Popcorn-Play: Project Architecture & Flow

1. Introduction

This document outlines the complete architecture, features, and data flows for the Popcorn-Play mobile application. The project is a movie and TV show discovery app built with **React Native (Expo)**. It leverages the **TMDb API** for all media-related content and **Appwrite** as its backend-as-a-service (BaaS) for user authentication and data storage.

2. Core Features & Functionality

2.1. User-Facing Features

- **User Authentication:** Secure sign-up, login, and logout functionality handled by Appwrite.
- **Content Discovery:**
  - Browse trending, popular, and top-rated movies and TV shows.
  - Powerful search functionality for specific titles.
  - Infinite scrolling and pagination for seamless browsing.
- **Detailed Information:**
  - Detailed screens for movies and TV shows featuring synopsis, cast, genres, release information, and ratings.
  - View profiles of cast and crew members.
  - Information on where to stream the content (streaming providers).
- **Media Playback:** In-app YouTube trailer playback.
- **Personalization:**
  - Ability for users to save their favorite movies and TV shows to a personal list.
  - A dedicated "Saved" or "Favorites" section to view stored items.
- **User Experience:**
  - A responsive and polished UI.
  - Shimmer/skeleton loaders to provide visual feedback during data fetching.

2.2. Backend Services

- **Movie & TV Show Data:** All media information, including metadata, images, and trailers, is sourced from the **The Movie Database (TMDb) API**.
- **User Management & Database:** User accounts and user-specific data (like favorite movies) are managed and stored using **Appwrite**.
- **API Communication:** The **Axios** library is used for making HTTP requests to both the TMDb API and the Appwrite backend.

3. System Architecture

3.1. Frontend (Client-Side) Architecture

- **Framework: React Native (Expo)** for cross-platform (iOS/Android) mobile app development.
- **Navigation: React Navigation** for managing screen transitions and the overall navigation stack (e.g., tabs and stacks).
- **State Management:** (Implicit) React's built-in state management (useState, useEffect, useContext) is likely used for managing component-level and shared state.
- **UI Components:** Custom-built components, potentially with a library like **Shimmer Placeholders** for loading states.

3.2. Backend (Server-Side) Architecture

- This project uses a **BaaS (Backend-as-a-Service)** model, meaning there is no custom-built server.
- **Appwrite:**
  - **Auth Service:** Manages all user authentication (creating accounts, sessions, etc.).
  - **Database Service:** Stores user-specific data, such as a list of saved movie/TV show IDs.
- **TMDb API:**
  - A third-party REST API that serves as the primary source for all movie and TV show data.

4. User & Data Flows

4.1. User Authentication Flow (with Appwrite)

1. **Sign Up:**
   - User enters their email and password on the sign-up screen.
   - The app calls the account.create() method from the Appwrite SDK.
   - Appwrite creates a new user in the project's user pool and returns a user object.
   - The app then likely calls account.createEmailSession() to log the user in immediately.
2. **Login:**
   - User enters their credentials on the login screen.
   - The app calls account.createEmailSession() with the user's credentials.
   - Appwrite validates the credentials and, if successful, returns a session object. The SDK handles storing this session.
3. **Authenticated Requests:**
   - For actions like saving a movie, the Appwrite SDK automatically includes the user's active session information in the request header, allowing Appwrite to verify the user's identity.

4.2. Movie Discovery Flow (with TMDb)

1. A user opens the app or navigates to a screen like "Trending Movies."
2. The app uses **Axios** to make a GET request to the relevant TMDb API endpoint (e.g., /trending/movie/week). The TMDb API Key is included in the request.
3. The TMDb API returns a JSON object containing a list of movies, including their IDs, titles, poster paths, etc.
4. The React Native app parses the JSON response and uses it to render a list of movie cards on the screen.
5. When a user clicks on a movie, the app uses the movie's ID to make another API call to a different TMDb endpoint (e.g., /movie/{movie_id}) to fetch detailed information for the movie details screen.

4.3. Saving a Favorite Movie Flow (with Appwrite)

1. An authenticated user is on the details screen for a movie and clicks the "Save" or "Favorite" button.
2. The app triggers a function that calls the Appwrite SDK's databases.createDocument() method.
3. This function sends the user's ID (obtained from the current session) and the movie's ID (from TMDb) to a specific collection in the Appwrite database (e.g., "Favorites").
4. Appwrite creates a new document in the collection, effectively linking the user to that movie ID.
5. To display the list of saved movies, the app calls databases.listDocuments(), querying the "Favorites" collection for all documents matching the current user's ID.

5. Backend Schema (Appwrite Collections)

A likely database structure within Appwrite would be:

Database: PopcornPlayDB

Collection: Favorites

- **Purpose:** To store the relationship between users and their saved movies/TV shows.
- **Attributes (Fields):**
    - userId (String, Required): Stores the ID of the user from Appwrite's authentication system. This would be the primary field for querying.
    - mediaId (String, Required): The ID of the movie or TV show from TMDb.
    - mediaType (String, Required): A field to distinguish between 'movie' and 'tv'.
    - posterPath (String, Optional): The path to the poster image, stored for easier retrieval without an extra API call.
    - title (String, Optional): The title of the media.