

## **Project Name:** Multi-Domain Recommender System

### **Objective:**

- Build a recommender system that suggests **Movies, Music, and Books** to users based on their preferences, search history, and user interactions.
- Provide a **centralized dashboard** for a multi-domain personalized experience.
- Use **Firebase Authentication** for secure login/sign-up and **Firestore** for storing user history.

### **Key Features:**

- User authentication (Sign Up, Login, Logout).
- Multi-domain recommendations: Movies (via TMDB), Music (via Spotify), Books (via collaborative filtering).
- Stores **user history** in Firestore to improve personalization.
- Streamlit dashboard with a modern interactive UI.

## **2 Methodology & Workflow**

### **Step 1: User Authentication**

- **Firebase Admin SDK** is used to:
  - Register users in Firebase Authentication.
  - Store user details and search history in Firestore.
- **Login Process:**
  - Users enter email/password.
  - Firebase REST API (`signInWithPassword`) validates credentials.
  - Successful login retrieves a user ID.
- **History Tracking:**
  - For each recommendation query, user search history is appended in Firestore for future personalization.

### **Step 2: Domain-Specific Recommendation**

#### **A. Movie Recommendations**

- **API Used:** TMDB (The Movie Database API).
- **Workflow:**
  1. User inputs a movie name.
  2. Call TMDB search API to get movie ID.
  3. Use TMDB recommendations API for similar movies.
  4. Filter out movies the user has already watched (from Firestore).
  5. Return movie title, poster URL, and link for display.
- **Technical Context:**
  1. Requests library handles REST API calls.
  2. User history stored in Firestore ensures personalized recommendations.
  3. Error handling for connection errors and empty results.

## B. Music Recommendations

- **API Used:** Spotify API (via `spotipy` Python library).
- **Workflow:**
  1. User inputs a genre or music type.
  2. Spotify API searches top tracks for the genre.
  3. Extract track name, artist, album cover, and Spotify URL.
  4. Display top 10 recommendations.
- **Technical Context:**
  1. `SpotifyClientCredentials` used for OAuth-based API authentication.
  2. JSON responses parsed for track metadata.
  3. Album cover URLs and external Spotify links enhance UI.

## C. Book Recommendations

- **Approach:** Collaborative Filtering using **K-Nearest Neighbors**.
- **Data Used:**
  - `model.pkl` → Trained KNN model.

- `book_pivot.pkl` → Pivot table of users × books with ratings.
- `final_rating.pkl` → Book metadata including `img_url` for poster display.
- `books_name.pkl` → List of all book titles.
- **Workflow:**
  - User selects a book.
  - KNN model finds **top 5-6 similar books** based on pivot table similarity.
  - Poster URLs fetched from `final_rating`.
  - Average rating computed using NaN-aware calculations.
- **Technical Context:**
  - `numpy` used for similarity calculations and handling missing ratings (`np.nan`).
  - `pickle` loads pre-trained KNN model and pivot tables.
  - The system avoids recommending books the user already interacted with using stored history.

### Step 3: Frontend / Dashboard

- **Framework:** Streamlit.
- **Features:**
  - Sidebar menu for Login/Sign Up.
  - Domain selection radio buttons.
  - Conditional display of movie/music/book recommendations.
  - User search history displayed dynamically.
  - Interactive recommendation cards with hover effect via **custom CSS**.

#### Technical Context:

- `st.session_state` tracks authentication state and `user_id`.
- `st.selectbox` and `st.radio` allow dynamic domain selection.
- Recommendations are rendered with HTML inside `st.markdown` for better UI styling.

## **3** Technical Stack

Component	Technology / Tool	Purpose
Backend	Python 3.12	Main programming language
Web framework	Streamlit	Dashboard and UI
Authentication	Firebase Auth + REST API	Secure user login/signup
Database	Firestore	User details & search history storage
Movie API	TMDB API	Fetch movie recommendations & metadata
Music API	Spotify API (spotipy)	Fetch music recommendations
Books	KNN model + Pickle files	Collaborative filtering recommendations
Data Processing	Numpy, Pandas	Matrix manipulations & mean calculations
UI Styling	Streamlit + Custom CSS	Interactive recommendation display

## 4 Technical Concepts You Can Explain in Interviews

### 1. Recommender Systems Concepts:

- Collaborative Filtering: recommending items based on user similarity or item similarity.
- User history filtering: avoiding redundant recommendations.
- Use of KNN for book recommendations.

### 2. APIs & REST Integration:

- TMDB API for movies.
- Spotify API using `spotipy` with OAuth client credentials.
- Firebase Authentication REST API for login.

### 3. Data Handling:

- Pivot tables for user-item ratings.
- Handling missing values with `np.nan`.
- Preprocessing and serialization using `pickle`.

### 4. Streamlit / UI:

- `st.session_state` for managing user session.
- Dynamic rendering of HTML + CSS in `st.markdown`.
- Efficient use of conditional rendering to avoid unnecessary API calls.

## 5. **Firestore Admin SDK:**

- Initializing Firebase app with credentials.
- Firestore CRUD operations:
  - `.collection("users").document(user_id).set(...)`
  - `.update()` to append user history.

## 6. **Error Handling / Optimization:**

- Exception handling for API failures.
- Filtering recommendations already seen by the user.
- Delayed loading avoided by lazy evaluation (fetching data only after button click).

# 5 **Methodology Summary**

## 1. **Requirement Gathering:** Need a multi-domain recommendation platform.

## 2. **Data Collection & Preprocessing:**

- TMDB movie metadata.
- Spotify music metadata.
- Book ratings data and pivot table.

## 3. **Modeling:**

- Train KNN for books (offline), save as `model.pkl`.

## 4. **Implementation:**

- Connect APIs with Python.
- Build user login/signup system with Firebase.
- Streamlit dashboard to unify multiple recommendation domains.

## 5. **Testing & Optimization:**

- Verify API calls, validate recommendations, optimize UI rendering.