

POKEPEDIA



DOCUMENTACIÓN DEL PROYECTO FINAL

Andrea Lozano Mercado

2º Desarrollo de Aplicaciones Multiplataforma

Programación Multimedia y Dispositivos Móviles

ÍNDICE

1. MANUAL DE USUARIO	1
1.1 Entrada a la Aplicación	1
1.2 Navegación	2
1.3 Página de Favoritos	3
1.4 Filtro tipos de Pokémon	5
1.5 Filtro de Generaciones	7
1.6 Detalles de pokémon	8
2. Documentación Técnica	12
2.1 Explicación General	11
2.2 Estructura de Carpetas	12
3. Aspectos de Mejora.....	22
Bibliografía.....	24

1. MANUAL DE USUARIO

Se presenta a continuación el manual de uso para el usuario de la aplicación.

1.1 ENTRADA A LA APLICACIÓN

La entrada a la aplicación se basa en una página donde se puede explorar pokémons disponiéndolos en una estructura denominada “scroll infinito”, según la cual se cargarán elementos conforme se descienda en la página, seleccionar un Pokémon para verlo en detalle, o bien buscar alguno concreto por nombre o tipo en la barra de búsqueda.

En la parte superior hay un icono de luna, a través de la cual se puede cambiar la aplicación de modo claro a modo oscuro y viceversa.

Por último, en la parte inferior se dispone de un drawer de navegación entre las distintas páginas de la aplicación.



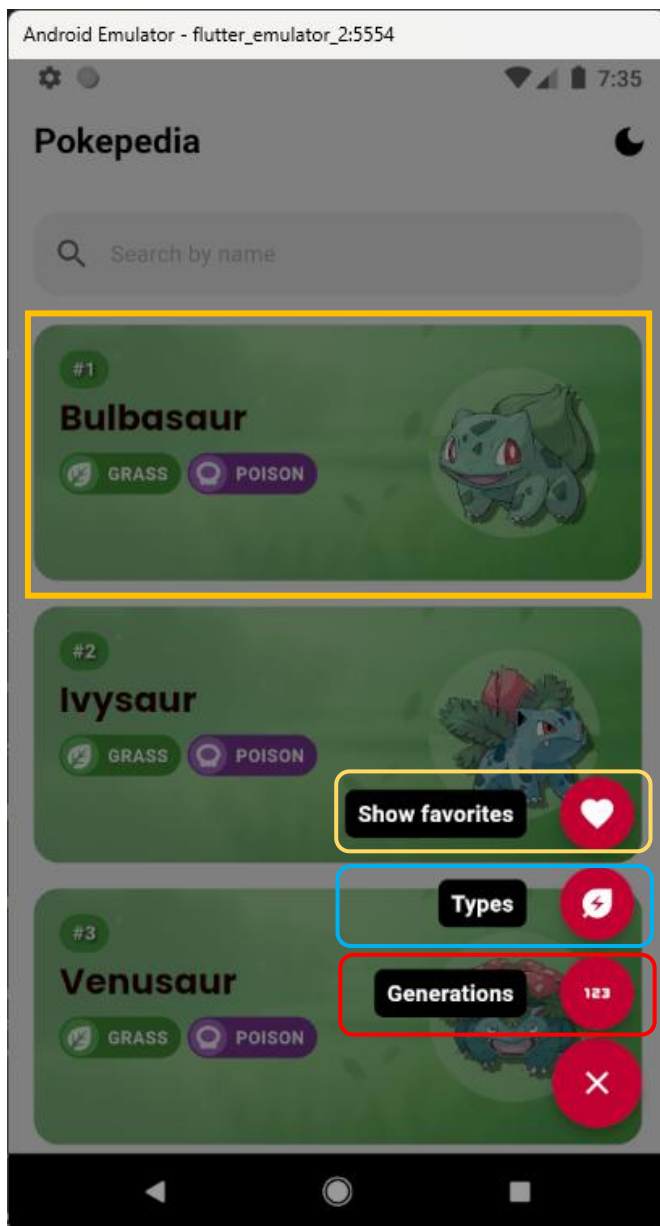
1.2 NAVEGACIÓN

La navegación en la aplicación se basa en un Widget denominado drawer.

En dicho drawer se dispone de los distintos enlaces de navegación.

Se disponen de, principalmente, tres opciones:

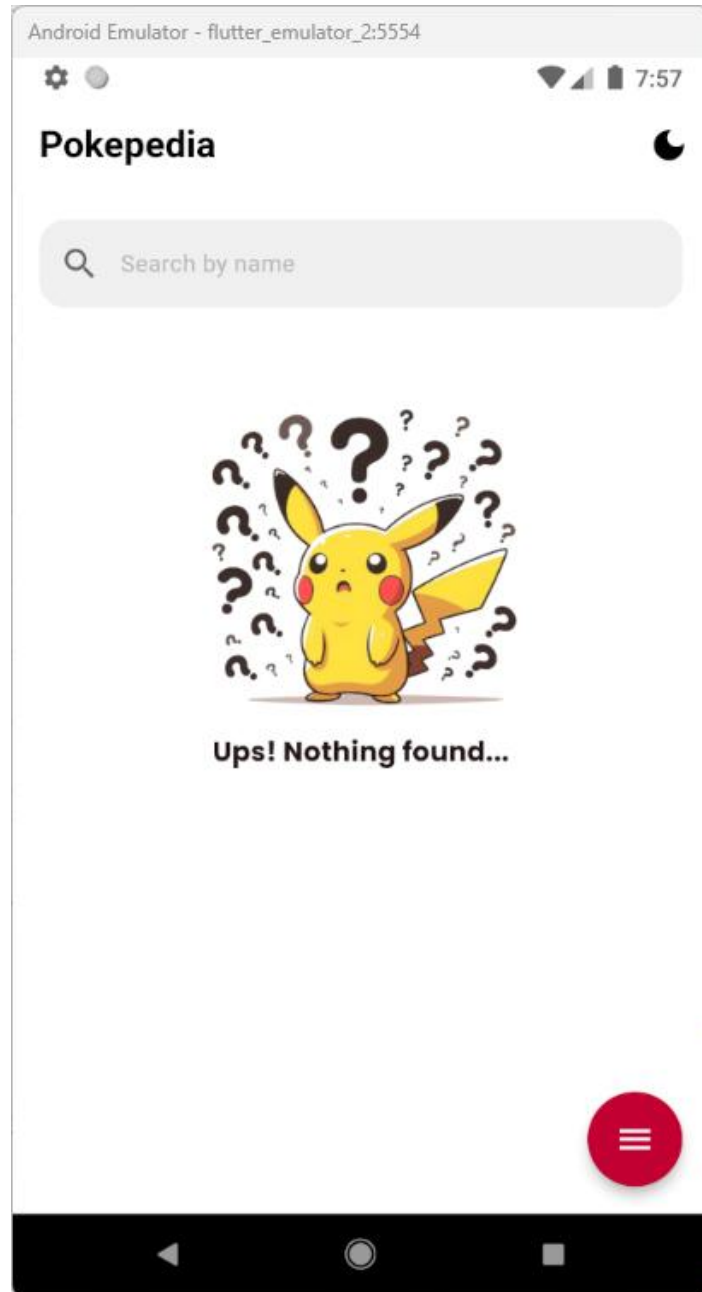
- Página de favoritos: Se trata de la página en la que se pueden ver los Pokémons favoritos añadidos.
- Filtro Tipos de Pokémon: En esta parte podremos filtrar un Pokémon por su tipo. Es decir, si seleccionamos tipo "Ghost" aparecerán todos los Pokémons de ese tipo.
- Filtro de Generaciones: Podremos filtrar por las diferentes generaciones de Pokémon existentes, desde la primera generación hasta la novena.



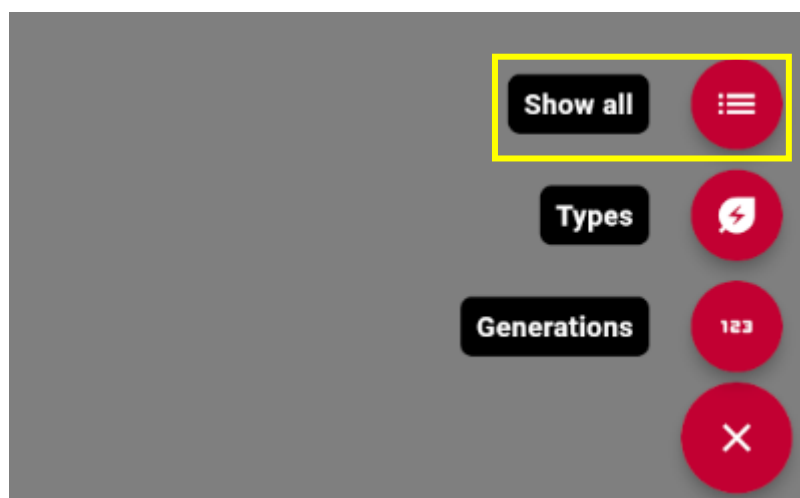
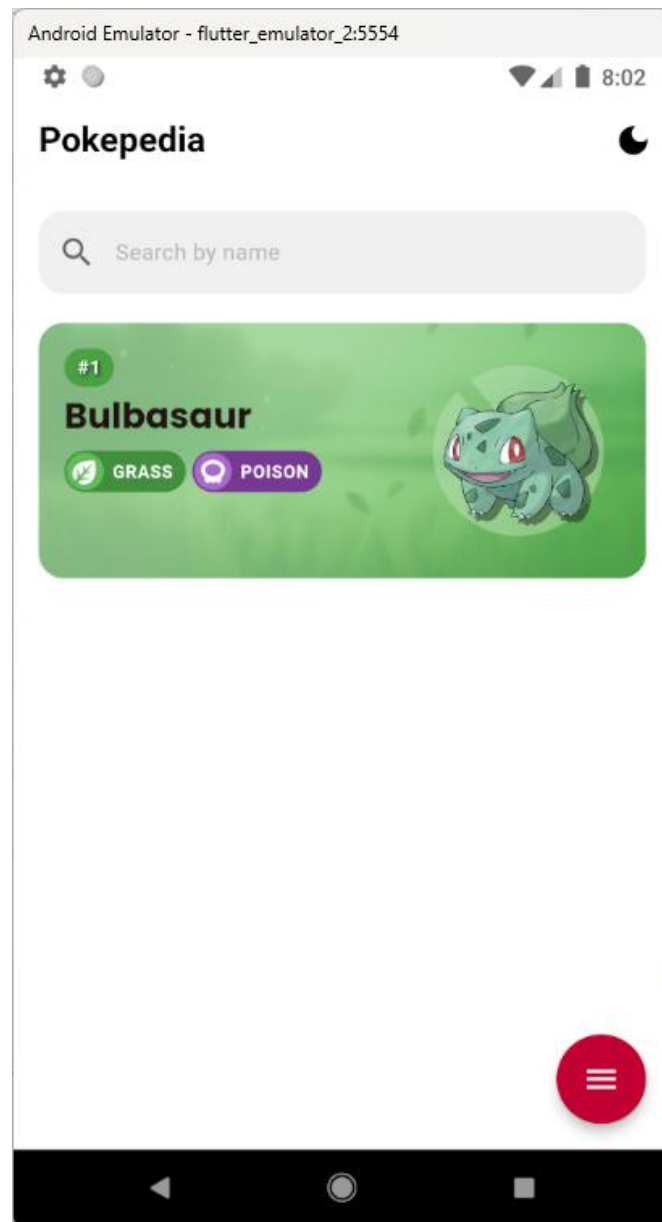
Además de estas opciones, de forma externa al drawer, se puede acceder a los detalles de un Pokémon pulsando en la tarjeta (recuadro naranja).

1.3 PÁGINA DE FAVORITOS

Inicialmente, la carga de la página de exploración contendrá únicamente un buscador (donde se podrán buscar tanto Pokémons por nombre como por tipo) y un mensaje de que actualmente no se han encontrado resultados. Esto se debe a que el usuario aún no ha añadido ningún Pokémon a sus favoritos.

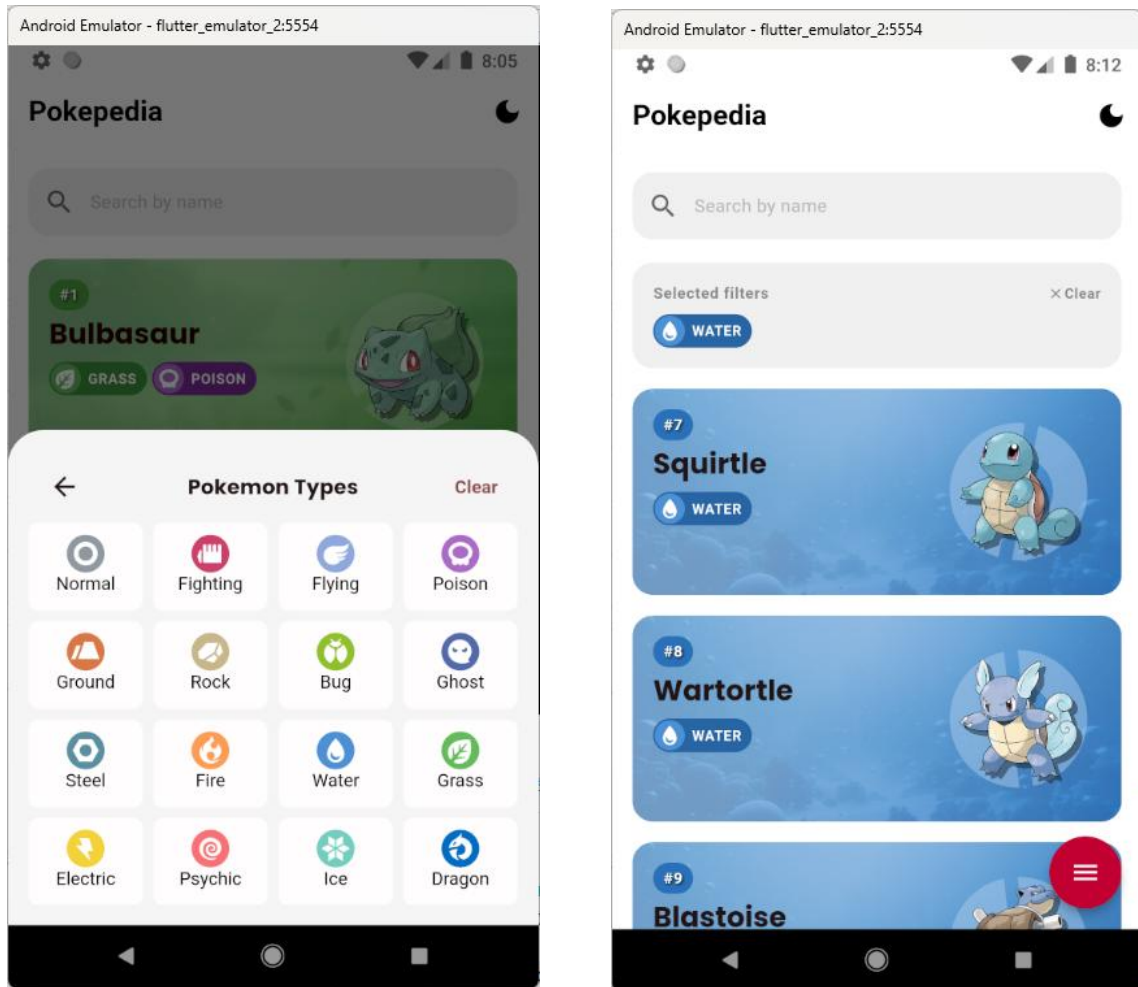


Una vez se marque un Pokémon como favorito, este aparecerá en dicha página. En esta página el drawer cambiará la opción de mostrar favoritos por mostrar todos.



1.4 FILTRO TIPOS DE POKÉMON

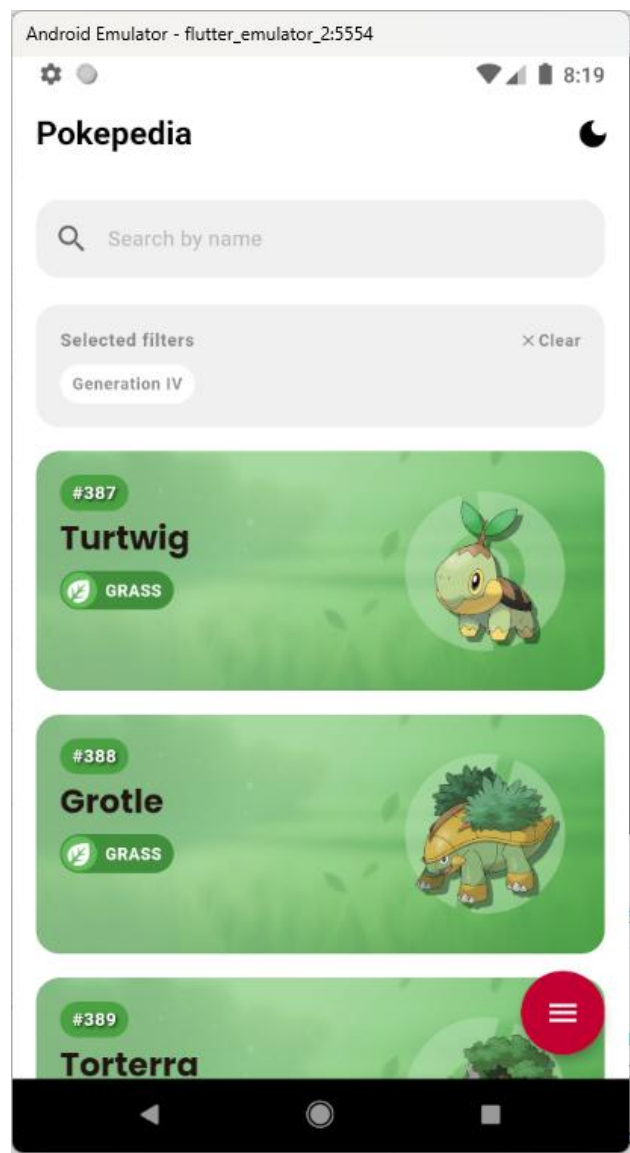
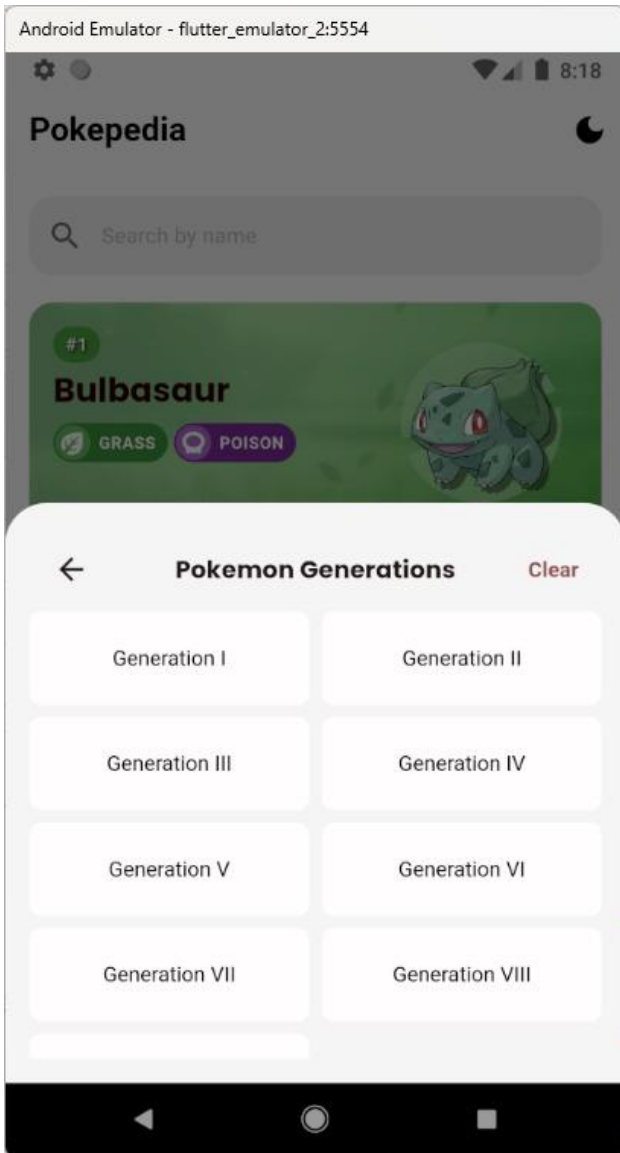
En esta pestaña se pueden observar todos los tipos de Pokémons existentes. Además, se dispondrá de una opción de limpiar el filtro y otra de volver a la pantalla inicial.



Se puede seleccionar más de un tipo para filtrar los diferentes Pokémons. Una vez elegido el/los filtro/s aparecerá una pantalla como la de la segunda imagen, donde se podrá limpiar el filtro, añadir otro e incluso buscar por nombre o inicial.

1.5 FILTRO DE GENERACIONES

Debido a la funcionalidad de la aplicación, tanto la estructura de la página como su funcionalidad es extremadamente parecida a la anterior. Sin embargo, en este caso se podrá filtrar por generación:

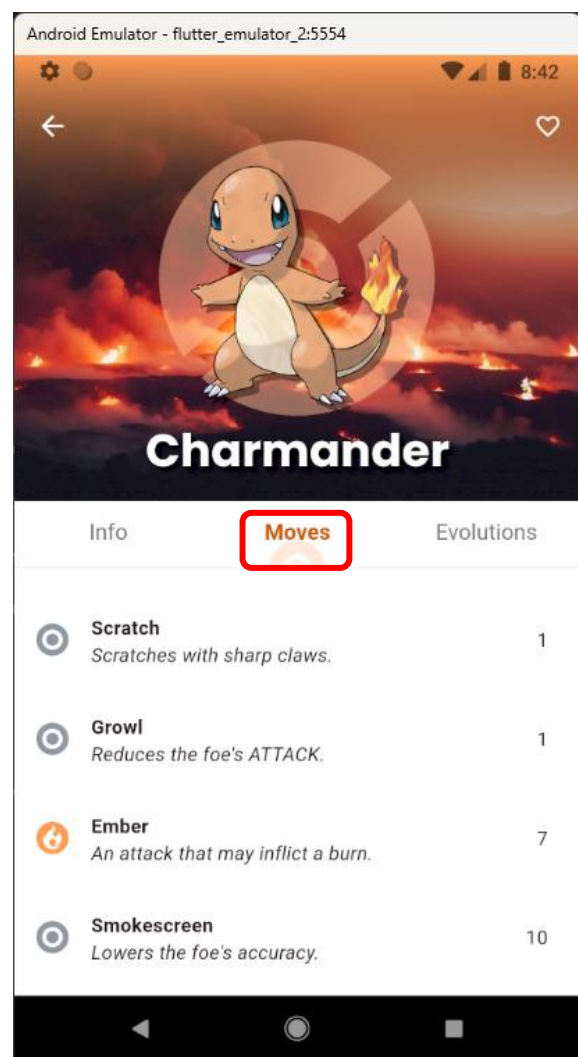
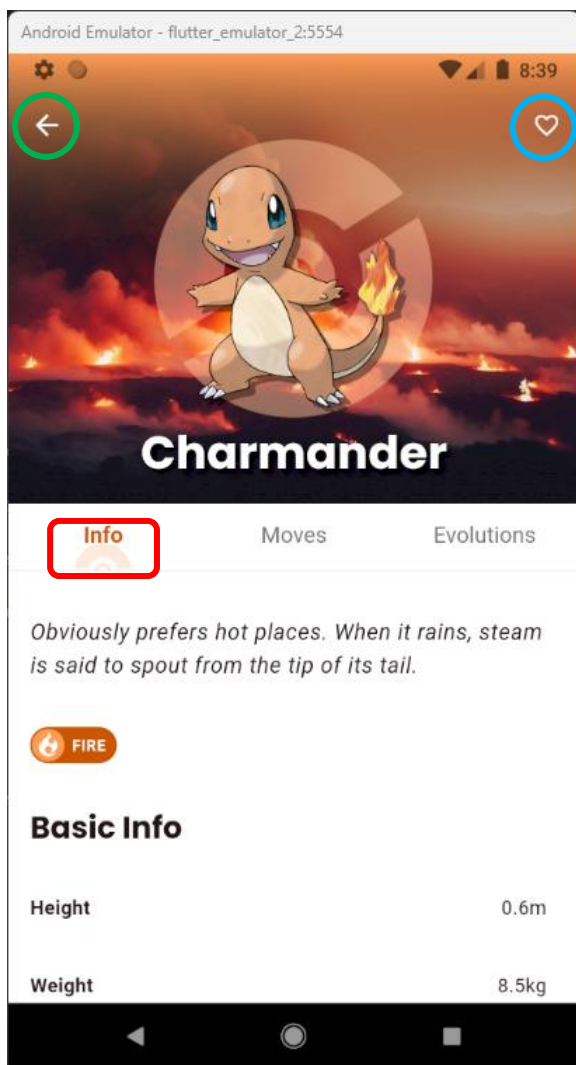


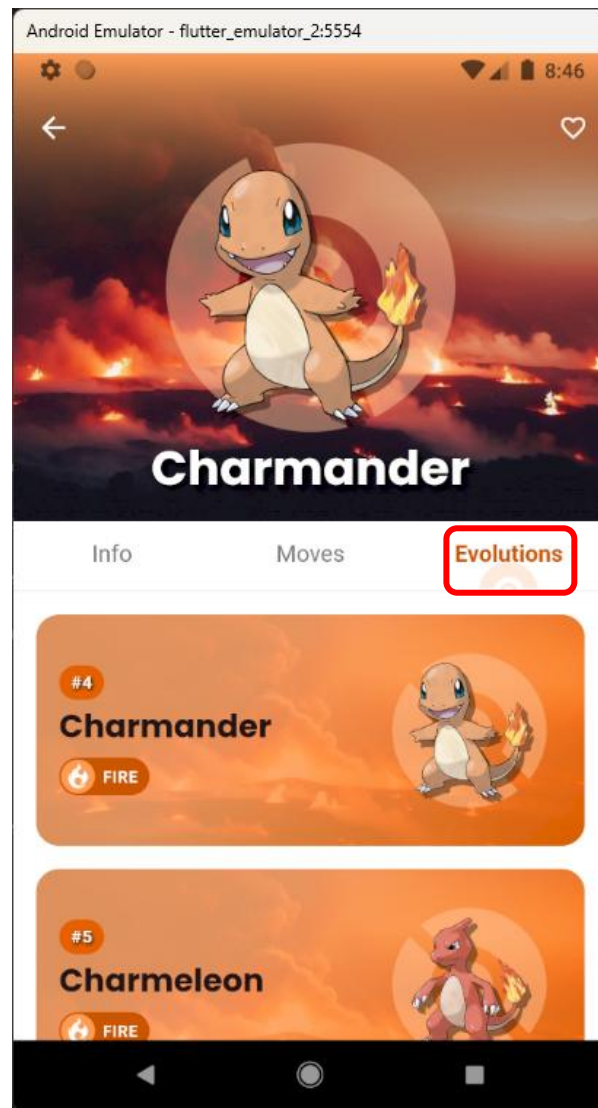
1.6 DETALLES DE POKÉMON

Se muestra a continuación la página de detalles de un Pokémon, la cual se divide en tres secciones principalmente a las que se puede acceder mediante un slider.

Además es donde se puede marcar (o desmarcar) a un Pokémon como favorito y volver a la pantalla anterior:

- Info: Se trata de la sección de donde se hace referencia a toda la información del Pokémon: tipo/s, información básica, efectividad de ataques (a favor o en contra), estadísticas de ataque, defensa, velocidad, etc., y las habilidades que posee.
- Moves: Sección donde se describen la potencia de los ataques y lo que puede producir en el Pokémon atacado.
- Evolutions: Sección donde se hace referencia a todas las evoluciones de dicho Pokémon, con la opción de acceder también a los detalles de los mismos.





2. DOCUMENTACIÓN TÉCNICA

Esta sección de la documentación estará dedicada a la parte técnica del proyecto, en la cual se explicará a rasgos generales el funcionamiento de la aplicación.

2.1 EXPLICACIÓN GENERAL

Pokepedia es una aplicación móvil desarrollada que permite a los usuarios explorar información detallada sobre los Pokémon. La app usa GraphQL para obtener datos de Pokémon en lugar de la API REST de PokeAPI, lo que mejora la eficiencia y evita problemas de over-fetching.

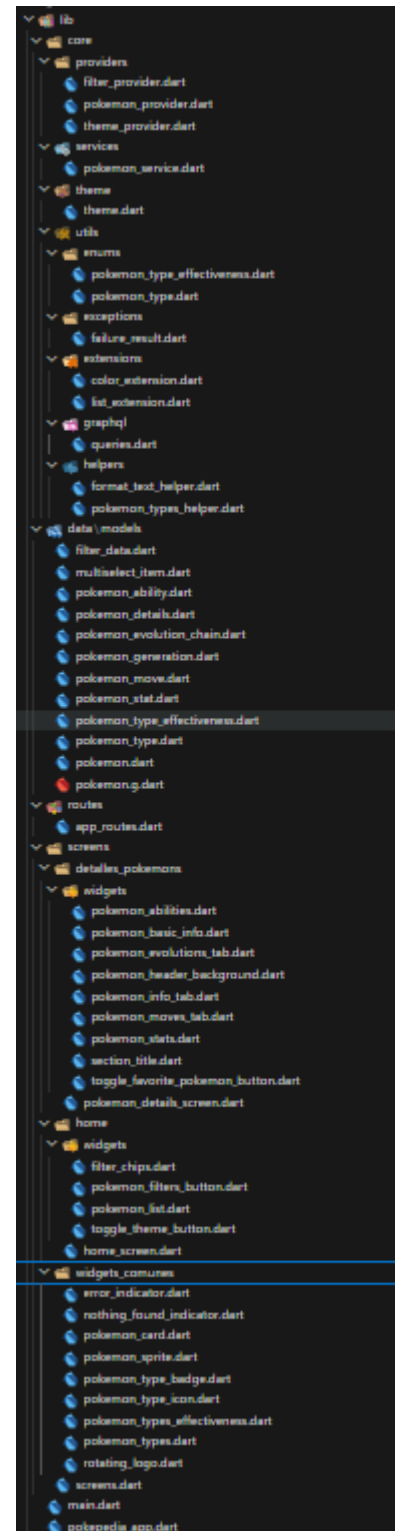
La aplicación Pokepedia se trata, en pocas palabras, de una aplicación desarrollada en Flutter. Dicha aplicación ha necesitado de:

- Una API externa de la cual se han extraído los datos acerca de los libros (POkeAPI).
- Una serie de bibliotecas adaptadas a un uso desde flutter, entre las cuales cabe destacar Hive, la cual está destinada al trabajo con bases de datos.

2.2 ESTRUCTURA DE CARPETAS

La estructura de carpetas del proyecto es la siguiente:

- Dentro de Core:
 - Providers: Se trata de una carpeta que contendrá todos los proveedores de estado.
 - Theme: Dónde se definen los temas claro/oscur.
 - Utils: En ella están todos los Helpers, excepciones, extensiones, las queries y utilidades generales.
- Data: Carpeta destinada a los modelos de datos y al archivo generado automáticamente por Hive.
- Screens: Dividida a su vez en otras subcarpetas para garantizar la modularidad, en ella hay clases y widgets con las estructuras principales de las distintas páginas de la aplicación.
- Widgets comunes: Conjunto de clases con diferentes widgets, distribuidos por widgets comunes de cada página.
- Clase main: Inicialización de Hive y proveedores.
- Clase PokepediaApp: Punto de entrada principal de la app.



2.2.1 CLASE MAIN

La clase main será el punto de entrada a la aplicación, y desde la cual se configura la inicialización de la aplicación y gestiona configuraciones previas como la restricción de la orientación y la inicialización de bases de datos

Será, por tanto, la entrada y el nexo de unión entre los estilos, la navegación y las distintas pantallas.

```
void main() async {
  WidgetsFlutterBinding.ensureInitialized();

  await SystemChrome.setPreferredOrientations([
    DeviceOrientation.portraitUp // Restringe la orientación a vertical
  ]);

  final appDir = await getApplicationDocumentsDirectory(); // Obtener directorio de documentos de la aplicación

  await Hive.initFlutter(appDir.path); // Inicializa Hive con la carpeta de documentos

  Hive.registerAdapter(PokemonAdapter()); // Registra el adaptador de Hive

  await Hive.openBox<Pokemon>('favorite_pokemons'); // Abre una caja de Hive

  // You, ayer * Arreglo de modelos + nuevos modelos + providers

  runApp(
    // Se utiliza MultiProvider para gestionar el estado de múltiples proveedores
    MultiProvider(
      providers: [
        ChangeNotifierProvider(create: (_) => ThemeProvider()), // Tema
        ChangeNotifierProvider(create: (_) => PokemonProvider()), // Pokémon
        ChangeNotifierProvider(create: (_) => FilterProvider()), // Filtros
      ],
      child: const PokepediaApp()
    ), // MultiProvider
  );
}
```

2.2.2 CORE

Esta carpeta contiene los componentes esenciales del proyecto, como los providers y servicios.

2.2.2.1 PokepediaAppTheme (theme.dart)

Este archivo define los temas de la aplicación, incluyendo tanto el tema claro como el oscuro.

- Colores principales: Se define un color primario y se generan variaciones para su uso en diferentes elementos de la UI.
- Fuentes: Se utiliza la tipografía Poppins, importada a través de Google Fonts.
- Estilos de texto: Se configuran distintos estilos para títulos, subtítulos y etiquetas.
- Tema claro: Define el color de fondo, los colores primarios y secundarios, el estilo de los botones y otros componentes.
- Tema oscuro: Ajusta la paleta de colores para entornos oscuros, asegurando buen contraste y

```
class PokepediaAppTheme {
  static const primaryColor = Color(0xFFC20032);

  static final materialColor = primaryColor.toMaterialColor();

  static const poppinsTextStyle = TextStyle(fontFamily: 'Poppins');

  static final titleStyle = poppinsTextStyle.copyWith(
    fontWeight: FontWeight.bold
  );

  // Define un estilo de texto para los títulos en negrita
  static const labelStyle = TextStyle(
    color: Colors.black45,
    fontWeight: FontWeight.bold
  );

  // Define los estilos de texto para diferentes elementos de la UI
  static final textTheme = TextTheme(
    headlineLarge: poppinsTextStyle,
    headlineMedium: poppinsTextStyle,
    headlineSmall: poppinsTextStyle,
    titleLarge: titleStyle,
    titleMedium: titleStyle,
    titleSmall: titleStyle,
    labelLarge: labelStyle,
    labelMedium: labelStyle,
    labelSmall: labelStyle
  );

  /// TEMA CLARO
  static final lightTheme = ThemeData(
```

legibilidad.

Los temas están diseñados para cambiar dinámicamente según la configuración del usuario.

2.2.2.2 Carpeta Providers

Esta carpeta contiene lo siguiente:

- ThemeProvider (theme.dart): Gestiona el tema claro y oscuro de la aplicación.
- FilterProvider: Maneja los filtros aplicados en la búsqueda de Pokémon.
- PokemonProvider: Administra la lista de Pokémon obtenidos de la API y los favoritos almacenados en Hive.

Estos providers utilizan ChangeNotifier y Provider para actualizar la interfaz de usuario de

```

class ThemeProvider with ChangeNotifier {
  static const sharedPreferencesKey = 'theme_mode';

  // Establecer el modo de tema por defecto como oscuro
  ThemeMode mode = ThemeMode.dark;

  // Getter para saber si el modo actual es oscuro
  bool get isDarkMode {
    return mode == ThemeMode.dark;
  }

  // Cargar el tema desde SharedPreferences
  void loadTheme() async {
    final prefs = await SharedPreferences.getInstance();
    var themeMode = prefs.getString(sharedPreferencesKey);

    if (themeMode != null) {
      mode = themeMode == 'light' ? ThemeMode.light : ThemeMode.dark;
    } else {
      mode = ThemeMode.light;
    }

    notifyListeners();
  }

  // Cambiar el modo de tema y guardarlo en SharedPreferences
  void setMode(ThemeMode themeMode) async {
    final prefs = await SharedPreferences.getInstance();
    await prefs.setString(sharedPreferencesKey, themeMode.name);
    mode = themeMode;
    notifyListeners();
  }
}

class FilterProvider with ChangeNotifier {
  // Variables para los filtros
  bool _showFavoritesOnly = false;
  String _searchText = '';
  List<PokemonGeneration> _generations = [];
  List<PokemonType> _types = [];
  List<PokemonType> _selectedTypes = [];
  List<PokemonGeneration> _selectedGenerations = [];

  // Getters para acceder a las propiedades privadas
  List<PokemonGeneration> get selectedGenerations => _selectedGenerations;
  List<PokemonType> get selectedTypes => _selectedTypes;
  bool get showFavoritesOnly => _showFavoritesOnly;

  // Método para obtener los datos de los filtros de la API
  Future<void> fetchFilterData() async {
    // Limpiar las listas de generaciones y tipos seleccionados
    _selectedGenerations.clear();
    _selectedTypes.clear();

    final (result, exception) = await PokemonService.fetchFilterData();

    // Manejo de errores si ocurre algún problema al obtener los filtros
    if (exception != null) {
      debugPrint("ERROR fetchFilterData ${exception.message}");
      return;
    }

    // Si obtenemos datos, asignamos las generaciones y tipos seleccionados
    if (result != null) {
      var (generations, types) = result;
      _selectedGenerations = generations;
      _selectedTypes = types;
      notifyListeners();
    }
  }
}

class PokemonProvider with ChangeNotifier {
  // Variables para gestionar los Pokémon
  final List<Pokemon> _pokemons = [];
  int _currentPage = 0;
  bool _isLoading = false;
  bool _hasException = false;
  final Box<Pokemon> _favoritesBox = Hive.box<Pokemon>('favorites');

  // Getters
  List<Pokemon> get favorites => _favoritesBox.value;
  bool get hasException => _hasException;
  bool get isLoading => _isLoading;
  int get currentPage => _currentPage;
  List<Pokemon> get pokemons => _pokemons;

  // Método que obtiene una lista de Pokémon según la página y los filtros
  // Realiza la búsqueda de Pokémon a partir de los filtros seleccionados
  Future<void> fetchPokemons({
    required List<PokemonGeneration> generations,
    required List<PokemonType> pokemonTypes,
    String? searchText,
    int? page,
  }) async {
    if (page != null) {
      _currentPage = page;
    }

    // Si es la primera página, limpiamos la lista de Pokémon
    if (page == 0) {
      _pokemons.clear();
    }

    // Realiza la búsqueda de Pokémon a partir de los filtros seleccionados
    final (result, exception) = await PokemonService.fetchPokemons(
      generations: generations,
      pokemonTypes: pokemonTypes,
      searchText: searchText,
      page: page,
    );

    if (exception != null) {
      _hasException = true;
      _isLoading = false;
      notifyListeners();
      return;
    }

    if (result != null) {
      _pokemons.addAll(result.pokemons);
      _isLoading = false;
      notifyListeners();
    }
  }
}

```

forma reactiva.

2.2.2.3 Carpeta Services

Contiene PokemonService que configura la conexión con la API de GraphQL y ejecuta las

consultas.

```
class PokemonService {
  static final _graphqlClient = GraphQLClient(
    link: HttpLink('https://beta.pokeapi.co/graphql/v1beta'),
    cache: GraphQLCache(),
  ); // GraphQLClient

  // Este método ejecuta cualquier consulta GraphQL y maneja errores
  static Future<GqlResponse> _executeQuery(QueryOptions queryOptions) async {
    try {
      final result = await _graphqlClient.query(queryOptions);

      if (result.hasException) {
        throw Exception(result.exception);
      }

      return (result, null);
    } on Exception catch (error) {
      return (null, error);
    }
  }

  // Ejecuta la consulta pokemonDetailQuery para obtener información detallada de un pokemon específico
  static Future<PokemonDetailsResponse> fetchDetails(int id) async {
    var response = await _executeQuery(QueryOptions(
      document: gql(pokemonDetailQuery),
      variables: {'id': id},
    ));

    var (result, exception) = response;
  }
}
```

2.2.3 CARPETA UTILS

La carpeta utils, aunque está dentro de core, es importante porque almacena diversas herramientas y funciones auxiliares utilizadas en toda la aplicación.

2.2.3.1 Carpeta Enums

Contiene enumeraciones utilizadas en la aplicación, como los tipos de Pokémon (PokemonTypeEnum) y la efectividad (PokemonTypeEffectivenessEnum).

```
enum PokemonTypeEffectivenessEnum {
  resistant, // Resistente a un tipo de ataque
  neutral, // Eficacia neutral
  vulnerable; // Vulnerable a un tipo de ataque

  // Método para analizar la eficacia de un tipo según un puntaje
  static PokemonTypeEffectivenessEnum parse(int score) {
    return switch (score) {
      > 0 => PokemonTypeEffectivenessEnum.resistant,
      < 0 => PokemonTypeEffectivenessEnum.vulnerable,
      _ => PokemonTypeEffectivenessEnum.neutral
    };
  }
}

// Obtener el puntaje asociado a cada tipo de eficacia
extension PokemonTypeEffectivenessEnumExtension on PokemonTypeEffectivenessEnum {
  int get score {
    return switch (this) {
      PokemonTypeEffectivenessEnum.resistant => 1,
      PokemonTypeEffectivenessEnum.neutral => 0,
      PokemonTypeEffectivenessEnum.vulnerable => -1
    };
  }
}

// Enum que representa los diferentes tipos de Pokemons en la app
enum PokemonTypeEnum {
  bug, dark, dragon, electric, fairy, fighting, fire, flying, ghost,
  ground, ice, normal, poison, psychic, rock, shadow, steel, stellar,

  // Método para convertir un string del tipo de Pokémon en un valor
  static PokemonTypeEnum parse(String pokemonType) {
    return switch (pokemonType) {
      'bug' => PokemonTypeEnum.bug,
      'dark' => PokemonTypeEnum.dark,
      'dragon' => PokemonTypeEnum.dragon,
      'electric' => PokemonTypeEnum.electric,
      'fairy' => PokemonTypeEnum.fairy,
      'fighting' => PokemonTypeEnum.fighting,
      'fire' => PokemonTypeEnum.fire,
      'flying' => PokemonTypeEnum.flying,
      'ghost' => PokemonTypeEnum.ghost,
      'grass' => PokemonTypeEnum.grass,
      'ground' => PokemonTypeEnum.ground,
      'ice' => PokemonTypeEnum.ice,
      'normal' => PokemonTypeEnum.normal,
      'poison' => PokemonTypeEnum.poison,
      'psychic' => PokemonTypeEnum.psychic,
      'rock' => PokemonTypeEnum.rock,
      'shadow' => PokemonTypeEnum.shadow,
      'steel' => PokemonTypeEnum.steel,
    };
  }
}
```

2.2.3.2 Carpeta Exceptions

Define excepciones personalizadas para manejar errores en las consultas GraphQL y otras operaciones.

2.2.3.3 Carpeta Extensions

Extensiones sobre clases nativas de Flutter y Dart para mejorar su funcionalidad. Ejemplos:

- ColorExtension: Métodos adicionales para manipulación de colores.

- ListExtension: Métodos para facilitar la manipulación de listas.

2.2.3.4 Carpeta GraphQL

Para obtener los datos de los Pokémon, la aplicación utiliza consultas GraphQL en lugar de REST. Esto evita el over-fetching y mejora el rendimiento.

- fetchFiltersDataQuery: Obtiene información sobre los tipos de Pokémon y generaciones disponibles.
- fetchPokemonsQuery: Recupera una lista de Pokémon con sus respectivos ID, nombre, tipos y sprites.
- pokemonDetailQuery: Obtiene detalles completos de un Pokémon, incluyendo estadísticas, habilidades y evoluciones.

```
const String fetchFiltersDataQuery = r'''
query GetFiltersData {
  types: pokemon_v2_type {
    id
    name
  }
  generations: pokemon_v2_generation {
    id
    name
  }
}
''';

// Recupera una lista de pokemons con sus respectivos ID, nombre, tipos y sprites
const String fetchPokemonsQuery = r'''
query GetPokemons($where: pokemon_v2_pokemon_bool_exp, $limit: Int, $offset: Int) {
  pokemon: pokemon_v2_pokemon(where: $where, limit: $limit, offset: $offset, order_by: {id: asc}) {
    id
    name
    types: pokemon_v2_pokemontypes {
      type: pokemon_v2_type {
        id
        name
      }
    }
    sprites: pokemon_v2_pokemonsprites {
      front_default: sprites(path: "other.official-artwork.front_default")
    }
  }
}
''';

// Obtiene detalles completos de un Pokémon por su id
const String pokemonDetailQuery = r'''
query GetPokemonDetails($id: Int!) {
  pokemon: pokemon_v2_pokemon_by_pk(id: $id) {
    id
    name
    height
    weight
    base experience
  }
}
```

El uso de GraphQL permite optimizar la carga de datos, ya que solo se solicitan los campos necesarios en cada consulta.

2.2.3.2 Carpeta Helpers

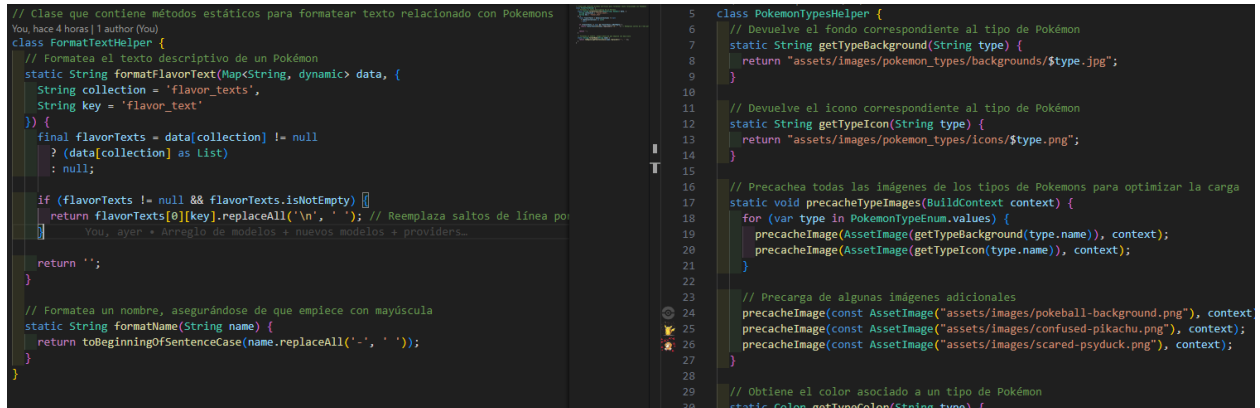
Contiene funciones de utilidad como:

- FormatTextHelper: Formatea textos como nombres de Pokémon y descripciones.

- `PokemonTypesHelper`: Proporciona colores y recursos gráficos para los tipos de Pokémon.

2.2.4 DATA

Contiene la carpeta `models` donde se definen modelos de datos para almacenar la información



```
// Clase que contiene métodos estáticos para formatear texto relacionado con Pokemons
You, hace 4 horas | 1 author (You)
class FormatTextHelper {
  // Formatea el texto descriptivo de un Pokémon
  static String formatFlavorText(Map<String, dynamic> data, {
    String collection = 'flavor_texts',
    String key = 'flavor_text'
  }) {
    final flavorTexts = data[collection] != null
      ? (data[collection] as List)
      : null;

    if (flavorTexts != null && flavorTexts.isNotEmpty) {
      return flavorTexts[0][key].replaceAll('\n', ' '); // Reemplaza saltos de línea por
    }

    return '';
  }

  // Formatea un nombre, asegurándose de que empiece con mayúscula
  static String formatName(String name) {
    return toBeginningOfSentenceCase(name.replaceAll('-', ' '));
  }
}

class PokemonTypesHelper {
  // Devuelve el fondo correspondiente al tipo de Pokémon
  static String getTypeBackground(String type) {
    return "assets/images/pokemon_types/backgrounds/$type.jpg";
  }

  // Devuelve el icono correspondiente al tipo de Pokémon
  static String getTypeIcon(String type) {
    return "assets/images/pokemon_types/icons/$type.png";
  }

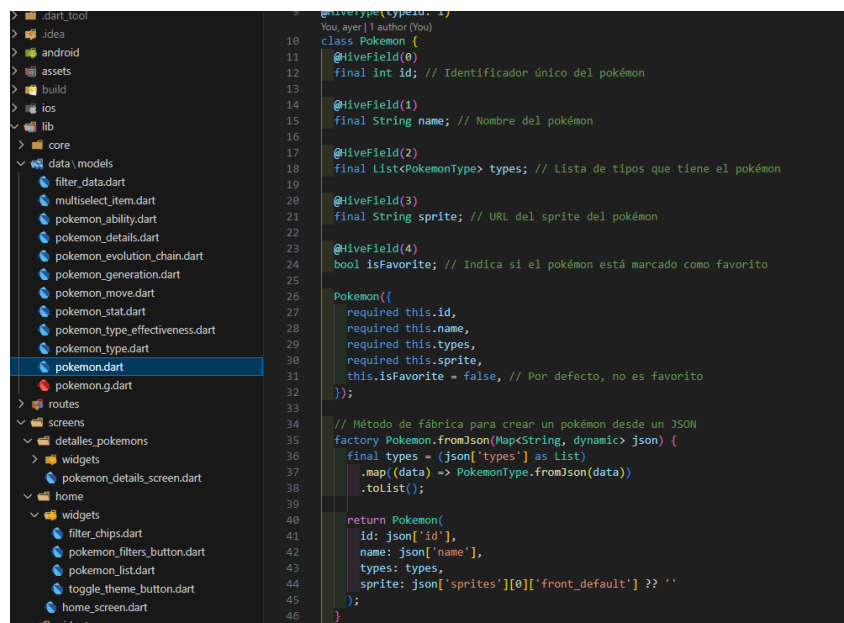
  // Precachea todas las imágenes de los tipos de Pokemons para optimizar la carga
  static void precacheTypeImages(BuildContext context) {
    for (var type in PokemonTypeEnum.values) {
      precacheImage(AssetImage(getTypeBackground(type.name)), context);
      precacheImage(AssetImage(getTypeIcon(type.name)), context);
    }
  }

  // Precarga de algunas imágenes adicionales
  precacheImage(const AssetImage("assets/images/pokeball-background.png"), context);
  precacheImage(const AssetImage("assets/images/confused-pikachu.png"), context);
  precacheImage(const AssetImage("assets/images/scared-psyduck.png"), context);
}

// Obtiene el color asociado a un tipo de Pokémon
static Color getTypeColor(String type) {
}
```

relevante de los Pokémon. El modelo principal es `Pokemon`, que contiene:

- ID: Identificador único del Pokémon.
- Nombre: Nombre del Pokémon.
- Tipos: Lista de tipos a los que pertenece el Pokémon.
- Sprite: Imagen del Pokémon.
- Favorito: Indica si el Pokémon está marcado como favorito.



```
@HiveType(typeId: 1)
You, ayer | 1 author (You)
class Pokemon {
  @HiveField(0)
  final int id; // Identificador único del pokémon

  @HiveField(1)
  final String name; // Nombre del pokémon

  @HiveField(2)
  final List<PokemonType> types; // Lista de tipos que tiene el pokémon

  @HiveField(3)
  final String sprite; // URL del sprite del pokémon

  @HiveField(4)
  bool isFavorite; // Indica si el pokémon está marcado como favorito

  Pokemon({
    required this.id,
    required this.name,
    required this.types,
    required this.sprite,
    this.isFavorite = false, // Por defecto, no es favorito
  });

  // Método de fábrica para crear un pokémon desde un JSON
  factory Pokemon.fromJson(Map<String, dynamic> json) {
    final types = (json['types'] as List)
      .map((data) => PokemonType.fromJson(data))
      .toList();

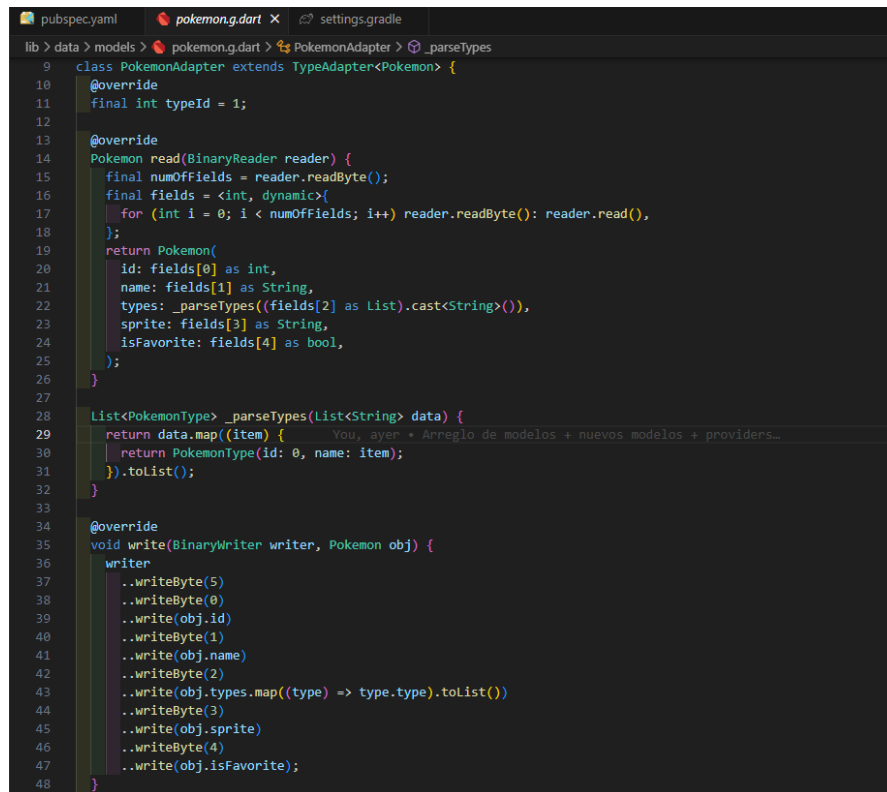
    return Pokemon(
      id: json['id'],
      name: json['name'],
      types: types,
      sprite: json['sprites'][0]['front_default'] ?? ''
    );
  }
}
```

El modelo está registrado en Hive a través de `PokemonAdapter`, lo que permite su almacenamiento y recuperación eficientes.

Hive se encarga de almacenar los Pokémon marcados como favoritos. Se utiliza la siguiente configuración:

- Apertura de la caja de almacenamiento: `Hive.openBox<Pokemon>('favorite_pokemons');`

- Añadir un Pokémon a favoritos: `_favoritesBox.put(pokemon.id, pokemon);`
- Eliminar un Pokémon de favoritos: `_favoritesBox.delete(pokemon.id);`
- Recuperar favoritos: `List<Pokemon> get favorites => _favoritesBox.values.toList();`



```

9 class PokemonAdapter extends TypeAdapter<Pokemon> {
10   @override
11   final int typeId = 1;
12
13   @override
14   Pokemon read(BinaryReader reader) {
15     final numOffFields = reader.readByte();
16     final fields = <int, dynamic>{};
17     for (int i = 0; i < numOffFields; i++) reader.readByte(); reader.read(),
18   };
19   return Pokemon(
20     id: fields[0] as int,
21     name: fields[1] as String,
22     types: _parseTypes((fields[2] as List).cast<String>()),
23     sprite: fields[3] as String,
24     isFavorite: fields[4] as bool,
25   );
26 }
27
28 List<PokemonType> _parseTypes(List<String> data) {
29   return data.map((item) {
30     return PokemonType(id: 0, name: item);
31   }).toList();
32 }
33
34 @override
35 void write(BinaryWriter writer, Pokemon obj) {
36   writer
37     ..writeByte(5)
38     ..writeByte(0)
39     ..write(obj.id)
40     ..writeByte(1)
41     ..write(obj.name)
42     ..writeByte(2)
43     ..write(obj.types.map((type) => type.type).toList())
44     ..writeByte(3)
45     ..write(obj.sprite)
46     ..writeByte(4)
47     ..write(obj.isFavorite);
48 }

```

La implementación de Hive permite una gestión eficiente de datos sin necesidad de una base de datos relacional pesada.

2.2.4 SCREENS

La carpeta screens contiene las diferentes pantallas de la aplicación. Se han creado varias secciones principales:

2.2.4.1 HomeScreen

Esta es la pantalla principal de la aplicación y contiene:

- Barra de búsqueda: Permite filtrar Pokémon por nombre.
- Botón de filtros: Abre un menú para aplicar filtros por tipo y generación.
- Lista de Pokémon: Muestra los Pokémon obtenidos desde la API.
- Paginación infinita: Se cargan más Pokémon conforme el usuario se desplaza hacia abajo.

```

class HomeScreen extends StatefulWidget {
  const HomeScreen({super.key});

  @override
  State<HomeScreen> createState() => _HomeScreenState();
}

You, hace 4 horas | 1 author (You)
class _HomeScreenState extends State<HomeScreen> {
  @override
  void initState() {
    super.initState();
    _fetchPokemons();
  }

  // Método para obtener Pokémon desde el proveedor
  void _fetchPokemons({int page = 0}) {
    final filterProvider = Provider.of<FilterProvider>(context, listen: false);
    final pokemonProvider = Provider.of<PokemonProvider>(context, listen: false);

    pokemonProvider.fetchPokemons(
      generations: filterProvider.selectedGenerations,
      pokemonTypes: filterProvider.selectedTypes,
      searchText: filterProvider.searchText,
      page: page
    );
  }

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: const Text('Pokepedia'),
        titleTextStyle: Theme.of(context).appBarTheme.titleTextStyle
          ?? Theme.of(context).textTheme.titleLarge?.copyWith(
            color: const Color.fromARGB(255, 0, 0, 0),
            fontFamily: 'Poppins-Black',
          ),
      ),
    );
  }
}

```

2.2.4.2 PokemonDetailsScreen

Muestra los detalles completos de un Pokémon:

- Información básica: Nombre, altura, peso, experiencia base.
- Tipos y efectividad: Muestra los tipos del Pokémon y su efectividad en combate.
- Movimientos: Lista de movimientos disponibles.
- Evoluciones: Muestra la cadena evolutiva del Pokémon.

Esta pantalla se organiza en pestañas para mejorar la experiencia del usuario.

```

16 class PokemonDetailsScreen extends StatelessWidget {
17   static const List<Tab> tabs = [
18     Tab(text: 'Info'),
19     Tab(text: 'Moves'),
20     Tab(text: 'Evolutions'),
21   ];
22   final Pokemon pokemon; // Recibe un objeto Pokemon para mostrar sus detalles
23
24   const PokemonDetailsScreen({
25     super.key,
26     required this.pokemon
27   });
28
29   @override
30   Widget build(BuildContext context) {
31     return DefaultTabController( // Controlador para manejar las pestañas
32       length: tabs.length,
33       child: Scaffold(
34         body: NestedScrollView( // Permite que las pestañas tengan un efecto de desplazamiento
35           headerSliverBuilder: (context, innerBoxIsScrolled) => [
36             _renderHeader(context), // Renderiza el encabezado de la pantalla
37             _renderTabs(context), // Renderiza las pestañas
38           ],
39           body: _renderBody() // Contenido de la pantalla
40         ) // NestedScrollView
41       ), // Scaffold
42     ); // DefaultTabController
43   }
44
45   Widget _renderHeader(BuildContext context) {
46     // Obtiene el proveedor de tema
47     final themeProvider = Provider.of<ThemeProvider>(context);
48
49     final headerColor = themeProvider.isDarkMode
50       ? hslColor.withLightness(0.25).toColor() // Color del encabezado según el tema oscuro
51       : pokemon.typeColor; // Usa el color del tipo de Pokémon en el tema claro

```

2.2.5 WIDGETS

Esta carpeta contiene los componentes reutilizables de la aplicación. Son muchos widgets, en este documento se explicarán los más importantes, el resto está comentado su funcionamiento en el código.

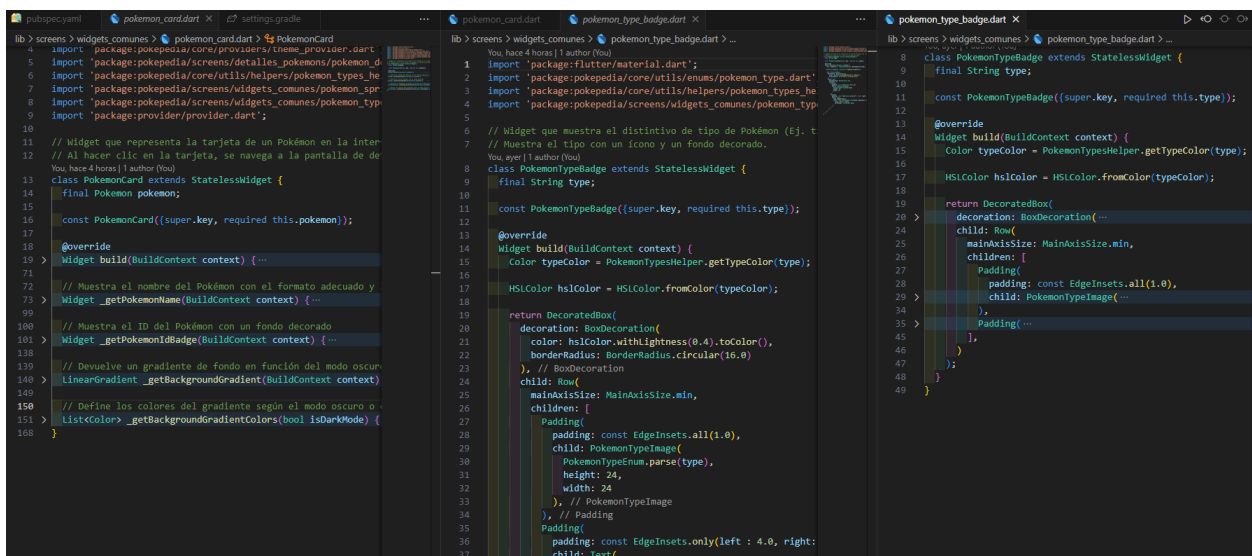
2.2.5.1 Widgets comunes

- **PokemonCard**: Tarjeta que muestra información resumida de un Pokémon y permite acceder a su pantalla de detalles.
- **PokemonSprite**: Widget que carga y muestra la imagen de un Pokémon con una animación.
- **PokemonTypeBadge**: Muestra un distintivo con el tipo del Pokémon.

2.2.5.2 Widgets Específicos de Pantallas

Son los widgets específicos de las pantallas principal y de detalles, son muchos pero los más interesantes son:

- **PokemonInfoTab**: Pestaña que muestra información detallada de un Pokémon.
- **PokemonMovesTab**: Pestaña con los movimientos del Pokémon.
- **PokemonEvolutionsTab**: Pestaña que muestra la evolución del Pokémon.
- **ThemeToggleButton**: Botón para alternar entre modo claro y oscuro.



Estos widgets mejoran la modularidad y reutilización del código dentro de la aplicación.

```

class PokemonInfoTab extends StatelessWidget {
  final PokemonDetails pokemon;

  const PokemonInfoTab({super.key, required this.pokemon});

  @override
  Widget build(BuildContext context) {
    return SingleChildScrollView(
      child: Padding(
        padding: const EdgeInsets.symmetric(vertical: 16.0),
        child: Column(
          crossAxisAlignment: CrossAxisAlignment.start,
          children: [
            Padding(
              padding: const EdgeInsets.all(16),
              child: Text(
                pokemon.description,
                style: Theme.of(context).textTheme.bodyLarge!.copyWith(
                  fontStyle: FontStyle.italic
                ),
              ), // Text
            ), // Padding
            Padding(
              padding: const EdgeInsets.all(16.0),
              child: PokemonTypes(pokemon: pokemon),
            ), // Padding
            PokemonBasicInfo(pokemon: pokemon),
            const SizedBox(height: 16),
            PokemonTypesEffectiveness(pokemon: pokemon),
            const SizedBox(height: 16),
            PokemonStats(pokemon: pokemon),
            const SizedBox(height: 16),
            PokemonAbilities(pokemon: pokemon)
          ],
        ), // Column
      ), // Padding
    ); // SingleChildScrollView
  }
}

```

```

class PokemonMovesTab extends StatelessWidget {
  final PokemonDetails pokemon;

  const PokemonMovesTab({
    super.key,
    required this.pokemon
  });

  @override
  Widget build(BuildContext context) {
    final textStyle = Theme.of(context).textTheme.bodyMedium!;

    return ListView.builder(
      padding: const EdgeInsets.symmetric(vertical: 16.0),
      itemCount: pokemon.sortedMoves.length,
      itemBuilder: (context, index) {
        final item = pokemon.sortedMoves[index];

        return ListTile(
          leading: PokemonTypeImage(
            PokemonTypeEnum.parse(item.type.name),
            height: 24,
            width: 24,
          ), // PokemonTypeImage
          title: Text(
            item.name,
            style: textStyle.copyWith(fontWeight: FontWeight.bold)
          ), // Text
          subtitle: Text(
            item.flavorText,
            style: textStyle.copyWith(fontStyle: FontStyle.italic)
          ), // Text
          trailing: Text(
            item.level.toString(),
            style: textStyle,
          ), // Text
        ); // ListTile
      }, // ListView.builder
    );
  }
}

```

```

class PokemonMovesTab extends StatelessWidget {
  final PokemonDetails pokemon;

  const PokemonMovesTab({
    super.key,
    required this.pokemon
  });

  @override
  Widget build(BuildContext context) {
    final textStyle = Theme.of(context).textTheme.bodyMedium!;

    return ListView.builder(
      padding: const EdgeInsets.symmetric(vertical: 16.0),
      itemCount: pokemon.sortedMoves.length,
      itemBuilder: (context, index) {
        final item = pokemon.sortedMoves[index];

        return ListTile(
          leading: PokemonTypeImage(
            PokemonTypeEnum.parse(item.type.name),
            height: 24,
            width: 24,
          ), // PokemonTypeImage
          title: Text(
            item.name,
            style: textStyle.copyWith(fontWeight: FontWeight.bold)
          ), // Text
          subtitle: Text(
            item.flavorText,
            style: textStyle.copyWith(fontStyle: FontStyle.italic)
          ), // Text
          trailing: Text(
            item.level.toString(),
            style: textStyle,
          ), // Text
        ); // ListTile
      }, // ListView.builder
    );
  }
}

```

```

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

  @override
  Widget build(BuildContext context) {
    final themeProvider = Provider.of<ThemeProvider>(context);
    return IconButton(
      color: const Color.fromARGB(255, 10, 10, 10),
      key: const Key('TOGGLE_THEME'),
      icon: Icon(
        themeProvider.mode == ThemeMode.dark
          ? Icons.light_mode
          : Icons.dark_mode,
      ), // Icon
      onPressed: () {
        themeProvider.toggleMode(); // Alterna el modo de tema
      },
    ); // IconButton
  }
}

```

3. ASPECTOS DE MEJORA

Evidentemente, esta aplicación no es más que una idea primaria y desarrollada en un tiempo muy acotado. Incluye aspectos como el trabajo simultáneo y compaginado entre dos fuentes de datos, siendo una la API externa y otra una base de datos local en fichero Hive.

Existen diversos aspectos de mejora y ampliación:

1. Optimización del Rendimiento

- **Lazy Loading de Imágenes:** Actualmente, se usan imágenes de alta resolución para los Pokémon. Se podría implementar `cached_network_image` con una estrategia de precarga y compresión para reducir la carga en la memoria.
- **Paginación Eficiente:** Aunque se usa carga infinita, se podría optimizar la paginación limitando la cantidad de Pokémon cargados en memoria a la vez y descartando los que ya no están en pantalla.
- **Mejor Manejo del Estado:** En `PokemonProvider`, cada vez que se actualiza la lista de Pokémon, se usa `notifyListeners()`, lo que puede impactar el rendimiento. Usar `ChangeNotifierProxyProvider` o `Riverpod` podría mejorar la eficiencia.

2. Mejora en la Experiencia de Usuario (UX)

- **Pantalla de Carga Más Dinámica:** En lugar de solo un `CircularProgressIndicator`, se podría agregar una animación más atractiva (como una Pokébola girando).
- **Transiciones Suaves:** Implementar transiciones de pantalla más fluidas con `PageRouteBuilder` y animaciones en `Hero` para los sprites de Pokémon.
- **Modo Offline:** Cachear los datos más recientes para permitir el uso de la app sin conexión (por ejemplo, los Pokémon ya vistos).

3. Mejoras en la Arquitectura y Código

- **División en Capas Más Definida:** Aunque `services` y `providers` ya están separados, se podría implementar una capa de repositorio (`PokemonRepository`) para desacoplar aún más la obtención de datos.
- **Uso de GraphQL Más Específico:** Actualmente, las consultas traen datos adicionales. Se podría optimizarlas seleccionando solo los campos estrictamente necesarios en cada vista.

4. Accesibilidad y Usabilidad

- **Modo Alto Contraste:** Mejorar la accesibilidad con una opción de alto contraste para usuarios con problemas de visión.
- **Compatibilidad con Lectores de Pantalla:** Agregar etiquetas `Semantics` para que los lectores de pantalla describan los elementos correctamente.

5. Funcionalidades Adicionales

- Comparación de Pokémon: Implementar una funcionalidad para comparar estadísticas entre dos Pokémon seleccionados.
- Historial de Búsqueda: Mostrar búsquedas recientes en la barra de búsqueda para facilitar la exploración.
- Modo Batalla Simulada: Crear una funcionalidad que permita a los usuarios seleccionar Pokémon y simular enfrentamientos según sus estadísticas y tipos.
- Pokémon de día: Cada vez que el usuario abra la aplicación, puede aparecer un Pokémon asociado al número de día actual.

BIBLIOGRAFÍA

- <https://docs.flutter.dev/>
- <https://e200.medium.com/flutter-infinite-scroll-with-rest-api-2b11f64b9d02>
- <https://www.scaler.com/topics/texteditingcontroller-flutter/>
- <https://youtu.be/-4pt1ACoR9c?si=4YoqLNo278VKN5gV>
- <https://www.geeksforgeeks.org/spring-boot-graphql-integration/>
- <https://pub.dev/documentation/graphql/latest/>
- https://github.com/dart-backend/graphql_dart
- <https://medium.com/@yazan99sh/flutter-hive-tutorial-93c0280baabb>
- <https://github.com/alanlgoncalves/pokedex/tree/master>
- <https://github.com/ArizArmeidi/FlutterDex/tree/main>
- <https://github.com/brunogabriel/pokedex-flutter/tree/main>
- Apuntes del temario de Programación Multimedia y Dispositivos Móviles, realizados por Francisco Miguel Fernández Banderas

Ubicado en: <https://moodle.iespablocicasso.es/>

- Muchas de las animaciones utilizadas e idea principal del proyecto han salido de aquí: https://github.com/pulakctl/flutter_dex