## Chosen APIs from REST Countries API

In this project, the REST Countries API v3.1 was used extensively to fetch country-related data. The API provided structured and comprehensive data that allowed the application to support searching, filtering, and viewing detailed information about countries. The following endpoints were integrated:

1. **GET /all?fields=name,cca3,flags,capital,region,languages,population**
    - **Purpose**: Fetch a list of all countries with selected fields for performance optimization.
    - **Used in**: Initial data loading and rendering of the full country list.
2. **GET /name/{name}?fields=name,cca3,flags,capital,region,languages,population**
    - **Purpose**: Search countries by name.
    - **Used in**: Search bar input to find matching country names.
3. **GET /region/{region}?fields=name,cca3,flags,capital,region,languages,population**
    - **Purpose**: Retrieve countries filtered by specific regions.
    - **Used in**: Region-based filtering dropdown or buttons.
4. **GET /lang/{language}?fields=name,cca3,flags,capital,region,languages,population**
    - **Purpose**: Retrieve countries where a specific language is spoken.
    - **Used in**: Language-based filtering functionality.
5. **GET /alpha/{code}**
    - **Purpose**: Fetch detailed information about a country using its alpha-3 code (CCA3).
    - **Used in**: Viewing complete details when clicking a specific country.
6. **GET /name/{query}?fields=name,cca3,flags**
    - **Purpose**: Used for autocomplete suggestions in the search bar.
    - **Used in**: Real-time search with dropdown suggestions as the user types.

These endpoints were chosen not only to cover the functional requirements but also to optimize performance by selecting only required fields using the ?fields= parameter, thus reducing payload size and speeding up response times.

---

## Challenges Faced and How They Were Resolved

### 1. Large API Response Size

- **Problem**: The full /all endpoint returns extensive data, slowing down the initial load.
- **Solution**: Used ?fields=name,cca3,flags,capital,region,languages,population to fetch only necessary data, reducing payload size and improving speed.

### 2. Language Filter Accuracy

- **Problem**: Filtering by language using /lang/{lang} occasionally returned unexpected results due to language codes vs. language names mismatch.

- **Solution**: Mapped user-friendly language names to valid language codes used in the API (e.g., “English” to “en”), and handled empty or invalid results with user feedback.

**3. Error Handling in Search and Suggestions**

- **Problem**: Searching for invalid or misspelled country names caused the app to break.
- **Solution**: Added error handling to return an empty array or fallback message when fetch fails or no matches are found:

**4. Debouncing for Autocomplete**

- **Problem**: Frequent API calls while typing in the search bar caused performance issues.
- **Solution**: Implemented a debounce mechanism to wait for the user to stop typing before making the API call, preventing unnecessary requests.

**5. Consistent Data Mapping**

- **Problem**: API responses were sometimes inconsistent (e.g., some countries missing capital or languages fields).
- **Solution**: Implemented null checks and fallback values during rendering to avoid app crashes and ensure a consistent UI.

**6. Code Reusability and DRY Principle**

- **Problem**: Repeated logic across different API calls increased code duplication.
- **Solution**: Abstracted all API calls into a dedicated `api.js` service module with reusable async functions, improving maintainability and clarity.

---

## Conclusion

This project demonstrates a robust integration of the REST Countries API with a modern React frontend. By carefully selecting and optimizing API endpoints, handling real-world data inconsistencies, and improving UI responsiveness, the application provides a smooth and reliable user experience. Challenges such as large payloads, language code mismatches, and dynamic filtering were overcome through thoughtful design patterns, clean code practices, and effective error handling.