**Design Document: Movie Recommendation Website**

**1. Introduction**

The Movie Recommendation Website is a web-based platform built to help users discover new movies based on trending data and user interests. The platform pulls movie information from The Movie Database (TMDb) API and presents users with recommendations, allowing them to browse movies, search by title, and view detailed information about each movie.

**This document outlines the system architecture, design considerations, user interface (UI) design, and key components of the application.**

**2. System Architecture**

The Movie Recommendation Website is built as a single-page application (SPA) using React.js for the frontend. All movie-related data, such as trending movies, genres, and detailed movie information, is fetched dynamically using the TMDb API. The system architecture is designed to efficiently handle API calls, manage state, and provide a responsive user experience.

**2.1 Components**

* **Frontend:**
  + React.js for building the user interface.
  + Axios for making HTTP requests to the TMDb API.
  + CSS for styling and responsive design.
* **API:**
  + TMDb API for fetching movie data.
  + API endpoints used:
    - Trending Movies
    - Movie Search
    - Movie Details
* **State Management:**
  + React’s useState and useEffect hooks for managing and handling component states.
  + Caching API results in local storage to reduce redundant API calls.

**2.2 Data Flow**

1. API Requests: The website fetches movie data from the TMDb API using Axios.
2. Data Handling: Fetched data is processed in React components to display movie recommendations or search results.
3. Rendering: The components are rendered based on the data received and user interactions (e.g., clicking on a movie for more details).

**3. UI/UX Design**

The design of the Movie Recommendation Website is focused on delivering a smooth, user-friendly experience. The website waits to fully load before displaying content to ensure that all movie data is available when the user interacts with the interface. This approach minimizes partial or incomplete views of the page, offering users a cohesive browsing experience from the start. The layout is responsive, ensuring compatibility across various devices such as desktops, tablets, and smartphones.

**3.1 Key UI**

1. **Header:**
   * Contains a search bar that allows users to search for movies by title.
   * Provides basic navigation options if needed, such as a link to the homepage or trending movies.
2. **Homepage:**
   * Displays trending movie recommendations once the data is fully loaded from the TMDb API.
   * Each movie card includes the movie poster, title, rating, and release date.
   * A loading spinner or progress bar is displayed while data is being fetched, ensuring users know that content is loading. Once all data is available, the content is fully rendered.
3. **Movie Search Results Page:**
   * Displays movies based on the user’s search input.
   * Movie cards show essential details, such as the poster, title, and rating.
   * The page only renders after fetching all search results, maintaining a complete and uniform presentation for the user.
4. **Movie Details Page:**
   * When a user clicks on a movie, they are taken to a detailed page that loads after fetching all related data.
   * Detailed information includes:
     + Movie title
     + Synopsis
     + Genres
     + Release date
     + User rating
     + Poster and/or backdrop image
   * Similar movie recommendations are shown at the bottom once the full movie details are loaded.

**3.2 Design Considerations**

* Loading Experience: Since the website waits until all data is fetched, a loading spinner or progress bar is prominently displayed, providing visual feedback to users while the data is being retrieved. This ensures users are informed about the loading process.
* Mobile-first Design: The website is fully responsive, adapting to various screen sizes. CSS media queries are used to adjust the number of movies displayed on different devices, such as reducing the number of visible movie cards on mobile screens for better readability and performance.
* Visual Hierarchy: The design emphasizes clarity, with movie posters and titles given prominence to make browsing easy. Key actions, such as searching and clicking for more details, are positioned clearly and intuitively.

**4. System Components**

**4.1 Frontend Components**

1. **App Component:**
   * The root component that handles the routing and structure of the entire application.
   * Uses React Router to navigate between different pages, including the homepage, actors, upcoming movies, and favorites.
2. **Header Component:**
   * Contains multiple navigation links for routing between different sections of the website:
     + **Home**: Redirects users to the main homepage, displaying trending movie recommendations.
     + **Actors**: Allows users to browse popular or featured actors and view detailed information about them.
     + **Upcoming**: Displays upcoming movie releases based on data fetched from the TMDb API.
     + **Favorites**: Allows users to view and manage their favorite movies saved directly in TMDb, enhancing the user's experience by retaining their preferences.
3. **MovieList Component:**
   * Displays lists of movies, including trending movies, upcoming movies, or search results.
   * Fetches data from the TMDb API’s relevant endpoints.
   * Renders each movie as a MovieCard with key details such as the poster, title, and rating.
4. **MovieCard Component:**
   * Displays essential movie information in a compact form, including the movie poster, title, rating, and release date.
   * Each movie card is clickable, linking to the MovieDetail component for more information.
5. **MovieDetail Component:**
   * Displays detailed information about a specific movie when a user clicks on a movie card.
   * Information includes the movie synopsis, cast, genres, release date, user rating, and similar movie recommendations.
   * Allows users to mark movies as favorites, which are saved directly in TMDb.
6. **ActorList and ActorDetail Components:**
   * **ActorList**: Displays a list of popular or featured actors, fetched from the TMDb API. Each actor is shown in a card format with their name and a photo.
   * **ActorDetail**: When a user clicks on an actor card, this component displays detailed information about the selected actor, including a list of their filmography, biographical information, and a profile image.

**5. API Integration**

The Movie Recommendation Website uses The Movie Database (TMDb) API to fetch all movie-related data. Below is a breakdown of the key API endpoints used and how they integrate with the app:

**5.1 Trending Movies**

* **Endpoint**: /trending/movie
* **Purpose**: Fetches the top trending movies. Displayed on the homepage.

**5.2 Search Movies**

* **Endpoint**: /search/movie
* **Purpose**: Allows users to search for movies by title. Displayed as search results.

**5.3 Movie Details**

* **Endpoint**: /movie/{movie\_id}
* **Purpose**: Retrieves detailed information about a specific movie. Displayed on the movie details page.

**5.4 Upcoming Movies**

* **Endpoint**: /movie/upcoming
* **Purpose**: Fetches movies that are set to be released in the near future. Displayed on the Upcoming page, allowing users to see what new movies are coming soon.

**6. Security Considerations**

* **API Key Protection**: The TMDb API requires an API key for access. To prevent unauthorized use, the API key is stored securely in environment variables.
* **HTTPS**: All API requests and responses are handled over HTTPS to ensure data is securely transmitted.

**7. Performance Optimization**

To ensure a smooth user experience on the Movie Recommendation Website, performance optimization strategies are employed with a focus on data fetching and loading processes. The primary approach includes:

**7.1 Data Fetching Strategy**

* The website is designed to wait until all necessary data is fetched from the TMDb API before rendering the page. This method prevents users from seeing incomplete or partially loaded content, ensuring they receive a complete view of the information.

**7.2 Loading Indicators**

* While data is being fetched, a loading spinner or progress indicator is displayed. This visual feedback reassures users that the application is working to retrieve their requested information, enhancing their experience during wait times.

**7.3 Optimized API Calls**

* API calls are optimized to fetch only the data that is necessary for each page view. This helps minimize loading times and reduces unnecessary data transfer, making the application more efficient.

**7.4 Caching Strategies**

* Although the application primarily fetches data in real-time, strategies for caching previously fetched data may be implemented in future iterations to further improve performance and reduce load times.

**8. Conclusion**

The Movie Recommendation Website was designed with a focus on simplicity, responsiveness, and efficiency. By leveraging React.js and TMDb's rich API, we created an engaging platform that helps users easily discover movies. The architecture is scalable, allowing for future expansion, such as user accounts, more personalized recommendations, and further performance enhancements.