FINAL PROJECT REPORT

SEMESTER 3, ACADEMIC YEAR: 2024-2025

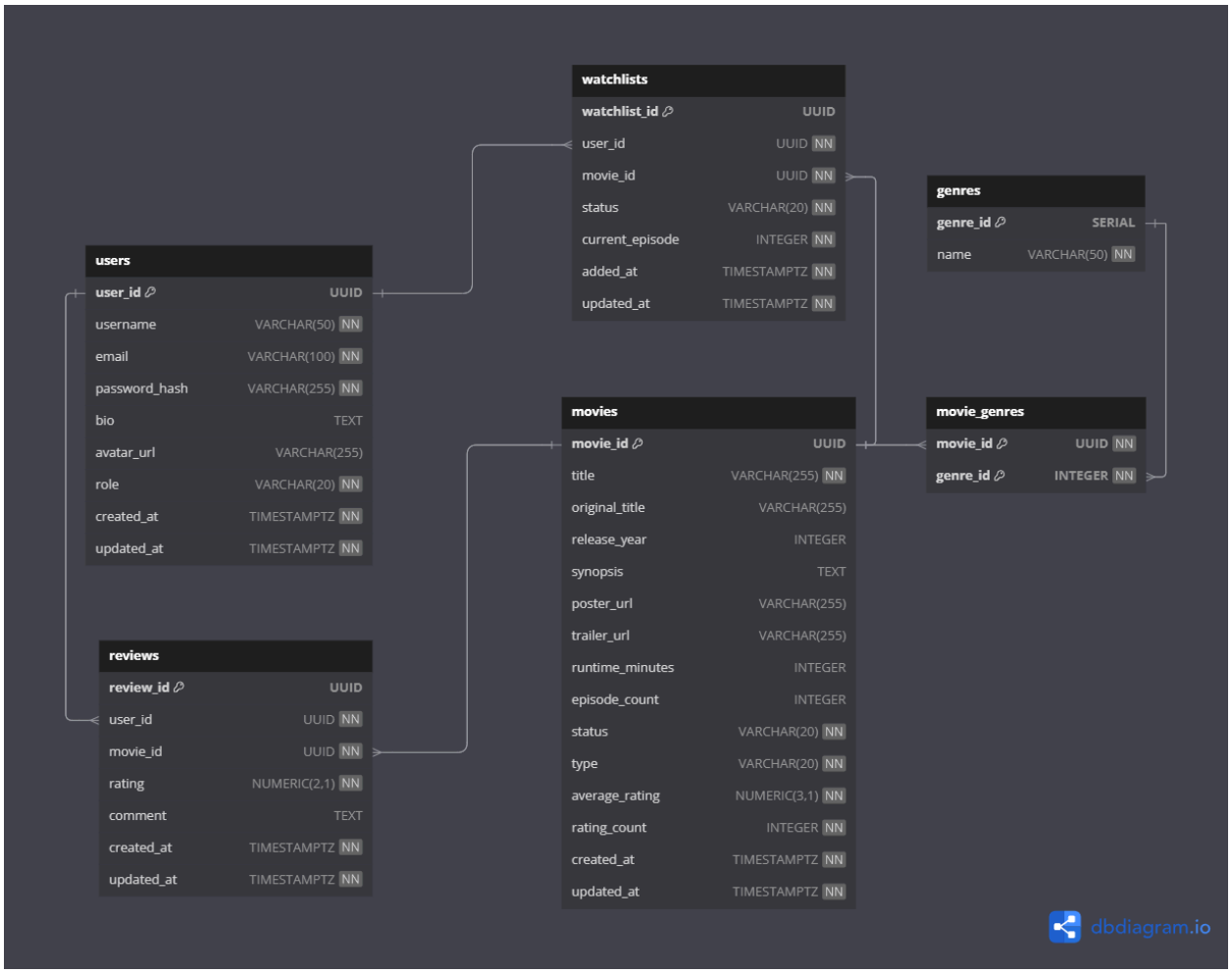
CT313H: WEB TECHNOLOGIES AND SERVICES

- **Project/Application name:** Movie & anime review and management platform "CineVerse"
- **GitHub links (for both frontend and backend):** https://github.com/24-25Sem3-Courses/ct313hm02-project-Giang0402.git
- **Link Demo Project:** https://www.youtube.com/watch?v=wgJzMVSXlVg
- **Student ID 1:** B2205976
- **Student Name 1:** Trần Hữu Giang
- **Student ID 2:** B2205983
- **Student Name 2:** Đỗ Đạt Hoa
- **Class/Group Number (e.g, CT313HM01):** CT313HM02

I. Introduction

- **Project/application description:**
  The CineVerse project aims to build a comprehensive website platform that serves as a digital library and online community for movie and anime lovers. Inspired by MyAnimeList and IMDB, CineVerse will provide features for managing watch lists, rating, commenting, and discovering new content. The main goal is to create a friendly space where users can easily track their interests, share opinions, and connect with a community of like-minded enthusiasts.

- **PDM:**


watchlists
watchlist_id UUID
user_id UUID NN
movie_id UUID NN
status VARCHAR(20) NN
current_episode INTEGER NN
added_at TIMESTAMPTZ NN
updated_at TIMESTAMPTZ NN
genres
genre_id SERIAL
name VARCHAR(50) NN
users
user_id UUID
username VARCHAR(50) NN
email VARCHAR(100) NN
password_hash VARCHAR(255) NN
bio TEXT
avatar_url VARCHAR(255)
role VARCHAR(20) NN
created_at TIMESTAMPTZ NN
updated_at TIMESTAMPTZ NN
movies
movie_id UUID
title VARCHAR(255) NN
original_title VARCHAR(255)
release_year INTEGER
synopsis TEXT
poster_url VARCHAR(255)
trailer_url VARCHAR(255)
runtime_minutes INTEGER
episode_count INTEGER
status VARCHAR(20) NN
type VARCHAR(20) NN
average_rating NUMERIC(3,1) NN
rating_count INTEGER NN
created_at TIMESTAMPTZ NN
updated_at TIMESTAMPTZ NN
movie_genres
movie_id UUID NN
genre_id INTEGER NN
reviews
review_id UUID
user_id UUID NN
movie_id UUID NN
rating NUMERIC(2,1) NN
comment TEXT
created_at TIMESTAMPTZ NN
updated_at TIMESTAMPTZ NN
dbdiagram.io

- **Task assignment sheet:**

	Tasks	Giang	Hoa
	Tasks performed together		
1	Design and implement Database	X	X
2	Design and implement Document REST API	X	X
3	Create structure for frontend and backend	X	X
	Project features (both frontend and backend)		
1	Register		X
2	Log in		X
3	Log out		X
4	Review movie		X
5	View profile		X
6	Search		X
7	View movie infomation	X	
8	Add movie to watchlist	X	
9	View watchlist	X	
10	Delete movie from watchlist	X	
11	Update movie in watchlist	X	
12	Add movie	X	

Other libraries used: (axios, vue-toastification)

In the ***cineverse.service.js*** file, Axios is used to create a centralized and robust service for handling all API communications**.** It's configured with advanced features to streamline requests, authentication, and error handling.

Here's a brief breakdown of how it's used:

- **Custom Instance Configuration**: A dedicated Axios instance named apiClient is created using axios.create(). This instance is pre-configured with a baseURL (/api) and default Content-Type headers, simplifying all subsequent API calls.
- **Request Interceptor for Authentication**: A request interceptor automatically attaches an authentication token to every outgoing request. Before a request is sent, it checks a Pinia store (useAuthStore) for a token and, if one exists, adds it to the Authorization header as a Bearer token. This eliminates the need to manually add the token for each protected API call.
- **Response Interceptor for Data & Error Handling**: A response interceptor processes all incoming responses from the server:
    - **Success Handling**: For successful responses, it automatically unwraps the data structure, often returning response.data.data directly. This provides the application components with the exact data they need, making the code cleaner.
    - **Centralized Error Handling**: It standardizes error messages from various types of failures (server response, network error, etc.). Critically, if it detects a 401 Unauthorized status, it automatically triggers the authStore.logout() function to sign the user out.
- **Organized API Methods**: The cineverseService object acts as a single, organized collection of all API functions, grouped by resource (e.g., Movies, Users, Reviews). Each function uses the pre-configured apiClient to perform a specific action (GET, POST, DELETE, etc.), ensuring all requests are consistent and benefit from the interceptors.

In ***backend-api\src\db\seeds\01_initial_movie_data.js*** file, axios is used to create a pre-configured, reusable API client for interacting with The Movie Database (TMDb).

An instance named tmdbApi is created using axios.create(). This configuration sets a baseURL for all requests and automatically includes default query parameters like the api_key and language in every call.

This tmdbApi instance is then used repeatedly throughout the script with the .get() method to fetch various data, such as genre lists, popular movies/TV shows, and detailed information for each media item. This approach simplifies the code by centralizing the API configuration and avoiding repetition.

Vue-toastification is a lightweight and highly customizable library for Vue.js applications. Its primary function is to display "toast" notifications small, non-disruptive pop-ups that typically appear in a corner of the screen. These notifications are ideal for providing immediate feedback on the result of an action, such as confirming a success, reporting an error, or issuing a warning.

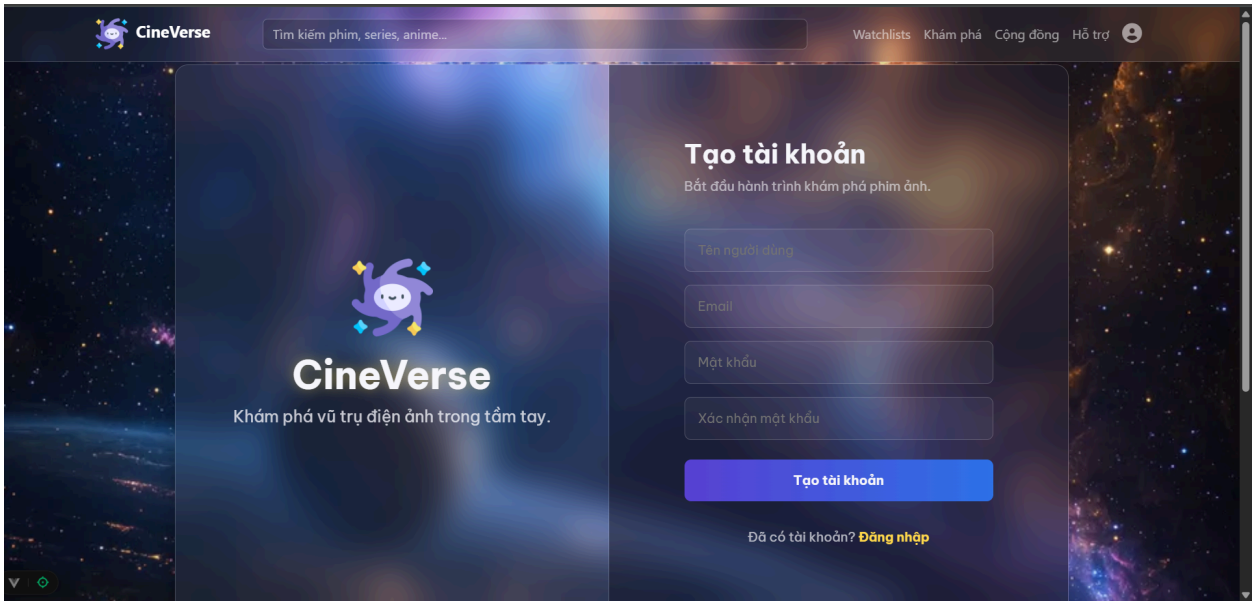
The primary and consistent usage pattern throughout the project is as follows:

1. **Integration with useMutation**: The notifications are directly tied to the results of actions (like adding, editing, or deleting data) via @tanstack/vue-query.
    - **On Success (onSuccess)**: toast.success() is called to inform the user that their action was completed successfully. Examples include:
        - A **"Đăng nhập thành công!"** notification in *AuthView.vue.*
        - A **"Cập nhật phim thành công!"** notification in *MovieManagementView.vue.*
        - A **"Đã gửi đánh giá thành công!"** notification in *MovieDetailView.vue.*
        - An **"Cập nhật ảnh đại diện thành công!"** notification in *UserProfile.vue.*
    - **On Failure (onError)**: toast.error() is called to report an error, helping the user understand why the action failed. Examples include:
        - An **"Email hoặc mật khẩu không chính xác."** notification in *AuthView.vue.*
        - An error notification when avatar upload fails in *UserProfile.vue.*
        - An error notification when a movie cannot be deleted in *MovieManagementView.vue.*
    - **Client-side Validation**: The library is also used to report user-side errors before submission, such as non-matching passwords or an invalid file upload.

II. Details of implemented features

- **Feature 1: Register**

- **Description:** The registration feature allows new users to create an account on the CineVerse platform by providing basic information such as a username, email address, and password.
- **Screenshots:**



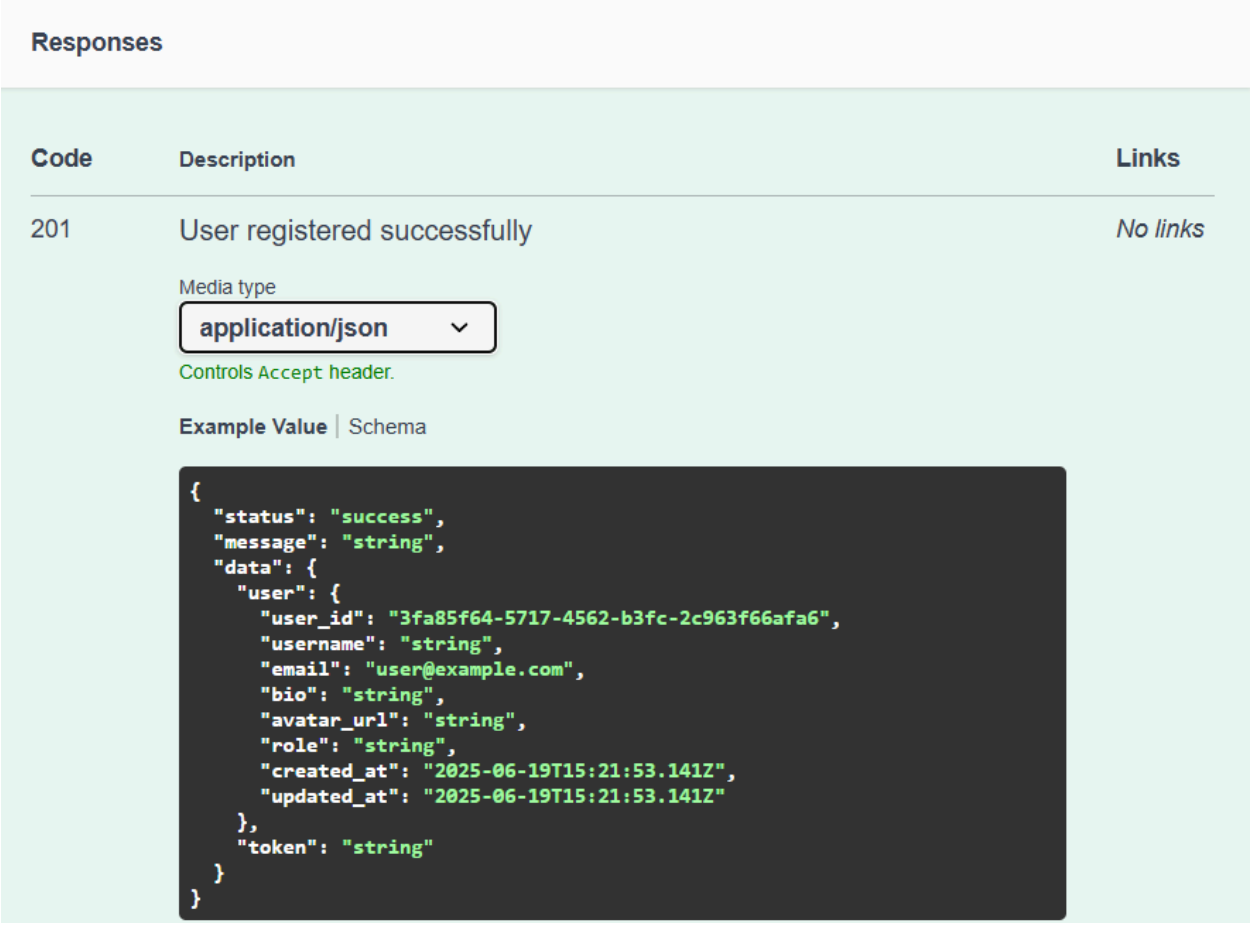
- **Implementation details:**
  - **Server-side APIs used:**
    - **Endpoint: POST /auth/register**
    - **Data format/structure sent (Request Body - JSON):**

Request body required application/json

Example Value | Schema

```
{
  "username": "testuser",
  "email": "user@example.com",
  "password": "password123"
}
```

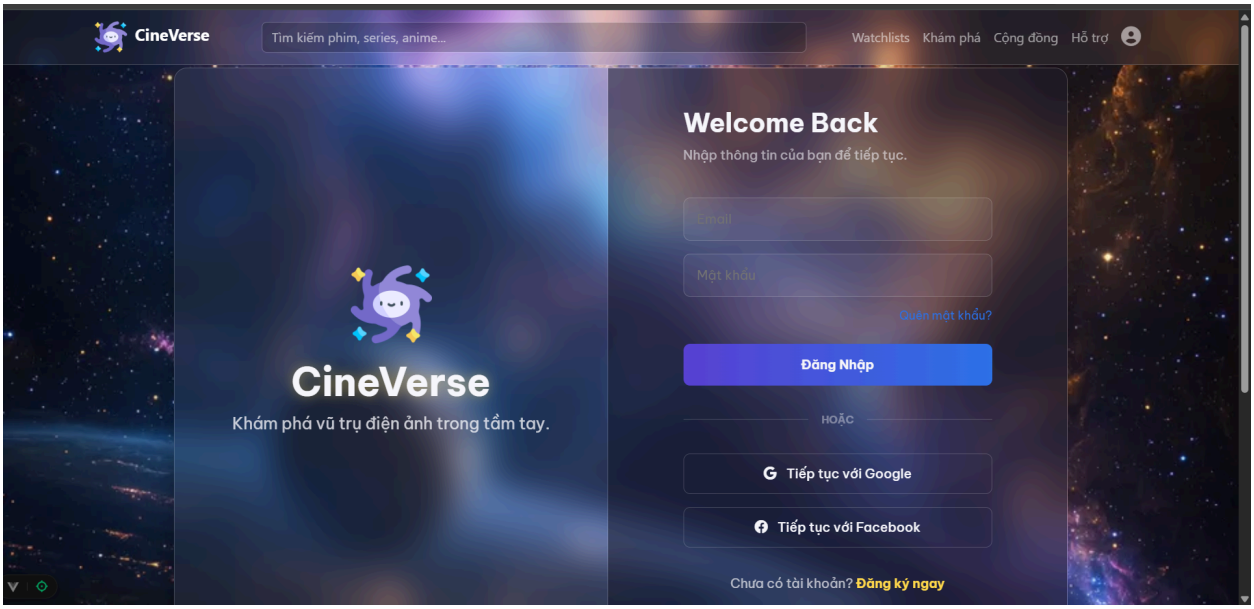
- **Data format/structure received (Response - JSON):**



+ **Data Read/Stored:** The new user's information (username, email, hashed password, and the default role 'user') is stored in the users table in the database.
+ **Client-side States:** The register feature utilizes client-side states, such as registerForm to store user input, isRegisterLoading to track submission status, formMessage for feedback, and reuses authStore to prefill login information after successful registration optionally.

- **Feature 2: Log in**

- **Description:** The login feature allows users who already have an account to access the CineVerse system by entering their email address and password.
- **Screenshots:**



- **Implementation details:**
  - + **Server-side APIs used:**
    - **Endpoint: POST /auth/login**
    - **Data format/structure sent (Request Body - JSON):**

Request body required application/json

Example Value | Schema

```
{
  "email": "user@example.com",
  "password": "password123"
}
```

- **Data format/structure received (Response - JSON):**

Responses

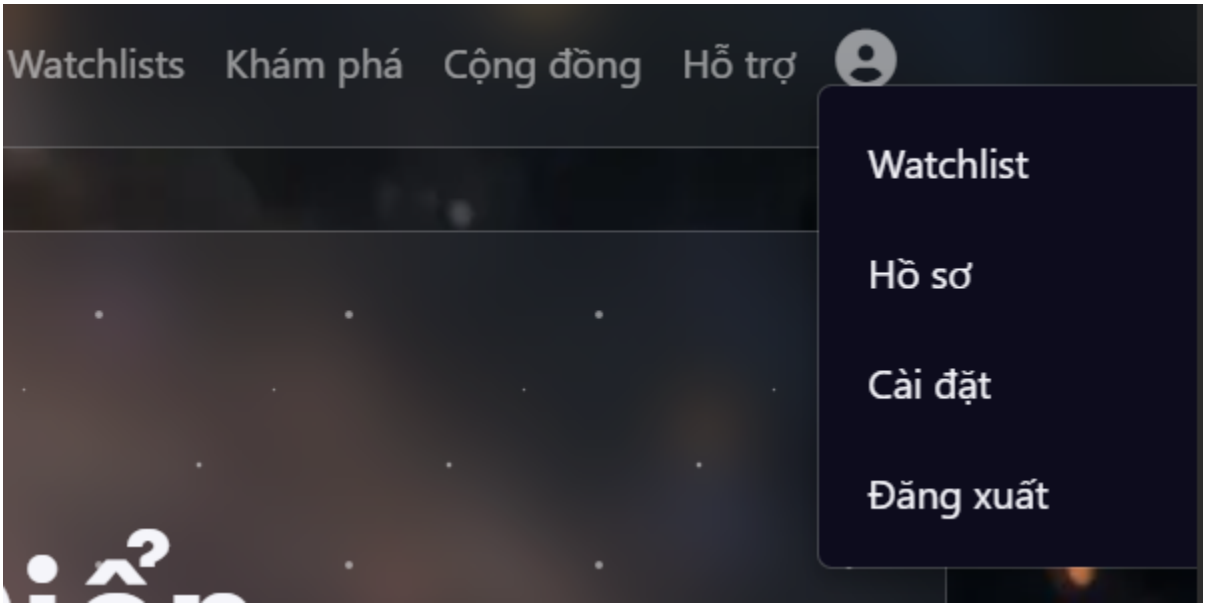
Code	Description	Links
200	User logged in successfully Media type: application/json Controls `Accept` header. **Example Value** | Schema	*No links*

```
{
  "status": "success",
  "message": "string",
  "data": {
    "user": {
      "user_id": "3fa85f64-5717-4562-b3fc-2c963f66afa6",
      "username": "string",
      "email": "user@example.com",
      "bio": "string",
      "avatar_url": "string",
      "role": "string",
      "created_at": "2025-06-19T15:48:25.902Z",
      "updated_at": "2025-06-19T15:48:25.902Z"
    },
    "token": "string"
  }
}
```

+ **Data Read/Stored:** Read user information (email and password_hash) from the users table for verification.
+ **Client-side States:** The login feature uses client-side states such as loginForm to store user input, isLoginLoading to track the loading status, formMessage for displaying feedback messages, and authStore to store the authenticated user's information and token upon successful login.

- **Feature 3: Log out**

- **Description:** The logout feature allows the current user to end their session on the CineVerse platform, ensuring that their login credentials are no longer valid and preventing further access to authenticated features without logging in again.
- **Screenshots:**

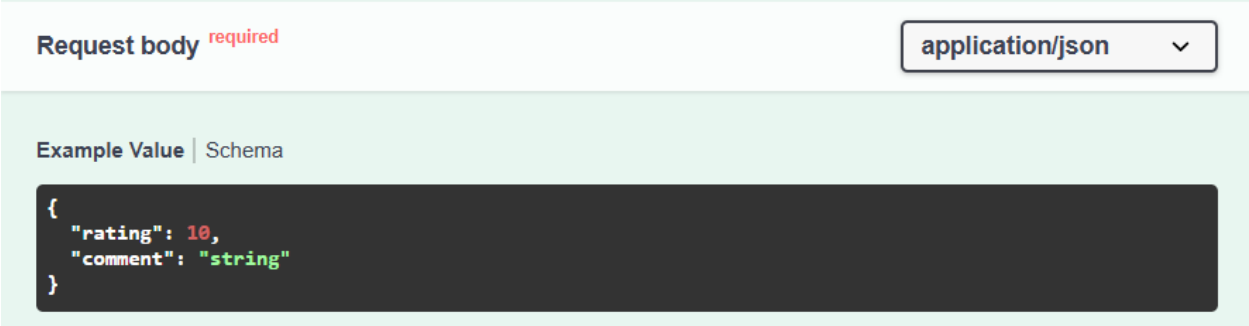


- **Implementation details:**
  + **Server-side APIs used:** No direct server-side API is used for the logout functionality because JWT is stateless. The token is invalidated by removing it from the client side.
  + **Data Read/Stored:** No data is read from or stored in the database. The token is removed from the browser's Local Storage or Session Storage.
  + **Client-side States:** The logout feature updates the authStore by clearing the user and token states, removes them from localStorage, and redirects the user to the homepage. No form or loading state is needed

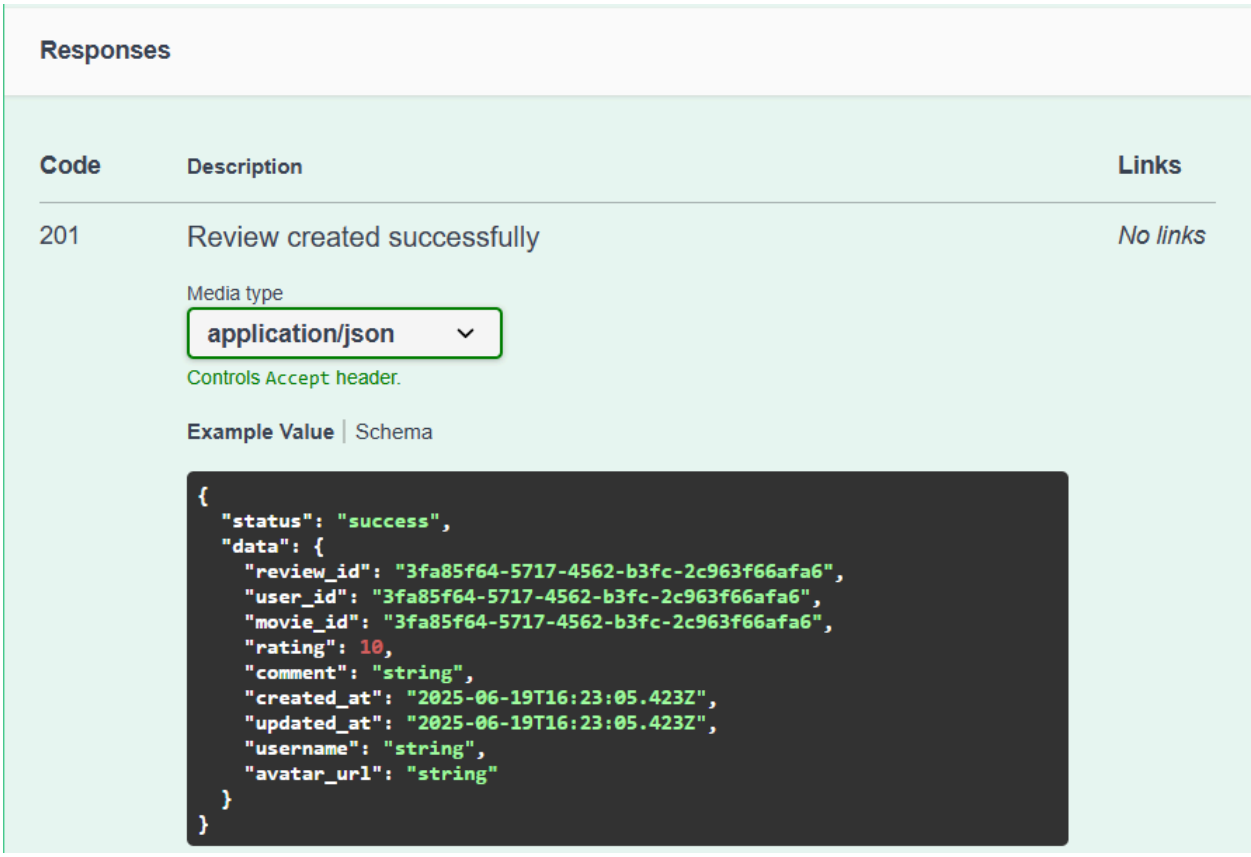
- **Feature 4: Write review**

- **Description:** This feature allows logged-in users to write or edit a detailed review of a movie, along with giving it a rating.
- **Screenshots:**

- **Implementation details:**
  - + **Server-side APIs used:**
    - **Endpoint: POST /reviews/movies/{movieId}**
    - **Data format/structure sent (Request Body - JSON):**



- **Data format/structure received (Response - JSON):**

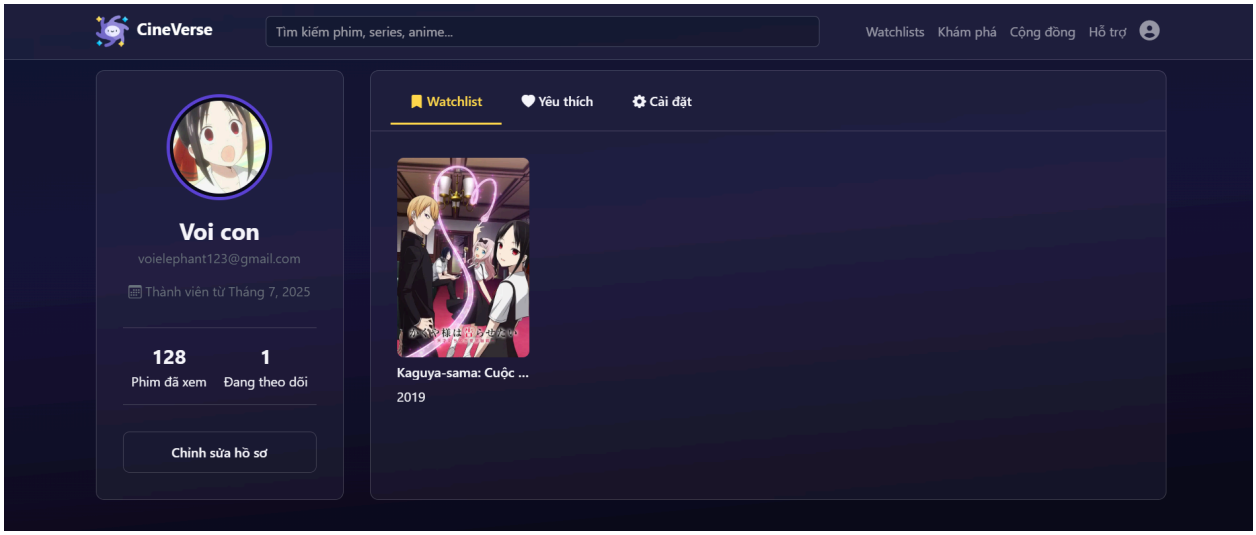


  - + **Data Read/Stored:** The rating and comment are stored in the reviews19 table. The average_rating and rating_count fields in the movies table are updated accordingly.

+ **Client-side States:** The "Write Review" feature uses states like isWritingReview and editingReviewId to manage form visibility and editing. Vue Query handles review data fetching and mutations. After submitting, related queries are invalidated to refresh the UI.

- **Feature 5: View profile**

- **Description:** The View Profile feature allows logged-in users to access their information page. Here, users can view basic details, including their username, email address, bio, and avatar.
- **Screenshots:**



- **Implementation details:**
  + **Server-side APIs used:**
    - **Endpoint: GET /users/me**
    - **Data format/structure sent (Request Body - JSON):**

▪ **Data format/structure received (Response - JSON):**

Responses

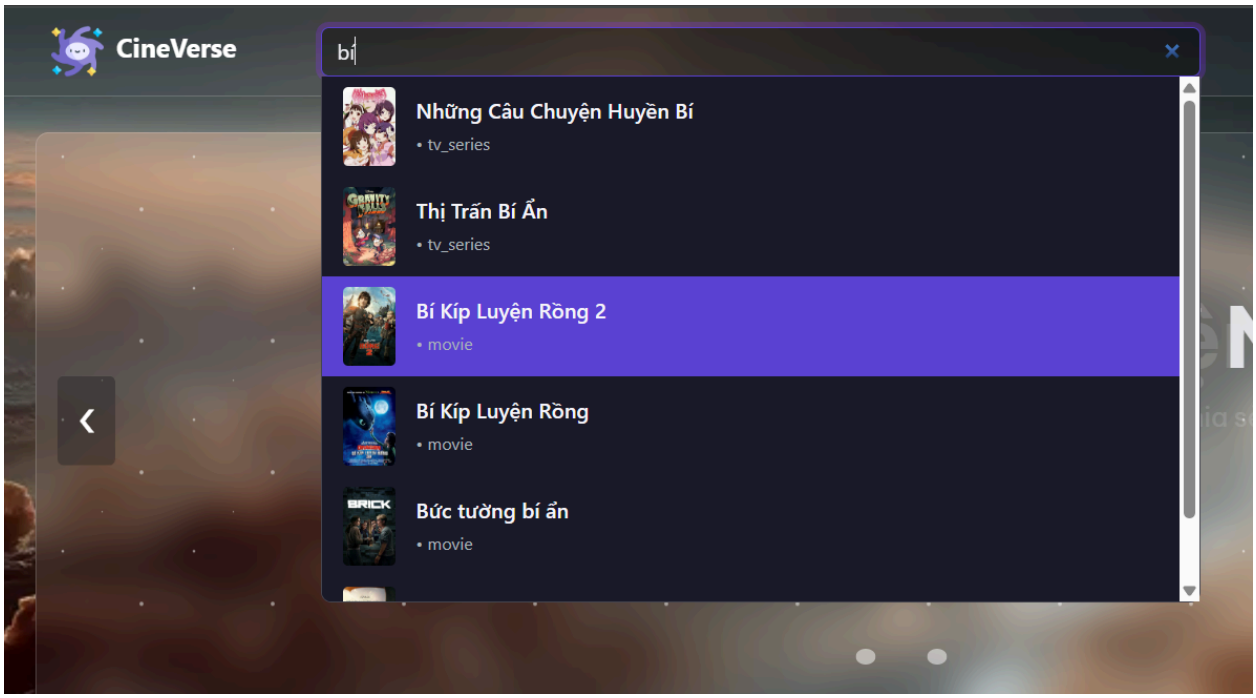
Code	Description	Links
200	User profile Media type: application/json Controls Accept header. **Example Value** | Schema	*No links*

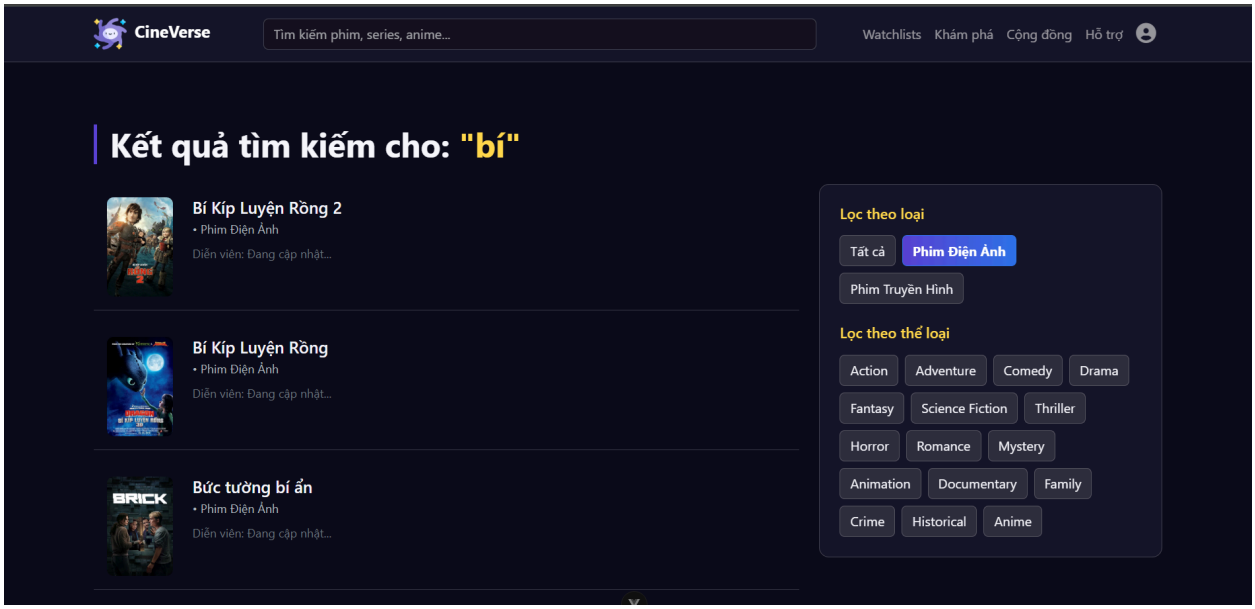

```
{
  "status": "success",
  "data": {
    "user_id": "3fa85f64-5717-4562-b3fc-2c963f66afa6",
    "username": "string",
    "email": "user@example.com",
    "bio": "string",
    "avatar_url": "string",
    "role": "string",
    "created_at": "2025-06-19T16:58:40.795Z",
    "updated_at": "2025-06-19T16:58:40.795Z"
  }
}
```


+ **Data Read/Stored:** The user profile information is read from the users table. No data is stored or modified during this operation.
+ **Client-side States:** The profile page manages UI state with activeTab to switch between sections like Watchlist, Favorites, and Settings. Data such as user profile and watchlist is fetched using Vue Query and cached. The avatar upload process uses a mutation state (isUploading) and updates the store and cache on success. Computed properties format the join date and handle conditional rendering while loading.

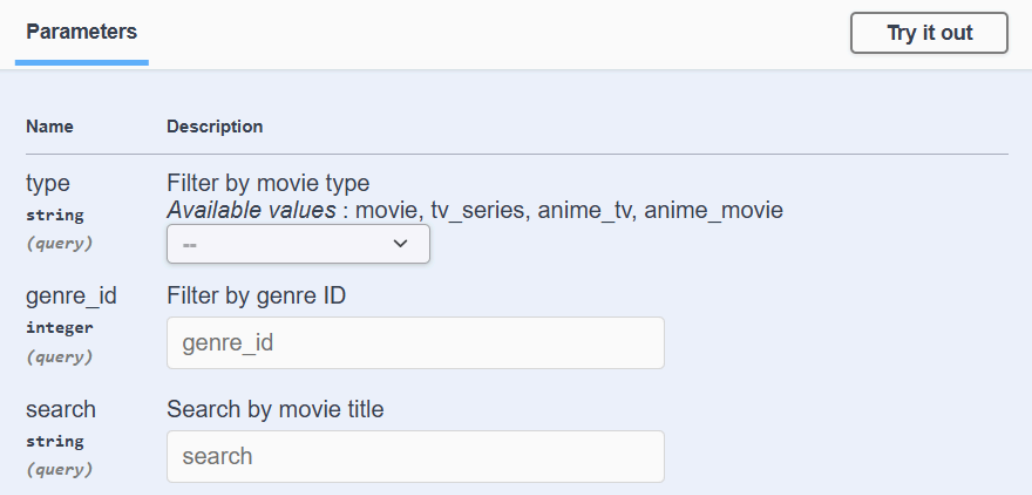
- **Feature 6: Search**

- **Description:** The search function allows users to search for movies or anime based on their titles. Search results will be displayed immediately or after the user presses Enter.
- **Screenshots:**

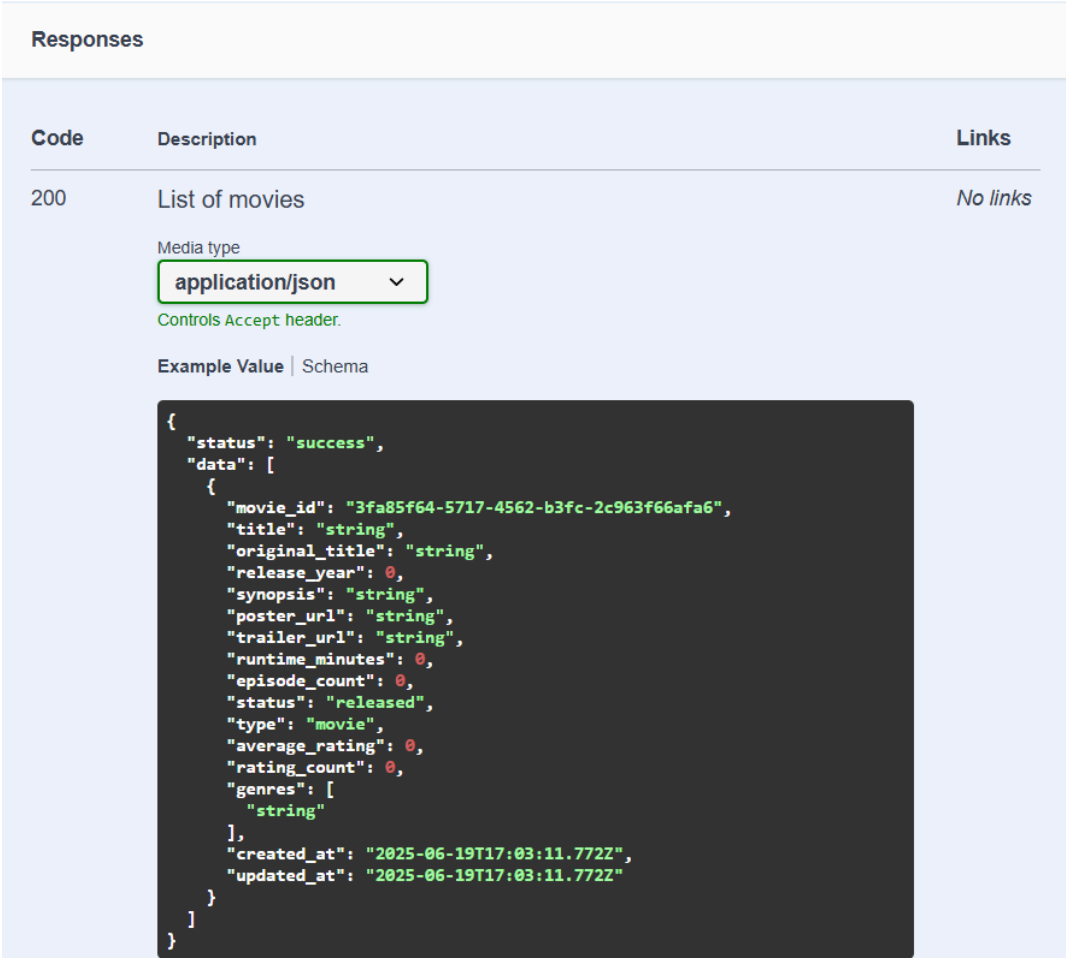




- **Implementation details:**
  - + **Server-side APIs used:**
    - **Endpoint: GET /movies**
    - **Data format/structure sent (Request - Query Parameter):**



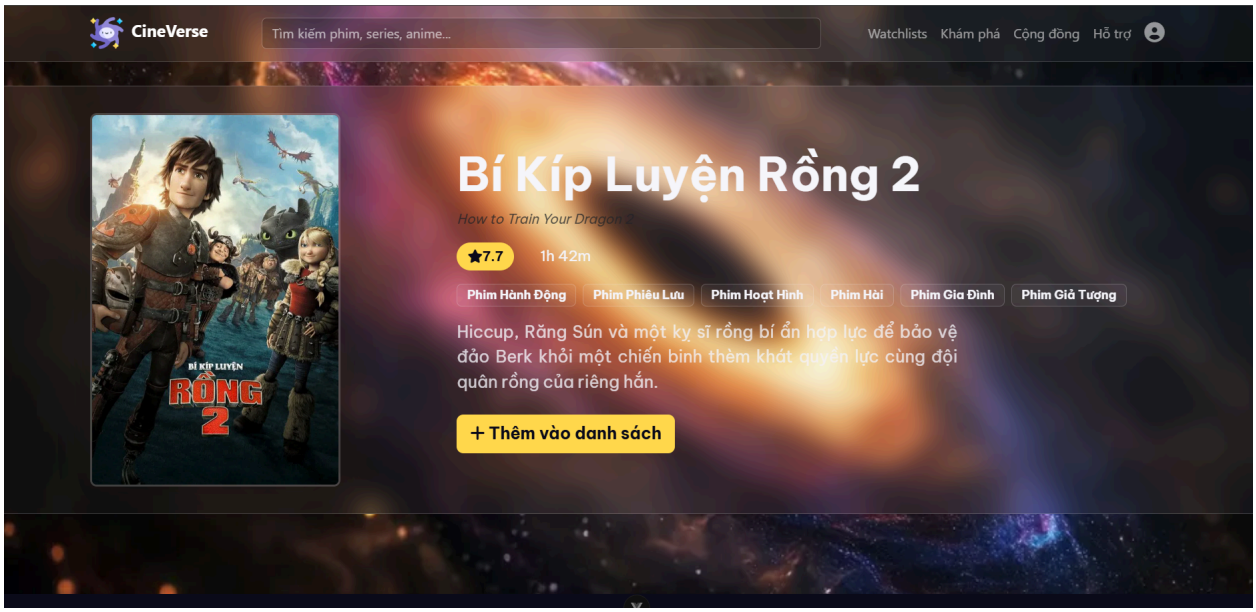
    - **Data format/structure received (Response - JSON):**



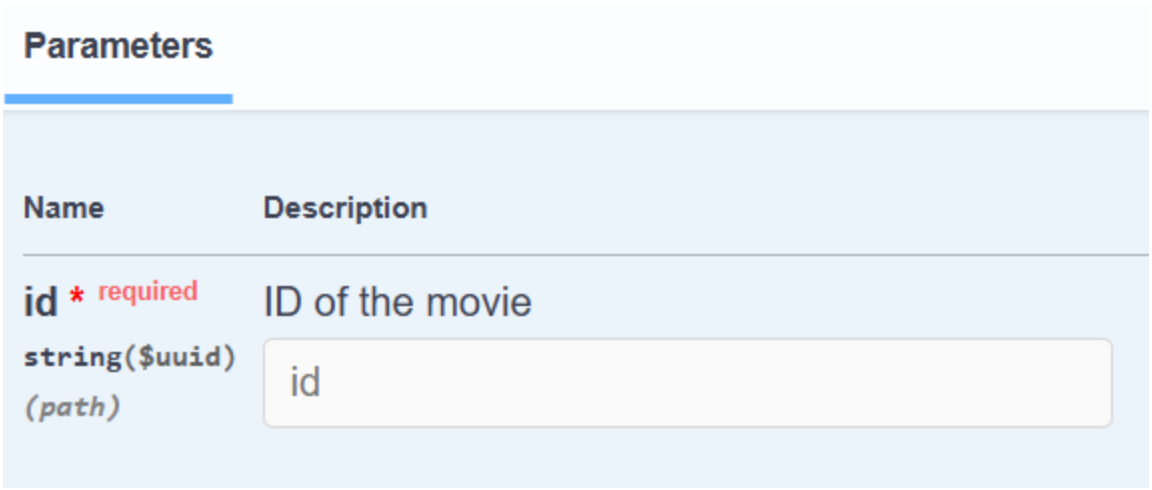
    + **Data Read/Stored:** Read data from the movies table.
    + **Client-side States:** The search feature uses reactive state for the search query (searchQuery), content type filter (activeFilter), and genre filter (activeGenre). When any of these states change, the movie list is automatically refetched via a Vue Query using a dynamic queryKey. It maintains the loading state (isLoading) and uses keepPreviousData to ensure smooth transitions. Results are displayed or a fallback message is shown if no match is found. Image errors are handled gracefully with a default poster placeholder.

- **Feature 7: View Movie Information**

- **Description:** This feature displays a detailed page for each movie or anime, including all stored information such as title, release year, synopsis, poster, trailer, average rating, and genre list.
- **Screenshots:**



- **Implementation details:**
    + **Server-side APIs used:**
        - **Endpoint: GET /movies/{id}**
        - **Data format sent (Request - Path Parameter):**



- **Data format received (Response - JSON):**

200

Movie details

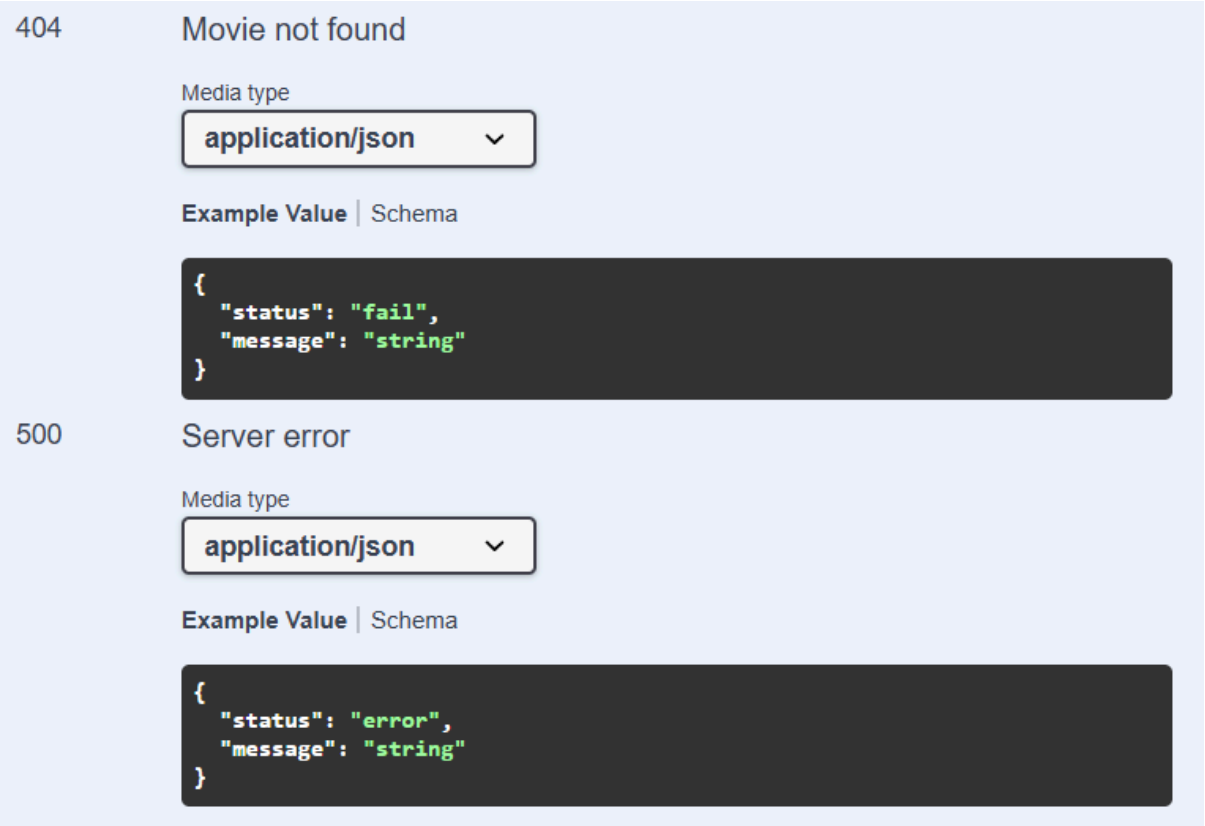
Media type

application/json

Controls Accept header.

Example Value | Schema


```
{
  "status": "success",
  "data": {
    "movie_id": "3fa85f64-5717-4562-b3fc-2c963f66afa6",
    "title": "string",
    "original_title": "string",
    "release_year": 0,
    "synopsis": "string",
    "poster_url": "string",
    "trailer_url": "string",
    "runtime_minutes": 0,
    "episode_count": 0,
    "status": "released",
    "type": "movie",
    "average_rating": 0,
    "rating_count": 0,
    "genres": [
      "string"
    ],
    "created_at": "2025-06-19T16:08:04.519Z",
    "updated_at": "2025-06-19T16:08:04.519Z"
  }
}
```

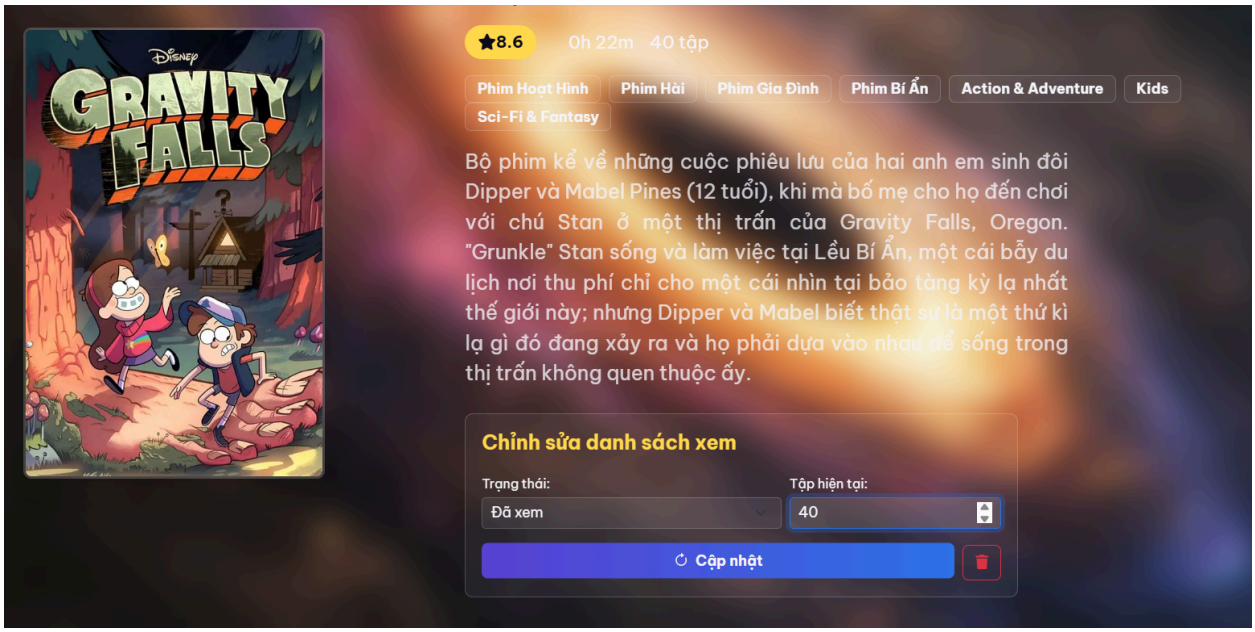


    + **Data Read/Stored:** Reads data from the movies, movie_genres, and genres tables.
    + **Client-side States:** This feature uses reactive state to manage the movie ID (movieId), loading states (isLoading, isMovieLoading, etc.), review form visibility (isWritingReview, editingReviewId), and watchlist form data (watchlistForm). Data is fetched using multiple Vue Query hooks for the movie detail, reviews, and watchlist status. State changes like adding/updating/removing a review or watchlist item trigger query invalidation to refresh data. Computed properties handle derived states like userHasReviewed, formatted runtime, and trailer embed URL.

- **Feature 8: Add Movie to Watchlist**

- **Description:** Allows logged-in users to add a movie or anime to their watchlist and choose a viewing status such as "Plan to Watch", "Watching", "Completed", or "Dropped".

- **Screenshots:**



- **Implementation details:**
  - **Server-side APIs used:**
    - **Endpoint: POST /watchlists**
    - **Data format sent (Request Body - JSON):**

Request body required

Example Value | Schema

```
{
  "movieId": "3fa85f64-5717-4562-b3fc-2c963f66afa6",
  "status": "watching",
  "currentEpisode": 0
}
```

    - **Data format received (Response - JSON):**

200 Watchlist item updated/added successfully

Media type

application/json

Controls `Accept` header.

Example Value | Schema


```
{
  "status": "success",
  "data": {
    "watchlist_id": "3fa85f64-5717-4562-b3fc-2c963f66afa6",
    "user_id": "3fa85f64-5717-4562-b3fc-2c963f66afa6",
    "movie_id": "3fa85f64-5717-4562-b3fc-2c963f66afa6",
    "status": "watching",
    "current_episode": 0,
    "added_at": "2025-06-19T16:13:05.606Z",
    "updated_at": "2025-06-19T16:13:05.606Z",
    "title": "string",
    "poster_url": "string",
    "type": "string",
    "release_year": 0
  }
}
```


400 Invalid input

Media type

application/json

Example Value | Schema


```
{
  "status": "fail",
  "message": "string"
}
```

401 Unauthorized

Media type

application/json

Example Value | Schema

```
{
  "status": "fail",
  "message": "string"
}
```

404 Movie not found

Media type

application/json

Example Value | Schema

```
{
  "status": "fail",
  "message": "string"
}
```

500 Server error

Media type

application/json

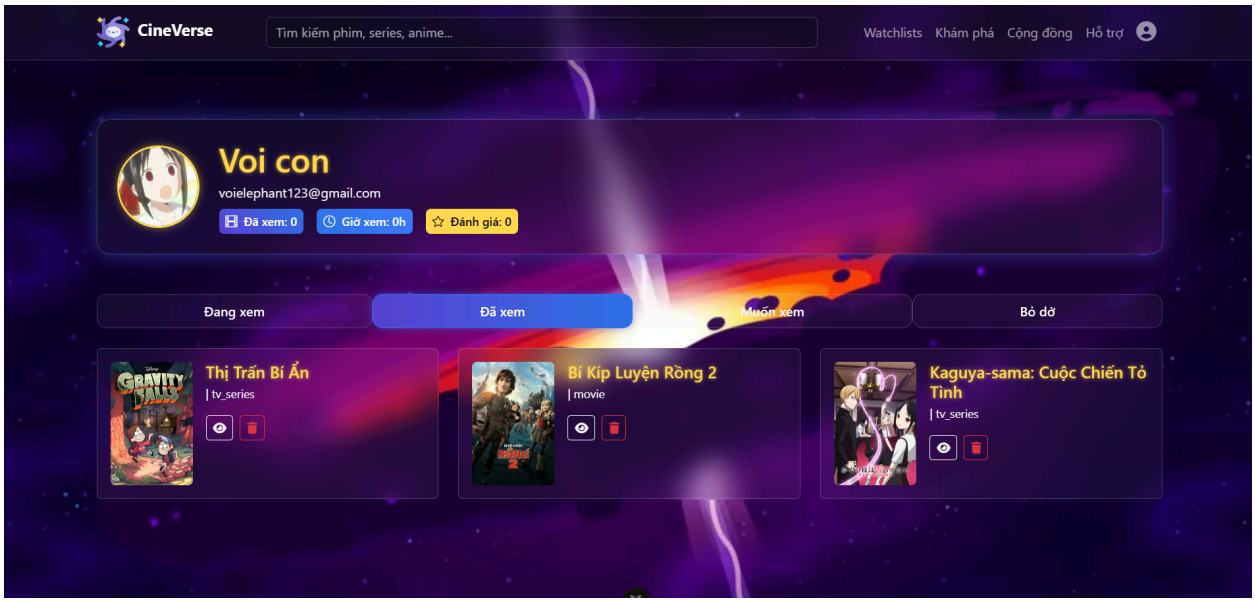
Example Value | Schema

```
{
  "status": "error",
  "message": "string"
}
```

+ **Data Read/Stored:** Data is inserted or updated in the watchlists table.
+ **Client-side States:** The component uses local state (watchlistForm) to manage the selected watch status and episode. It checks the current watchlist state with Vue Query (currentWatchlistItem) and updates the UI accordingly. Vue Mutation handles adding or updating the item, with isUpdatingWatchlist tracking submission status. State changes automatically trigger data re-fetch to reflect the latest watchlist state.

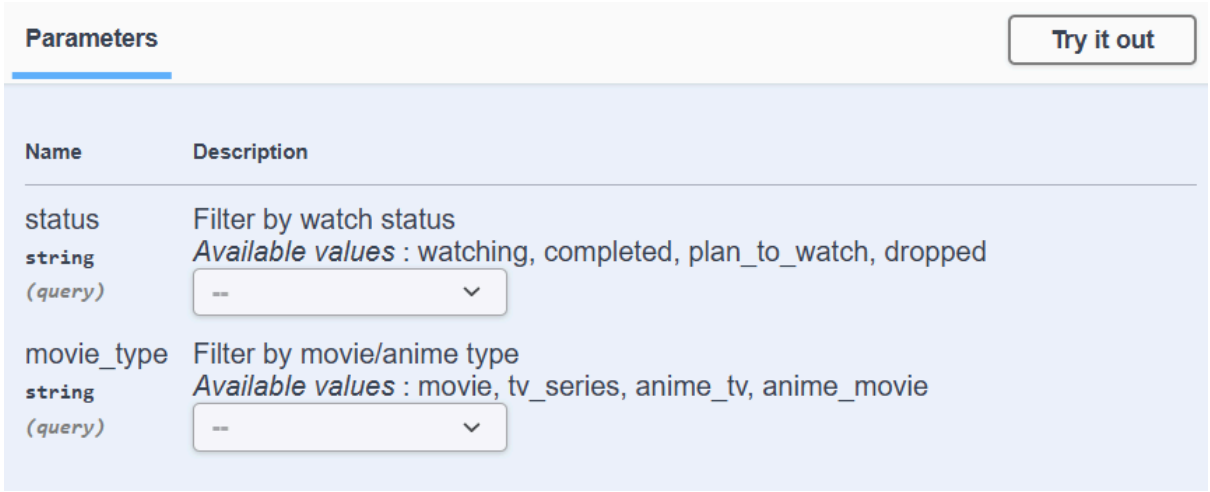
- **Feature 9: View Watchlist**

- **Description:** Displays the list of movies and anime that a user has added to their watchlist, categorized by viewing status such as "Watching", "Completed", "Plan to Watch", or "Dropped".
- **Screenshots:**

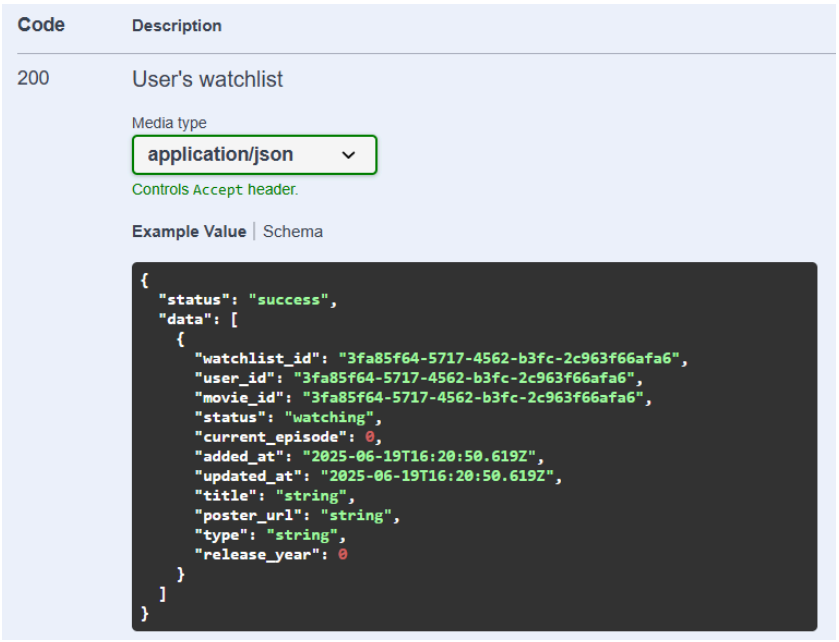


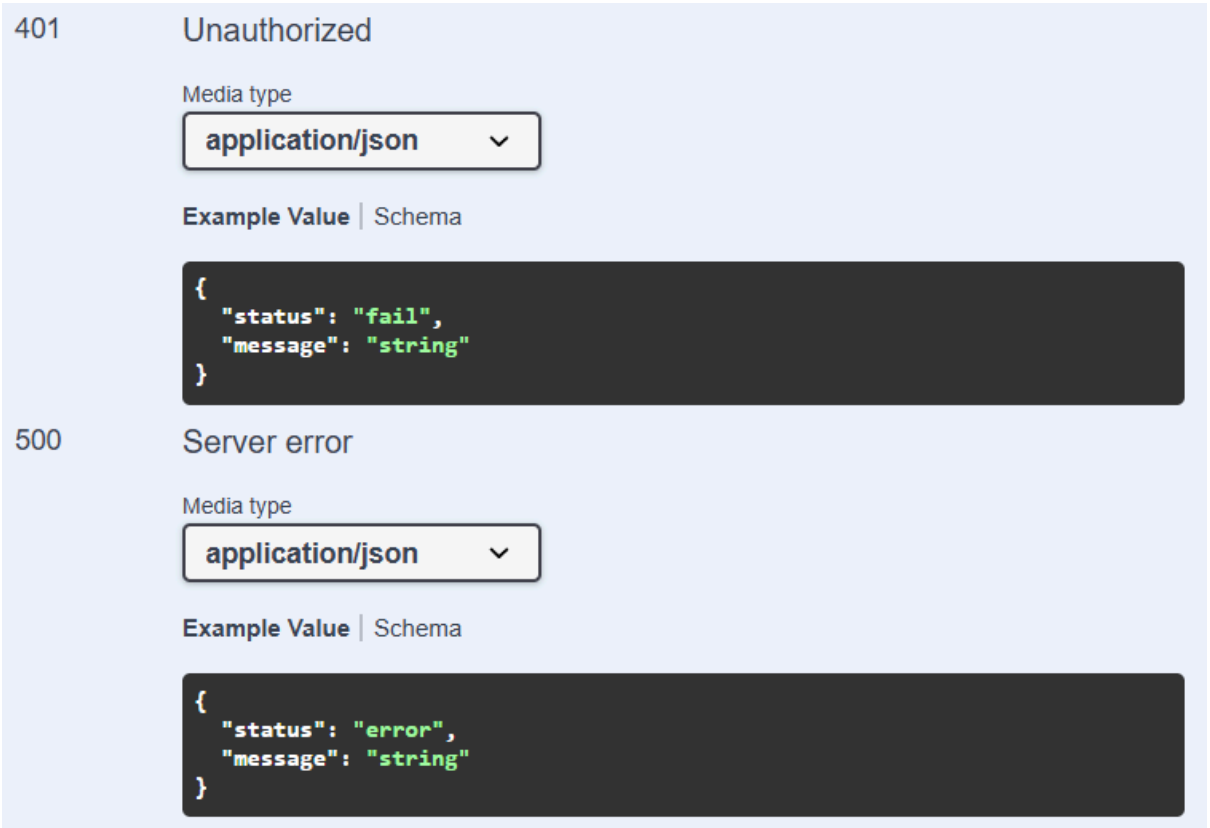
- **Implementation details:**
  - **Server-side APIs used:**
    - **Endpoint:**
      GET /watchlists?status={status}&movie_type={type}

    - **Data format sent (Request - Query Parameters):**



    - **Data format received (Response - JSON):**

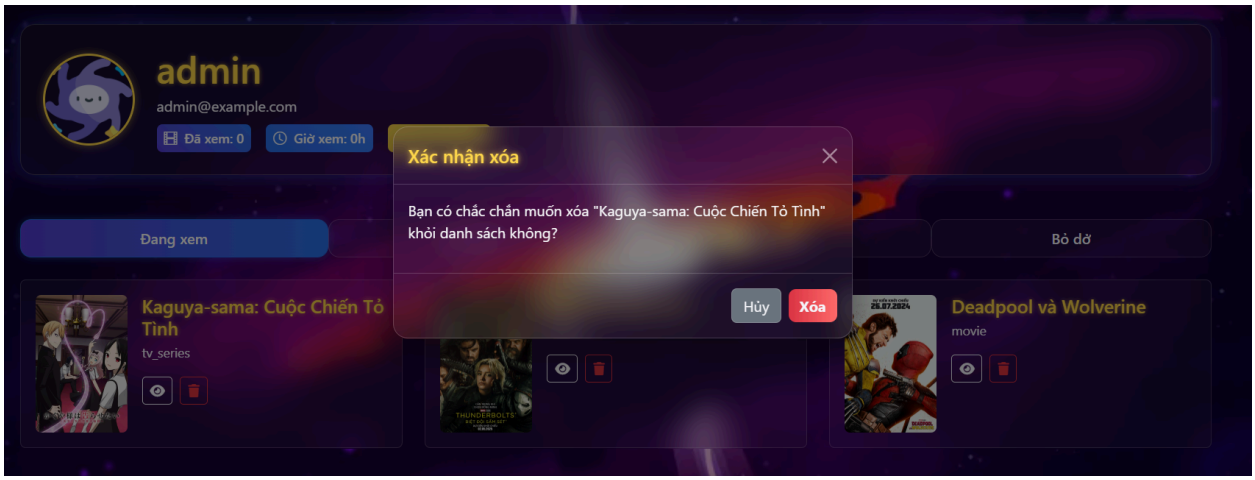




+ **Data Read/Stored:** Reads data from the watchlists and movies tables.
+ **Client-side States:** The component uses local state (activeTab) to switch between watchlist categories like "Watching", "Completed", etc. Vue Query

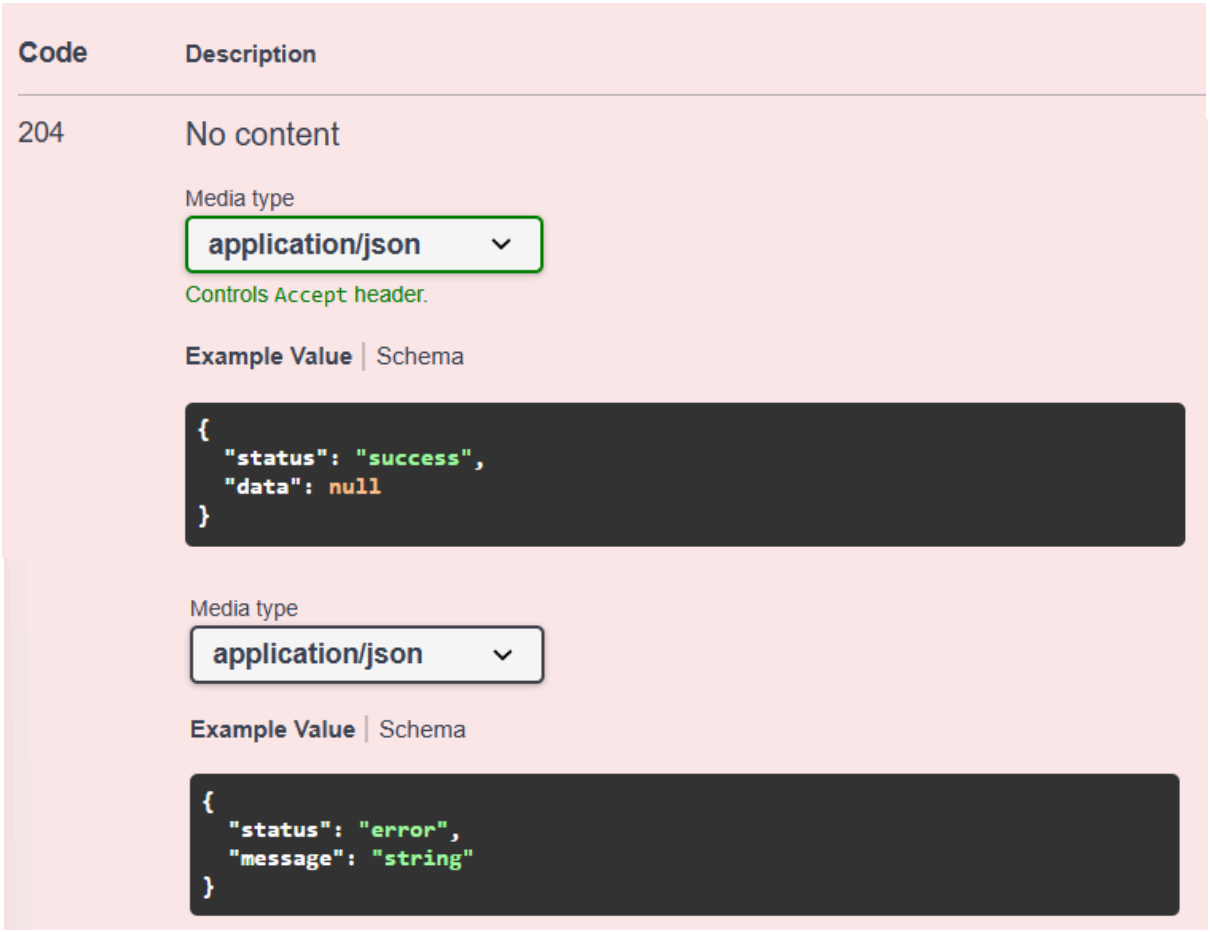
fetches both user profile and watchlist data based on the active tab. Optimistic updates are applied when removing items, with cache manipulation handled via Vue Query's onMutate and onError. Loading states (loading) are derived from query states for smooth UI feedback.

- **Feature 10: Delete Movie from Watchlist**
- **Description:** Allows users to remove a specific movie or anime from their watchlist.
- **Screenshots:**



- **Implementation details:**
  - **Server-side APIs used:**
    - **Endpoint: DELETE /watchlists/{movieId}**
    - **Data format sent (Request - Path Parameter):**

- **Data format received (Response - JSON):**

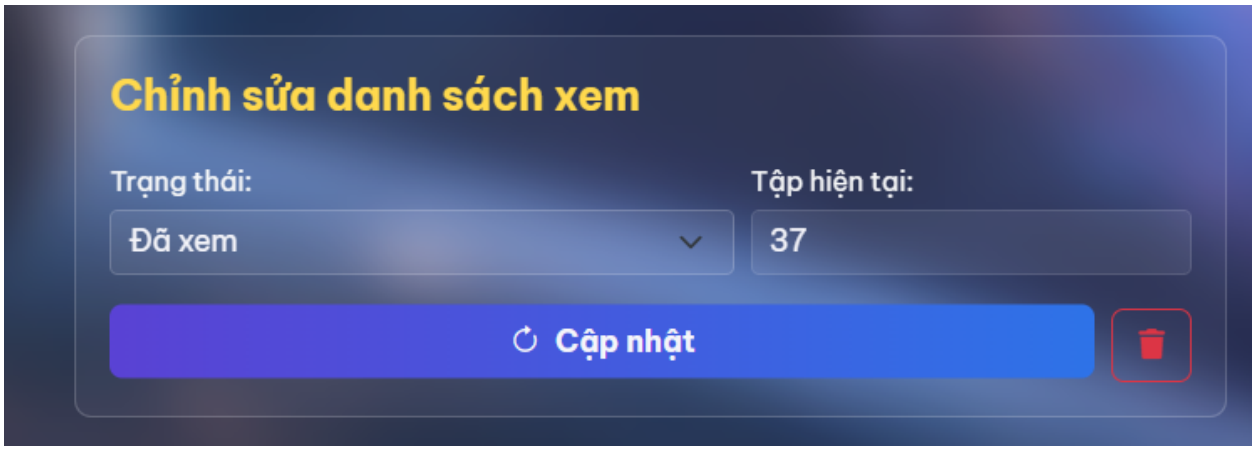


  + **Data Read/Stored:** Data is deleted from the watchlists table.
  + **Client-side States:** The component uses a local handleRemoveFromWatchlist method triggered on user confirmation. Vue Query's useMutation handles the delete action with an optimistic update. The cached watchlist is updated immediately via onMutate, and rolled back in onError if deletion fails. Query invalidation in onSettled ensures server-client consistency.

- **Feature 11: Update Movie in Watchlist**

- **Description:** Allows users to update the viewing status (e.g., from "Plan to Watch" to "Watching" or "Completed") and the number of episodes watched for a movie or anime in their watchlist.

- **Screenshots:**



- **Implementation details:**
  - + **Server-side APIs used:**
    - **Endpoint: POST /watchlists**
    - **Data format sent (Request Body - JSON):**

Request body required | application/json

Example Value | Schema

```
{
  "movieId": "3fa85f64-5717-4562-b3fc-2c963f66afa6",
  "status": "watching",
  "currentEpisode": 0
}
```

  - **Data format received (Response - JSON):**
  - + **Data Read/Stored:** Data is updated in the watchlists table.

200 Watchlist item updated/added successfully

Media type: application/json

Controls Accept header.

Example Value | Schema

```
{
  "status": "success",
  "data": {
    "watchlist_id": "3fa85f64-5717-4562-b3fc-2c963f66afa6",
    "user_id": "3fa85f64-5717-4562-b3fc-2c963f66afa6",
    "movie_id": "3fa85f64-5717-4562-b3fc-2c963f66afa6",
    "status": "watching",
    "current_episode": 0,
    "added_at": "2025-06-19T16:35:05.160Z",
    "updated_at": "2025-06-19T16:35:05.160Z",
    "title": "string",
    "poster_url": "string",
    "type": "string",
    "release_year": 0
  }
}
```

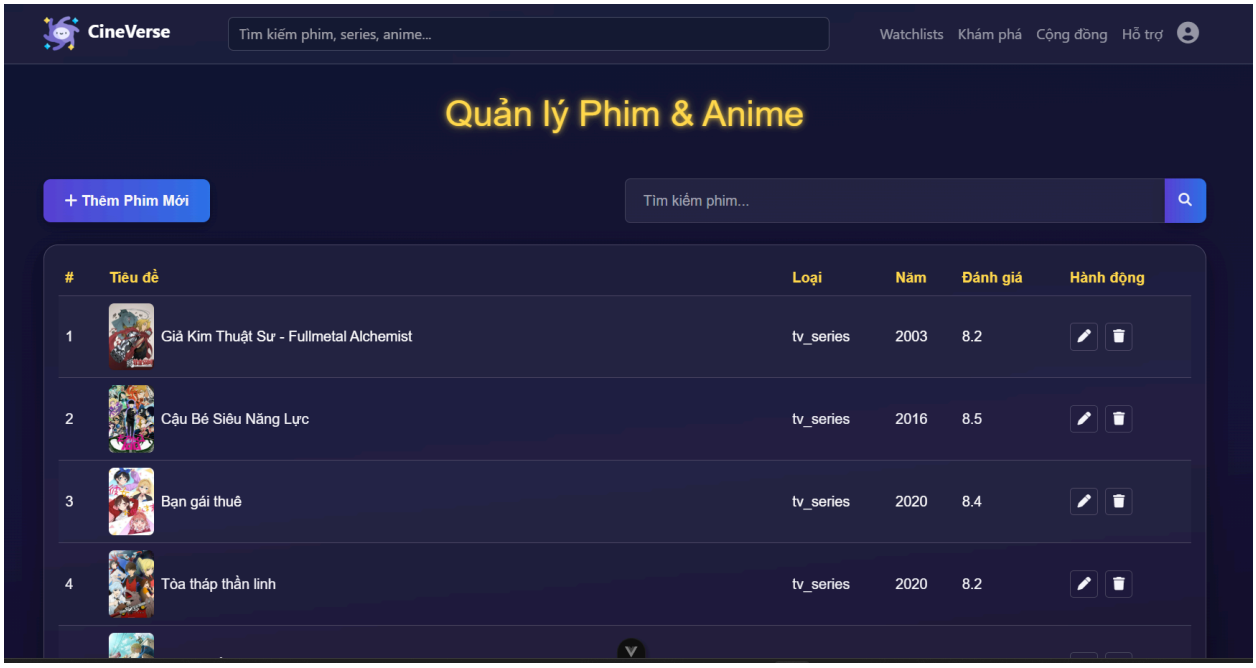
+ **Client-side States:** Local states like watchlistForm store the current status and episode. When users update the info, handleUpdateWatchlist triggers a Vue Query mutation. The UI is updated reactively after success via query invalidation. While updating, isUpdatingWatchlist shows a loading state and disables controls.

- **Feature 12: Add Movie (Admin)**

- **Description:**

This feature is for admin users, allowing them to add information about a new movie or anime to the CineVerse database.

- **Screenshots:**



- **Implementation details:**

+ **Server-side APIs used:**

▪ **Endpoint: POST /api/movies**

▪ **Data format sent (Request Body - JSON):**

- **Data format received (Response - JSON):**

201 Movie created successfully

Media type

application/json

Controls `Accept` header.

Example Value | Schema


```
{
  "status": "success",
  "data": {
    "movie_id": "3fa85f64-5717-4562-b3fc-2c963f66afa6",
    "title": "string",
    "original_title": "string",
    "release_year": 0,
    "synopsis": "string",
    "poster_url": "string",
    "trailer_url": "string",
    "runtime_minutes": 0,
    "episode_count": 0,
    "status": "released",
    "type": "movie",
    "average_rating": 0,
    "rating_count": 0,
    "genres": [
      "string"
    ],
    "created_at": "2025-06-19T16:40:56.425Z",
    "updated_at": "2025-06-19T16:40:56.425Z"
  }
}
```


403 Forbidden (not admin)

Media type

application/json

Example Value | Schema


```
{
  "status": "fail",
  "message": "string"
}
```


500 Server error

Media type

application/json

Example Value | Schema


```
{
  "status": "error",
  "message": "string"
}
```

+ **Data Read/Stored:** Movie data is saved in the movies table, and genre associations are stored in the movie_genres table.
+ **Client-side States:** Admin states include movieForm (form input model), isEditing (mode control), and modal instances (movieModalInstance). When an admin adds a movie, handleSubmitMovie triggers a mutation to the backend. During submission, isSubmitting shows a loading spinner. After success, Vue Query invalidates adminMovies, updates the table, and hides the modal. Genre selection is dynamically populated from availableGenres.