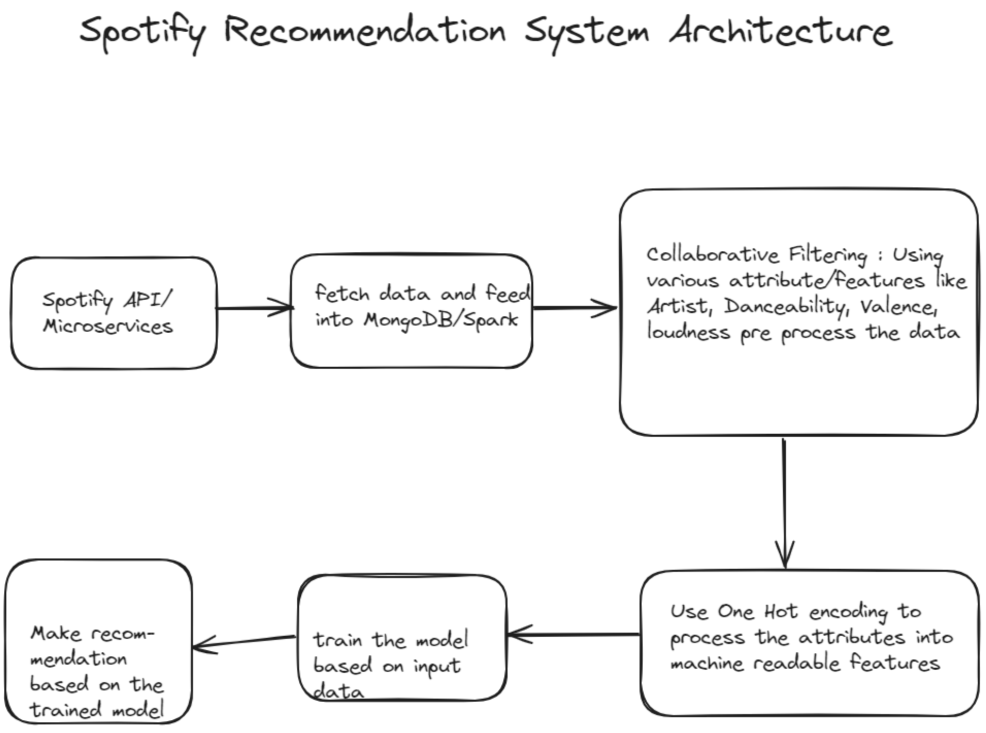
Group Submission
Name : Abhishek Kumar Singh (2022og04009)
Name : Priyesh Kumar Singh (2022og04026)


Spotify Recommendation System Architecture
Spotify API/ Microservices
fetch data and feed into MongoDB/Spark
Collaborative Filtering : Using various attribute/features like Artist, Danceability, Valence, loudness pre process the data
Use One Hot encoding to process the attributes into machine readable features
train the model based on input data
Make recom-mendation based on the trained model

Designing a system that supports both real-time stream processing and batch analytics for a music recommendation platform involves considering various components, databases, and trade-offs. Here's a high-level outline of the design:

Components:

Backend Services: Use a microservices architecture to handle different functionalities like user management, recommendation generation, data processing, etc.

APIs: Develop RESTful APIs to enable interaction between frontend applications and backend services. These APIs will handle user requests, data retrieval, and recommendations.

Data Models: Define robust data models to store user profiles, track information, and user interactions. These models should support efficient retrieval and processing.

Rationale and Trade-offs:

CAP Theorem Consideration:

For real-time stream processing, consistency and availability are crucial. Consistency ensures that users get timely and accurate recommendations, while availability ensures the system remains responsive.

For batch analytics, where large volumes of data are processed offline, consistency might be slightly relaxed in favor of availability and partition tolerance.

Tech Choices:

NoSQL Database: MongoDB provides flexibility in handling diverse data and scaling horizontally. DynamoDB offers low-latency access to data at any scale.

Batch Analytics: Apache Spark, due to its in-memory processing capabilities and ability to handle large-scale data analytics efficiently.

Design Rationale:

Microservices: To ensure scalability, fault isolation, and ease of maintenance, microservices architecture is preferred. Each service can focus on a specific task and can be independently scaled.

NoSQL Database Choice: MongoDB offers flexibility in handling unstructured or semi-structured data, crucial for storing diverse track and user information.

Batch Analytics: Apache Spark's distributed computing capabilities allow for efficient processing of historical data to improve recommendation algorithms.

This design aims to strike a balance between real-time responsiveness for user recommendations and batch analytics for refining recommendation models. The trade-offs involve maintaining consistency in real-time processing while ensuring scalability and fault tolerance across both real-time and batch processing scenarios.

Query Execution Loom Video Link :

OLTP Mongo DB Loom Link : https://www.loom.com/share/623073f1a30f4f5299aea94052f1e45e

OLAP Spark Loom Link : https://www.loom.com/share/a360768105d345b0b65b52f553f19e24