Experiment 4

Name: Abhishek Sharma
UID: 23BCS12330
Branch: CSE
Section/Group: KRG 3-A
Semester: 6th
Date of Performance: 05/02/2026
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 identify **functional and non-functional requirements** of an OTT streaming system
- To design a **high-level architecture (HLD)** for a large-scale video streaming platform
- To understand **video streaming protocols (HLS / DASH)** and adaptive bitrate streaming
- To apply **CAP theorem trade-offs** in OTT systems
- To understand the role of **Kafka, CDN, object storage, chunking, and encoding pipelines**
- To study how **availability is prioritized over consistency** in media streaming systems

3. Tools:

- **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:-

1. Client should be able to create an account on the OTT platform
2. Client should be able to login and manage sessions securely
3. Client should be able to subscribe to different plans after successful login
4. Client should be able to search movies / TV shows using title or keywords (ElasticSearch)
5. Client should be able to watch videos in multiple resolutions (480p, 720p, 1080p, 4K)

6. System should support adaptive streaming based on internet bandwidth
7. Client should be able to view thumbnails, descriptions, and metadata of videos
8. (Optional) System may support recommendations based on viewing history

B. Non-Functional Requirements:-

1. **Scalability**

- Target Users: 200–300 Million active users
- Total Videos: ~20,000 videos, average duration ~1 hour
- System must handle millions of concurrent streams

2. **Availability vs Consistency (CAP Theorem)**

- Availability >>> Consistency

Reason:

- Video streaming must not stop even during partial failures
- Slight delays in metadata updates are acceptable
- Subscription and payment data require stronger consistency

Example:

- Video playback unavailable → Critical failure
- Slight delay in subscription update → Acceptable

3. **Latency**

- Target latency: 50–80 ms
- Video should play with zero or negligible buffering

5. High Level Design (HLD):

The system follows a microservices-based architecture:

API Gateway

- Handles authentication, authorization, routing, and rate limiting

User Service

- User registration, login, JWT-based authentication
- Stores user data in MySQL

Subscription & Payment Service

- Manages subscription plans and payments
- Ensures strong consistency for transactions

Search Service

- Uses ElasticSearch for fast movie/TV show search

Video Metadata Service

- Stores metadata (title, description, thumbnails) in NoSQL DB

Video Upload & Processing Pipeline

- Uploaded videos are stored in Blob Storage
- Chunker Service splits videos into small segments
- Video Encoder generates multiple resolutions (480p, 720p, 1080p, 4K)
- Kafka coordinates asynchronous processing

Streaming Service

- Generates HLS (.m3u8) or DASH (.mpd) manifests
- Client fetches chunks based on bandwidth

CDN

- Caches most frequently watched video chunks
- Reduces latency and server load

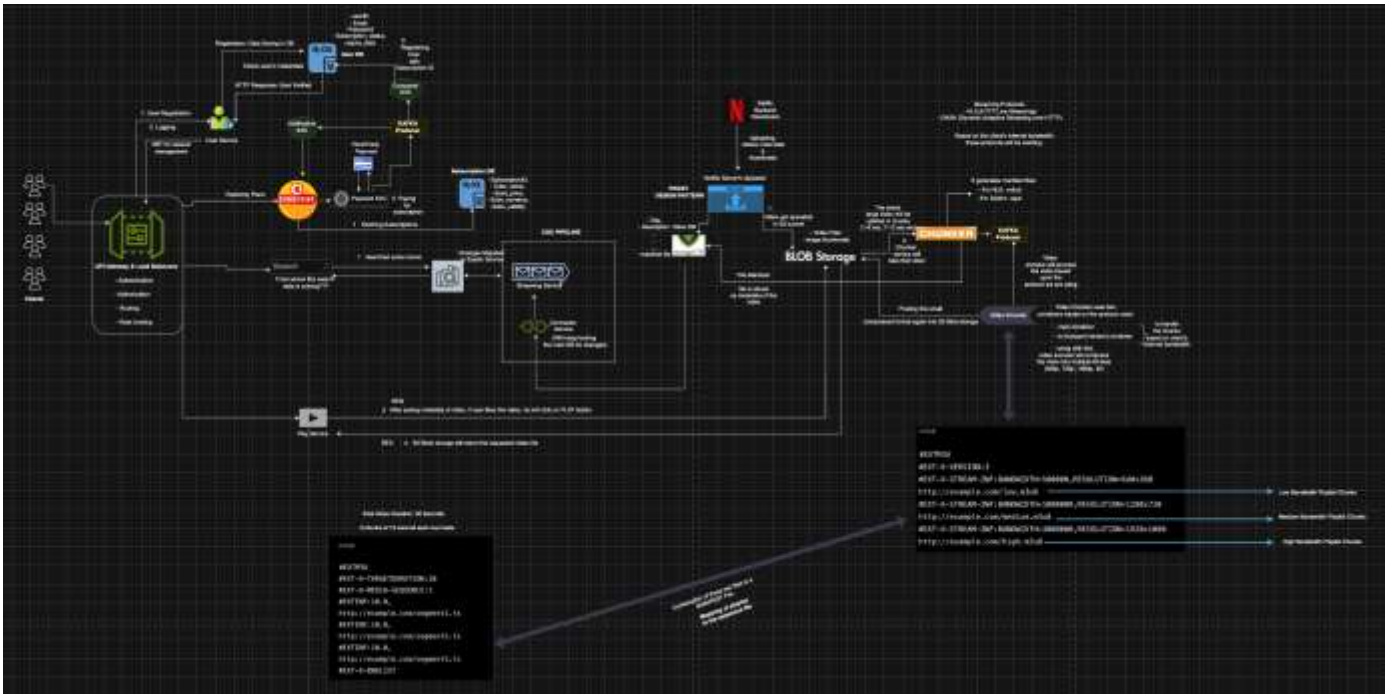
6. **Video Streaming Workflow**:

- User clicks Play Video

- Client requests manifest file (.m3u8 / .mpd)
- Manifest contains URLs of video chunks with different bitrates
- Client measures network bandwidth
- Appropriate resolution chunks are fetched dynamically
- CDN serves cached chunks if available
- Video playback continues smoothly with adaptive bitrate

7. **Scalability Solution:**

- Horizontal scaling of all microservices
- API Gateway acts as load balancer
- Kafka decouples video processing pipeline
- CDN caching minimizes latency and origin load
- Blob storage efficiently stores large video files
- Stateless services allow easy scaling



7. Learning Outcomes (What I Have Learnt):

- Understood how real-world OTT platforms like Netflix are designed
- Learned difference between functional and non-functional requirements
- Gained strong understanding of adaptive bitrate streaming (HLS/DASH)
- Understood CAP theorem trade-offs in streaming systems
- Learned how Kafka, CDN, and chunking pipelines work together
- Learned how to design highly available, low-latency distributed systems