Experiment 4

Student Name: Gourav Sharma	**UID: 23BCS10857**
Branch: CSE	**Section/Group: KRG 3-A**
Semester: 6th	**Date of Performance:06/02/2026**
Subject Name: System Design	**Subject Code: 23CSH-314**

1. **Aim**: To design and analyze a scalable OTT (Over-The-Top) video streaming platform similar to Netflix / Amazon Prime, which allows users to register, subscribe, search, and stream video content efficiently while ensuring high availability, low latency, and massive scalability using modern distributed system concepts.

2. **Objective**:

   - To design a scalable OTT video streaming platform for efficient content delivery.
   - To provide secure user registration and subscription management.
   - To enable fast search and seamless video streaming with low latency.
   - To ensure high availability and fault tolerance using distributed systems.
   - To support massive scalability and performance optimization for large user bases.

3. **Tools Used:**
   - **Draw.io** – System Architecture & HLD Diagram
   - **MySQL** – User, subscription, and payment data
   - **MongoDB / NoSQL DB** – Video metadata storage
   - **ElasticSearch** – Search functionality for movies and TV shows
   - **Apache Kafka** – Asynchronous event streaming and video pipeline coordination
   - **Redis / CDN Cache** – Caching frequently accessed content
   - **Blob Storage (Amazon S3 equivalent)** – Video and media storage

4. **System Requirements:**

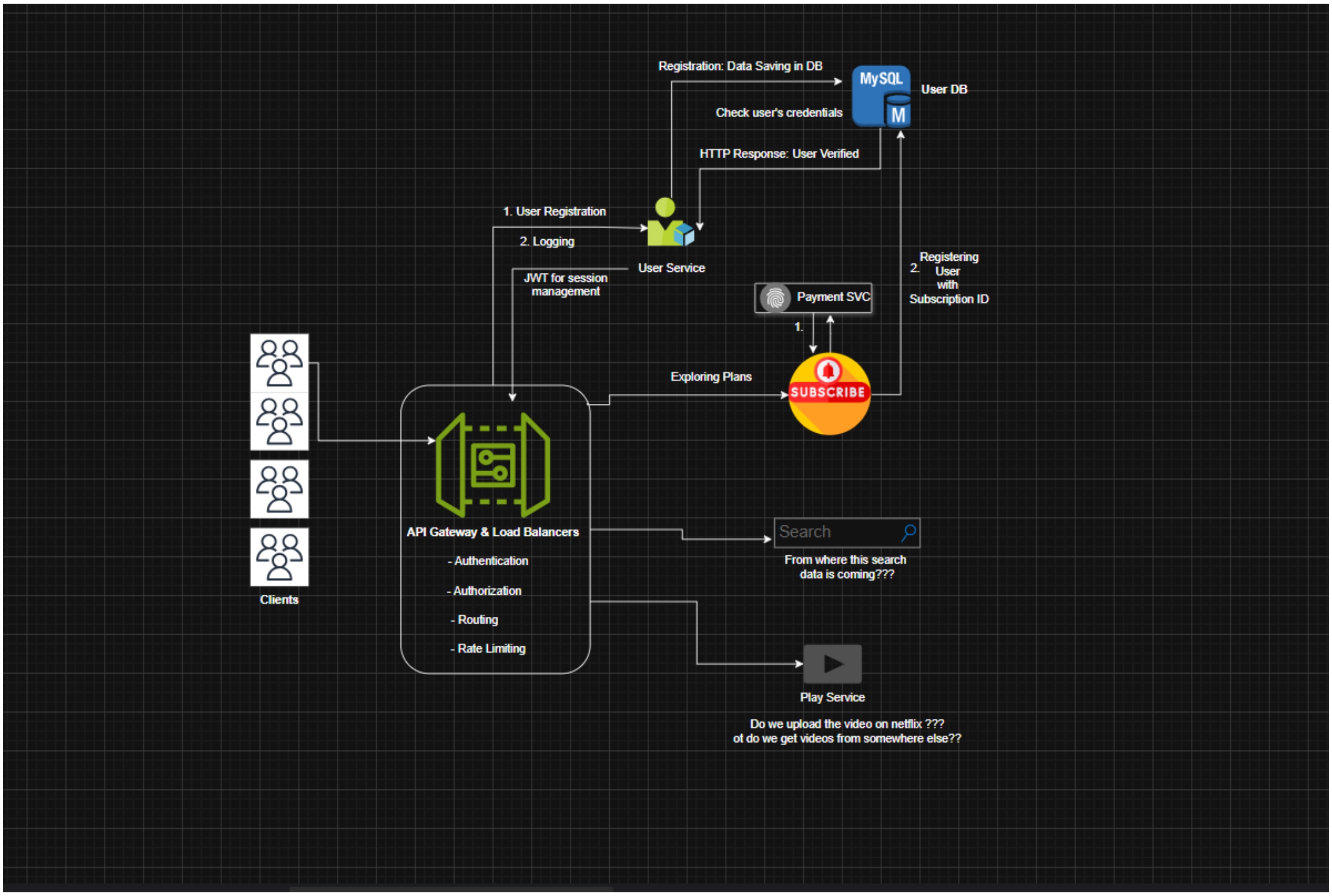
A. Functional Requirements

- The user should be able to create an account on the OTT platform.
- The user should be able to log in and choose an appropriate subscription plan.
- The user should be able to search movies and TV shows using titles or names (Elastic Search).
- The user should be able to stream movies and TV shows in multiple resolutions (480p, 720p, 1080p, 4K).
- The system should provide personalized recommendations for movies and TV shows.

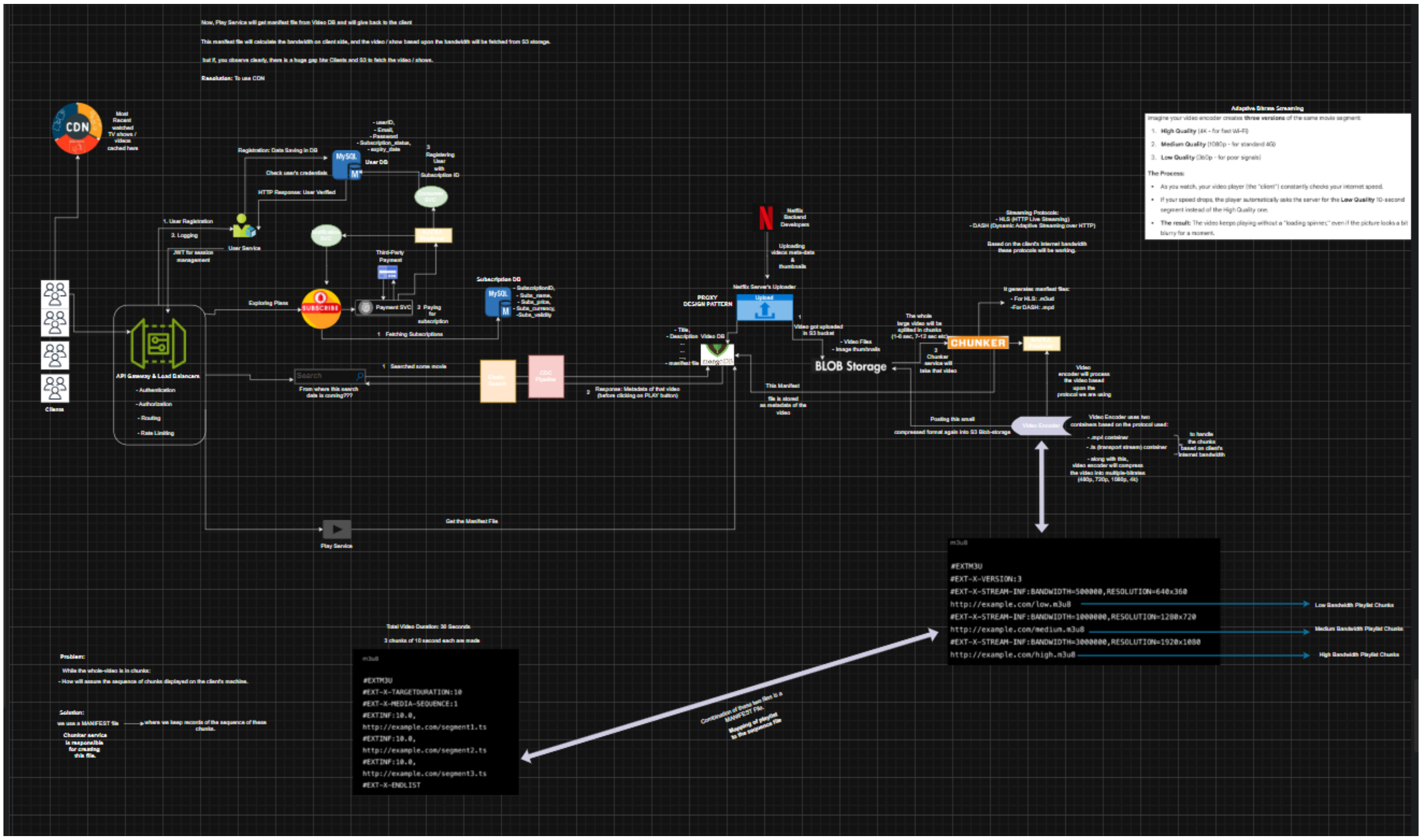
B. Non-Functional Requirements

- **Scalability**
  200-300M, for which let's say total videos we are having are 20K videos (~1 hour each)
- **Availability and Consistency**
  CAP Theorem: Availability >>>>> Consistency
  Availability on watching TV shows and movies
  Consistency in making payments and in subscription plans
- **Latency: 50 - 80 ms**
  Client should be able to see the video with zero or neglibile buffering..

5. High Level Design (HLD):


Registration: Data Saving in DB
MySQL
M
User DB
Check user's credentials
HTTP Response: User Verified
1. User Registration
2. Logging
User Service
JWT for session management
Payment SVC
2. Registering User with Subscription ID
1.
Exploring Plans
SUBSCRIBE
Clients
API Gateway & Load Balancers
- Authentication
- Authorization
- Routing
- Rate Limiting
Search
From where this search data is coming???
Play Service
Do we upload the video on netflix ???
ot do we get videos from somewhere else??

6. Low Level Design (LLD):



Now, Play Service will get manifest file from Video DB and will give back to the client

This manifest file will calculate the bandwidth on client side, and the video / show based upon the bandwidth will be fetched from S3 storage.

but if, you observe clearly, there is a huge gap b/w Clients and S3 to fetch the video / shows.

Resolution: To use CDN

Adaptive Bitrate Streaming

Imagine your video encoder creates **three versions** of the same movie segment:

1. **High Quality** (4K - for fast Wi-Fi)
2. **Medium Quality** (1080p - for standard 4G)
3. **Low Quality** (360p - for poor signals)

The Process:

- As you watch, your video player (the "client") constantly checks your internet speed.
- If your speed drops, the player automatically asks the server for the **Low Quality** 10-second segment instead of the High Quality one.
- **The result:** The video keeps playing without a "loading spinner," even if the picture looks a bit blurry for a moment.

m3u8

```
#EXTM3U
#EXT-X-VERSION:3
#EXT-X-STREAM-INF:BANDWIDTH=500000,RESOLUTION=640x360
http://example.com/low.m3u8
#EXT-X-STREAM-INF:BANDWIDTH=1000000,RESOLUTION=1280x720
http://example.com/medium.m3u8
#EXT-X-STREAM-INF:BANDWIDTH=3000000,RESOLUTION=1920x1080
http://example.com/high.m3u8
```

Total Video Duration: 30 Seconds

3 chunks of 10 second each are made

m3u8

```
#EXTM3U
#EXT-X-TARGETDURATION:10
#EXT-X-MEDIA-SEQUENCE:1
#EXTINF:10.0,
http://example.com/segment1.ts
#EXTINF:10.0,
http://example.com/segment2.ts
#EXTINF:10.0,
http://example.com/segment3.ts
#EXT-X-ENDLIST
```

Problem:

While the whole-video is in chunks:

- How will assure the sequence of chunks displayed on the client's machine.

Solution:

we use a MANIFEST file → where we keep records of the sequence of these chunks.

Chunker service is responsible for creating this file.

CDN
Most Recent watched TV shows / videos cached here
Clients
API Gateway & Load Balancers
- Authentication
- Authorization
- Routing
- Rate Limiting
1. User Registration
2. Logging
JWT for session management
User Service
Registration: Data Saving in DB
Check user's credentials
HTTP Response: User Verified
MySQL
User DB
- userID,
- Email,
- Password
- Subscription_status,
- expiry_date
3 Registering User with Subscription ID
Consumer SVC
Notification SVC
KAFKA Producer
Third-Party Payment
Exploring Plans
SUBSCRIBE
Payment SVC
2 Paying for subscription
1 Fetching Subscriptions
Subscription DB
- SubscriptionID,
- Subs_name,
- Subs_price,
- Subs_currency,
-Subs_validity
Search
From where this search data is coming???
1 Searched some movie
Elastic Search
CDC Pipeline
2 Response: Metadata of that video (before clicking on PLAY button)
Netflix Backend Developers
Uploading videos meta-data & thumbnails
Netflix Server's Uploader
Upload
PROXY DESIGN PATTERN
- Title,
- Description
Video DB
- manifest file
mongoDB
1 Video got uploaded in S3 bucket
- Video Files
- Image thumbnails
BLOB Storage
This Manifest file is stored as metadata of the video
The whole large video will be splited in chunks (1-6 sec, 7-12 sec etc)
CHUNKER
2 Chunker service will take that video
KAFKA Producer
Streaming Protocols:
- HLS (HTTP Live Streaming)
- DASH (Dynamic Adaptive Streaming over HTTP)
Based on the client's internet bandwidth these protocols will be working.
It generates manifest files:
- For HLS: .m3u8
-For DASH: .mpd
Video encoder will process the video based upon the protocol we are using
Posting this small compressed format again into S3 Blob-storage
Video Encoder
Video Encoder uses two containers based on the protocol used:
- .mp4 container
- .ts (transport stream) container
to handle the chunks based on client's internet bandwidth
- along with this, video encoder will compress the video into multiple-bitrates (480p, 720p, 1080p, 4k)
Low Bandwidth Playlist Chunks
Medium Bandwidth Playlist Chunks
High Bandwidth Playlist Chunks
Combination of these two files is a MANIFEST File.
Mapping of playlist to the sequence file
Play Service
Get the Manifest File

7. Scalability Solution

- Horizontal scaling is used to add multiple service instances to handle increasing user traffic.
- Load balancers are implemented to distribute requests evenly across backend servers.
- Content Delivery Networks (CDNs) are used to cache videos closer to users for faster streaming.
- Microservices architecture enables independent scaling of critical services like streaming and search.
- Auto-scaling mechanisms dynamically adjust resources based on real-time demand.

8. Learning Outcomes (What I Have Learnt)

- Understand the architecture of a large-scale OTT video streaming platform.
- Learn how distributed systems ensure scalability, availability, and fault tolerance.
- Gain knowledge of adaptive video streaming and CDN-based content delivery.
- Analyze real-world trade-offs using CAP theorem in system design.
- Develop skills to design and evaluate scalable, high-performance web applications.