Experiment 3

Student Name: Aaditya Anand **UID: 23BCS10446**
Branch: CSE **Section/Group: KRG 3-A**
Semester: 6th **Date of Performance:29/01/2026**
Subject Name: System Design **Subject Code: 23CSH-314**

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

2. Objective:

- To understand how a social media application works.
- To design both functional and non-functional requirements.
- To create the system architecture (High-Level Design – HLD).
- To design modules and classes (Low-Level Design – LLD).
- To implement core APIs for user authentication, posts, feed, likes, and comments.

3. Tools Used:

- Python – Used for implementing backend logic and URL generation algorithms.
- Flask – A lightweight web framework for developing RESTful APIs.
- Draw.io – Used for designing system architecture diagrams (HLD and LLD).

4. System Requirements:

A. Functional Requirements

- The client should be able to register and log in to the application.
- The client should be able to create posts such as text, images, and videos.
- The client should be able to follow other users or send friend requests.
- The client should be able to like and comment on posts.
- The client should be able to view a feed containing 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.

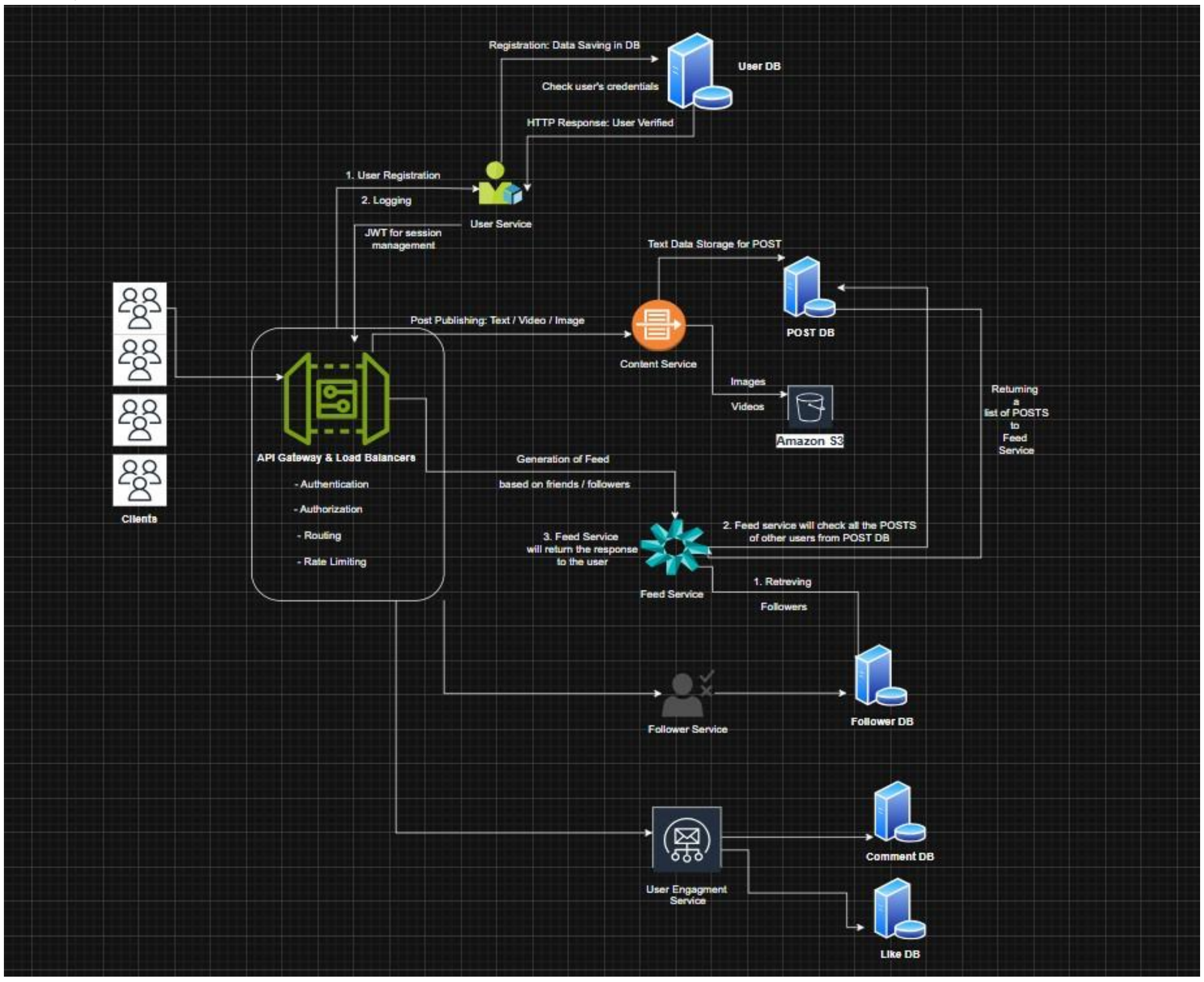
Reason:
If the application is not operational when required, there is no purpose in developing it. Minor delays in content propagation are acceptable compared to complete downtime.

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

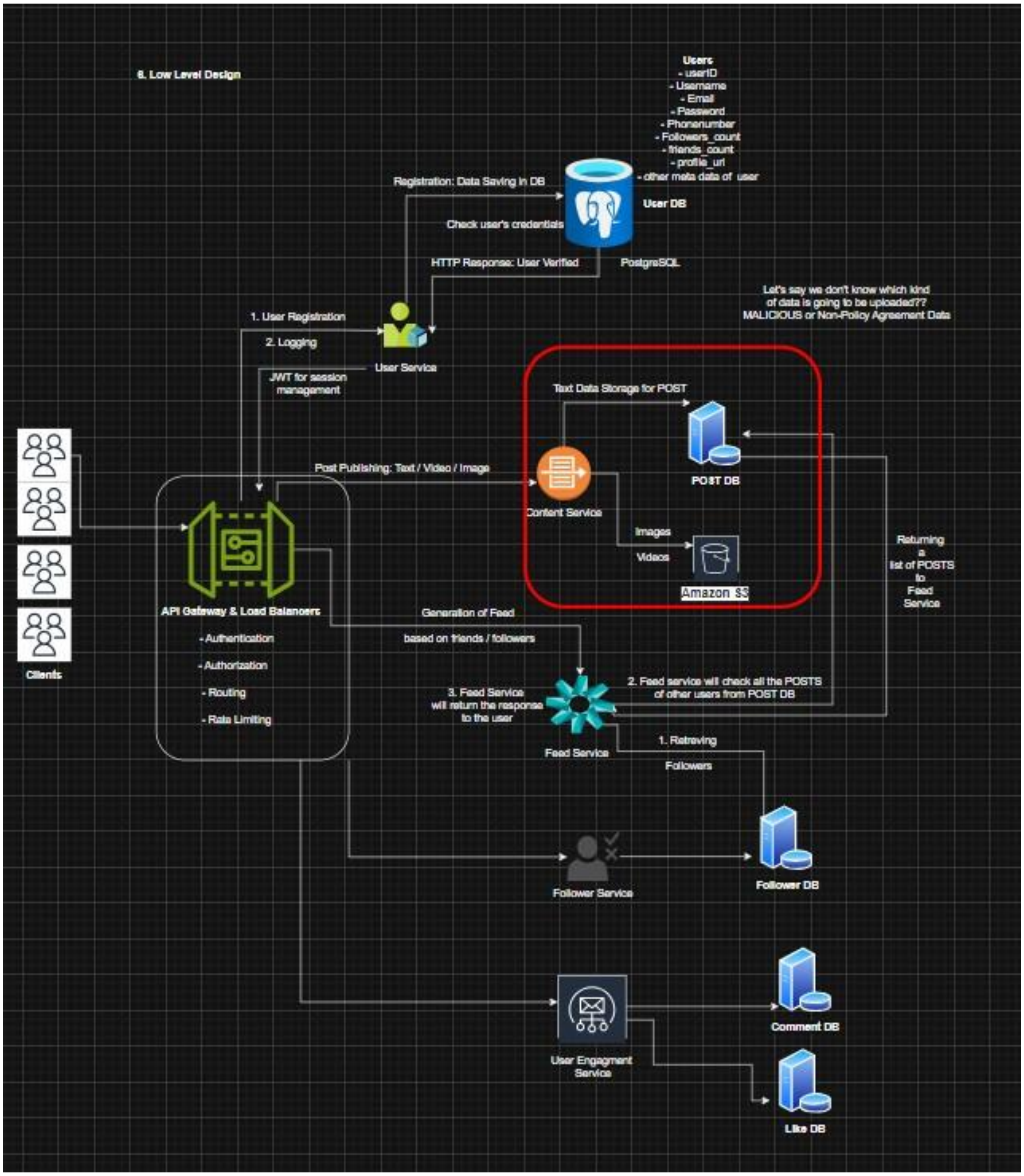
- **Latency**
  Uploading or publishing a post should take approximately 500 milliseconds to ensure a smooth user experience.

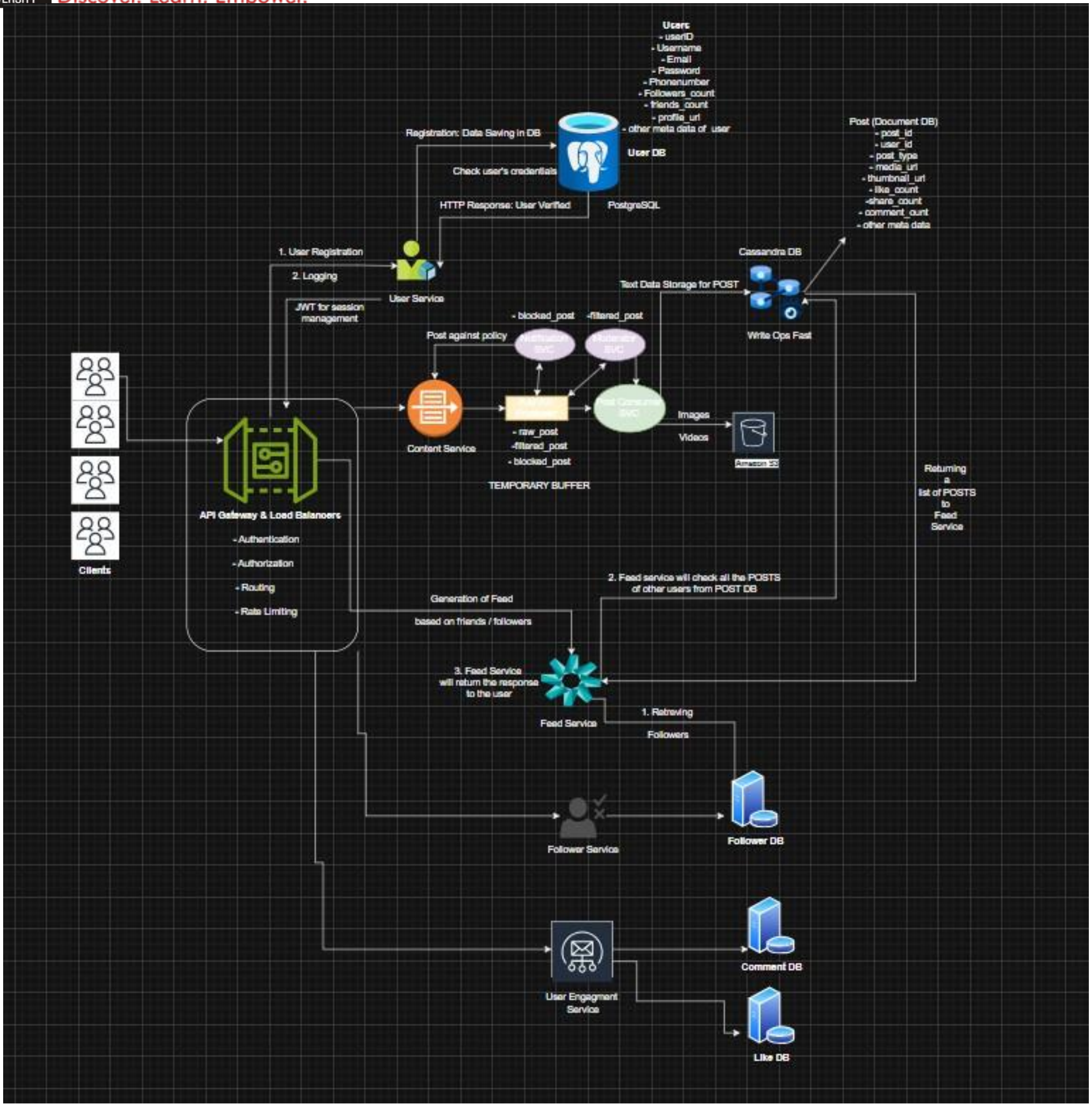
5. High Level Design (HLD):

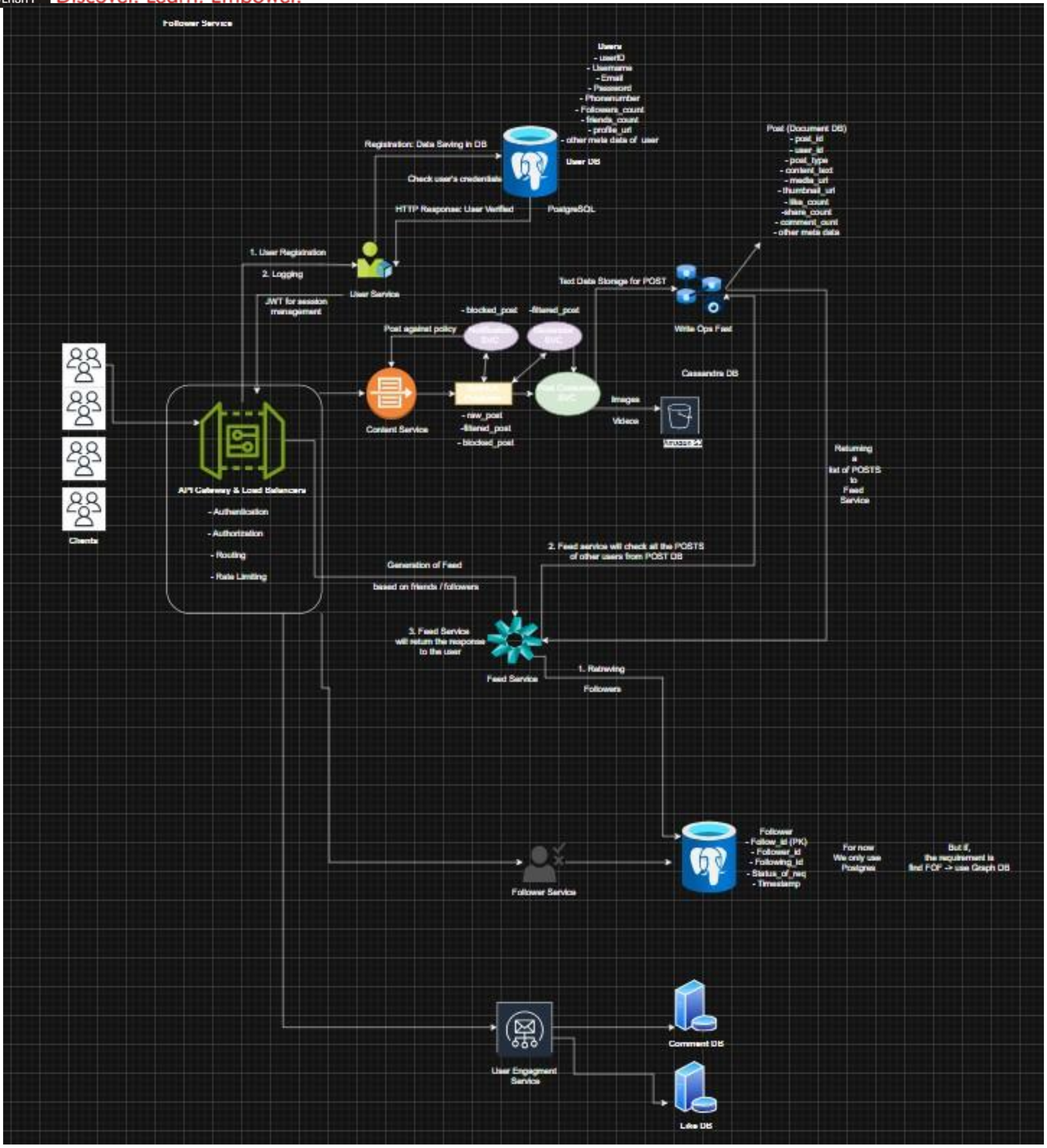
The system follows a MICRO-SERVICE ARCHITECTURE / DISTRIBUTED:

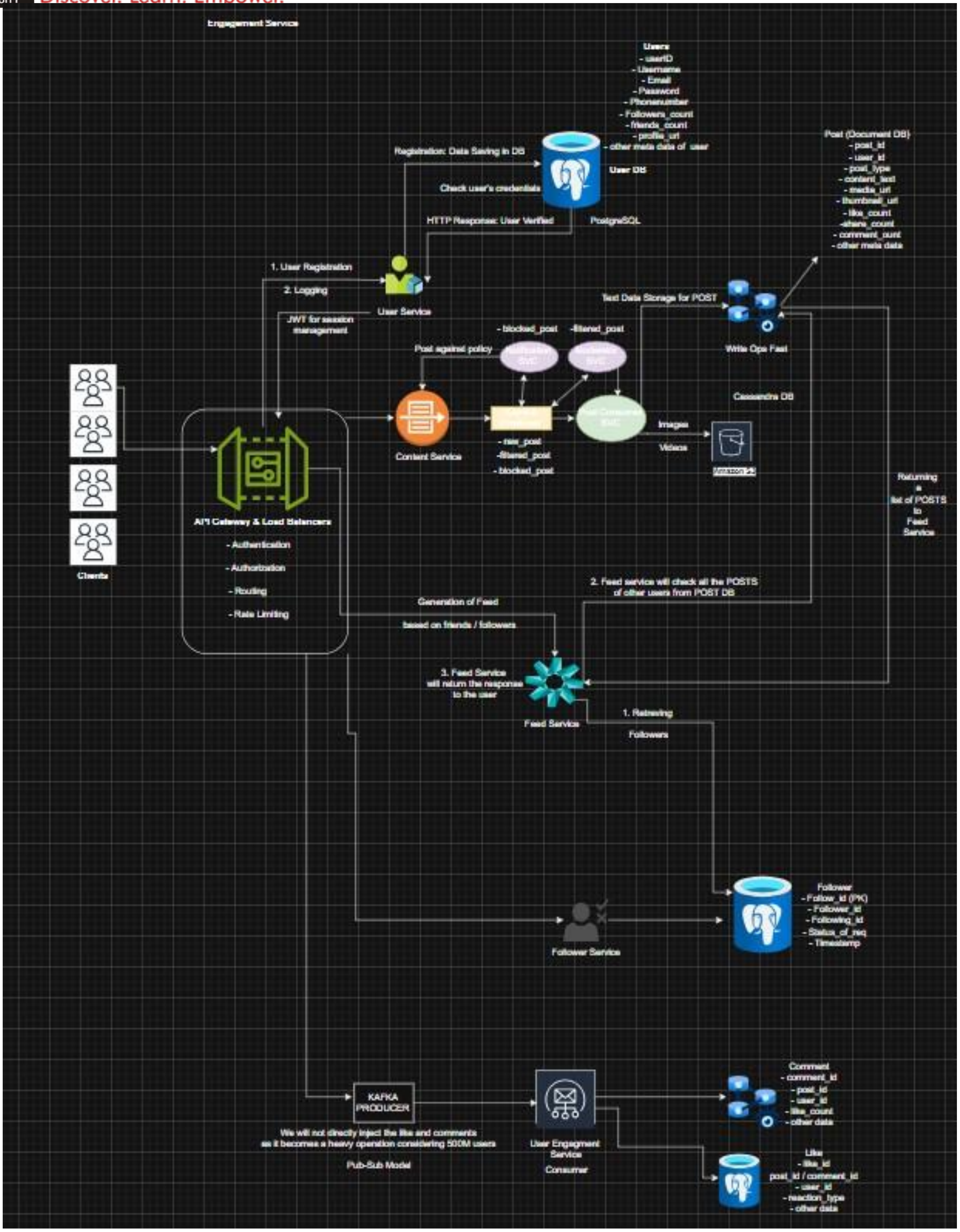

Registration: Data Saving in DB
User DB
Check user's credentials
HTTP Response: User Verified
1. User Registration
2. Logging
User Service
JWT for session management
Text Data Storage for POST
POST DB
Post Publishing: Text / Video / Image
Content Service
Images
Videos
Amazon S3
Returning a list of POSTS to Feed Service
Clients
API Gateway & Load Balancers
- Authentication
- Authorization
- Routing
- Rate Limiting
Generation of Feed
based on friends / followers
2. Feed service will check all the POSTS of other users from POST DB
3. Feed Service will return the response to the user
Feed Service
1. Retreving
Followers
Follower Service
Follower DB
Comment DB
User Engagment Service
Like DB

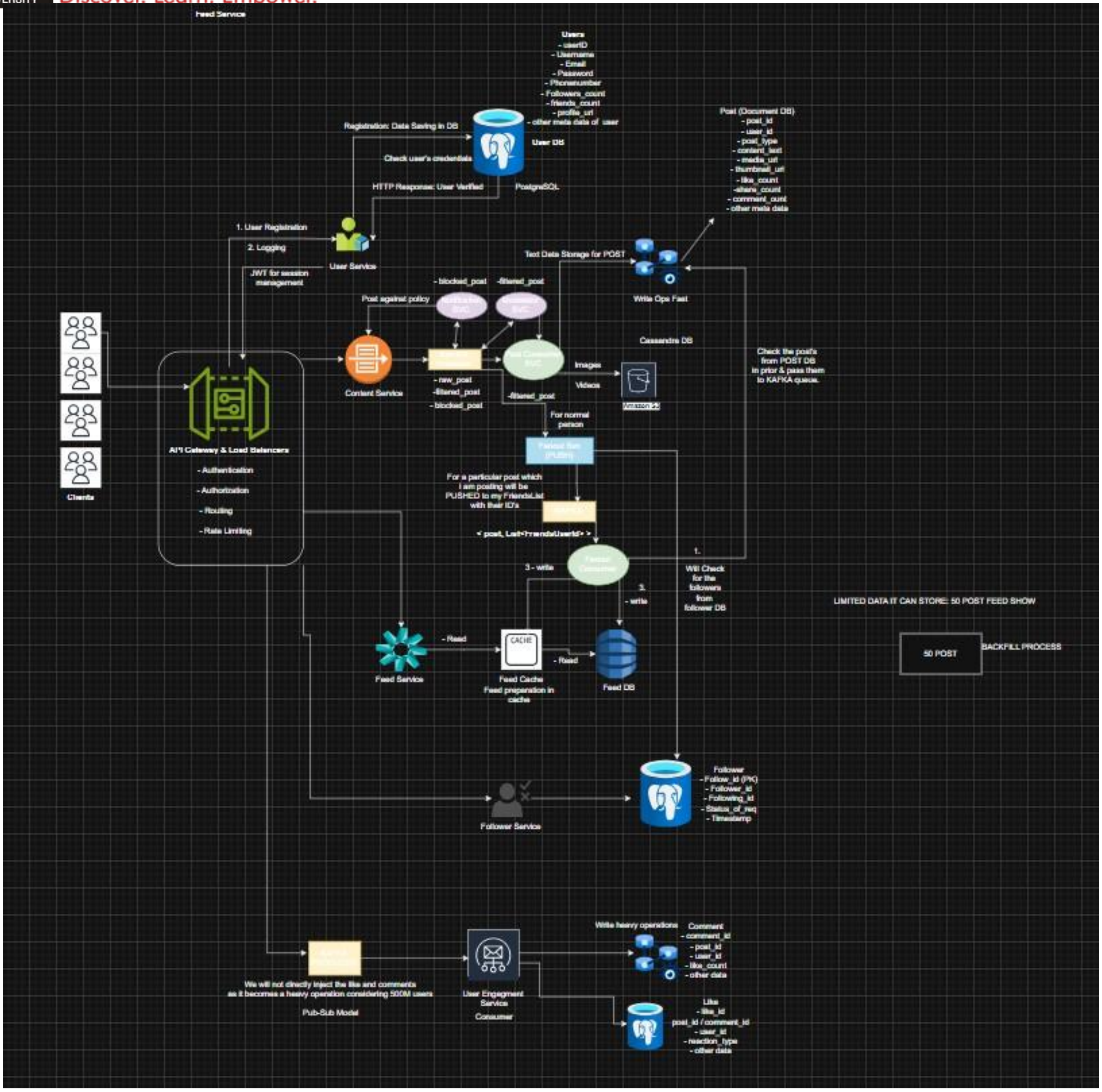
6. Low Level Design (LLD):

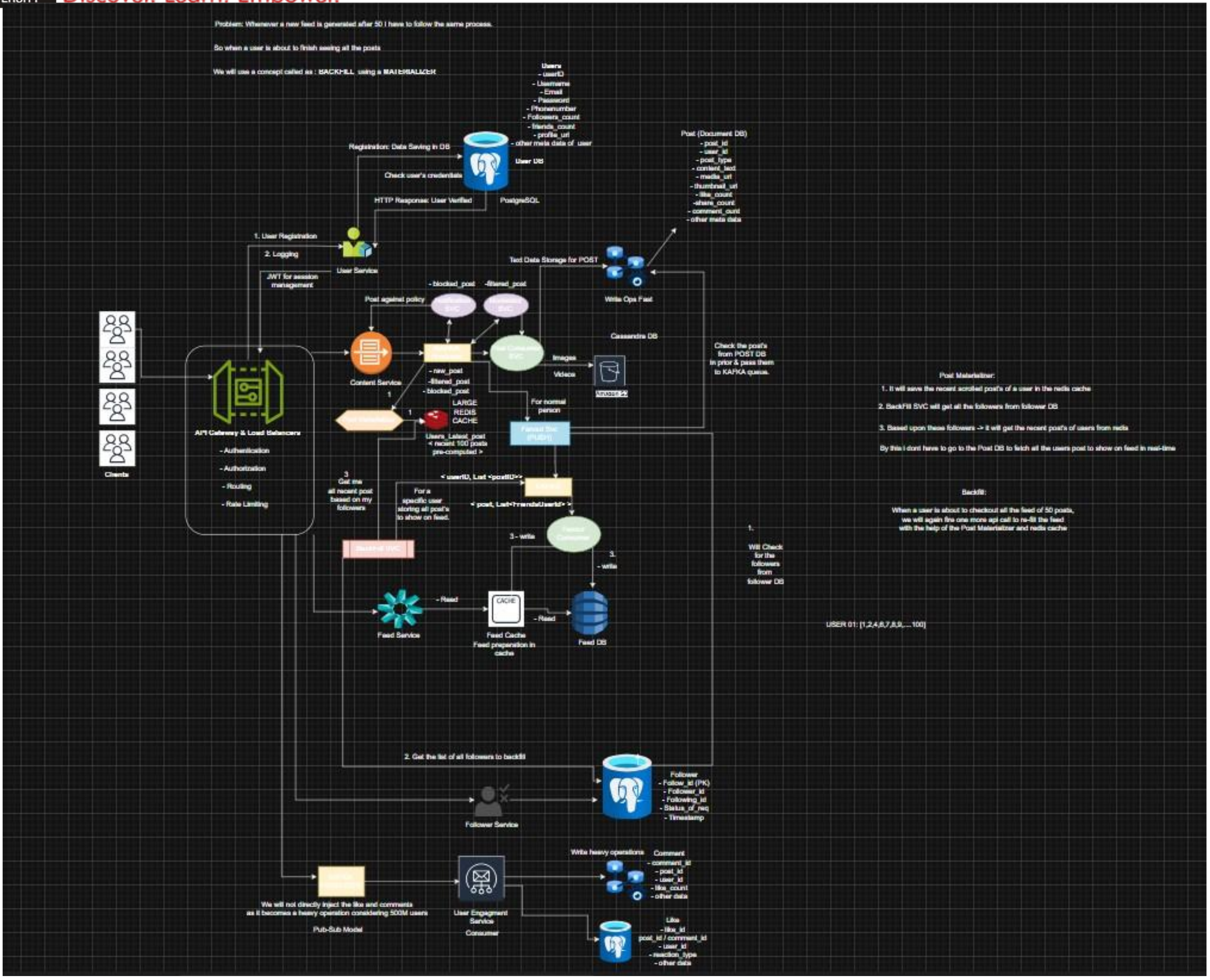

6. Low Level Design
Users
- userID
- Username
- Email
- Password
- Phonenumber
- Followers_count
- friends_count
- profile_url
- other meta data of user
Registration: Data Saving in DB
Check user's credentials
User DB
HTTP Response: User Verified
PostgreSQL
Let's say we don't know which kind of data is going to be uploaded?? MALICIOUS or Non-Policy Agreement Data
1. User Registration
2. Logging
User Service
JWT for session management
Text Data Storage for POST
POST DB
Post Publishing: Text / Video / Image
Content Service
Images
Videos
Amazon S3
Returning a list of POSTS to Feed Service
API Gateway & Load Balancers
- Authentication
- Authorization
- Routing
- Rate Limiting
Clients
Generation of Feed
based on friends / followers
2. Feed service will check all the POSTS of other users from POST DB
3. Feed Service will return the response to the user
Feed Service
1. Retreving
Followers
Follower Service
Follower DB
Comment DB
User Engagment Service
Like DB


Users
- userID
- Username
- Email
- Password
- Phonenumber
- Followers_count
- friends_count
- profile_url
- other meta data of user
Registration: Data Saving in DB
Check user's credentials
HTTP Response: User Verified
User DB
PostgreSQL
Post (Document DB)
- post_id
- user_id
- post_type
- media_url
- thumbnail_url
- like_count
-share_count
- comment_ount
- 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
Comment DB
User Engagment Service
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_cunt
- 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
Images
Videos
Amazon S3
Content Service
- raw_post
-filtered_post
- blocked_post
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
- 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
Follower Service
Comment DB
User Engagment Service
Like DB


Engagement Service
Users
- userID
- Username
- Email
- Password
- Phonenumber
- Followers_count
- friends_count
- profile_url
- other meta data of user
User DB
PostgreSQL
Registration: Data Saving in DB
Check user's credentials
HTTP Response: User Verified
Post (Document DB)
- post_id
- user_id
- post_type
- content_text
- media_url
- thumbnail_url
- like_count
-share_count
- comment_ount
- 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
Images
Videos
Amazon S3
Content Service
- raw_post
-filtered_post
- blocked_post
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
- 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


Feed Service
Users
- userID
- Username
- Email
- Password
- Phonenumber
- Followers_count
- friends_count
- profile_url
- other meta data of user
Registration: Data Saving in DB
Check user's credentials
HTTP Response: User Verified
User DB
PostgreSQL
Post (Document DB)
- post_id
- user_id
- post_type
- content_text
- media_url
- thumbnail_url
- like_count
-share_count
- comment_ount
- 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
Content Service
- new_post
-filtered_post
- blocked_post
-filtered_post
Cassandra DB
Images
Videos
Amazon S3
Check the post's from POST DB in prior & pass them to KAFKA queue.
For normal person
Clients
API Gateway & Load Balancers
- Authentication
- Authorization
- Routing
- Rate Limiting
For a particular post which I am posting will be PUSHED to my FriendsList with their ID's
< post, List<FriendsUserId> >
1.
Will Check for the followers from follower DB
3 - write
3.
- write
LIMITED DATA IT CAN STORE: 50 POST FEED SHOW
50 POST
BACKFILL PROCESS
- Read
CACHE
- Read
Feed Service
Feed Cache
Feed preparation in cache
Feed DB
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 Engagment Service
Consumer
Like
- like_id
post_id / comment_id
- user_id
- reaction_type
- other data



Problem: Whenever a new feed is generated after 50 I have to follow the same process.

So when a user is about to finish seeing all the posts

We will use a concept called as : BACKFILL using a MATERIALIZER

Post Materializer:

1. It will save the recent scrolled post's of a user in the redis cache
2. BackFill SVC will get all the followers from follower DB
3. Based upon these followers -> it will get the recent post's of users from redis

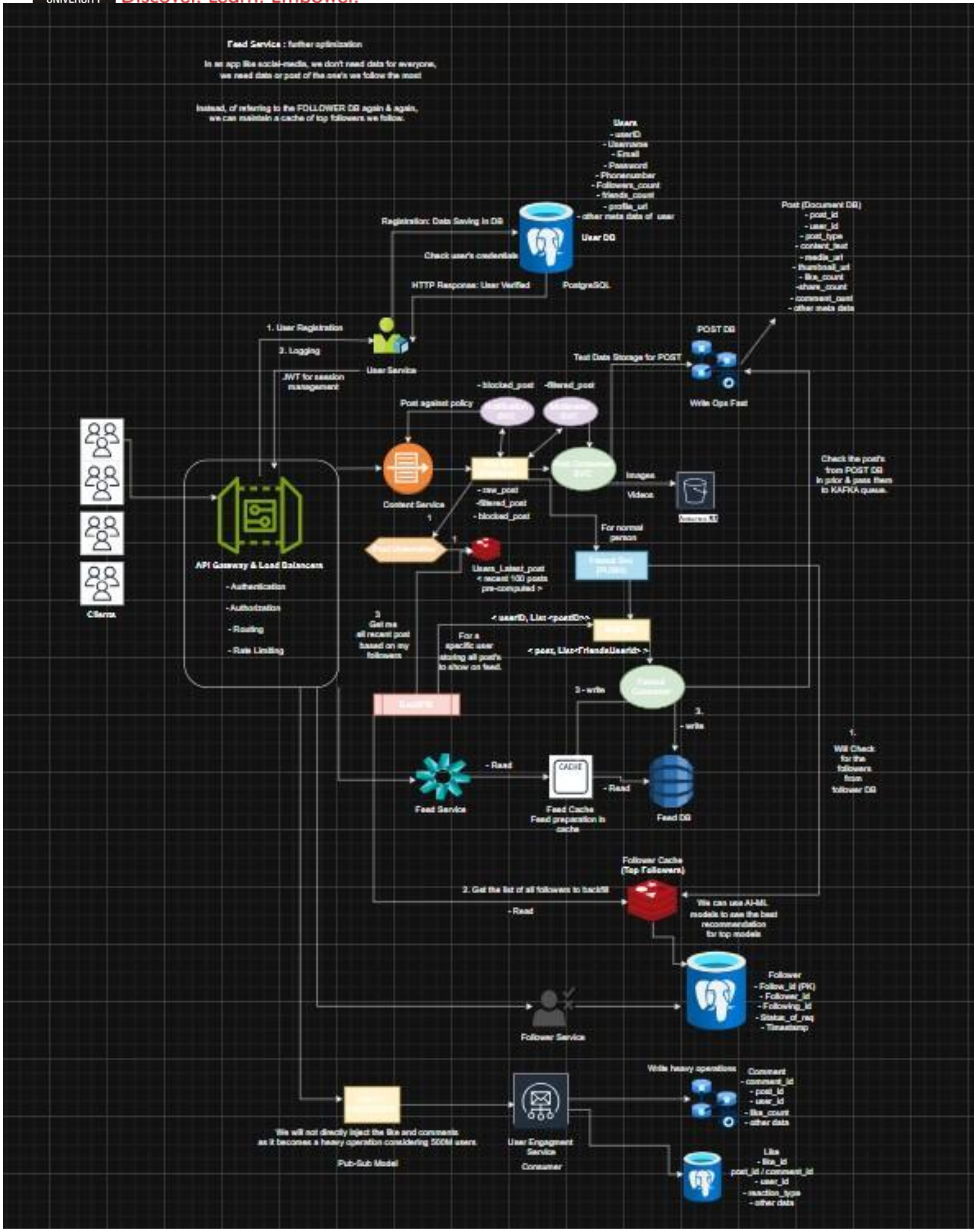
By this I dont have to go to the Post DB to fetch all the users post to show on feed in real-time

Backfill:

When a user is about to checkout all the feed of 50 posts, we will again fire one more api call to re-fill the feed with the help of the Post Materializer and redis cache

USER 01: [1,2,4,6,7,8,9,....100]

We will not directly inject the like and comments as it becomes a heavy operation considering 500M users


Feed Service : further optimization
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.
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
3. Logging
User Service
JWT for session management
POST DB
Text Data Storage for POST
Write Ops Fast
- blocked_post
-filtered_post
Post against policy
Check the post's from POST DB in prior & pass them to KAFKA queue.
Clients
API Gateway & Load Balancers
- Authentication
-Authorization
- Routing
- Rate Limiting
Content Service
1
- raw_post
-filtered_post
- blocked_post
Images
Videos
For normal person
1
Users_Latest_post
< recent 100 posts pre-computed >
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
CACHE
- Read
Feed Service
Feed Cache
Feed preparation in cache
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 Engagment Service
Consumer
Like
- like_id
post_id / comment_id
- user_id
- reaction_type
- other data

7. 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 tools like Redis or Memcached to reduce database load for frequently accessed data such as feeds and user profiles.
- Store media files (images and videos) using cloud storage and a CDN to improve performance and reduce server load.
- Apply asynchronous processing and message queues like Kafka or RabbitMQ for tasks such as notifications and feed updates.

8. Learning Outcomes

- Understood how a social media application system works.
- Learned how 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 an understanding of core APIs for user management, posts, and feed handling.