Movie Library

DSCI 551 Final Project - Team #79

Team Details

Student #1

- Name: Pooridon Rattanapairote
- Email: prattana@usc.edu
- USC ID: 1469709999

Student #2

- Name: Pannawat Chauychoo
- Email: pchauych@usc.edu
- USC ID: 7282127237

Team members background and skills

Pooridon R.

- Background: Product Manager at Garena Online (Thailand) Co., Ltd
- Skills: Java, Python, SQL

Pannawat C.

- Background: Technology consultant
- Skills: R, Python, SQL

Project requirements

Our project involves developing a comprehensive database system integrated with both frontend and backend applications, ensuring compatibility and efficiency across various aspects:

Database

- The system must efficiently store vast amounts of data across multiple databases, utilizing partitioning strategies to enhance scalability and minimize latency, ensuring robust data management
- It should enable precise data operations, allowing for effective data insertion and retrieval through specific functions, facilitating smooth data handling

Admin Interface (Data Managers)

- A straightforward user interface (UI) needs to be designed for Data Managers. This UI will enable them to perform data operations such as adding, removing, and updating data entries, either individually or in batches. These changes must be accurately reflected in the primary database and its replicas (if any) to maintain data integrity and consistency

Frontend Interface (End-Users)

- We will develop an application for a specific platform, either web or mobile, designed for end-user engagement. This approach allows us to focus on optimizing user experience and interface design for that particular platform
- The application must be engineered to handle user interactions swiftly, ensuring quick responses when accessing or managing data in the database. This includes efficient data retrieval and updates, providing a seamless and responsive user experience

This project demands a cohesive integration of database management techniques with user-centric application development to ensure a seamless, efficient, and scalable system.

Planned Implementation

Idea:

- Netflix shows library & recommendation engine
- Allow users to search movies by directors, actors, reviews, etc
- Allow users to filter and sort the result by specific attributes
- Set up: database (storage), admin interface (data-manager), recommendation system, frontend interface (for end-user)

Backend

Database

- Database: MongoDB Atlas
- Library: PyMongo
- Type: Document-based

Server

- Library: Flask

Collections

- Movies: This collection would store information about each movie, including title, director, cast, genres, release date, and any other relevant metadata.
- Users: This collection would contain user profiles, including information like username, email, subscription type, and preferences.
- People: This collection would contain director or actor details, including name, award, age, and type
- UserBehavior: This collection would track user interactions with the platform, such as movies clicked, movies watched (by clicking watched button), movies like & dislike. Its field should include as follows:
    - "_id"
    - "userId": Reference to Users collection
    - "movieId": Reference to Movies collection
    - "actions": {
        "liked": User liked the movie
        "disliked": User disliked the movie
        "watched": User has watched the movie
        "clicked": User has clicked the movie
      }
    - "clickedOn": Date when the user watched the movie

Database Replication Strategy

We will implement a MongoDB replication strategy using a replica set, which is a group of MongoDB server instances that maintain the same data set.

Planned implementation:

1. Replica Set Configuration: We will configure a MongoDB replica set consisting of several nodes: one primary node that receives all write operations, and multiple secondary nodes that replicate the primary node's data

2. Data Synchronization: MongoDB uses an oplog (operations log) on the primary node to maintain a record of all changes to the database. Secondary nodes replicate this oplog and apply the operations to their data sets in an asynchronous process
3. Automatic Failover: In the event of a primary node failure, the replica set will automatically perform an election to determine which secondary node will become the new primary. This process ensures minimal downtime and continuous availability of the database for both read and write operations.
4. Read Distribution: To optimize performance and distribute the workload, we can configure read preferences to allow read operations from secondary nodes. This strategy is particularly beneficial for read-intensive applications like ours, where users frequently query movie and user interaction data.
5. Scalability: As our platform grows, the replica set can be scaled horizontally by adding more secondary nodes.
6. Backup and Recovery: The use of secondary nodes in the replica set facilitates efficient backup processes without impacting the performance of the primary node. Regular backups will be scheduled from secondary nodes, ensuring data durability and providing a robust recovery mechanism in case of data loss.

- Libraries:
    - Flask
- Function:
    - Allow administrators to add, update, delete movies and people (casts & directors)
        - Allow add and delete by batch (multiple rows) by using file uploading with provided template
    - Allow administrators to add, update, delete other collections such as UserBehavior and Users

Admin Interface (Data Managers)

- A graphical user interface for only admins
- Libraries:
    - PySimpleGUI
- Function:
    - Allow administrators to add, update, delete movies and people (casts & directors)
        - Allow add and delete by batch (multiple rows) by using file uploading with provided template
    - Allow administrators to add, update, delete other collections such as UserBehavior and Users

Frontend (End-Users)

- Library: Streamlit (https://github.com/streamlit/streamlit)

- Design:
    - First page: Search bar
        - White background with black “What do you want to see today?” text in the middle
        - Below the text will be a search bar for query
            - Text placeholder: some suggestions for queries
    - Second page: Show top 3 recommendation based on query
        - Use chat GPT to filter for relevant information based on the query
        - Allow to click onto poster for more information
            - Poster on the left
            - Detail information on the right
        - Allow refresh buttons to generate new recommendations after feedback
        - “Explore more” button at the bottom with arrow pointing downwards
    - Third page: Catalog of movies
        - Allow filter based on different criterias: actors, directors, genre, ratings
        - 2 main tabs: catalog, history
- User Flow: Type into the query -> Go through the top 3 recommendations (potential action: watched already, favorite, dislike) -> Click more to explore the catalog -> Update recommendation page based on their behavior -> Track and show browsing history in history tab
- Caching:Reduce the load of the database by storing current session cache

Team responsibilities

Tony: Backend (database and admin interface)
Pan: Front end (webpage design and recommendation engine)

Timeline

Milestone	Wk1 (2 Feb)	Wk2 (9 Feb)	Wk3 (16 Feb)	Wk4 (23 Feb)	Wk5 (1 Mar)	Wk6 (8 Mar)	Wk7 (15 Mar)	Wk8 (22 Mar)	Wk9 (29 Mar)	Wk10 (5 Apr)	Wk11 (12 Apr)	Wk12 (19 Apr)	Wk13 (26 Apr)
Project Proposal													
Design													
Database													
Backend Interface													
Frontend Interface													
Database													
Implementation													
Testing													
DB Replication													
Backend (Data Managers)													
Implementation													
Testing													
Frontend (End-Users)													
Implementation													
Testing													
Recommendation engine													
Final Delivery													
Full Demo													
Final Report													