PROJECT PROPOSAL

Letterboxd Clone - A Social Movie Tracking Database System

Group Members:
Ibrahim Johar Farooqi (23K-0074)
Syed Ukkashah Ahmed (23K-0055)

1. Introduction and Brief Description

The Letterboxd Clone project aims to develop a database-driven social platform inspired by Letterboxd, a popular website for film enthusiasts. This system will allow users to track movies they've watched, rate and review films, maintain personal watchlists, and engage socially by following other users. The platform will emphasise data management and user interactions, making it a functional web application suitable for a database course project.

The core idea is to create a "management system" for movie-related data, where users can log their viewing history, share opinions, and discover content through community features. This project addresses real-world scenarios like inventory control (managing a movie catalogue), ordering (users adding movies to watchlists or submitting reviews), and billing (analogous to assigning ratings or follows as "value" metrics). It will be built with a focus on relational database design, ensuring scalability for up to thousands of users and movies. The system will be implemented using PostgreSQL as the database, with a back-end API and front-end interface to demonstrate full functionality.

This project is not a "toy" system; it involves complex relationships (e.g., many-to-many user follows and movie genres) and operational features, making it educational for database concepts like normalisation, queries, and transactions.

2. What Functions Should the System Perform?

The system will perform a variety of operational and analytical functions, aligned with course examples like inventory control, billing, and ordering. These functions ensure the platform is interactive and data-driven:

- **Inventory Control Analogue:** Manage a centralised movie catalogue, allowing admins or users to add/view movie details (e.g., title, director, genres). This acts as a "stock" of films that users can interact with.
- **Ordering Analogue:** Enable users to add movies to personal watchlists (like placing an order), submit reviews, and assign ratings. This includes logging watched movies and tracking user activity
- **User Management:** Register/login users, update profiles, and manage follows for social networking.
- **Reporting and Analytics:** Generate insights like top-rated movies, user review counts, or genre-based recommendations.

These functions make the system operational, supporting multiple user roles (e.g., regular user for personal tracking, admin for moderation).

3. Front-End and Back-End Technologies

- **Back-End:** PostgreSQL for the database (handling schema, queries, views, and transactions). Node.js with Express.js framework for the API server, using Sequelize ORM for database interactions (with raw SQL for advanced queries to meet course requirements). This setup allows secure handling of CRUD operations and user input.
- **Front-End:** HTML/CSS/JS Frameworks

4. Core Modules

The system will be divided into modular components for better organisation: (might add/modify modules depending on the project's development)

1. **User Module:** Handles authentication, profiles, and social features (e.g., following users).
2. **Movie Module:** Manages the film database, including search and details viewing.
3. **Review and Rating Module:** Allows users to post reviews/ratings and view others'.

4. **Watchlist Module:** User-specific lists for planned viewings.
5. **Analytics Module:** Generates reports using database queries (e.g., top movies by genre).
6. **Admin Module:** Basic moderation tools (e.g., delete inappropriate reviews).

These modules interact via the database, ensuring data consistency.

5. Possible Functionalities

Beyond core features, the following enhancements could be added (prioritized based on time):

- **Basic:** User registration/login, movie search, add/review/rate movies, manage watchlists, follow users.
- **Advanced:** Genre-based recommendations (using aggregates like AVG ratings), activity feeds (showing followed users' reviews), search with filters (e.g., by year or director).
- **Stretch Goals:** Email notifications for follows (via external API if needed), user statistics dashboards, or integration with external movie data sources like TMDB API for auto-populating movie details.
- **Limitations:** No real-time chat or mobile app; focus on web-based CRUD and queries to align with course scope.

6. Entity Relationship Design

6.1 Description of Entities

The system includes approximately 5 core entities to ensure complexity without being a "toy" system. These entities represent key concepts in a movie-tracking platform and expand to 8 tables through normalisation (including junction tables for many-to-many relationships). Each entity is described below:

1. **User:** Represents platform users, storing profile information. (Table: User)
2. **Movie:** Represents films in the catalogue, with details like title and director. (Table: Movie)

3. **Review:** Captures user opinions on movies. (Table: Review)
4. **Rating:** Stores numerical scores for movies (separate from reviews for flexibility). (Table: Rating)
5. **Watchlist:** Tracks movies users plan to watch. (Table: Watchlist)

Supporting entities/tables:

- **Follower:** Handles user follows (many-to-many).
- **Genre:** Categorises movies.
- **MovieGenre:** Links movies to genres (junction table).

This setup provides relational depth, with at least 7-8 tables total.

6.2 Tables and Attributes

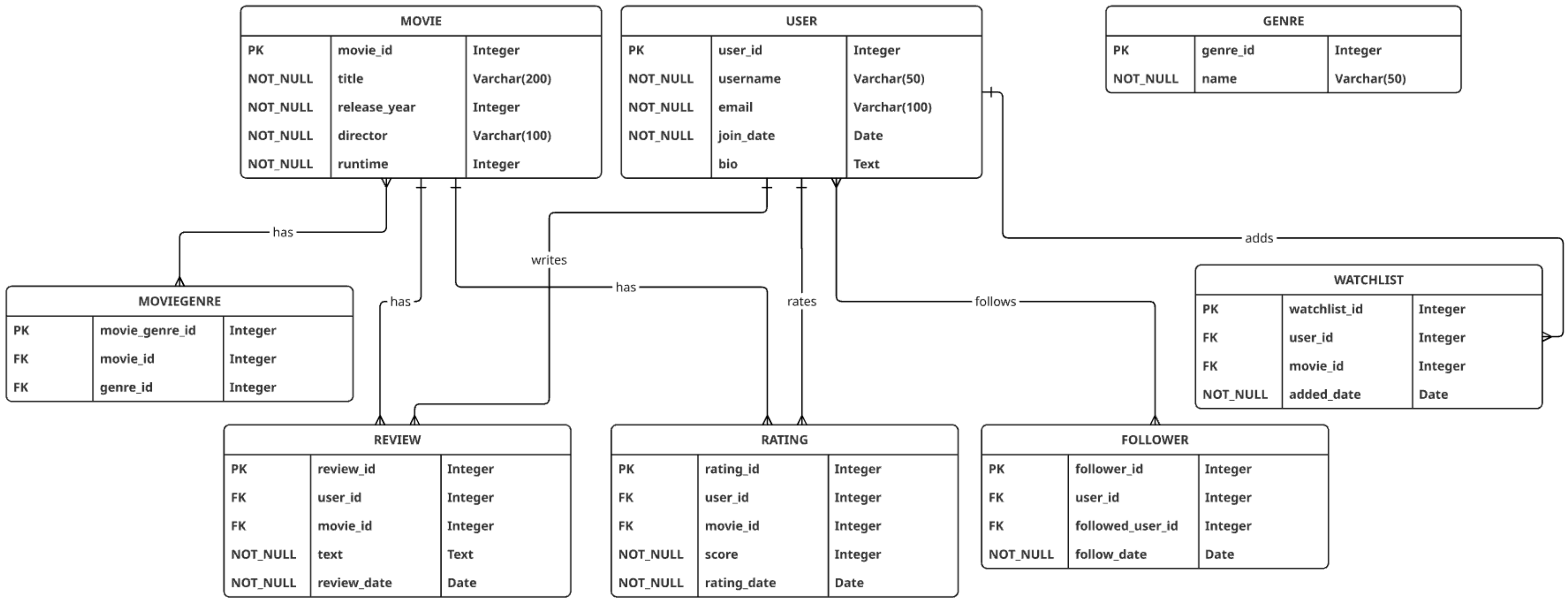
Below is a table summarising the 8 tables, their attributes, and primary/foreign keys (might add more tables/modify existing tables depending on the project's development):

Table Name	Purpose	Attributes	Primary Key	Foreign Keys
User	User profiles	user_id (SERIAL), username (VARCHAR(50) UNIQUE), email (VARCHAR(100) UNIQUE), join_date (DATE), bio (TEXT)	user_id	None
Movie	Movie catalog	movie_id (SERIAL), title (VARCHAR(100)), release_year	movie_id	None

Table Name	Purpose	Attributes	Primary Key	Foreign Keys
		(INT), director (VARCHAR(50)), runtime (INT)		
Review	User reviews	review_id (SERIAL), user_id (INT), movie_id (INT), text (TEXT), review_date (TIMESTAMP)	review_id	user_id (User), movie_id (Movie)
Rating	User ratings	rating_id (SERIAL), user_id (INT), movie_id (INT), score (DECIMAL(2,1)), rating_date (TIMESTAMP)	rating_id	user_id (User), movie_id (Movie)
Watchlist	Planned movies	watchlist_id (SERIAL), user_id (INT), movie_id (INT), added_date (TIMESTAMP)	watchlist_id	user_id (User), movie_id (Movie)
Follower	User follows	follower_id (SERIAL), user_id (INT), followed_user_id (INT),	follower_id	user_id (User), followed_user_id (User)

Table Name	Purpose	Attributes	Primary Key	Foreign Keys
		follow_date (TIMESTAMP)		
Genre	Movie genres	genre_id (SERIAL), name (VARCHAR(50) UNIQUE)	genre_id	None
MovieGenre	Movie-genre links	movie_genre_id (SERIAL), movie_id (INT), genre_id (INT)	movie_genre_id	movie_id (Movie), genre_id (Genre)

6.3 E-R Diagram



MOVIE

PK	movie_id	Integer
NOT_NULL	title	Varchar(200)
NOT_NULL	release_year	Integer
NOT_NULL	director	Varchar(100)
NOT_NULL	runtime	Integer

USER

PK	user_id	Integer
NOT_NULL	username	Varchar(50)
NOT_NULL	email	Varchar(100)
NOT_NULL	join_date	Date
	bio	Text

GENRE

PK	genre_id	Integer
NOT_NULL	name	Varchar(50)

MOVIEGENRE

PK	movie_genre_id	Integer
FK	movie_id	Integer
FK	genre_id	Integer

WATCHLIST

PK	watchlist_id	Integer
FK	user_id	Integer
FK	movie_id	Integer
NOT_NULL	added_date	Date

REVIEW

PK	review_id	Integer
FK	user_id	Integer
FK	movie_id	Integer
NOT_NULL	text	Text
NOT_NULL	review_date	Date

RATING

PK	rating_id	Integer
FK	user_id	Integer
FK	movie_id	Integer
NOT_NULL	score	Integer
NOT_NULL	rating_date	Date

FOLLOWER

PK	follower_id	Integer
FK	user_id	Integer
FK	followed_user_id	Integer
NOT_NULL	follow_date	Date

7. Implementation Phase: Database Features to Implement

Tables and Views: Create 8 tables (User, Movie, Review, Rating, Watchlist, Follower, Genre, MovieGenre) with constraints (primary keys, foreign keys, UNIQUE). Define views for reusable queries, such as top-rated movies or user activity summaries.

Queries:

- **Per User/Role:** One query per role (e.g., regular user: view watchlist; admin: count reviews per user).
- **Joins:** Multi-table queries to combine data (e.g., users, movies, reviews).
- **Advanced SQL:** Subqueries (e.g., users with above-average reviews), window functions (e.g., ranking movies by rating), triggers (e.g., auto-update average ratings), and PL/pgSQL procedures (e.g., for adding reviews).
- **User Input:** Parameterised queries/procedures to accept inputs (e.g., user ID for watchlist retrieval).

CRUD Operations:

- **Insert:** Add new users, movies, reviews, ratings, or watchlist entries.
- **Update:** Modify ratings, user profiles, or review text.
- **Delete:** Remove watchlist entries or reviews.

Data Population: Import mock data or use a small external dataset (e.g., TMDB CSV) for ~100 movies, 50 users.

These features ensure a functional, interactive system with robust database operations, aligned with course goals.

8. Conclusion

This Letterboxd Clone project provides a comprehensive database system that meets course objectives, demonstrating design rigour and implementation skills. With the proposed tech stack and features, it can be developed by two people in one week, focusing on PostgreSQL for the database core. Future expansions could include mobile support, but the current scope ensures feasibility and educational value.