Artifact

Data Distribution OpenVidStreamer

by Radoslav Radev

Advanced Software Engineering Semester 6

13/04/2024

ver. 1.0

Data Requirements per System part:

1. Account Microservice

Usage: Manages user account information, registration, login, subscription, and payment.

Data Storage:

- **Data Requirements:**
  - **Users Data:** Securely store user-related data such as name, email, password, subscription status.
- **Suitable Data Storage:** Relational databases
  - **Chosen**: MySQL
  - **Justification:** MySQL is a robust relational database management system that supports complex queries, ensures data integrity, and provides ACID (Atomicity, Consistency, Isolation, Durability) properties. It is suitable for storing structured user data and handling frequent read/write operations efficiently.

2. Video Library Microservice

Usage: Manages video library metadata.

Data Storage:

- **Data Requirements:**
  - **Video Metadata:** Store details such as video titles, descriptions, tags, upload dates.
- **Suitable Data Storage:** Relational databases
  - **Chosen**: MySQL
  - **Justification:** MySQL is well-suited for managing structured data with relationships, such as video metadata. It supports indexing and querying capabilities that are essential for efficient retrieval of video details.

3. Video Streamer Microservice

Usage: Streams video content to users.

Data Storage:

- **Data Requirements:**
  - **Video Files:** Stream video files stored in the storage bucket.
- **Suitable Data Storage:** Storage Bucket (e.g., AWS S3)

    - **Justification:** Using a storage bucket for video files ensures scalable, durable, and cost-effective storage for large video files. It allows for efficient streaming and supports high availability and redundancy.

4. Recommendation Algorithm Microservice

Usage: Provides video recommendations to users.

Data Storage:

- **Data Requirements:**
    - **User Watch Statistics:** Stores user interaction data such as watched videos, viewing history, and preferences.
    - **Cache Data:** Frequently accessed recommendation data for quick retrieval.
- **Suitable Data Storage:** MySQL (for user statistics) and Redis (for caching) for further explanation on caching please refer to the artifact about adding caching to openVidStreamer (Radev, 13/04/2024, #)
    - **Justification:**
        - **MySQL:** Suitable for storing structured and relational data like user watch statistics, which require complex queries and data integrity.
        - **Redis:** An in-memory data structure store used for caching frequently accessed data, providing low-latency data access and improving the performance of recommendation algorithms.

5. Render Microservice

Usage: Renders video content for the video library.

Data Storage:

- **Data Requirements:**
    - **Rendered Videos:** access raw video files, and save processed video chunks.
- **Recommended Storage:** Storage Bucket (e.g., AWS S3)
    - **Chosen**: Custom build NFS PV storage bucket
    - **Justification:** The storage bucket provides scalable and durable storage for large video files, facilitating easy access and retrieval by the video library and streaming services.

6. Upload Microservice

Usage: Handles video uploads, forwards them to Render MS for processing, and updates the video library.

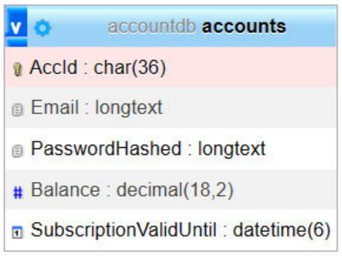
Data Storage:

- **Data Requirements:**
    - **Video Files:** store unprocessed video files.
- **Recommended Storage:** Storage Bucket (e.g., AWS S3)
    - **Chosen**: Custom build NFS PV storage bucket
    - **Justification:** Using a storage bucket ensures scalable, reliable, and cost-effective storage for large volumes of video files. It supports efficient upload and retrieval processes.

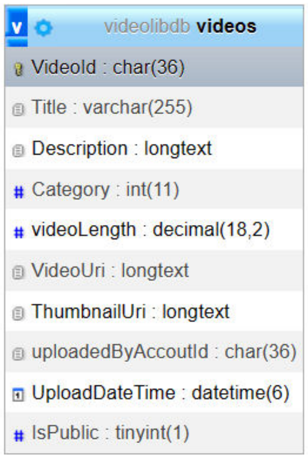
Summary of Data Storage:

- **Relational Databases:** AccountDB, VideoMetadataDB, UserWatchStatistics
    - Used for structured, relational data that require complex queries and data integrity.
- **Redis Cache:** Caching layer for frequently accessed recommendation data, improving performance.
- **Storage Bucket (e.g., AWS S3):** Used for storage of large video files, supporting video rendering, uploading, and streaming services.

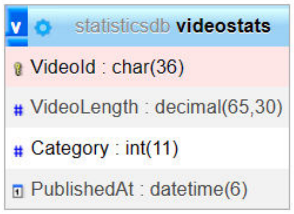
Overview of Relational Data:



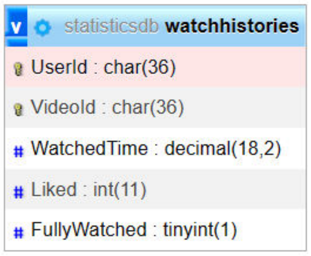
accountdb **accounts**
AccId : char(36)
Email : longtext
PasswordHashed : longtext
Balance : decimal(18,2)
SubscriptionValidUntil : datetime(6)



videolibdb **videos**
VideoId : char(36)
Title : varchar(255)
Description : longtext
Category : int(11)
videoLength : decimal(18,2)
VideoUri : longtext
ThumbnailUri : longtext
uploadedByAccoutId : char(36)
UploadDateTime : datetime(6)
IsPublic : tinyint(1)



statisticsdb **videostats**
VideoId : char(36)
VideoLength : decimal(65,30)
Category : int(11)
PublishedAt : datetime(6)



statisticsdb **watchhistories**
UserId : char(36)
VideoId : char(36)
WatchedTime : decimal(18,2)
Liked : int(11)
FullyWatched : tinyint(1)

The current Design is "Database per Service" approach, but specifically emphasizing decentralized data management.

This decentralized approach means that each service manages its own database, and any relationships that would traditionally be enforced through foreign keys are now managed at the application level through messaging. This is part of a broader strategy to ensure that services are decoupled and can be developed, deployed, and scaled independently.