**WORKSHEET 4**

**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 video streaming application**
**Steps:**

1. **Functional Requirements:**
   1.Client should be able to create account on the OTT platform.
   2. After the successfull login, client should be able to opt for the subscription plans.
   3. Client should be able to search for the shows/movies based on the video title or names.
   4. Client should be able to watch the videos / tv shows in multiple different resolutions (480p,720p,1080p, 4k etc.)
   5. Recommendation for TV shows and movies.

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

3. **API Design:**
   **A. User On-boarding API's**
   1. **POST Call: https://www.netlfix.com/user/register**
   2. **POST Call:** https://www.netlfix.com/user/login
   3. **PUT Call**: https://www.netlfix.com/user/update

**Subscription:**

1. GET Call: https://www.netlfix.com/search?q={movie_name}
   Response: List<Video_ID> + some meta data of video -> Pagination

2. GET Call: https://www.netlfix.com/{video_ID}
   Response: Metadata of the video (JSON)

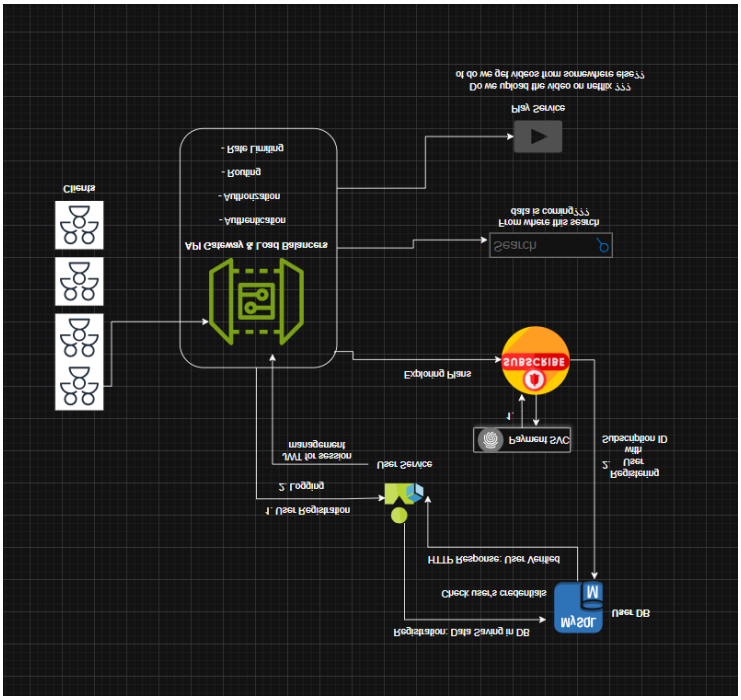
3.GET Call: https://www.netlfix.com/play/{video_ID}

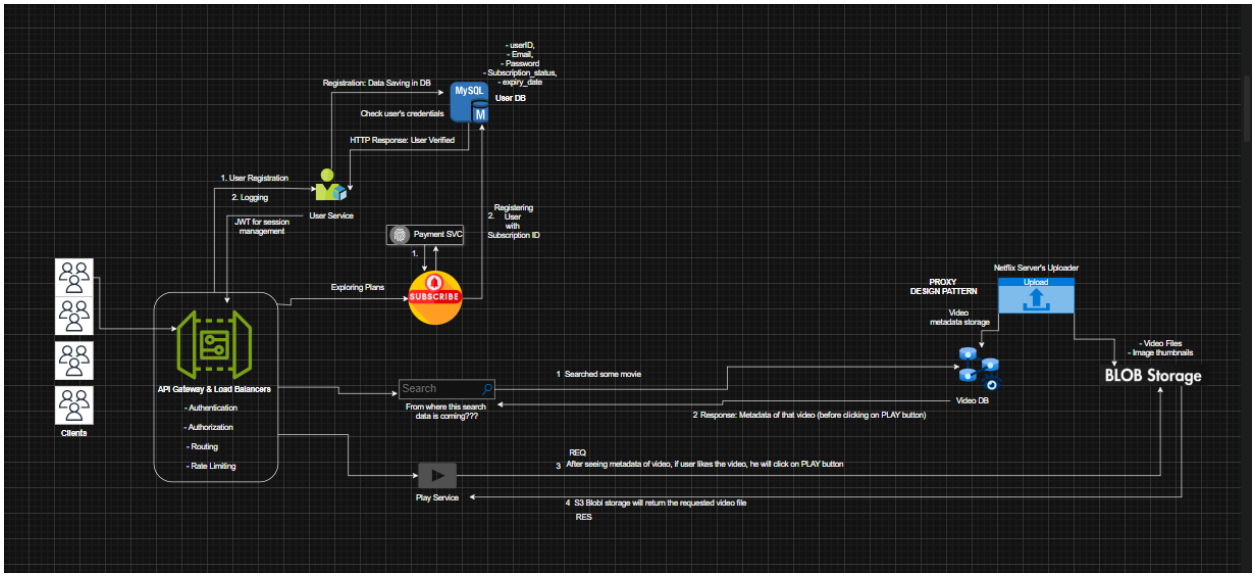
## 4. High-Level Design:

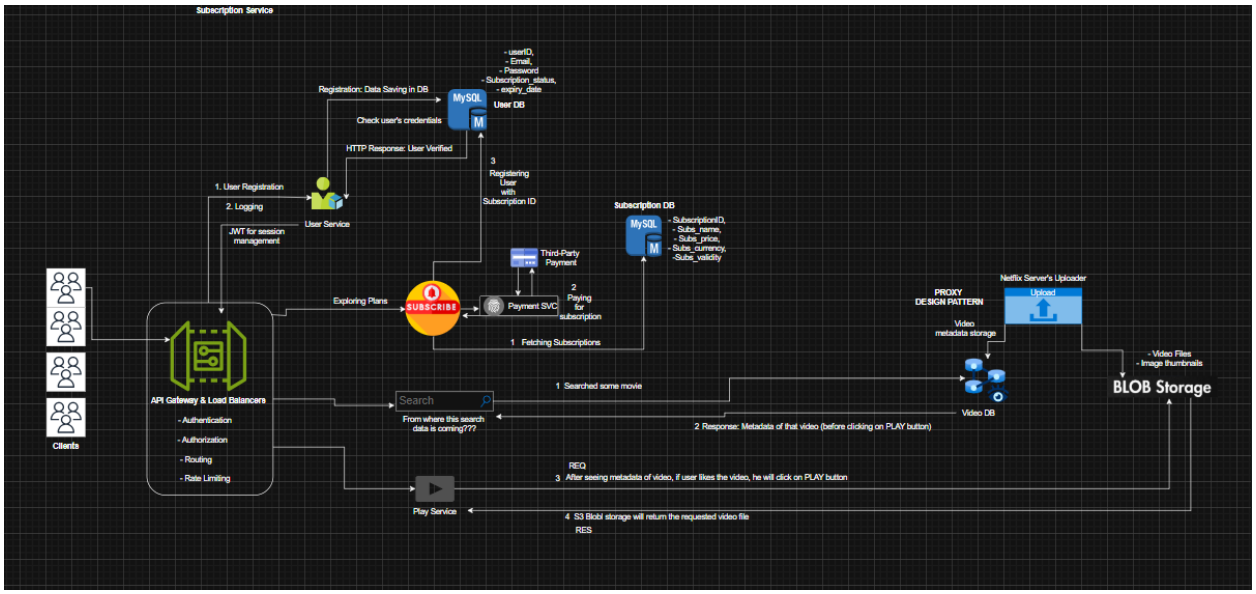
Now According to the functional requirement of the system, we can identify that :

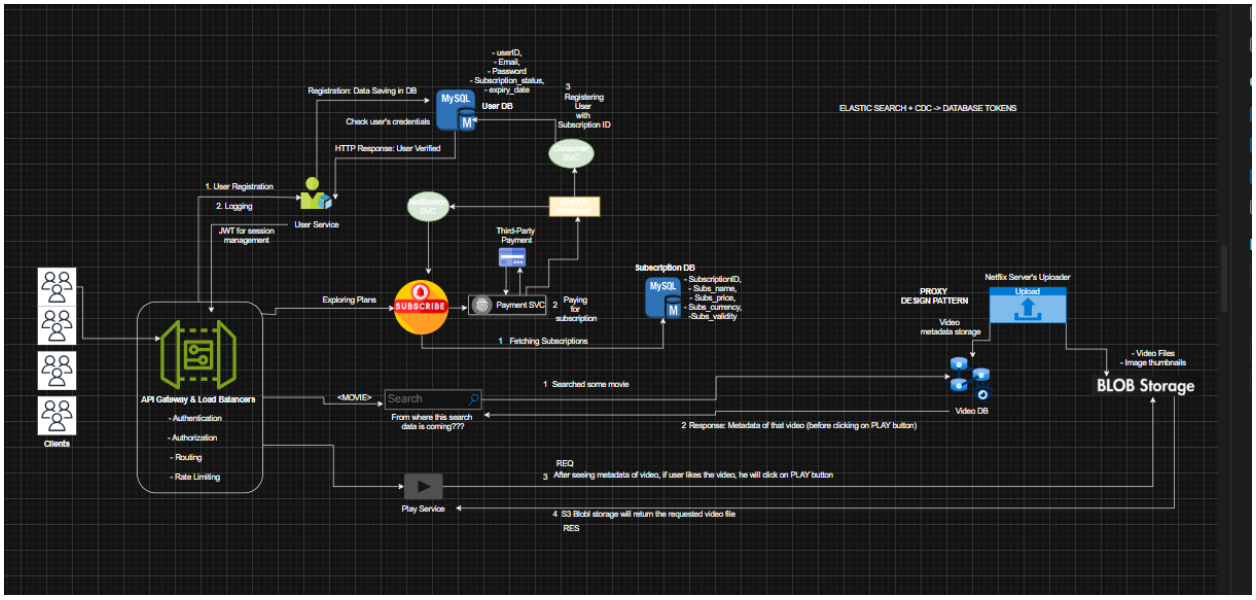
There will be a client who is requesting, then there will a server upon which computation will be going on, and lastly there will be an database in which storage will be done.

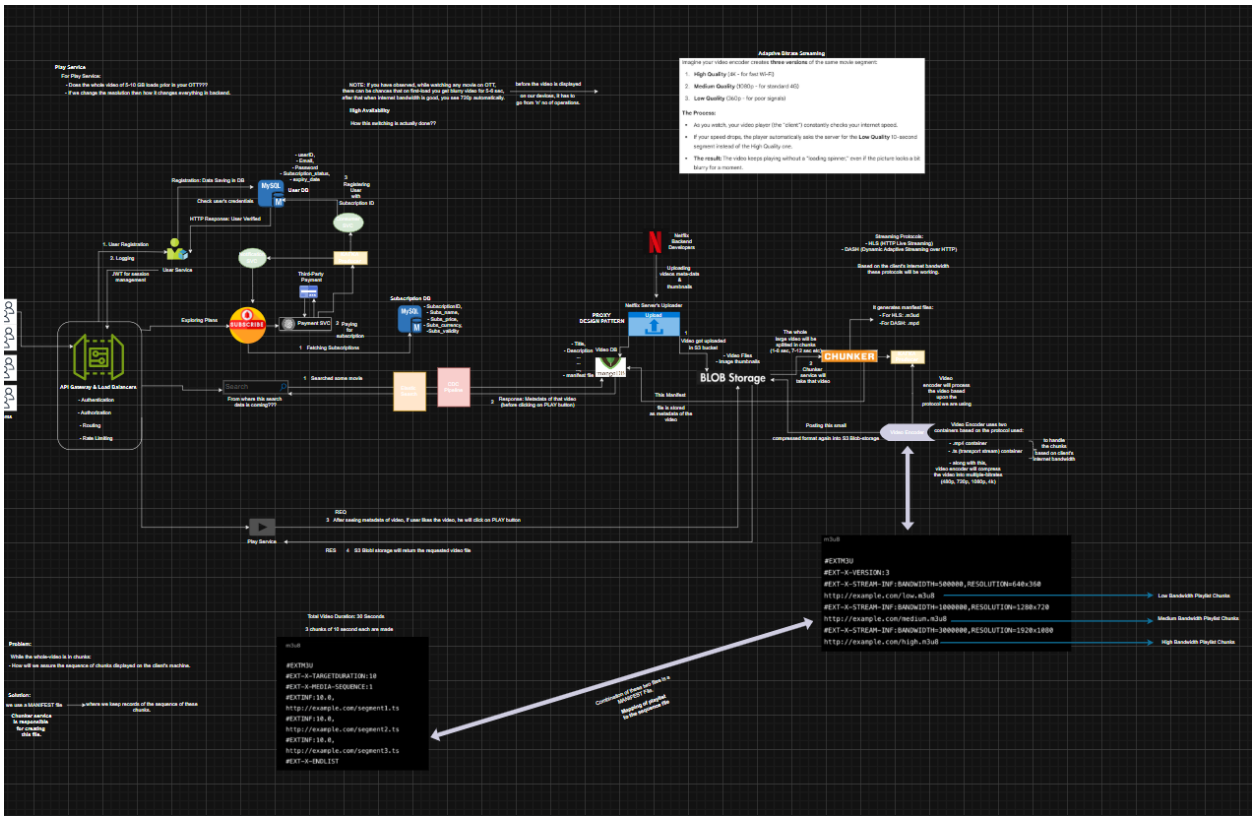
## 5. Low-Level Design:

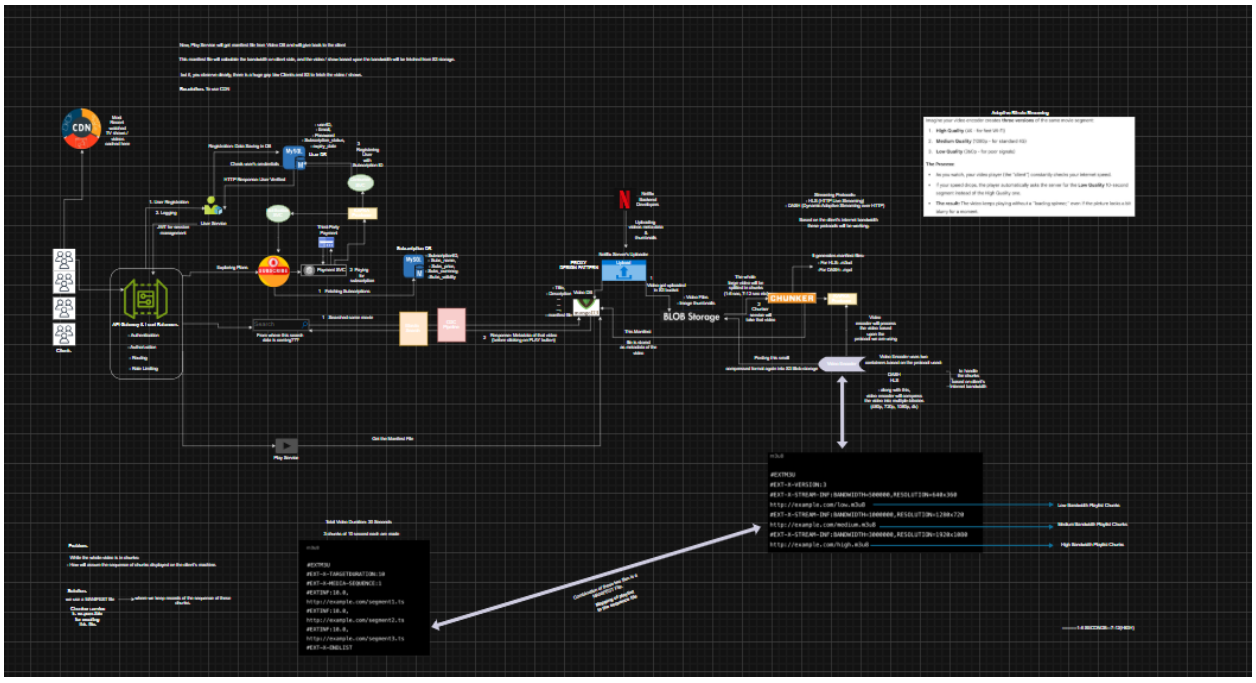



- userID,
- Email,
- Password
- Subscription_status,
- expiry_date
Registration: Data Saving in DB
MySQL
User DB
Check user's credentials
HTTP Response: User Verified
1. User Registration
2. Logging
User Service
JWT for session management
Payment SVC
1.
2. Registering User with Subscription ID
Exploring Plans
SUBSCRIBE
Clients
API Gateway & Load Balancers
- Authentication
- Authorization
- Routing
- Rate Limiting
Search
From where this search data is coming???
1 Searched some movie
2 Response: Metadata of that video (before clicking on PLAY button)
PROXY DESIGN PATTERN
Netflix Server's Uploader
Upload
Video metadata storage
Video DB
- Video Files
- Image thumbnails
BLOB Storage
REQ
3 After seeing metadata of video, if user likes the video, he will click on PLAY button
Play Service
4 S3 Blobl storage will return the requested video file
RES


Subscription Service
- userID,
- Email,
- Password
- Subscription_status,
- expiry_date
Registration: Data Saving in DB
MySQL
User DB
Check user's credentials
HTTP Response: User Verified
3 Registering User with Subscription ID
1. User Registration
2. Logging
User Service
JWT for session management
Subscription DB
- SubscriptionID,
- Subs_name,
- Subs_price,
- Subs_currency,
- Subs_validity
Third-Party Payment
Payment SVC
2 Paying for subscription
Exploring Plans
SUBSCRIBE
1 Fetching Subscriptions
Clients
API Gateway & Load Balancers
- Authentication
- Authorization
- Routing
- Rate Limiting
Search
From where this search data is coming???
1 Searched some movie
2 Response: Metadata of that video (before clicking on PLAY button)
PROXY DESIGN PATTERN
Netflix Server's Uploader
Upload
Video metadata storage
Video DB
- Video Files
- Image thumbnails
BLOB Storage
REQ
3 After seeing metadata of video, if user likes the video, he will click on PLAY button
Play Service
4 S3 Blobl storage will return the requested video file
RES


- userID,
- Email,
- Password
- Subscription_status,
- expiry_date
User DB
Registration: Data Saving in DB
Check user's credentials
HTTP Response: User Verified
3 Registering User with Subscription ID
ELASTIC SEARCH + CDC -> DATABASE TOKENS
1. User Registration
2. Logging
User Service
JWT for session management
Third-Party Payment
Subscription DB
- SubscriptionID,
- Subs_name,
- Subs_price,
- Subs_currency,
- Subs_validity
Exploring Plans
SUBSCRIBE
Payment SVC
2 Paying for subscription
1 Fetching Subscriptions
PROXY DESIGN PATTERN
Netflix Server's Uploader
Upload
Video metadata storage
Video DB
- Video Files
- Image thumbnails
BLOB Storage
Clients
API Gateway & Load Balancers
- Authentication
- Authorization
- Routing
- Rate Limiting
<MOVIE>
Search
From where this search data is coming???
1 Searched some movie
2 Response: Metadata of that video (before clicking on PLAY button)
REQ
3 After seeing metadata of video, if user likes the video, he will click on PLAY button
Play Service
4 S3 Blobl storage will return the requested video file
RES


Play Service
For Play Service:
- Does the whole video of 5-10 GB loads prior in your OTT???
- If we change the resolution then how it changes everything in backend.
NOTE: If you have observed, while watching any movie on OTT, there can be chances that on first-load you get blurry video for 5-6 sec, after that when internet bandwidth is good, you see 720p automatically.
before the video is displayed on our device, it has to go from 'n' no of operations.
High Availability
How this switching is actually done??
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.
Netflix Backend Developers
Uploading videos meta-data & thumbnails
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 get uploaded in S3 bucket
The whole large video will be splitted in chunks (1-2 sec, 7-12 sec etc.)
CHUNKER
Chunker service will take that video
Video encoder will process the video based upon the protocol we are using.
Video Encoder uses two containers based on the protocol used:
- .mp4 container
- .ts (transport stream) container
- along with this, video encoder will compress the video into multiple videos (480p, 720p, 1080p, 4k)
to handle the chunks based on client's internet bandwidth
Posting this small compressed format again into S3 Blobl-storage
This Manifest file is stored as metadata of the video
Response: Metadata of that video (before clicking on PLAY button)
REQ
3 After seeing metadata of video, if user likes the video, he will click on PLAY button
RES
4 S3 Blobl storage will return the requested video file
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
Low Bandwidth Playlist Chunks
Medium Bandwidth Playlist Chunks
High Bandwidth Playlist Chunks
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
Combination of every link there is MANIFEST file
Mapping of chunks to the respective file
Problem:
While the whole-video is in chunks:
- How will we 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
BLOB Storage
CHUNKER
#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
#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