Project Report: API Integration in "Onyx"
Project Title: Onyx - Movie & TV Discovery Application Developer

1. 1. Introduction

For my mobile application project, I developed "Onyx," a streaming-style discovery app built with React Native and Expo. The core functionality of the app relies on real-time data integration. I chose to use The Movie Database (TMDB) API as my primary data source because it provides comprehensive metadata for movies and TV shows, including ratings, cast members, and high-quality images.

2. Technical Stack

- **Framework: React Native (via Expo)**
- **HTTP Client: Axios (v1.13.2)**
- **Data Source: TMDB API v3**

3. Implementation Details

3.1 Setup and Configuration

I started by creating a dedicated service file (services/api.js) to handle all network logic. To avoid repetition, I configured a global axios instance with the Base URL https://api.themoviedb.org/3.

- **Authentication:** I authenticated my requests using a Bearer Token in the headers. For this prototype, I stored the API_KEY directly in the file to ensure immediate connectivity during development.

3.2 Data Fetching Strategies

My goal was to create a diverse and engaging user experience, so I implemented several specific fetching strategies:

- **Dashboard Content:** To populate the Home Screen, I created distinct functions for different content types.
  - getTrendingMovies: Fetches the daily trending list so users see fresh content every 24 hours.
  - getPopularMovies, getTopRatedMovies, and getUpcomingMovies: These functions target specific endpoints (like /movie/upcoming) to organize content into horizontal scrollable rows.

- **Dynamic Category Filtering:** For TV shows, I avoided writing four separate functions. Instead, I wrote a single flexible function getTVShows(category) that accepts a parameter (like 'on_the_air' or 'popular'). This injects the category directly into the URL path (/tv/${category}), making the code much cleaner and easier to maintain.
- **Optimized Detail Retrieval:** One of the challenges was getting all the necessary data for the "Details" screen without making the app slow. I utilized TMDB's prepend_to_response feature in my getDetails function.
    - *How it works:* By appending ?prepend_to_response=credits,similar,videos to the request, I was able to fetch the movie's metadata, cast list, similar recommendations, and trailers in a **single HTTP request**. This significantly reduced loading times compared to making four separate calls.
- **Search Functionality:** I implemented a searchContent function using the /search/multi endpoint. This was efficient because it allows the user to find both movies and TV shows in the same results list without toggling filters.

3.3 Error Handling

During development, I learned that network requests can often fail due to poor connection. To prevent the app from crashing, I wrapped every API call in a try...catch block.

- **Fallback Strategy:** If a fetch fails, my code logs the error to the console and returns an empty array [] or null. This ensures that even if the API is down, the UI components will simply render nothing instead of breaking the entire application.