EXPERIMENT NO: 6

Student Name: Kanaboina Yashwanth
UID 23BCS10501
Branch: BE-CSE
Section/Group: KRG-1A
Semester: 6th
Date of Performance: 26/02/2026
Subject Name: System Design
Subject Code: 23CSH-314

Aim

To design a highly scalable video streaming platform similar to YouTube that supports heavy video uploading, adaptive bitrate streaming, and real-time social interactions with high availability and low latency.

Objectives

- Understand the architecture of a massive-scale media streaming system.
- Identify functional requirements such as direct-to-cloud uploads and personalized feeds.
- Identify non-functional requirements including CDN caching, high throughput, and latency constraints.
- Analyze the decoupling of synchronous API requests using event-driven stream processing.
- Design REST APIs for video management, search, and user interactions.

Procedure

1. Studied real-world video streaming architectures and content delivery networks.
2. Identified core entities such as Users, Videos, Interactions, and View Histories.
3. Analyzed the Pre-Signed URL pattern to bypass API Gateway bottlenecks for large file uploads.
4. Collected functional and non-functional requirements for media delivery.
5. Designed APIs for video processing, metadata retrieval, and social engagement.
6. Evaluated streaming protocols (HLS/DASH) and separated thumbnail storage for optimization.
7. Analyzed stream processing (Apache Flink/Kafka) for real-time, high-volume view aggregation.

Functional Requirements

1. System should allow users to register, log in, and manage accounts.
2. System should support uploading large raw video files efficiently.
3. System should stream video smoothly using Adaptive Bitrate Streaming based on network conditions.
4. System should support real-time social interactions (likes, dislikes, comments).
5. System should allow users to search for videos by title or metadata.

6. System should generate a personalized home page feed based on watch history.

Core Entities of the System

- Users
- Videos
- Interactions (Likes/Dislikes)
- Comments
- ViewHistory

API Design

1. Authentication API

A. **Register**: POST /api/v1/auth/register
B. **Login**: POST /api/v1/auth/login

2. Video & Feed API

A. **Initialize Upload**: POST /api/v1/videos/upload/init (Returns S3 Pre-Signed URL)
B. **Get Video Metadata**: GET /api/v1/videos/{video_id}
C. **Get Personalized Feed**: GET /api/v1/feed (Pagination Supported)
D. **Search Videos**: GET /api/v1/search?q={query}

3. Interaction API

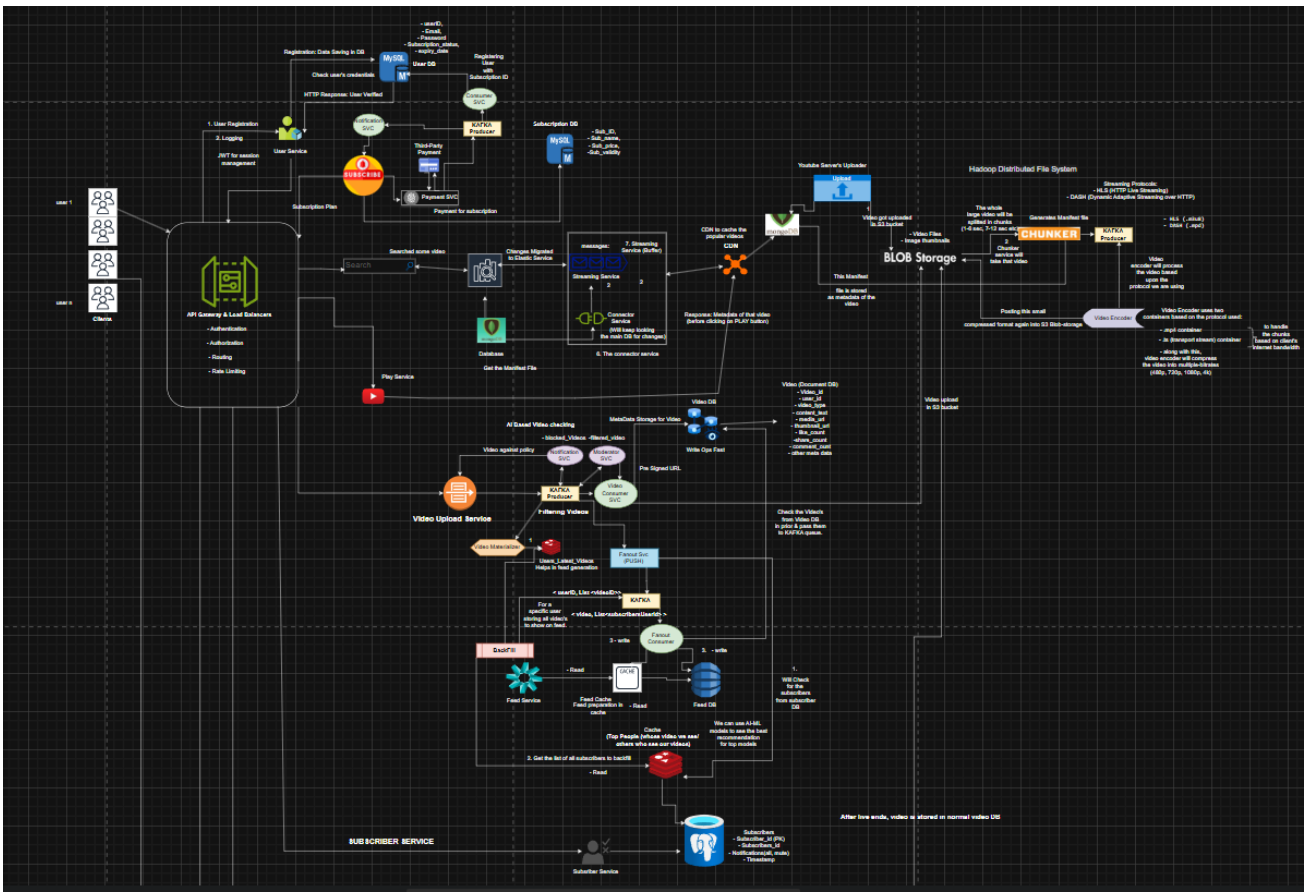
A. **Like Video**: POST /api/v1/videos/{video_id}/like
B. **Add Comment**: POST /api/v1/videos/{video_id}/comments
C. **Get Comments**: GET /api/v1/videos/{video_id}/comments

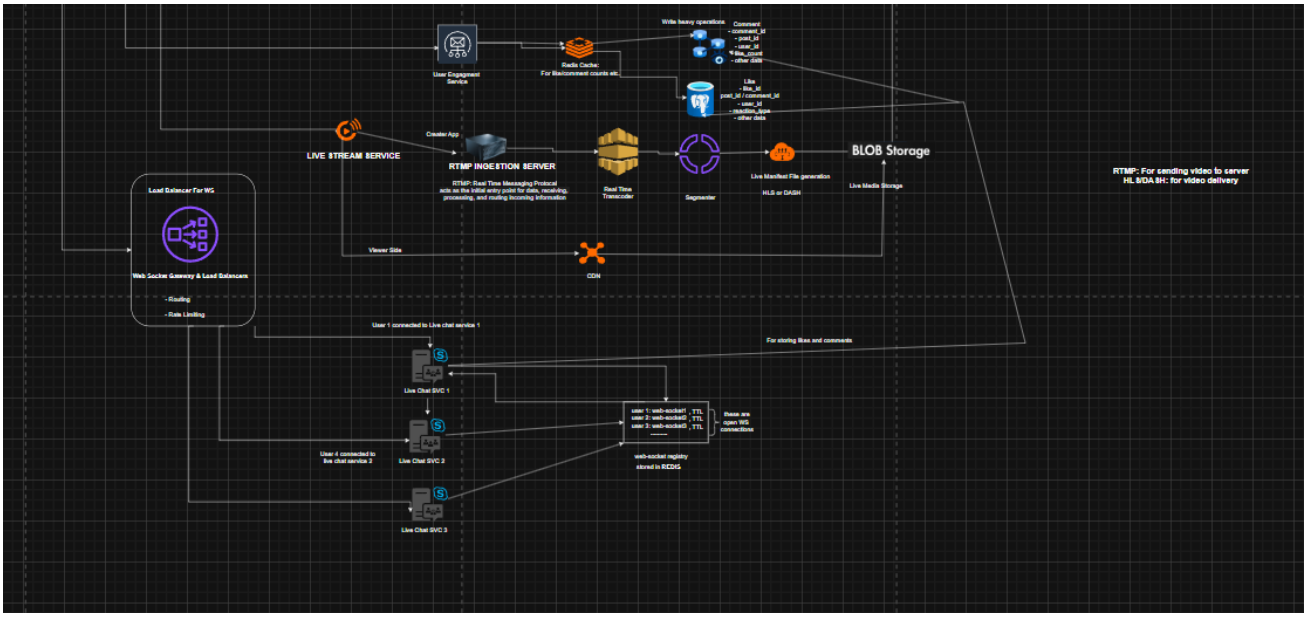
Non-Functional Requirements

1. System must handle extremely high throughput for concurrent uploads and millions of simultaneous views.
2. High Availability must be prioritized, ensuring no single point of failure.
3. System should follow an Eventual Consistency model for social metrics (view counts, likes) to optimize database write loads.
4. Video delivery latency should be minimized via Edge Caching and a Global CDN.
5. Transcoding pipeline must be horizontally scalable to handle viral upload spikes.

High Level Design (HLD)

The system consists of Client Applications, an API Gateway, and specialized microservices (Auth, Upload, Video Metadata, Feed, and Interactions). Heavy uploads bypass the backend via Pre-Signed URLs directly to Object Storage. Asynchronous transcoding pipelines process videos into HLS/DASH segments, which are delivered globally via a CDN. High-volume interactions are managed using Redis (Likes), NoSQL clusters (Comments), and a Kafka-to-Apache Flink stream processing pipeline for accurate view aggregation.


BLOB Storage
Hadoop Distributed File System
CHUNKER
Video Upload Service
BUBSCRIBER SERVICE


LIVE STREAM SERVICE
RTMP INGESTION SERVER
BLOB Storage
Web Socket Gateway & Load Balancer
RTMP: For sending video to server
HLS/DASH: for video delivery

Outcome

- Designed a highly scalable, event-driven video streaming architecture.
- Identified critical functional and non-functional requirements for media delivery.
- Designed REST APIs and decoupled asynchronous processing pipelines.
- Understood the application of CDNs, adaptive bitrate streaming, and stream processing for massive-scale systems.