Universidad Nacional de Colombia
Facultad de Ingeniería

Tutorial - Construcción del Proyecto desde Cero

Presentado por:

Ana Sofia Rodriguez Neira
Juliana Parra Caro
Carlos Javier Cuervo Baracaldo
Christian Alejandro Sanabria Pinzon

Asignatura:
Ingeniería de Software I

Bogotá, D.C
24 / 10 / 2025

En este tutorial se explicará paso a paso como instalar y realizar la configuración básica para desarrollar un proyecto en Flutter**.**

1. Lenguaje y Framework Seleccionados

Dart - El Lenguaje

Dart es un lenguaje moderno desarrollado por Google, diseñado específicamente para crear aplicaciones multiplataforma. Su sintaxis es clara y fácil de aprender, especialmente para desarrolladores con experiencia en Java o C#. Dart compila a código nativo, lo que garantiza un alto rendimiento.

Flutter - El Framework

Flutter es el framework de UI de Google para crear aplicaciones nativas compiladas desde una única base de código. Su principal ventaja es el uso de widgets personalizables que se renderizan directamente sobre el canvas del sistema operativo, eliminando la necesidad de puentes que afecten el rendimiento.

2. Configuración del Entorno

Hay varias opciones para la configuración. Va desde cómo instalar flutter, instalar android Studio, hasta cómo configurar el emulador. A grandes rasgos, este corto tutorial se puede dividir en:

Instala Flutter desde VS Code

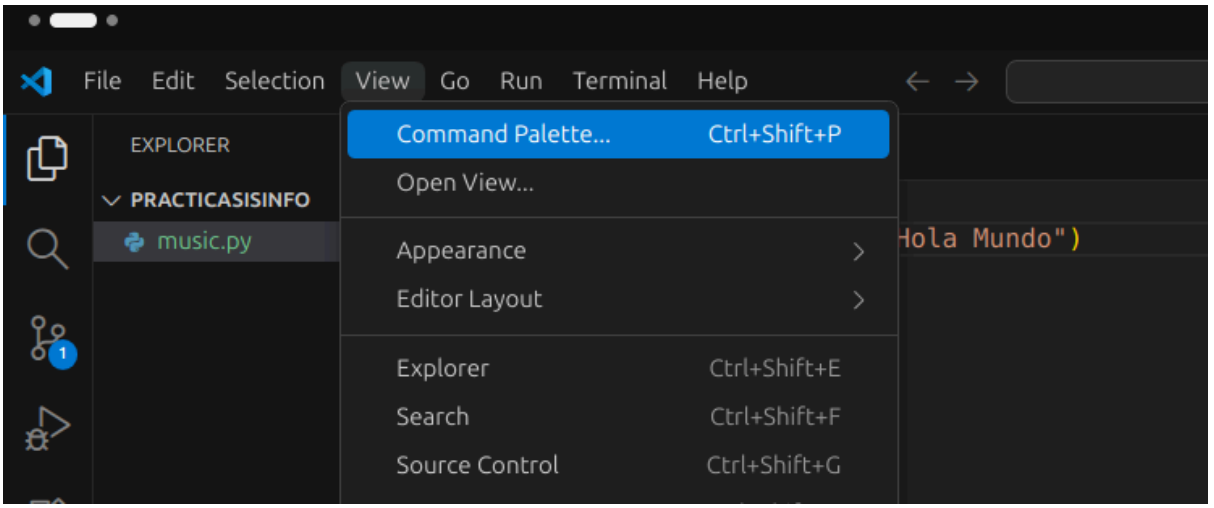
1. Ingresa al sitio web de Flutter en este link, y selecciona tu sistema operativo para poder ver qué requisitos tiene y como instalarlos. En resumen:
    a. Windows: Tener Git y Visual Studio Code.
    b. MacOS: Tener Git y Visual Studio Code.
    c. Linux: Tener visual Studio Code y los paquetes:
        i. `curl`
        ii. `git`
        iii. `unzip`
        iv. `xz-utils`
        v. `zip`
        vi. `libglu1-mesa`

        Alternativamente, para distribuciones basadas en Debian se pueden correr los comandos:

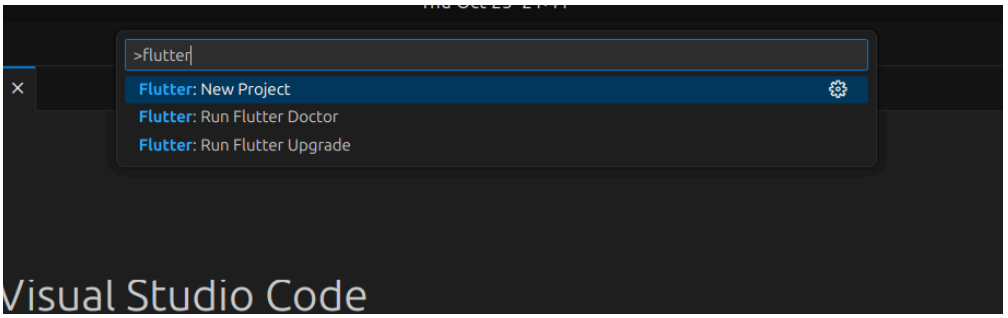
```
sudo apt-get update -y && sudo apt-get upgrade -y

sudo apt-get install -y curl git unzip xz-utils zip libglu1-mesa
```

2. Abrir VS Code.

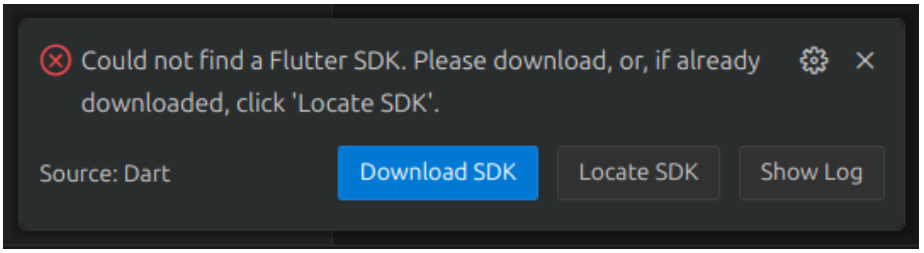
3. Accede a el [marketplace de extensiones] para VS Code e instala la extensión de Flutter. Si el navegador lo pide, abre VS Code.
4. En VS Code, abre la paleta de comandos con Control + Shift + P, o ve a **View** > **Command Palette.**

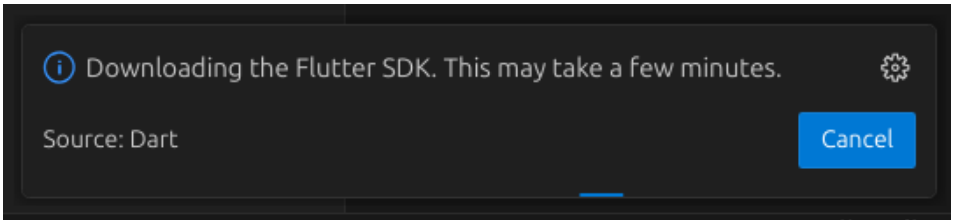


5. Escribe **flutter**, y selecciona **Flutter: New Project.**



6. VS Code te va a pedir que ubiques el Flutter SDK. En la opción **Download SDK**, ubica la carpeta para instalar el SDK y la instalación comenzará.





Si la descarga es exitosa, flutter doctor empezará la instalación.

7. Ahora sigue configurar el path del SDK. Determina la ruta absoluta de donde lo descargaste, y corres el siguiente comando en la terminal:
   `echo 'export PATH="<path-to-sdk>:$PATH"' >> ~/.bash_profile`
   Recuerda reemplazar **path-to-sdk** a tu ruta absoluta.

   Esto es para bash, pero cambia un poco respecto al shell que tengas. Puedes verificarlo con `echo $SHELL` desde tu terminal. Para revisar el comando para otros shell revisa aquí.

```
echo 'export PATH="/home/juli/Documents/code/test/flutter/bin:$PATH"' >> ~/.bash_profile
```

8. Ahora, debes reiniciar las configuraciones del shell con el comando `source ~/.bash_profile` . Si todo salió bien, `flutter -- version` y `dart -- version` debe retornar las versiones.

Instala Flutter desde Android studio

También puedes usar Flutter desde Android Studio. Nuevamente, el proceso varía dependiendo del sistema operativo. En este documento se explicará el proceso en Linux.

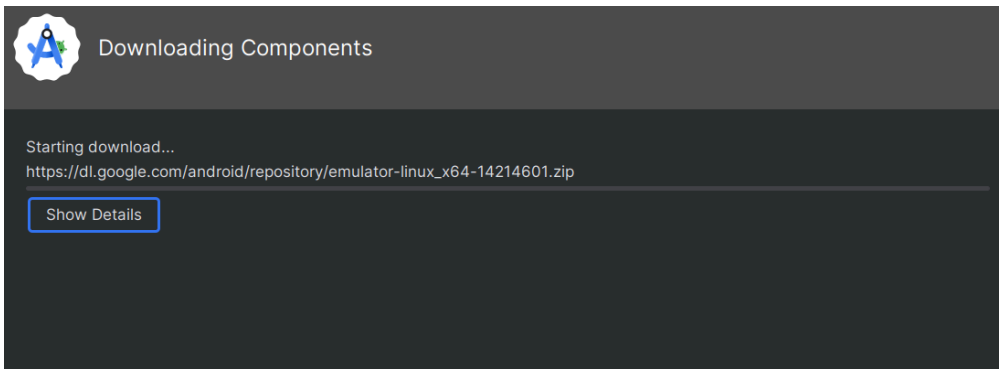
1. Descomprime el archivo que se descarga desde el sitio oficial y muévelo a /usr/local/, una ubicación más apropiada para tus aplicaciones.

```
$ sudo mv /home/juli/Downloads/android-studio /usr/local/
```

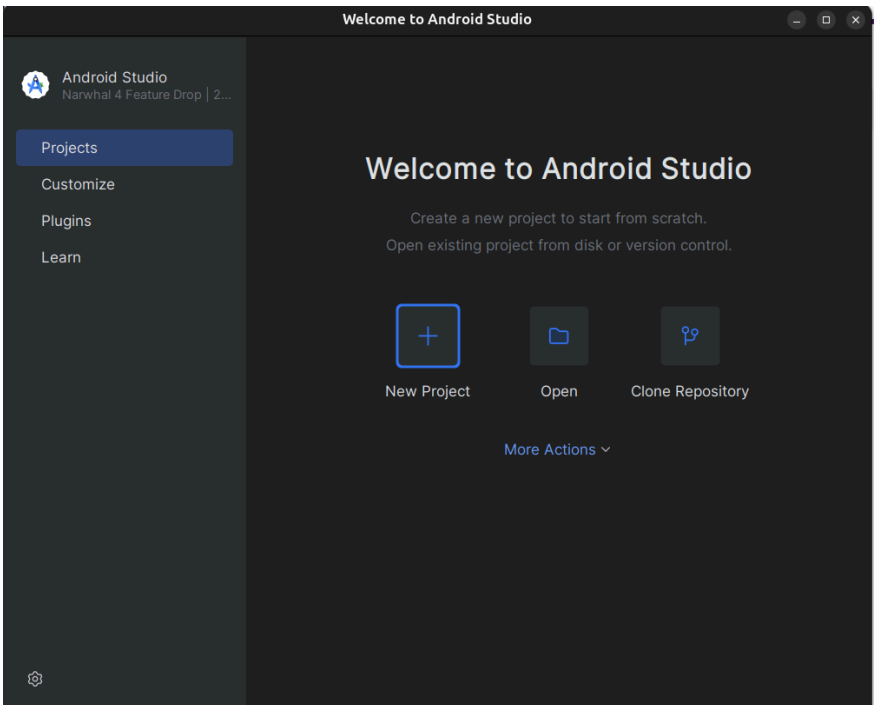
2. Cambia de directorio al archivo de binarios de android-studio, e inicia Android Studio desde allí con el comando `./studio.sh.`

```
~$ cd /usr/local/android-studio/bin
/usr/local/android-studio/bin$ ./studio.sh
```

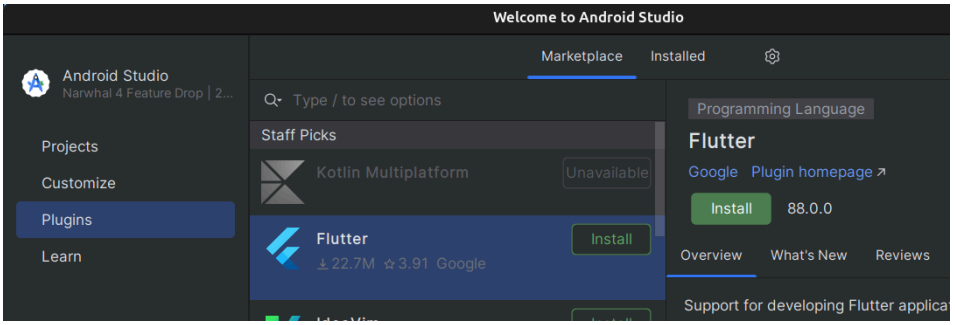
3. Se abrirá el Wizard de instalación, por lo que solo debes escoger tus preferencias de descarga. Puedes dejar las que vienen por default, aceptar términos y condiciones, y dejar que el programa haga el resto. Va a tomar un poco de tiempo. Luego de que termine, oprime el botón Finish.



4. ¡Ya terminamos de instalar Android Studio!



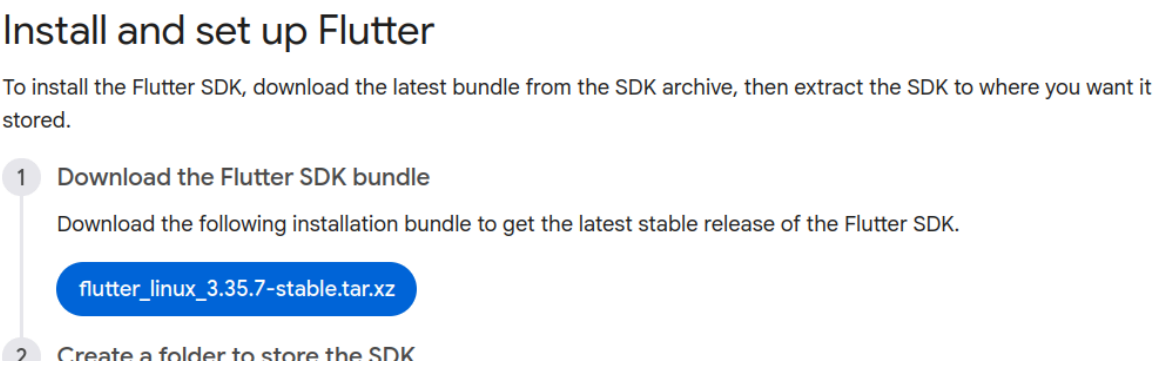
5. Para instalar Flutter, vamos a la sección de Plugins, buscamos Flutter y Dart, y oprimimos Install.



6. Luego del paso 5, debes instalar Flutter a tu computador. Tienes dos opciones, dirigirte al paso 8 de la configuración de entorno o instalarlo manual. Como ya tienes las instrucciones para hacerlo desde VS Code, repasamos los pasos para descargarlo y hacerlo manualmente.

Instala VS Code manualmente

1. Desde este sitio web, oprime el botón azul para descargar Flutter.



2. Crea una nueva carpeta y extrae el archivo (idealmente en /usr/bin). Recuerda cambiar la versión dependiendo de la que descargaste.

```
sudo tar -xf ~/Downloads/flutter_linux_3.35.7-stable.tar.xz -C /usr/bin/
```

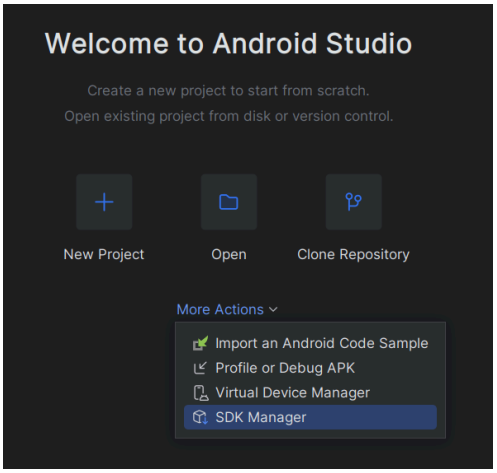
3. Ahora tienes que cambiar el path. Haz el paso 7 de la sección de Instalar Flutter con VS Code desde la carpeta /usr/bin/flutter.

```
ter$ echo 'export PATH="/usr/bin/flutter/bin:$PATH"' >> ~/.bash_profile
```

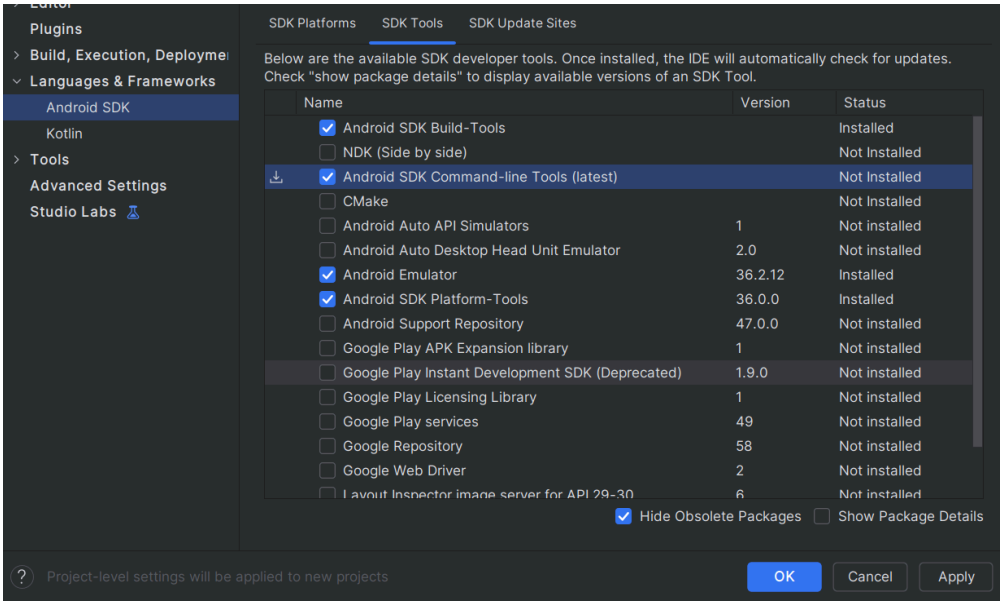
4. Repite el paso 8 de la sección de Instalar Flutter con VS Code.

```
juli@juli-ThinkPad-T480s:~$ source ~/.bash_profile
juli@juli-ThinkPad-T480s:~$ flutter --version
Flutter 3.35.7 • channel stable • https://github.com/flutter/flutter.git
Framework • revision adc9010625 (3 days ago) • 2025-10-21 14:16:03 -0400
Engine • hash 6b24e1b529bc46df7ff397667502719a2a8b6b72 (revision 035316565a) (3
days ago) • 2025-10-21 14:28:01.000Z
Tools • Dart 3.9.2 • DevTools 2.48.0
juli@juli-ThinkPad-T480s:~$ dart --version
Dart SDK version: 3.9.2 (stable) (Wed Aug 27 03:49:40 2025 -0700) on "linux_x64"
```

5. De vuelta a Android Studio. Tenemos que verificar que tenemos todos los componentes de Android necesarios para nuestro proyecto. Desde la pantalla inicial abre More Options, y SDK Manager.



6. Verifica que tengas los componentes que aparecen en la imagen en SDK Tools. Si no, marcarlos y oprime OK. Esto instalará lo necesario.



¡Tendrías todo listo para empezar! En este punto ya puedes configurar el emulador para ver tu aplicación desde el computador o puedes conectar un teléfono Android para verla, depende de ti. Por si prefieres configurar el emulador, te dejamos los pasos a continuación.

Configura el Emulador de Android

1. Correo el comando:

```
flutter doctor --android-licenses
```

2. Acepta las licencias con “y”.
3. Corre el comando `flutter doctor`. Esto va a revisar todo y asegurarse que estén todos los componentes. Si ves errores en Android Toolchain o Android Studio, revisa y ejecuta `flutter doctor` nuevamente.

```
juli@juli-ThinkPad-T480s:~$ flutter doctor
Doctor summary (to see all details, run flutter doctor -v):
[✓] Flutter (Channel stable, 3.35.7, on Ubuntu 24.04.3 LTS 6.14.0-33-generic,
    locale en_US.UTF-8)
[✓] Android toolchain - develop for Android devices (Android SDK version 36.1.0)
```

4. ¡Ya tienes tu emulador listo! Puedes empezar a desarrollar y ver lo que haces desde tu computador.

3. Librerías ORM utilizadas

Se uso sqflite como el ORM (Object-Relational Mapper) o más bien como driver de base de datos.

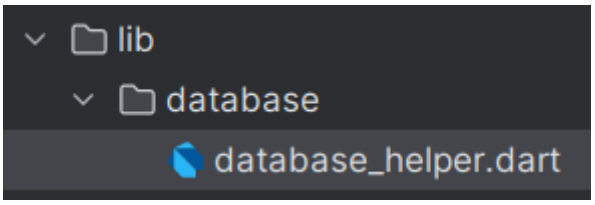
Librería	Propósito
sqflite	Maneja la conexión, creación y consultas a SQLite (la base de datos local del dispositivo).
path	Ayuda a crear rutas seguras entre carpetas del sistema (por ejemplo, dónde guardar el archivo `.db`).
path_provider	Permite acceder a directorios del dispositivo (para guardar la BD en la carpeta correcta).

Para instalarlas, se corrió el comando:

```
juli@juli-ThinkPad-T480s:~/StudioProjects/test_entrega_4$ flutter pub add sqflite path path_provider
Resolving dependencies...
Downloading packages...
  characters 1.4.0 (1.4.1 available)
+ ffi 2.1.4
  flutter_lints 5.0.0 (6.0.0 available)
  lints 5.1.1 (6.0.0 available)
  material_color_utilities 0.11.1 (0.13.0 available)
  meta 1.16.0 (1.17.0 available)
```

4. Configuración y levantamiento de la base de datos

El archivo principal que hace esto es el `database_helper.dart`, que implementa el **patrón Singleton** (una sola instancia de conexión a BD). En lib, creamos el directorio database, y le incluimos el archivo database_helper.dart.



Allí se inicializa `hola_mundo.db` y la tabla `usuarios`. El patrón Singleton evita múltiples conexiones.

```dart
import 'package:sqflite/sqflite.dart';
import 'package:path/path.dart';

class DatabaseHelper {
  static final DatabaseHelper _instance = DatabaseHelper._internal();
  factory DatabaseHelper() => _instance;

  static Database? _database;
  DatabaseHelper._internal();

  Future<Database> get database async {
    if (_database != null) return _database!;
    _database = await _initDB();
    return _database!;
  }

  Future<Database> _initDB() async {
    final dbPath = await getDatabasesPath();
    final path = join(dbPath, 'hola_mundo.db');
    return await openDatabase(
      path,
      version: 1,
      onCreate: (db, version) async {
        await db.execute('''
          CREATE TABLE usuarios(
            id INTEGER PRIMARY KEY AUTOINCREMENT,
            nombre TEXT
          )
        ''');
      },
    );
  }
```

Ahora, creamos la entidad usuario en un nuevo directorio `lib/models/usuario.dart`

```dart
class Usuario {
  final int? id;
  final String nombre;

  Usuario({this.id, required this.nombre});

  Map<String, dynamic> toMap() => {
    'id': id,
    'nombre': nombre,
  };

  factory Usuario.fromMap(Map<String, dynamic> map) => Usuario(
    id: map['id'] as int?,
    nombre: map['nombre'] as String,
  );
}
```

Para el acceso a la base de datos, se crea un archivo `usuario_repository.dart` en un nuevo directorio `lib/repositories/`:

```dart
import '../database/database_helper.dart';
import '../models/usuario.dart';

class UsuarioRepository {
  final DatabaseHelper _dbHelper = DatabaseHelper();

  Future<int> insertUsuario(Usuario usuario) async {
    final db = await _dbHelper.database;
    return await db.insert('usuarios', usuario.toMap());
  }

  Future<List<Usuario>> getUsuarios() async {
    final db = await _dbHelper.database;
    final result = await db.query('usuarios');
    return result.map((e) => Usuario.fromMap(e)).toList();
  }

  Future<void> deleteAll() async {
    final db = await _dbHelper.database;
    await db.delete('usuarios');
  }
}
```

Finalmente creamos una pequeña interfaz para probar la base de datos. Esto se hace modificando el archivo `main.dart`. Si no hay usuarios, muestra un botón para añadir usuarios, si existe uno, muestra un saludo al usuario existente. También aparece un botón para borrar el usuario.

5. Contenido Archivos .yml

Por default, contiene el nombre del proyecto, dependencias y otra información relacionada al proyecto. Luego de correr el comando de la sección 2, se incluye automáticamente:

```
sqflite: ^2.4.2
path: ^1.9.1
path_provider: ^2.1.5
```

6. Ejecución Hola Mundo

Este es el resultado de la ejecución en el emulador de Android: