DEPARTMENT OF
COMPUTER SCIENCE & ENGINEERING
Discover. Learn. Empower.

Experiment 3

Student Name: Devjot Singh
UID: 23BCS10864
Branch: BE-CSE
Section/Group: KRG 2-A
Semester: 6th
Date of Performance:28/01/2026
Subject Name: System Design
Subject Code: 23CSH-314

1. **Aim:** To design a social media platform similar to Facebook or Instagram

2. **Objective**:

- To understand social media application workflow.
- To design functional and non-functional requirements.
- To create system architecture (HLD).
- To design modules/classes (LLD).
- To implement core APIs for user authentication, posts, feed, likes, and comments.

3. **Tools Used:**

- Python – Backend logic implementation and URL generation algorithms.
- Flask – Lightweight web framework for developing RESTful APIs.
- Draw.io – Designing system architecture diagrams (HLD & LLD).

4. **System Requirements:**

A. Functional Requirements

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

B. Non-Functional Requirements

- Scalability
  The system should support up to 500 Million Daily Active Users (500M DAU).

• Availability and Consistency

Since this is a social media application, Availability is prioritized over Consistency.

Reason:

If the application is not operational when required, there is no purpose in developing it. Minor delays in content propagation are acceptable compared to complete downtime.

Example:If Instagram is down for 1 hour, it is a major issue.

However, if a post takes 500 ms to reach followers while the system remains available, it is acceptable Hence:

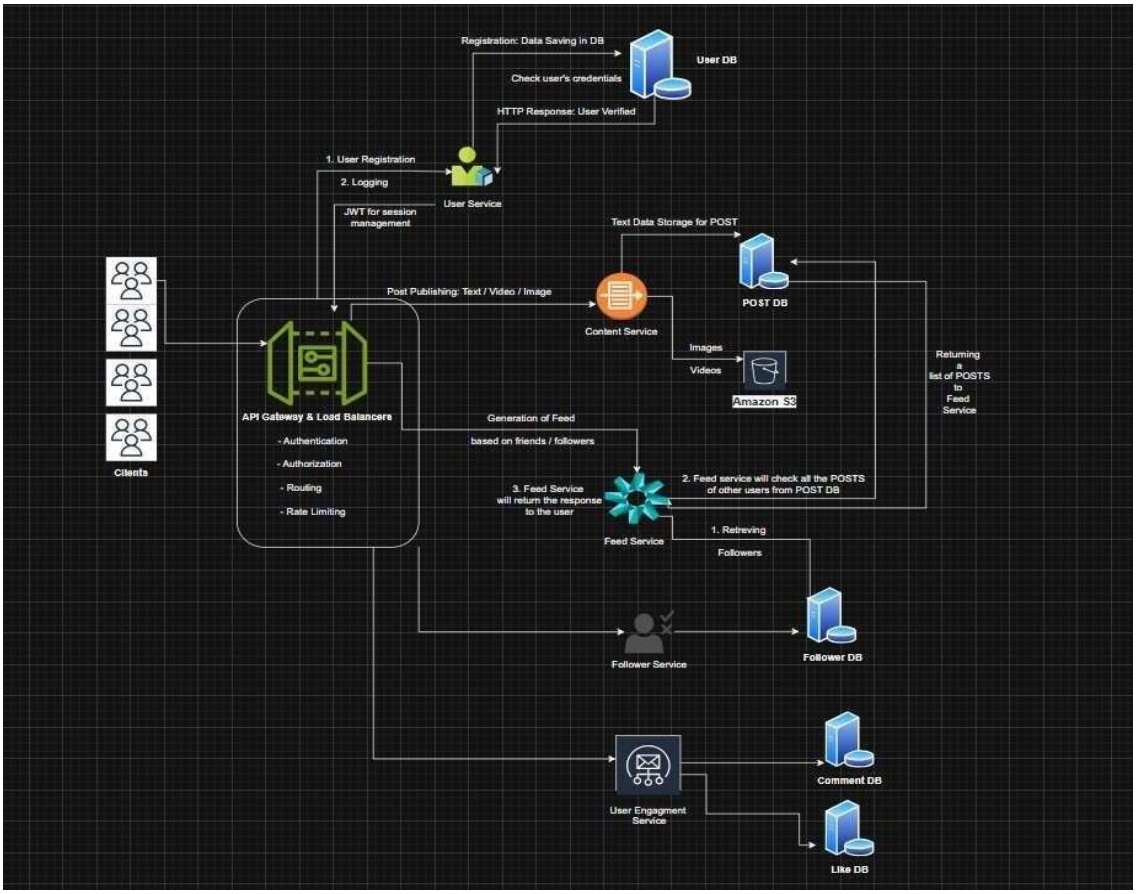
Availability >>> Consistency

• Latency

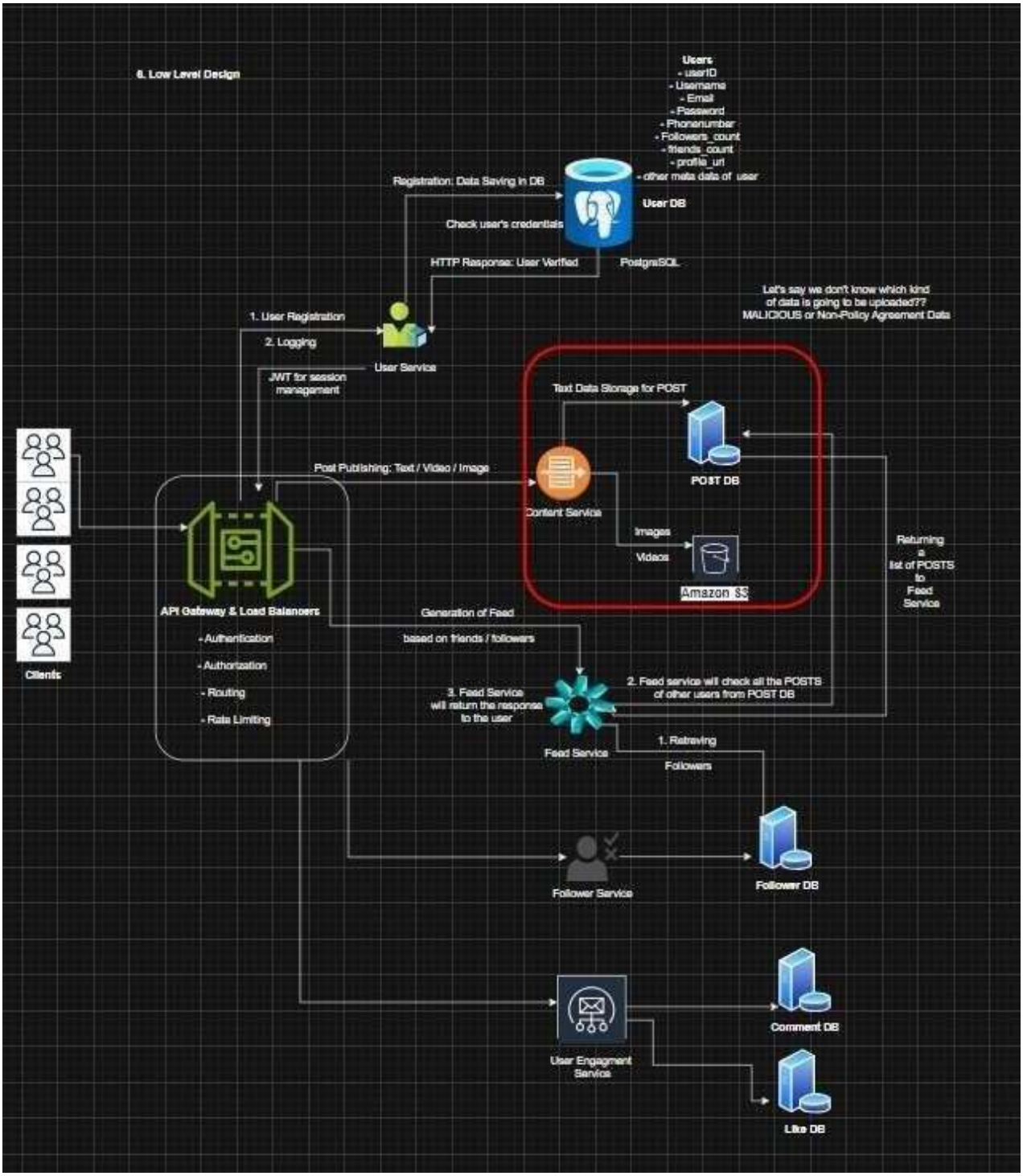
Uploading or publishing a post should take approximately 500 milliseconds to ensure a smooth user experience.

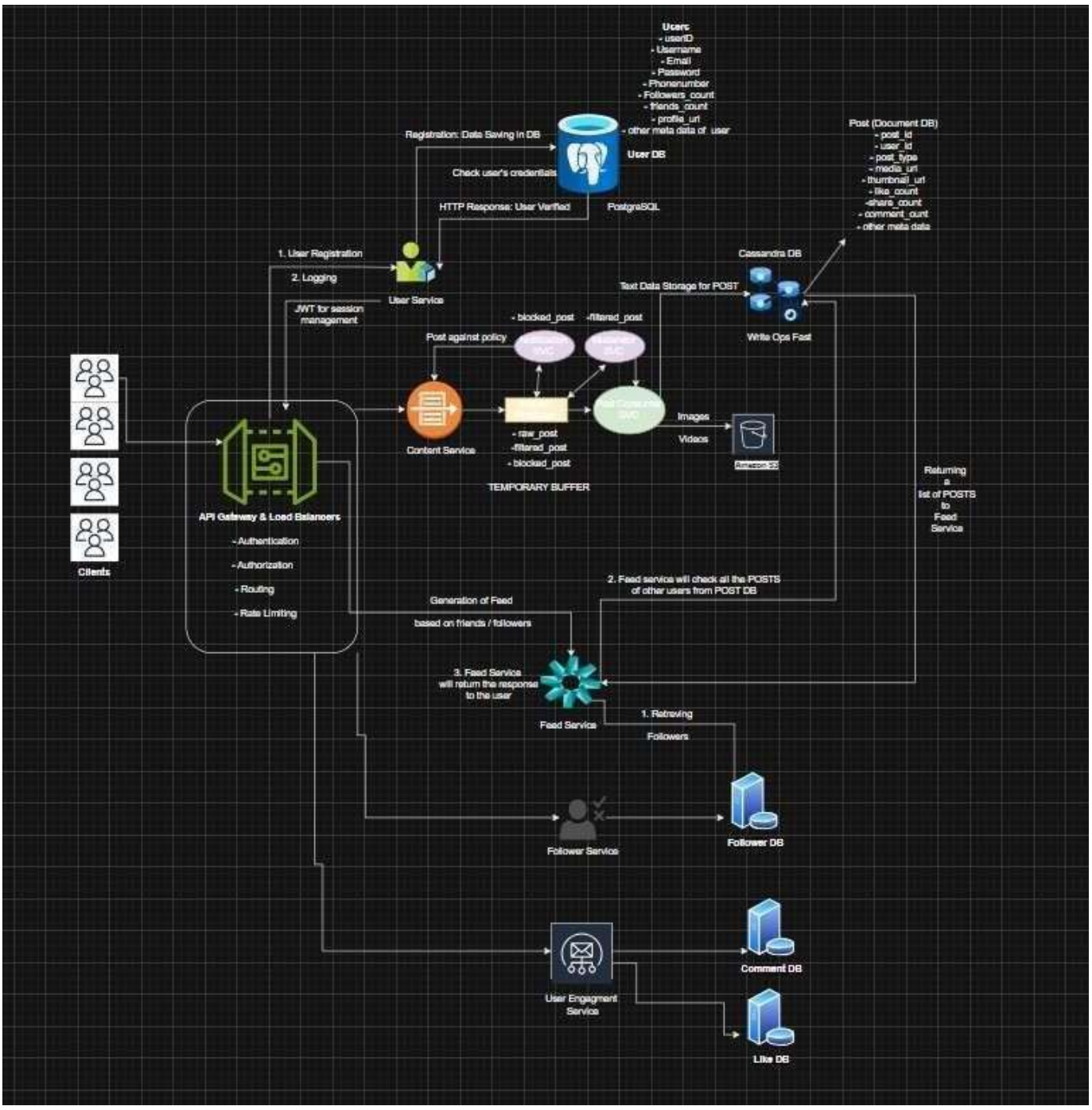
5. High Level Design (HLD):

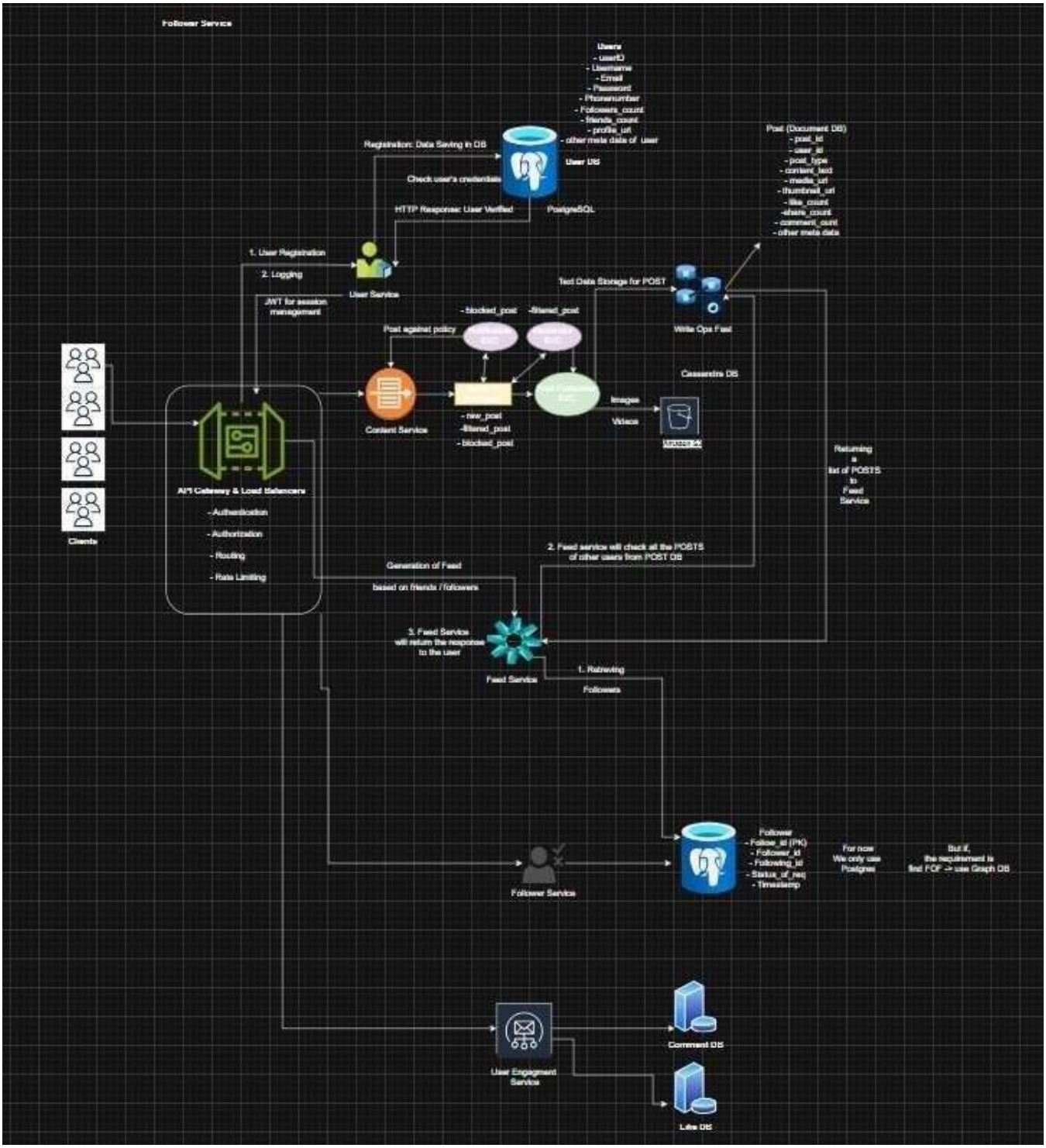
The system follows a MICRO-SERVICE ARCHITECTURE / DISTRIBUTED:

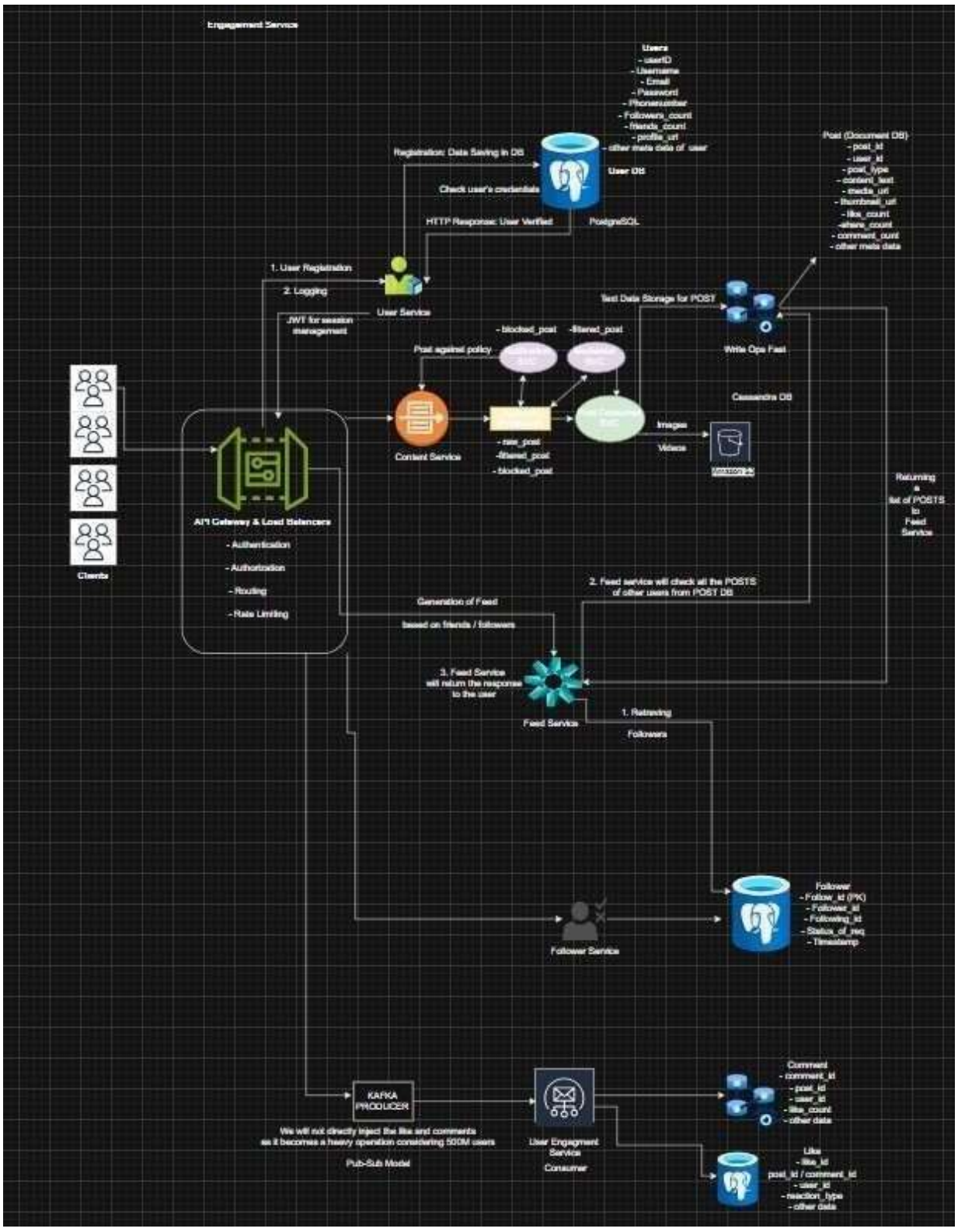


6. Low Level Design (LLD):

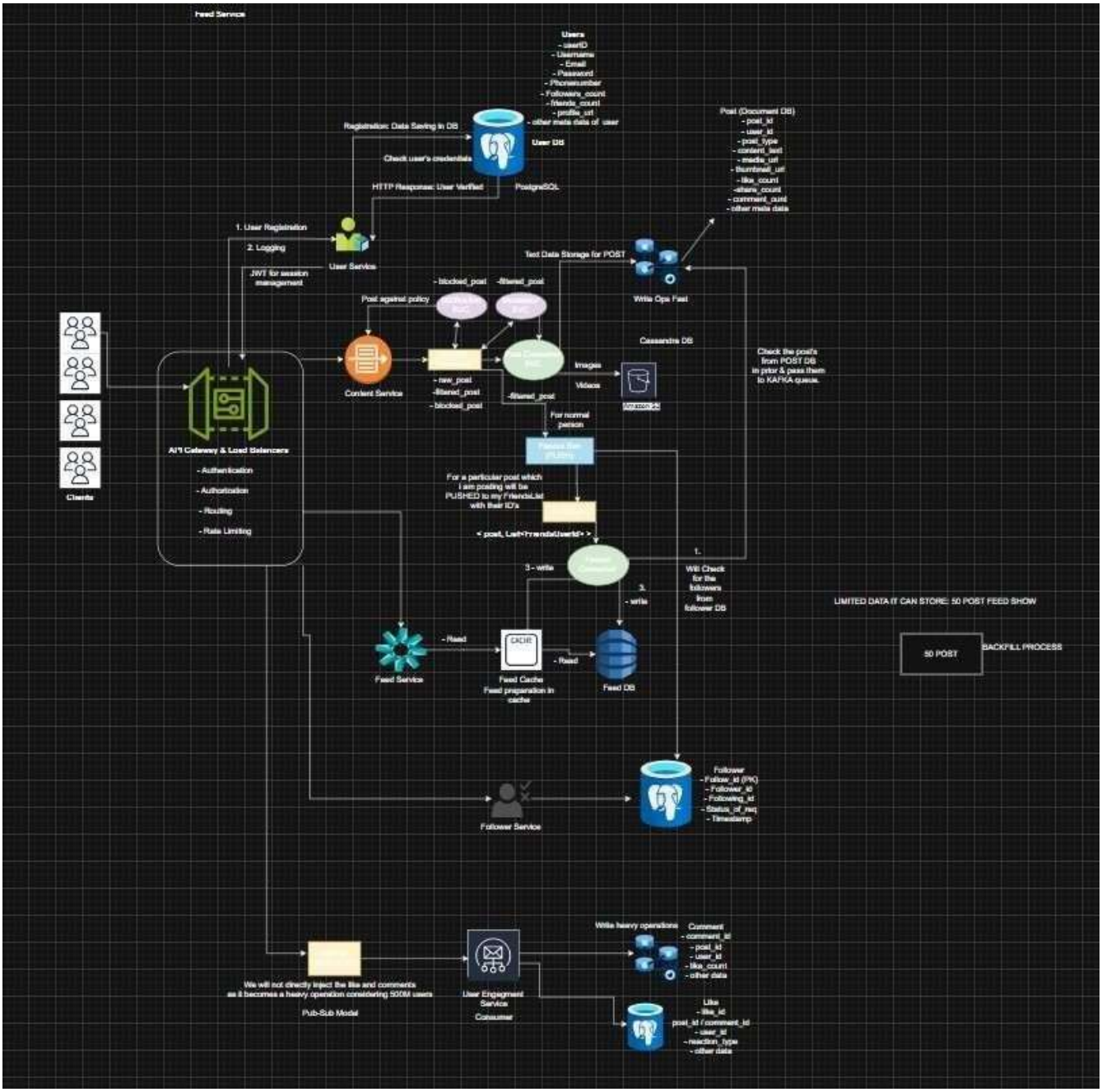



User:
- userID
- Username
- Email
- Password
- Phonenumber
- Followers_count
- friends_count
- profile_url
- other meta data of user
Registration: Data Saving in DB
User DB
Check user's credentials
HTTP Response: User Verified
PostgreSQL
Post (Document DB)
- post_id
- user_id
- post_type
- media_url
- thumbnail_url
- like_count
-share_count
- comment_count
- other meta data
1. User Registration
2. Logging
User Service
JWT for session management
Cassandra DB
Text Data Storage for POST
Write Ops Fast
- blocked_post
-filtered_post
Post against policy
Content Service
- raw_post
-filtered_post
- blocked_post
TEMPORARY BUFFER
Images
Videos
Amazon S3
Clients
API Gateway & Load Balancers
- Authentication
- Authorization
- Routing
- Rate Limiting
Returning a list of POSTS to Feed Service
2. Feed service will check all the POSTS of other users from POST DB
Generation of Feed
based on friends / followers
3. Feed Service will return the response to the user
Feed Service
1. Retreving
Followers
Follower Service
Follower DB
User Engagment Service
Comment DB
Like DB

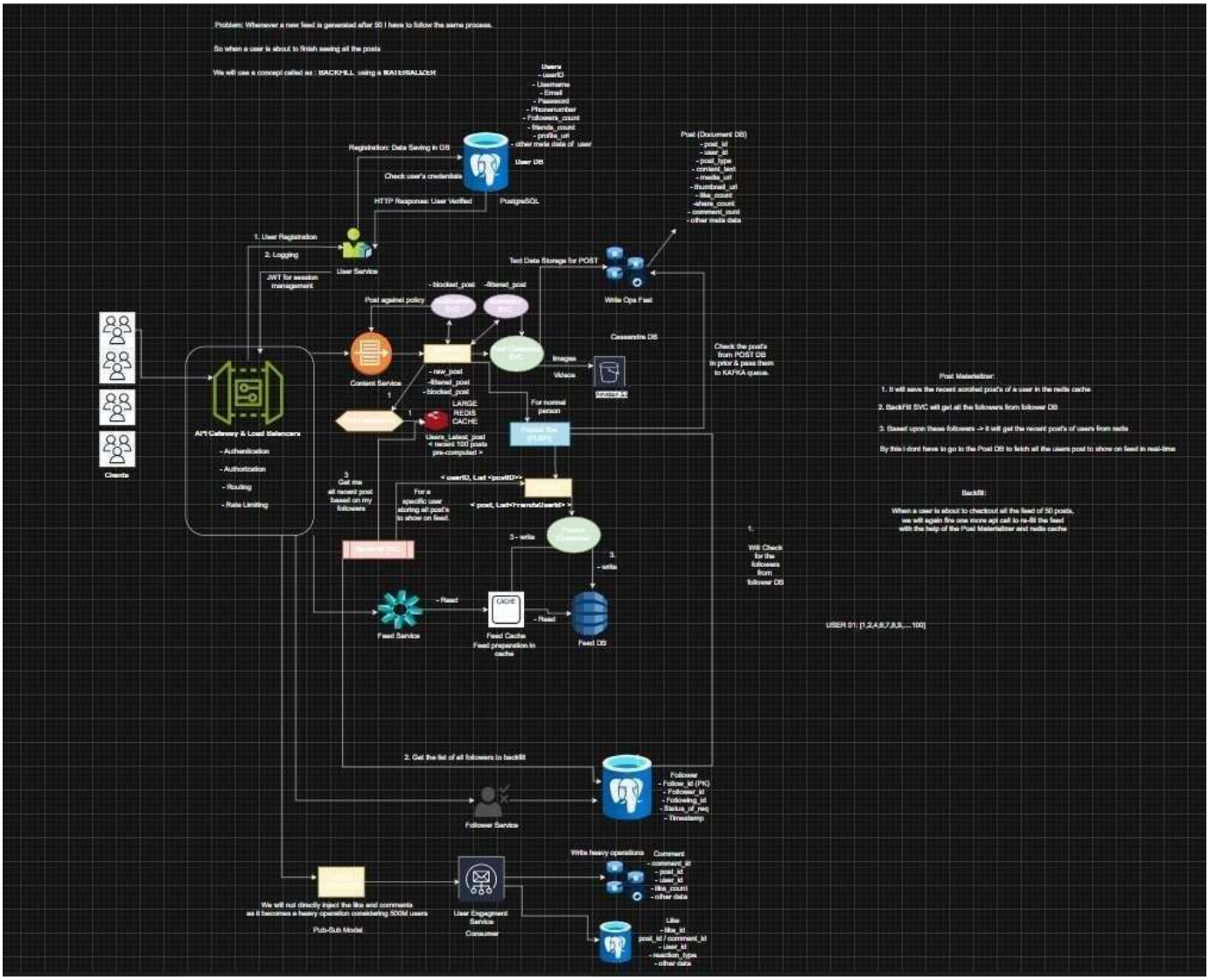

Follower Service
Users
- userID
- Username
- Email
- Password
- Phonenumber
- Followers_count
- friends_count
- profile_url
- other meta data of User
Registration: Data Saving in DB
User DB
Check user's credentials
HTTP Response: User Verified
PostgreSQL
Post (Document DB)
- post_id
- user_id
- post_type
- content_text
- media_url
- thumbnail_url
- like_count
- share_count
- comment_count
- other meta data
1. User Registration
2. Logging
User Service
JWT for session management
Text Data Storage for POST
Write Ops Feed
- blocked_post
- filtered_post
Post against policy
Cassandra DB
Content Service
- new_post
-filtered_post
- blocked_post
Images
Videos
Amazon S3
Clients
API Gateway & Load Balancers
- Authentication
- Authorization
- Routing
- Rate Limiting
Returning a list of POSTS to Feed Service
2. Feed service will check all the POSTS of other users from POST DB
Generation of Feed based on friends / followers
3. Feed Service will return the response to the User
Feed Service
1. Retrieving Followers
Follower Service
Follower
- Follow_id (PK)
- Follower_id
- Following_id
- Status_of_req
- Timestamp
For now We only use Postgres
But if, the requirement is find FOF -> use Graph DB
User Engagment Service
Comment DB
Like DB

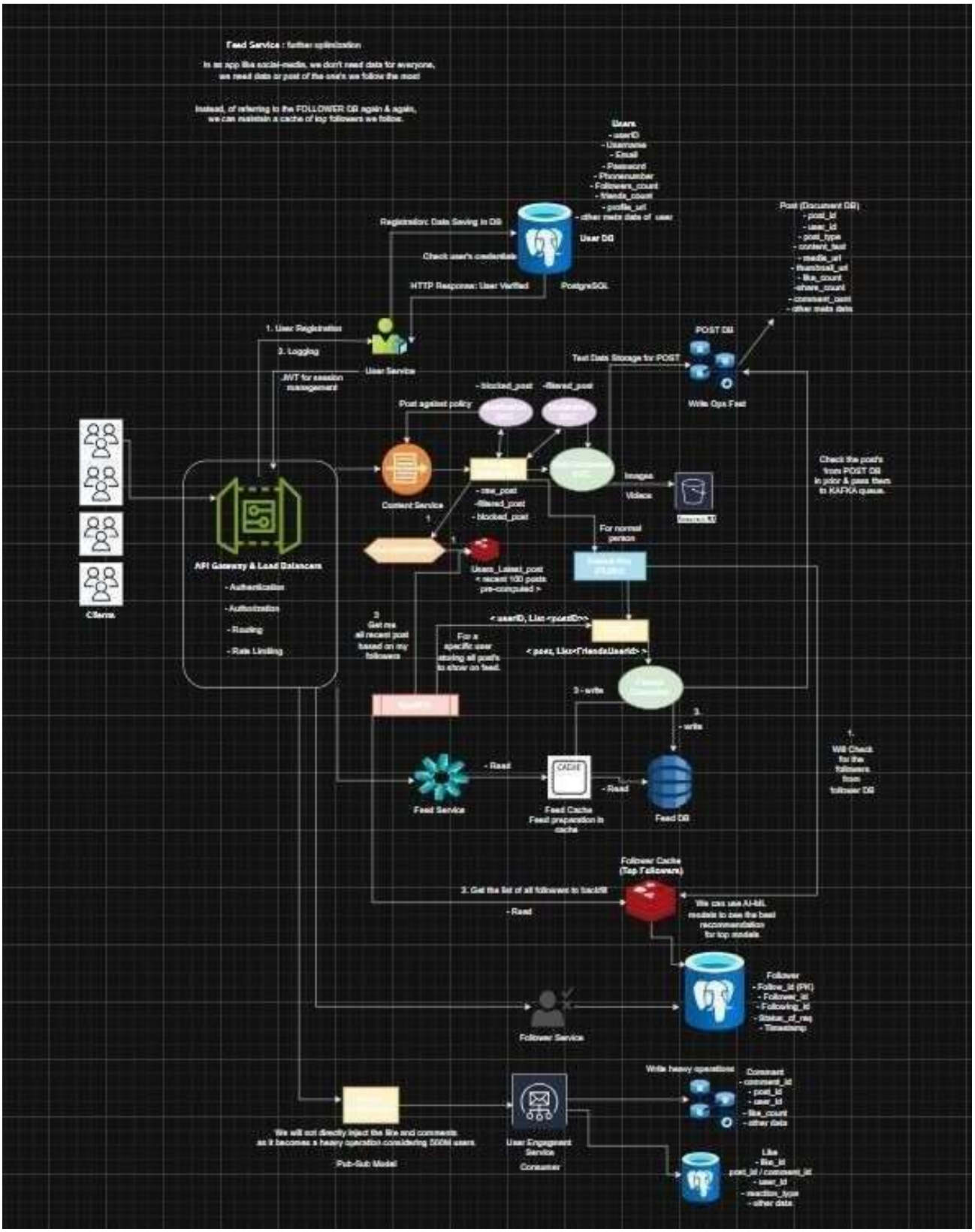

Engagement Service
Users
- userID
- Username
- Email
- Password
- Phonenumber
- Followers_count
- friends_count
- profile_url
- other meta data of user
Registration: Data Saving in DB
User DB
Check user's credentials
HTTP Response: User Verified
PostgreSQL
Post (Document DB)
- post_id
- user_id
- post_type
- content_text
- media_url
- thumbnail_url
- like_count
- share_count
- comment_count
- other meta data
1. User Registration
2. Logging
User Service
JWT for session management
Text Data Storage for POST
Write Ops Fast
- blocked_post
- filtered_post
Post against policy
Cassandra DB
Content Service
Images
Videos
- raw_post
-filtered_post
- blocked_post
Returning a list of POSTS to Feed Service
Clients
API Gateway & Load Balancers
- Authentication
- Authorization
- Routing
- Rate Limiting
2. Feed service will check all the POSTS of other users from POST DB
Generation of Feed
based on friends / followers
3. Feed Service will return the response to the user
Feed Service
1. Retreiving
Followers
Follower
- Follow_id (PK)
- Follower_id
- Following_id
- Status_of_req
- Timestamp
Follower Service
KAFKA PRODUCER
We will not directly inject the like and comments as it becomes a heavy operation considering 500M users
Pub-Sub Model
User Engagment Service
Consumer
Comment
- comment_id
- post_id
- user_id
- like_count
- other data
Like
- like_id
post_id / comment_id
- user_id
- reaction_type
- other data

DEPARTMENT OF
COMPUTER SCIENCE & ENGINEERING



DEPARTMENT OF
COMPUTER SCIENCE & ENGINEERING




Feed Service : further optimization
In an app like social-media, we don't need data for everyone, we need data or post of the user's we follow the most
Instead, of referring to the FOLLOWER DB again & again, we can maintain a cache of top followers we follow.
Users
- userID
- Username
- Email
- Password
- Phonenumber
- Followers_count
- friends_count
- profile_url
- other meta data of user
Registration: Data Saving in DB
User DB
Check user's credentials
HTTP Response: User Verified
PostgreSQL
Post (Document DB)
- post_id
- user_id
- post_type
- content_text
- media_url
- thumbnail_url
- like_count
- share_count
- comment_count
- other meta data
1. User Registration
2. Logging
JWT for session management
User Service
POST DB
Text Data Storage for POST
Write Ops Feed
- blocked_post
- filtered_post
Post against policy
Content Service
- raw_post
- filtered_post
- blocked_post
Images
Videos
Check the post's from POST DB in prior & pass them to KAFKA queue.
For normal person
Users_Latest_post
< recent 100 posts pre-computed >
API Gateway & Load Balancers
- Authentication
- Authorization
- Routing
- Rate Limiting
Clients
3
Get me all recent post based on my followers
For a specific user storing all post's to show on feed.
< userID, List <postID> >
< post, List<FriendsUserId> >
3 - write
3.
- write
1.
Will Check for the followers from follower DB
- Read
Feed Service
CACHE
Feed Cache
Feed preparation in cache
- Read
Feed DB
Follower Cache
(Top Followers)
2. Get the list of all followers to backfill
- Read
We can use AI-ML models to see the best recommendation for top models.
Follower
- Follow_id (PK)
- Follower_id
- Following_id
- Status_of_req
- Timestamp
Follower Service
Write heavy operations
Comment
- comment_id
- post_id
- user_id
- like_count
- other data
We will not directly inject the like and comments as it becomes a heavy operation considering 500M users.
Pub-Sub Model
User Engagement Service
Consumer
Like
- like_id
post_id / comment_id
- user_id
- reaction_type
- other data

6. Scalability Solution

- Use horizontal scaling by adding multiple application servers behind a load balancer.
- Implement database sharding to distribute user and post data across multiple databases.
- Use caching (Redis/Memcached) to reduce database load for frequently accessed data like feeds and profiles.
- Store media files (images/videos) using CDN and cloud storage to improve performance and reduce server load.
- Apply asynchronous processing and message queues (Kafka/RabbitMQ) for tasks like notifications and feed updates.

7. Learning Outcomes (What I Have Learnt)

- Understood the workflow of a social media application system.
- Learned to design functional and non-functional requirements.
- Gained knowledge of high-level and low-level system architecture.
- Learned how scalability and availability are achieved in large-scale systems.
- Developed understanding of core APIs for user management, posts, and feed handling.