



INSTITUTO POLITÉCNICO NACIONAL

Manual técnico Aplicación móvil



SOLEYA

Una aplicación para la búsqueda y compra de zapatos dentro de su gran catálogo, disponible para toda la comunidad que busque el calzado que se ajuste a sus necesidades como consumidor.

Elaborado por el Equipo 1: *Mewtwo*

Integrantes:

- Gómez Pliego Carlos Uriel
- Gonzales Sabas Luis Ever
- Jiménez Rogel Sergio
- Mondragón Mejía Ángel Amed
- Paz Alonso Gabriel
- Pérez Galeana Jemmy Alondra

Profesor: Gustavo Martínez Vázquez

Fecha: 17 de junio de 2025





INDICE

1	Introducción.....	4
2	Objetivos del sistema.....	4
3	Público objetivo	4
4	Requerimientos técnicos.....	5
4.1	Requerimientos mínimos de Hardware	5
4.2	Requerimientos mínimos de Software	5
5	Herramientas utilizadas durante el desarrollo	6
5.1	Flutter	6
5.2	Dart.....	6
5.3	Android Studio	6
5.4	DB Browser for SQLite.....	7
5.5	Firebase.....	7
5.6	GitHub.....	7
6	INSTALACIÓN Y CONFIGURACION.....	8
6.1	Android Studio	8
6.1.1	Emulador	11
6.1.2	Flutter.....	13
6.1.3	DB Browser para SQLite	17
6.1.4	Firebase.....	19
6.1.5	Configuración de Firebase con Flutter.....	21
6.2	Instalación del proyecto.....	22
7	Arquitectura general.....	25
8	Estructura del proyecto	26
8.1	Modelos de datos	27
8.2	Pantallas	29
8.2.1	welcome_screen	29
8.2.2	custom_bottom_nav_bar	34
8.2.3	inicio	35
8.2.4	busquedascreen	40
8.2.5	favoritos_screen.....	41



8.2.6	cart_screen	42
8.2.7	perfil_screen	48
8.3	Proveedores	50
8.3.1	cart_provider	50
8.4	Servicios	50
8.4.1	db_local	51
8.4.2	firestore_service	52
8.5	Utilidades	53
8.5.1	Navigation	53
8.6	Widgets	54
8.7	animated_favorite_icon	54
8.8	animated_page_wrapper	55
8.9	animations	55
8.10	confetti_overlay	56
8.11	Custom_Page_Route	56
8.12	LoadingScreen	57
8.13	navigation_helper	57
8.14	particle_explosion	58
8.15	product_card	59
8.16	producto_tile	59
8.17	slide_fase_page_wrapper	60
8.18	SlideFadeIn	60
8.19	Main	61
9	Configuración	63
9.1	firebase_options	63
9.2	firebase.json	63
9.3	pubspec.yaml	64



1 Introducción

En el presente manual técnico se documenta de manera detallada el desarrollo de la aplicación móvil “Zoleya”, una solución digital orientada a facilitar la venta de calzado (zapatos, tenis, sandalias, etc.) desde dispositivos móviles. La aplicación fue desarrollada utilizando el framework Flutter y el lenguaje de programación Dart, implementada y probada en el entorno Android Studio, con almacenamiento local mediante SQLite y funciones complementarias de autenticación utilizando Firebase.

Este documento está destinado a desarrolladores, técnicos de soporte y futuros mantenedores del sistema que requieran comprender el funcionamiento interno de la aplicación, su arquitectura, lógica de negocio, estructura de datos, herramientas utilizadas y procedimientos para su instalación, mantenimiento y actualización.

2 Objetivos del sistema

El desarrollo de la aplicación "**Soleya**" tuvo como propósito central brindar una solución tecnológica eficiente y fácil de usar para la venta de calzado en línea, tanto para el usuario final como para el administrador o propietario de la tienda.

Los objetivos específicos del sistema son:

- Permitir la visualización dinámica de productos organizados por nombre, precio, imagen y talla.
- Implementar funciones de búsqueda en tiempo real para facilitar el acceso a los productos deseados.
- Permitir al usuario marcar productos como favoritos y añadirlos a un carrito de compras.
- Mantener persistencia de datos mediante una base local SQLite.
- Garantizar el acceso mediante autenticación con Firebase.
- Ofrecer una interfaz amigable y optimizada para dispositivos móviles Android.

3 Público objetivo

El público objetivo de la aplicación "**Soleya**" está compuesto principalmente por:

- **Usuarios generales:** personas interesadas en adquirir calzado de forma rápida, práctica y desde su teléfono móvil, sin necesidad de asistir físicamente a una tienda.



- **Administradores y desarrolladores:** encargados del mantenimiento, actualización y evolución técnica de la aplicación.
- **Emprendedores y pequeños negocios de calzado:** que desean una herramienta accesible y personalizable para digitalizar sus ventas.
- **Estudiantes y docentes:** que pueden utilizar la aplicación como referencia para prácticas académicas relacionadas con desarrollo móvil y arquitectura de software.

4 Requerimientos técnicos

4.1 Requerimientos mínimos de Hardware

- Procesador: Intel Core i5 (4 núcleos) o AMD Ryzen 5
- Memoria RAM: 8 GB mínimo (16 GB recomendado)
- Disco duro: 10 GB libres (preferiblemente SSD)
- Pantalla: Resolución mínima de 1366x768
- Conectividad: Acceso a Internet (obligatorio para Firebase)
- USB Puerto USB disponible (para probar en dispositivos físicos)

4.2 Requerimientos mínimos de Software

- Sistema operativo: Windows 10 / macOS 11+ / Linux (64-bit)
- Android Studio: Arctic Fox 2020.3.1 o superior
- Flutter SDK: Versión 3.0.0 o superior
- Dart SDK: Versión 2.17.0 o superior
- Java JDK: Java 11 o superior
- DB Browser for SQLite: Última versión estable
- Navegador Web: Google Chrome o Mozilla Firefox (para documentación y acceso a Firebase)
- Emulador Android: Habilitado y configurado en Android Studio



5 Herramientas utilizadas durante el desarrollo

Durante el desarrollo de la aplicación "**Soleya**", se utilizaron diversas herramientas de software que facilitaron el diseño, programación, pruebas y gestión tanto la aplicación misma como de la base de datos.

5.1 Flutter

Framework de desarrollo multiplataforma utilizado para construir aplicaciones móviles con una interfaz moderna y fluida. Gracias a su sistema de widgets personalizados, permitió diseñar componentes reutilizables y altamente configurables, lo que facilitó la creación de una UI consistente en Android e iOS. Además, su arquitectura reactiva optimizó la actualización de la interfaz en tiempo real, mejorando la experiencia de usuario. Flutter también simplificó la integración con APIs externas y la gestión del estado, lo que ayudó en el desarrollo de funcionalidades avanzadas como el carrito de compras y la sincronización de datos con Firebase.

5.2 Dart

Dart, el lenguaje base de Flutter, aportó varias ventajas en la organización de la lógica del sistema. Su enfoque orientado a objetos permitió estructurar el código de manera modular, facilitando la escalabilidad de la aplicación. Además, su compilación ahead-of-time (AOT) optimizó el rendimiento en dispositivos móviles al convertir el código fuente en binario nativo. Con su integración a Flutter, Dart permitió manejar asincronía de manera eficiente mediante Futures y Streams, lo que fue fundamental para gestionar respuestas de bases de datos y conexiones a la nube sin bloquear el flujo de la aplicación.

5.3 Android Studio

IDE oficial que brindó un entorno completo para el desarrollo en Flutter. Android Studio destacó en varias áreas clave:

- **Emulación de dispositivos:** Facilitó la prueba en distintos tamaños de pantalla y versiones de Android sin necesidad de hardware físico.
- **Gestión de dependencias:** Gracias a su integración con Gradle, permitió manejar librerías y plugins de terceros de manera eficiente.
- **Depuración avanzada:** Proporcionó herramientas de perfilado, inspección de memoria y rastreo de rendimiento, lo que ayudó a optimizar la ejecución de la aplicación.



- **Soporte para múltiples plataformas:** Aunque se centró en Android, su compatibilidad con Flutter ayudó a ejecutar código y probar diseños también en iOS.



5.4 DB Browser for SQLite

Esta aplicación de escritorio fue clave para visualizar y depurar la base de datos local. En el desarrollo de la aplicación móvil, SQLite almacenó información esencial como:

- Listado de productos.
- Datos de usuarios favoritos.
- Historial de compras en el carrito.

DB Browser permitió la ejecución de **consultas SQL manuales**, verificación de integridad de los datos y edición de registros sin necesidad de modificar el código de la app, lo que aceleró el proceso de pruebas y corrección de errores.

5.5 Firebase

Plataforma de Google utilizada para gestionar la autenticación de usuarios y optimizar el almacenamiento de datos en la nube. Entre sus funciones principales se destacan:

- **Firebase Authentication:** Permitió gestionar el registro e inicio de sesión de usuarios utilizando correo electrónico, Google y otros métodos de autenticación.
- **Firestore:** Base de datos en la nube utilizada para almacenar y sincronizar datos en tiempo real, lo que ayudó a mantener actualizado el catálogo de productos.
- **Firebase Cloud Messaging:** Facilitó el envío de notificaciones push a los usuarios, mejorando la interacción con la aplicación.
- **Crashlytics:** Herramienta de monitoreo de errores que detectó fallas en tiempo real y ayudó a depurar problemas en producción.

5.6 GitHub

Sistema de control de versiones basado en la web. Permitió almacenar y compartir el código fuente del proyecto entre los integrantes del equipo de desarrollo, facilitando el trabajo colaborativo, seguimiento de cambios y gestión de versiones. Proporciono principalmente:



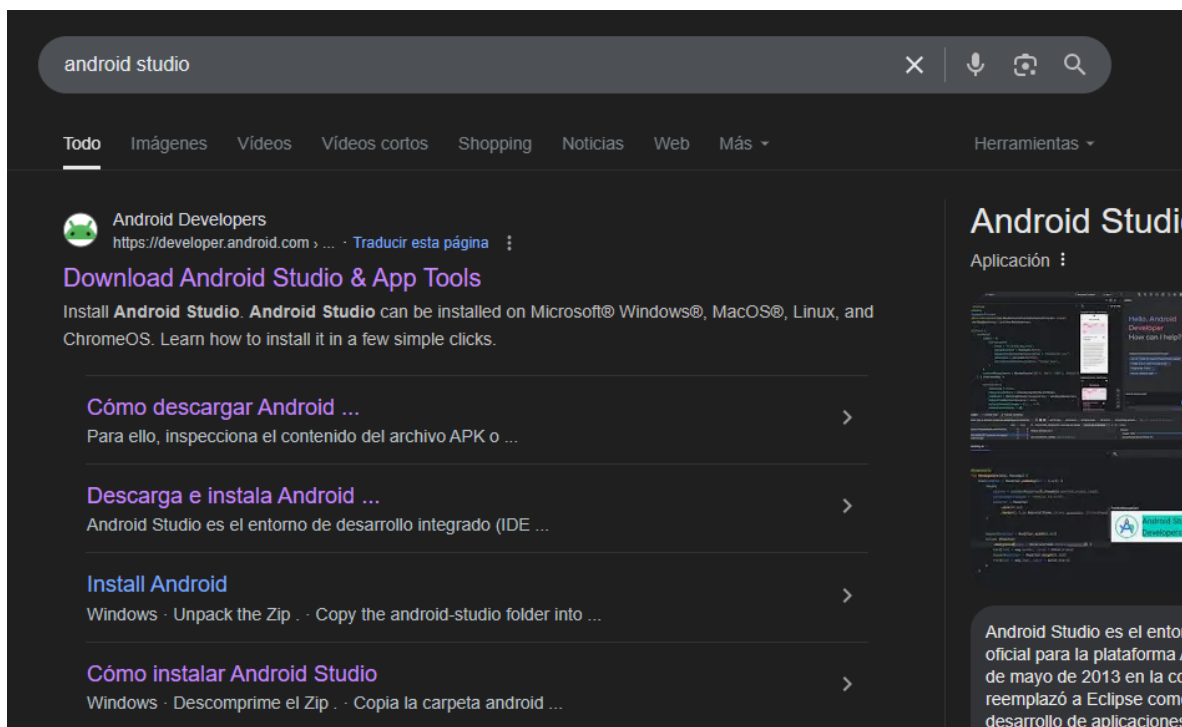
- **Control de versiones:** Facilitó el seguimiento de cambios en el código, permitiendo revertir a versiones anteriores cuando era necesario.
- **Trabajo colaborativo:** A través de repositorios compartidos, los desarrolladores pudieron trabajar en distintas ramas sin afectar la versión principal.
- **Integración continua:** GitHub Actions permitió automatizar pruebas y despliegues, mejorando la eficiencia del desarrollo.
- **Revisión de código:** Las herramientas de pull requests y comentarios permitieron realizar auditorías internas y garantizar calidad en cada actualización.

6 INSTALACIÓN Y CONFIGURACION

6.1 Android Studio

Lo primero, para comenzar a adentrarse en el proyecto habrá que obtener el IDE Android Studio a través del sitio web oficial, para ello es tan simple como buscar en un navegador “android studio” y seleccionar la opción que diga “Download Android Studio & App Tools” o seguir el siguiente enlace

<https://developer.android.com/studio?hl=es-419>



Ya dentro del enlace se mostrarán varias opciones, pero para descargar la versión más reciente de Android Studio que se ajuste a tu sistema operativo hay que



desplazarse hacia abajo hasta la sección “Descargas de Android Studio” y seleccionar la descarga según el sistema operativo



Plataforma	Paquete de Android Studio
Windows (64 bits)	android-studio-2024.3.2.15-windows.exe Recomendado
Windows (64 bits)	android-studio-2024.3.2.15-windows.zip Sin instalador .exe
Mac (64 bits)	android-studio-2024.3.2.15-mac.dmg
Mac (64 bits, ARM)	android-studio-2024.3.2.15-mac_arm.dmg
Linux (64 bits)	android-studio-2024.3.2.15-linux.tar.gz
ChromeOS	android-studio-2024.3.2.15-cros.deb



Antes de comenzar la descarga se necesitarán aceptar los Términos y Condiciones por el uso de la licencia del kit para desarrollo de Software de Android



14. Condiciones legales generales

14.1 El Contrato de Licencia representa en su totalidad el contrato legal entre usted y Google, regula el uso que haga proporcionarle conforme a un contrato por escrito independiente) y reemplaza por completo cualquier contrato anterior que, si Google no ejerce ni impone un derecho o solución legal especificado en el Contrato de Licencia (o del esto no se considerará como una renuncia a los derechos de Google y se entenderá que Google seguirá siendo bene que tenga jurisdicción para decidir sobre este asunto dictamina que alguna disposición de este Contrato de Licencia Contrato de Licencia. Las disposiciones restantes del Contrato de Licencia continuarán siendo válidas y aplicables. 14 de empresas de las que Google es la casa matriz serán beneficiarios terceros del Contrato de Licencia y que esas otr disposición del Contrato de Licencia que les confiera un beneficio (o que tengan derechos a su favor) y que podrán a ninguna empresa serán beneficiarios terceros del Contrato de Licencia. 14.5 RESTRICCIONES SOBRE LA EXPORTACIÓN DE ESTADOS UNIDOS. USTED DEBE CUMPLIR CON TODAS LAS LEYES Y NORMATIVAS INTERNACIONALES. ESTAS LEYES INCLUYEN RESTRICCIONES SOBRE LOS DESTINOS, LOS USUARIOS FINALES Y LA FINALIDAD. 14.6 Ni us otorgan en el Contrato de Licencia sin la aprobación previa por escrito de la otra parte. Ni usted ni Google podrán de Licencia sin la aprobación previa por escrito de la otra parte. 14.7 El Contrato de Licencia y la relación con Google qui estado de California, independientemente de su conflicto con las disposiciones legales. Usted y Google aceptan som condado de Santa Clara, California, para que resuelvan todo problema legal que surja de este Contrato de Licencia o aún podrá solicitar recursos judiciales (o una clase equivalente de compensación legal urgente) en cualquier jurisdic

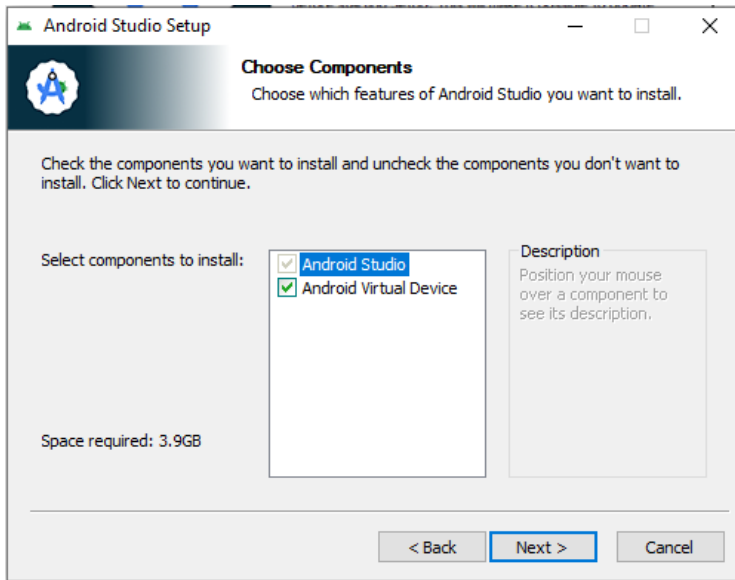
☐ Leí y acepto los Términos y Condiciones anteriores

Descargar Android Studio Meerkat Feature Drop | 2024.3.2 Patch 1 para Windows

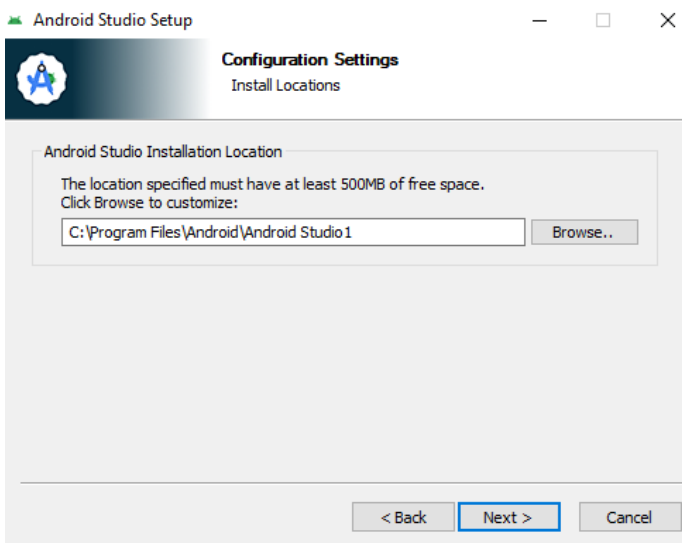
android-studio-2024.3.2.15-windows.exe

Ya terminada la descarga, para comenzar la instalación de Android Studio, desde el explorador de archivos con clic derecho sobre el archivo hay que seleccionar la opción “ejecutar como Administrador” o con doble clic sobre el archivo comenzara la descarga mostrando las siguientes ventanas para configurar la instalación





La opción Android Studio es para instalar la aplicación y la opción Android Virtual Device es para instalar las configuraciones necesarias para utilizar el



Se deberá escoger la ubicación en la que se instalará Android Studio

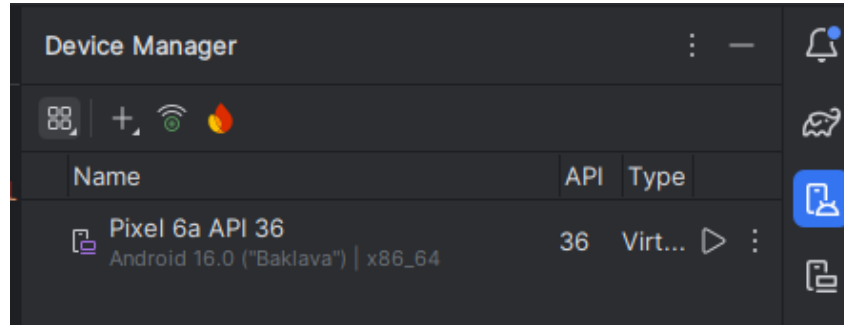
Y finalmente después de un tiempo Android Studio estará instalado con o sin el simulador de un dispositivo Android.

6.1.1 Emulador

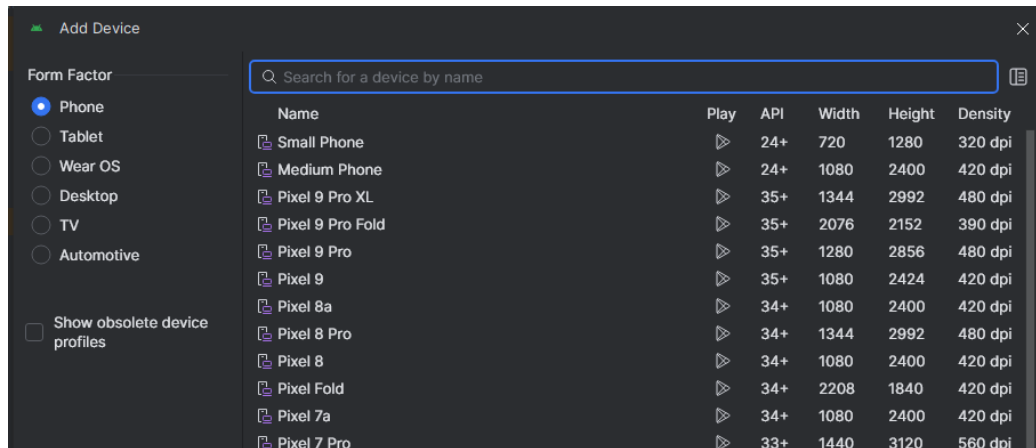
Para realizar pruebas de la aplicación sin utilizar un dispositivo físico conectado, se necesita utilizar un emulador de un dispositivo android, por lo tanto se deberá hacer lo siguiente



Del lado derecho se encuentra varios iconos, entre los cuales se encuentra el icono de “Device Manager”, con el cual se podrá observar el icono de ‘+’ el cual servirá para añadir un nuevo dispositivo



Después de seleccionar la opción Add a new device>Create Virtual Device se mostrará una pantalla con distintos tipos de dispositivos y a su vez distintos tipos de modelos a elegir.





Posterior a elegir el dispositivo y modelo, se mostrarán las especificaciones físicas del dispositivo junto con las configuraciones de este, como lo son: la versión de API, la versión de android, el nombre del dispositivo y los servicios disponibles.



Add Device

Configure virtual device

Device Additional settings

Name
Pixel 8a

Select system image
Use the filters to help find the system image that you prefer. The combination of device profile and system image is only an approximation of the equivalent physical hardware.

API
API 36 "Baklava"; Android 16.0 ⓘ

Services
Google Play Store ⓘ

System Image	API
★ Google Play Intel x86_64 Atom System Image	36
⬇️ Pre-Release 16 KB Page Size Google Play Intel x86_64 Atom System Image	36

☐ Show system images with SDK extensions ⓘ

☐ Show unsupported system images

Device
OEM Google
Supported API Levels 34+

System Image
API Level 36
Services Google Play
ABI x86_64
Translated ABI arm64-v8a

Screen
Resolution 1080 × 2400
Density 420 dpi

Cancel Previous Next **Finish**

Ya descargadas y establecidas las configuraciones del dispositivo se terminará el proceso de configuración con el botón “Finish” y el dispositivo estará listo para utilizarse.

6.1.2 Flutter

La instalación de Flutter se realizará a través de la página, para encontrar dicha página se deberá buscar en el navegador “Flutter” y seleccionar la opción “Get Started”, o puedes seguir enlace que te llevara al comienzo del proceso de instalación

https://docs.flutter.dev/get-started/install?gclid=aw.ds&gad_source=1&gad_campaignid=19926981051&gclid=Cj0KCQjwmK_CBhCEARIsAMKwcD5D4AoaNrI9PFmOX044Cj9v8nTda4QmC1YtWj0v60WqnF-dVO9RPdoaAodbEALw_wcB

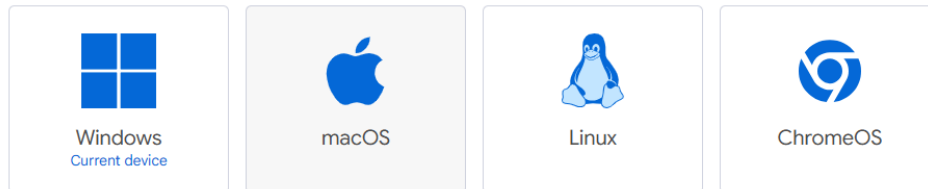


Dentro de la página se deberá seleccionar el tipo de sistema operativo del dispositivo en el que se instalará Flutter



Choose your development platform to get started

[Get started](#) > [Install](#)



① Developing in China

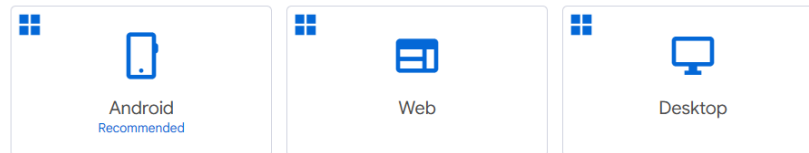
If you want to use Flutter in China, check out [using Flutter in China](#). If you're not developing in China, ignore this notice and follow the other instructions on this page.

如果你正在中国的网络环境下配置 Flutter，请参考 [在中国网络环境下使用 Flutter](#) 文档。

Después se debe seleccionar el ecosistema en el que se realizarán las futuras aplicaciones

Choose your first type of app

[Get started](#) > [Install](#) > [Windows](#)



Lo siguiente será descargar el SDK comprimido de Flutter, extraer el SDK en la ubicación donde se quiera guardar y añadir Flutter a las variables de entorno

Primero instalar Flutter haciendo clic en “Flutter_windows_[version]-stable.zip” y esperar a que se termine de descargar el archivo

Download then install Flutter

To install Flutter, download the Flutter SDK bundle from its archive, move the bundle to where you want it stored, then extract the SDK.

1. Download the following installation bundle to get the latest stable release of the Flutter SDK.

[flutter_windows_3.32.4-stable.zip](#)

For other release channels, and older builds, check out the [SDK archive](#).

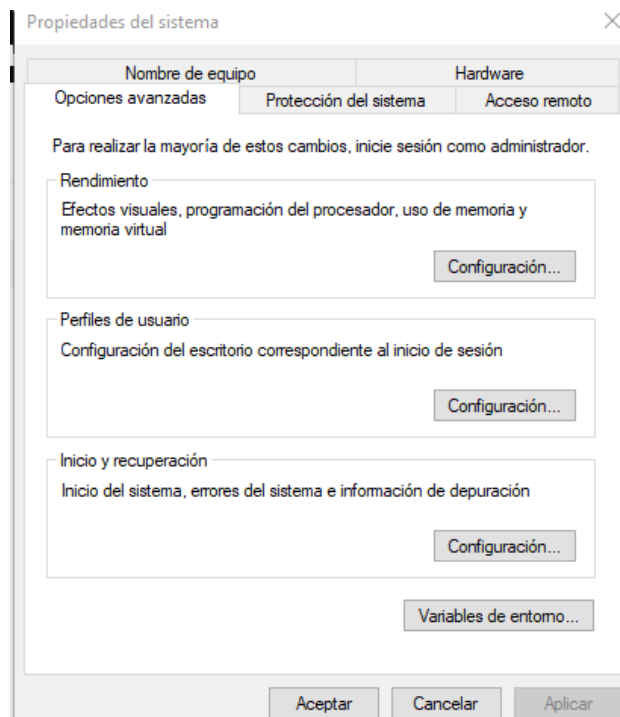
The Flutter SDK should download to the Windows default download directory: `%USERPROFILE%\Downloads`.

If you changed the location of the Downloads directory, replace this path with that path. To find your Downloads directory location, check out this [Microsoft Community post](#).



Ya descargado se deberá descomprimir el archivo dejando una carpeta llamada “flutter” con todos los datos para programar con flutter, entre los cuales se encuentra la carpeta bin que servirá para establecer la variable de entorno.

Para entrar a las variables de entorno desde Windows el método más simple es escribir en el buscador “Editar las variables de entorno del sistema” y seleccionar la opción resultante, de forma que se realizara lo siguiente



Propiedades del sistema:

Entre las opciones se deberá seleccionar “Variables de entorno”

Editar variables de entorno:

Dentro de este recuadro se deberá de pegar la dirección de la carpeta bin dentro de la carpeta flutter, para ello haciendo doble clic en un espacio vacío permitirá editar ese espacio y así pegar la dirección, ya para finalizar solo abra que cerrar las pestañas haciendo clic en “Aplicar” y/o “Aceptar” .



Variables de entorno:

La sección que debe ser configurada es “Path” en la sección “Variables del sistema”



Variables de usuario para Charly

Variable	Valor
OneDrive	C:\Users\DELL\OneDrive
OneDriveConsumer	C:\Users\DELL\OneDrive
Path	C:\Users\DELL\AppData\Local\Programs\Python\Launcher\;C:\Use...
TEMP	C:\Users\DELL\AppData\Local\Temp
TMP	C:\Users\DELL\AppData\Local\Temp

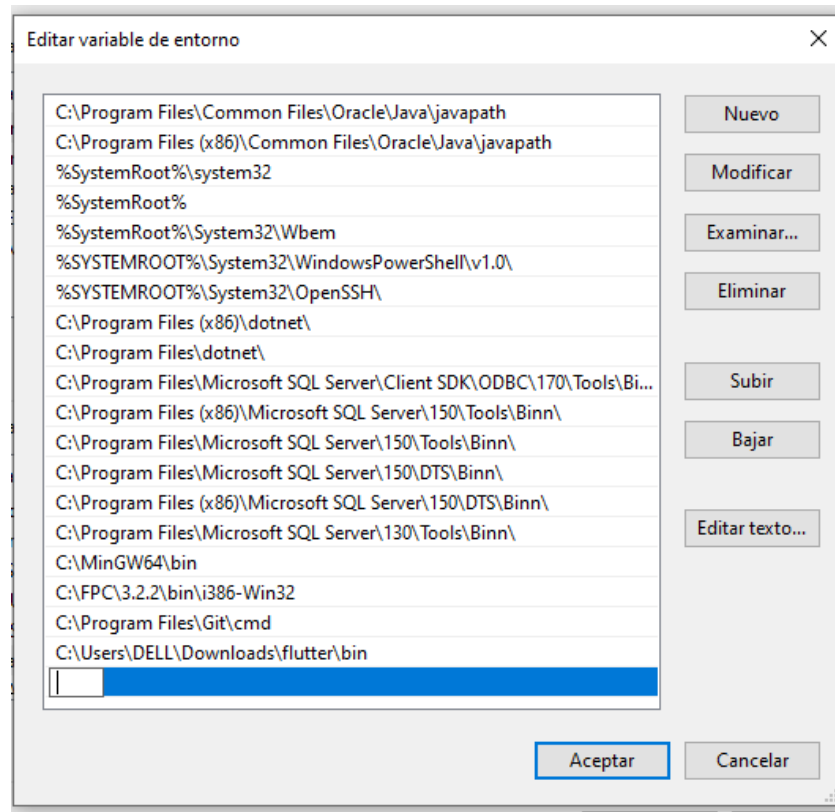
Nueva... Editar... Eliminar

Variables del sistema

Variable	Valor
Path	C:\Program Files\Common Files\Oracle\Java\javapath;C:\Program ...
PATHEXT	.COM;.EXE;.BAT;.CMD;.VBS;.VBE;.JS;.JSE;.WSF;.WSH;.MSC
PROCESSOR_ARCHITECTURE	AMD64
PROCESSOR_IDENTIFIER	Intel64 Family 6 Model 58 Stepping 9, GenuineIntel
PROCESSOR_LEVEL	6
PROCESSOR_REVISION	3a09
PSModulePath	%ProgramFiles%\WindowsPowerShell\Modules;C:\Windows\svste...

Nueva... Editar... Eliminar

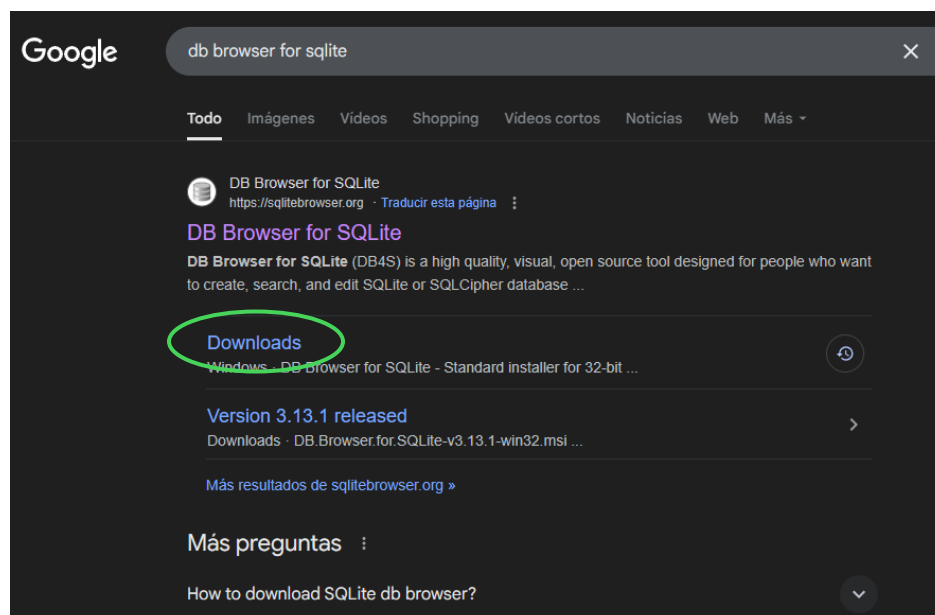
Aceptar Cancelar



6.1.3 DB Browser para SQLite

Para observar y probar la base de datos con sqlite se utilizó la herramienta DB Browser para SQLite.

La instalación es a través del enlace dentro de su página web





Dentro de la página habrá distintos enlaces para descargar la versión de la app que se ajuste a tu sistema operativo



[About](#) [Download](#) [Blog](#) [Docs](#) [GitHub](#) [Gitter](#) [Patreon](#)

Downloads

([Please consider sponsoring us on Patreon](#) 😊)

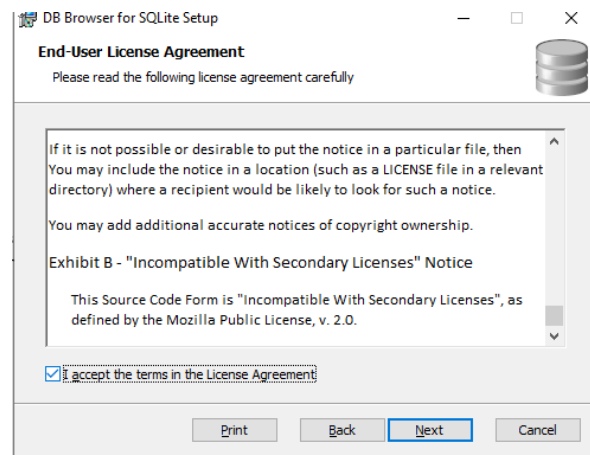
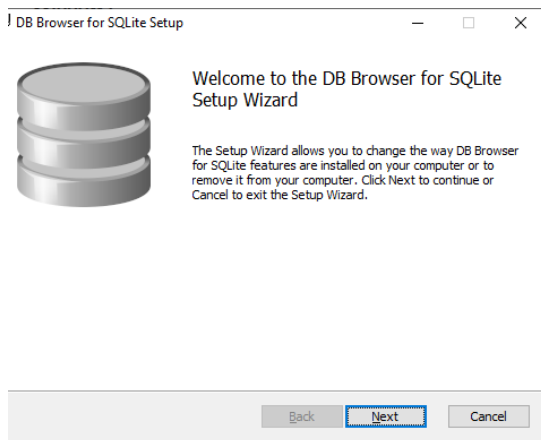
Windows

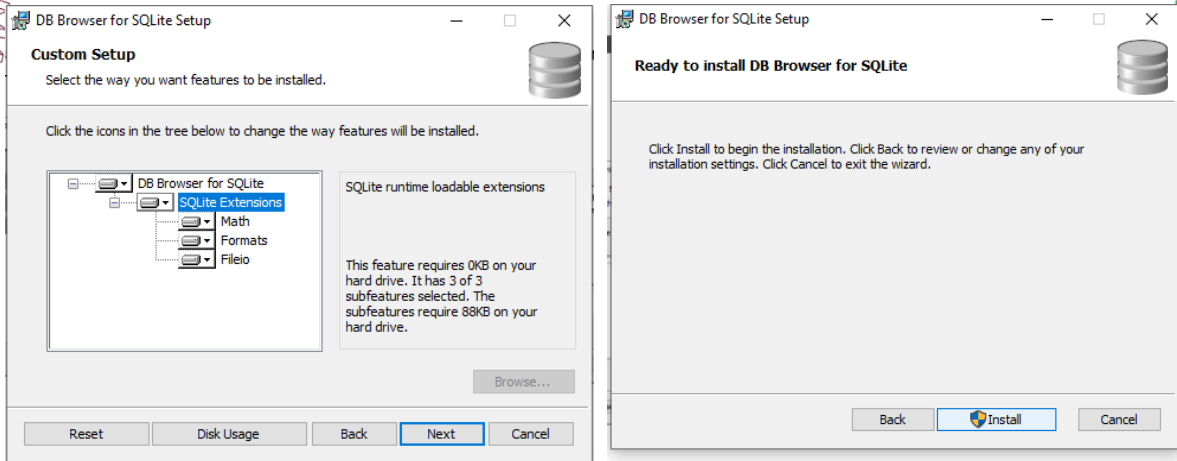
Our latest release (3.13.1) for Windows:

- [DB Browser for SQLite – Standard installer for 32-bit Windows](#)
- [DB Browser for SQLite – .zip \(no installer\) for 32-bit Windows](#)
- [DB Browser for SQLite – Standard installer for 64-bit Windows](#)
- [DB Browser for SQLite – .zip \(no installer\) for 64-bit Windows](#)

Free code signing provided by [SignPath.io](#), certificate by [SignPath Foundation](#).

Ya descargado, solo queda ejecutarlo como administrador y aceptar o rechazar las opciones para que la instalación comience.

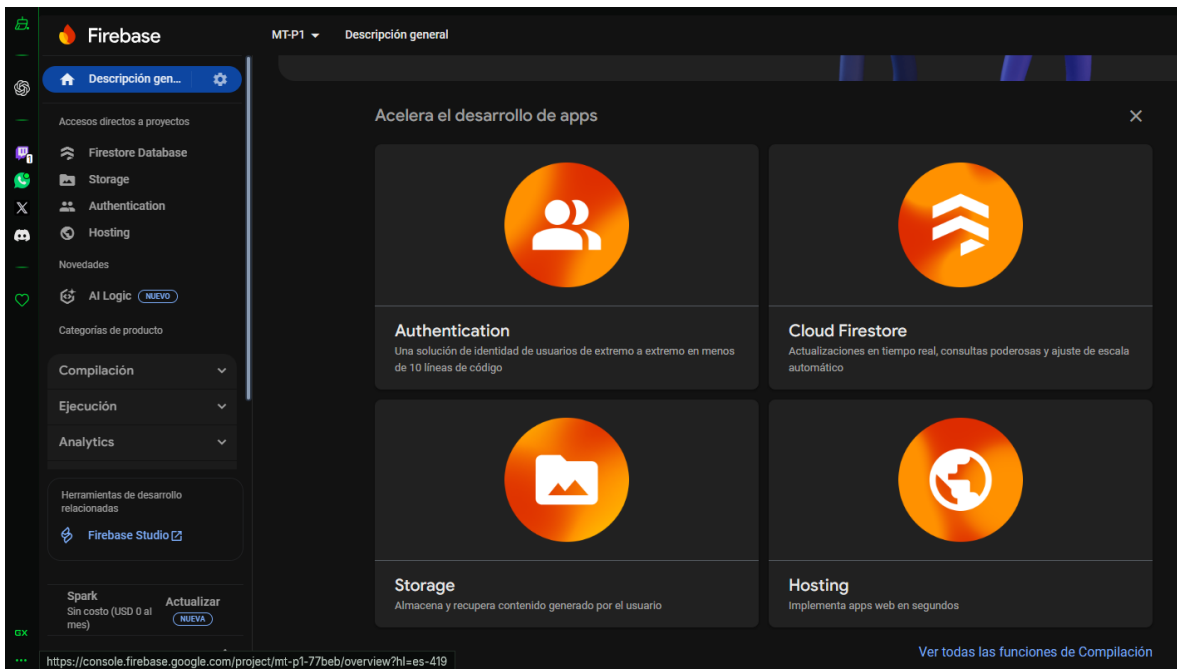




6.1.4 Firebase

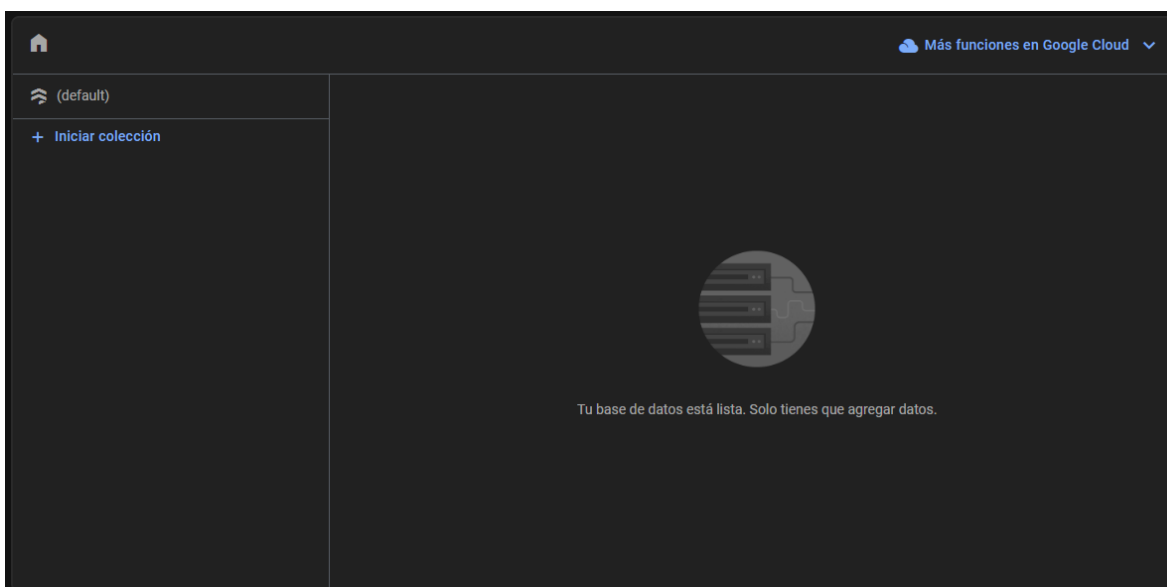
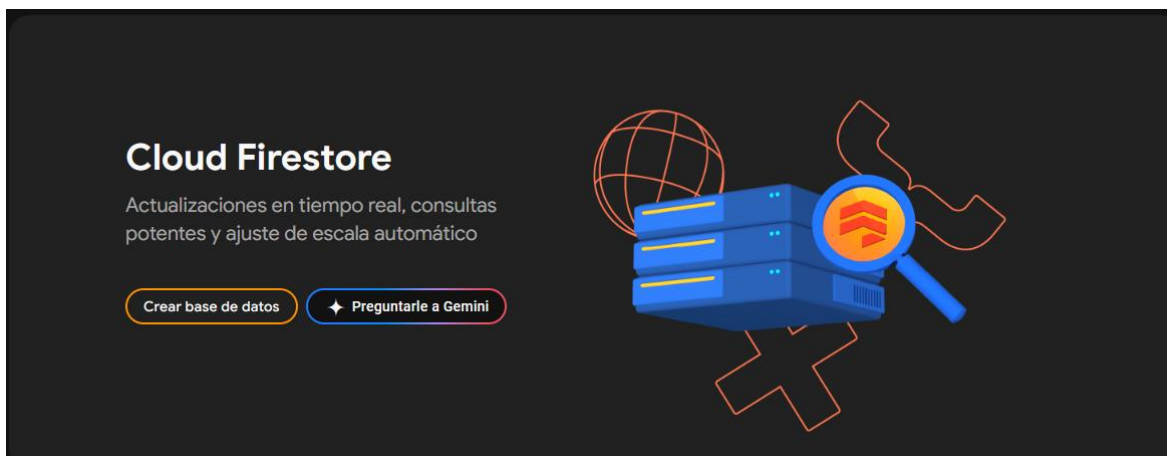
Firebase necesitara de una cuenta de Google para crear y administrar los proyectos a desarrollarse

Después de realizar el registro de la cuenta tendrás la opción de crear tu proyecto, por lo que tendrás que darle un nombre al proyecto y aceptar o negar Gemini y Google analytics para terminar de crear el proyecto de Firebase.





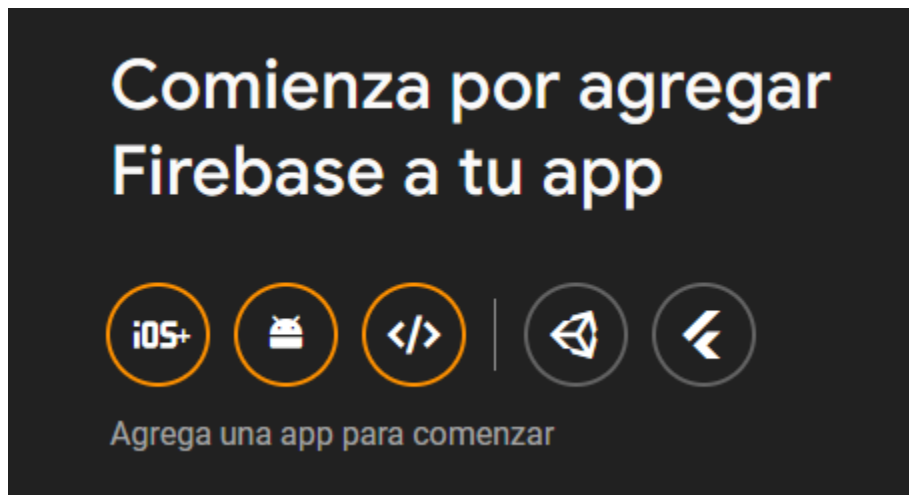
Para crear la base de datos hay que seleccionar la opción “Cloud firestore”, seleccionar “Crear base de datos” y configurar el nombre, la ubicación y las reglas de seguridad en base a lo que se necesite.





6.1.5 Configuración de Firebase con Flutter

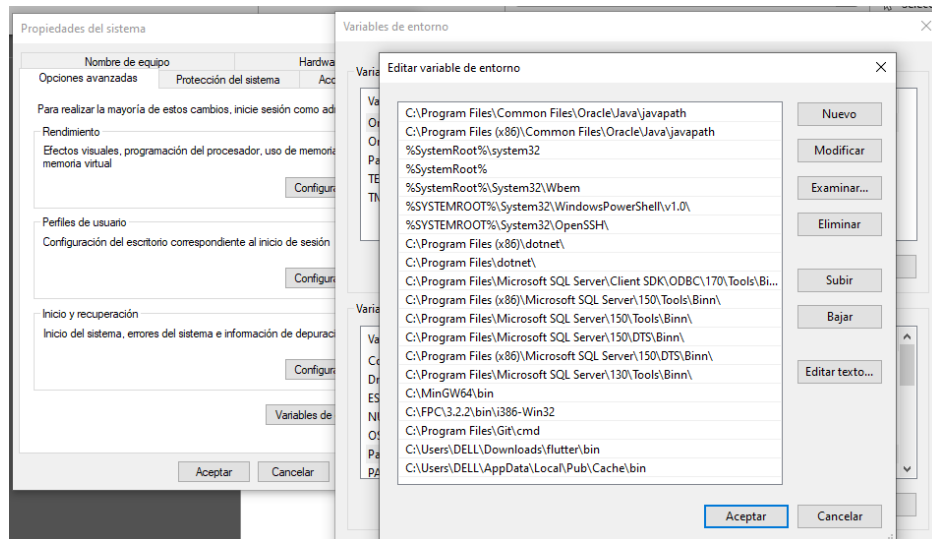
En el apartado “Descripción general” de Firebase se mostrarán, en la sección “Comienza por agregar Firebase a tu app”, las opciones para el tipo de aplicación a desarrollar



Después, aparecerán las opciones para instalar los archivos necesarios para establecer la conexión, la propia documentación de Firebase te guiara para instalar el “Firebase CLI” y el propio “SDK de flutter” de ser necesario

En el caso de “Firebase CLI” una vez ya ejecutado y realizadas las instrucciones de Firebase se creará el archivo “firebase_options.dart”

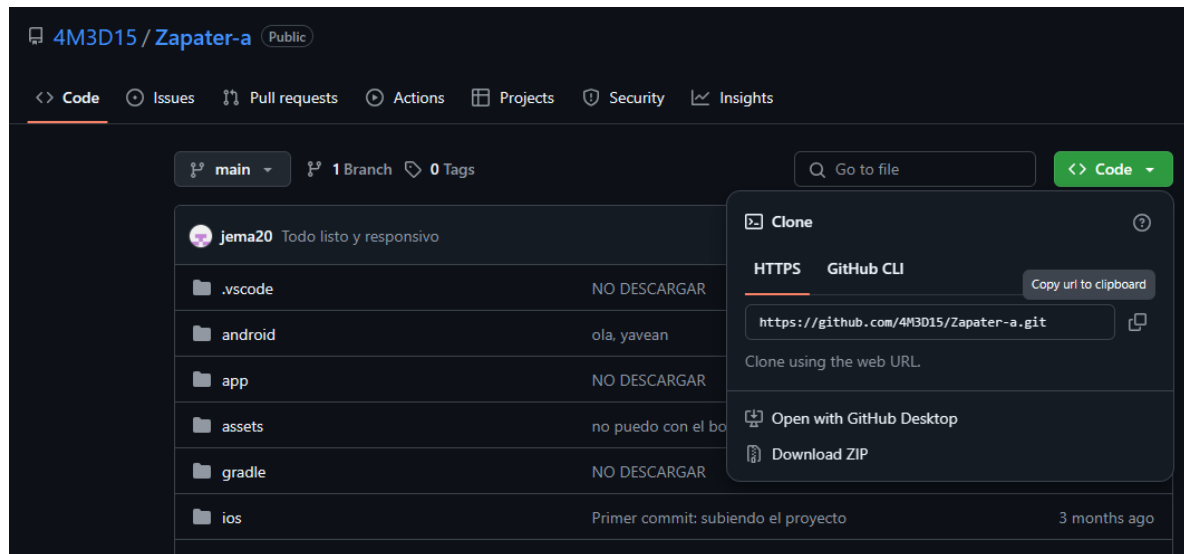
De no poder ejecutar “flutterfire” en la línea de comandos se tendrá que incluir la ubicación del archivo “flutterfire.bat” en las variables de entorno, este usualmente se descarga en “C:\Users\[TuUsuario]\AppData\Local\Pub\Cache\bin”



6.2 Instalación del proyecto

El proyecto se encuentra disponible en el repositorio de GitHub

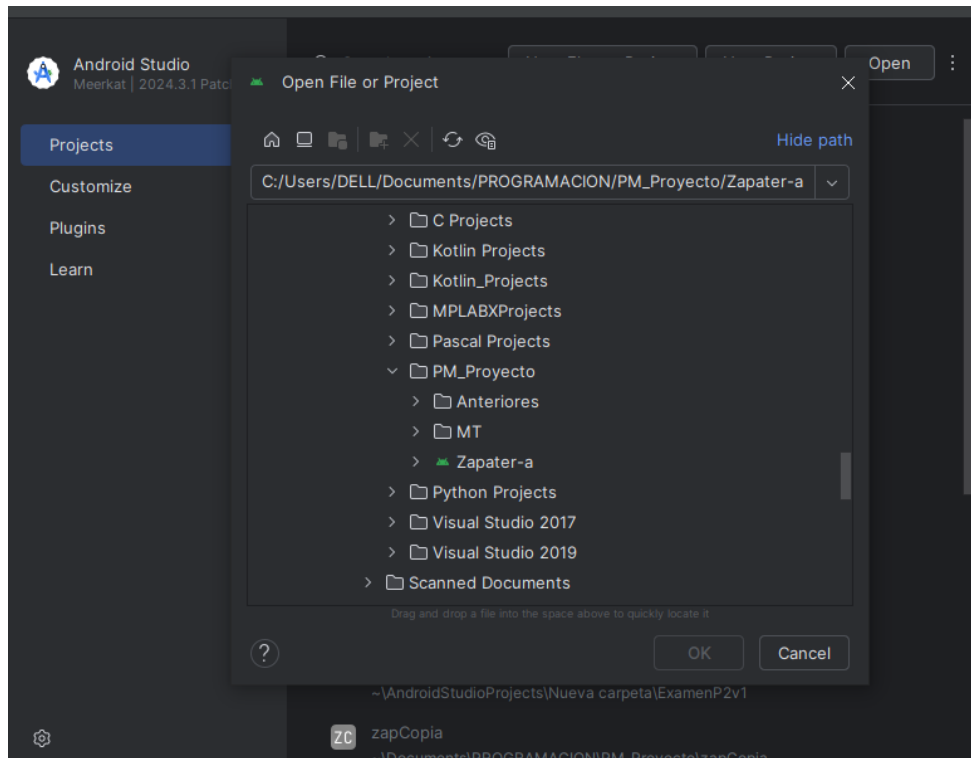
<https://github.com/4M3D15/Zapater-a> y para realizar su descarga es a través de la sección Code en la opción Code>HTTPS>Download ZIP



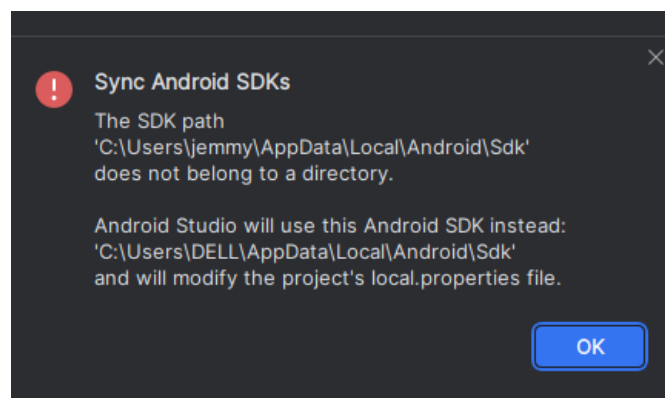
Una vez descargado el archivo se deberá descomprimir en la carpeta en la que se encuentran los proyectos de Android Studio, normalmente esta carpeta se encuentra en C:\Users\[TuUsuario].



Para abrir el programa dentro de Android Studio, en la pantalla de inicio, hay que seleccionar la opción “Open” para abrir el explorador de archivos y seleccionar el archivo descomprimido.



Ya encontrado y abierto el archivo, antes de mostrar el código se mostrará una pantalla de error debido a la dirección del SDK de Android, sin embargo al seleccionar la opción “OK” la dirección se ajustará a la del dispositivo actual.





Para que todas las librerías se descarguen y puedan ser utilizadas se debe de ejecutar el comando 'pub get' de flutter en la ubicación del archivo "pubspec.yaml", Android Studio tiene la opción "Pub get" para ejecutar el comando automáticamente.

A screenshot of the Android Studio interface. The top bar shows the 'main.dart' file is selected. Below the top bar, the 'Flutter commands' tab is active, displaying the 'pubspec.yaml' file. The file contains the following code:

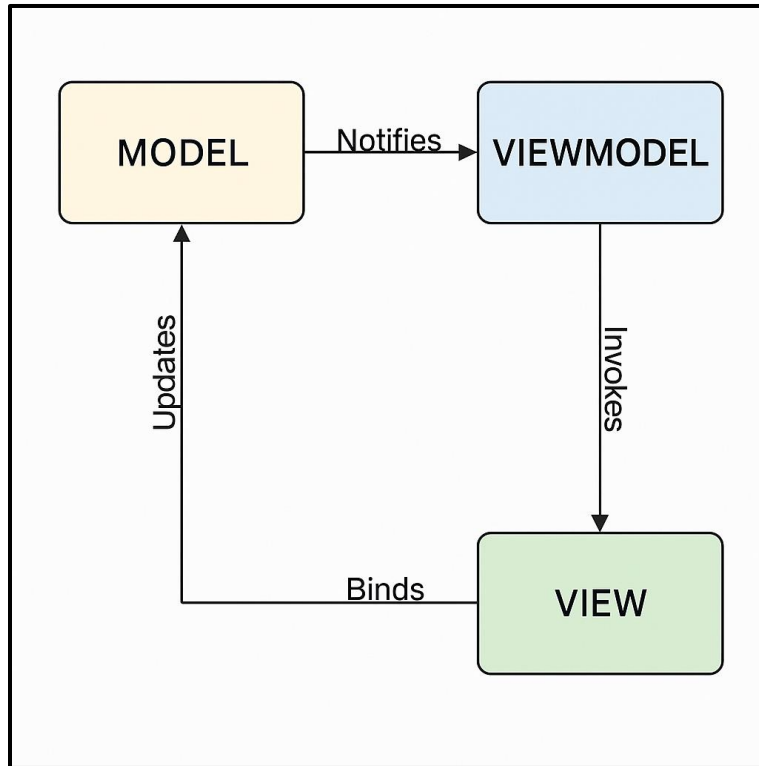
```
1 name: zapato
2 description: "A new Flutter project."
3
4 publish_to: 'none'
5
6 version: 1.0.0+1
7
8 environment:
9   sdk: ">=2.17.0 <4.0.0"
10
11 dependencies:
12   flutter:
13     sdk: flutter
14   cupertino_icons: ^1.0.8
15   provider: ^6.0.0
16   carousel_slider: ^4.2.1
17   flutter_typeahead: ^5.2.0
18   google_maps_flutter: ^2.5.0
19   firebase_core: ^3.0.0
20   firebase_auth: ^5.5.3
21   cloud_firestore: ^5.0.0
22   animations: ^2.0.7
```

The right side of the interface shows the 'Flutter doctor' output, indicating that all dependencies are satisfied.



7 Arquitectura general

La aplicación "Soleya" fue diseñada bajo una arquitectura modular basada en el patrón **MVVM (Model-View-ViewModel)**. Esta arquitectura permite una separación clara entre la lógica de negocio, la interfaz gráfica y el manejo del estado, facilitando el mantenimiento, escalabilidad y pruebas del sistema.



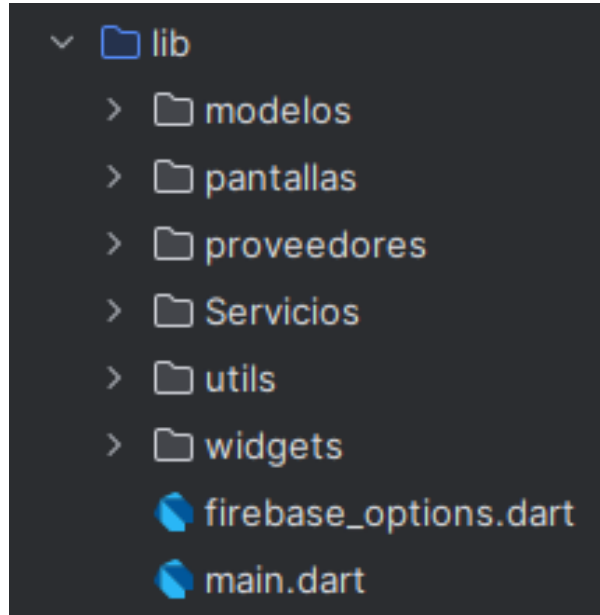
Componentes clave:

- **Modelo (Model):** Representa los datos y estructuras de la base de datos y API. Se encuentra en archivos como `producto_model.dart`, `cart_model.dart`, y `favoritos_model.dart`.
- **Vista (View):** Pantallas que el usuario ve y con las que interactúa. Se encuentra en `bienvenida.dart`, `busquedascreen.dart`, `cart_screen.dart`, entre otras.
- **ViewModel / Controlador de estado:** Gestionado por `ChangeNotifierProvider` y modelos de estado como `FavoritosModel`, que permiten reactividad y gestión del estado global.



8 Estructura del proyecto

La aplicación se organiza de forma lógica para separar responsabilidades. La estructura principal dentro de la carpeta lib/ es:

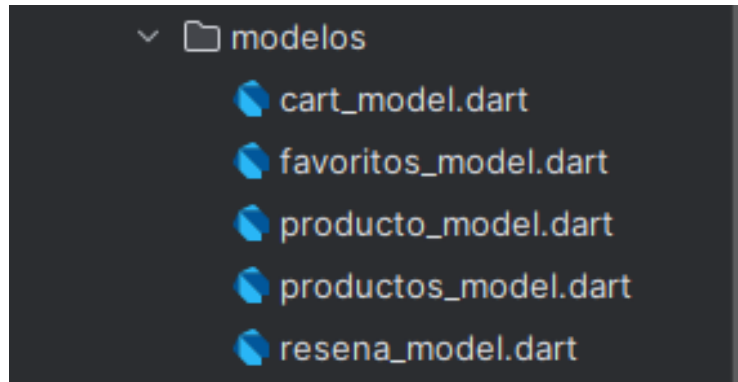


- **modelos/**: Define las clases de datos para productos, favoritos, carrito, etc.
- **pantallas/**: Incluye todas las pantallas que componen la navegación del usuario durante la ejecución de la aplicación.
- **proveedores/**: Contiene modelos que gestionan el estado, como el listado de favoritos o el carrito.
- **Servicios/**: Sección que puede manejar peticiones a bases de datos o servicios externos.
- **utils/**: Funciones utilitarias y constantes.
- **widgets/**: Contiene widgets como el ícono animado de favoritos, tarjetas de productos, botones, etc.



8.1 Modelos de datos

Se utilizan modelos Dart para representar entidades clave de la aplicación.



Dentro se encuentran todas las distintas clases/entidades usadas para guardar y manejar los datos en varias de las pantallas asociadas

- cart_model, producto_model y resena_model

Estos modelos son utilizados por los proveedores y las pantallas para construir la lógica de interacción con el usuario.

```
class CartItem {  
  final String id;  
  final String nombre;  
  final String imagen;  
  final double precio;  
  final String talla;  
  int cantidad;  
}
```

Modelo de datos para representar los productos que el carrito usara y manejara

```
class Producto {  
  final String id;  
  final String nombre;  
  final String categoria;  
  final String descripcion;  
  final double precio;  
  final String imagen;  
  final String sexo;  
  final String talla;  
  final String color;  
}
```

Modelo de datos para representar y manejar los productos obtenidos de la base de datos



```
class Resena {  
    final String usuario;  
    final String comentario;  
    final int rating;  
    final DateTime fecha;
```

Modelo de datos para representar el formato de las reseñas que se mostraran y guardaran en cada producto

- favoritos_model y productos_model

Como excepciones o complementos estos controladores permiten la gestión de sus respectivos tipos de datos, es decir, obtener, modificar, eliminar y actualizar la información referente al nombre del controlador.

```
Future<void> obtenerProductos() async {  
    isLoading = true;  
    error = null;  
    notifyListeners();  
  
    try {  
        _productos = await _firestoreService.obtenerProductos();  
    } catch (e) {  
        error = 'Error al cargar los productos, comuniquese con soporte tecnico\n(Error $e)';  
        debugPrint(error);  
        print('Error al obtener productos: $e');  
    } finally {  
        isLoading = false;  
        notifyListeners();  
    }  
}
```

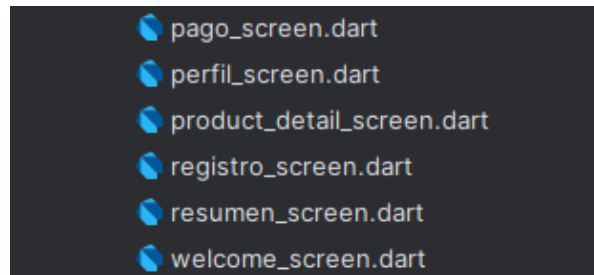
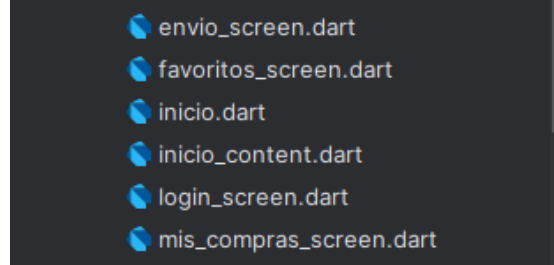
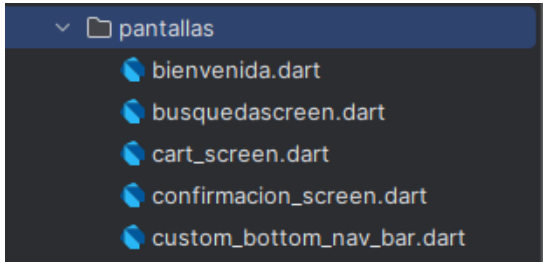
Consulta de datos desde Firestone, guardados en una lista de tipo "Producto"

```
// Agrega un producto a la lista de favoritos  
void agregarFavorito(Producto producto) {  
    if (!_favoritos.contains(producto)) {  
        operaciones_db().addFavorito(producto);  
        _favoritos.add(producto);  
        //Subir a la db en la nube  
        //Base local  
  
        notifyListeners(); // Notifica a los consumidores de cambios  
    }  
}  
  
// Remueve un producto de la lista de favoritos  
void removerFavorito(Producto producto) {  
    operaciones_db().deleteFavorito(producto);  
    _favoritos.remove(producto);  
    notifyListeners(); // Notifica a los consumidores de cambios  
}
```

Agregar y eliminar datos de tipo "Producto" que coincidan con los productos marcados



8.2 Pantallas



8.2.1 welcome_screen

Esta pantalla da la bienvenida al usuario con opciones claras para iniciar sesión o registrarse. El diseño está centrado en la experiencia de usuario, mostrando un botón para cada acción principal (ElevatedButton y OutlinedButton). Usa navegación con pantalla de carga para moverse entre pantallas (navigateWithLoading).

```
ElevatedButton(  
  onPressed: () =>  
    navigateWithLoading(context, const LoginScreen()),  
  style: ElevatedButton.styleFrom(  
    backgroundColor: Colors.black,  
    padding: const EdgeInsets.symmetric(  
      vertical: 16, horizontal: 64), // EdgeInsets.symmetric  
    shape: RoundedRectangleBorder(  
      borderRadius: BorderRadius.circular(12),  
    ), // RoundedRectangleBorder  
  ),  
  child: const Text(  
    'Iniciar Sesión',  
    style: TextStyle(fontSize: 18, color: Colors.white),  
  ), // Text  
), // ElevatedButton
```



```
OutlinedButton(  
  onPressed: () => navigateWithLoading(  
    context, const RegistroScreen(),  
  style: OutlinedButton.styleFrom(  
    side: const BorderSide(color: Colors.black),  
    padding: const EdgeInsets.symmetric(  
      vertical: 16, horizontal: 64), // EdgeInsets.symmetric  
    shape: RoundedRectangleBorder(  
      borderRadius: BorderRadius.circular(12),  
    ), // RoundedRectangleBorder  
  ),  
  child: const Text(  
    'Registrarse',  
    style:  
      TextStyle(fontSize: 18, color: Colors.black87),  
  ), // Text  
), // OutlinedButton
```

- login_screen

Permite a los usuarios existentes ingresar con su correo electrónico y contraseña. Se validan ambos campos y, en caso de éxito, se accede a la pantalla principal. Se utiliza 'TextFormField' con validadores y un ElevatedButton para enviar los datos. El manejo de errores está gestionado con mensajes visuales (ScaffoldMessenger).

```
final TextEditingController _emailCtrl = TextEditingController();  
final TextEditingController _passCtrl = TextEditingController();
```

```
child: TextFormField(  
  controller: _emailCtrl,  
  decoration: InputDecoration(  
    labelText: 'Correo electrónico',  
    filled: true,  
    fillColor: Colors.white,  
    border: OutlineInputBorder(borderRadius: BorderRadius.circular(12)),  
  ), // InputDecoration  
  keyboardType: TextInputType.emailAddress,  
  validator: (v) {  
    if (v == null || v.isEmpty) return 'Ingresa tu correo';  
    if (!RegExp(r'^[a-zA-Z0-9+@]+\.[a-zA-Z0-9+@]+\.[a-zA-Z0-9+@]+').hasMatch(v)) return 'Correo inválido';  
    return null;  
  },  
), // TextFormField
```



```
    TextButton(  
      onPressed: () => Navigator.pop(context),  
      child: const Text('Cancelar'),  
    ), // TextButton  
    ElevatedButton(  
      onPressed: () {  
        Navigator.pop(context);  
        _emailCtrl.text = usuario['correo'];  
        _passCtrl.text = usuario['password'];  
        _signIn();  
      },  
      child: const Text('Aceptar'),  
    ), // ElevatedButton
```

Para realizar el inicio de sesión se utiliza la función “_signIn” la cual establece conexión con la base de datos en Firebase

```
try {  
  await _auth.signInWithEmailAndPassword(  
    email: _emailCtrl.text.trim(),  
    password: _passCtrl.text,  
  );  
}
```

- registro_screen

Proporciona el formulario de registro. Los campos incluyen nombre, correo electrónico y contraseña. La validación básica verifica que los campos no estén vacíos. Al validarse los datos exitosamente, el usuario es registrado usando FirebaseAuth para la creación de la cuenta.

```
class _RegistroScreenState extends State<RegistroScreen> {  
  final _formKey = GlobalKey<FormState>();  
  final TextEditingController _nombreController = TextEditingController();  
  final TextEditingController _apellidoController = TextEditingController();  
  final TextEditingController _emailController = TextEditingController();  
  final TextEditingController _passwordController = TextEditingController();  
  final TextEditingController _confirmPasswordController = TextEditingController();
```



```
TextFormField(  
  controller: _nombreController,  
  inputFormatters: [  
    FilteringTextInputFormatter.allow(RegExp(r'[a-zA-ZáéíóúÁÉÍÓÚÑ\s]')),  
    TextInputFormatter.withFunction(  
      (oldValue, newValue) => newValue.copyWith(  
        text: newValue.text.toUpperCase(),  
        selection: newValue.selection,  
      )),  
  ], // TextInputFormatter.withFunction  
),  
decoration: InputDecoration(  
  labelText: "Nombre",  
  border: OutlineInputBorder(borderRadius: BorderRadius.circular(10)),  
  filled: true,  
  fillColor: Colors.white,  
  prefixIcon: const Icon(Icons.person),  
), // InputDecoration  
validator: (v) => v == null || v.isEmpty ? "Por favor ingresa tu nombre" : null,  
), // TextFormField
```

```
child: ElevatedButton(  
  onPressed: _isLoading ? null : _registerUser,  
  style: ElevatedButton.styleFrom(  
    backgroundColor: Colors.black,  
    padding: const EdgeInsets.symmetric(vertical: 15),  
    shape: RoundedRectangleBorder(borderRadius: BorderRadius.circular(10)),  
  ),  
  child: _isLoading  
    ? const SizedBox(  
      height: 24,  
      width: 24,  
      child: CircularProgressIndicator(color: Colors.white, strokeWidth: 2),  
    ) // SizedBox  
    : const Text("Registrarse", style: TextStyle(color: Colors.white, fontSize: 18)),  
), // ElevatedButton
```

```
try {  
  final userCredential = await _auth.createUserWithEmailAndPassword(  
    email: _emailController.text.trim(),  
    password: _passwordController.text,  
  );  
  final user = userCredential.user;  
  if (user != null) {  
    await _firestore.collection('usuarios').doc(user.uid).set({  
      'nombre': _nombreController.text.trim(),  
      'apellido': _apellidoController.text.trim(),  
      'correo': _emailController.text.trim(),  
      'avatar': "",  
      'fechaRegistro': FieldValue.serverTimestamp(),  
    });  
  }  
}
```




En ambos casos, al realizar el registro o inicio de sesión se mostrará la opción de guardar el usuario en la base de datos local para hacer más rápido el inicio de sesión después de cerrarla anteriormente



```
Future<bool?> _preguntarGuardarLocal() async {
  return showDialog<bool>({
    context: context,
    barrierDismissible: false,
    builder: (context) => AlertDialog(
      title: const Text('Guardar inicio de sesión'),
      content: const Text('¿Quieres guardar tus datos para iniciar sesión localmente?'),
      actions: [
        TextButton(
          onPressed: () => Navigator.of(context).pop(false),
          child: const Text('No'),
        ), // TextButton
        TextButton(
          onPressed: () => Navigator.of(context).pop(true),
          child: const Text('Si'),
        ), // TextButton
      ],
    ), // AlertDialog
  );
}
```

```
try {
  final user_id = _auth.currentUser?.uid;
  final doc = await _firestore.collection('usuarios').doc(user_id).get();
  final data = doc.data();

  if (guardarLocal == true && data != null) {
    await operaciones_db().setUsuarioLocal(data, _passwordController.text);
  }

  context.read<CartProvider>().obtenerCarrito();
  context.read<FavoritosModel>().obtenerFavoritos();
  context.read<ProductosModel>().obtenerProductos();
} catch (e) {
  print('Error al guardar el usuario en local: $e');
}
```



8.2.2 custom_bottom_nav_bar

Este archivo define una barra de navegación personalizada que permite cambiar entre las secciones principales: Inicio, Buscar, Favoritos, Bolsa y Perfil. Utiliza BottomNavigationBar con BottomNavigationBarItem y un controlador de estado para identificar la pestaña activa.

```
final List<_NavBarItemData> _items = const [
  _NavBarItemData(icon: Icons.home, label: 'Inicio'),
  _NavBarItemData(icon: Icons.search, label: 'Buscar'),
  _NavBarItemData(icon: Icons.favorite_border, label: 'Favoritos'),
  _NavBarItemData(icon: Icons.shopping_bag_outlined, label: 'Bolsa'),
  _NavBarItemData(icon: Icons.person_outline, label: 'Perfil'),
];
```

modelo _NavBarItemData para cada sección

```
class _NavBarItemData {
  final IconData icon;
  final String label;

  const _NavBarItemData({
    required this.icon,
    required this.label,
  });
}
```



8.2.3 inicio

Es el contenedor principal de la vista de inicio. Llama al widget “inicio_content” y establece la estructura superior. Maneja también el índice de navegación activa con la barra inferior personalizada.

Construcción de la pantalla en base al índice de la pantalla actual

```
Expanded(  
  child: _buildPantalla(_currentIndex),  
), // Expanded
```

```
Widget _buildPantalla(int index) {  
  switch (index) {  
    case 0:  
      return InicioContent(scrollController: _scrollController);  
    case 1:  
      return const BusquedaScreen();  
    case 2:  
      return const FavoritosScreen();  
    case 3:  
      return const CartScreen();  
    case 4:  
      final user = FirebaseAuth.instance.currentUser;  
      return user != null  
        ? const ProfileScreen()  
        : const WelcomeScreen();  
    default:  
      return const SizedBox.shrink();  
  }  
}
```



- inicio_content

Contiene el contenido real del inicio: una cuadrícula de productos (PageView.builder) que se cargan desde la base de datos en Firebase. Cada producto se muestra usando el widget “producto_tile”, y al tocar uno, se navega a “product_detail_screen” (Navigator.pushNamed).

Imagen del producto

```
Image.network(  
  p.imagen,  
  fit: BoxFit.fitWidth,  
  errorBuilder: (context, error, stackTrace) {  
    return const Icon(Icons.image_not_supported);  
  },  
, // Image.network
```

Icono de favorito

```
Positioned(  
  top: 8,  
  right: 8,  
  child: Consumer<FavoritosModel>(  
    builder: (_, fav, __) {  
      final isFav = fav.esFavorito(p);  
      return AnimatedFavoriteIcon(  
        esFavorito: isFav,  
        onTap: () => isFav  
          ? fav.removerFavorito(p)  
          : fav.agregarFavorito(p),  
      ); // AnimatedFavoriteIcon  
    },  
  ), // Consumer  
, // Positioned
```



Función de filtrado y botones de filtrado



```
// Método para filtrar productos según el sexo seleccionado, ignorando mayúsculas
Padding(
  padding: const EdgeInsets.symmetric(horizontal: 16),
  child: Center(
    child: Wrap(
      alignment: WrapAlignment.center,
      spacing: 12,
      runSpacing: 8,
      children: ['Todos', 'Hombre', 'Mujer', 'Niño'].map((sexo) {
        final activo = _sexoSeleccionado.toLowerCase() == sexo.toLowerCase();
        return ElevatedButton(
          onPressed: () {
            setState(() => _sexoSeleccionado = sexo);
          },
          style: ElevatedButton.styleFrom(
            backgroundColor: activo ? Colors.black87 : Colors.grey.shade300,
            foregroundColor: activo ? Colors.white : Colors.black,
            shape: RoundedRectangleBorder(borderRadius: BorderRadius.circular(20)),
            padding: const EdgeInsets.symmetric(horizontal: 16, vertical: 8),
          ),
          child: Text(capitalize(sexo)),
        ); // ElevatedButton
      }).toList(),
    ), // Wrap
  ), // Center
), // Padding
```

```
// Método para filtrar productos según el sexo seleccionado, ignorando mayúsculas
List filtrarProductos(List productos) {
  if (_sexoSeleccionado.toLowerCase() == 'todos') {
    return productos;
  } else {
    final sexoFiltro = _sexoSeleccionado.toLowerCase();
    final filtrados = productos.where((p) {
      // Asume que p.sexo es String
      final sexoProducto = (p.sexo ?? '').toString().toLowerCase();
      print('Producto: ${p.nombre}, sexo: $sexoProducto, filtro: $sexoFiltro');
      return sexoProducto == sexoFiltro;
    }).toList();

    print('Productos filtrados: ${filtrados.length}');
    return filtrados;
  }
}
```



- product_detail_screen

Muestra los detalles del producto seleccionado, incluyendo imagen, precio, descripción y un botón para agregar al carrito. Utiliza parámetros para recibir el producto desde la pantalla anterior. Implementa también animaciones y una funcionalidad de favoritos.

Cargar los datos del producto desde Firebase.

```
Future<void> _cargarProducto() async {  
  final doc = await FirebaseFirestore.instance  
    .collection('productos')  
    .doc(widget.productId)  
    .get();  
  if (doc.exists) {  
    setState(() {  
      producto = Producto.fromFirestore(doc);  
      final tallaMap = doc.data()?['Talla'] as Map<String, dynamic>? ?? {};  
      tallasStock.clear();  
      tallaMap.forEach((talla, stock) {  
        tallasStock[talla] = stock as int;  
      });  
      if (tallasStock.isNotEmpty) {  
        tallaSeleccionada = tallasStock.keys.first;  
      }  
    });  
  
    await _cargarResenas();  
    setState(() {  
      _isLoading = false;  
    });  
  }  
}
```

Cargar los datos de las reseñas acerca del producto desde Firebase

```
Future<void> _cargarResenas() async {  
  final snapshot = await FirebaseFirestore.instance  
    .collection('productos')  
    .doc(widget.productId)  
    .collection('reseñas')  
    .orderBy('fecha', descending: true)  
    .get();  
  
  final resenas = snapshot.docs.map((doc) => Resena.fromMap(doc.data())).toList();  
  setState(() {  
    _resenas.clear();  
    _resenas.addAll(resenas);  
  });  
}
```



Selector de tallas



```
void _mostrarSelectorTallas() {  
  showModalBottomSheet(  
    context: context,  
    builder: (_) => Padding(  
      padding: const EdgeInsets.all(16),  
      child: Wrap(  
        spacing: 10,  
        children: tallasStock.entries.map((entry) {  
          final talla = entry.key;  
          final stock = entry.value;  
          return ChoiceChip(  
            label: Text('$talla ($stock disponibles)'),  
            selected: talla == tallaSeleccionada,  
            onPressed: (_) {  
              setState(() {  
                tallaSeleccionada = talla;  
                cantidad = min(1, stock);  
              });  
              Navigator.pop(context);  
            },  
          ); // ChoiceChip  
        }).toList(),  
      ), // Wrap  
    ), // Padding  
  );  
}
```

Subir reseña del producto a Firebase

```
void _agregarResena() async {  
  if (_comentarioController.text.trim().isEmpty || _calificacionSeleccionada == 0) return;  
  
  final nuevaResena = Resena(  
    usuario: 'Anónimo',  
    comentario: _comentarioController.text.trim(),  
    rating: _calificacionSeleccionada,  
    fecha: DateTime.now(),  
  ); // Resena  
  
  setState(() {  
    _resenas.insert(0, nuevaResena);  
    _comentarioController.clear();  
    _calificacionSeleccionada = 0;  
  });  
  
  await FirebaseFirestore.instance  
    .collection('productos')  
    .doc(widget.productId)  
    .collection('reseñas')  
    .add(nuevaResena.toMap());  
  
  late OverlayEntry entry;  
  entry = OverlayEntry(builder: (_) {  
    return ConfettiOverlay(onComplete: () {  
      entry.remove();  
    }); // ConfettiOverlay  
  }); // OverlayEntry  
  Overlay.of(context).insert(entry);  
}
```



8.2.4 busquedascreen

Permite buscar productos en tiempo real. Utiliza un TextField que detecta la entrada del usuario y filtra la lista de productos. Los resultados se muestran en un ListView. La lógica se basa en coincidencias de texto con los nombres de productos disponibles.

Obtención de los productos de Firebase y filtrado de los productos dependiendo de la búsqueda

```
Future<void> _loadProductos() async {  
  final cargados = await _firestoreService.obtenerProductos();  
  setState(() {  
    productos = cargados;  
    filteredProductos = cargados;  
  });  
}  
  
void _searchProducts(String query) {  
  final lower = query.toLowerCase();  
  setState(() {  
    filteredProductos = productos.where((p) {  
      return p.nombre.toLowerCase().contains(lower) ||  
        p.descripcion.toLowerCase().contains(lower) ||  
        p.categoria.toLowerCase().contains(lower) ||  
        p.color.toLowerCase().contains(lower) ||  
        p.sexo.toLowerCase().contains(lower);  
    }).toList();  
  
    sugerencias = productos  
      .where((p) => p.nombre.toLowerCase().startsWith(lower))  
      .map((p) => p.nombre)  
      .toList();  
  });  
}
```




8.2.5 favoritos_screen

Muestra una lista de productos marcados como favoritos. Los datos se gestionan con un FavoritosModel que usa ChangeNotifier para actualizar dinámicamente la interfaz cuando se añaden o eliminan elementos. Se reutilizan componentes de producto como producto_tile.dart.

Obtener los productos para el carrusel de productos sugeridos

```
Firestore.instance.collection('productos').get().then((snap) {  
  _sugerencias = snap.docs.map((d) => Producto.fromFirestore(d)).toList();  
  setState(() {});  
  _timer = Timer.periodic(const Duration(seconds: 5), (_) {  
    if (_pageController.hasClients && (_sugerencias?.isNotEmpty ?? false)) {  
      final next = (_currentCarousel + 1) % _sugerencias!.length;  
      _pageController.animateToPage(  
        next,  
        duration: const Duration(milliseconds: 400),  
        curve: Curves.easeInOut,  
      );  
    }  
  }); // Timer.periodic  
}).catchError((e) => debugPrint('Error cargando sugerencias: $e'));
```

Opción para eliminar todos los favoritos

```
onPressed: () async {  
  if (!_sinInternet) {  
    ScaffoldMessenger.of(context).showSnackBar(  
      const SnackBar(content: Text("Solo vista, sin acciones.")),  
    );  
    return;  
  }  
  final favModel = context.read<FavoritosModel>();  
  final overlay = Overlay.of(context);  
  for (final prod in List<Producto>.from(favModel.favoritos)) {  
    final key = _tileKeys[prod];  
    final box = key?.currentContext?.findRenderObject() as RenderBox?;  
    if (box != null) {  
      final pos = box.localToGlobal(box.size.center(Offset.zero));  
      late OverlayEntry exp;  
      exp = OverlayEntry(  
        builder: (_) => ParticleExplosion(  
          position: pos,  
          onComplete: () => exp.remove(),  
        ), // ParticleExplosion  
      ); // OverlayEntry  
      overlay.insert(exp);  
      await _playExplosionSound();  
      await Future.delayed(const Duration(milliseconds: 100));  
    }  
    favModel.removeFavorito(prod);  
    await Future.delayed(const Duration(milliseconds: 150));  
  }  
}
```



```
// Lista favoritos o mensaje vacio
Container(
  constraints: BoxConstraints(
    maxHeight: screenHeight * 0.45,
  ), // BoxConstraints
  child: _items.isEmpty
    ? const Center(
      child: Padding(
        padding: EdgeInsets.all(20.0),
        child: Text('No has seleccionado ningún favorito', style: TextStyle(fontSize: 18)),
      ), // Padding
    ) // Center
    : Scrollbar(
      thumbVisibility: true,
      child: ListView.builder(
        physics: const BouncingScrollPhysics(),
        shrinkWrap: true,
        itemCount: _items.length,
        itemBuilder: (context, index) {
          return SlideFadeInFromBottom(
            delay: Duration(milliseconds: 100 * (index + 1)),
            child: _buildFavoriteTile(_items[index]),
          ); // SlideFadeInFromBottom
        },
      ), // ListView.builder
    ), // Scrollbar
), // Container
```

8.2.6 cart_screen

Presenta los productos añadidos al carrito. Muestra cantidad, subtotal y botones para modificar el contenido o continuar al envío. Se usa el modelo CartModel para calcular totales y cantidades dinámicamente.



Vista del listado de los productos en el carrito/bolsa



Imagen

```
ClipRect(  
  borderRadius: BorderRadius.circular(10),  
  child: Image.network(  
    item.imagen,  
    width: 80,  
    height: 80,  
    fit: BoxFit.fitWidth,  
    errorBuilder: (context, error, stackTrace) {  
      return const Icon(Icons.image_not_supported);  
    },  
  ), // Image.network  
) // ClipRect
```

Disminuir cantidad

```
IconButton(  
  icon: const Icon(Icons.remove),  
  onPressed: () {  
    if (!_sinInternet) {  
      ScaffoldMessenger.of(context).showSnackBar(  
        const SnackBar(content: Text("Solo vista, sin acciones.")),  
      );  
      return;  
    }  
    if (item.cantidad > 1) {  
      cant.updateQuantity(item, item.cantidad - 1);  
    }  
  },  
) // IconButton
```



Aumentar cantidad



```
const TextStyle(fontWeight: FontWeight.bold), // Text
IconButton(
  icon: const Icon(Icons.add),
  onPressed: () async {
    if (!_sinInternet) {
      ScaffoldMessenger.of(context).showSnackBar(
        const SnackBar(content: Text("Solo vista, sin acciones.")),
      );
      return;
    }
    final nuevoValor = item.cantidad + 1;
    final stockDisponible =
      await cart.getStockDisponible(item.id, item.talla);

    if (nuevoValor <= stockDisponible) {
      cart.updateQuantity(item, nuevoValor);
    } else {
      ScaffoldMessenger.of(context).showSnackBar(
        SnackBar(
          content: Text(
            "Solo hay $stockDisponible unidades disponibles para la talla ${item.talla}.",
          ), // Text
          backgroundColor: Colors.orange,
        ), // SnackBar
      );
    }
  },
), // IconButton
```

Total y subtotal del carrito/bolso

```
const Divider(),
Row(
  mainAxisAlignment:
    MainAxisAlignment.spaceBetween,
  children: [
    const Text("Subtotal"),
    Text(
      "\${cart.totalPrice.toStringAsFixed(2)}",
    ),
  ],
), // Row
const SizedBox(height: 8),
Row(
  mainAxisAlignment:
    MainAxisAlignment.spaceBetween,
  children: [
    const Text("Total",
      style: TextStyle(
        fontWeight:
          FontWeight.bold,
        fontSize: 16)), // TextStyle, Text
    Text(
      "\${cart.totalPrice.toStringAsFixed(2)}",
      style: const TextStyle(
        fontWeight: FontWeight.bold,
        fontSize: 16), // TextStyle
    ), // Text
  ],
), // Row
```



Realizar compra y continuar con la siguiente pantalla



```
    } else {  
      final productos =  
        List<CartItem>.from(cart.items);  
      final total = cart.totalPrice;  
      cart.clearCart();  
      Navigator.push(  
        context,  
        MaterialPageRoute(  
          builder: (context) => EnvioScreen(  
            productos: productos,  
            total: total,  
          ), // EnvioScreen  
        ), // MaterialPageRoute  
      );  
    }  
  }  
}
```

- **envio_screen**

Formulario para ingresar datos de envío como dirección, ciudad y teléfono. Se valida la entrada antes de permitir continuar. Utiliza TextEditingController para capturar información

Validación de los campos de texto

```
Widget _campoTexto(TextEditingController c, String label, int index) {  
  final esDireccion = label != 'Código Postal';  
  return SlideFadeIn(  
    index: index,  
    child: Padding(  
      padding: const EdgeInsets.only(bottom: 12),  
      child: TextFormField(  
        controller: c,  
        textCapitalization:  
          esDireccion ? TextCapitalization.characters : TextCapitalization.none,  
        inputFormatters: label == 'Código Postal'  
          ? [FilteringTextInputFormatter.digitsOnly]  
          : [UpperCaseTextFormatter()],  
        keyboardType: label == 'Código Postal' ? TextInputType.number : TextInputType.text,  
        maxLength: label == 'Código Postal' ? 5 : null,  
        buildCounter: (_, {int currentLength = 0, bool isFocused = false, int? maxLength}) =>  
          null,  
        decoration: InputDecoration(  
          labelText: label,  
          border: const OutlineInputBorder(),  
        ), // InputDecoration  
        validator: (v) {  
          if (v == null || v.trim().isEmpty) return 'Este campo es obligatorio';  
          if (label == 'Código Postal' && v.trim().length != 5) {  
            return 'El código postal debe tener 5 dígitos';  
          }  
          return null;  
        },  
      ), // TextFormField  
    ), // Padding  
  ); // SlideFadeIn  
}
```



```
Future.delayed(const Duration(seconds: 1), () {  
  Navigator.pushNamed(  
    context,  
    '/pago',  
    arguments: {  
      'direccion': direccion,  
      'productos': widget.productos,  
      'total': widget.total,  
    },  
  );  
}); // Future.delayed
```

- **pago_sceen**

Simula una pantalla de pago, donde el usuario selecciona método de pago (por ahora visual) y confirma el pedido. Se prepara la orden para el resumen.

Opciones sobre el método de pago a elegir

```
Expanded(  
  child: RadioListTile<String>(  
    title: const Text('Tarjeta de débito'),  
    value: 'debito',  
    groupValue: _tipoTarjeta,  
    onChanged: (v) => setState(() {  
      _tipoTarjeta = v!;  
      _isCreditoSelected = false;  
    }),  
    contentPadding: EdgeInsets.symmetric(horizontal: isSmallWidth ? 8 : 16),  
  ), // RadioListTile  
, // Expanded  
Expanded(  
  child: RadioListTile<String>(  
    title: const Text('Tarjeta de crédito'),  
    value: 'credito',  
    groupValue: _tipoTarjeta,  
    onChanged: (v) => setState(() {  
      _tipoTarjeta = v!;  
      _isCreditoSelected = true;  
    }),  
    contentPadding: EdgeInsets.symmetric(horizontal: isSmallWidth ? 8 : 16),  
  ), // RadioListTile  
, // Expanded
```



Campo de texto con validación del texto

```
TextFormField(  
  controller: _nombreController,  
  decoration: const InputDecoration(  
    labelText: 'Nombre del Titular',  
    helperText: 'Se convierte automáticamente en mayúsculas',  
    border: OutlineInputBorder(),  
  ), // InputDecoration  
  validator: (value) => value == null || value.isEmpty ? 'Este campo es obligatorio' : null,  
  inputFormatters: [  
    FilteringTextInputFormatter.allow(RegExp(r'[a-zA-Z\s]')),  
    UpperCaseTextFormatter(),  
  ],  
), // TextFormField
```



Boton de pago y finalización de compra

```
ElevatedButton(  
  onPressed: () {  
    if (_formKey.currentState?.validate() ?? false) {  
      _confirmarPago(direccion, productos, total);  
    }  
  },  
  child: const Text('Pagar ahora'),  
  style: ElevatedButton.styleFrom(  
    backgroundColor: Colors.black,  
    foregroundColor: Colors.white,  
    minimumSize: Size(double.infinity, buttonHeight),  
    shape: RoundedRectangleBorder(  
      borderRadius: BorderRadius.circular(30),  
    ), // RoundedRectangleBorder  
  ),  
), // ElevatedButton
```

- **resumen_screen**

Resumen final del pedido, mostrando productos, dirección y forma de pago con varios Text. Se muestra un mensaje de éxito o error. Puede incluir una animación de confeti si la compra fue exitosa (usando “confetti_overlay”).



Boton para volver al inicio



```
child: ElevatedButton(  
  onPressed: () {  
    Navigator.of(context).pushAndRemoveUntil(  
      MaterialPageRoute(builder: (context) => InicioScreen()),  
      (route) => false,  
    );  
  },  
  style: ElevatedButton.styleFrom(  
    backgroundColor: Colors.black,  
    foregroundColor: Colors.white,  
    shape: RoundedRectangleBorder(  
      borderRadius: BorderRadius.circular(30),  
    ), // RoundedRectangleBorder  
  ),  
  child: const Text('Volver al inicio'),  
), // ElevatedButton
```

8.2.7 perfil_screen

Contiene la información del perfil del usuario y accesos a editar perfil, cerrar sesión o ver compras. Muestra nombre y correo. Utiliza FirebaseAuth para mostrar la información de usuario autenticado.

Cargar la información de Firebase o de SQLite

```
Future<void> _loadUserData() async {  
  try {  
    _user = _auth.currentUser!;  
    if (_sinInternet) {  
      final usuario = await operaciones_db().getUsuario();  
      _nameController.text = capitalize(usuario['nombre'] ?? '');  
      _lastNameController.text = capitalize(usuario['apellido'] ?? '');  
      _avatarFile = usuario['avatar'] != null && usuario['avatar'] != "" ? File(usuario['avatar']) : null;  
    } else {  
      final doc = await _firestore.collection('usuarios').doc(_user.uid).get();  
      final data = doc.data();  
      if (data != null) {  
        _nameController.text = capitalize(data['nombre'] ?? '');  
        _lastNameController.text = capitalize(data['apellido'] ?? '');  
        _avatarFile = data['avatar'] != null && data['avatar'] != "" ? File(data['avatar']) : null;  
      }  
    }  
  } catch (e) {  
    print('Error al cargar datos del usuario: $e');  
  }  
  setState(() {});  
}
```




Elegir foto de la galería y comprimirla para almacenarla dentro de la aplicación

```
Future<void> _pickAvatar() async {  
  final picker = ImagePicker();  
  final picked = await picker.pickImage(source: ImageSource.gallery);  
  if (picked == null) return;  
  
  final compressedBytes = await FlutterImageCompress.compressWithFile(  
    picked.path,  
    quality: 60,  
  );  
  
  if (compressedBytes == null) return;  
  
  final directory = await getApplicationDocumentsDirectory();  
  final fileName = 'avatar_${_user.uid}.jpg';  
  final filePath = p.join(directory.path, fileName);  
  
  final avatarFile = File(filePath);  
  await avatarFile.writeAsBytes(compressedBytes);  
  
  setState(() {  
    _avatarFile = avatarFile;  
    _avatarLocalPath = filePath;  
  });  
}
```

Actualizar la información del usuario en Firebase y SQLite

```
await operaciones_db().actualizarUsuario(updateData);  
  
await _firestore.collection('usuarios').doc(_user.uid).update(updateData);
```

Cerra sesión de Firebase

```
Future<void> _signOut() async {  
  await _auth.signOut();  
  Navigator.pushReplacement(  
    context,  
    MaterialPageRoute(builder: (_) => const WelcomeScreen()),  
  );  
}
```



- **mis_compras_screen**

Muestra las compras realizadas por el usuario en forma de lista, con detalles de fecha, productos y estado. La fuente de datos puede ser una lista simulada o datos reales almacenados en Firestore o SQLite.

Obtener los pedidos realizados con la cuenta en base al correo y ordenado por la fecha del pedido

```
stream: FirebaseFirestore.instance
    .collection('pedidos')
    .where('correo', isEqualTo: correoUsuario)
    .orderBy('fecha', descending: true)
    .snapshots(),
```

8.3 Proveedores

8.3.1 cart_provider

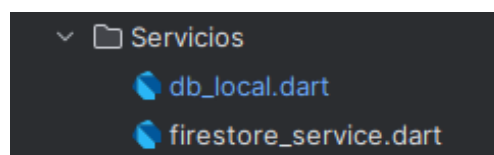
Contiene todas las instrucciones para mostrar, eliminar, modificar y añadir productos del tipo “CartItem” usando las bases de datos de Firebase y SQLite

Obtener y guardar los productos en el carrito/bolso

```
6      List<CartItem> get items => _items;
7
8
9      double get totalPrice => _items.fold(
10         0.0, (total, item) => total + item.precio * item.cantidad);
11
12      Future<void> obtenerCarrito() async{
13         _items = await operaciones_db().getCarrito();
14         notifyListeners();
15      }
```

Las instrucciones del CRUD se encuentra en “db_local”

8.4 Servicios





8.4.1 db_local

Todas las instrucciones que involucren el uso y creación de la base de datos local utilizando SQLite

Creación de la base de datos y las tablas dentro de esta

```
Future<Database> initDB() async {  
  final path = join(await getDatabasesPath(), 'Local.db');  
  //print('data: $data');  
  return await openDatabase(path, version: 1, onCreate: (db, version) async {  
    await db.execute('''  
      CREATE TABLE carrito (  
        id_carrito INTEGER PRIMARY KEY AUTOINCREMENT,  
        id_user TEXT,  
        id_producto TEXT,  
        nombre TEXT,  
        imagen TEXT,  
        precio REAL,  
        talla TEXT,  
        cantidad INTEGER  
      )  
    ''');  
  });  
}
```

instrucción para las consultas

Instrucción de consulta con el nombre de la tabla

```
final query = await local.query(  
  'usuario',  
  columns: ['nombre', 'apellido', 'avatar'], // e
```

Nombre de las columnas a consultar

```
columns: ['nombre', 'apellido', 'avatar'], // e
```

Condición y datos para realizar la condición marcada como '?'

```
where: 'user_uid = ?', whereArgs: [user_id]);
```

Instrucción para eliminar [Nombre de la tabla, condición y valor para la condición]

(Para eliminar por completo no se coloca una condición)

```
await local.delete('usuario', where: 'user_uid = ?', whereArgs: [id_user]);
```



Instrucción para insertar [Nombre de la tabla, Columna: Dato a ingresar y (de poder haber un conflicto con los datos) ConflictAlgorithm.replace]

```
await local.insert('favoritos', {
  'id_producto': producto.id,
  'id_user': user_id,
  'nombre': producto.nombre,
  'categoria': producto.categoria,
  'descripcion': producto.descripcion,
  'precio': producto.precio,
  'imagen': producto.imagen,
  'sexo': producto.sexo,
  'talla': producto.talla,
  'color': producto.color
}, conflictAlgorithm: ConflictAlgorithm.replace);
```

Instrucción para actualizar [Nombre de la tabla, Nombre de la columna: Nuevo dato, Condición y Valor de la condición]

```
await local.update('carrito', {
  'cantidad': cantidad
}, where: 'id_producto = ? and talla = ? and id_user = ?', whereArgs: [producto.id, producto.talla, user_id]);
```

8.4.2 firestore_service

Consulta para obtener los productos de la Firestore y almacenarlos en una lista de tipo “Producto”

```
class FirestoreService {
  final CollectionReference _productosRef =
    FirebaseFirestore.instance.collection('productos');

  Future<List<Producto>> obtenerProductos() async {
    final snapshot = await _productosRef.get();
    return snapshot.docs.map((doc) => Producto.fromFirestore(doc as DocumentSnapshot<Map<String, dynamic>>)).toList();
  }
}
```



8.5 Utilidades

8.5.1 Navigation

Permite controlar la navegación desde lugares donde no se tiene acceso directo al `BuildContext`, como en clases `Provider`, servicios, o controladores.

Además muestra una pantalla de carga con un círculo de progreso

```
Future<void> navigateWithLoading(BuildContext context, Widget page, {bool replace = false}) async {
  showDialog(
    context: context,
    barrierDismissible: false,
    builder: (context) => const Center(
      child: CircularProgressIndicator(),
    ), // Center
  );

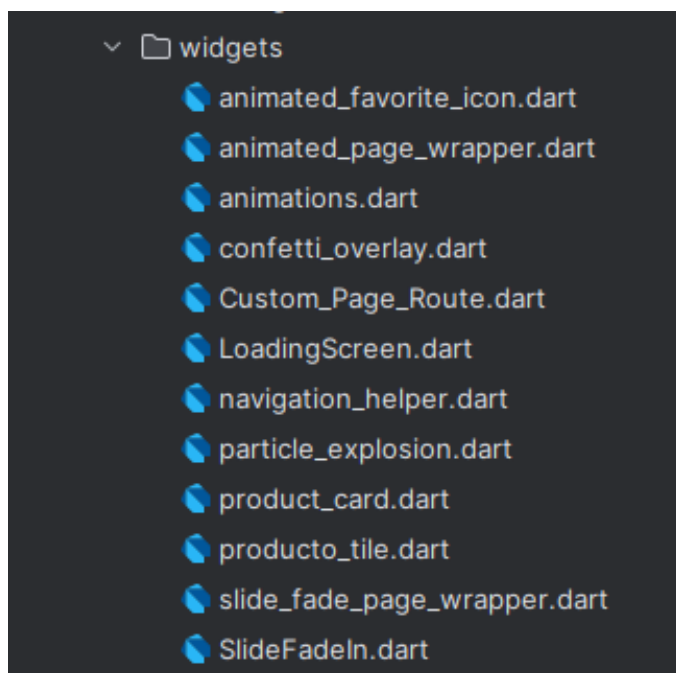
  await Future.delayed(const Duration(milliseconds: 800)); // Simula carga o espera

  if (replace) {
    Navigator.of(context).pop(); // Cierra el diálogo
    Navigator.of(context).pushReplacement(
      MaterialPageRoute(builder: (_) => page),
    );
  } else {
    Navigator.of(context).pop(); // Cierra el diálogo
    Navigator.of(context).push(
      MaterialPageRoute(builder: (_) => page),
    );
  }
}
```



8.6 Widgets

La carpeta widgets contiene componentes visuales **reutilizables y personalizados** diseñados para modularizar y estandarizar partes clave de la interfaz de usuario en la aplicación. Estos widgets ayudan a mantener el código limpio, escalable y centrado en la reutilización.



8.7 animated_favorite_icon

Este widget define un ícono de "favorito" animado usando GestureDetector, AnimatedSwitcher y Icon. Permite alternar entre estado de favorito y no favorito con animación de escala al presionar. Es altamente reutilizable.

```
Widget build(BuildContext context) {  
  return GestureDetector(  
    onTap: () => onTap(),  
    child: AnimatedSwitcher(  
      duration: const Duration(milliseconds: 300),  
      child: esFavorito  
        ? const Icon(Icons.favorite, color: Colors.red, key: ValueKey(1))  
        : const Icon(Icons.favorite_border, color: Colors.black, key: ValueKey(0)),  
    ), // AnimatedSwitcher  
  ); // GestureDetector  
}
```



8.8 animated_page_wrapper

Provee un contenedor animado (SlideFadePageWrapper) que permite aplicar transiciones suaves a cualquier child usando animaciones SlideTransition y FadeTransition. Ideal para entradas suaves de pantallas o widgets.

8.9 animations

Contiene funciones y widgets utilitarios para animaciones, como FadeInAnimation y SlideFadeInAnimation. Reutiliza AnimationController y Tween para animaciones complejas que combinan movimiento y opacidad.

Animación de carga con CircularProgressIndicator y Navigator.push para el cambio de pantalla

```
/// Función global para navegar con loading
Future<void> navigateWithLoading(BuildContext context, Widget page, {bool replace = false}) async {
  showDialog(
    context: context,
    barrierDismissible: false,
    builder: (_) => const Center(
      child: CircularProgressIndicator(),
    ), // Center
  );

  await Future.delayed(const Duration(milliseconds: 800)); // simula carga

  Navigator.of(context).pop(); // cierra el loading

  if (replace) {
    Navigator.of(context).pushReplacement(
      MaterialPageRoute(builder: (_) => page),
    );
  } else {
    Navigator.of(context).push(
      MaterialPageRoute(builder: (_) => page),
    );
  }
}
```



8.10 confetti_overlay

Implementa una superposición de confeti con ConfettiWidget. Utiliza OverlayEntry y un controlador (ConfettiController) para mostrar partículas celebratorias.

Establecimiento del comportamiento del confeti y su posicionamiento

```
Widget buildEmitter(Alignment alignment) {  
  return Align(  
    alignment: alignment,  
    child: ConfettiWidget(  
      confettiController: _controller,  
      blastDirection: pi / 2, // Hacia abajo  
      blastDirectionality: BlastDirectionality.directional,  
      emissionFrequency: 0.08,  
      numberOfParticles: 12,  
      gravity: 0.6,  
      maxBlastForce: 20,  
      minBlastForce: 10,  
      shouldLoop: false,  
    ), // ConfettiWidget  
  ); // Align  
}  
  
@override  
Widget build(BuildContext context) {  
  return Positioned.fill(  
    child: IgnorePointer(  
      child: Stack(  
        children: [  
          buildEmitter(Alignment.topLeft),  
          buildEmitter(Alignment.topCenter),  
          buildEmitter(Alignment.topRight),  
        ],  
      ), // Stack  
    ), // IgnorePointer  
  ); // Positioned.fill  
}
```

8.11 Custom_Page_Route

Define rutas de navegación personalizadas (CustomPageRoute) utilizando PageRouteBuilder para animaciones de transición (slide + fade). Controla duración, opacidad y desplazamiento en la navegación entre pantallas.



8.12 LoadingScreen

Proporciona una pantalla de carga animada con un spinner centrado. Usa Scaffold y CircularProgressIndicator, ideal para mostrar durante peticiones de red u operaciones asíncronas.

Pantalla de carga con un CircularProgressIndicator()

```
@override
Widget build(BuildContext context) {
  return Stack(
    children: [
      child,
      Positioned.fill(
        child: Container(
          color: Colors.black.withOpacity(0.3),
          child: const Center(
            child: CircularProgressIndicator(),
          ), // Center
        ), // Container
      ), // Positioned.fill
    ],
  ); // Stack
}
```

8.13 navigation_helper

Función utilitaria navigateWithFade que facilita navegar entre pantallas con transición suave. Reutiliza PageRouteBuilder para animaciones FadeTransition, simplificando el control de navegación visual.



8.14 particle_explosion

Animación avanzada para representar una "explosión de partículas", útil para feedback visual llamativo. Usa CustomPainter, TickerProvider, Canvas y lógica matemática para representar trayectorias de partículas.



Creación de las partículas con propiedades aleatorias

```
_particles = List.generate(35, (_) {  
  final angle = _random.nextDouble() * 2 * pi;  
  final speed = _random.nextDouble() * 6 + 2; // Más rango  
  final size = _random.nextDouble() * 2 + 2;  
  final color = Color.fromARGB(  
    255,  
    _random.nextInt(256),  
    _random.nextInt(256),  
    _random.nextInt(256),  
  ); // Color.fromARGB  
  return _Particle(  
    direction: Offset(cos(angle), sin(angle)) * speed,  
    color: color,  
    size: size,  
  ); // _Particle  
}); // List.generate
```

Función para dibujar las partículas

```
@override  
void paint(Canvas canvas, Size size) {  
  for (final p in particles) {  
    final offset = p.direction * progress * 10;  
    final paint = Paint()..color = p.color.withOpacity(1 - progress);  
    canvas.drawCircle(size.center(offset), p.size * (1 - progress), paint);  
  }  
}
```



8.15 product_card

Widget visual que muestra una tarjeta de producto. Usa GestureDetector, Image.network, Text, y AnimatedFavoritelcon. Ofrece diseño atractivo y preparado para navegación a detalles del producto.

Cambio de pantallas al presionar e información mostrada del producto

```
onTap: () {  
  Navigator.pushNamed(context, '/product', arguments: producto);  
},
```

```
child: Image.network(  
  producto.imagen, // Usando  
  fit: BoxFit.cover,  
  width: double.infinity,  
), // Image.network
```

```
Text(  
  producto.nombre, // Usando nombre desde Firestore  
  style: const TextStyle(fontWeight: FontWeight.bold),  
) , // Text
```

```
Text(  
  "\${producto.precio}", // Usando precio desde Fire  
  style: const TextStyle(color: Colors.green),  
) , // Text
```

8.16 producto_tile

Versión en formato lista de la tarjeta de producto. Presenta imagen, nombre y precio en una fila, optimizada para pantallas como favoritos o resultados de búsqueda. También usa AnimatedFavoritelcon. Igual que el widget anterior pero con diferencias en los tamaños



8.17 slide_fase_page_wrapper

Contenedor reutilizable que envuelve widgets con animaciones combinadas de slide y fade. Se integra con AnimatedPageWrapper para lograr transiciones suaves y reutilizables en distintas pantallas.

Configuración de tiempo y posición de desplazamiento

```
@override
Widget build(BuildContext context) {
  return AnimatedSwitcher(
    duration: const Duration(milliseconds: 600),
    transitionBuilder: (Widget child, Animation<double> animation) {
      final offsetAnimation = Tween<Offset>(
        begin: const Offset(0, -0.1),
        end: Offset.zero,
      ).animate(CurvedAnimation( // Tween
        parent: animation,
        curve: Curves.easeOut,
      )); // CurvedAnimation

      return SlideTransition(
        position: offsetAnimation,
        child: FadeTransition(
          opacity: animation,
          child: child,
        ), // FadeTransition
      ); // SlideTransition
    },
    child: child,
  ); // AnimatedSwitcher
}
```

8.18 SlideFadeIn

Widget animado que aplica animación de entrada con deslizamiento + desvanecimiento. Muy útil para listas o elementos que deben ingresar visualmente al mostrarse.



8.19 Main

Este archivo es el **punto de entrada de la aplicación Flutter**. Se encarga de inicializar la app, configurar proveedores globales (como el carrito de compras y favoritos), y definir la estructura básica de navegación.



Inicialización de Firebase

```
// Inicializa Firebase
await Firebase.initializeApp(
  options: DefaultFirebaseOptions.currentPlatform,
);
```

Ejecución inicial de funciones de múltiples proveedores

```
MultiProvider(
  providers: [
    ChangeNotifierProvider(create: (_) => CartProvider()..obtenerCarrito()),
    ChangeNotifierProvider(create: (_) => FavoritosModel()..obtenerFavoritos()),
    ChangeNotifierProvider(create: (_) => ProductosModel()..obtenerProductos()),
  ],
  child: const MyApp(),
), // MultiProvider
```

Establecimiento del nombre de las rutas en las que se navegara después

```
routes: {
  '/welcome': (context) => const WelcomeScreen(),
  '/login': (context) => const LoginScreen(),
  '/registro': (context) => const RegistroScreen(),
  '/': (context) => const InicioScreen(),
  '/cart': (context) => const CartScreen(),
  '/perfil': (context) => const ProfileScreen(),
  '/pago': (context) => const PagoScreen(),
  // '/resumen' se maneja dinámicamente con onGenerat
},
```



Realizar instrucciones al momento de navegar a la ruta especificada




```
onGenerateRoute: (settings) {  
  if (settings.name == '/product') {  
    final productId = settings.arguments as String;  
    return MaterialPageRoute(  
      builder: (_) => ProductDetailScreen(productId: productId),  
    ); // MaterialPageRoute  
  }  
  
  if (settings.name == '/resumen') {  
    final args = settings.arguments as Map<String, dynamic>;  
    return MaterialPageRoute(  
      builder: (_) => ResumenScreen(  
        direccion: args['direccion'],  
        productos: args['productos'],  
        total: args['total'],  
      ), // ResumenScreen  
    ); // MaterialPageRoute  
  }  
  
  return null;  
}
```



9 Configuración

9.1 firebase_options



 `firebase_options.dart`

Este archivo contiene la **configuración automática** generada por Firebase para que la app Flutter pueda conectarse correctamente a los servicios de Firebase (como Auth, Firestore, etc.).

Especifica la plataforma que se utilizara para la aplicación

```
static FirebaseOptions get currentPlatform {  
  if (kIsWeb) {  
    return web;  
  }  
  switch (defaultTargetPlatform) {  
    case TargetPlatform.android:  
      return android;  
    case TargetPlatform.iOS:  
      return ios;  
    case TargetPlatform.macos:  
      return macos;  
    case TargetPlatform.windows:  
      return windows;  
    case TargetPlatform.linux:  
      throw UnsupportedError(  
        'DefaultFirebaseOptions have not been configured for linux - '  
        'you can reconfigure this by running the FlutterFire CLI again.',  
      );  
    default:  
      throw UnsupportedError(  
        'DefaultFirebaseOptions are not supported for this platform.',  
      );  
  }  
}
```

9.2 firebase.json

 `firebase.json`

Es un archivo de configuración **usado por Firebase CLI** para definir comportamientos relacionados con el **hosting web** (si lo usas), así como reglas, funciones, y otros recursos de Firebase.



9.3 pubspec.yaml



 pubspec.yaml

Es el archivo de configuración principal de un proyecto Flutter. Define metadatos, dependencias, assets, fonts y más.

Dependencias	Uso
cupertino_icons	Proporciona íconos estilo iOS (usados por defecto en apps con diseño Cupertino). No se usa directamente, pero puede aparecer en temas o botones si usas widgets iOS.
provider	Gestión de estado. Se utiliza para controlar favoritos (FavoritosModel), carrito de compras (CartProvider), y probablemente otros datos globales.
carousel_slider	Crea carruseles deslizables. Se usa para mostrar productos destacados, sugerencias, promociones, o imágenes de productos.
flutter_typeahead	Muestra sugerencias mientras el usuario escribe. Se usa en el buscador para autocompletar productos.
google_maps_flutter	Muestra mapas de Google en la app. Se usa para indicar ubicaciones (como tiendas, puntos de entrega o usuario).
firebase_core	Inicializa Firebase en la app. Requerido para usar cualquier servicio de Firebase (Auth, Firestore, etc.).
firebase_auth	Gestión de usuarios: registro, inicio de sesión, recuperación de contraseña. Se utiliza para autenticar a los usuarios.
cloud_firestore	Base de datos en tiempo real. Se usa para almacenar y recuperar productos, favoritos, usuarios, pedidos, etc.
animations	Animaciones predefinidas (fade, scale, slide, etc.). Mejora transiciones y UI. Puede estar en widgets personalizados o pantallas.
audioplayers	Reproduce efectos de sonido o música. Se puede usar para confirmar acciones, alertas, o celebraciones.
confetti	Muestra animaciones de confeti. Generalmente se usa como efecto visual en eventos importantes (ej. compra exitosa).



intl	<i>Internacionalización y formato de fechas, monedas, etc. Se usa en precios, fechas de pedido, formato de números.</i>
image_picker	<i>Permite al usuario seleccionar o tomar fotos. Se usa para subir imágenes de perfil, productos, etc.</i>
connectivity_plus	<i>Detecta si hay conexión a Internet. Se usa para mostrar mensajes de error si no hay red o ajustar comportamiento offline.</i>
sqflite	<i>Base de datos SQLite local. Se usa para almacenamiento persistente local, como historial de búsqueda, favoritos offline o carrito.</i>
path	<i>Manipula rutas de archivos. Soporte para funciones de almacenamiento o acceso a archivos locales. Complementa sqflite o image_picker.</i>
path_provider	<i>Localiza directorios seguros del sistema (como cache o documentos). Se usa para guardar imágenes, archivos temporales, etc.</i>
flutter_image_compress	<i>Comprime imágenes antes de subirlas o almacenarlas. Mejora el rendimiento, ahorro de datos y espacio de almacenamiento.</i>

Archivos dentro de assets para utilizarse a lo largo de la aplicación

```
flutter:  
  uses-material-design: true  
  assets:  
    - assets/sounds/explosion.mp3  
    - assets/avatar.png
```