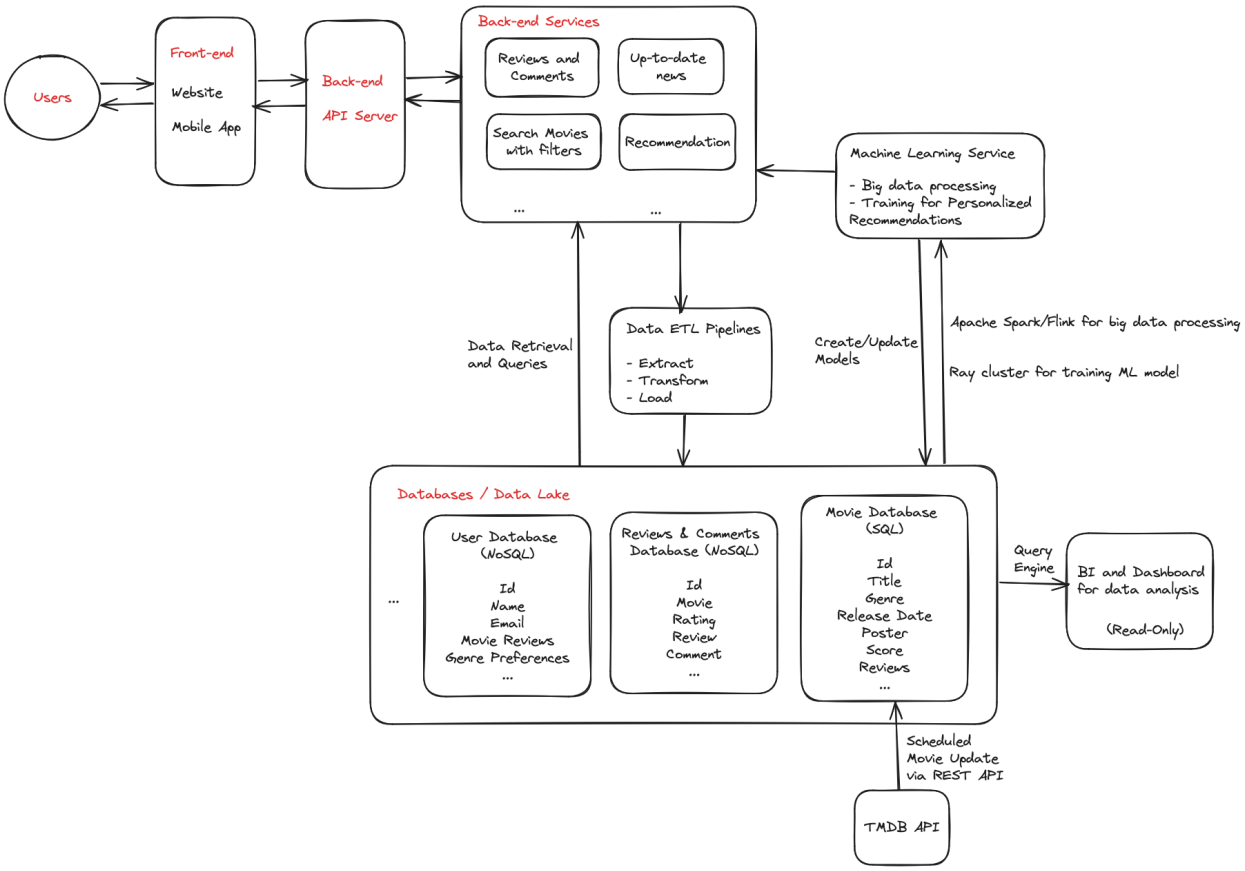
Movie Reviews Platform

Purpose of the Application

The application aims to be a platform built for movie fans. It strives to create an open and friendly space for people to share opinions, comments and ratings about movies of many genres and languages. Audiences can access the most up-to-date news about directors, actors, actresses and movies, and search for movies with a rich collection of filters. The application will also intelligently recommend movies to users, based on their past reviews/comments, and genre preferences.

Design and System Requirements



1. **Front-end Servers**
    - **Description**

      As a platform built for individual movie enthusiasts, the application will be available on the web and mobile app, making it very convenient for users to access. Unlike a banking system that requires extreme consistency, high availability is the most important non-functional requirement for our application, in the worst-case scenario, as long as it is not a monetary use case (e.g. purchase/payment), we should always prioritize availability over consistency. We could use a Kubernetes cluster to host front-end servers, as Kubernetes is one of the best choices for hosting stateless applications.
    - **Technologies**: HTML + CSS + React + TypeScript
    - **Requirements**: Low-latency, high-availability, auto-scaling
    - **Tests**: unit tests, end-to-end automated selenium tests, load testing

2. **API Servers**
    - **Description**

      The application will have a decoupled front and back-end for better scalability, high availability and easier maintenance. Latency is very critical in creating a supreme customer experience. Front-end will talk to the back-end through the API server, and the API will utilize GraphQL over REST style to improve performance by minimizing data transferred through the network. We will use web frameworks like FastAPI for their high concurrency to reduce request latency further. The servers will have auto-scaling enabled to promptly react upon traffic spikes. Since HTTP is a stateless protocol, we can deploy it using Kubernetes.
    - **Technologies**: FastAPI + Python
    - **Requirements**: Low-latency, high-availability, auto-scaling
    - **Tests**: unit tests, integration tests with database, load testing

3. **Back-end Servers**

    - **Description**

      These are the core servers of our business logic. Each service should be evaluated and scaled independently, based on the nature of the traffic load.

    - **Requirements**: based on the nature of the specific service, requirements may change, but generally these services should have high availability and consistency.

4. **Data Storage**

    - **Description**

      On a high level, we will have two types of databases. An OLTP NoSQL database for fast insertions and modifications, for example, reviews and comments, which are created/edited/deleted at a very high frequency, but normally each one is independent, therefore key-value/document databases like MongoDB/DynamoDB are the preferred choices.

      The movie data will be stored in an OLAP-fashion SQL database. For example, when the size is small we can use a local DuckDB, or when it grows to a large scale, we can migrate the data to a cloud service like BigQuery in Google Cloud. The logic is that movie data is normally fixed and will less likely get updated. Also, movie data is generally read-heavy but not write-heavy, we need to access it for movie searching, recommendation, and business intelligence/analytics use cases. To support different complex queries, the database needs to be fully ANSI-compliant.

      On top of databases, we also need to have a data lake like S3/Minio to serve as a loading dock, we can dump all kinds of unprocessed data here before we figure out how to use them.

    - **Technologies**: MongoDB/DynamoDB(NoSQL) + DuckDB(SQL) + S3/Minio for data lake

    - **Requirements**: high availability, strong/eventual consistency

- **Tests**: integration tests, load testing

5. **Business Intelligence / Analytics Platform**
    - **Description**

      These are read-only components on top of our data storage. Data is already pre-cleaned and processed so that we can avoid as many run-time transformations/joins as possible.
    - **Technologies**: BigQuery / Apache Pinot or Druid
    - **Requirements**: high availability, multi-tenant, fast batch/streaming data ingestion
    - **Tests**: load testing

6. **Data Processing and Machine Learning Platform**
    - **Description**

      These servers are responsible for processing large-scale batch/streaming data, and training machine-learning models for movie recommendation service. Tools we can utilize include Apache Spark/Flink, Ray cluster etc. For high-performance and decoupled architecture, a decent-sized Kafka/Message Queue system is also required.
    - **Technologies**: Apache Spark/Flink, Ray, MLflow etc.
    - **Requirements**: abundant computing resources, multi-tenant, fast training
    - **Tests**: model validations, prediction accuracy