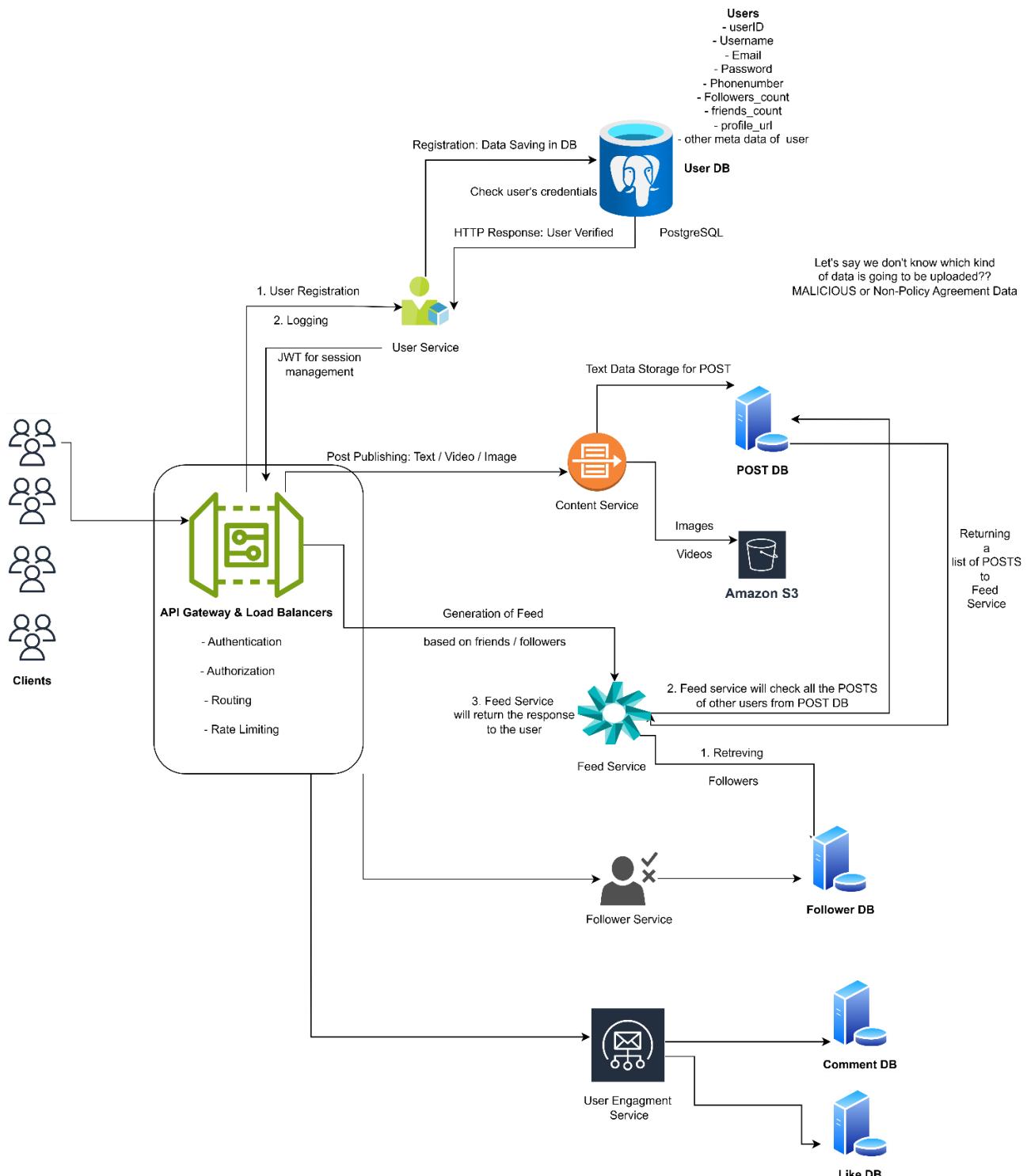
5- High-Level Design

Now According to the functional requirement of the system, we can identify that : We have to follow a distributed / micro-services approach not the monolithic one.



6- Low- Level Design

USER SERVICE

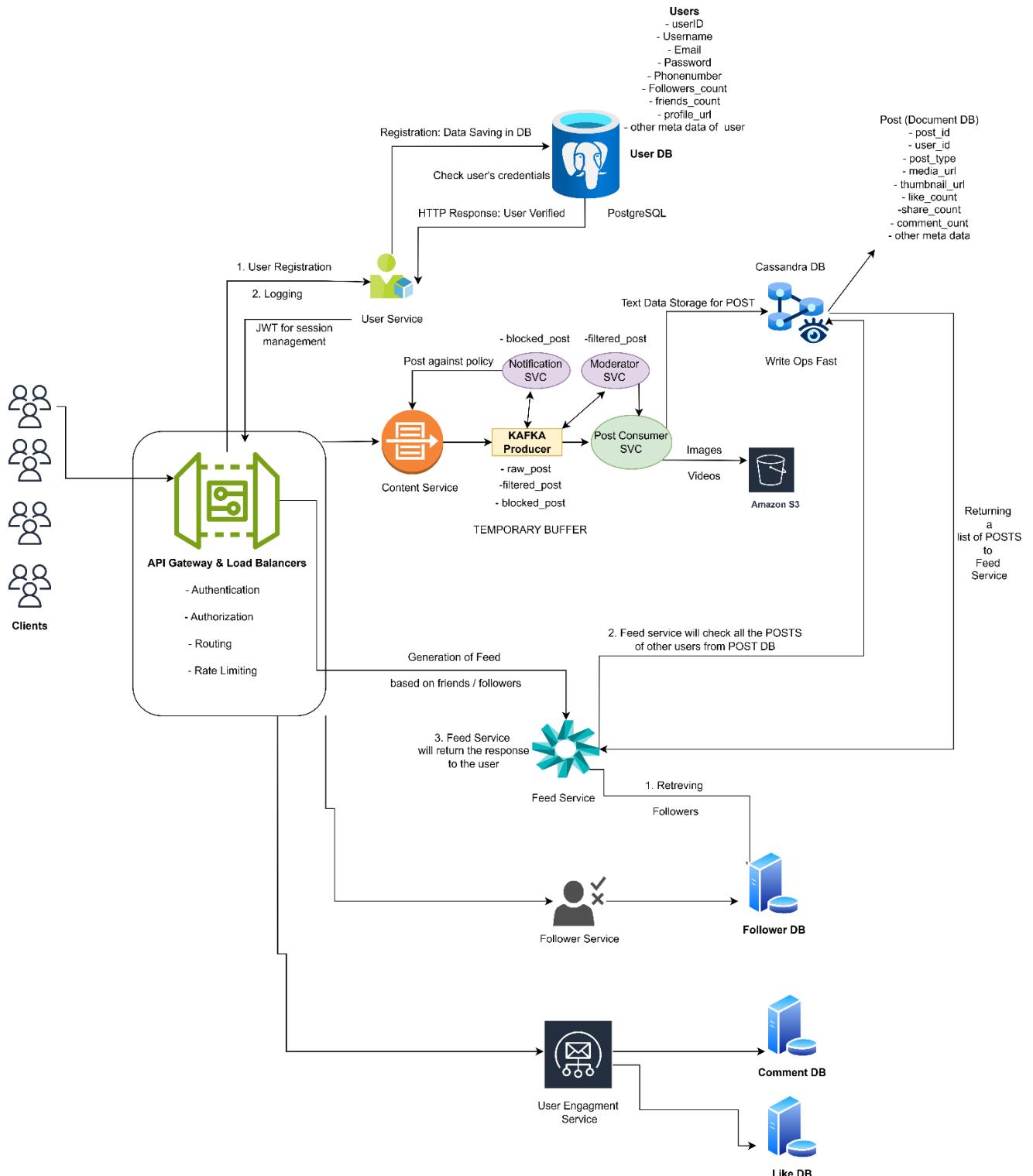




DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.

CONTENT SERVICE

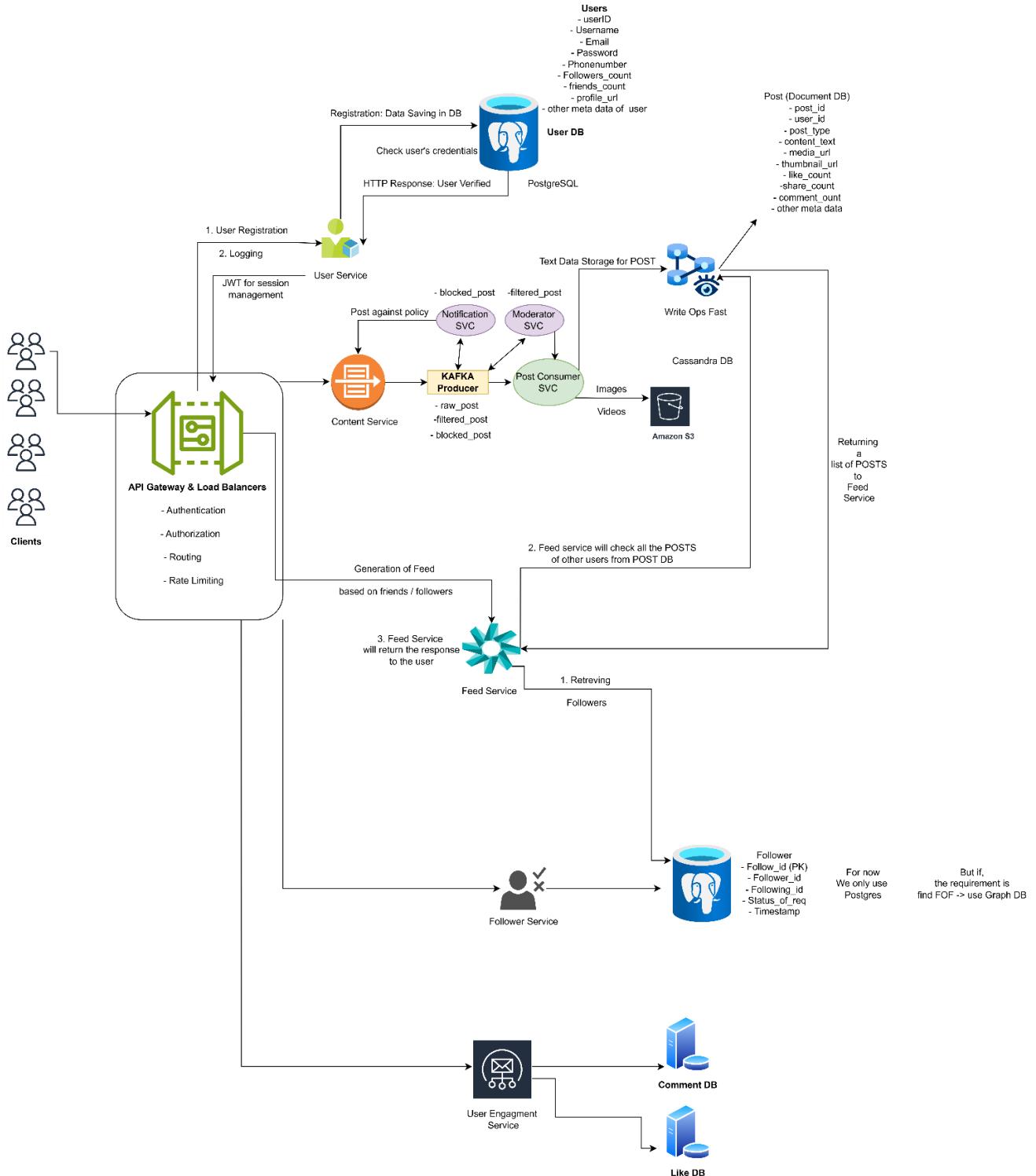




DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.

FOLLOWER SERVICE

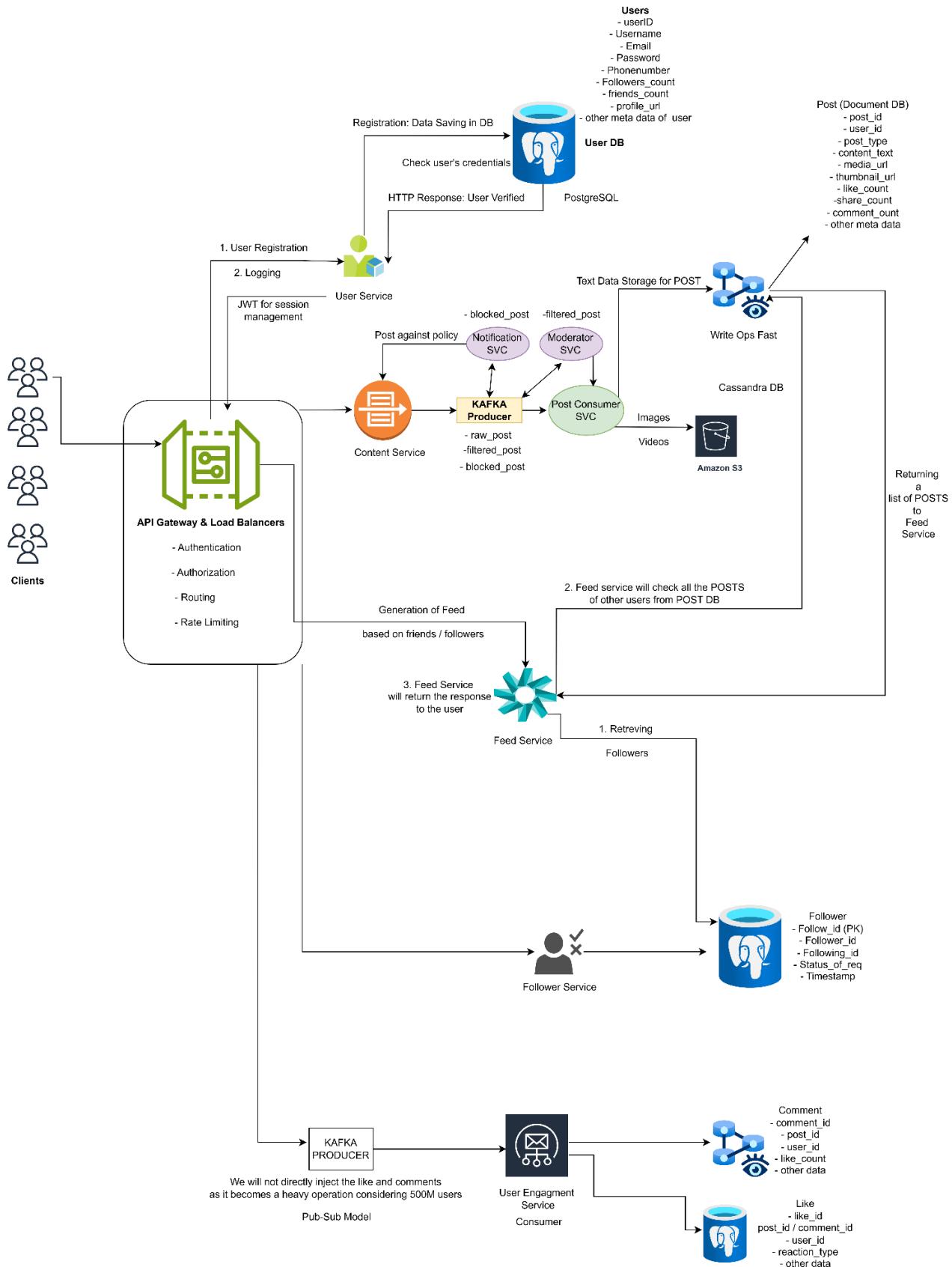




DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.

ENGAGEMENT SERVICE





DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.

FEED SERVICE

Problems With Feed Service

Feed service has to check data again and again from two databases, i.e., POST DB and FOLLOWER DB

Which cost hight database read / write operations, every time we have to generate the feed.

What if i get some of the pre-computed data for feed service rather than going again and again to POST DB and FOLLOWER DB

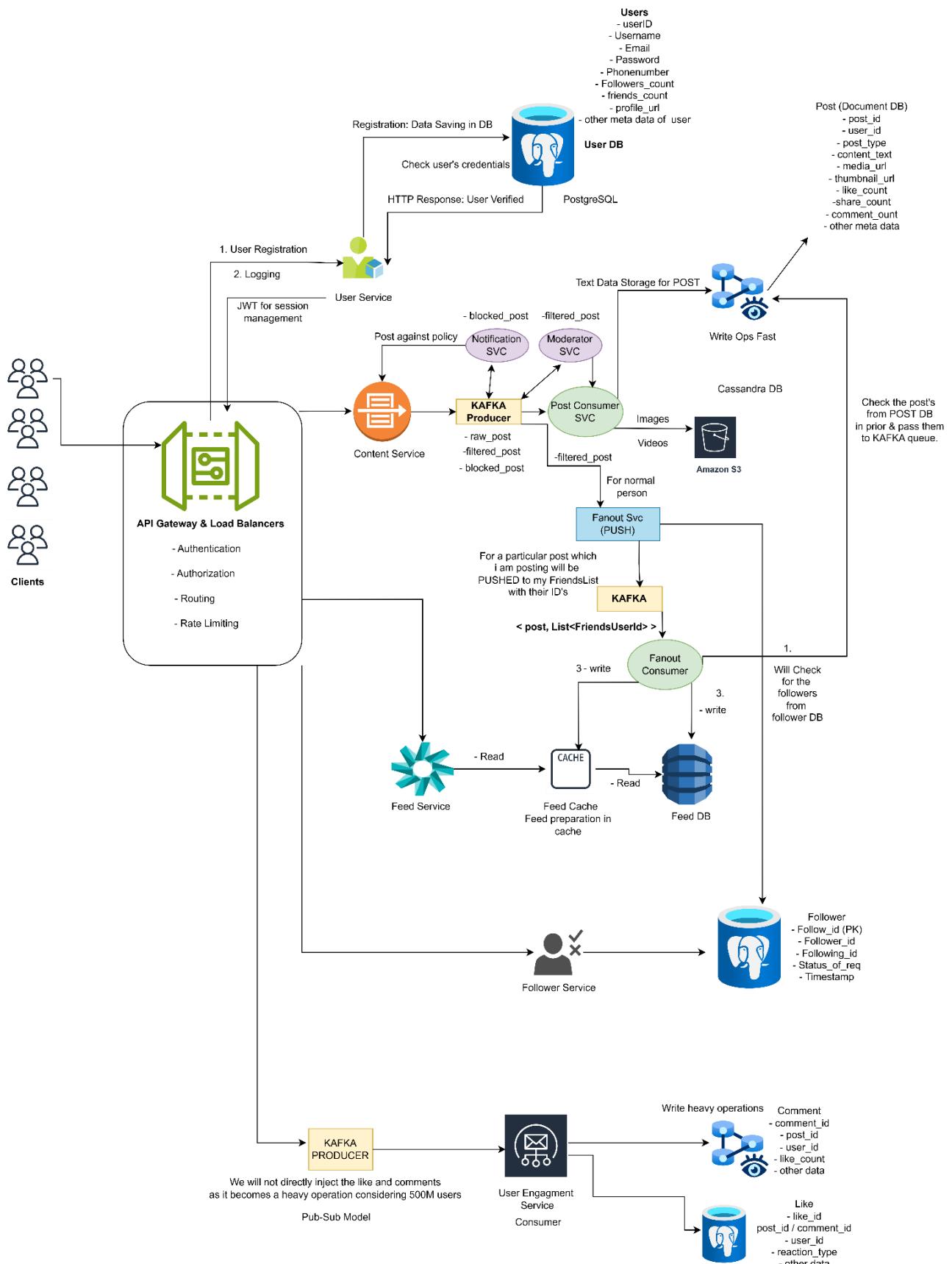
Now this is a common problem while designing the apps like social media & the solution is: **FANOUT MODEL**

User Type	Strategy Used	Reason
Normal Users	Fanout-on-Write (Push)	Their follower count is low, so pushing to all feeds is fast and cheap.
Celebrities	Fanout-on-Read (Pull)	We don't push their posts. Instead, when a follower loads their feed, the system "pulls" the celebrity's recent posts and merges them on the fly.



DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.

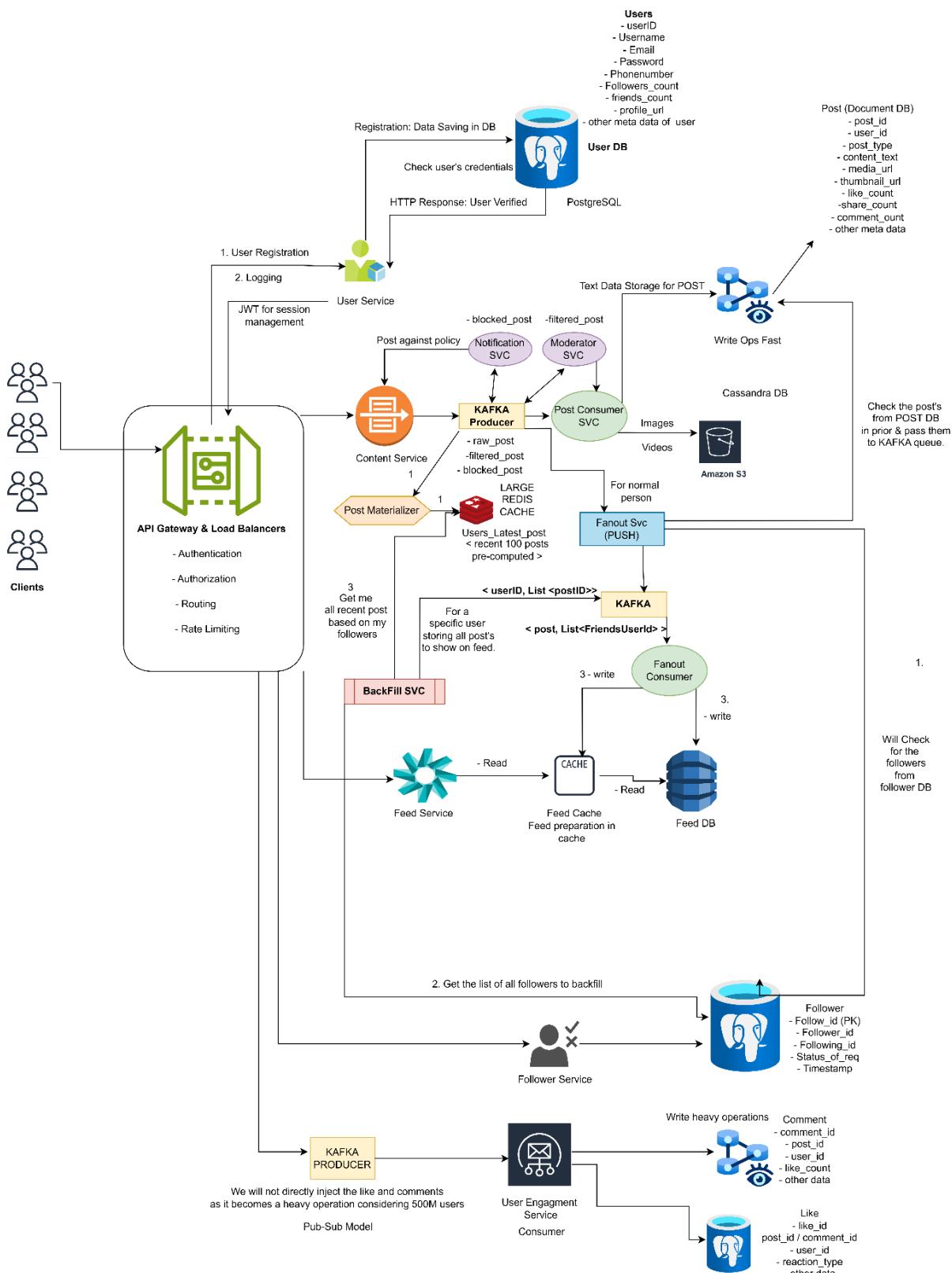


Problem: Whenever a new feed is generated after 50 posts, we have to follow the same process. So, when a user is about to finish seeing all the posts. We will use a concept called : **BACKFILL** using a **MATERIALIZER**



DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.



Post Materializer

1. It stores the recently scrolled posts of a user in the Redis cache.
 2. The Backfill service retrieves the list of followers from the Follower Database.
 3. Based on these followers, it fetches their recent posts from Redis.
- This approach avoids frequent access to the Post Database, enabling real-time feed generation with lower latency.



DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.

Backfill

When a user is about to consume the entire feed (for example, 50 posts), an additional API call is triggered to refill the feed using the Post Materializer and Redis cache.

In an app like social-media, we don't need data for everyone, we need data or post of the one's we follow the most.

Instead, of referring to the FOLLOWER DB again & again, we can maintain a cache of top followers we follow.

