Technical Report: Onyx Application API Integration

1. Executive Summary

The "Onyx" application leverages The Movie Database (TMDB) API to provide a rich content experience for users. This API serves as the backbone for the application's dynamic data, supplying information on movies, TV shows, trending content, and detailed metadata such as cast, genres, and ratings. The integration is built using axios for HTTP requests, ensuring efficient data fetching and state management within the React Native environment.

2. API Configuration and Architecture

2.1 Base Configuration

The application connects to the API version 3 of The Movie Database.

- **Base URL:** https://api.themoviedb.org/3
- **Client Library:** The project uses axios (v1.13.2) to handle HTTP requests.
- **Instance Creation:** A dedicated axios instance is created with default headers to streamline requests across the application.

2.2 Authentication

Authentication is handled via a **Bearer Token**.

- **Method:** The API key is passed in the Authorization header as Bearer <API_KEY>.
- **Implementation:** The key is hardcoded directly into the services/api.js file.
  - *Security Note:* Hardcoding sensitive API keys in the source code (const API_KEY = '...') is a security vulnerability. In a production environment, this should be moved to environment variables (e.g., .env files) or a secure key management system to prevent unauthorized usage or quota theft.

2.3 content-Type

The API is configured to accept and send JSON data, indicated by the 'Content-Type': 'application/json' header.

3. Implemented Endpoints

The application defines a service layer in Onyx/services/api.js that abstracts specific API endpoints into reusable asynchronous functions.

3.1 Movie Discovery

The application implements four distinct strategies for movie discovery:

- **Trending Movies:** Fetches movies trending specifically for the current day using /trending/movie/day.
- **Popular Movies:** Retrieves a list of currently popular movies via /movie/popular.
- **Top Rated:** Fetches the highest-rated movies of all time using /movie/top_rated.
- **Upcoming:** Retrieves movies scheduled for release via /movie/upcoming.
- **Parameters:** All these endpoints are set to fetch page 1 and utilize language=en-US.

3.2 TV Show Discovery

TV show data is fetched dynamically based on a category parameter.

- **Endpoint:** /tv/${category}
- **Supported Categories:** The code comments suggest support for 'airing_today', 'on_the_air', 'popular', and 'top_rated'.

- **Implementation:** getTVShows function accepts a category argument to switch between these endpoints dynamically.

3.3 Detailed Content Metadata

To populate the "Details" screen, the app fetches comprehensive data for a single item.

- **Endpoint:** /${type}/${id} (where type is 'movie' or 'tv').
- **Appended Responses:** The call utilizes TMDB's prepend_to_response feature (?prepend_to_response=credits,similar,videos) to fetch the main details, cast credits, similar movies/shows, and video trailers in a single HTTP request. This reduces network overhead significantly.

3.4 Search Functionality

- **Endpoint:** /search/multi
- **Function:** Allows users to search for both movies and TV shows simultaneously.
- **Parameters:**
  - query: URL-encoded search string.
  - include_adult: Explicitly set to false to filter out adult content.
  - page: Defaults to page 1.

4. Data Models and Consumption

4.1 Response Parsing

The API service functions generally return response.data.results (an array of items) for list views, or response.data (a single object) for detail views.

4.2 Image Handling

TMDB provides relative paths for images (e.g., /poster_path.jpg). The application constructs the full URL on the client side.

- **Base Image URL:** https://image.tmdb.org/t/p/original.
- **Usage:** This is concatenated with the poster_path from the API response to render high-resolution images in components like DetailsScreen and HeroBanner.

4.3 Key Data Fields Used

Based on the UI components (DetailsScreen.jsx and HomeScreen.jsx), the application relies on the following fields:

- id: Unique identifier for routing and watchlist management.
- title / name: The name of the content (handles the difference between movie and TV naming conventions).
- poster_path: For cover art.
- vote_average: Used to calculate the "Match" percentage (e.g., Math.round(details.vote_average * 10)).

- release_date / first_air_date: Used to display the release year.
- overview: A text summary of the plot.
- credits.cast: An array used to display actor names.
- genres: An array of genre objects used to display tags.

5. Performance and Error Handling

5.1 Concurrency

In the HomeScreen, the application utilizes Promise.all to fetch Trending, Popular, Top Rated, and Upcoming data simultaneously.

- **Benefit:** This reduces the initial load time significantly compared to awaiting each request sequentially.
- **Loading State:** A unified loading state manages the UI, displaying an activity indicator until all initial data fetches are complete.

5.2 Error Management

Every API function in services/api.js is wrapped in a try/catch block.

- **Logging:** Errors are logged to the console (e.g., console.error('Error fetching trending movies:', error)).
- **Fallbacks:** Functions are designed to return an empty array [] or null upon failure, preventing the application from crashing due to undefined data in UI components.

6. Conclusion

The Onyx project demonstrates a standard, effective integration of the TMDB API. It covers all essential features expected of a streaming-style app: discovery, search, and detailed information. The code is modular, separating data fetching logic (api.js) from UI presentation (jsx files), which promotes maintainability. Addressing the hardcoded API key and image sizing are the primary technical steps required to move this from a prototype to a production-ready application.

rely.