Build the "My List" feature

Overview

You are enhancing your OTT platform to include a new feature called "My List," which allows users to save their favourite movies and TV shows to a personalised list. This feature requires backend services for managing the user's list, including adding, removing, and listing saved items.

Objective

Implement the APIs for the “My List” feature on the backend so that any client (web or mobile apps) can easily consume these APIs to complete the feature. Ensure the solution is scalable, performant, and includes integration tests.

Functional Requirements

Implement the following features:

- **Add to My List** - add a movie or TV show to the user's list. Each item can be identified by a unique ID, and the user's list should not contain duplicates.
- **Remove from My List** - remove an item from the user's list using the item's unique ID.
- **List My Items** - retrieve all items in the user's list. The response should be paginated to handle potentially large lists efficiently.

Context

Assuming you are building this feature for an existing system, you will need to understand how a `User`, `Movie` and `TVShow` is represented and stored.

Here is how they are represented:

```typescript
type Genre = 'Action' | 'Comedy' | 'Drama' | 'Fantasy' | 'Horror' | 'Romance' | 'SciFi';

interface User {
  id: string;
  username: string;
  preferences: {
    favoriteGenres: Genre[];
    dislikedGenres: Genre[];
  };
  watchHistory: Array<{
    contentId: string;
    watchedOn: Date;
    rating?: number;
  }>;
}

interface Movie {
  id: string;
  title: string;
  description: string;
  genres: Genre[];
  releaseDate: Date;
  director: string;
  actors: string[];
}

interface TVShow {
  id: string;
  title: string;
  description: string;
  genres: Genre[];
  episodes: Array<{
    episodeNumber: number;
    seasonNumber: number;
    releaseDate: Date;
    director: string;
    actors: string[];
  }>;
}
```

Non-Functional Requirements

- **Performance of “List My Items”** - “List My Items” will be used every time a user opens the app or navigates to the “Home Screen” of the app. Hence, it needs to be extremely performance (think in under 10 milliseconds).
- **Integration Tests** - Write integration tests covering each API endpoint, including success and error cases.

Technical Requirements

- **Backend**: Use TypeScript. Feel free to use any web framework (like Express.js or Nest.js).
- **Database**: Use MongoDB preferably (that’s what we use at STAGE). If you are not comfortable with MongoDB, feel free to use any open-source relational database.
- **Testing Framework**: feel free to use any testing framework of your choice.

- **Deployment**: deploy your service at any hosting service of your choice.
- **CI/CD Pipelines**: use any service of your choice.

Submission Guidelines

- **Assumptions**:
  - Assume basic user authentication is in place; you can use a mock user ID for testing.
  - In the `Context` section, we have shared representation of the core entities. For your assignment, feel free to create relevant database schemas for these entities to implement your feature.
    - **Data Scripts**: Since we will setup your assignment locally to test it, provide additional data scripts to create initial data like users, movies, TV shows, lists, etc.
- Provide a Git repository containing your code, including all source files, test files, and any configuration files needed to run the application.
- Include a `README.md` file with:
  - Instructions for setting up and running your application and tests.
  - A brief explanation of your design choices, particularly how you optimized for performance and scalability.
  - Any assumptions you made during implementation.

Evaluation Criteria

- **Functionality**: The code meets all the functional requirements.
- **Code Quality**: Code is clean, modular, and follows best practices for TypeScript development.
- **Design**: The API, database schema, and caching strategy are well-designed for scalability and performance.
- **Testing**: Integration tests are comprehensive and demonstrate an understanding of testing strategies and coverage.
- **Overall**: approach this assignment with your highest quality for execution, as if you are building this to deploy on production. So your submission should be highest quality work.