Pokemon Explorer App Documentation

Christos N. Ioannidis, LinkedIn

February 19, 2025

1 Introduction

This document outlines the development of the **Pokémon Explorer App**, a PokeDex like mobile application that interacts with the PokéAPI to provide users with relevant Pokémon data.

The App's required functionality includes:

- **Type Selection and Search:** Users should be able to choose one of 10 specific Pokémon types—Fire, Water, Grass, Electric, Dragon, Psychic, Ghost, Dark, Steel, and Fairy. Additionally, they should be able to search for Pokémon by name within the selected type.
- **Pokémon Listing:** The application should display a list of Pokémon belonging to the selected type. Initially, only the first 10 Pokémon should be shown, with an option for users to load more Pokémon dynamically by fetching additional results from the API.
- **Pokémon Details:** For each Pokémon, the application should provide detailed information, including its name, image, and key stats such as HP, Attack, and Defense.

2 Repository and Source Code

The source code for the Pokemon Explorer App is available on Github. The repository's main directory includes a **Builds** folder including builds for Windows and Android.

3 Libraries and Frameworks Used

- **Flutter** - A UI framework for cross-platform mobile application development, built using the Dart programming language.
- **Dart** - The primary programming language used for Flutter development, known for its efficiency and fast performance.

- **Dio** - A powerful and flexible HTTP client for handling API requests, supporting features such as interceptors, global configuration, and error handling.
- **Animated Background** - A package used to create dynamic and visually appealing animated backgrounds in the application. Used to create the moving PokeBalls background effect.
- **Google Fonts** - A package that enables the use of a wide variety of fonts from the Google Fonts library. Used to mimic fonts seen across Pokemon games.
- **Flutter Glow** - A package used to add glow effects to UI elements. It was used to Implement the glow of the top-left button.
-

4 Design Phase

The design phase of the application was conducted using Figma. Inspiration was drawn from the various **PokeDex** devices seen across various Pokemon games.

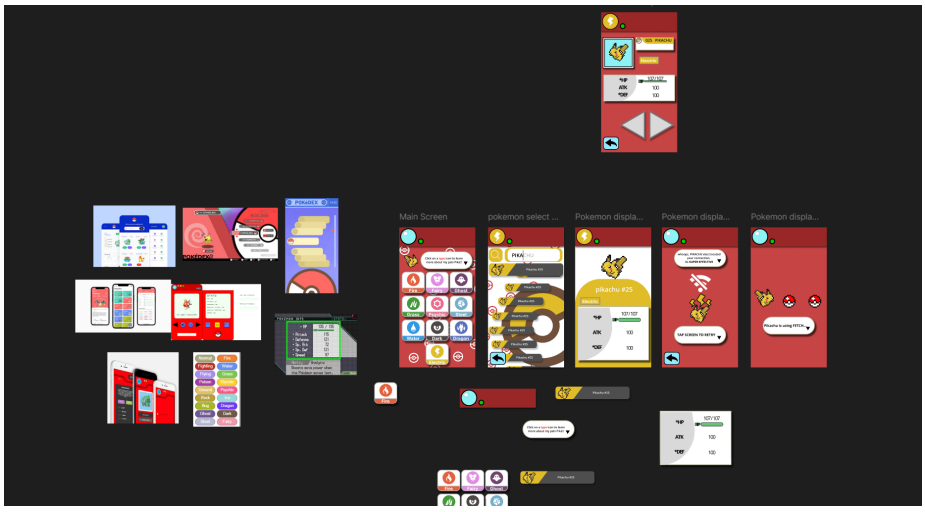

Figure 1: Figma designing of the Pokemon Explorer App.

5 Application Functionality

5.1 Initial Screen

The app supports multiple aspect ratios as can be observed below. As an artistic touch the Pokemon displayed on the top-left is different each time. The image is fetched randomly using the **PokeAPI**, corresponding code is presented below.

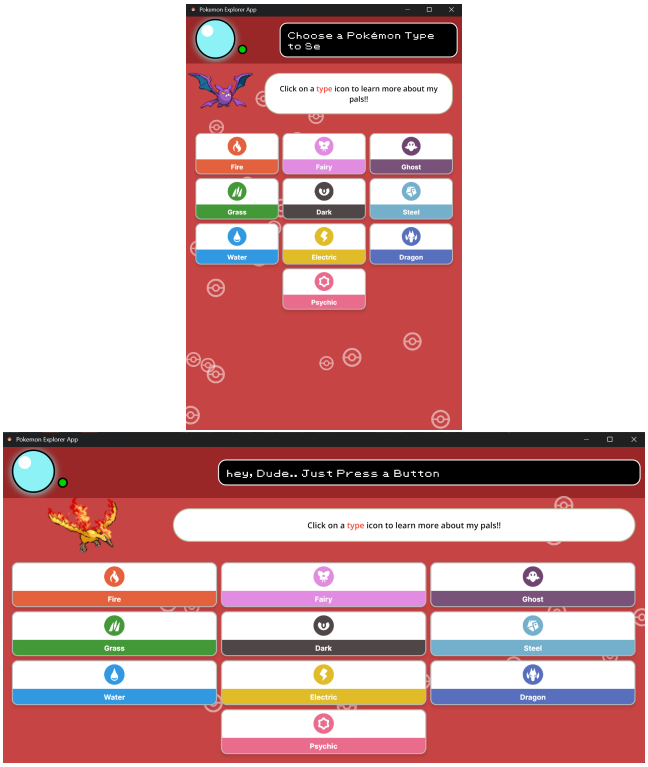

Figure 2: Initial screen in different aspect ratios.

Listing 1: Dart code for fetching random Pokemon

```
Future<String> fetchRandomPokemon() async {
  final randomId = Random().nextInt(898) + 1;
  final response = await Dio().get('https://pokeapi.co/api/v2/
      pokemon/$randomId');

  if (response.statusCode == 200) {
    return response.data['sprites']['front_default'];
  } else {
    throw Exception('Failed to load Pokemon');
  }
}
```

5.2 Listing Pokemon of a Specific Type, Search Functionality

After clicking to a **Type** button, the user is presented with a scroll-able list of Pokemon of the chosen type. Initially only 10 Pokemon are loaded, when the user reaches the border of the list, it expands with 10 more. A brief description of the Pokemon in the center is displayed below the list. The user can search for a Pokemon by it's name or PokeDex ID. Tapping on a Pokemon entry opens a more detailed screen.

Figure 3: Pokemon displayed by type and search functionality.

Listing 2: Function to fetch Pokémon entry data

```
static Future<Pokemon?> fetchPokemonData(BuildContext context,
    String name) async {
    final response = await _safeRequest(context, 'pokemon/$name');

    if (response != null) {
      final speciesResponse = await _safeRequest(context, response.
          data['species']['url']);

      String description = "No␣description␣available.";
      if (speciesResponse != null) {
        for (var entry in speciesResponse.data['flavor_text_entries
            ']) {
          if (entry['language']['name'] == 'en') {
            description = entry['flavor_text']
                .replaceAll('\n', ' ')
                .replaceAll('\f', ' ');
            break;
          }
        }
      }



      return Pokemon( \\Pokemon class to keep data in a stuctured
          format
        id: response.data['id'],
        name: response.data['name'],
        spriteUrl: response.data['sprites']['front_default'],
        types: List<String>.from(response.data['types'].map((t) =>
            t['type']['name'])),
        description: description,
        hp: response.data['stats'][0]['base_stat'],
        attack: response.data['stats'][1]['base_stat'],
        defense: response.data['stats'][2]['base_stat'],
      );
    }
    return null;
}
```

Listing 3: Dart code for fetching Pokemon description

```
Future<String> fetchPokemonDescription(String name) async {
  final response = await Dio().get('https://pokeapi.co/api/v2/
      pokemon-species/$name');

  if (response.statusCode == 200) {
    return (response.data['flavor_text_entries'] as List)
      .firstWhere((entry) => entry['language']['name'] == 'en')['
          flavor_text'];
  } else {
    throw Exception('Failed to fetch description');
  }
}
```

Listing 4: Dart code for fetching all Pokemon names belonging to a specific type, along with their PokeAPI links (PokeAPI does not support SQL so all pokemon all names had to be fetched beforehand)

```
Future<List<String>> fetchPokemonByType(String type) async {
  final response = await Dio().get('https://pokeapi.co/api/v2/type/
      $type');

  if (response.statusCode == 200) {
    return (response.data['pokemon'] as List)
      .map((p) => p['pokemon']['name'] as String)
      .toList();
  } else {
    throw Exception('Failed to fetch Pokemon by type');
  }
}
```

5.3 Displaying Pokemon Details

Once a Pokemon is selected, its details are displayed.

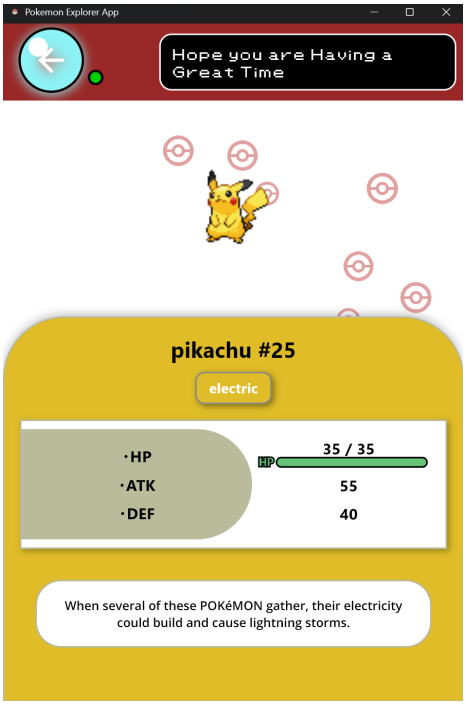

Figure 4: Pokemon display screen.

5.4 Hints

The upper-right display shows hints (or random remarks) relative to the page the user currently is.

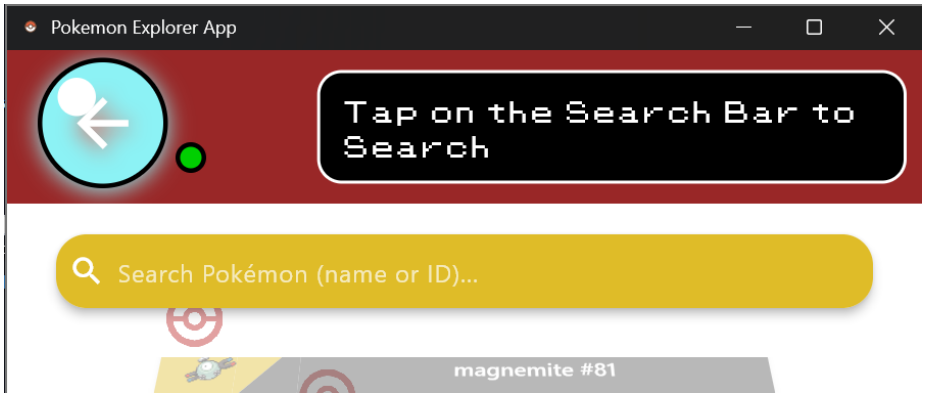

Figure 5: Hint screen.