Pontificia Universidad Católica Madre y Maestra

CSTI-1870-5238 - Desarrollo Móvil

Tema:

Avance Pokédex - Desarrollo Móvil

Integrantes:

1014-3525 Randae Garcia Sanchez

1014-3611 Starlin Frías Luna

Profesor:

Freddy Peña

Santiago de los Caballeros, República Dominicana

Miércoles 13 de Noviembre del 2024

Introducción:

En el presente documento se encuentra la documentación con respecto al proyecto final de desarrollo móvil, el cual consiste en la implementación de una Pokédex utilizando el framework de Flutter. Este mismo debe ser desarrollado con el uso de las mejores prácticas, además, con una excelente calidad de código, sin mencionar la importancia de un buen diseño tanto en lo visual, como en el backend para lograr mayor fluidez y experiencia de los usuarios. El proyecto se desarrolló utilizando técnicas como Graphql para consultar el API de Poké API la cual contiene la información de miles de pokemones junto con sus evoluciones y cualquier información útil de los mismos. Esta es la consola utilizada para las consultas: https://beta.pokeapi.co/graphql/console/.

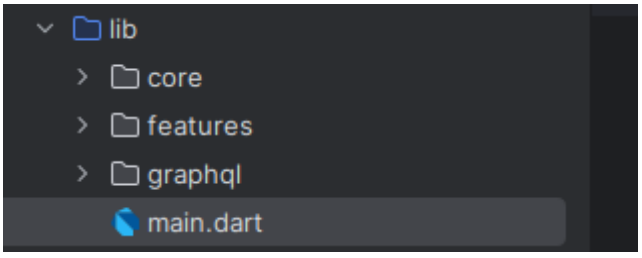
Diseño:

El diseño que utilizamos fue con el objetivo de separar y modular cada componente y widget lo más posible, todo con el propósito de lograr la reutilización de los mismos para así evitar las duplicaciones de código innecesario que lo único que hacen es complicar y dificultar la escalabilidad del proyecto. Por ello, creamos un archivo.dart por cada widget, además, del uso de diferentes modelos para pasear los datos del JSON obtenido del API a estos, así logramos un fácil acceso a los datos, y un manejo de una forma más organizada.

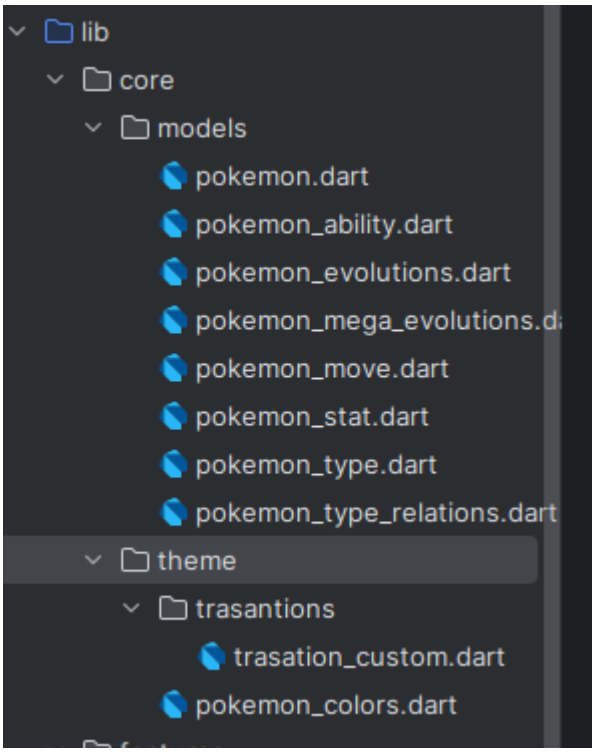
Estructura:

El proyecto cuenta con la siguiente estructura de archivos y directorios, esta misma facilita la integración de nuevos widgets, como además la escalabilidad a nuevas características que se deseen añadir, sin mencionar el fácil entendimiento para personas ajenas al código

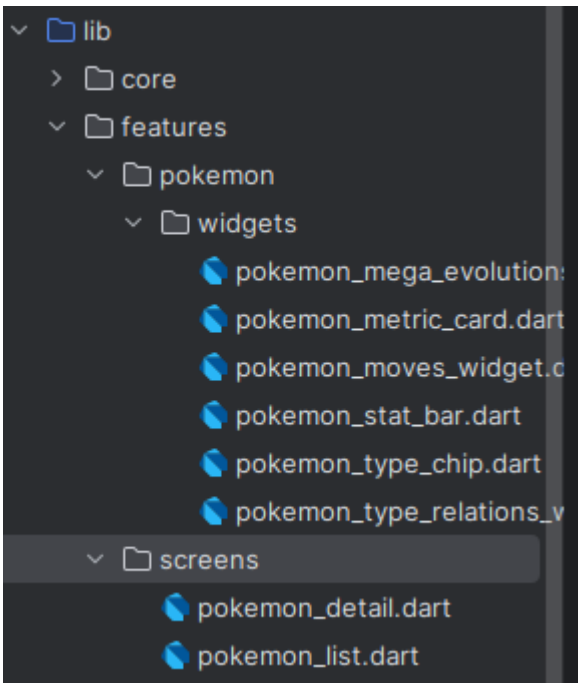
En primer lugar contamos con 3 directorios principales y el archivo main.dart dentro de lib, los cuales se visualizan de la siguiente manera:



En el directorio de core tenemos 2 directorios mas que son models y themes, el primero cuenta con los diferentes modelos que se utilizan para encapsular los datos recibidos en el JSON desde el API, el segundo por otro lado contiene las transacciones para los cambios de pantalla y los colores para los diferentes tipos de Pokemones. De esta forma se visualiza:

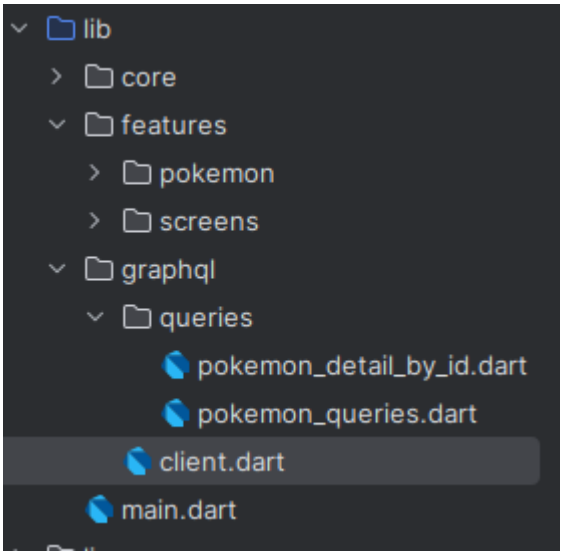


En el directorio principal de features contamos con un directorio llamado “Pokemon” el cual tiene otro llamado “widgets” el cual tiene dentro los diferentes componentes para ser utilizados tanto en la pantalla de detalle como en la lista en general con respecto a los Pokemones. Además, aquí dentro de este directorio llamado “features” se tiene otra carpeta llamada “screens” la cual contiene las diferentes pantallas a mostrar al usuario, en este caso la lista y la pantalla de detalle.



Por último, tenemos el directorio de “graphql” la cual contiene una carpeta llamada “queries” en esta se encuentran las diferentes consulta a la API de: “https://beta.pokeapi.co/graphql/v1beta”

De forma directa contamos con un archivo llamado “client.dart” el cual como su nombre lo indica es la creación del cliente para realizar las peticiones a la API.



Funcionamiento de la aplicación:

Uso de la API GraphQL:

En esta sección se explicará el uso de la API con graphql en el proyecto, desde como funciona a cómo se realiza. Esta documentación cubre la implementación de GraphQL que se conecta con el endpoint GraphQL Beta de PokéAPI. La aplicación utiliza el paquete graphql_flutter para gestionar las operaciones GraphQL.

- **Configuración del Cliente GraphQL**

  La aplicación inicializa un cliente GraphQL en `client.dart`:

```dart
import 'package:flutter/material.dart';
import 'package:graphql_flutter/graphql_flutter.dart';

ValueNotifier<GraphQLClient> setupGraphQLClient() {
  final HttpLink httpLink = HttpLink(
    'https://beta.pokeapi.co/graphql/v1beta',
  );

  return ValueNotifier(
    GraphQLClient(
      link: httpLink,
      cache: GraphQLCache(store: InMemoryStore()),
    ),  // GraphQLClient
  );  // ValueNotifier
}
```

  Utiliza HttpLink para conectarse al endpoint GraphQL de PokéAPI. Implementa caché en memoria con GraphQLCache para lograr mayor velocidad. Devuelve un cliente envuelto en ValueNotifier para actualizaciones reactivas.

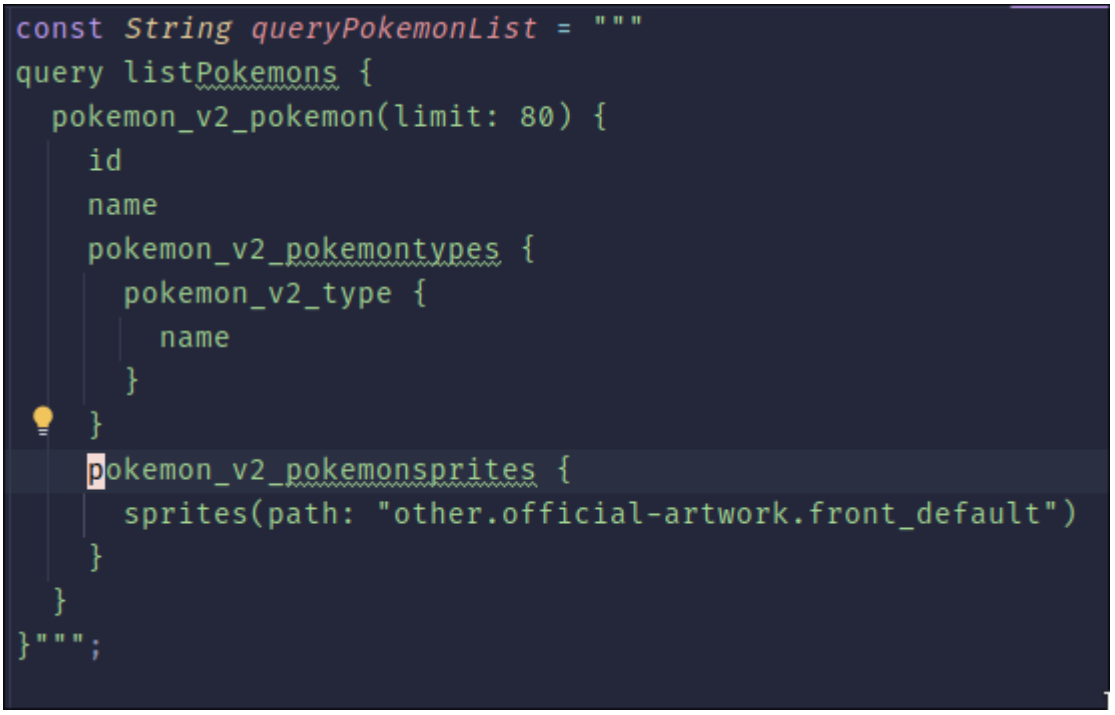
- **Integración en la Aplicación**

  El cliente GraphQL se integra en la raíz de la aplicación en main.dart:

```dart
import 'package:flutter/material.dart';
import 'package:graphql_flutter/graphql_flutter.dart';
import 'graphql/client.dart';
import 'features/screens/home_page.dart';

void main() {
  runApp(const MyApp());
}

class MyApp extends StatelessWidget {
  const MyApp({super.key});

  @override
  Widget build(BuildContext context) {
    return GraphQLProvider(
      client: setupGraphQLClient(),

      child: MaterialApp(
        debugShowCheckedModeBanner: false,
        title: "Pokedex Flutter",
        theme: ThemeData(
            primarySwatch: Colors.red
        ),  // ThemeData

        home: const HomePage(),
      ),  // MaterialApp
    );  // GraphQLProvider
  }
}
```

  El widget GraphQLProvider hace que el cliente esté disponible en todo el árbol de widgets.

La consulta para listar a los pokemones se encuentra en el archivo “”. Esta se hace lo más concisa posible para evitar overfetching y maximizar la eficiencia, aunque cabe destacar que la complejidad de la consulta aumentará a lo largo del desarrollo de la aplicación.



```
const String queryPokemonList = """
query listPokemons {
  pokemon_v2_pokemon(limit: 80) {
    id
    name
    pokemon_v2_pokemontypes {
      pokemon_v2_type {
        name
      }
    }
    pokemon_v2_pokemonsprites {
      sprites(path: "other.official-artwork.front_default")
    }
  }
}""";
```

La consulta para mostrar los detalles de un pokemon se encuentra en el archivo “”. No se mostrará en este documento porque, al ser una consulta que consigue mucha información de un pokemon y de sus evoluciones, resulta muy extensa.

A medida que se avance con el proyecto, tomaremos en cuenta si es necesario crear diferentes consultas para aplicar distintos filtros o continuaremos expandiendo las ya existentes aumentando su versatilidad.