# Video Streaming Microservice Documentation

# I. Introduction

## Project Overview

### Background

The Video Streaming API project was initiated by our client, a major Hollywood production company, in response to the evolving landscape of digital entertainment consumption.

This customer is focused on building a cutting-edge video streaming platform and aims to develop a robust API that provides more functionality to content managers and improves user engagement.

### Industry Context

The entertainment industry is experiencing a paradigm shift towards digital platforms. To remain competitive, our customers recognize the need to provide a seamless and feature-rich video streaming experience. This API serves as a central tool to achieve this goal.

### Business Objectives

The main business goals of the Video Streaming API project are:

- Facilitating content managers to effectively publish and manage videos.
- Enables monetization of user viewing through detailed engagement metrics.
- Improve the overall user experience on video streaming platforms.

# Scope

## Inclusions

The Video Streaming API project includes the following main features:

- Stores information about the video, including metadata.
- Stream video content seamlessly.
- Track user engagement actions such as impressions and views.
- Provides important functionality such as video publishing, metadata management, and list deletion.

## Exclusions

The following features are outside the scope of the current project:

- Advanced analytics that go beyond basic engagement metrics.
- User account management, authentication, and authorization (managed by another authentication service).

# Purpose

## Problem Statement

The emergence of digital platforms requires advanced video streaming solutions that align with customers' business goals. The lack of  dedicated APIs for video management and engagement tracking prevents customers from optimizing content delivery and maximizing revenue from user views.

## Solution

The Video Streaming API aims to address the identified challenges by providing a comprehensive set of functionalities for content managers. This includes seamless video publishing, efficient metadata management, and accurate tracking of user engagement metrics. The solution is designed to integrate seamlessly into the client's existing ecosystem, promoting scalability and adaptability to future requirements.

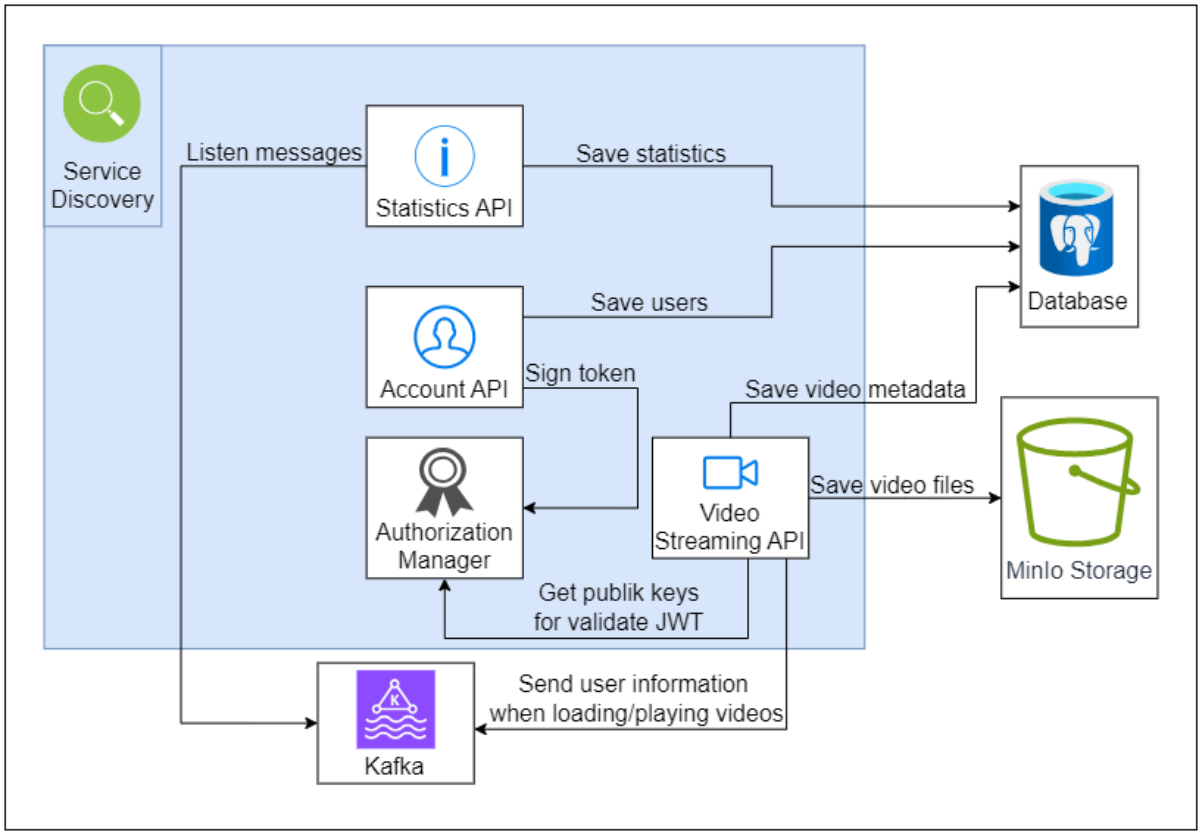
## Expected Outcomes

By the end of the project, the client anticipates achieving:

- Streamlined video publishing processes for content managers.
- Enhanced metadata management for improved content discoverability.
- Accurate and actionable engagement metrics for monetization strategies.
- A foundation for future enhancements and innovations in the video streaming platform.

# II. Architecture

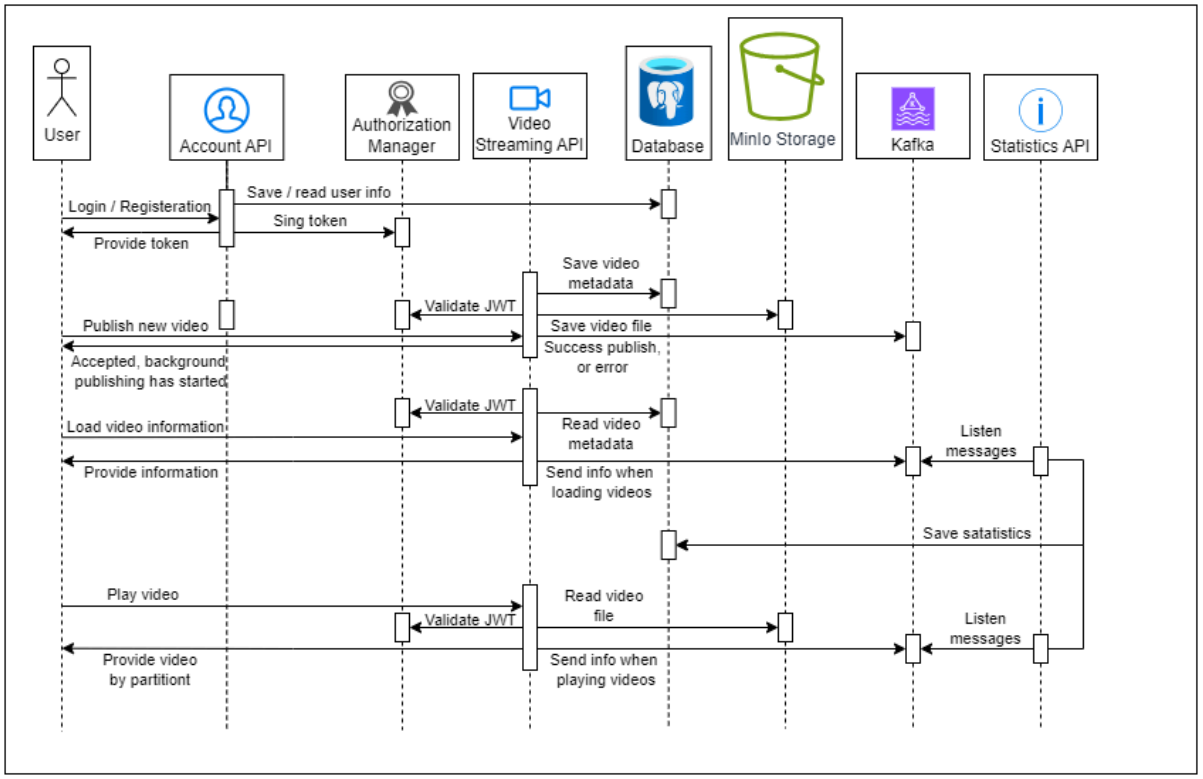
## High level view




The high-level architecture diagram illustrates the main components of the Video Streaming API, including the integration with MinIO for video content storage and PostgreSQL for metadata persistence.

At this stage, the logic for working with users and statistics is located in the Video Streaming service. In the future, this logic can be encapsulated in separate services, and interaction with the service responsible for statistics can be performed through communication via a message using Kafka.

## Low level view



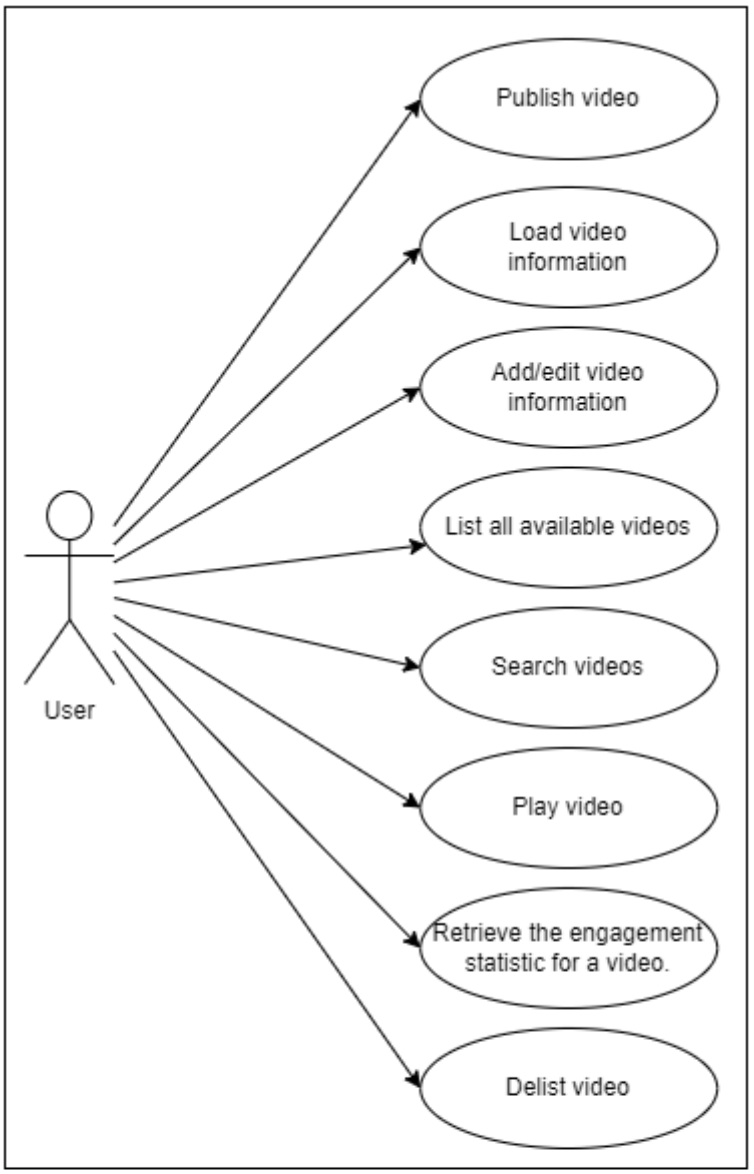

This section delves into the detailed architecture of each component, highlighting their responsibilities and interactions. Specifically, it outlines the integration with MinIO for video content storage and PostgreSQL for metadata persistence.

## Technology Stack

The following technologies were chosen to implement the video streaming API:

- **Java Development Kit (JDK) 21**: Chosen for  modern features and long-term support.
- **Spring Boot 3.2**: Selected for its ease of building robust and scalable Java applications.
- **Gradle**: Used for dependency management and build lifecycle.
- **MinIO**: Utilized for scalable and secure object storage, handling the storage of video content.
- **PostgreSQL**: Chosen as the relational database for persisting metadata related to videos.

# III. Use Case Diagrams




## Use Case 1: Publish Video

- **Description:** Users can publish videos through the Video Streaming API. This involves providing the video file and associated metadata (e.g., title, synopsis, director, etc.).
- **Preconditions:** User is authenticated and authorized to upload videos.
- **Postconditions:** The video is stored in MinIO, and metadata is stored in the PostgreSQL database.

## Use Case 2: Load video information

- **Description:** Users can retrieve information about a specific video through the Video Streaming API. This includes details such as title, synopsis, director, cast, release year, genre, and running time.
- **Preconditions:** The video exists in the system.
- **Postconditions:** The API returns the requested video information.

### Use Case 3: Add/edit video information

- **Description:** Users can add or edit metadata associated with a video.
- **Preconditions:** User is authenticated and authorized to modify video information.
- **Postconditions:** The updated metadata is stored in the PostgreSQL database.

### Use Case 4: List all available videos

- **Description:** Users can retrieve a list of all available videos through the Video Streaming API.
- **Preconditions:** There are videos available in the system.
- **Postconditions:** The API returns a list of available videos.

### Use Case 5: Search videos

- **Description:** Users can search for videos based on specific criteria (e.g., movies directed by a specific director) through the Video Streaming API.
- **Preconditions:** There are videos available in the system.
- **Postconditions:** The API returns a result set based on the search/query criteria.

### Use Case 6: Play video

- **Description:** Users can play a video through the Video Streaming API. The API retrieves the video content from MinIO and provides a playback experience.
- **Preconditions:** The video exists in the system, and the user has appropriate permissions.
- **Postconditions:** The video is played for the user.

### Use Case 7: Retrieve the engagement statistic for a video

- **Description:** Users can retrieve engagement statistics for a specific video through the Video Streaming API. This includes impressions (loading a video) and views (playing a video).
- **Preconditions:** There are videos available in the system.
- **Postconditions:** The API returns engagement statistics for the video.

### Use Case 8: Delist video

- **Description:** Users can delist (soft delete) a video through the Video Streaming API. This involves marking the video as inactive or hidden.
- **Preconditions:** User is authenticated and authorized to delist video. The video exists in the system.
- **Postconditions:** The video is marked as delisted in the database.

# IV. System Components

## Microservice Component

### Service Discovery

The Service Discovery Microservice, based on Netflix Eureka, plays a crucial role in managing and coordinating the dynamic and distributed nature of microservices within a system. Its primary purpose is to enable automatic service registration and discovery, facilitating seamless communication and collaboration among microservices.

## Authorization Manager

The Authorization Manager Microservice is a critical component in a microservices architecture, designed to handle token-based authentication and authorization. Leveraging Spring Security for authentication and io.jsonwebtoken for token handling, this microservice provides two key endpoints.

### Endpoints

1. **GET /auth/discovery/key**
    - **Purpose:** Returns the public key used for verifying RS256-signed tokens.
    - **Security:** Publicly accessible, allowing any microservice to obtain the public key necessary for validating RS256-signed tokens.
2. **POST /auth/sign**
    - **Purpose:** Generates and signs tokens, embedding user information in the payload.
    - **Security:** Restricted access, requiring authentication and authorization to ensure only authorized microservices can generate tokens.

### Token Generation

1. **Synchronous Token Generation**
    - **Algorithm:** Uses HMAC SHA-256 (HS256).
    - **Process:** Microservices send authentication requests with credentials. Upon successful authentication, the Authorization Manager generates a signed token. For verification, the same private key with which the token was signed is required.
2. **Asynchronous Token Generation**
    - **Algorithm:** Uses RSA SHA-256 (RS256).
    - **Process:** Similar to synchronous token generation, but leverages RSA for asynchronous signing. Enables secure token generation with private and public key pairs. To verify, must send a request to receive a public key from the Authorization Manager.

## Account Manager

The Account Manager Microservice handles user authentication and registration, providing JWT tokens upon successful login and registration. Additionally, it authorizes access to user information through various endpoints, including fetching user details by ID and retrieving a list of all users.

### Endpoints

1. **POST /user/login:**
    - **Type:** Public
    - **Purpose:** Allows users to log in, providing a JWT token upon successful authentication.
2. **POST /user/registration:**
    - **Type:** Public
    - **Purpose:** Enables user registration, providing a JWT token upon successful registration and login.
3. **GET /user/info:**
    - **Type:** Authorized
    - **Purpose:** Provides information about the currently logged-in user.
4. **GET /user/byId/{userId}:**
    - **Type:** Authorized
    - **Purpose:** Retrieves user information by ID, accessible only to authorized users.
5. **GET /user/all:**
    - **Type:** Authorized
    - **Purpose:** Provides information about all users, accessible only to authorized users.

## Video Streaming

The Video Streaming Microservice is responsible for publishing new videos, loading video information, and providing video playback functionalities. Additionally, it leverages Kafka to send statistics information messages when loading video information or playing a video. All endpoints are authorized.

### Endpoints

1. **POST /video/save:**
    - **Purpose:** Publishes a new video
    - **Request Body:** Requires a video file (**file** param) and additional video information (**info** param).
2. **GET /video/play/{videoId}:**
    - **Purpose:** Provides a segment of the video specified by **videoId**
3. **GET /video/metadata/all:**
    - **Purpose:** Provides information about all available videos.
4. **GET /video/metadata/{authorId}:**
    - **Purpose:** Provides information about all videos by a specific author identified by **authorId**.
5. **GET /video/metadata/author:**
    - **Purpose:** Provides information about all videos published by the currently **logged-in** user.
6. **GET /video/metadata/deleted:**
    - **Purpose:** Provides information about all deleted (not active) videos.
7. **DELETE /video/metadata/{videoId}:**

- **Purpose:** Marks a video as not active (soft delete) based on the specified videoId.

8. **GET /video/metadata/search:**
    - **Purpose:** Provides video information based on specified search criteria (genre, director, keyword, yearBefore, yearAfter, year).

### Statistics Manager

The Statistics Manager Microservice is responsible for saving and providing statistics for various video actions. It listens to Kafka messages to collect statistics data when users load video information or play a video. The microservice offers endpoints to retrieve comprehensive statistics, including Impressions (loading a video) and Views (playing a video) with associated video IDs. All endpoints are authorized.

#### Endpoints

1. **GET /stats/total:**
    - **Purpose**: Provides all video engagement statistics.
    - **Response:** Returns an array of StatisticsResponse containing videoId, Impressions, and Views.
2. **GET /stats/author:**
    - **Purpose:** Provides all video engagement statistics for the currently logged-in user.
    - **Response:** Returns an array of StatisticsResponse containing videoId, Impressions, and Views.
3. **GET /stats/byUserId/{userId}:**
    - **Purpose:** Provides all video engagement statistics for a specific user identified by userId.
    - **Response:** Returns an array of StatisticsResponse containing videoId, Impressions, and Views.
4. **GET /stats/byVideoId/{videoId}:**
    - **Purpose:** Provides all video engagement statistics for a specific video identified by videoId.
    - **Response:** Returns a StatisticsResponse containing videoId, Impressions, and Views.

## MinIO Component

MinIO is an open-source, high-performance object storage solution designed for scalability and efficiency. As a cloud-native storage system, MinIO enables organizations to build scalable and distributed storage infrastructure, providing a reliable platform for storing and retrieving unstructured data such as documents, images, and videos. Its key features include high scalability, multi-cloud compatibility, and compatibility with the S3 API, making it a versatile choice for modern cloud-native applications and data-intensive workloads.

MinIO empowers developers with a simple and scalable object storage solution that seamlessly integrates into various cloud environments.

### Kafka Component

Kafka is a high-throughput, fault-tolerant, and distributed streaming platform that serves as a reliable backbone for real-time data processing. It utilizes a publish-subscribe model to enable the seamless exchange of messages between producers and consumers, supporting scalable and fault-tolerant event-driven architectures.

Kafka is widely used for real-time data streaming, event sourcing, log aggregation, and other applications requiring efficient and distributed data handling. Its key features include horizontal scalability, fault tolerance, and support for event-driven microservices, making it a fundamental component in modern data architectures.

### PostgreSQL Component

PostgreSQL is a powerful open-source relational database management system known for its robustness, extensibility, and advanced features. As an ACID-compliant database, PostgreSQL ensures data integrity through Atomicity, Consistency, Isolation, and Durability. Its extensibility allows the integration of custom functions, operators, and data types, while advanced querying capabilities with SQL make it suitable for complex data retrieval tasks.

PostgreSQL is highly scalable, supporting sharding and replication to handle growing data volumes, making it a versatile choice for transactional applications, relational data storage, and sophisticated data management requirements.

## V. Deployment

Deployment process provides a streamlined approach to containerizing and orchestrating microservices, Kafka, MinIO, and PostgreSQL using Docker Compose. It facilitates scalability, maintainability, and ease of deployment in a containerized environment. Adjust configurations based on project specifics and requirements.

## VI. Security Considerations

### Authentication and Authorization

Security considerations for the Authorization Manager using Spring Security with HS256 and RS256 involve implementing strong authentication and authorization mechanisms.

For HS256, which is a synchronous algorithm, a shared secret key is utilized for token generation, ensuring secure authentication. In contrast, RS256, an asynchronous algorithm, employs public-private key pairs for enhanced security. Key management, including secure storage and distribution, is crucial for both algorithms.

Spring Security integration is vital for comprehensive authentication and authorization, defining roles and permissions for controlled access. Public and authorized endpoints, such as **/auth/discovery/key** and **/auth/sign**, must be carefully configured to prevent misuse. User information is embedded in the token payload for seamless authorization across microservices.

Secure communication is emphasized, requiring HTTPS for all interactions between microservices and the Authorization Manager. Key considerations also include token expiry, secure token storage, and continuous monitoring to detect and respond to any suspicious activities. Implementing these security measures ensures the integrity, confidentiality, and secure functioning of the Authorization Manager in a microservices environment.

### Data Encryption

Securely encrypting user passwords is paramount for protecting sensitive information. The BCryptPasswordEncoder is chosen for its efficacy in thwarting unauthorized access. Through the use of a unique and random salt, coupled with a configurable work factor, passwords are hashed, creating a fixed-length cryptographic representation stored alongside the salt in the database.

During authentication, the stored salt is retrieved, and the entered password is hashed for comparison. BCrypt's adaptive nature and built-in salting mechanism fortify defenses, while regular parameter updates and compliance with security standards ensure a resilient and proactive approach to password encryption. This process provides a robust foundation for secure user authentication within the system.

## VII. Future Enhancements

### Potential Improvements

#### Modular Security Library

Implement a modular approach by separating token verification, currently embedded in the security filter common login, into a dedicated security library. This enhances code organization and promotes reusability across multiple services.

#### User Roles and Access Control

Introduce user roles to the authentication system, allowing for fine-grained access control. Certain endpoints can be restricted based on roles, enhancing security and ensuring that users have access only to the functionalities appropriate for their roles.

#### Video Rating Feature

Enhance user engagement by introducing a video rating feature. This functionality enables users to assign ratings to videos, providing valuable feedback and potentially influencing content recommendations. Implementing this feature enriches the platform and enhances the overall user experience.

# VIII. Conclusion

## Summary

In summary, the development and implementation of the Video Streaming API represent a significant milestone in meeting the evolving demands of the digital entertainment industry. This comprehensive documentation has provided insights into the various facets of the project, from its inception to the technical intricacies of the solution.

## Achievements

The successful creation of the Video Streaming API has resulted in the following key achievements:

- **Empowered Content Managers**: Content managers now have a powerful tool at their disposal, enabling them to publish videos seamlessly, manage metadata effectively, and respond dynamically to user engagement metrics.
- **Enhanced User Experience**: The integration of the API into the video streaming platform has contributed to an elevated user experience. Users can now enjoy a more streamlined and personalized content consumption journey.
- **Monetization Opportunities**: The accurate tracking of user engagement metrics, including impressions and views, has laid the foundation for effective monetization strategies. The client is well-positioned to capitalize on user viewership and optimize revenue streams.
- **Scalability and Adaptability**: Leveraging technologies such asSpring Boot, MinIO, PostgreSQL, and Gradle, the Video Streaming API is designed with scalability and adaptability in mind. This ensures that the solution can evolve alongside the dynamic requirements of the digital entertainment landscape.

## Future Outlook

As the Video Streaming API becomes an integral part of the client's video streaming platform, we look forward to exploring future opportunities for innovation and enhancement. Continuous collaboration and feedback will be invaluable in refining the API, ensuring it remains at the forefront of industry standards.

In conclusion, the Video Streaming API stands as a testament to our commitment to delivering high-quality solutions that meet the unique needs of our clients and contribute to the advancement of the digital entertainment landscape.