**Progress Report: CineVault - Movie Database**

**Team 6 :**

Chukwunenye Uwaeme: **Frontend Engineer**

Tahsym Richburg: **Backend Engineer**

Demetrius Philbert: **Documentation Specialist**

Dominique Paige: **Documentation Specialist**

### **1. Introduction**

CineVault is a movie database designed to help users discover and explore films in a fun, easy, and interactive way. With a user-friendly interface, CineVault makes it simple to search for movies by genre, cast, release dates.

With so many movies out there, it can feel overwhelming to find something you’ll love. CineVault takes the stress out of the process by offering a platform that goes beyond basic details, helping you uncover hidden gems, learn more about your favorite films, and connect with other movie lovers through reviews and recommendations.

CineVault isn’t just about finding movies it’s about creating a space where people can share their opinions, recommend favorites, and build a community around a shared love of cinema, all through an intuitive and engaging experience.

### **2. Design**

CineVault is built around a solid and carefully planned relational database to ensure the application is fast, reliable, and easy to maintain. Our design approach focuses on keeping things organized, making it easy to add new features, and ensuring the system can grow as more users and data are added.

At the heart of our database is an **Entity-Relationship Diagram (ERD)**, which maps out the key pieces of data we store, like movies, actors, and users, and shows how they’re all connected.

Here’s how it’s set up:

**Entity-Relationship Diagram (ER)**

The Entity-Relationship Diagram (ERD) is the structural framework for CineVault’s database. It defines the key entities (categories of data), their attributes (specific details about the data), and the relationships that link them together.

**Entities and Attributes:**

**Movie(Stores core details about each film):**

Includes movie\_id, title, release\_date, duration, rating, and language.

**Genres(Categorizes movies by type):**

Includes genre\_id, genre\_name, and description.

**Reviews(Captures user opinions on movies):**

Includes review\_id, rating, content, user\_id, movie\_id, and review\_date.

**Users(Manages information about CineVault’s users):**

Includes user\_id, username, email, password, and is\_admin.

**Movie\_Genres:** watchlist\_id, user\_id, movie\_id and date\_added

**Watchlist:** user\_id, movie\_id, and watchlist\_id

**Relationships Between Entities:**

**Movies to Genres:** Many-to-Many

A movie can belong to multiple genres, and each genre can include numerous movies.

**Movies to Reviews:** One-to-Many.

A movie can have multiple user reviews, but each review corresponds to one movie.

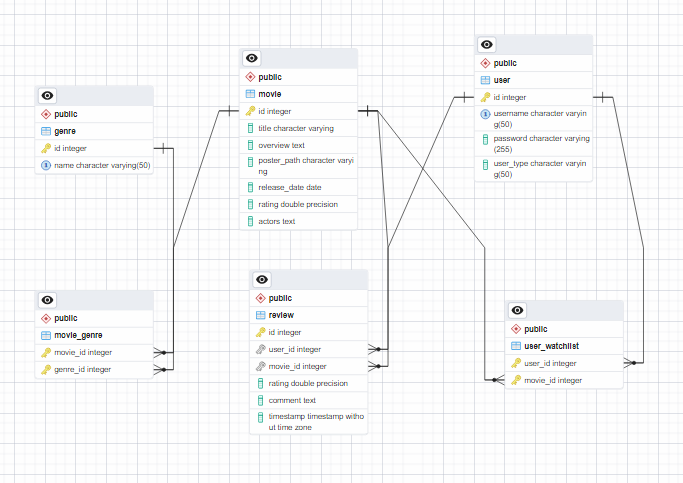
**Users to Reviews:** One-to-Many.

A user can write multiple reviews, but each review is tied to a single user.

**Users to Watch List:** Many-to-Many

A direct relationship between **Users** and **Movies** is inefficient, as both can have multiple associations with each other.

This ERD design ensures that CineVault’s database is optimized for quick searches, accurate data retrieval, and easy expansion to support new features or entities.



**UML Diagram**

The UML diagram represents the system's classes, their attributes, methods, and relationships illustrates the system's structure, defining the main classes within the CineVault application. It outlines the attributes (data members) and methods (functions) of each class, as well as the relationships between them.

**Classes and their Components:**

**Movie:** Handles movie-related functionalities like adding reviews, actors, genres, and awards.

**Genre:** Organizes genres and fetches movies by genre.

**Watchlist:** Manages users movie watchlists

**Review:** Manages individual reviews, including ratings and content.

**User:** Handles user accounts, login, and registration.

**Methods:**

**MovieController:** Methods like getMovieById, addMovie, updateMovie, and deleteMovie.

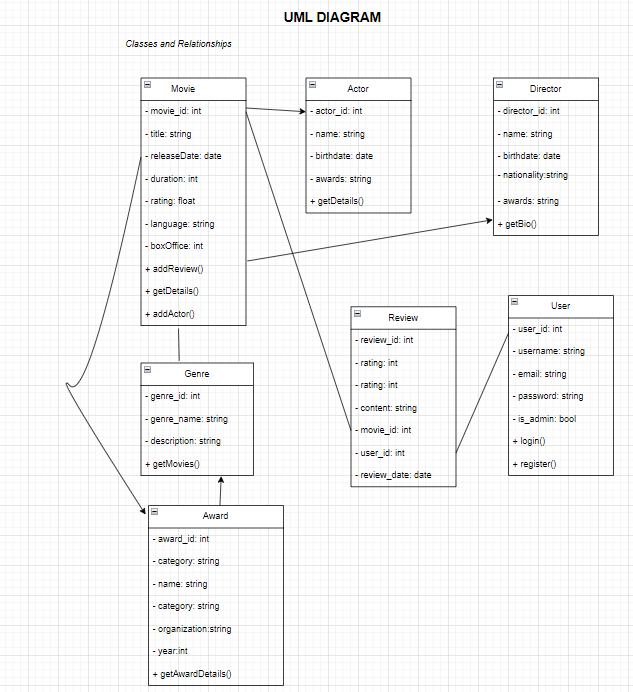
**ReviewController:** Methods to manage reviews, such as addReview and getReviewsForMovie.

**UserController:** Handles user management with methods like createUser and getUserProfile.

**WatchlistController:** Includes addToWatchlist, getWatchlistByUser

**Relationships:**

* Movie connects with Genre and Watchlist with one-to-many or many-to-many relationships.
* User connects to Review (one-to-many).
* Database Schema
* The database schema is designed with normalized tables to reduce redundancy and improve data integrity.



**Key Foreign Key Constraints and Tables:**

**Movies:** movie\_id (PK).

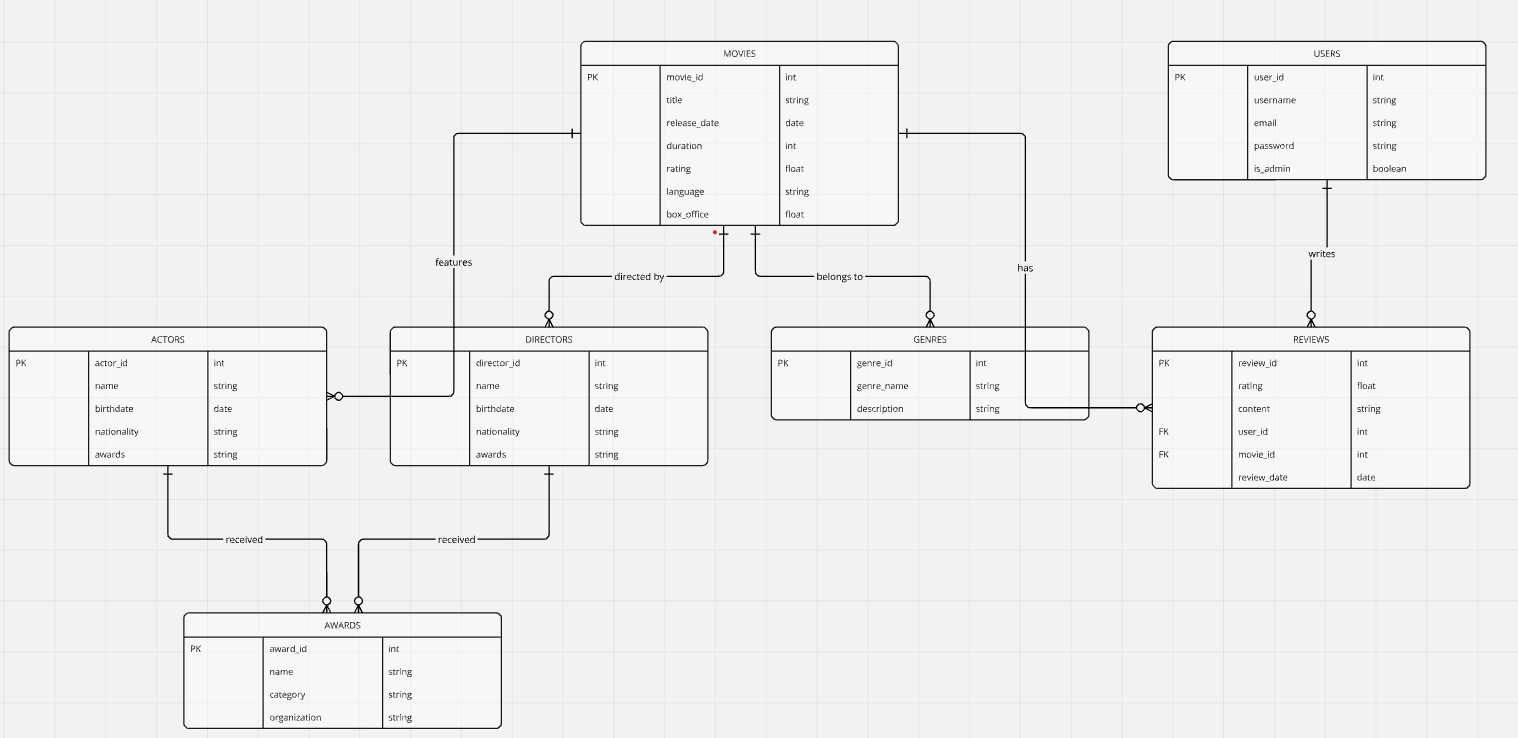
**Genres:** genre\_id (PK).

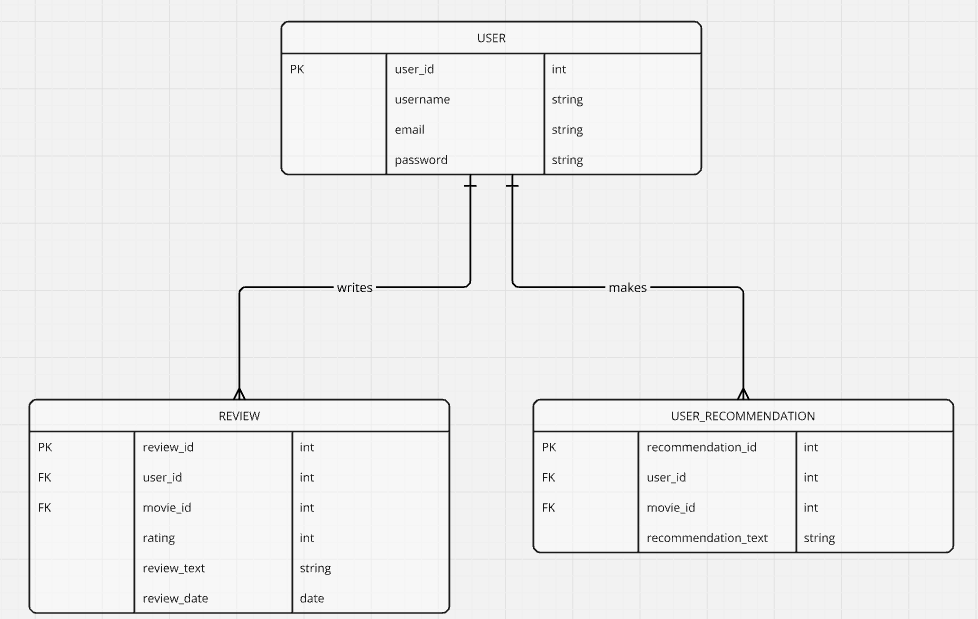
**Users:** user\_id (PK).

**Reviews:** review\_id (PK), foreign keys: user\_id, movie\_id.

**Watchlist:** watchlist\_id(PK), foreign keys user\_id, movie\_id

Junction tables like MoviesGenres, MoviesReview, and MoviesWatchlist for many-to-many relationships.





### **3. Implementation**

**Frontend Implementation**

We implemented the frontend using Next.js 15, leveraging Shadcn and Tailwind CSS for a clean, responsive, and customizable UI. The project features a range of reusable components, such as movie cards, input forms, and modals, designed to ensure seamless user interaction. The frontend communicates with the backend via API endpoints to manage movies, user profiles, reviews, and watchlists.

Additionally, we included an Admin Page accessible only to users with administrative privileges, allowing them to manage movie data, including adding, updating, and deleting movies. This page enhances the application's functionality by enabling secure and efficient administrative operations.

**Navigation and Route Handling:**

NextJS - React full stack framework that allows for easy page navigation and dynamic routing

**Styling and Page Design:**

Tailwind CSS - A utility first CSS library

**Component Library:**ShadCn - A customizable component library used to build interactive web applications

**Backend Implementation**

CineVault's backend was implemented by integrating several components:

**Backend Framework:**

Built using Flask API Routes, with a clear folder structure separating API logic, authentication, and database operations.

**Database Integration:**

PostgreSQL was used with a normalized schema to store movies, users, reviews, and watchlist data. Queries were executed via the pg Node.js library.

**Authentication:**

JWT-based authentication secured sessions, while NextAuth.js provided Google OAuth login. Role-based access control was implemented for admin and editor privileges.

**TMDB API Integration:**

External movie data was dynamically fetched from the TMDB API using environment variables for secure API key management and loaded into the database with a load script.

**Features:**

Endpoints handled fetching, updating, and deleting movies, managing reviews, and allowing users to maintain a watchlist.

**Testing & Deployment:**

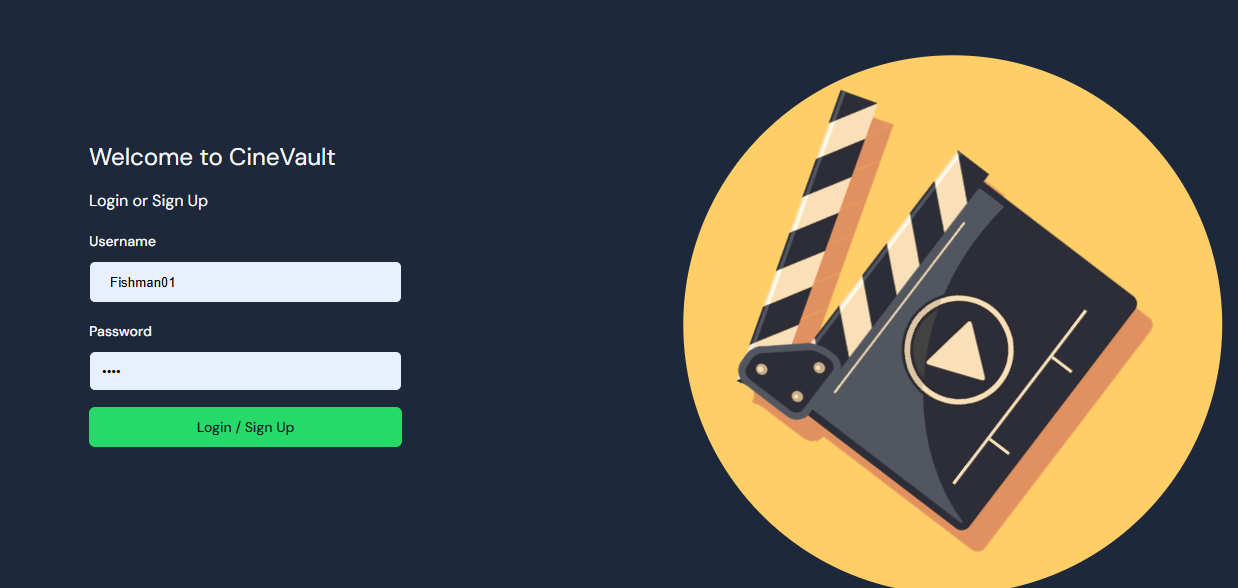
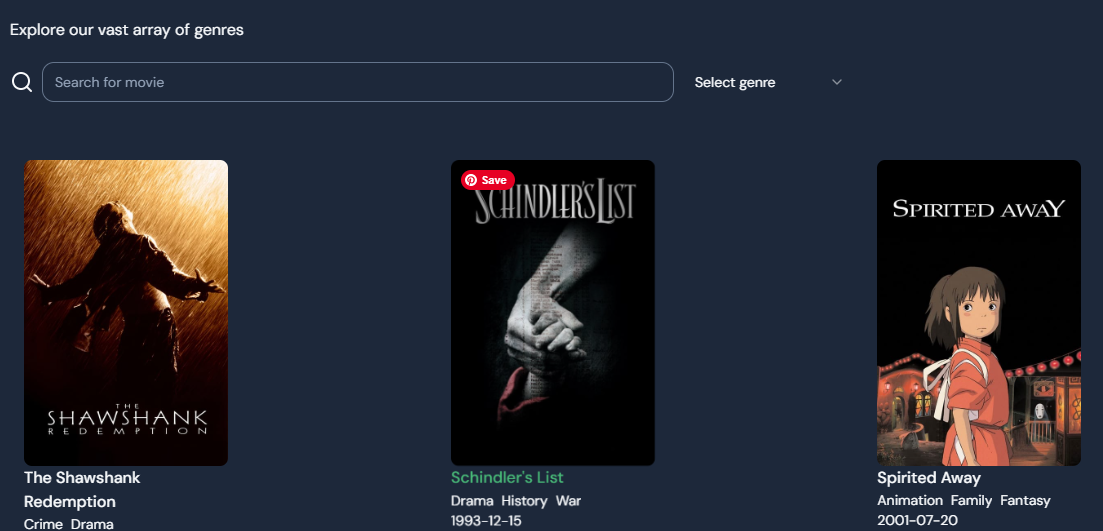
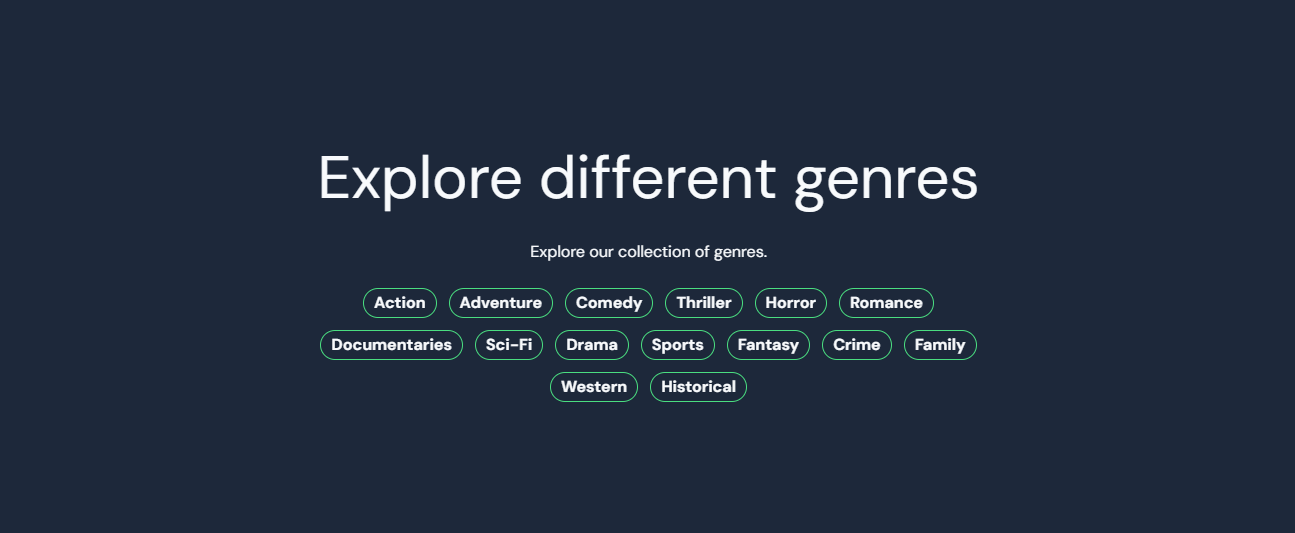
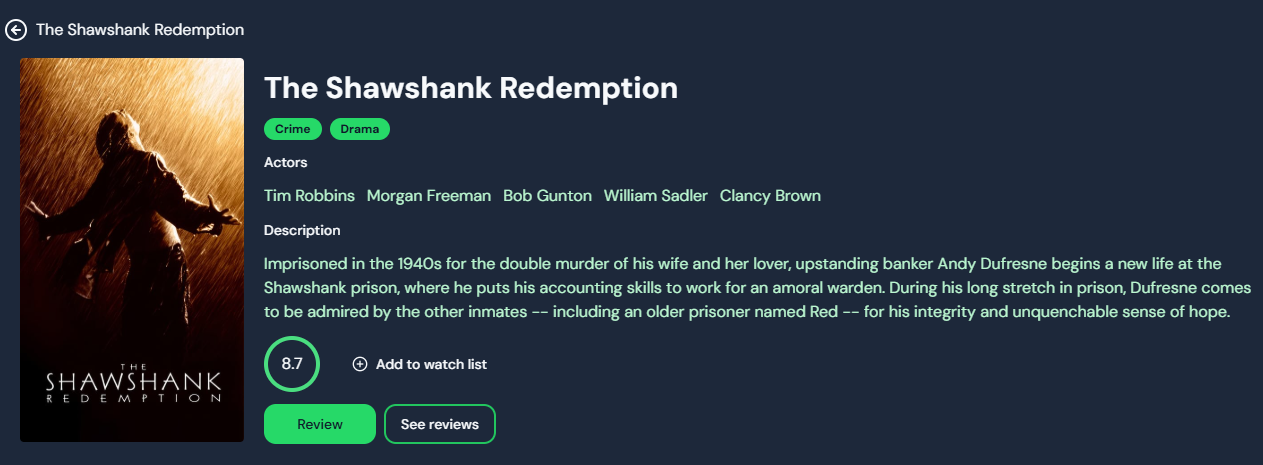
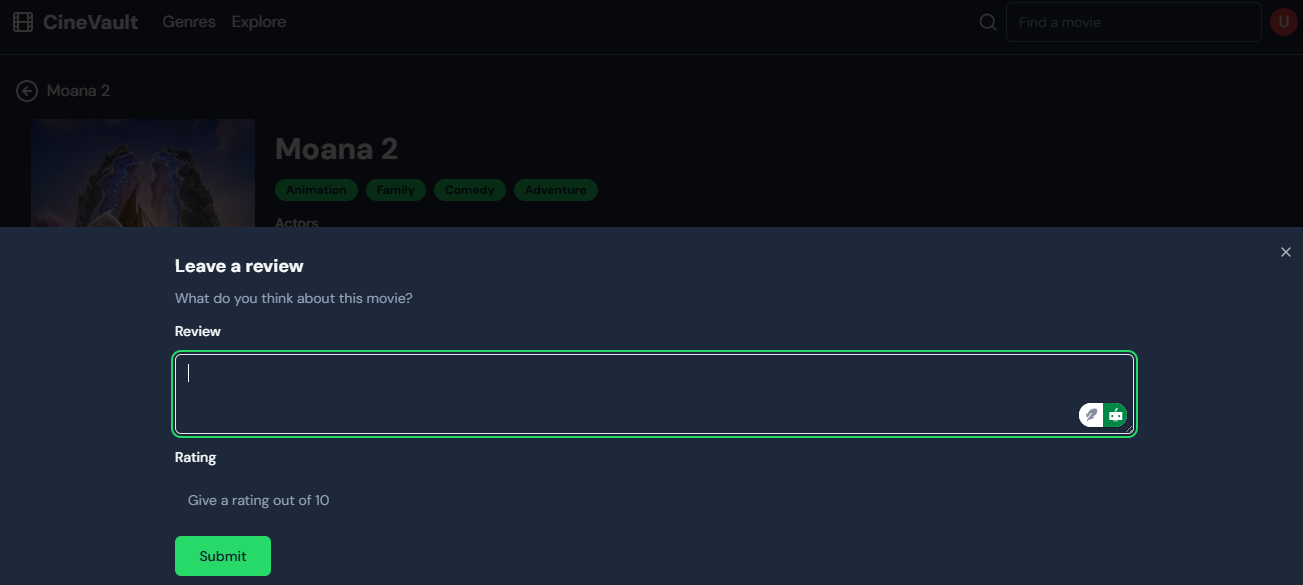
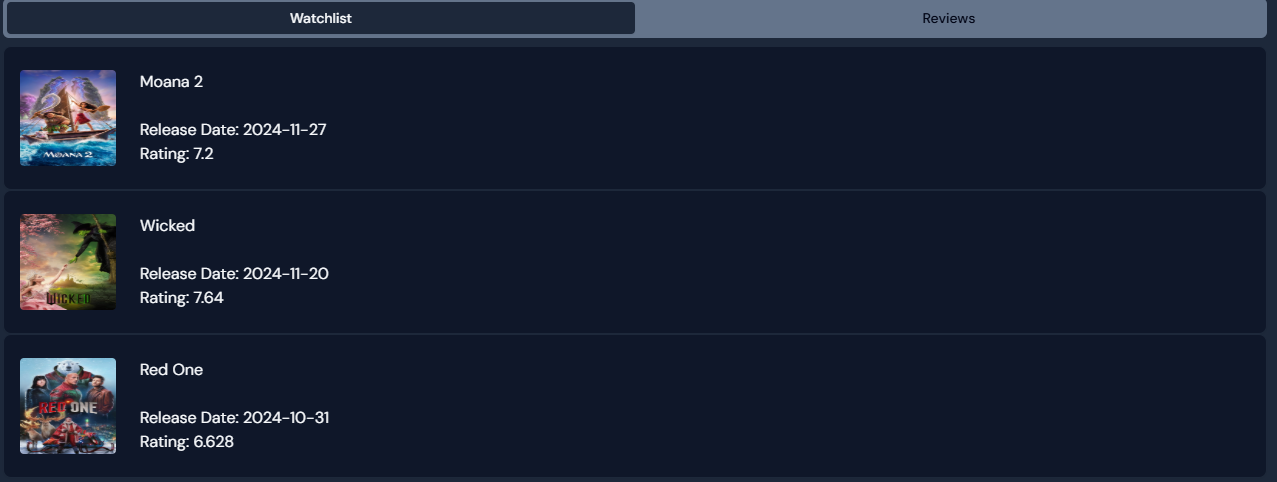
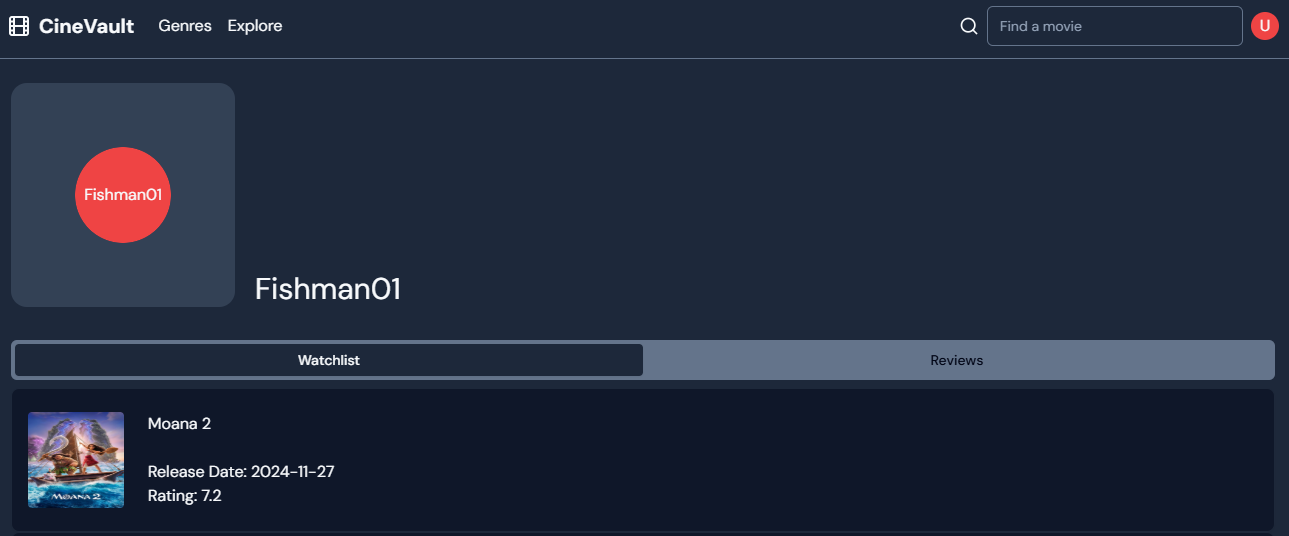
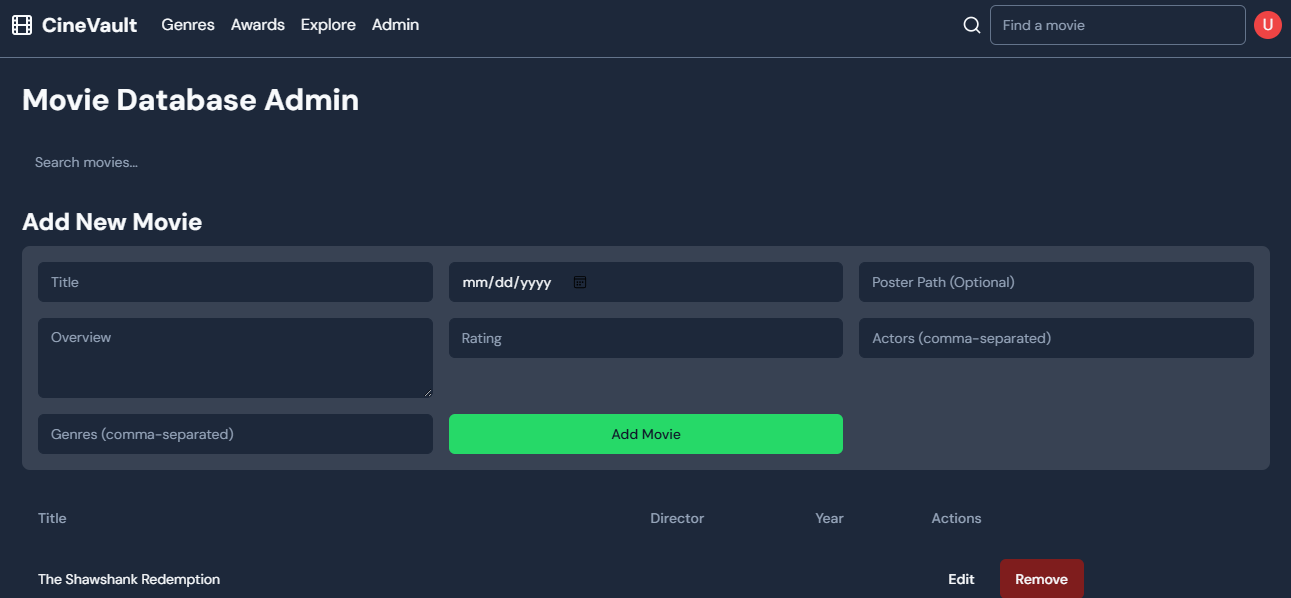
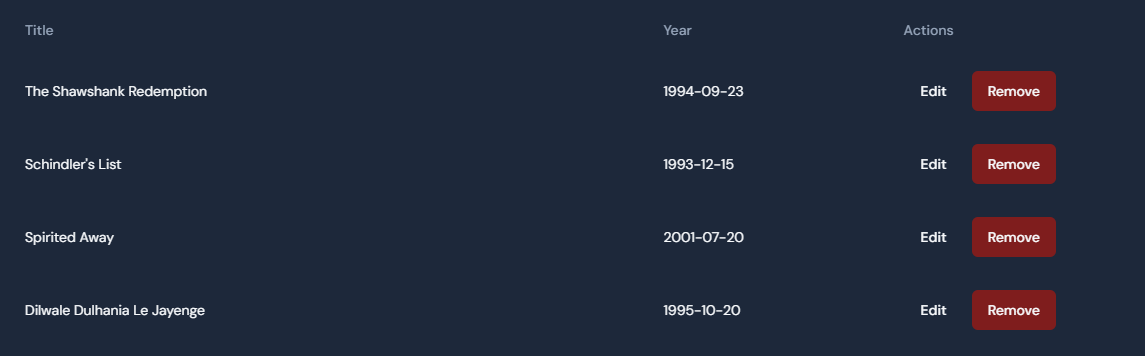
API and database queries were tested using Postman and pgAdmin. The backend was deployed on a cloud platform with secure environment configurations.

### **4. Functionality**

**Frontend Functionality**

CineVaults frontend is powered by NextJS 15 which is a react full stack frame works that allows for easy opage navigation and data routing.

**Features:**

1. User Authentication: Users can log in or register securely, with persistent session handling.  
   
2. Explore Page: Browse movies with options to search and filter by genre and name.  
   *  
   *
3. Movie Details: View detailed information about movies, including reviews, genres, and cast.  
   **
4. Reviews: Add and view user-generated reviews for each movie.  
   **
5. Watchlist Management: Add and remove movies from a personalized watchlist directly from the movie detail or user profile pages.  
   
6. Profile Page: Users can view their watchlist and reviews in a centralized profile section.  
   
7. Admin Page: Accessible only to admin users, this page provides tools to add, edit, or delete movies in the database.  
   *  
   *

**Backend Functionality**

CineVault's backend powers the movie catalog system by handling API requests, managing data interactions with the PostgreSQL database, authenticating users, and enabling role-based access control. The backend uses PostgreSQL to store data for users, movies, comments, and ratings

**Features -**

**Movies API:**

* Fetch all movies, trending movies, or movies by category/title.
* Add, update, and delete movie records (role-restricted).
* Manage ratings, comments, and reviews for movies.

**User Authentication:**

* JWT-based authentication and session management.
* Google OAuth for streamlined login.
* Role-based access control (Viewer, Editor, Admin).

**Watchlist Management:**

* Add, remove, and fetch movies from a user's watchlist.

**Integration with TMDB:**

* Fetch movie data dynamically from the TMDB API.

**Tech Stack:**

* Language: Pyhton
* Framework: Flask
* Database: PostgreSQL
* Authentication: Flask
* External APIs: TMDB API for movie data

**Folder Structure:**

Server/

├── app/

│ ├── \_\_init\_\_.py #initalized the connection between backend and database

│ ├── .env # has all the environment variables

│ ├── load\_movies.py # Loads the movies from the api into database

│ ├── models.py

│ ├── routes.py

├── migrations/

├── venv/

├── app.py #file that runs the backend

├── hash.py #file to generate a hashed password for an admin user

├── requirements.txt # contains all the libraries required for the backend

**Endpoints  
  
Authentication**

* POST /auth - Authenticate user (login/registration).

**Movies**

* GET /api/movies - Fetch all movies in the database.
* GET /api/genres/<string:genre\_name>/movies - Fetch movies by genre.
* GET /api/movies/recent - Fetch 10 most recently released movies.
* GET /api/movies/<int:movie\_id> - Fetch details of a specific movie.
* POST /api/movies/add - Add a new movie to the database (Admin only).
* PUT /api/movies/<int:movie\_id>/update - Update movie details (Admin only).
* DELETE /api/movies/<int:movie\_id>/delete - Remove a movie from the database (Admin only).

**Watchlist**

* GET /api/user/<int:user\_id>/watchlist - Fetch a user's watchlist.
* POST /api/user/<int:user\_id>/watchlist/add - Add a movie to the user's watchlist.
* POST /api/user/<int:user\_id>/watchlist/remove - Remove a movie from the user's watchlist.
* Reviews
* POST /api/movies/<int:movie\_id>/review - Add a review for a movie.
* GET /api/movies/<int:movie\_id>/reviews - Fetch reviews for a specific movie.

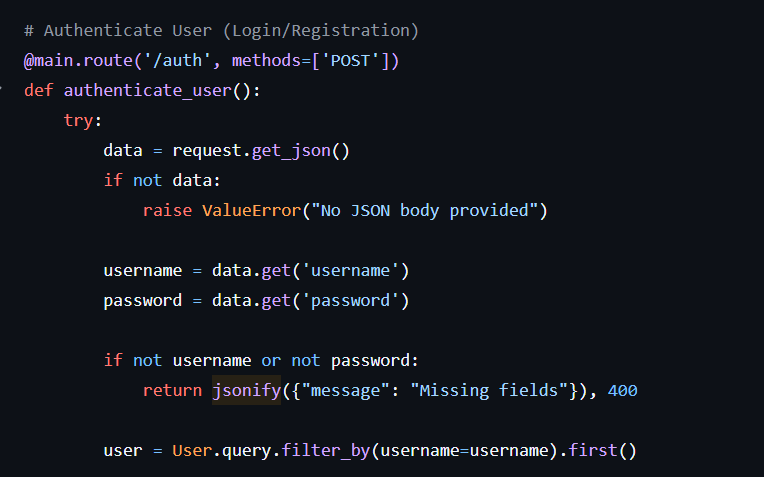
**Reviews**

* POST /api/movies/<int:movie\_id>/review - Add a review for a movie.
* GET /api/movies/<int:movie\_id>/reviews - Fetch reviews for a specific movie.

**TMDB Integration**

* GET /api/tmdb/getMovies - Fetch movies from the TMDB API.

***IMAGES ATTACHED ON NEXT PAGE***





### **5. User Manual**

1. Login / Register to create an account
2. Dashboard

* After logging in, you will see your personalized dashboard displaying:
  + **Your Watchlist**: Movies you’ve saved for later.
  + **Reviews**: A summary of the reviews you’ve written.

1. Explore Movies

* Use the navigation bar at the top to explore CineVault's movie database:
  + **Genres**: Browse movies by genre (e.g., Comedy, Drama, Action).
  + **Explore:** View trending movies or search for specific titles, actors, or directors.

1. Search Functionality

* Use the search bar in the Explore page to find movies by title, actor, genre, or director.
* Filter results using dropdown menus for genres, actors, or directors.

**Detailed Features -**

**Adding Movies to Watchlist:**

1. Navigate to a movie’s detail page.
2. Click the Add to Watchlist button.
3. Your watchlist will be updated on your dashboard.

**Viewing and Removing Movies from Watchlist:**

* On your dashboard, click the Watchlist tab.
* To remove a movie, click the "Remove" button next to the movie.

**Adding Reviews**

1. Navigate to a movie’s detail page.
2. Click Review to leave your thoughts.
3. Enter your review and provide a rating (out of 10), then click Submit.
4. Reviews will appear under the movie’s details and on your dashboard.

**Viewing Movie Details**

* Select any movie to see detailed information:
* Title, genre, cast, release date, rating, and description.
* User reviews and the option to add your own.

Some features, such as adding or deleting movie records, are restricted to admin accounts.

### **6. Conclusion**

The **CineVault** project has successfully created a rich, user-friendly platform for movie discovery, blending modern web technologies to offer an engaging and seamless experience. By using MySQL for a structured and efficient database, combined with Next.js for building the front end, we’ve developed a platform where users can explore movies in depth—covering everything from cast and genre to release dates and reviews.

With features like personalized watchlists, user-generated ratings, and movie reviews, CineVault adds a social layer to the movie discovery process. It allows users to track their favorite films and interact with the community by sharing opinions and recommendations. The secure authentication system and role-based access control ensure that user data is protected, giving both regular users and admins a secure and efficient experience.

This project has been an excellent opportunity to apply full-stack development principles, from the careful design of the database to the integration of backend systems and the creation of an intuitive front-end. Our team worked closely together, bringing technologies like TypeScript, Next.js, and third-party APIs to life in a way that results in a scalable, maintainable application.

Overall, CineVault is more than just a movie database—it's an interactive space where users can connect with movies and fellow movie lovers in a personalized way. This project has laid a strong foundation for future improvements and demonstrated our ability to tackle real-world software development challenges.

### **7. Implementation Video :** <https://youtu.be/KLKqdF5qh-Y>

### **8. User Manual Video:** <https://youtu.be/_GqiDfExW1Y>

### **9. Contribution of group members**

**Chukwunenye Uwaeme:** Designed and implemented the frontend for Cinevault.

* Landing Page- Created the landing page to advertise the features and offerings of the CineVault platform. The page provides an introduction to the website's purpose and encourages users to explore the movie database.
* File Structure Setup- Organized the frontend into a clean, modular file structure to maintain scalability and ease of development:
  + - Components: Contains reusable UI elements such as buttons, navigation bars, and containers, which help maintain consistency across the site.
    - Pages: Houses the actual pages of the website, each representing different views or features (e.g., home page, search results page).
    - Assets: Includes static files like logos, images, and branding materials necessary for the site's visual identity.
* Create Profile- Designed and implemented the user profile page where users can view and update their personal details, such as their username and email.
* Movie Search Functionality- Developed the movie search feature, enabling users to search for movies based on various criteria (e.g., title, genre). This allows for a personalized and efficient browsing experience.
* Leave Review / Rating- Implemented functionality for users to leave ratings and reviews on movies, contributing to the platform's community aspect.

**Tahsym Richburg :**

Designed and implemented the backend for Cinevault.

* Developed API routes for movie management, user authentication, watchlists, and reviews.
* Integrated PostgreSQL for persistent data storage, ensuring a well-structured and normalized database schema.
* Implemented secure user authentication using JWT and integrated Google OAuth for streamlined login.

Created the following diagrams to guide the development process:

* ER Schema Diagram: Outlined the database entities, attributes, and relationships.
* Referential Integrity Diagram: Detailed foreign key constraints to ensure data consistency.
* UML Diagram: Mapped out system classes, their methods, and the relationships between components.

Demetrius Philbert: Played a pivotal role in overseeing project documentation, ensuring that all deliverables and requirements were clearly articulated and readily accessible for team members and stakeholders.

**Dominique Paige :**

* Project Documentation Management- Played a central role in managing project documentation, ensuring that all deliverables, requirements, and updates were clearly articulated, organized, and accessible to all team members.
* Team Meetings and Scheduling- Assisted in organizing team meetings, setting clear agendas, and tracking milestones to ensure the group stayed on schedule and that all project phases were completed on time. As well as hosted Zoom sessions to facilitate remote team meetings and discussions, ensuring effective collaboration and communication throughout the project's development.
* Collaborated closely with Demetrius Philbert on compiling project updates and maintaining consistent communication within the team to ensure everyone was aligned on progress and next steps.
* Shared Drive and Document Storage- Created and managed a shared drive where all project-related documents were stored, allowing easy access and ensuring that important materials were well-organized for the group.