TUTORIAL DE USO DE LA APLICACIÓN INFOAUTOS

Aplicación de Información y Comparación de Autos

Autor: Julio Diaz Gil.

ÍNDICE

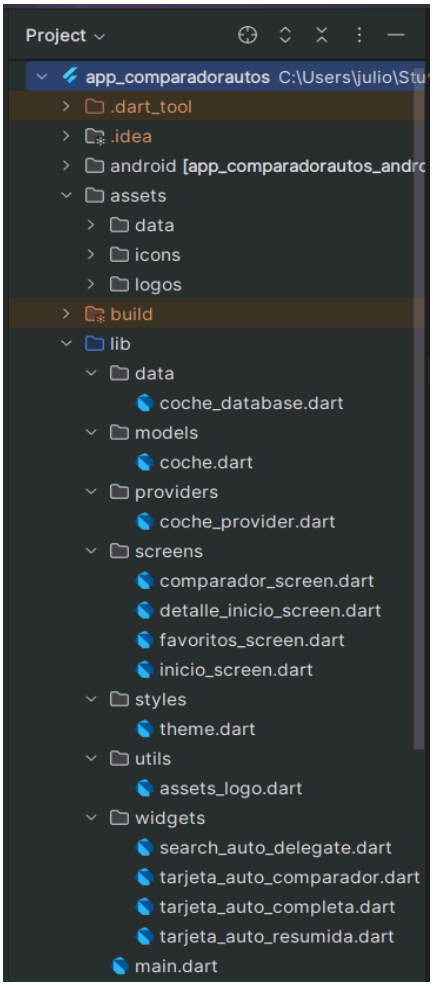
INTRODUCCIÓN SOBRE LA APP

InfoAutos es una aplicación móvil desarrollada con Flutter y programada en Dart, diseñada para ofrecer una herramienta intuitiva y eficiente para la consulta y comparación de automóviles. Permite a los usuarios explorar fichas técnicas, comparar características entre distintos modelos y gestionar una lista de autos favoritos. La aplicación utiliza SQLite como sistema de base de datos local para almacenar información sobre los vehículos, y SharedPreferences para guardar configuraciones o datos ligeros del usuario. Además, emplea el patrón Provider para una gestión de estado limpia y reactiva, asegurando un funcionamiento fluido y organizado.

- ## ESTRUCTURA DE LA APLICACIÓN

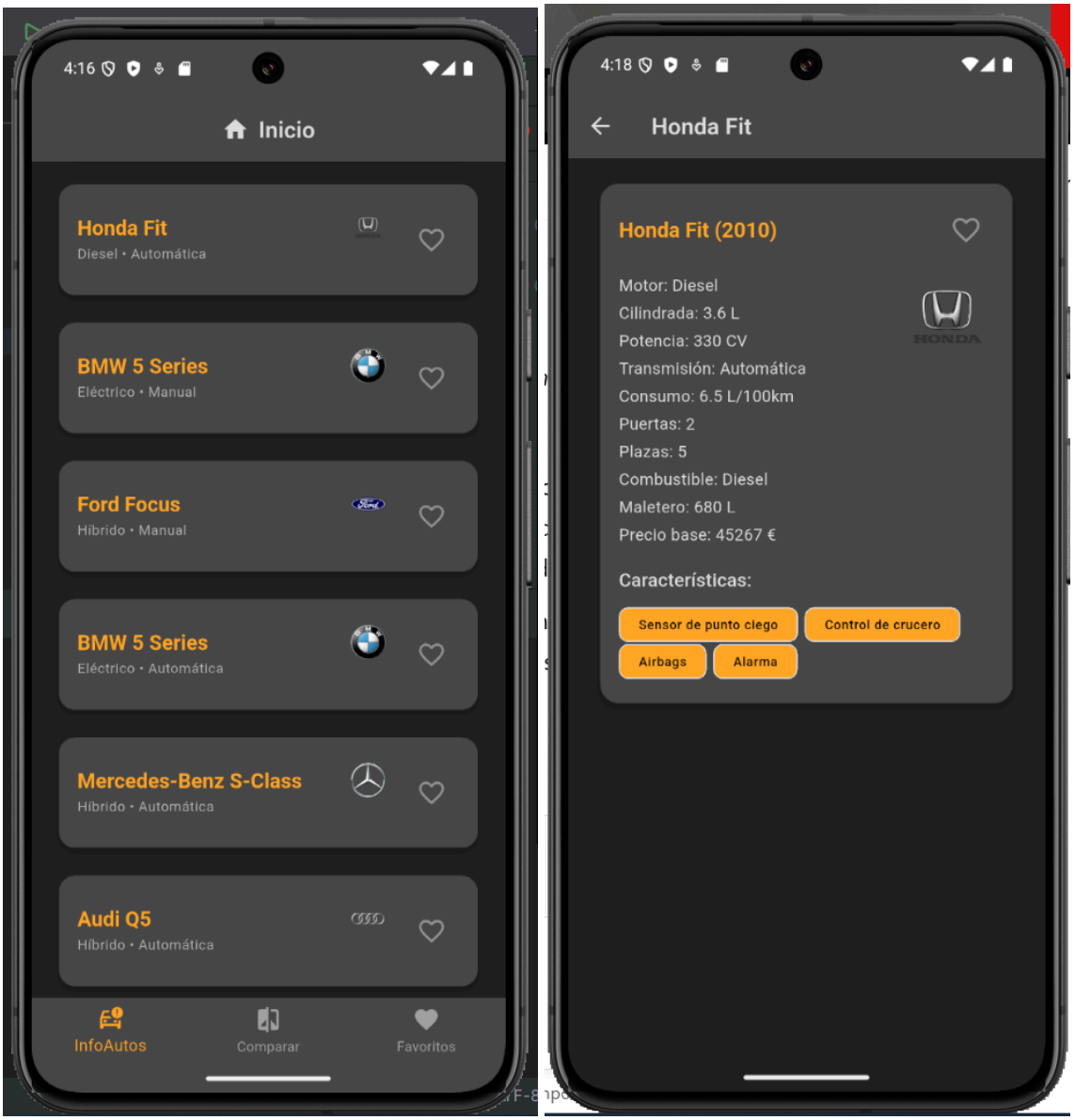
- **Assets:** todas las imágenes y archivos necesarios en la aplicación.
    - Data.
    - Icons.
    - Logos.

- **Data:** contiene la clase encargada de gestionar la base de datos mediante SQLite.
- **Models:** representa las estructuras de datos utilizadas en la aplicación.
- **Providers:** contiene las clases encargadas de gestionar el estado de la aplicación y comunicar los datos entre la lógica y la interfaz de usuario.
- **Screens:** contiene las clases que representan las diferentes pantallas de la aplicación.
- **Styles:** contiene las definiciones de estilos globales usados en la aplicación.
- **Utils:** la generación dinámica de rutas para acceder al logo de la aplicación.
- **Widgets:** contiene las diferentes tarjetas con las información que se muestran en las pantallas.



PANTALLA DE INICIO

- Capturas de pantalla:



- Descripción de la pantalla:

La pantalla principal muestra un listado con todos los coches disponibles para consulta. El usuario puede desplazarse verticalmente para explorar los modelos y acceder al detalle de cada coche tocando un elemento de la lista. Tiene un AppBar arriba que muestra en la pantalla que estas.

Se utiliza un FutureBuilder para cargar la lista de coches de forma asíncrona desde la base de datos local mediante el coche_provider.dart.

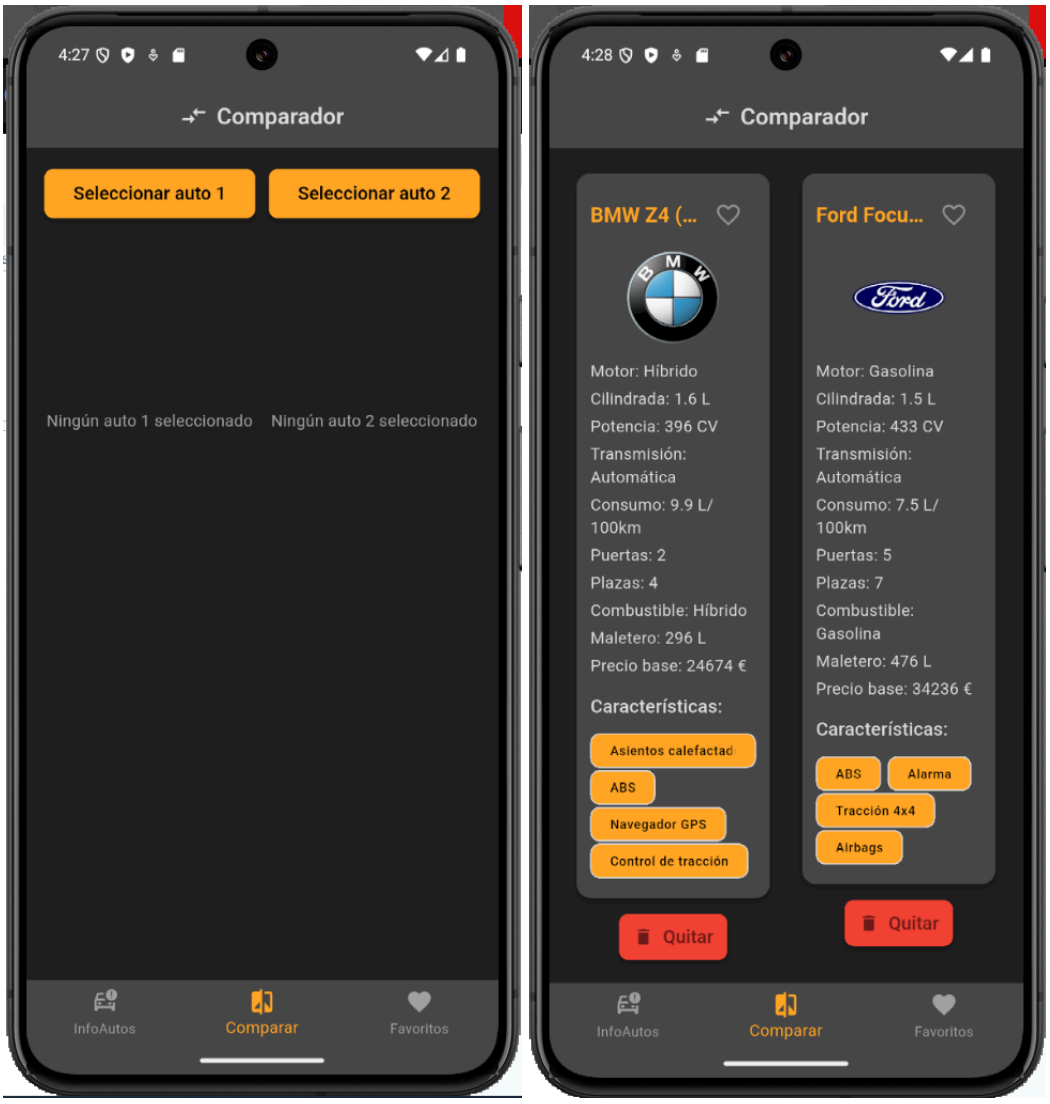
- **Fragmentos clave del código:**

```dart
/// Pantalla principal que muestra la lista de todos los vehículos disponibles.
///
/// Cada coche se presenta mediante una tarjeta resumida (`TarjetaAutoResumida`).
/// Al pulsar sobre una tarjeta, se navega a la pantalla de detalles (`InicioDetalleScreen`).
class InicioScreen extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    final prov = Provider.of<CocheProvider>(context);
    final List<Coche> listaCoches = prov.todos;

    return ListView.builder(
      padding: const EdgeInsets.symmetric(vertical: 12),
      itemCount: listaCoches.length,
      itemBuilder: (context, index) {
        final coche = listaCoches[index];

        /// Al tocar una tarjeta, se navega a una vista detallada del coche.
        return GestureDetector(
          onTap: () {
            Navigator.push(
              context,
              MaterialPageRoute(
                builder: (_) => InicioDetalleScreen(coche: coche),
              ),  // MaterialPageRoute
            );
          },
          child: Padding(
            padding: const EdgeInsets.symmetric(horizontal: 12, vertical: 6),
            child: TarjetaAutoResumida(coche: coche),
          ),  // Padding
```

- **Explicación del código:**
  - Se usa provider para acceder al estado y datos.
  - FutureBuilder espera a que se carguen los coches antes de mostrarlos.
  - Cada coche se representa con un ListTitle.
  - Al pulsar un coche, se navega a la pantalla de detalle pasando el objeto seleccionado.

PANTALLA DEL COMPARADOR

- Capturas de pantalla:



- Descripción de la pantalla:

La pantalla de comparación permite al usuario seleccionar dos coches diferentes del catálogo y ver sus características técnicas en paralelo.

Se muestran atributos clave como marca, modelo, año, tipo de combustible, potencia... El diseño está organizado en dos columnas para facilitar la comparación directa. También añade un botón para eliminar cualquiera de los dos coches.

- **Fragmentos clave del código:**

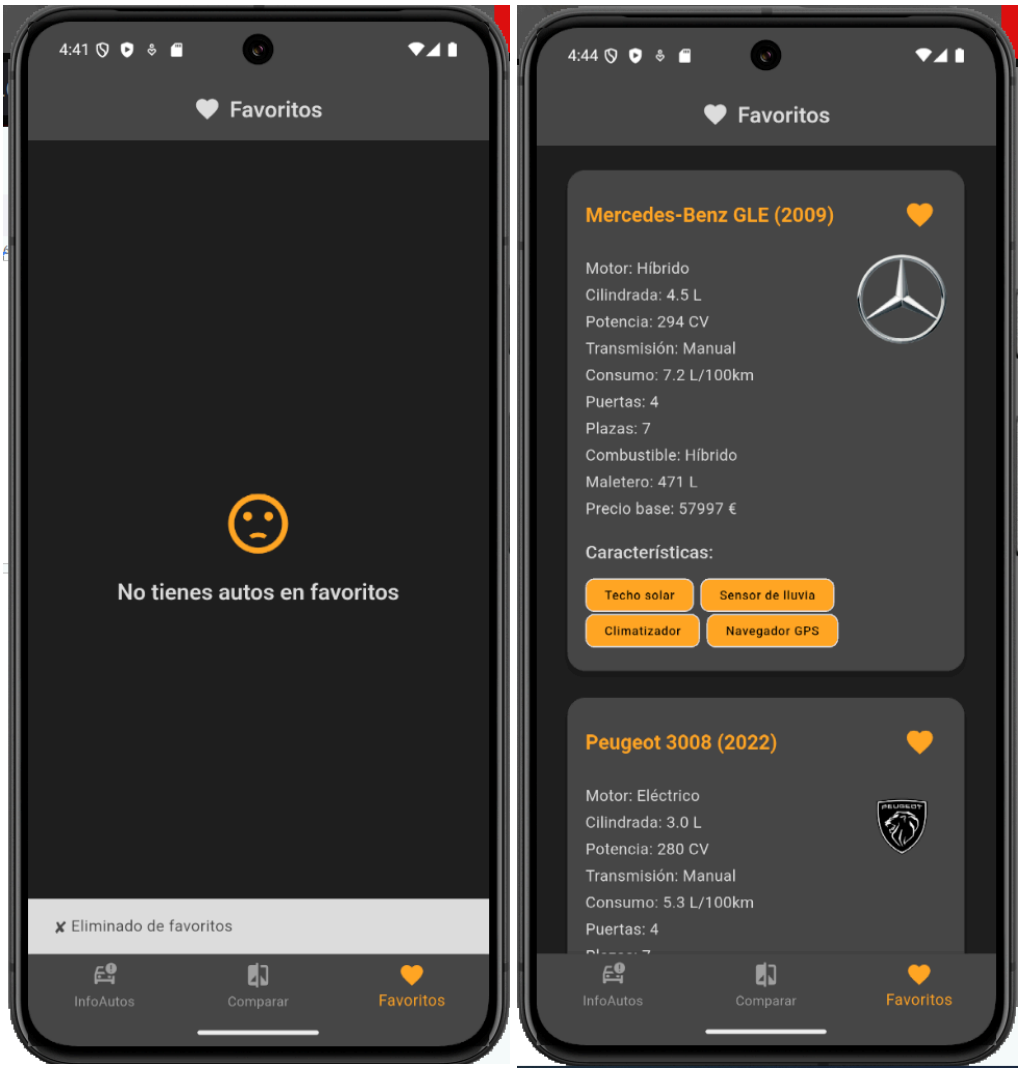
```dart
/// Botones para seleccionar los coches a comparar.
/// Se utiliza un buscador personalizado mediante [SearchAutoDelegate].
Row(
  children: [
    Expanded(
      child: ElevatedButton(
        onPressed: () async {
          final Coche? resultado = await showSearch<Coche?>(
            context: context,
            delegate: SearchAutoDelegate(prov.todos),
          );
          if (resultado != null) prov.asignarComparado(1, resultado);
        },
        child: Text(
          c1 == null
              ? 'Seleccionar auto 1'
              : '${c1.marca} ${c1.modelo}',
          textAlign: TextAlign.center,
        ),  // Text
      ),  // ElevatedButton
    ),  // Expanded
    const SizedBox(width: 12),
```

```dart
/// Tarjetas comparativas lado a lado.
/// Si no hay coche seleccionado, se muestra un mensaje de marcador de posición.
/// Debajo de cada tarjeta aparece un botón de "Quitar" centrado, con fondo rojo.
Row(
  crossAxisAlignment: CrossAxisAlignment.start,
  children: [
    Expanded(
      child: Column(
        children: [
          c1 == null
              ? _placeholderTexto('Ningún auto 1 seleccionado', cs, tt)
              : TarjetaAutoComparador(coche: c1),
          const SizedBox(height: 6),
          if (c1 != null)
            Center(
              child: ElevatedButton.icon(
                onPressed: () => prov.asignarComparado(1, null),
                icon: const Icon(Icons.delete, size: 18, color: Colors.black54),
                label: const Text('Quitar', style: TextStyle(color: Colors.black54)),
                style: ElevatedButton.styleFrom(
                  backgroundColor: cs.error,
                  padding: const EdgeInsets.symmetric(horizontal: 12, vertical: 8),
                ),
              ),  // ElevatedButton.icon
            ),  // Center
        ],
      ),  // Column
    ),  // Expanded
    const SizedBox(width: 12),
```

- **Explicación del código:**
    - Se usa Provider para gestionar los coches seleccionados para comparar.
    - Cada botón lanza un buscador personalizado para seleccionar un coche.
    - Los coches seleccionados se muestran con tarjetas comparativas lado a lado.
    - Debajo de cada tarjeta hay un botón para quitar el coche correspondiente.
    - Si no hay coche seleccionado, se muestra un mensaje de marcador de posición.

PANTALLA DE FAVORITOS

- Capturas de pantalla:



- Descripción de la pantalla:

La pantalla de favoritos muestra una lista personalizada con los coches que el usuario ha marcado previamente como favoritos. Esta funcionalidad permite acceder rápidamente a los vehículos de interés sin necesidad de buscarlos nuevamente.

Cada coche se presenta de forma similar a como aparece en otras secciones: incluyendo su marca, modelo, logo y datos clave. Si no hay favoritos guardados, se muestra un mensaje informativo al usuario.

- **Fragmentos clave del código:**

```
/// Muestra mensaje de estado si no hay favoritos registrados.
if (listaFav.isEmpty) {
 return Center(
   child: Column(
     mainAxisAlignment: MainAxisAlignment.center,
     children: [
       Icon(Icons.sentiment_dissatisfied,
           size: 64, color: cs.secondary),  // Icon
       const SizedBox(height: 16),
       Text(
         'No tienes autos en favoritos',
         style: tt.headlineMedium,
         textAlign: TextAlign.center,
       ),  // Text
/// Muestra la lista de coches favoritos en forma de tarjetas.
return Container(
  color: cs.background,
  child: ListView.separated(
    padding: const EdgeInsets.all(16),
    itemCount: listaFav.length,
    separatorBuilder: (_, __) => const SizedBox(height: 12),
    itemBuilder: (context, index) {
      final coche = listaFav[index];
      return TarjetaAutoCompleta(coche: coche);
    },
```

- **Explicación del código:**
  - Se usa Provider para obtener la lista de coches marcados como favoritos.
  - La lista se muestra con un ListView, donde cada coche aparece como una tarjeta.
  - Los datos de los favoritos se guardan localmente mediante SharedPreferences.

CLASES PRINCIPALES BACKEND

- **CLASE: coche database.dart:**

Esta clase encapsula toda la lógica de acceso a la base de datos local SQLite para los objetos Coche. Implementa el patrón **Singleton**, asegurando que toda la aplicación use una única instancia para mantener la integridad de los datos y optimizar recursos.

- *Métodos principales:*

get_database():

Método getter que inicializa la base de datos si aún no está lista. Garantiza que siempre se devuelva una instancia válida, usando await _initDB().

_initDB():

Abre o crea físicamente el archivo .db en la ruta local del dispositivo. Se utiliza la función getDatabasesPath() de sqlite para asegurar compatibilidad entre plataformas.

insertCoche():

Inserta un coche en la tabla. Si ya existe un coche con el mismo id, lo reemplaza (ConflictAlgorithm.replace).

- *Aclaraciones:*

SQLite + JSON: Usar sqLite junto con carga desde JSON permite una base local rápida, funcional offline y sin necesidad de depender de un backend externo.

Patrón Singleton: Evita que se creen múltiples instancias de la base de datos, lo cual podría provocar conflictos o consumo innecesario de memoria.

Conversión con toMap y fromMap:

Estos métodos permiten transformar los objetos Coche a un formato que SQLite puede guardar (Map<String, dynamic>) y viceversa.

- **CLASE: coche.dart:**

Esta clase representa un modelo de coche con sus principales características técnicas y de equipamiento. Se usa para mostrar, comparar y almacenar coches dentro de la aplicación. Incluye métodos para convertir los objetos a mapas (Map<String, dynamic>) y JSON, y viceversa. Esto facilita guardar y recuperar datos de la base de datos SQLite y cargar datos iniciales desde archivos JSON de forma sencilla y organizada.

- *Métodos Principales:*

toMap()

Convierte el objeto Coche en un mapa de clave-valor, que es el formato requerido para almacenar los datos en SQLite. La lista de características se guarda como una cadena separada por comas para simplificar el almacenamiento.

fromMap()

Crea un objeto Coche a partir de un mapa obtenido de la base de datos. Maneja valores nulos y convierte la cadena de características en una lista para que sea más fácil de usar dentro de la app.

toJson() y fromJson()

Permiten convertir el objeto a y desde formato JSON, utilizado principalmente para cargar datos iniciales desde archivos en assets o para exportar datos.

- *Aclaraciones:*

Separar las conversiones a mapas y JSON ayuda a que el modelo sea más fácil de manejar y no esté demasiado atado a cómo guardamos los datos. Así podemos usarlo en varias partes de la app sin problemas, ya sea para guardar info, mostrarla o enviarla. También, que el id sea opcional está bien porque SQLite se encarga de crear el id cuando toca, y así podemos crear coches en memoria antes de guardarlos sin complicarnos.

- **CLASE: coche provider.dart:**

Esta clase se encarga de manejar el estado global de los coches en la app: la lista completa, los favoritos y la comparación entre dos coches. Usa ChangeNotifier para que los widgets que dependen del estado se actualicen automáticamente cuando cambian datos.

- **Métodos principales:**

cargarCoches(List<Coche> coches): Sustituye la lista completa de coches por una nueva y avisa a la interfaz para que se actualice.

alternarFavorito(Coche coche): Añade o quita un coche de favoritos. Si ya está, lo elimina; si no, lo añade. Además, guarda los favoritos en SharedPreferences para que se mantengan aunque cierres la app.

esFavorito(Coche coche): Comprueba si un coche está marcado como favorito.

asignarComparado(int lado, Coche? coche): Asigna un coche al primer o segundo espacio para comparación, o limpia esa posición si se pasa null.

limpiarComparador(): Quita los dos coches asignados para comparar, dejando los campos vacíos.

guardarFavoritos(): Guarda en SharedPreferences solo los IDs de los coches favoritos, para conservarlos entre sesiones.

cargarFavoritosDesdePreferencias(): Lee los IDs guardados y rellena la lista de favoritos con los coches que coinciden en la lista principal. Así se recuperan los favoritos cuando se inicia la app.

- **Aclaraciones:**

Guardar únicamente los IDs en SharedPreferences para que la aplicación sea mas eficiente y también previene la duplicación de datos. Mantener la lista de favoritos sincronizada con la lista principal evita posibles inconsistencias. El uso de ChangeNotifier permite que la interfaz se actualice automáticamente ante cualquier cambio, haciendo que la aplicación sea más reactiva y fluida.