MovieFlix - Next.js Movie Discovery App

Introduction

The Movie Database is a modern web application built with Next.js that allows users to discover, search, and save their favorite movies. The application uses The Movie Database (TMDB) API to fetch movie data and provides a seamless user experience with server-side rendering, responsive design, and smooth animations.

Key features include:

- Browse popular movies on the homepage
- Search for movies by title
- View detailed information about each movie including cast, crew, and trailers
- Save favorite movies for later viewing
- Fully responsive design that works on all devices
- Server-side rendering for improved performance and SEO
- Smooth animations and transitions for an engaging user experience

Installation and Setup

Prerequisites

- Node.js (v16.x or higher)
- Bun package manager
- TMDB API key (obtainable from https://www.themoviedb.org/settings/api)

Installation Steps

1. Clone the repository:

```
git clone <repository-url>
cd Noon-Frontend-Task
```

2. Install dependencies:

```
bun install

3. Create a `.env.local` file in the root directory and add your TMDB API key:
```

API_KEY=your_tmdb_api_key_here

```
4. Start the development server:
```

```bash
# Using Bun
bun run dev
```

5. Open your browser and navigate to http://localhost:3000

Building for Production

To create an optimized production build:

```
# Using Bun
bun run build
bun run start
```

Architecture & Design Decisions

Tech Stack

- **Next.js 14**: For improved routing and server-side rendering capabilities
- **TypeScript**: For type safety and improved developer experience
- **CSS Modules**: For component-scoped styling without conflicts
- **Zustand**: For global state management (favorites functionality)

Component Structure

The application follows a modular architecture with clear separation of concerns:

- **Server Components**: Used for data fetching and initial HTML rendering
- **Client Components**: Used for interactive elements and client-side state management
- **Hooks**: Custom hooks for reusable logic like data fetching and pagination
- **Stores**: Global state management for features like favorites

Performance Optimization

Several techniques were implemented to ensure optimal performance:

- **Server-Side Rendering (SSR)**: All movie detail pages use SSR to improve initial load time and SEO
- **Suspense and Streaming**: React Suspense is used to stream UI components as they become ready
- **Image Optimization**: Using Next.js Image component for automatic image optimization
- **Data Caching**: API responses are cached to reduce unnecessary network requests
- **Code Splitting**: Automatic code splitting to reduce initial load size

Challenges & Solutions

Challenge 1: Balancing SSR and Client-Side Interactivity

Challenge: Implementing server-side rendering while maintaining responsive client-side interactivity.

Solution: I created a hybrid approach where the initial data fetching happens on the server for faster page loads, but client components handle user interactions and state updates. This is implemented through a clear separation between server and client components, with server components handling data fetching and passing the data to client components that manage interactivity.

Challenge 2: Smooth Navigation Between Pages

Challenge: Ensuring smooth transitions between pages without jarring loading states.

Solution: I implemented custom loading states using Next.js's loading.tsx files and React Suspense. Additionally, I added prefetching on movie cards to start loading data before the user clicks, and created a custom backdrop loading component that loads images asynchronously without blocking the UI.

Challenge 3: Responsive Design Across Devices

Challenge: Creating a consistent experience across various screen sizes.

Solution: I built a fully responsive design using CSS modules with media queries. The layout intelligently adapts based on screen size, with components like MovieDetails using the isMobile state to adjust their rendering. This ensures the application looks great and functions well on everything from mobile phones to large desktop screens.

Challenge 4: Managing Global State

Challenge: Maintaining favorite movies across page navigations and browser sessions.

Solution: I implemented a Zustand store to manage the global favorites state. This approach provides a simple API for components to add, remove, and check favorite status while persisting the data to localStorage for persistence between sessions.

Features

- **Movie Discovery**: Browse popular movies with infinite scrolling pagination
- **Detailed Movie Information**: View comprehensive details including synopsis, cast, crew, and production information
- **Movie Search**: Search functionality with results showing as you type
- **Favorites System**: Save and manage favorite movies with persistent storage
- **Responsive Design**: Optimized for all screen sizes from mobile to desktop
- **Server-Side Rendering**: Improved SEO and initial load performance
- **Smooth Animations**: Page transitions and UI interactions with subtle animations for a polished feel
- **Error Handling**: Graceful error states with user-friendly messages
- **Accessibility**: Focus on keyboard navigation and screen reader compatibility

---