Instituto Tecnológico Del Valle De Oaxaca

Materia: Desarrollo de aplicaciones para dispositivos móviles II

Tema 4. Framework

Tarea 1. Investigación de un framework para el desarrollo de aplicaciones móviles híbridas

Nombre del estudiante: Velasco García Claudia Guadalupe

Número de control: 18920061

Semestre: 9° Grupo: A

Carrera: Ingeniería en informática

Docente: M.I.T.I. Ambrosio Cardoso Jiménez

Semestre: Agosto - Diciembre 2022

7 de diciembre de 2022

Introducción

Hoy en día la tecnología ha ido avanzando de una manera exponencial, si nos detenemos a pensar en el desarrollo que ha tenido con las aplicaciones, podríamos decir que han marcado una tendencia en cuanto a la forma que los usuarios pueden acceder a los datos de diversas organizaciones, pasando así por aplicaciones de escritorio, aplicaciones web y ahora las aplicaciones móviles.

Actualmente existen tres tipos de aplicaciones móviles, primero podríamos mencionar a las aplicaciones nativas que como su nombre lo indica son desarrolladas para el sistema nativo de cada dispositivo, también estas las aplicaciones web y finalmente las aplicaciones híbridas o multiplataforma.

Es así que a continuación, te presentaremos una investigación acerca de las aplicaciones híbridas que técnicamente son aquellas capaces de funcionar en distintos sistemas operativos móviles, además abarcaremos un framework llamado flutter para el desarrollo de dichas aplicaciones, así como también una bitácora de su instalación, configuración y desarrollo básico de una aplicación realizada con este framework.

Contenido

Aplicaciones móviles híbridas

Son aplicaciones de software que combinan elementos de aplicaciones nativas y aplicaciones web. Las aplicaciones híbridas, a diferencia de las nativas, son aquellas capaces de funcionar en distintos sistemas operativos móviles. Entre ellos: Android, iOS y Windows Phone. De esta manera, una misma app puede utilizarse en cualquier smartphone o tablet, indistintamente de su marca o fabricante.

Para ello, estas aplicaciones tienen componentes que permiten la adaptabilidad de un mismo código a los requerimientos de cada sistema.

Ventajas

Las apps híbridas presentan varios beneficios frente a las aplicaciones nativas, pero todo depende del tipo de desarrollo que se requiera. Entre las utilidades de las aplicaciones híbridas se encuentra:

- Diferentes sistemas: estas aplicaciones se pueden descargar en cualquier sistema y dispositivo.
- Mejor costo: al emplear un núcleo unificado para todos los sistemas operativos, se reducen los costos en cuanto al desarrollo y el diseño.
- Escalabilidad: Es posible escalar estas apps híbridas de forma más sencilla.
- Mayor distribución.

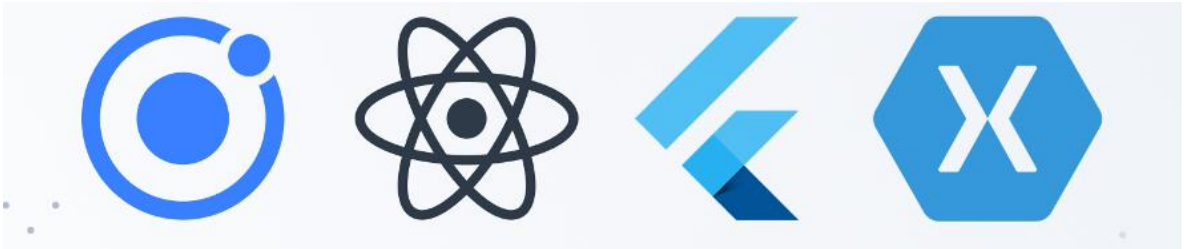
Desventajas

- No aprovechan de manera tan natural las capacidades del hardware.
- Experiencia de usuario más baja en comparación a las aplicaciones nativas.
- El rendimiento es más lento, ya que se trata de tecnología web integrada a plataformas móviles.

Frameworks para el desarrollo de aplicaciones híbridas

Los frameworks son plantillas o estructuras previas que facilitan el desarrollo de software y aplicaciones. Para el desarrollo de aplicaciones híbridas podemos mencionar:

1. **Ionic:** Una de sus principales ventajas es que ofrece una amplia gama de elementos de interfaz de usuario, facilitando la personalización.
2. **React Navite:** Es un framework de programación de aplicaciones nativas multiplataforma que está basado en JavaScript y ReactJS.
3. **Flutter:** El framework para aplicaciones híbridas de Google permite crear apps muy creativas y personalizadas. Además de contar con una gran versatilidad, su interfaz se caracteriza por un potente rendimiento.
4. **Xamarin:** Es una plataforma de código abierto, su principal ventaja es la abstracción que es casi absoluta de la capa de aplicación, permitiendo crear una app desarrollada en C# que se compila de manera nativa tanto a iOS, Android y Windows.



Flutter

Framework para el desarrollo de aplicaciones móviles híbridas

Flutter es un marco de trabajo de código abierto de Google para crear aplicaciones multiplataforma compiladas de forma nativa a partir de una única base de código.

Su objetivo es permitir a los desarrolladores que lancen aplicaciones de alto rendimiento que se adapten de forma natural a diferentes plataformas. Abarcando las diferencias en los comportamientos de scrolling, tipografía, iconos, y más.

Funciona con Dart que es un lenguaje de programación que usa el paradigma de Programación Orientada a Objetos (OOP). Además, ha sido creado por Google y es considerado por algunos como “el lenguaje de programación del futuro” por su versatilidad de uso.

Flutter Dart

Caracteristicas

Entre sus principales carcateristicas podemos mencionar las siguientes:

- Flutter funciona con Dart, un lenguaje optimizado para aplicaciones rápidas en cualquier plataforma.
- **Es multiplataforma:** Puede implementarse en múltiples dispositivos desde una única base de código.
- **Tiene la función de recarga en caliente:** Ayuda a experimentar, crear interfaces de usuario, agregar funciones y corregir errores

rápida y fácilmente. La recarga en caliente funciona inyectando archivos de código fuente actualizados en la Dart Virtual Machine (VM) en ejecución. Después de que la VM actualiza las clases con las nuevas versiones de campos y funciones, el marco de Flutter reconstruye automáticamente el árbol de widgets, lo que le permite ver rápidamente los efectos de sus cambios.

- **Diseño adaptativo:** Facilita la implementación de diseños adaptables y receptivos.
- **Soporte IDE:** Puede crear aplicaciones con Flutter utilizando cualquier editor de texto combinado con las herramientas de línea de comandos de Flutter. Además, se pueden agregar complementos que brindan finalización de código, resaltado de sintaxis, asistencia para la edición de widgets, soporte para ejecutar y depurar, etc.
- **Herramientas de desarrollo:** DevTools es un conjunto de herramientas de rendimiento y depuración para Dart y Flutter.
- Cuenta con el respaldo y el uso de Google, además es mantenido por una comunidad de desarrolladores globales.
- A diferencia de otros frameworks que separan vistas, controladores de vistas, layouts y otras propiedades, Flutter tiene un modelo de objeto unificado y consistente: el widget.
- **Programación declarativa en Flutter:** El estilo de programación que utiliza Flutter se llama Programación Declarativa, inspirado en el estilo de programación de React y que va de la mano con el paradigma de Programación Funcional.

Ventajas

Sus principales ventajas son las siguientes:

- Es de código abierto
- **Rapidez:** El código de Flutter se compila en código de máquina ARM o Intel, así como en JavaScript, para un rendimiento rápido en cualquier dispositivo.
- **Productividad:** Permite actualizar el código y ver los cambios casi al instante, sin perder el estado.
- **Flexibilidad:** El motor de renderizado Flutter te permite controlar cada píxel y su biblioteca de widgets se adapta automáticamente a cualquier pantalla.
- **Rendimiento nativo:** Flutter se compila en código de máquina nativo para aplicaciones rápidas.
- Nos brinda el control en nuestra base de código con pruebas automatizadas, herramientas para desarrolladores, entre otras.

Desventajas

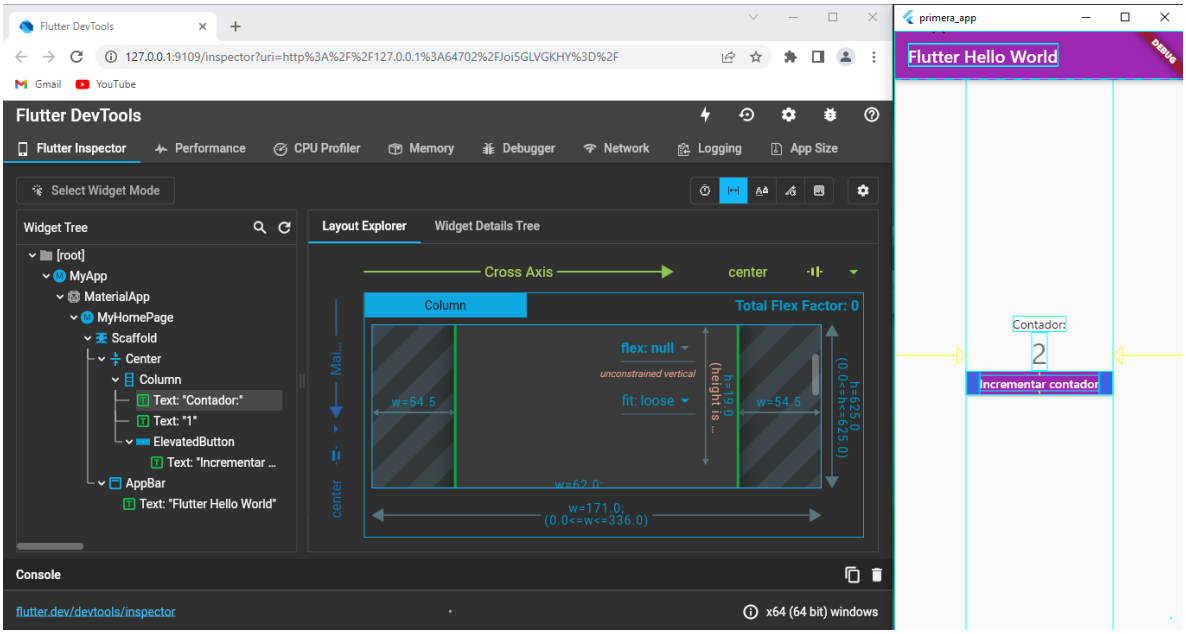
- El código del programa puede volverse confuso al integrar los widgets.
- En caso de actualizar aspectos del diseño en los sistemas operativos, hay que actualizar los módulos Flutter. Como los módulos se integran en el programa de manera fija, también hay que compilar el programa e instalarlo en los dispositivos.
- Las aplicaciones tienden a ser pesadas o tener archivos de gran tamaño.

Herramientas de desarrollo (DevTools)

DevTools es un conjunto de herramientas de rendimiento y depuración para Dart y Flutter.

Estas son algunas de las cosas que se puede hacer con DevTools:

- Inspeccionar el diseño de la interfaz de usuario y el estado de una aplicación de Flutter.
- Diagnostica problemas de rendimiento de bloqueo de la interfaz de usuario en una aplicación de Flutter.
- Perfilado de CPU para una aplicación Flutter o Dart.
- Creación de perfiles de red para una aplicación Flutter.
- Depuración a nivel de fuente de una aplicación Flutter o Dart.
- Depura problemas de memoria en una aplicación de línea de comandos Flutter o Dart.
- Ver información general de registro y diagnóstico sobre una aplicación de línea de comandos de Flutter o Dart en ejecución.
- Analizar el código y el tamaño de la aplicación.



Lo nuevo de Flutter 3.3 y Dart 2.18

Flutter 3.3

Esta versión trae actualizaciones para Flutter web, escritorio, mejoras en el manejo de texto en el rendimiento. Podemos mencionar las siguientes:

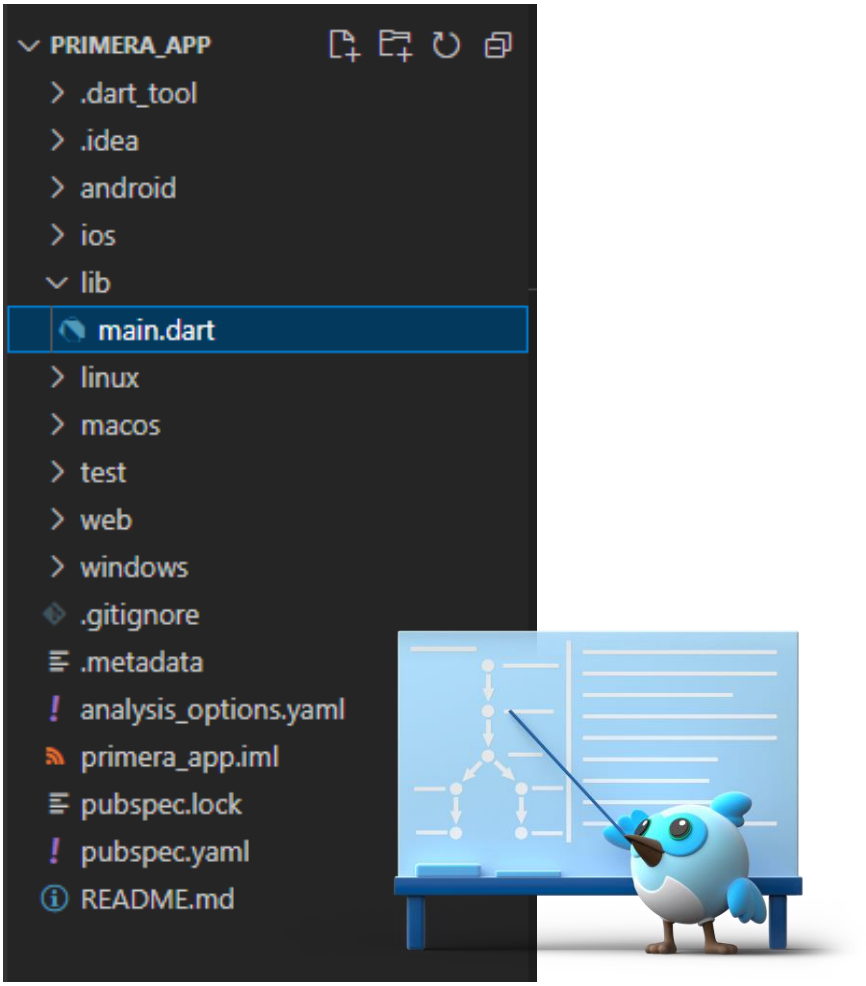
- Estructura
    - Selección global: Con la introducción del SelectableAreawidget, cualquier hijo del SelectableAreawidget tiene la selección habilitada de forma gratuita.
    - Entrada de panel táctil: Brinda soporte mejorado para la entrada del trackpad.
    - Flutter ahora es compatible con la entrada de escritura a mano de Scribble con Apple Pencil en iPadOS. Esta función está habilitada de manera predeterminada en CupertinoTextField, TextFieldy EditableText.
- Diseño de materiales 3
    - Esta versión incluye actualizaciones de IconButton, Chipsy variantes grandes y medianas para AppBar.
- Mejoras en la extensión de VS Code: Ahora se puede agregar múltiples dependencias separadas por comas en un solo paso usando Dart: Add Dependency .
- Actualizaciones de Flutter DevTools: Mejoras de UX y rendimiento en las tablas de visualización de datos para un desplazamiento más rápido y menos nervioso de grandes listas de eventos.

Dart 2.18

Esta versión presenta una vista previa de la interoperabilidad de Objective-C y Swift y un nuevo paquete de red iOS/macOS construido sobre esta interoperabilidad. También contiene una inferencia de tipo mejorada para funciones genéricas, mejoras de rendimiento para el código asíncrono, nuevas características de pub.dev y limpieza de nuestras herramientas y bibliotecas principales.

Estructura del proyecto

Al crear un proyecto en Flutter nos genera por defecto cada una de las carpetas y archivos.



Carpetas:

- idea: Usado para que Android Studio guarde su configuración para el proyecto en curso.
- android: En esta carpeta se genera todo lo necesario para crear una aplicación Android a partir del proyecto Flutter en lenguaje Java / Kotlin. Cuando se instala la aplicación en el emulador, lo que hace Android Studio es generar el código necesario en formato 'nativo' de Android dentro de esta carpeta. Normalmente no será necesario acudir a modificar nada en esta carpeta.

- ios: En esta carpeta se genera todo lo necesario para crear una aplicación IOS a partir del proyecto Flutter. Normalmente no será necesario acudir a modificar nada en esta carpeta.
- lib: Donde se encuentran normalmente los archivos que conforman el código fuente de la aplicación Flutter.
- test: Carpeta donde se guardan los archivos para realizar pruebas sobre la aplicación de Flutter

Archivos:

- .metadata: Administrado por Flutter automáticamente y se usa para controlar las propiedades del proyecto Flutter y actualizaciones. Este archivo solo lo cambia Flutter y no debe ser modificado manualmente.
- .gitignore: Indica todos los archivos y directorios que deben ser ignorados en caso de subir a un repositorio git nuestro proyecto.
- .pubspec.lock: Indica cómo se construye el archivo pubspec.yaml. No debe editarse.
- .pubspec.yaml: Archivo que permite:
    - Establecer la configuración general del proyecto como nombre, descripción y versión del proyecto
    - Indicar las dependencias del proyecto
    - Indicar archivos de recursos, como imágenes, audios, etc.
- nombreproyecto.iml: Este archivo siempre se nombra de acuerdo con el nombre del proyecto y contiene más configuraciones del proyecto Flutter. No debe modificarse.
- README.md: Archivo en la que aparece la descripción del proyecto. Es el archivo que aparece en un alojamiento github en internet.

Widgets básicos

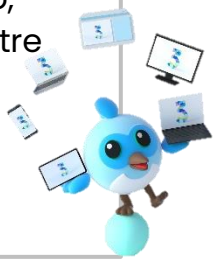
Los widgets de Flutter son construidos usando un moderno framework que toma inspiración de React. La idea central es que construyas la UI de widgets. Los Widgets describen cómo debería ser su vista, dada su configuración y estado actuales. Cuando el estado de un widget cambia, el widget reconstruye su descripción, que el framework difiere de la descripción anterior para determinar los cambios mínimos necesarios en el árbol de renderizado subyacente para la transición de un estado al siguiente.

Flutter viene con un conjunto de widgets básicos, de los cuales los siguientes son de uso muy común:

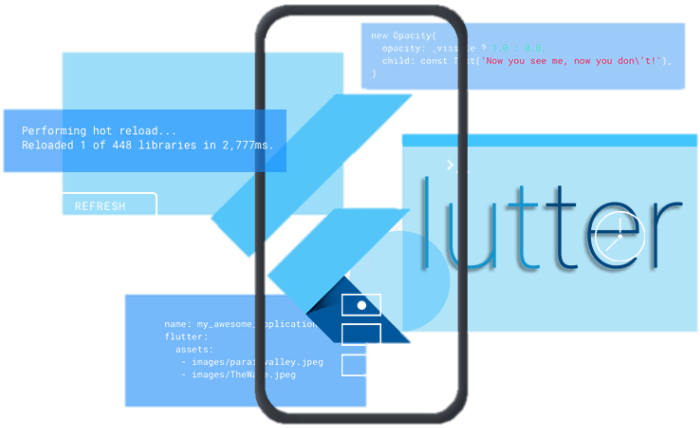
- **Text:** Representa un texto o cadena de caracteres, admite diversos parámetros o propiedades, además del texto literal que será mostrado. Podemos definirle estilos y características de apariencia particulares.
- **Row:** Permite organizar y posicionar otros widgets hijos en forma de fila o renglón.
- **Column:** Permite organizar y posicionar otros widgets hijos en forma de columna.
- **Stack:** Permite apilar widgets hijos con respecto al eje Z (elevación) en la interfaz.
- **Container:** Es uno de los widgets más versátiles de Flutter y permite organizar otros widgets como si fuera una caja o contenedor. Este widget es equivalente a los divs de html y admiten atributos similares de apariencia.

También dispone principalmente de dos tipos de widgets:

Widgets con estado o StatefulWidgets	Widgets sin estado o StatelessWidgets
• Son todos los widgets con los que el usuario de la aplicación puede tener una interacción directa. Por ejemplo: checkbox, radio, slider, form. Este tipo de widgets por lo general disparan algún evento, acción o comportamiento determinado como respuesta a la interacción con el usuario.	• Los widgets de este tipo no reaccionan ante el intento de interacción con el usuario. No tienen asociado una acción o comportamiento particular. Son estáticos en la interfaz. Por ejemplo: icono, texto, contendor con color, entre otros.



Ambos tipos de widgets heredan sus características y comportamientos de una clase principal Widget.



Bitácora

Instalación de flutter 3.3

Fecha: 7 de diciembre de 2022

Objetivos: Realizar una correcta instalación del framework Flutter, además de comprender aspectos básicos y su estructura.

Requisitos del sistema:

- Sistema operativo: Windows 10 o posterior (64 bits), basado en x86-64.
- Espacio en disco: 1,64 GB (no incluye espacio en disco para IDE/herramientas).
- Herramientas: Flutter depende de que estas herramientas estén disponibles en su entorno.
    - Windows PowerShell 5.0 o posterior
    - Git para Windows 2.x, con la opción usar Git desde el símbolo del sistema de Windows

Pasos:

1. Primero nos dirigimos a la siguiente liga para poder instalar Flutter y damos clic en el sistema operativo con el que contemos, en nuestro caso lo instalaremos en el sistema Windows.

   https://docs.flutter.dev/get-started/install

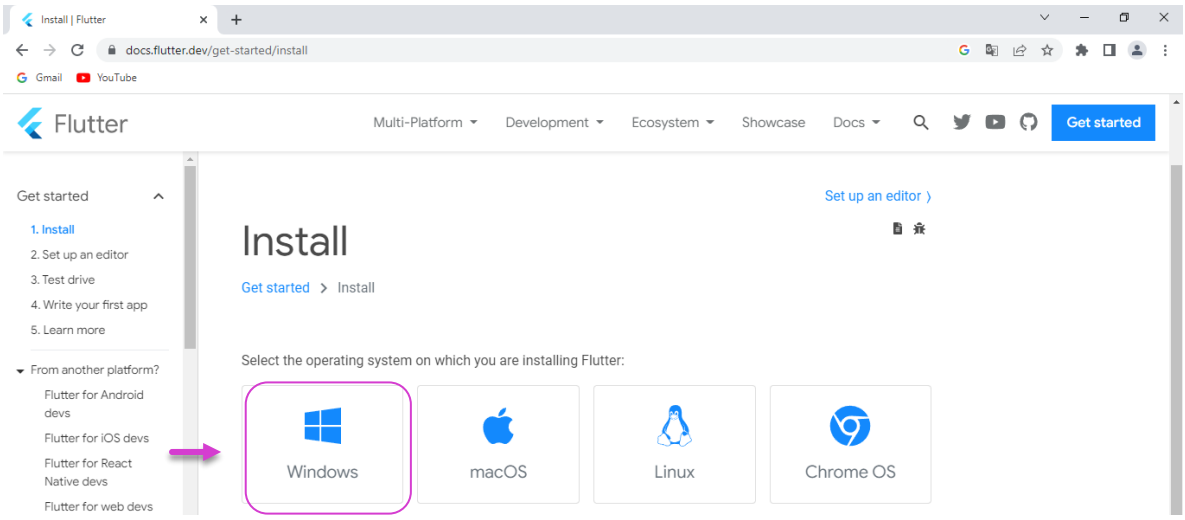


Figura 1. Elección del sistema operativo

2. A continuación, nos va a redirigir a la sección del sistema operativo que hemos seleccionado, indicando los requerimientos necesarios para su instalación. En esta misma sección vamos a descargar el paquete de instalación de para obtener la versión más reciente stable release del SDK Flutter.

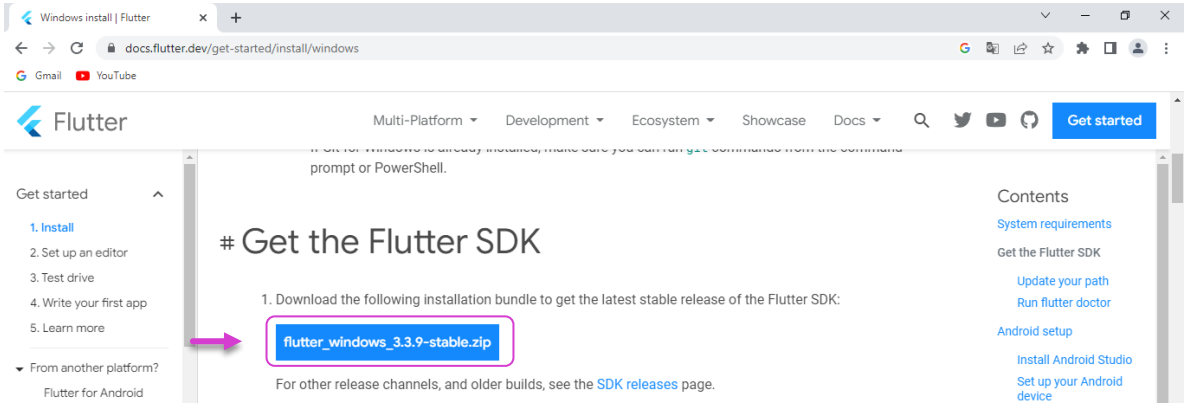


Figura 2. Descarga del SDK del Flutter

3. Ahora debemos extraer el archivo zip que hemos descargado en el paso anterior.

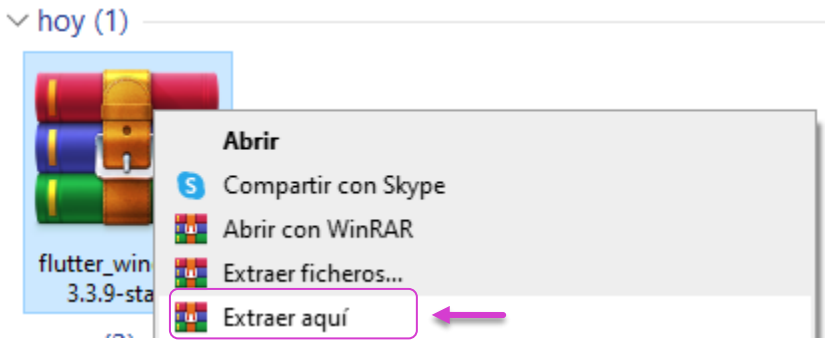


Figura 3. Extracción del archivo zip

4. Debemos colocar el contenido que se ha extraído en la ubicación que deseemos para Flutter SDK.

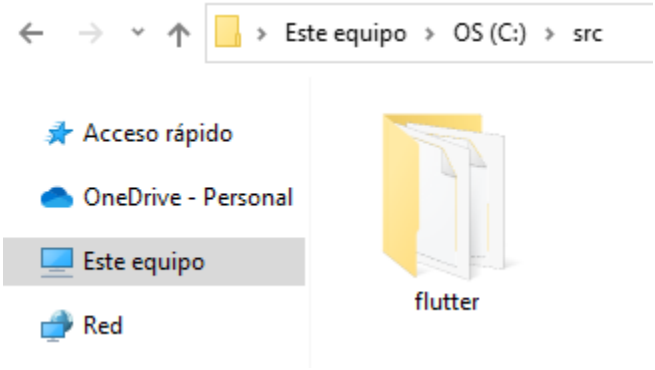


Figura 4. Mover el contenido al directorio C:\src\flutter

Configuracion del path

1. Desde la barra de búsqueda de inicio de Windows podemos buscar "Editar las variables de entorno del sistema"

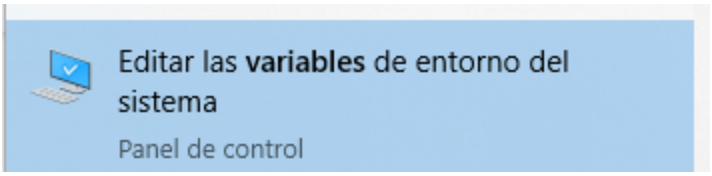


Figura 5. Variables de entorni

2. Después damos clic en el botón "Variables de entorno"

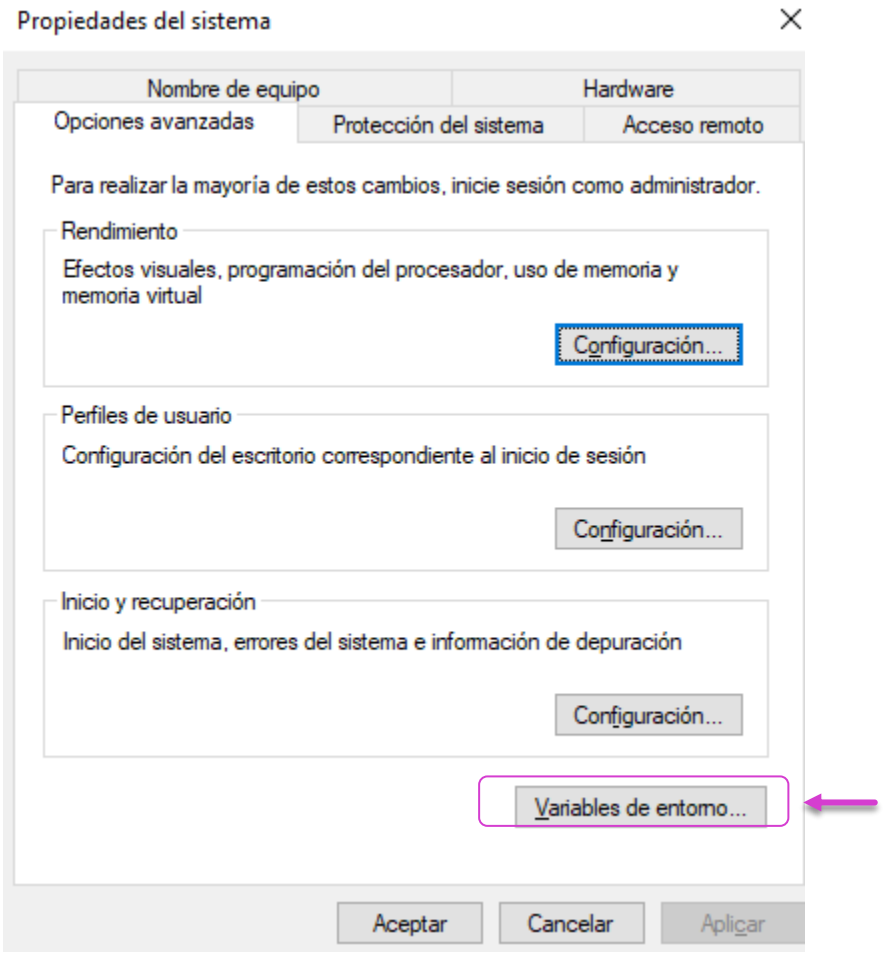


Figura 6. Editar las variables de entorno

3. Seleccionamos la variable del sistema “Path” y la editamos.

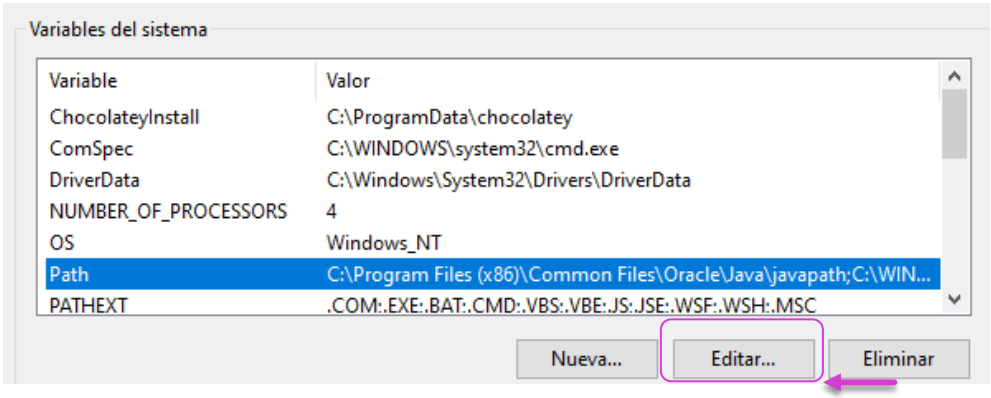


Figura 7. Editar el path

4. Agregamos en el “Path” la ruta C:\src\flutter\bin y damos clic en aceptar.

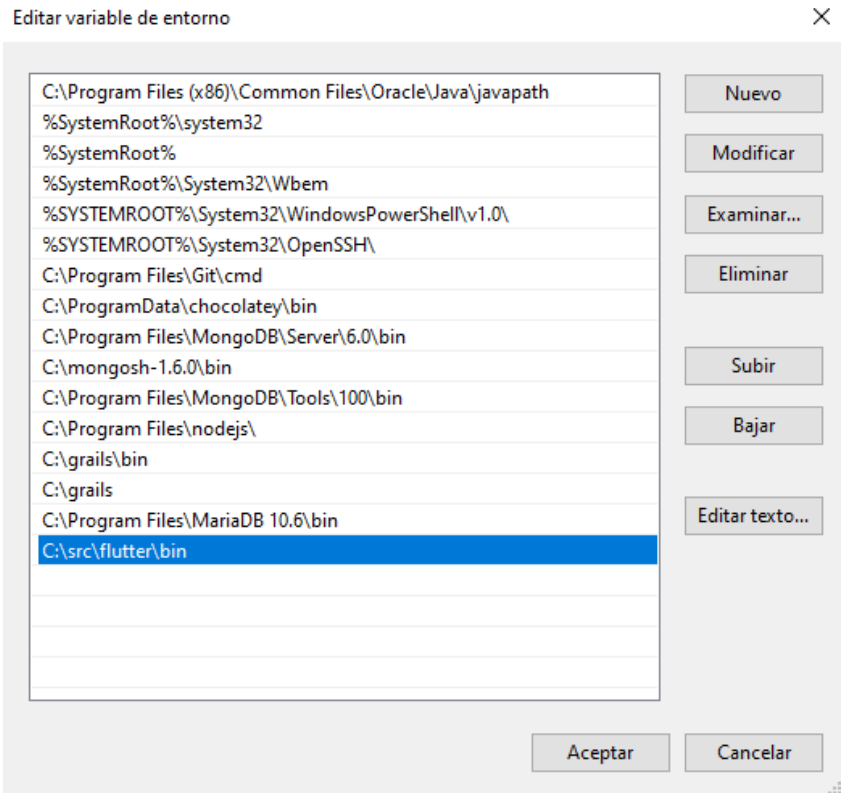


Figura 8. Agregar la ruta de flutter

Ejecutar flutter doctor

Desde una consola vamos a colocar el comando **"flutter doctor"** para ver si hay algunas dependencias de la plataforma que necesitemos para completar la configuración.

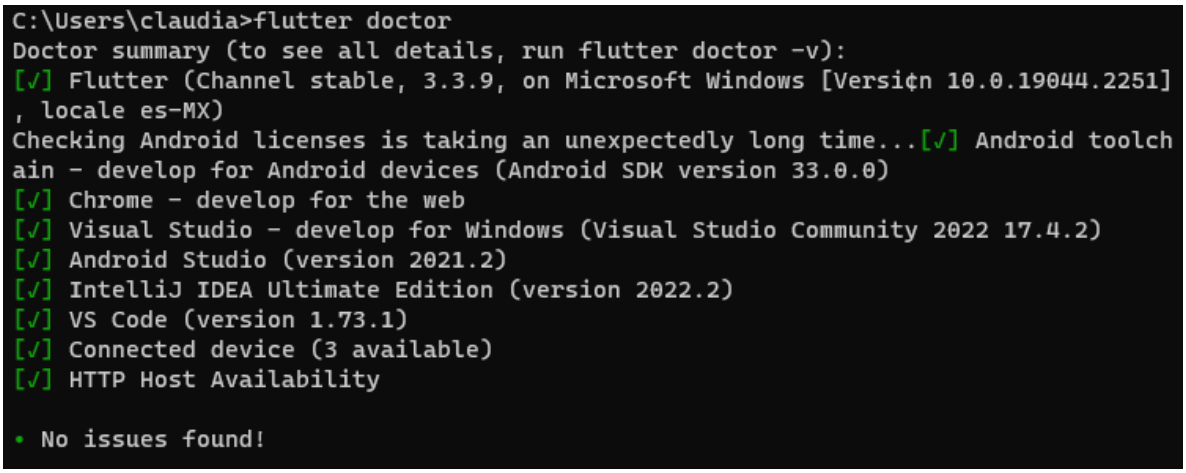


Figura 9. Ejecución de flutter doctor

Crear una aplicación en flutter

Pasos:

1. Primero debemos crear una carpeta el cual va a contener nuestro proyecto y lo arrastramos hacia Visual Studio Code.

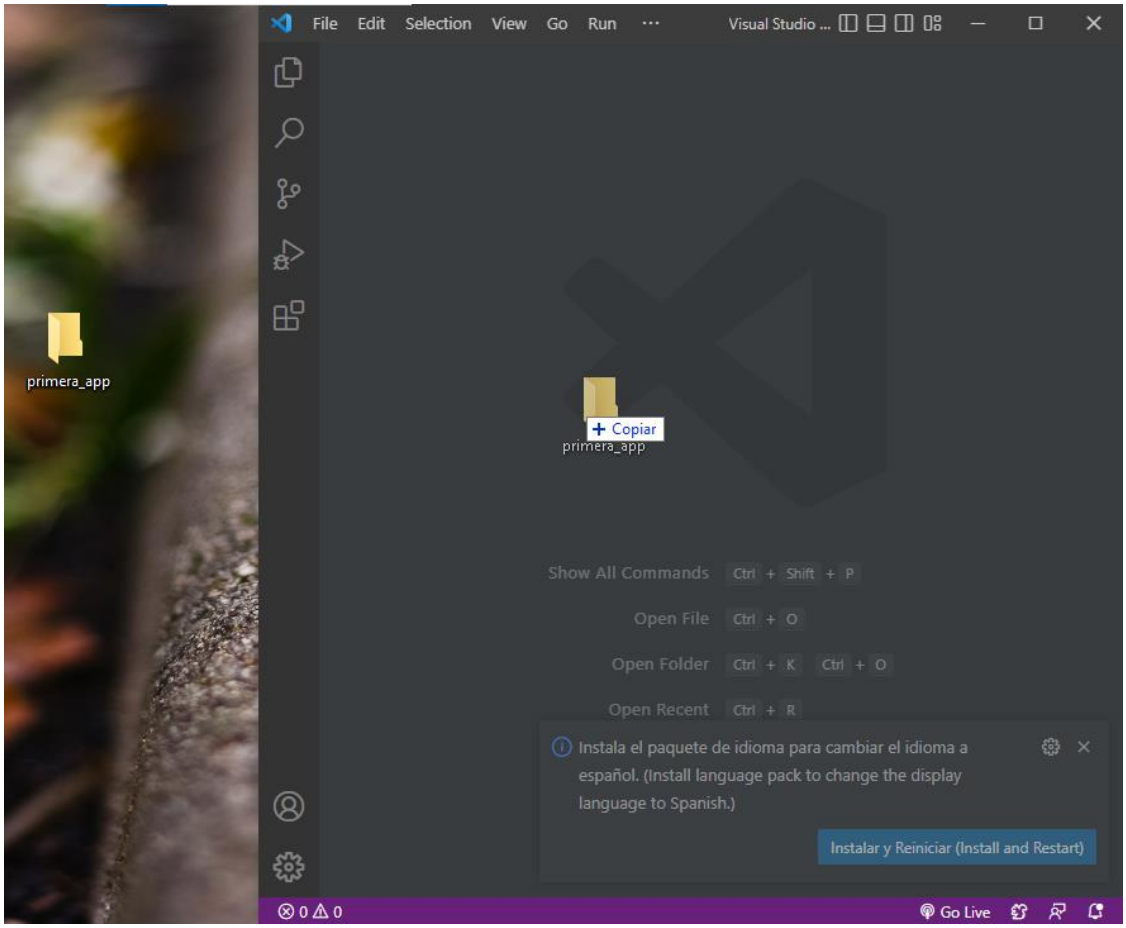


Figura 10. Abrir nuestra carpeta

2. Ahora nos dirigimos a la sección **"Terminal > New Terminal"** para poder ejecutar nuestros comandos de una manera sencilla en la ubicación de nuestra carpeta.

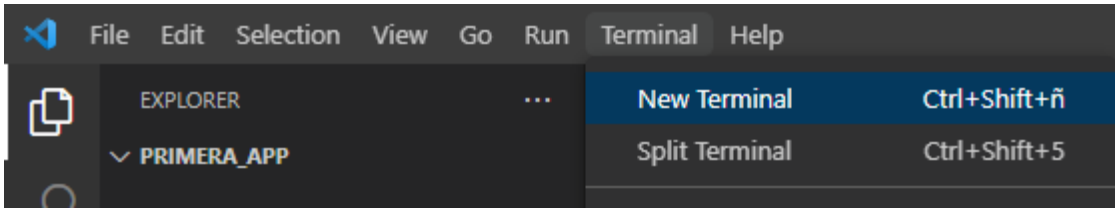


Figura 11. Abrir una nueva terminal

3. En la terminal que hemos abierto debemos colocar la instrucción **“flutter create . “** para poder crear nuestro proyecto.

```
PS C:\Users\claudia\Desktop\primera_app> flutter create .
```

Figura 12. Comando para crear nuestro proyecto en flutter

```
Creating project ....
Running "flutter pub get" in primera_app...                     3.8s
Wrote 127 files.

All done!
In order to run your application, type:

  $ cd .
  $ flutter run

Your application code is in .\lib\main.dart.
```

Figura 13. Ejecución del comando

4. Automáticamente nos va a crear los archivos en nuestra carpeta y nos a avisar que el proyecto principal está en **“lib› main.dart”**

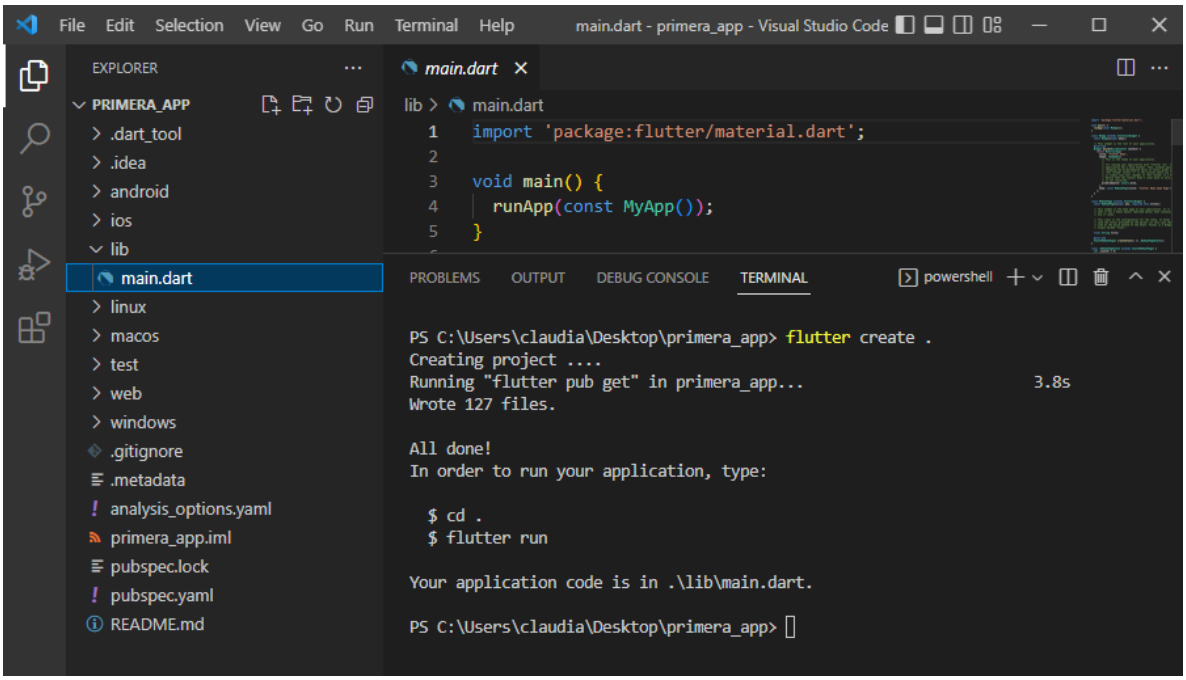


Figura 14. Estructura de nuestro proyecto

5. Si deseamos correr esta aplicación en nuestro navegador podemos utilizar el comando **“flutter run -d windows”**

```
PS C:\Users\claudia\Desktop\primera_app> flutter run -d windows
```

Figura 16. Comando para ejecutar la aplicación

```
PS C:\Users\claudia\Desktop\primera_app> flutter run -d windows
Launching lib\main.dart on Windows in debug mode...
Building Windows application...
Syncing files to device Windows...                                569ms

Flutter run key commands.
r Hot reload.
R Hot restart.
h List all available interactive commands.
d Detach (terminate "flutter run" but leave application running).
c Clear the screen
q Quit (terminate the application on the device).

 Running with sound null safety

An Observatory debugger and profiler on Windows is available at:
http://127.0.0.1:50798/nU2hBTZwQ58=/
The Flutter DevTools debugger and profiler on Windows is available at:
http://127.0.0.1:9100?uri=http://127.0.0.1:50798/nU2hBTZwQ58=/
```

Figura 17. Resultado de ejecución

Por defecto nuestra aplicación crea un contador que va aumentando cuando damos clic en el botón.

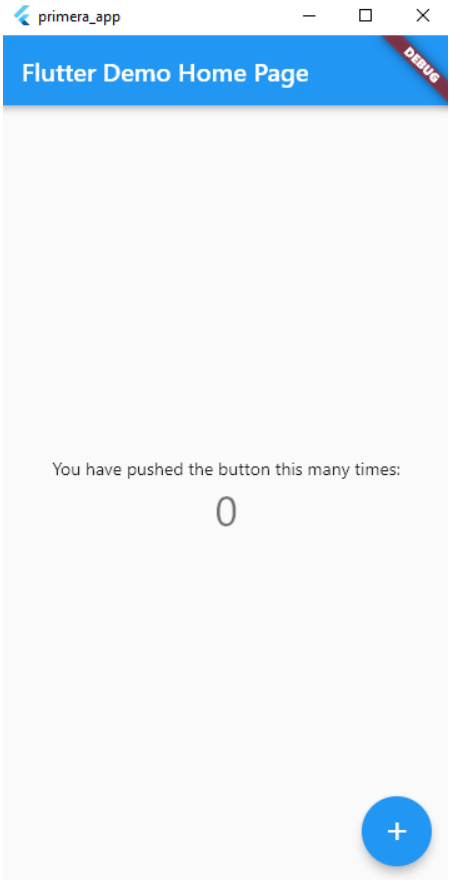


Figura 18. Aplicación inicial

6. A continuación, podemos describir e ir realizando cambios en el código que tenemos por defecto.
   La primera línea de código que generalmente nos vamos a encontrar en nuestros archivos .dart es:

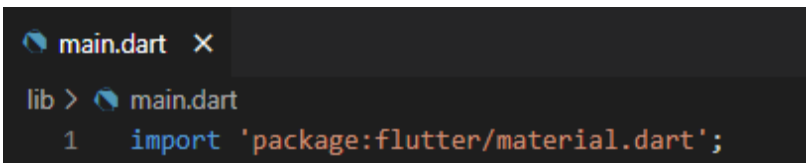


Figura 19. Importar el paquete de Material Design

Con la anterior instrucción importaremos a nuestro código el paquete de todos los Widgets de Material Design que son la base principal de clases con la que trabaja Flutter.

En Dart, el método main() será el punto de inicio de nuestra aplicación. Por su parte, runApp() es la función que inicializa y da vida a la aplicación definida por el Widget App.

```
3   void main() {
4     runApp(const MyApp());
5   }
```

Figura 20. Método main

MyApp() es la clase principal que representa la raíz de la aplicación. Esta clase extiende o hereda de StatelessWidget que es la clase padre de todos los widgets. MyApp() posee un método heredado llamado build() encargado de construir o mostrar todo el árbol de widgets anidados que hayamos definido con MaterialApp(), que es la clase que implementa Flutter para dar vida, forma y apariencia a todos los widgets de nuestra interfaz de usuario.

```
7   class MyApp extends StatelessWidget {
8     const MyApp({super.key});
9
10    @override
11    Widget build(BuildContext context) {
12      return MaterialApp(
13        title: 'Flutter',
14        theme: ThemeData(
15          primarySwatch: Colors.purple,
16        ),
17        home: const MyHomePage(title: 'Flutter Hello World'),
18      );
19    }
20  }
```

Figura 21. Clase MyApp

En la clase MyHomePage guardamos el atributo title y a continuación en el método createState devolvemos nuestro estado asociado que tendrá el método build que construye la vista.

```
22  class MyHomePage extends StatefulWidget {
23    const MyHomePage({super.key, required this.title});
24    final String title;
25
26    @override
27    State<MyHomePage> createState() => _MyHomePageState();
28  }
```

Figura 22. Clase MyHomePage

En la clase _MyHomePageState simplemente tendremos la función _incrementCounter para que cada vez que el usuario presione el botón sea llamada dicha función y aumente nuestro contador antes declarado.

```
30  class _MyHomePageState extends State<MyHomePage> {
31    int _counter = 0;
32
33    void _incrementCounter() {
34      setState(() {
35        _counter++;
36      });
37    }
```

Figura 23. Clase _MyHomePageState

A continuación, tenemos el Widget para especificar la interfaz de usuario de la página de inicio.

```dart
@override
Widget build(BuildContext context) {
  return Scaffold(
    appBar: AppBar(
      title: Text(widget.title),
    ),
    body: Center(
      child: Column(
        mainAxisAlignment: MainAxisAlignment.center,
        children: <Widget>[
          const Text(
            'Contador:',
          ),
          Text(
            '$_counter',
            style: Theme.of(context).textTheme.headline4,
          ),
          ElevatedButton(onPressed:_incrementCounter, child: const Text('Incrementar contador'))
        ],
      ),
    ),
  );
}
}
```

Figura 24. Widget MateralApp

7. Una vez realizados los cambios colocamos en la terminal la letra **"r"** para recarga la ejecución de nuestra aplicación.

```
PS C:\Users\claudia\Desktop\primera_app> flutter run -d windows
Launching lib\main.dart on Windows in debug mode...
Building Windows application...
Syncing files to device Windows...                                569ms

Flutter run key commands.
r Hot reload.
R Hot restart.
h List all available interactive commands.
d Detach (terminate "flutter run" but leave application running).
c Clear the screen
q Quit (terminate the application on the device).

 Running with sound null safety

An Observatory debugger and profiler on Windows is available at:
http://127.0.0.1:50798/nU2hBTZwQ58=/
The Flutter DevTools debugger and profiler on Windows is available at:
http://127.0.0.1:9100?uri=http://127.0.0.1:50798/nU2hBTZwQ58=/

Performing hot reload...
Reloaded 1 of 594 libraries in 821ms (compile: 145 ms, reload: 222 ms, reassemble: 196 ms).
```

Figura 25. Recargamos la aplicación

Y ya se verían los cambios realizados.

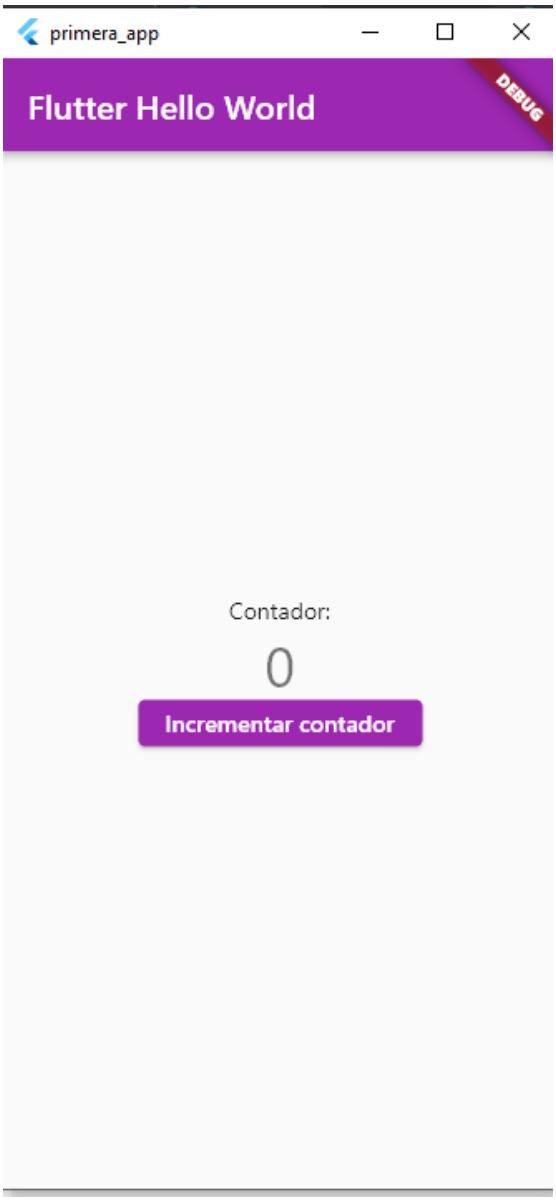


Figura 26. Ejecución de la aplicación

También podemos crear un **“hola mundo”** colocando el siguiente código en el archivo **“main.dart”**

lib > main.dart

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

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

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

  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'Welcome to Flutter',
       theme: ThemeData(
        primarySwatch: Colors.purple,
      ),
      home: Scaffold(
        appBar: AppBar(
          title: const Text('Welcome to Flutter'),
        ),
        body: const Center(
          child: Text('Hello World'),
        ),
      ),
    );
  }
}
```

Figura 27. Código para un “hola mundo”

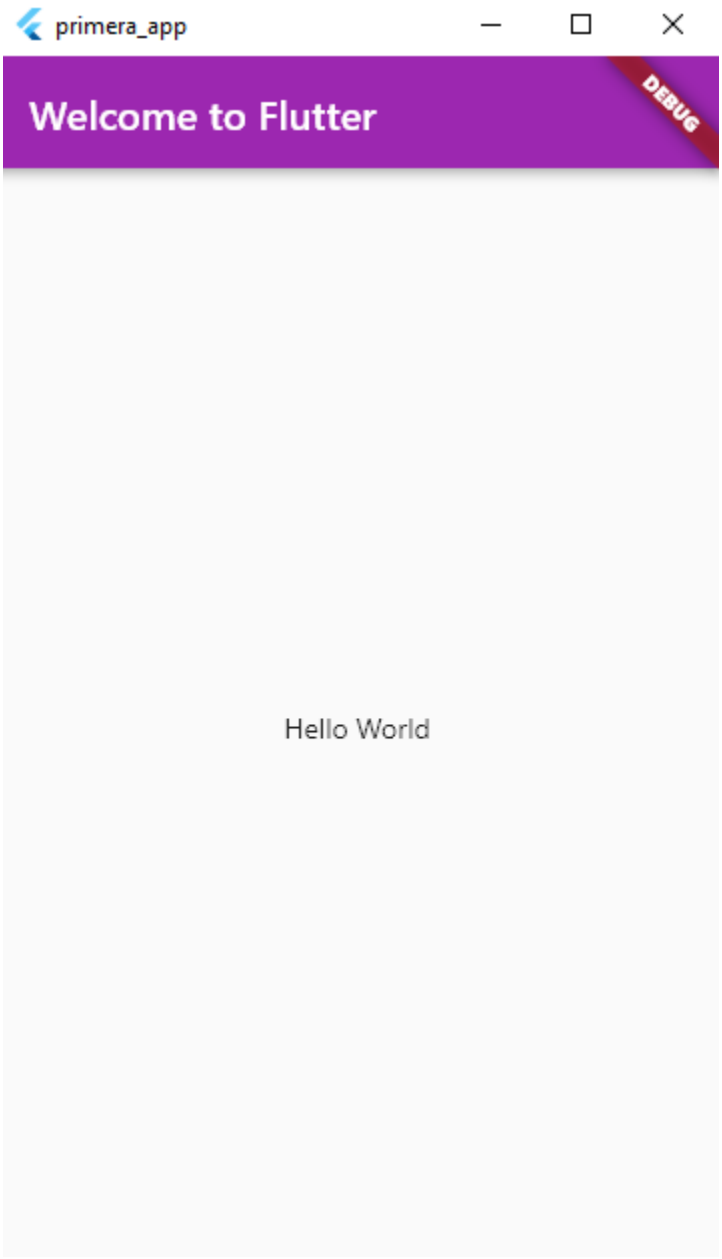


Figura 28. Ejecución del “hola mundo”

Generación del APK en Flutter

Pasos:

1. Una forma para crear el apk de nuestra aplicación es dirigiéndonos al directorio de nuestro proyecto y luego debemos ejecutar la siguiente instrucción:

flutter build apk --release

```
PS C:\Users\claudia\Desktop\primera_app> flutter build apk --release
```

Figura 29. Comando para crear el APK de flutter

2. Después de unos minutos nos indicará que se ha realizado exitosamente y nos proporcionará el directorio en donde fue creado.

```
Running Gradle task 'assembleRelease'...                          720.0s
√  Built build\app\outputs\flutter-apk\app-release.apk (16.5MB).
```

Figura 30. Mensaje exitoso

3. Nos dirigimos a la carpeta **"flutter-apk"** y vemos que se ha creado nuestro archivo como **"app-release.apk".**

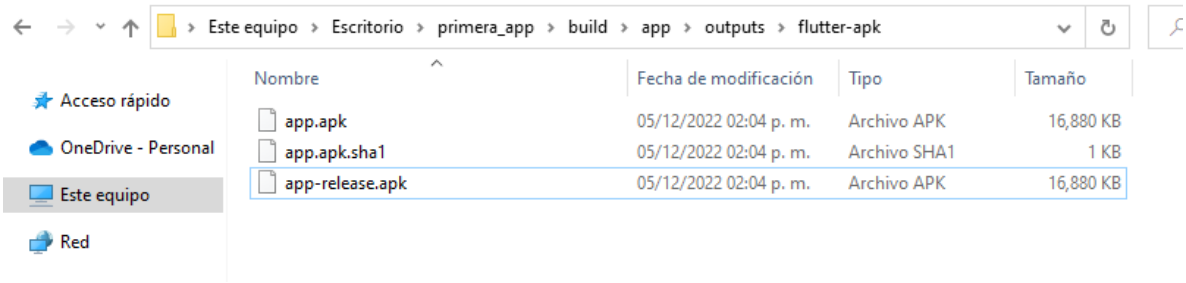


Figura 31. App-release.apk

Errores

Error de Android toolchain

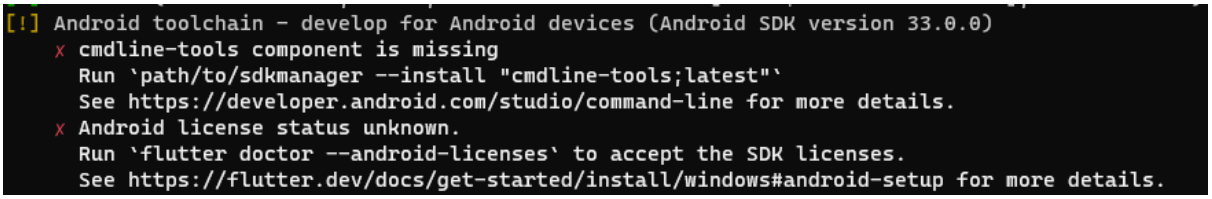


Figura 32. Error de Android toolchain

Solución:

1. Estando en el directorio C:\Users\claudia\AppData\Local\Android\Sdk\tools\bin debemos colocar la siguiente instrucción:

   sdkmanager.bat --install "cmdline-tools;latest"

Figura 33. Solución de cmdline-tools

2. Luego debemos colocar la instrucción para aceptar las licencias e ir colocando la letra “y” indicando que estamos de acuerdo con ellas.

   flutter doctor --android-licenses

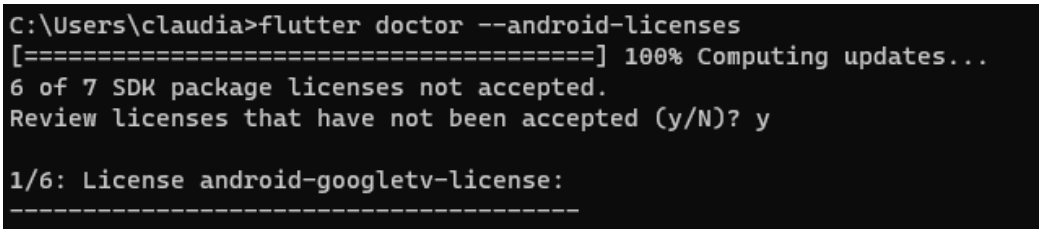


Figura 34. Aceptar las licencias de Android

Error en visual Studio

```
[x] Visual Studio - develop for Windows
    x Visual Studio not installed; this is necessary for Windows development.
      Download at https://visualstudio.microsoft.com/downloads/.
      Please install the "Desktop development with C++" workload, including all of its default components
```

Figura 35. Error de Visual Studio

Solución:

1. Instalamos “Desarrollo para el escritorio con C++” junto con la instalación de Visual Studio. https://visualstudio.microsoft.com/downloads/

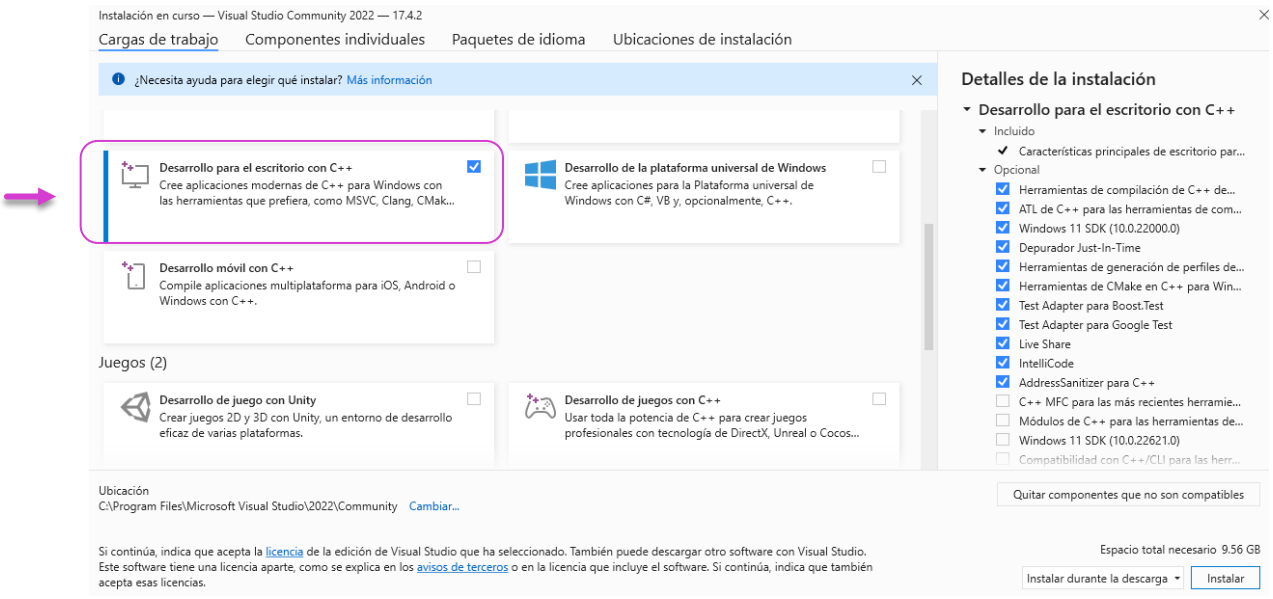


Figura 36. Instalación de “Desarrollo para el escritorio con C++”

Conclusión

Como ya vimos, las aplicaciones móviles híbridas tienen muchas ventajas, entre ellas podemos mencionar que permiten a los desarrolladores escribir un código para una aplicación móvil y que esta se pueda adaptar a múltiples plataformas, lo que significa que se reducirían los costos de desarrollo, ya que si desarrolláramos una aplicación nativa se tendría que mantener el doble de código.

También podemos mencionar, que existen frameworks para el desarrollo de aplicaciones híbridas como flutter que ofrece un desarrollo de código a una gran velocidad por su recarga en caliente, además de un rápido y constante renderizado de vistas, entre otras cosas.

Aunque también existen desventajas al desarrollar aplicaciones híbridas, la principal que podría mencionar es que no logran aprovechar al máximo los recursos de los dispositivos. Con relación a flutter las aplicaciones tienden a ser pesadas o tener archivos de gran tamaño.

Podemos concluir que los tres tipos de aplicaciones móviles son muy buenas, solo debemos de tomar en cuenta el tipo de aplicación que se adapte a nuestras necesidades.

Referencias

Bambu Editorial. (2022, 22 julio). *¿Qué es una app híbrida?* Bambú Mobile. https://www.bambu-mobile.com/que-es-una-app-hibrida/

Benito, M. (2022, 27 enero). *Conoce los mejores frameworks para aplicaciones híbridas*. FP Online. https://fp.uoc.fje.edu/blog/conoce-los-mejores-frameworks-para-aplicaciones-hibridas/

Chisholm, K. J. (2022, 16 septiembre). *What's new in Flutter 3.3 - Flutter*. Medium. https://medium.com/flutter/whats-new-in-flutter-3-3-893c7b9af1ff

Command-line tools /. (s. f.). Android Developers. https://developer.android.com/studio/command-line

Ibarra. (s. f.). *Aplicaciones móviles híbridas* (1.ª ed.). https://www.pucesi.edu.ec/webs2/wp-content/uploads/2021/02/Aplicaciones-M%C3%B3viles-H%C3%ADbridas-2020.pdf

MoureDev by Brais Moure. (2021, 28 diciembre). *FLUTTER: COMO Crear una APP DESDE CERO (para Principiantes)*. YouTube. https://www.youtube.com/watch?v=-pWSQYpkkjk

Platzi: Cursos online profesionales de tecnología. (s. f.). https://platzi.com/clases/1386-flutter/16319-estructura-de-un-programa-en-flutter/

Resumen Técnico. (s. f.). Flutter. https://esflutter.dev/docs/resources/technical-overview

What's new. (s. f.). Flutter. https://docs.flutter.dev/whats-new

Windows install. (s. f.). Flutter. https://esflutter.dev/docs/get-started/install/windows

Write your first Flutter app, part 1. (s. f.). Flutter. https://docs.flutter.dev/get-started/codelab