Workshop 2

Juan Sebastian Colorado Caro, Walter Alejandro Suarez Fonseca Department of Computer Engineering
Universidad Distrital Francisco José de Caldas
Bogotá, Colombia
Email: jscoloradoc@udistrital.edu.co, wasuarezf@udistrital.edu.co
Click here to Git Repository

Abstract

The explosive growth of short-form video platforms like TikTok has highlighted the importance of robust, scalable, and efficient data architectures to support high-throughput video ingestion, real-time user interaction, and personalized content recommendation. This project proposes the database system design for a scalable short-form video platform inspired by TikTok, focusing on the integration of both relational and NoSQL technologies to balance transactional integrity with high-performance data retrieval. The system architecture includes a hybrid storage model, ETL pipelines for analytics, and query strategies for user engagement tracking and content recommendations. Initial queries demonstrate the system's capability to support user stories and business needs, such as retrieving user activity logs, trending content, and personalized video feeds.

Index Terms

data, analytics, databases, big data, SQL, NoSQL

I. Introduction

Inspired by TikTok's success, this project focuses on the design of a scalable data system architecture tailored for such a platform. The proposed system incorporates a hybrid data storage approach using both relational databases—for managing structured entities like users and video metadata—and NoSQL solutions—for handling unstructured or semi-structured data such as comments, likes, and video interaction metrics. The architecture supports real-time access, data-driven recommendations, and long-term analytics. This document outlines the architectural components, the type of information the system must handle, and key queries that illustrate the retrieval and processing of essential data to meet both business requirements and user expectations. Additionally, improvements from previous workshop iterations are integrated to refine the data model and system capabilities.

II. Data System Architecture

A. High-level architecture diagram

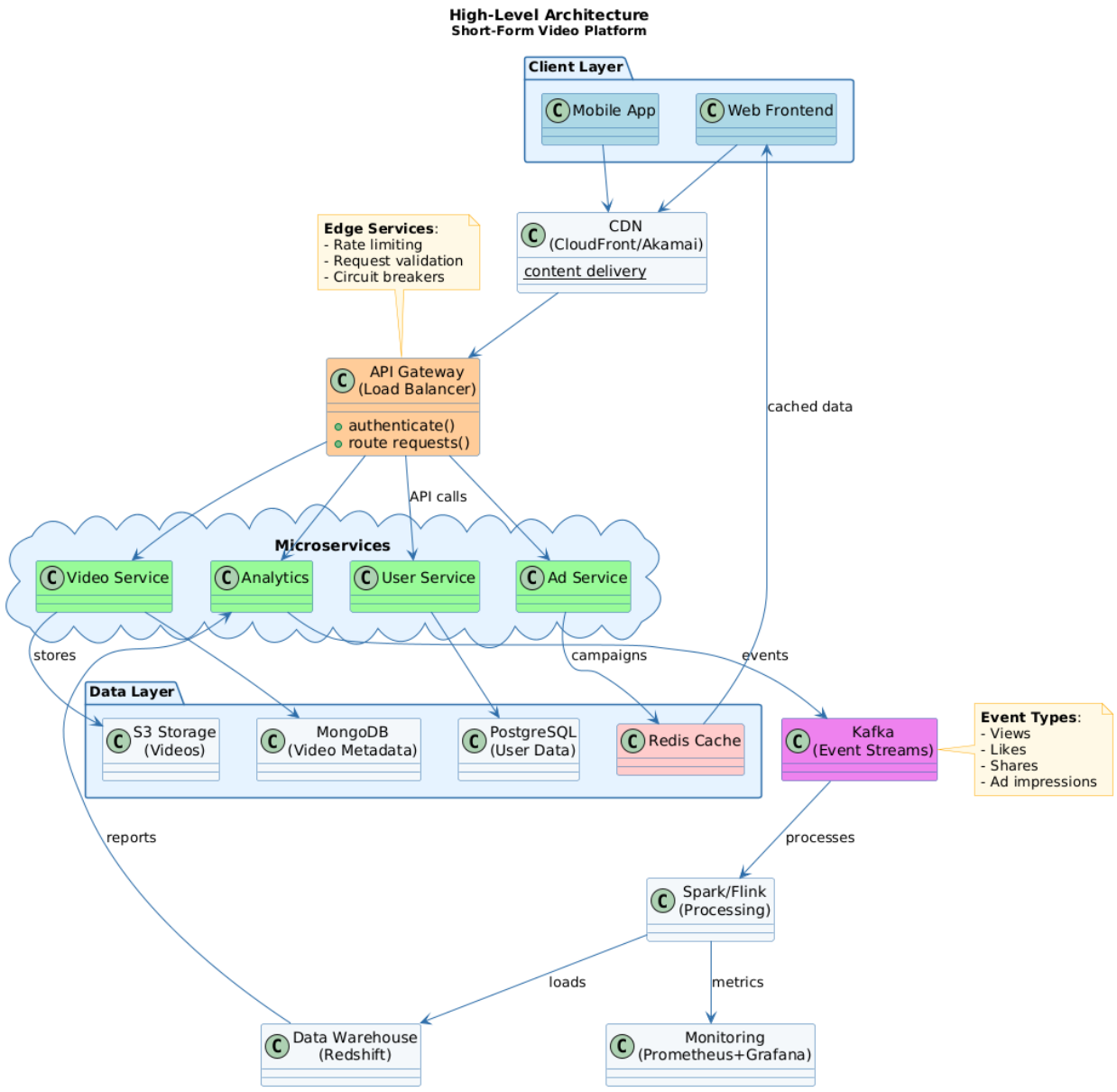

Fig. 1. High-level architecture diagram

III. Technology Stack

This section summarizes the core technologies used in the platform architecture, focusing on the role of each component, the selected technology, and the rationale behind the choice.

1) **Frontend**

Role: User interface for viewers, creators, advertisers, and admins.

Technology: React (Web), Flutter (Mobile)
Why: Cross-platform development with rich UI support and strong community.

2) **API Gateway**
Role: Routes incoming requests and handles authentication, rate-limiting, and CORS.
Technology: AWS API Gateway / NGINX
Why: Provides centralized access control, scalability, and integration with microservices.

3) **Microservices Layer**
Role: Encapsulates domain-specific logic (users, videos, ads, payments, moderation).
Technology: Node.js / Spring Boot
Why: Supports modular development, independent scaling, and CI/CD pipelines.

4) **Containerization and Orchestration**
Role: Packages and manages microservices.
Technology: Docker + Kubernetes
Why: Ensures portability, auto-scaling, and high availability in production.

5) **Ingestion Layer**
Role: Ingests real-time interaction and video events.
Technology: Apache Kafka / Amazon Kinesis
Why: High-throughput, durable, and fault-tolerant streaming platform.

6) **Relational Database**
Role: Stores structured transactional data (users, payments, authentication).
Technology: PostgreSQL
Why: ACID-compliant with strong consistency and robust SQL capabilities.

7) **NoSQL Database**
Role: Manages semi-structured data like video metadata and interactions.
Technology: MongoDB Atlas
Why: Provides flexible schema design and horizontal scalability.

8) **Caching Layer**
Role: Stores trending content and session data for quick access.
Technology: Redis / Memcached
Why: Ultra-fast in-memory data access reduces latency and database load.

9) **Processing and Analytics**
Role: Performs real-time analytics on user engagement and interactions.
Technology: Apache Spark Streaming
Why: Supports both stream and batch processing with scalable computation.

10) **Data Lake and Warehouse**
Role: Stores raw logs and aggregated data for reporting and ML.
Technology: Amazon S3 / Redshift / HDFS

Why: Scalable, cost-effective storage optimized for long-term data analysis.

11) **Business Intelligence Dashboards**
Role: Provides data visualizations and reports to stakeholders.
Technology: Power BI / Grafana
Why: Easy integration, supports real-time insights, and enables decision-making.

12) **Authentication**
Role: Manages secure login, roles, and sessions.
Technology: OAuth2 / JWT (Auth0 / Firebase)
Why: Secure, token-based, and compatible with social login and role-based access.

13) **Cloud Infrastructure**
Role: Hosts all platform components with global availability and autoscaling.
Technology: AWS / GCP
Why: Offers reliability, multi-region deployment, and managed services.

14) **CI/CD Pipelines**
Role: Automates building, testing, and deploying of microservices.
Technology: GitHub Actions / GitLab CI
Why: Enables rapid, safe, and repeatable deployment workflows.

A. Components and roles Data Flow

This section describes the function of each architectural component and the data flow between them in the proposed short-form video platform.

Frontend (Web & Mobile Clients)

Role: Provides the user interface for viewers, content creators, advertisers, and administrators. It sends user-generated events (likes, comments, uploads) and receives content feeds, notifications, and analytics.
Data Flow: Sends HTTPS requests to the backend through the API Gateway and receives dynamic content via APIs and dashboards.

API Gateway / Load Balancer

Role: Routes incoming requests to the corresponding microservices, enforces authentication, rate limiting, and load balancing.
Data Flow: Acts as the communication hub between frontend clients and backend microservices.

Microservices Layer

Role: Contains independent services for different domains:

- **User Service** – registration, login, profile management.
- **Video Service** – upload, metadata, feed logic.
- **Ad Service** – campaign creation, targeting, analytics.
- **Payment Service** – transactions, monetization.

- **Moderation Service** – report handling and content filtering.

Data Flow: Services interact with databases (PostgreSQL, MongoDB), and publish events to Kafka.

Ingestion Layer (Kafka / Kinesis)

Role: Processes real-time data streams, decoupling event producers and consumers.
Data Flow: Receives events from microservices and delivers them to processing engines and data stores.

Storage Layer

PostgreSQL: For structured transactional data (users, payments).
MongoDB: For flexible, semi-structured data (videos, interactions).
Redis / Memcached: For caching trending videos, sessions, and fast-access data.
Data Flow: Microservices write/read data directly; Spark processes and updates MongoDB/Data Lake; Redis returns cached results to frontend.

Processing Engine (Apache Spark Streaming)

Role: Performs real-time analytics (views, engagement, retention).
Data Flow: Consumes events from Kafka, writes processed insights to MongoDB and the Data Lake.

Data Lake / Warehouse (S3, Redshift, HDFS)

Role: Stores raw logs and processed data for long-term analytics and reporting.
Data Flow: Receives data from Spark and microservices; queried by BI dashboards.

Serving Layer

BI Dashboards: (Power BI, Grafana) Display real-time metrics for creators, advertisers, and admins.
Public APIs: Provide structured data to frontend apps.
Data Flow: Reads from MongoDB or Data Warehouse; returns content and analytics to frontend.

Cloud Infrastructure (AWS / GCP)

Role: Hosts all components and enables autoscaling, deployment pipelines, observability, and resilience.
Data Flow: Manages and orchestrates underlying services and deployments across regions.

End-to-End Data Flow Example

1) A user uploads a video via the frontend.
2) The request flows through the API Gateway to the Video Service.
3) Metadata is saved in MongoDB and the event is published to Kafka.
4) Kafka sends the event to Spark Streaming for processing.
5) Spark updates aggregated metrics in MongoDB and the Data Lake.
6) Dashboards visualize insights from the Data Lake.
7) Redis caches popular videos for faster frontend delivery.

IV. Information Requirements

The system must retrieve and manage various types of information to support the functionalities outlined in the business model and user stories. Below is a list of the main types of information, their descriptions, and how they support the platform's logic.

1. User Profiles and Authentication Data

Description: Information about users, including credentials (for authentication), profiles, roles (viewer, creator, advertiser, admin), and preferences.
Link to Business Model: Essential for all customer segments (users, creators, advertisers, admins) to interact with the platform.
Link to User Stories:

- User registration and authentication (Functional Requirement #1)
- Follow creators and receive notifications (User Story: *Follow a Creator*)

2. Video Metadata

Description: Details about uploaded videos, such as title, description, tags, duration, uploader ID, geolocation, and engagement metrics (likes, comments, shares).
Link to Business Model: Core to the value proposition for creators and users, enabling content discovery and monetization.
Link to User Stories:

- Video upload and management (Functional Requirement #2)
- Search for videos with filters (User Story: *Search for Videos*)
- Discover trending videos (User Story: *Discover Trending Videos*)

3. User Interactions

Description: Data on user interactions with videos, including likes, comments, shares, watch time, and follow actions.
Link to Business Model: Drives engagement metrics for advertisers and creators, supporting monetization.
Link to User Stories:

- Like a video (User Story: *Like a Video*)
- Comment on a video (User Story: *Comment on a Video*)
- Real-time analytics for advertisers (User Story: *View Real-Time Analytics*)

4. Advertisements and Campaigns

Description: Information about ad campaigns, including targeting parameters (demographics, interests), budgets, impressions, clicks, and conversions.
Link to Business Model: Key revenue stream from advertisers; supports targeted ad placements.
Link to User Stories:

- Create ad campaigns (User Story: *Create Ad Campaign*)
- Edit campaign targeting (User Story: *Edit Campaign Targeting*)
- Export campaign reports (User Story: *Export Campaign Report*)

5. Monetization and Transactions

Description: Records of financial transactions, such as virtual gifts, ad revenue payouts, and sponsored content payments.
Link to Business Model: Critical for creator monetization and platform revenue streams.
Link to User Stories:

- Process payments for creators (Functional Requirement #6)
- Pause/resume ad campaigns (User Story: *Pause/Resume Campaign*)

6. Content Moderation Reports

Description: Data on flagged videos, including report reasons, severity levels, and moderation status (approved/rejected).
Link to Business Model: Ensures platform health and compliance, supporting community safety.
Link to User Stories:

- Report inappropriate content (Functional Requirement #5)
- Review flagged videos (User Story: *Review Content Reports*)

7. Analytics and Business Intelligence (BI)

Description: Aggregated metrics for dashboards, including user retention, engagement trends, revenue reports, and content performance.
Link to Business Model: Provides actionable insights for admins, creators, and advertisers.
Link to User Stories:

- Access BI dashboards (Functional Requirement #7)
- Monitor system health (User Story: *Monitor System Health*)

V. Query Proposals

This section presents a set of representative queries designed to demonstrate how the proposed data system retrieves key information to support business needs and user interactions. In accordance with the hybrid architecture of the platform, the queries are divided between relational (SQL) and non-relational (NoSQL) approaches, each serving different access patterns and performance requirements.

For each of the main information requirements identified in the system—ranging from user profiles to business intelligence metrics—this section provides at least one example query, along with a brief explanation of its purpose and the insight it delivers. When applicable, the queries demonstrate how SQL and NoSQL complement each other by handling different aspects of the same functional domain (e.g., video metadata vs. engagement analytics).

A. Sample Queries

1) User Profiles and Auth Data:

- SQL Query

```
SELECT u.user_id, u.username, u.email, u.role,
COUNT(f.follower_id) AS follower_count
FROM User u
LEFT JOIN Follow f ON u.user_id = f.followed_id
WHERE u.role = 'CREATOR'
GROUP BY u.user_id, u.username, u.email, u.role
ORDER BY follower_count DESC
LIMIT 10;
```

Purpose: Retrieve the top 10 creators with the most followers. This helps identify high-impact users for promotional or monetization purposes.

2) Video Metadata:

- SQL Query

```
SELECT video_id, title, description, upload_datetime
FROM Video
WHERE visibility = 'PUBLIC'
ORDER BY upload_datetime DESC
LIMIT 10;
```

- noSQL

```
db.Reactions.find({ video_id: 123 }, { likes_count: 1 });
```

Purpose: The SQL query retrieves recently uploaded public videos. The NoSQL query gets real-time engagement data (like counts).

3) Video Metadata:

- NoSQL Query

```
db.Views.aggregate([
  { $match: { video_id: 123 } },
  { $unwind: "$view_events" },
  { $group: { _id: null, avg_duration: { $avg: "$view_events.duration" }
]);
```

Purpose: Calculate average watch time of a specific video to measure engagement.

Insight: This data is essential for performance scoring, recommendation engines, and creator insights. NoSQL enables high-throughput ingestion and aggregation of watch data.

4) Advertisements and Campaigns:

- SQL Query

```
SELECT campaign_id, name, budget, start_date, end_date
FROM Campaign
WHERE advertiser_id = 21 AND end_date >= CURRENT_DATE;
```

- NoSQL Query

```
db.AdImpressions.countDocuments({ campaign_id: 21, type: "click" });
```

Purpose: SQL fetches active campaign metadata. NoSQL counts user clicks for real-time campaign performance.

Insight: SQL supports campaign management, while NoSQL enables rapid, scalable analytics of ad engagement.

5) Monetization and Transactions:

- SQL Query

```
SELECT u.username, t.amount, t.type, t.transaction_datetime
FROM Transaction t
JOIN User u ON t.user_id = u.user_id
WHERE t.type = 'SUBSCRIPTION'
ORDER BY t.transaction_datetime DESC
LIMIT 10;
```

Purpose: Display recent subscription transactions, including user info, for administrative and accounting purposes.

Insight: Critical for verifying revenue flows and creator payments.

6) Content Moderation Reports:

- SQL Query

```
SELECT r.video_id, r.reason, r.status, u.username AS reported_by
FROM ContentReport r
JOIN User u ON r.reporter_id = u.user_id
WHERE r.status = 'PENDING';
```

Purpose: List all pending reports with context for moderators to review inappropriate content.

Insight: Enables content moderation workflow, ensuring safety and compliance.

7) Analytics and Business Intelligence (BI):

- SQL Query

```
SELECT DATE(transaction_datetime) AS day,
       SUM(CASE WHEN type = 'SUBSCRIPTION' THEN amount ELSE 0 END)
       AS subs_revenue,
       SUM(CASE WHEN type = 'AD_PAYMENT' THEN amount ELSE 0 END)
       AS ads_revenue
FROM Transaction
GROUP BY day
ORDER BY day DESC
LIMIT 7;
```

Purpose: Compare platform revenue by source over the past week.

- NoSQL Query

```
db.Views.aggregate([
  { $unwind: "$view_events" },
  { $group: { _id: "$video_id", views: { $sum: 1 } } },
  { $sort: { views: -1 } },
  { $limit: 5 }
]);
```

Purpose: Find top-performing videos by views. NoSQL computes it from raw event data.

8) System Performance Metrics:

- NoSQL Query

```
db.SystemMetrics.aggregate([
  { $match: { metric: "latency",
    timestamp: { $gte: ISODate("2025-05-28T00:00:00Z") } } },
  { $group: { _id: "$endpoint", avg_latency: { $avg: "$value" } } }
]);
```

Purpose: Compute average latency per API endpoint for a given day.

Insight: Critical for monitoring and scaling backend services; identifies performance bottlenecks in the platform.

B. Needs Leverage

The hybrid architecture of the platform is intentionally designed to take advantage of the distinct strengths of relational and non-relational data systems. Relational (SQL) databases are used for structured, transactional data that requires consistency, referential integrity, and auditability—such as user accounts, subscriptions, payments, and moderation workflows. These queries support administrative tasks, financial tracking, and long-term analytics.

In contrast, NoSQL (document-based) storage is optimized for high-velocity, schema-flexible, and interaction-heavy data, such as video views, user reactions, comments, session logs, and ad impressions. These queries support real-time features like video feed generation, engagement metrics, live analytics dashboards, and monitoring.

In practice, both technologies work in tandem: SQL provides a reliable backbone for core business operations, while NoSQL enables scalable, low-latency access to dynamic content and behavior-driven insights. For example, while SQL might compute total subscription revenue per day for business intelligence dashboards, NoSQL allows instant retrieval of a user's watch history or a video's live engagement statistics.

VI. Improvements to Workshop 1

A. ER Diagram

During Workshop 1, a preliminary ER diagram was created to represent the data model of the short-form video platform. However, this initial version was highly simplified and lacked many of the core entities and relationships required to support the business model, user interactions, and monetization features described in the project.

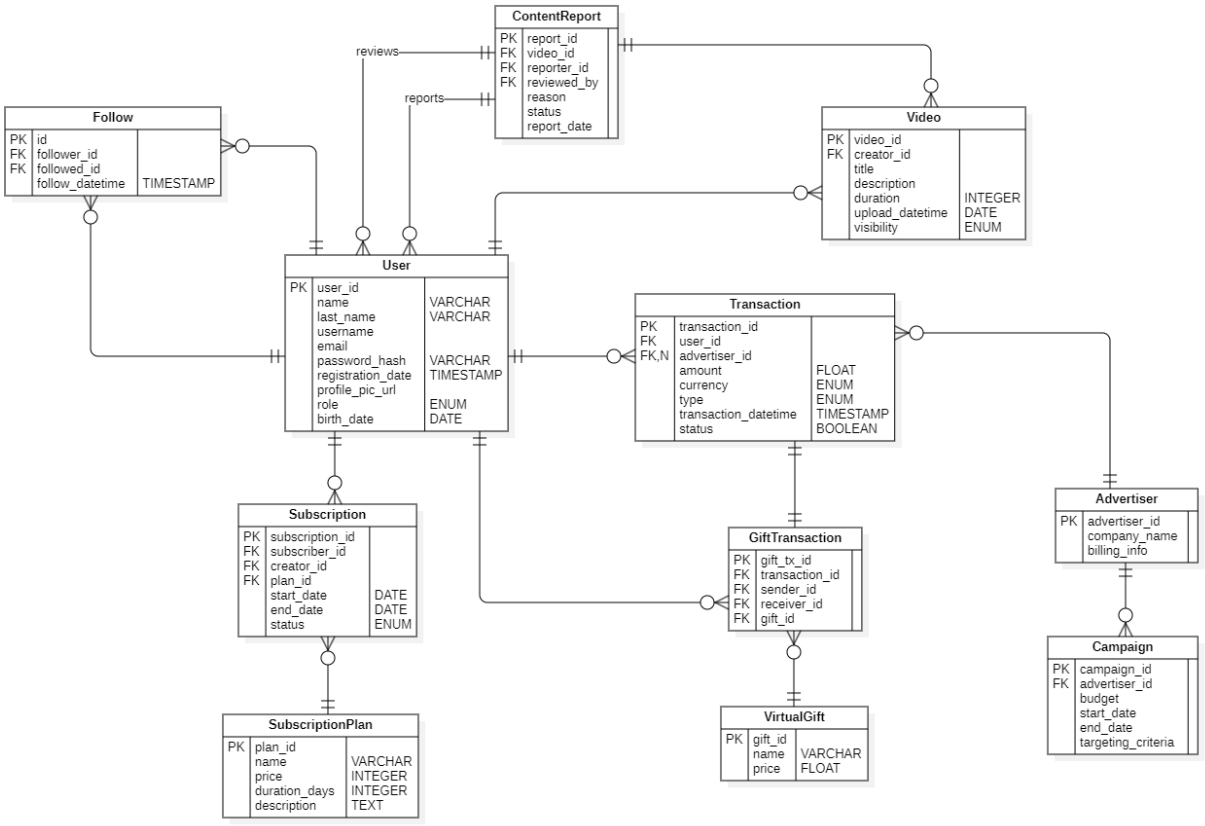

Fig. 2. ER Diagram Improved

1) Key improvements made:

- **Expanded Entity Set:** The new ER diagram introduces all essential entities aligned with the business logic, including Subscription, SubscriptionPlan, GiftTransaction, VirtualGift, Campaign, ContentReport, Follow, and a refined Transaction entity.

- **Role Management:** Instead of separating user types into different tables, the revised design uses a unified User table with a role attribute (GENERAL, CREATOR, ADMIN, etc.), simplifying access control and relationships.

- **Monetization Support:** The model now fully supports creator monetization through subscriptions and virtual gifts, along with advertiser billing via linked transactions.

- **Moderation Logic:** The new ContentReport table allows users to report inappropriate content, with the ability for administrators to review and act upon it.

- **Normalized Relationships:** Many-to-many relationships such as user follows and video interactions are handled using separate association tables like Follow, improving scalability and clarity.

- **Consistency with NoSQL Design:** The improved ER diagram complements the NoSQL structure by maintaining structured, transactional data in SQL while allowing unstructured, high-volume interactions to be handled separately.

B. NoSQL Structure designed

During workshop 1 no NoSQL database structure was proposed, this is the initial structure that meets the criteria of our project, and complements the data in the SQL database.

Collection	**Purpose**	**Key Fields**	**Indexes**
Comments	Store all comments grouped by video	{ _id, video_id, comments:[{comment_id, user_id, text, timestamp}] }	video_id, comments.user_id
Reactions	Aggregate like/reaction counts (with optional history)	{ _id, video_id, likes_count, reactions:[{user_id, type, timestamp}] }	video_id
Views	Log view events for analytics	{ _id, video_id, view_events:[{user_id, duration, timestamp}] }	video_id, view_events.timestamp
Sessions	Sequence of user actions per session	{ _id, user_id, session_id, events:[{type, video_id, timestamp}] }	user_id, session_id
FeedCache	Cached personalized feed	{ _id (as user_id), feed:[{video_id, rank, score}], last_updated}	_id
AdImpressions	Log ad impressions/clicks for campaign analytics	{ _id, campaign_id, advertiser_id, user_id, video_id, timestamp, type, device, location, duration_viewed, interacted}	campaign_id, user_id
SystemMetrics	Technical system logs (latency, errors, etc.)	{ _id, timestamp, metric, value, endpoint, status, server_id}	timestamp, metric

TABLE I
NOSQL DATABASE COLLECTIONS