WORKSHEET 3

Student Name: Khushi Khemka **UID: 23BCS10652**
Branch: BE-CSE **Section/Group: Krg_3A**
Semester: 6th **Subject Name: System Design**
Subject Code: 23CSH-314

Aim: To design a social media application (Instagram/Facebook)

Steps:

1. **Functional Requirements:**

A. Client should be able to register and login to the application.

B. Client should be able to create post (text / image / videos)

C. Client should be able to follow each other (or send friend requests)

D.Client should be able to like or comment on the post

E. Client should be able to view the feed of post from users they follow.

2. **Non-Functional Requirements:**

A. Scalability: 500M DAU.

B. Consistency & Availability:

In this case -> Social Media

1. We need our application to be HIGHLY available, then consistent
Reason: If it is not operational / Functional when it was required, then there is no
meaning of developing the application.

C. Latency: (Uploading speed of publishing post): 500ms to upload post

3. **API Design:**

A. User On-boarding API's

1. **User Registeration:** POST API CALL: POST / api / users / register_user
2. **User Login: POST API CALL:** POST / api / users / login
3. **User Data Display: GET API CALL:** GET / api / users / {user_id} / profile
4. **User Data Update: PUT API CALL:** PUT / api / users / {user_id} / profile

B. User Post's

1. POST / api / user_id / posts
2. GET / api / posts / {post_id}
3. PUT / api / posts / {post_id}
4. DELETE / api / posts / {post_id}
5. GET / api / posts / feed / limit = {limit} & offset = {offset} : PAGINATION
6. GET / api / users / {user_id} / posts: PAGINATION
   For a particular user if we want to fetch the post's

C. User Interactions

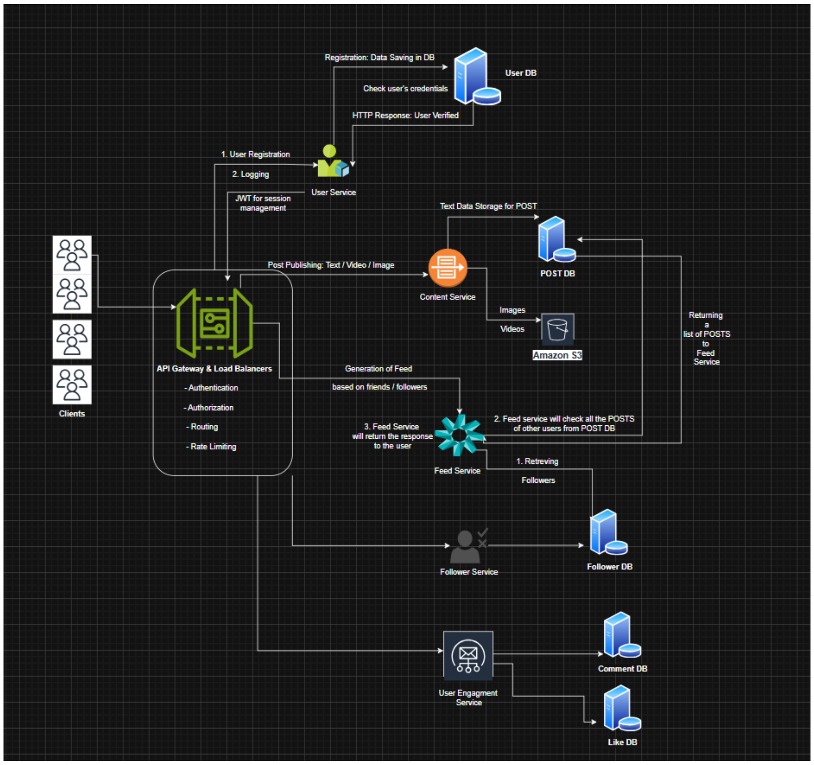
1. POST / api / posts / {post_id} / like
2. DELETE / api / posts / {post_id} / unlike
3. POST / api / posts / {post_id} / comments
4. GET / api / posts / {post_id} / comments
5. PUT / api / comments / {comment_id}
6. DELETE / api / comments /{post_id}/ {comment_id}
7. POST / api / users / {user_id} /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

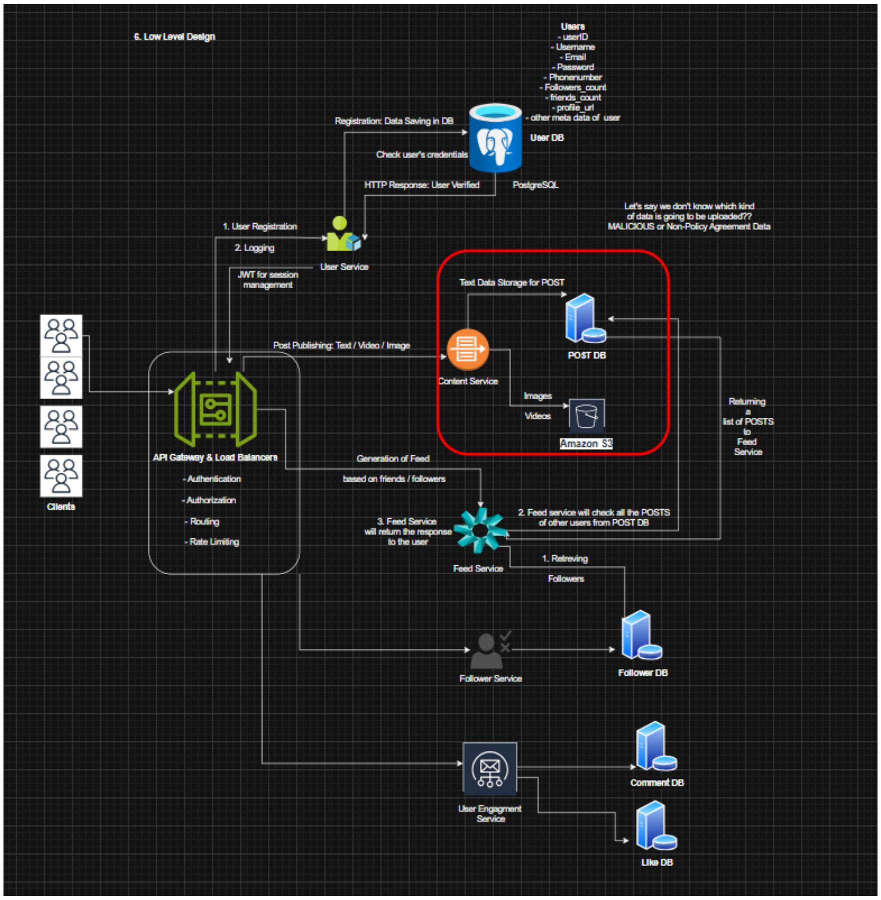
4. High-Level Design:

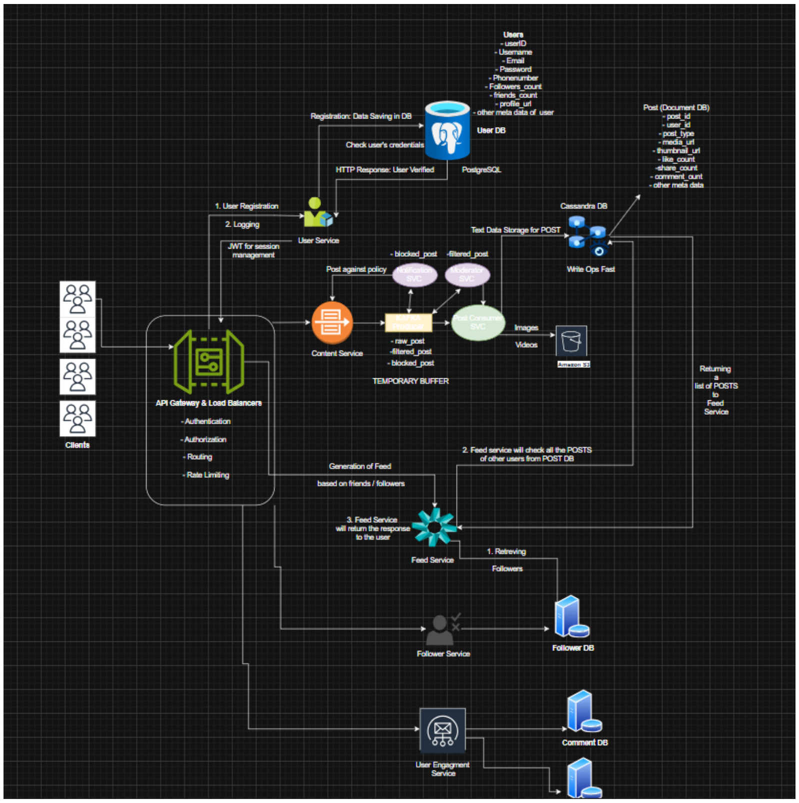
Now According to the functional requirement of the system, we can identify that :

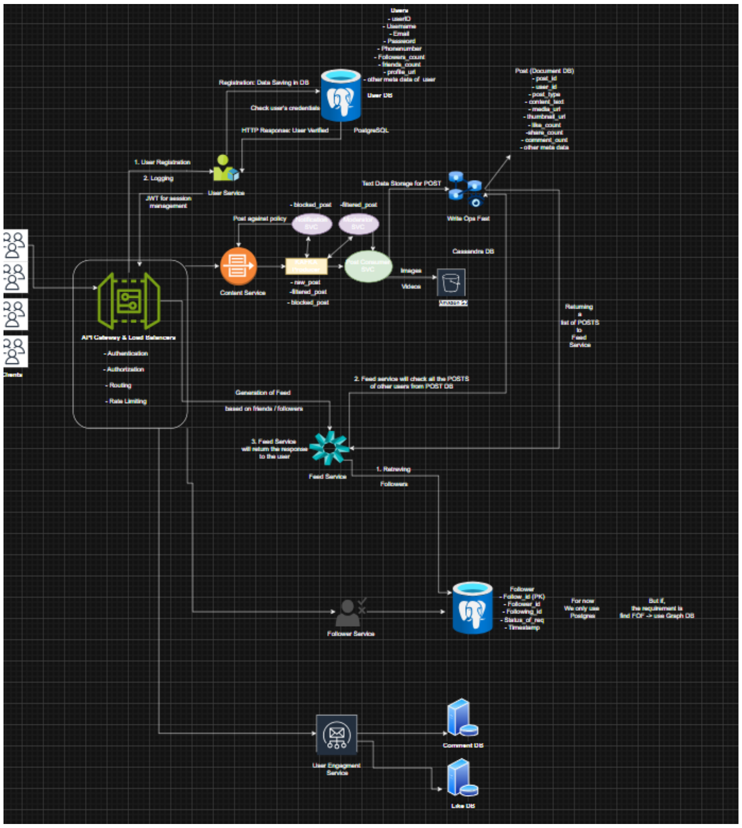
There will be a client who is requesting, then there will a server upon which computation will be going on, and lastly there will be an database in which storage will be done.

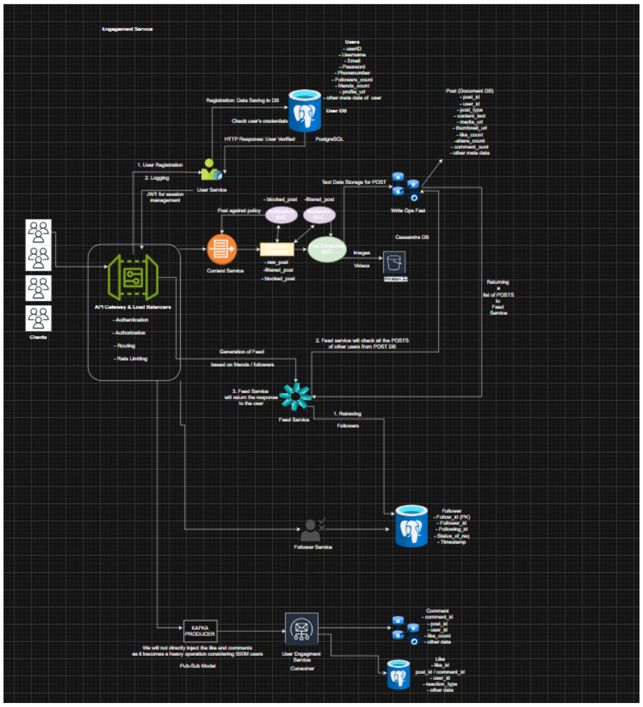
5. Low-Level Design:

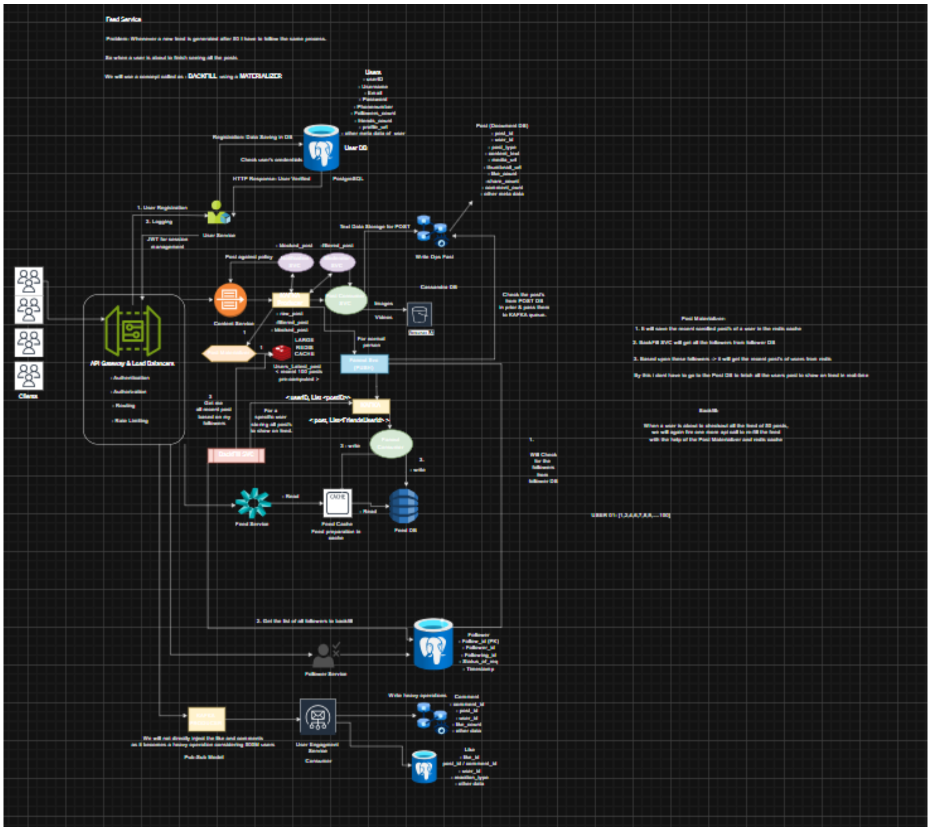

Registration: Data Saving in DB
User DB
Check user's credentials
HTTP Response: User Verified
1. User Registration
2. Logging
User Service
JWT for session management
Text Data Storage for POST
POST DB
Post Publishing: Text / Video / Image
Content Service
Images
Videos
Amazon S3
Returning a list of POSTS 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
User Engagment Service
Comment DB
Like DB


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
User DB
HTTP Response: User Verified
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
Notification
SVC
Moderator
SVC
Content Service
KAFKA
Producer
Post Consumer
SVC
Images
Videos
Amazon S3
- raw_post
-filtered_post
- blocked_post
TEMPORARY BUFFER
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. Retreving
Followers
Follower Service
Follower DB
Comment DB
User Engagment
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
1. User Registration
2. Logging
User Service
JWT for session management
Post (Document DB)
- post_id
- user_id
- post_type
- content_text
- media_url
- thumbnail_url
- like_count
-share_count
- comment_count
- other meta data
Text Data Storage for POST
Write Ops Fast
Cassandra DB
- blocked_post
-filtered_post
Post against policy
Notification SVC
Moderator SVC
KAFKA Producer
Post Consumer SVC
- new_post
-filtered_post
- blocked_post
Content Service
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
- 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
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
User DB
PostgreSQL
Registration: Data Saving in DB
Check user's credentials
HTTP Response: User Verified
1. User Registration
2. Logging
User Service
JWT for session management
Post (Document DB)
- post_id
- user_id
- post_type
- content_text
- media_url
- thumbnail_url
- like_count
-share_count
- comment_count
- other meta data
Text Data Storage for POST
Write Ops Fast
Cassandra DB
- blocked_post
-filtered_post
Post against policy
Notification SVC
Moderator SVC
KAFKA Producer
Post Consumer SVC
- new_post
-filtered_post
- blocked_post
Content Service
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
- 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

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 curated 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 realtime

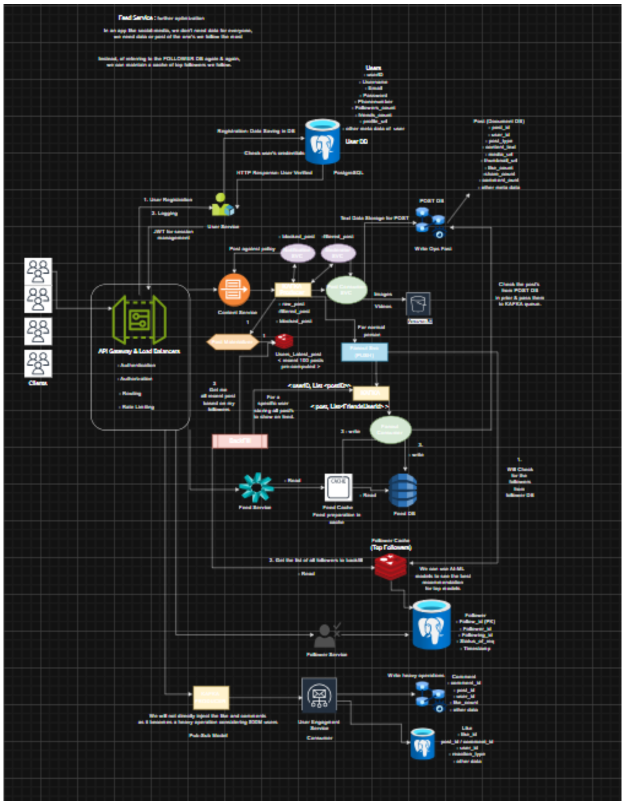
Backfill

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.

Pub-Sub Model



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.

We can use AI-ML models to use the best recommendation for top models.

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

Pub-Sub Model