



Módulo 08: Programación multimedia y dispositivos móviles

ALUMNO: JUAN XIAO PAREDES CARRERO-BLANCO (S2AM)
Profesor: Sergi Carreras Sala
Curso: 2024-2025

Índice

1. Flutter.....	3
1.1 Introducción a Flutter.....	3
1.2 Descarga e Instalación.....	3
2. Android Studio.....	7
2.1 Introducción a Android Studio.....	7
2.2 Descarga e Instalación.....	8
3. Visual Studio Code	17
3.1 Introducción a Visual Studio Code.....	17
3.2 Descarga e Instalación	17
4. Nuestra primera Aplicación.....	20
4.1 Nuestra primera Aplicación con Android Studio	20
4.2 Mi primera Aplicación en Visual Studio Code	22
5. Crear una aplicación por Windows con Flutter	24
6. Crear y entender una UI con FlutterFlow.io	24
6.1 Mi FlutterFlow.io personalizado	27
7. Formularios.....	29
7.1 Formularios de Ejemplo	29
7.2 Formularios en FormBuilder	31
7.2.1 Formulario A	31
7.2.2 Formulario B	32
7.2.3 Formulario C	34
7.2.4 Formulario D	35
8 APIS	36
8.1 Introducción.....	36
8.2 Mi API personalizada.....	37
8.2.1 Main	37
8.2.2 Añadir una nueva serie	40
8.2.3 Detalles de las cartas	42
8.2.4 Detalles de las paginas	45
8.2.5 Listas de Series.....	48
8.2.6 Conectarse a mi api	48
8.3 Probar mi API	49
9 Api para mostrar Actores	53
9.1 Introducción al ejercicio	53
9.2 Modificación del Código	54

9.2.1 Primer paso dar de alta	54
9.2.2 Copiar de los archivos del Programa	55
9.2.4 Añadir lógica a los archivos de actores.....	56
9.2.4.5 Carpeta Widgets	66
9.3 Funcionamiento de la aplicación	67
10. SupaBase	70
10.1 Introducción.....	70
10.1.1 Crear una cuenta de SupaBase	71
10.2 Introducción al Ejercicio	71
10.3 DBeaver.....	72
10.3.1 Preparación del DBeaver con SupaBase	73
10.4 Explicación del Código	84
10.4.1 Explicación del Ejemplo1 del PPT.....	84
10.4.2 Explicación del Ejemplo2 del PPT.....	93

1. Flutter

1.1 Introducción a Flutter

Flutter es un software development kit para interfaces de usuario de código abierto creado por Google. Se usa para desarrollar aplicaciones cross platform desde una sola base de código para web, Android, iOS, Fuchsia, Linux, macOS y Windows. Flutter fue lanzado en mayo de 2017. Google lo usa internamente para aplicaciones como Google Pay y Google Earth. También se usa en otras empresas como ByteDance y Alibaba.

En el último año, ha sufrido un crecimiento muy grande en cuanto a su popularidad. Eso se debe a su velocidad de desarrollo, experiencia nativa y renderización de la interface. El 3 de marzo de 2021, en el evento virtual llamado "Flutter Engage", Google lanzó Flutter 2. Este fue el cambio oficial más grande que tuvo el SDK.

1.2 Descarga e Instalación

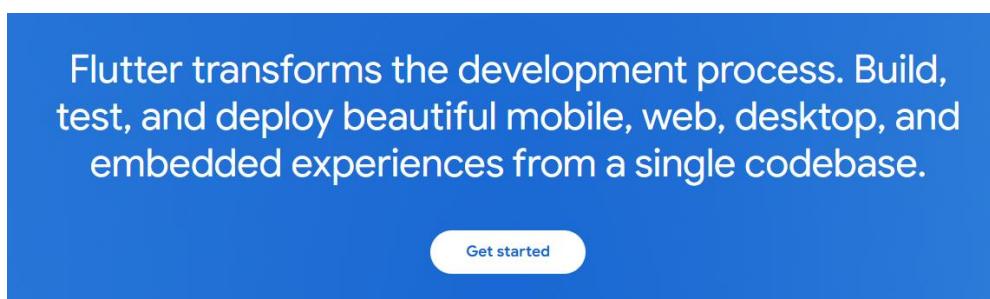
Ahora vamos a seguir lo que nos piden que hagamos en la práctica que es descargarnos el Android Studio, el Visual Studio Code y el Flutter, para eso voy a mostráros lo que nos piden en la práctica:

- Queremos instalar los entornos de desarrollo Android Studio y Visual Studio Code en el portátil con las extensiones para trabajar con Dart y Flutter.
- Una vez instalados ejecutaremos una aplicación de ejemplo en cada IDE y realizaremos unas pequeñas modificaciones. Comprobaremos que se ejecutan correctamente en el móvil.
- Habrá que crear una cuenta personal en GitHub, donde iremos subiendo todas las App's del curso y su documentación.
- ¡¡¡CUENTA!!!! En todas las actividades es necesario generar una buena documentación, explicando todos los pasos que hemos necesitado para conseguir nuestros objetivos y mostrando imágenes de ejemplo.

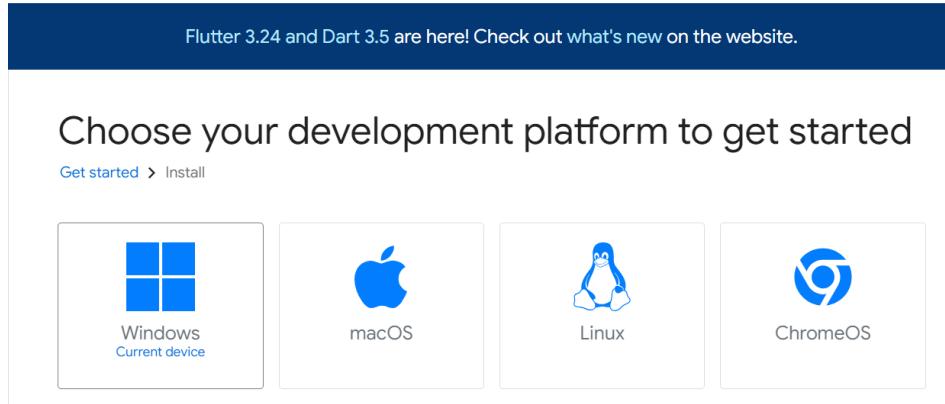
Ahora vamos a empezar a instalarnos el Flutter, para eso lo primero que haremos será ir a su pagina oficial donde nos podremos descargar el Flutter, también os dejo abajo un link que os llevará directamente a su página.

Enlace: <https://flutter.dev/>

Una vez dentro del la web, desplazaremos la rueda del ratón para abajo y donde '**Get Started**' le daremos para empezar a descargar e instalar.



Una vez dentro, nos mostrara otra pantalla donde tendremos que elegir el tipo de plataforma en la que nos queremos descargar, en mi caso elegire la de Windows, pero vosotros podreis elegir cualquiera de las que hay.



Flutter 3.24 and Dart 3.5 are here! Check out what's new on the website.

Choose your development platform to get started

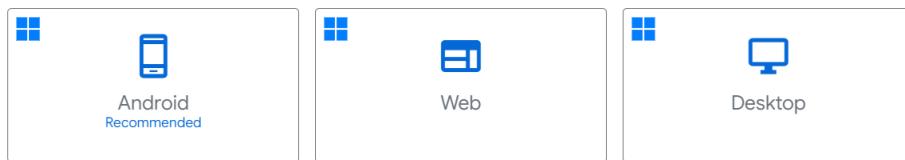
Get started > Install

-  Windows
Current device
-  macOS
-  Linux
-  ChromeOS

Después de elegir nuestro tipo de plataforma en la que queremos que se nos descargue, nos llevara a elegir el tipo de aplicación que queremos orientado a hacer, en nuestro caso como el modulo va de dispositivos móviles, elegiremos el de Android. **Advertencia**, si tienes iPhone, no podrás hacer la práctica, porque el Flutter no nos deja la opción, por eso es necesario un móvil Android.

Choose your first type of app

Get started > Install > Windows



-   Android
Recommended
-  Web
-  Desktop

Your choice informs which parts of Flutter tooling you configure to run your first Flutter app. You can set up additional platforms later. If you don't have a preference, choose [Android](#).

Cunado estemos dentro nos mostrar mucha información sobre como instalar, configuraciones antes de la instalación, dentro de la instalación, pero lo primero que queremos hacer es descargar el Flutter, para eso nos iremos para abajo y nos detendremos donde pone '**Instalar el SDK de Flutter**'. Una vez localizado procederemos a seleccionar la opción de descargar e instalar y le daremos al zip para que empiece la descarga.

Instalar el SDK de Flutter

Para instalar el SDK de Flutter, puedes usar la extensión Flutter de VS Code o descargar e instalar el paquete de Flutter tú mismo.

Utilice VS Code para instalar [Descargar e instalar](#)

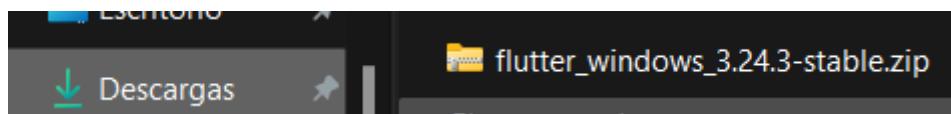
Descargue y luego instale Flutter

Para instalar Flutter, descargue el paquete Flutter SDK de su archivo, mueva el paquete a donde quiera almacenarlo y luego extraiga el SDK.

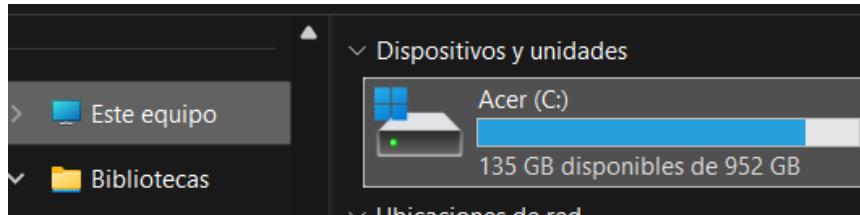
1. Descargue el siguiente paquete de instalación para obtener la última versión estable del SDK de Flutter.

[flutter_windows_3.24.3-stable.zip](#)

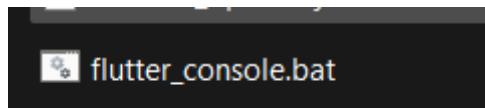
Cuando acabe de descargar el zip, nos lo encontraremos en descargas y lo único que tendremos que hacer será descomprimirlo.



Recomendación, al descomprimir el zip, será mejor que lo guardes en tu dispositivo, por seguridad y genera una carpeta dentro donde puedas guardar el Flutter.



Cuando lo tengamos descargado y guardado correctamente, nos iremos a la carpeta de **flutter** y la abriremos y buscaremos un **.bat** para comprobar que todo este correcto a la hora de poder conectar al *Arduino* y el *Visual*.



Cuando lo abramos se nos abrirá una pantalla de control de sistema que en teoría es la consola del Flutter. En ella podremos observar que nos muestra un par de comandos que podemos realizar, como uno de ellos es el; **flutter help**, o el **flutter doctor** (que es el que vamos a utilizar mas a menudo para esta practica).

```
#####
##      ##
##      ##      ##      ##      ##      ##      ##      ##
##      ##      ##      ##      ##      ##      ##      ##
#####      ##      ##      ##      ##      ##      ##      ##
##      ##      ##      ##      ##      ##      ##      ##
##      ##      ##      ##      ##      ##      ##      ##
##      ##      ##      ##      ##      ##      ##      ##

WELCOME to the Flutter Console.
=====
Use the console below this message to interact with the "flutter" command.
Run "flutter doctor" to check if your system is ready to run Flutter apps.
Run "flutter create <app_name>" to create a new Flutter project.

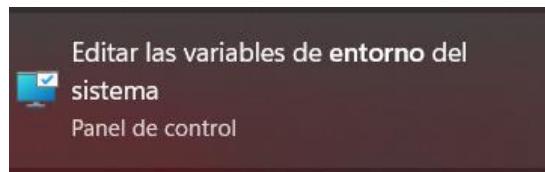
Run "flutter help" to see all available commands.

Want to use an IDE to interact with Flutter? https://flutter.dev/ide-setup/
Want to run the "flutter" command from any Command Prompt or PowerShell window?
Add Flutter to your PATH: https://flutter.dev/setup-windows/#update-your-path

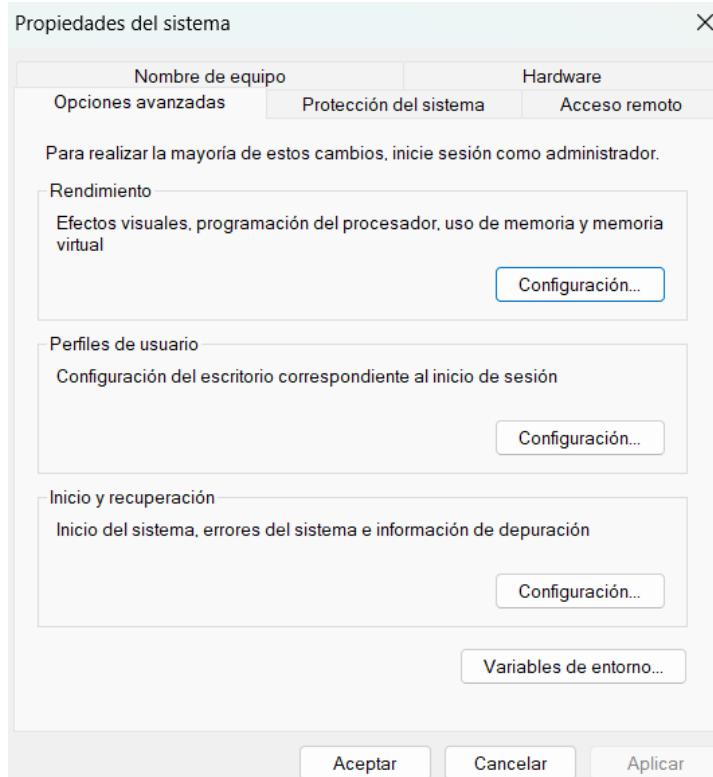
=====

C:\Users\juanp>
```

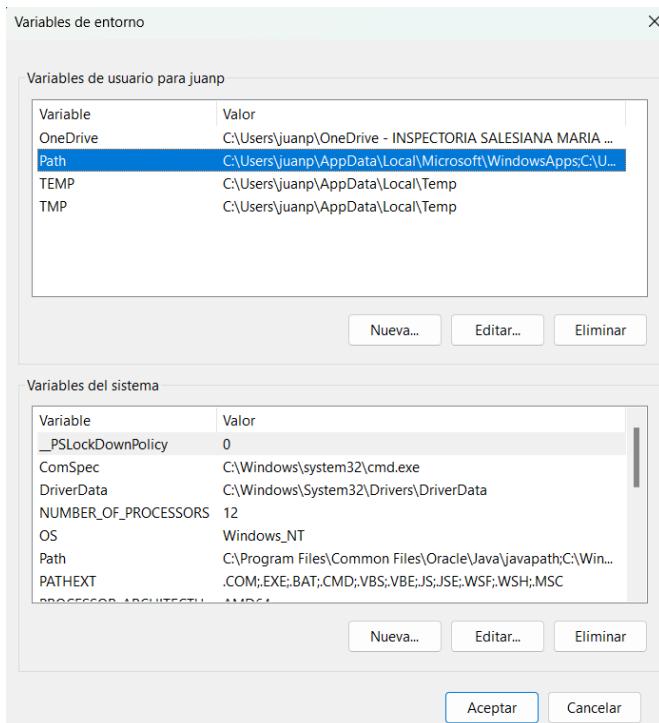
Ahora nos iremos al símbolo de Windows y buscaremos en el panel de control, pondremos **entorno**, y le daremos a la primera opción que nos muestra.



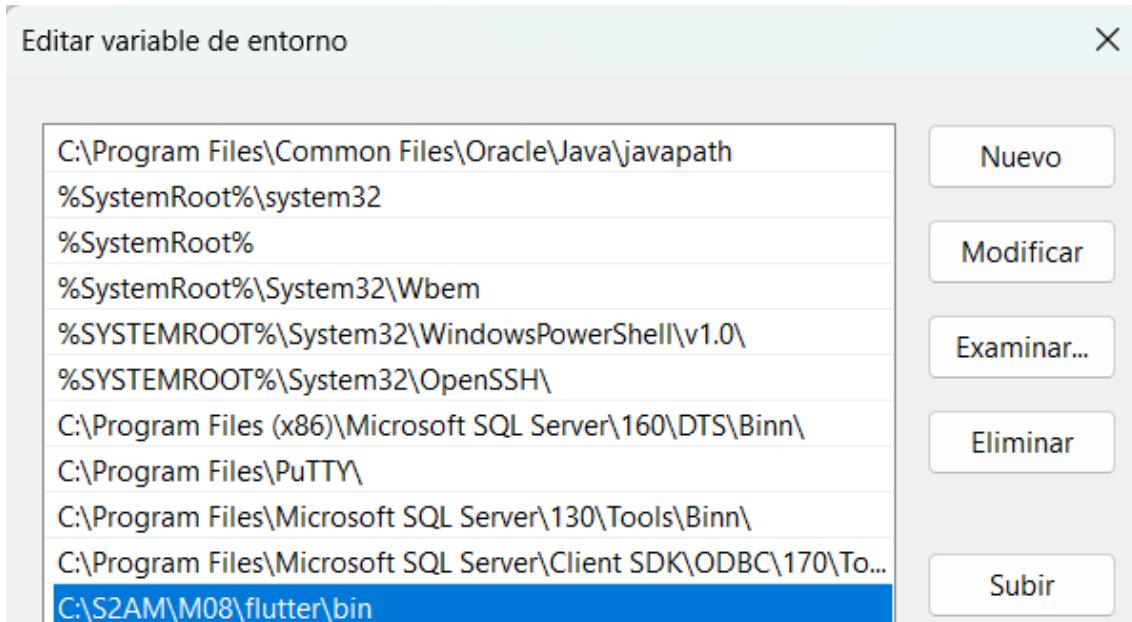
Una vez dentro se nos mostrará una nueva pestaña donde podremos ver varias opciones que podemos editar, si nos fijamos, la que nos interesa será la de '**Variables de entorno**'.



Cuando le hayamos dado nos mostrara una pantalla donde tendremos que seleccionar el que dice '**Path**' y le damos a editar. Advertencia, tenemos que hacerlo en el de arriba, no en el de abajo.



Entonces tendremos que copiar la ruta de acceso de donde tenemos el flutter la carpeta de '**bin**'. Y añadiremos una nueva ruta y después aceptaremos todo.



Cuando lo tengamos todo, tendremos uno de los muchos pasos a realizar, entonces podremos empezar a descargar el **Android Studio**.

2. Android Studio

2.1 Introducción a Android Studio

Android Studio es el entorno de desarrollo integrado oficial para la plataforma Android. Fue anunciado el 16 de mayo de 2013 en la conferencia Google I/O, y reemplazó a Eclipse como el IDE oficial para el desarrollo de aplicaciones para Android. La primera versión estable fue publicada en diciembre de 2014.

Está basado en el software IntelliJ IDEA Community Edition de JetBrains y ha sido publicado de forma gratuita a través de la Licencia Apache 2.0. Está disponible para las plataformas. Ha sido diseñado específicamente para el desarrollo de Android.

Estuvo en etapa de vista previa de acceso temprano a partir de la versión 0.1, en mayo de 2013, y luego entró en etapa beta a partir de la versión 0.8, lanzada en junio de 2014. La primera compilación estable, la versión 1.0, fue lanzada en diciembre de 2014.

Desde el 7 de mayo de 2019, Kotlin es el lenguaje preferido de Google para el desarrollo de aplicaciones de Android. Aun así, Android Studio admite otros lenguajes de programación, como Java y C++.



2.2 Descarga e Instalación

Para descargarnos el Android Studio lo primero que vamos a hacer es ir a su página web e instalarnos la versión mas nueva o la más actualizada.

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

Cuando le demos a la opción de descargar, se nos mostrara una pantalla donde nos pondrás unos términos y condiciones donde tendremos que leerlas y aceptarlas, una vez aceptada ya nos dejara descargar el **Android Studio**.

El Contrato de Licencia, que estará disponible en el sitio web donde se download el SDK.

14. Condiciones legales generales

14.1 El Contrato de Licencia representa en su totalidad el contrato legal entre usted y Google, regulando (o proporcionarle conforme a un contrato por escrito independiente) y reemplaza por completo cualquier acuerdo que, si Google no ejerce ni impone un derecho o solución legal especificado en el Contrato (aplicable), esto no se considerará como una renuncia a los derechos de Google y se entenderá que 14.3 Si un tribunal que tenga jurisdicción para decidir sobre este asunto dictamina que alguna disposición afecta al resto del Contrato de Licencia. Las disposiciones restantes del Contrato de Licencia para los miembros del grupo de empresas de las que Google es la casa matriz serán beneficiarios terceros y no podrán imponer directamente cualquier disposición del Contrato de Licencia que les confiera un beneficio de lo mencionado, nadie más ni ninguna empresa serán beneficiarios terceros del Contrato de Licencia. LAS LEYES Y NORMATIVAS DE EXPORTACIÓN DE ESTADOS UNIDOS. USTED DEBE CUMPLIR CON EXPORTACIÓN QUE SE APLICAN AL SDK. ESTAS LEYES INCLUYEN RESTRICCIONES SOBRE LOS DERECHOS QUE PODRÁN ASIGNAR NI TRANSFERIR LOS DERECHOS QUE SE OTORGAN EN EL CONTRATO DE LICENCIA SIN LA APROBACIÓN DE LAS RESPONSABILIDADES U OBLIGACIONES DEL CONTRATO DE LICENCIA SIN LA APROBACIÓN PREVIA POR ESCRITO DE UN CONTRATO DE LICENCIA SE REGIRÁN POR LAS LEYES DEL ESTADO DE CALIFORNIA, INDEPENDIENTEMENTE DE SU JURISDICCIÓN EXCLUSIVA DE LOS TRIBUNALES UBICADOS EN EL CONDADO DE SANTA CLARA, CALIFORNIA, PARA QUÉ ESTÁ RELACIONADO CON ÉL. NO OBSTANTE, USTED ACEPTA QUE GOOGLE AÚN PODRÁ SOLICITAR RECURSOS JUDICIALES EN SU JURISDICCIÓN. 27 de julio de 2021

Leí y acepto los Términos y Condiciones anteriores

[Descargar Android Studio Koala Feature Drop | 2024.1.2 para Windows](#)

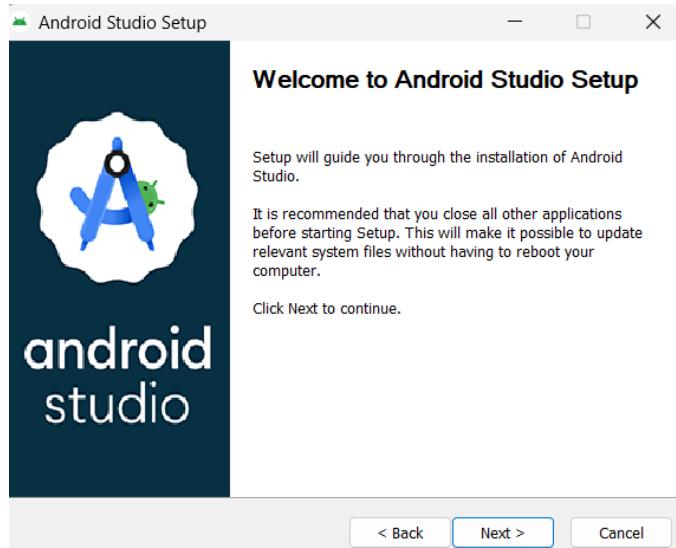
android-studio-2024.1.2.12-windows.exe

Y ya tendremos el Android para instalar, simplemente le vamos a abrir y empezaremos toda la configuración básica.



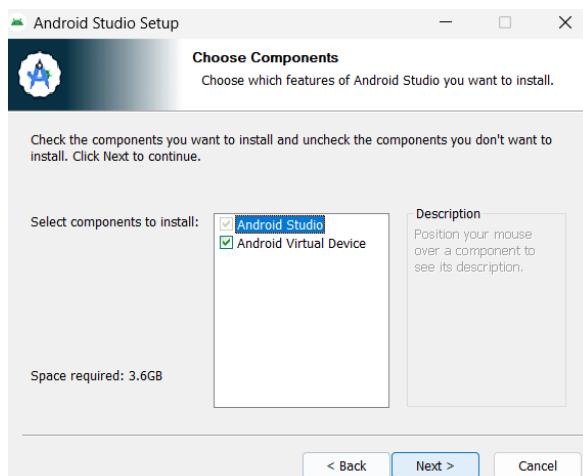
android-studio-2024.1.2.12-windows.exe

Entonces se nos abrirá una nueva pantalla donde nos mostrara un poco sobre la historia del programa, simplemente le daremos a siguiente, y podremos continuar la instalación.

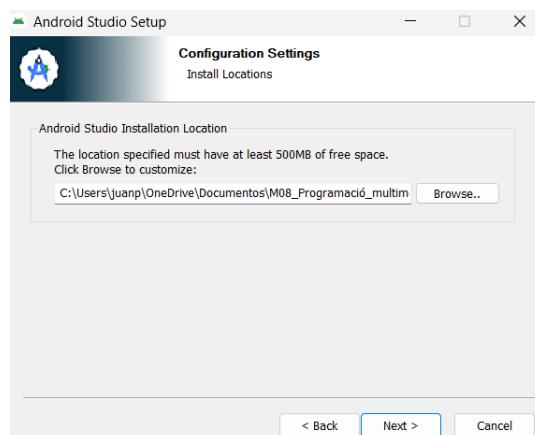


Después nos
nueva y aquí
que elegir el tipo de
simplemente seleccionamos los dos y le damos a siguiente.

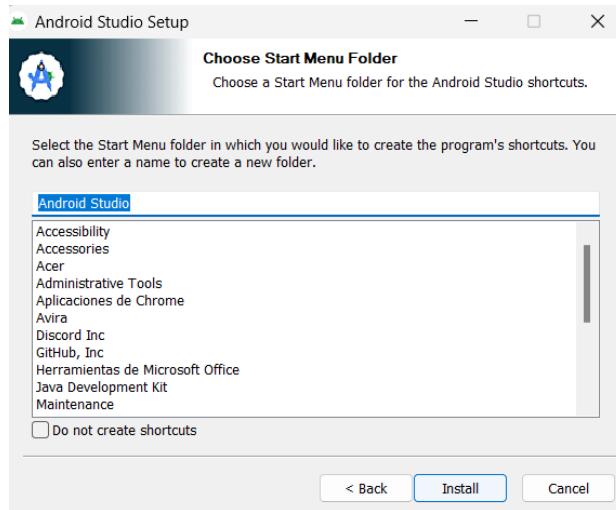
mostrará una
tendremos
componentes,



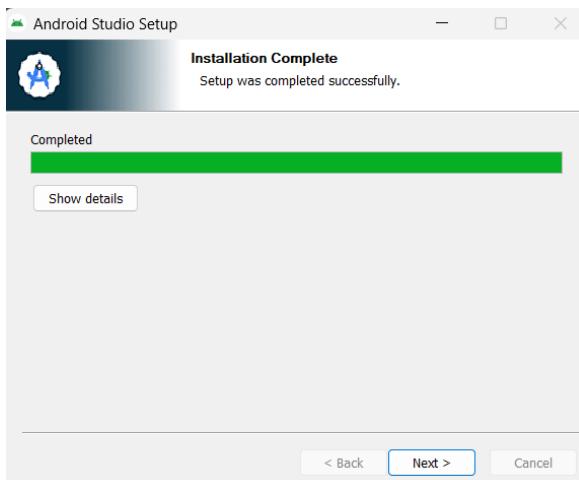
Entonces nos pedirá que seleccionemos donde queremos que se nos instale y donde
guardarla.



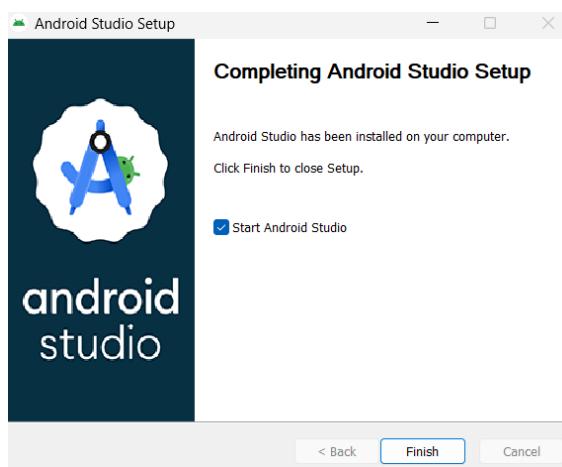
A más, a más, se nos abrirá otra pestaña nueva donde esta vez tendremos que seleccionar la
carpeta de menú, y después le daremos a instalar.



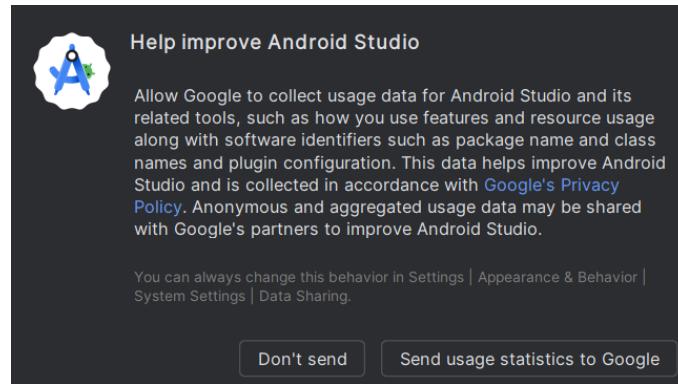
Se nos abrirá una pestaña de carga de la instalación, y cuando acabe le daremos a siguiente.



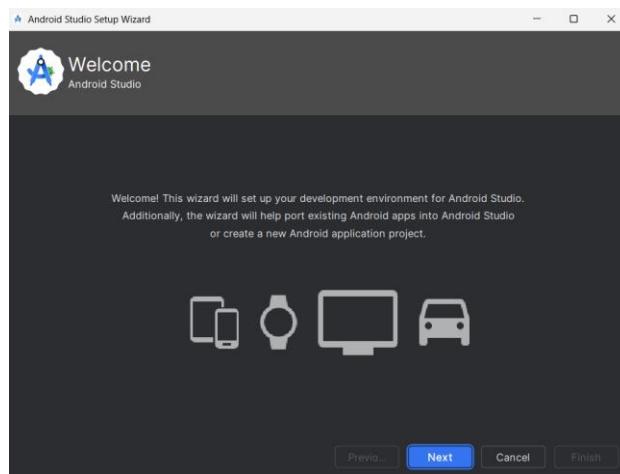
Entonces ya casi podremos comenzar a trabajar con el programa, simplemente le damos a finish y se nos abrirá.



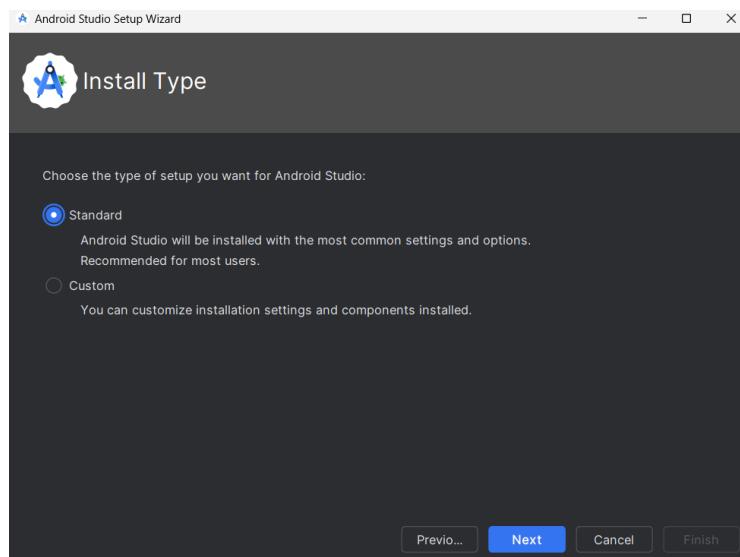
En esta pestaña nos dice información para mejorar el Android pero nosotros no estamos interesados, no es necesario, aquí nosotros le daremos a '**'Don't send'**'.



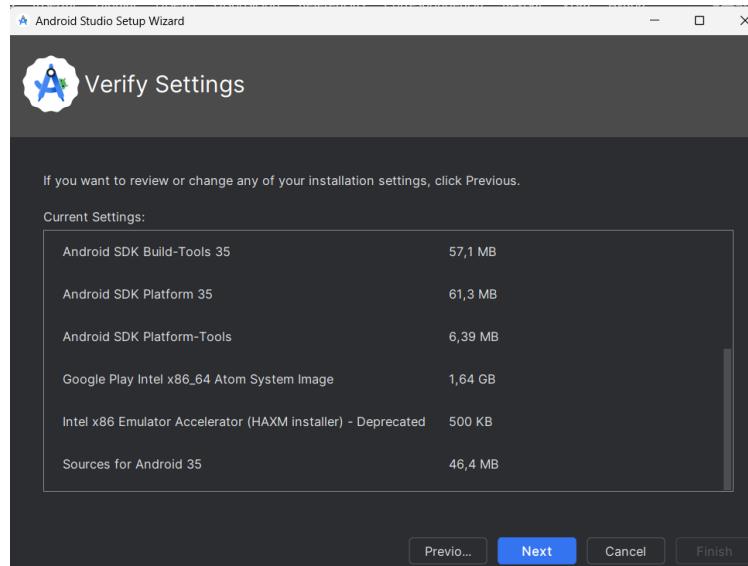
Ahora ya solo faltara una pequeña instalación y configuración y ya podremos empezar, le daremos a siguiente.



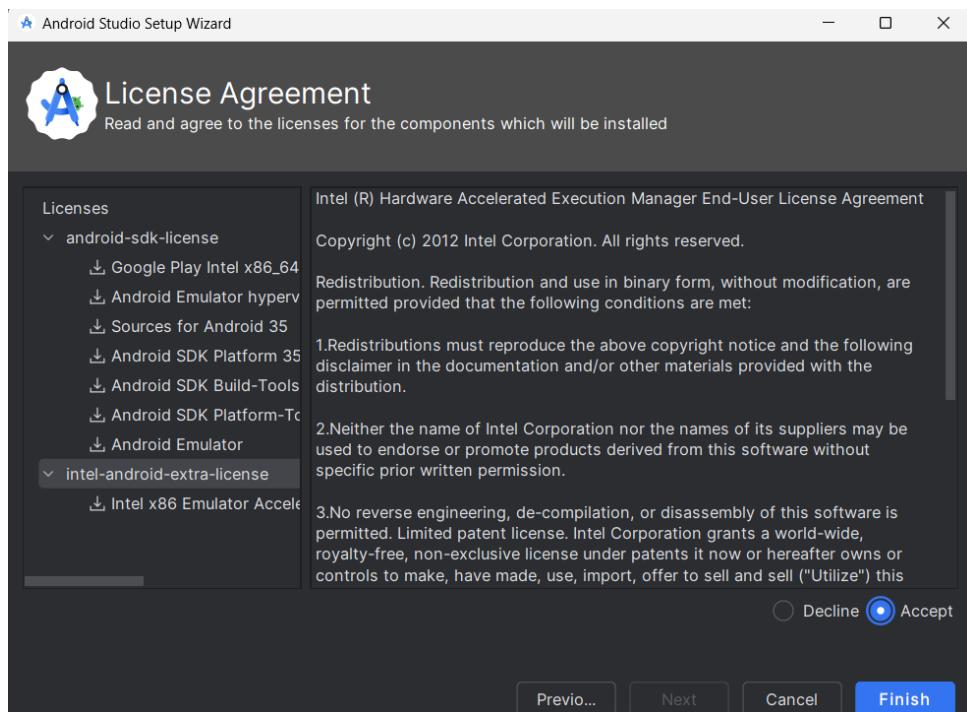
Aquí lo dejamos predeterminado y le damos a siguiente.



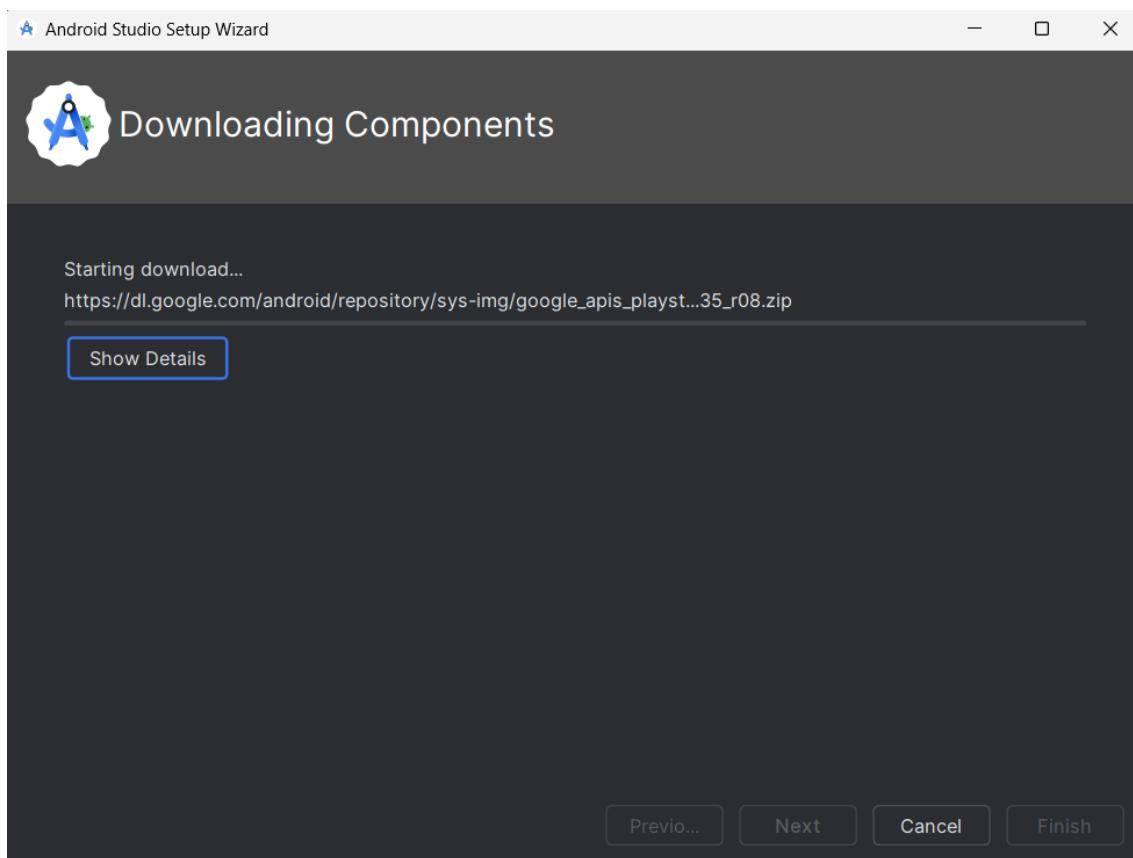
Aquí podremos ver todo lo que nos va a instalar, y le daremos a siguiente.



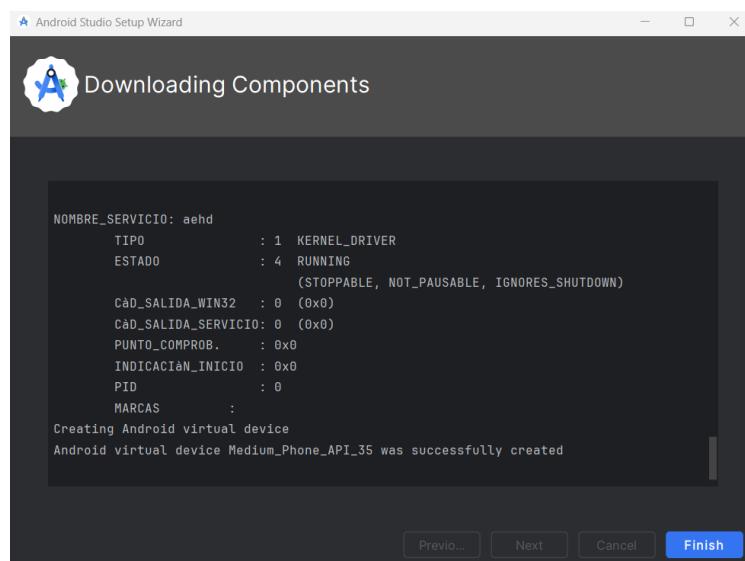
Ahora seleccionamos la casilla de '**Accept**' y le damos a finish.



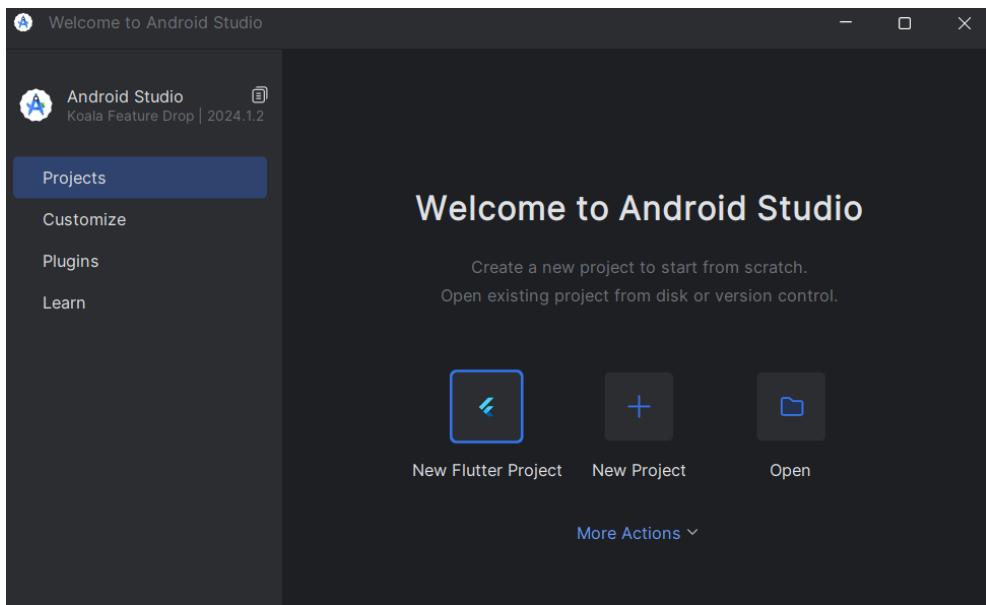
Y empezara a descargar todos los componentes necesarios para el Android.



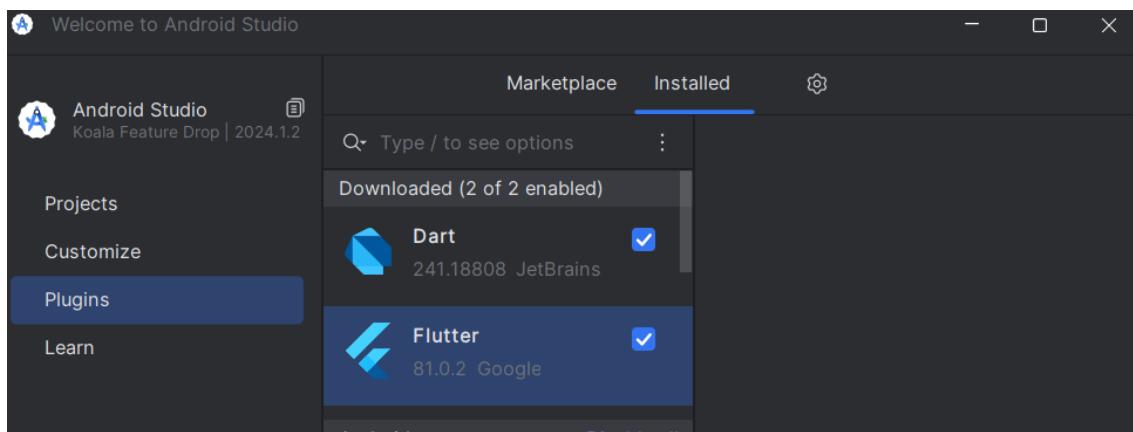
Una vez finalizado le daremos a otra vez a finish.



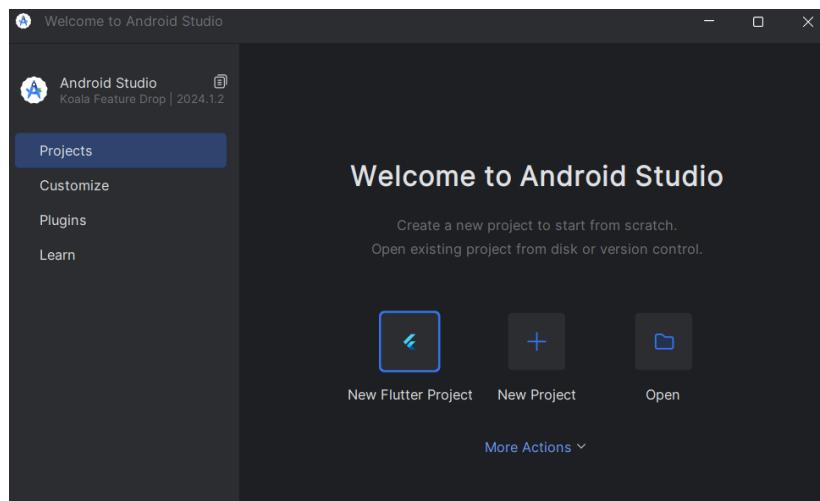
Ahora ya podremos empezar a utilizar el Andriod.



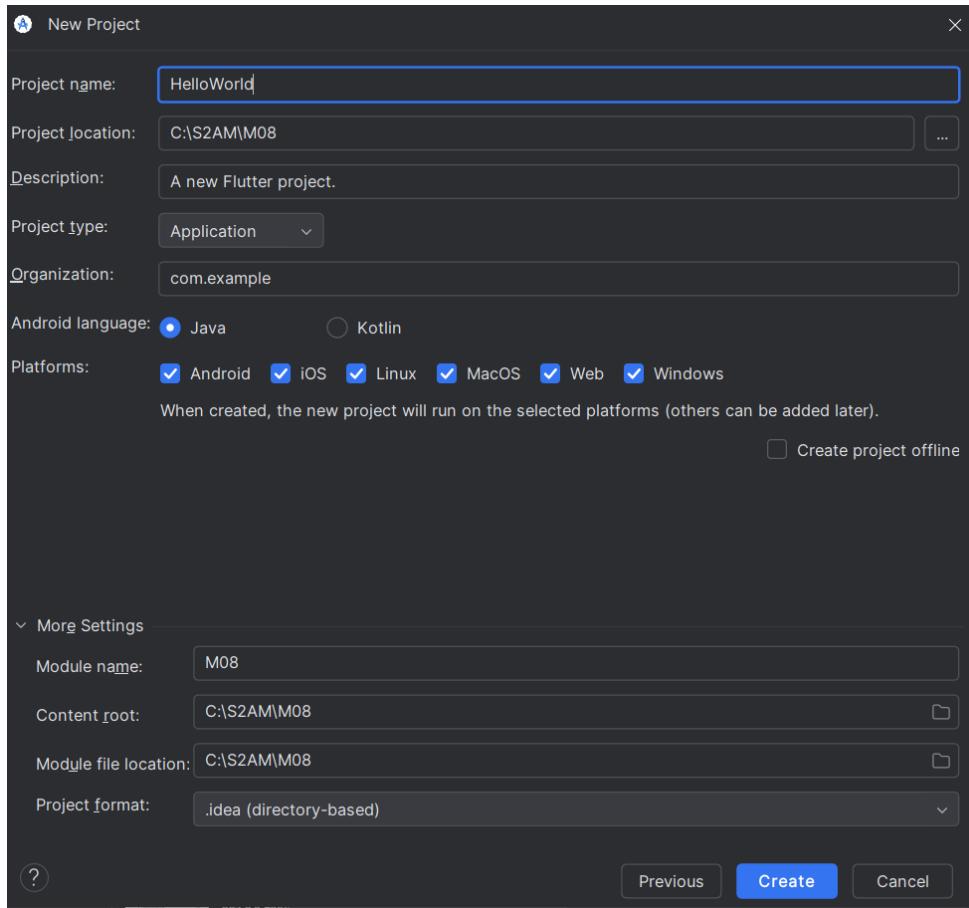
Pero para que el Flutter y el Android Studio se conecte vamos a necesitar instalar las extensiones, para eso nos vamos a la opción de '**Plugins**' y buscamos en el buscador Flutter y nos la instalamos.



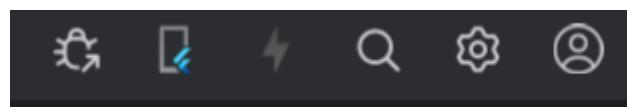
Una vez descargado ya podremos empezar a utilizar le Android con Flutter, y vamos a darle a nuevo proyecto per con el Flutter.



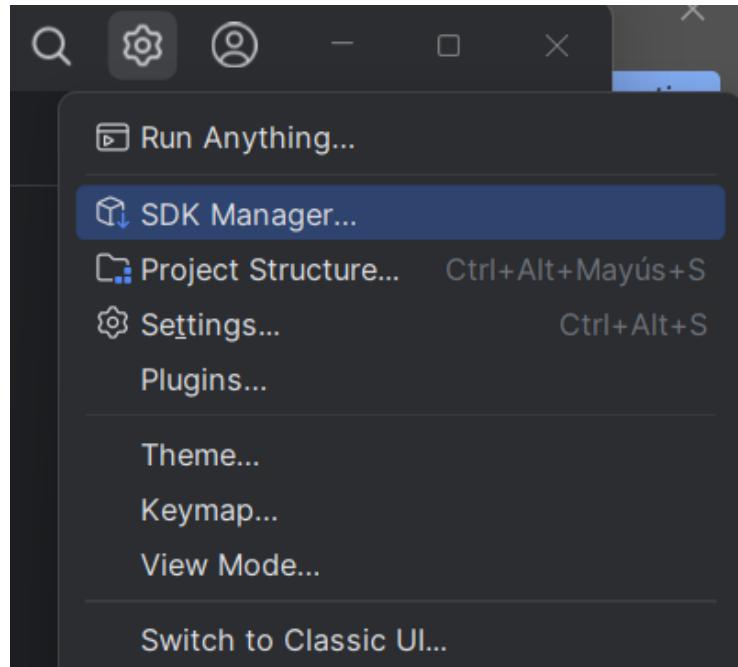
Cuando hayamos clicado nos pedirá que le pongamos el nombre del proyecto, la ubicación donde se guardara, el tipo de lenguaje que será Java y ya lo podremos crear.



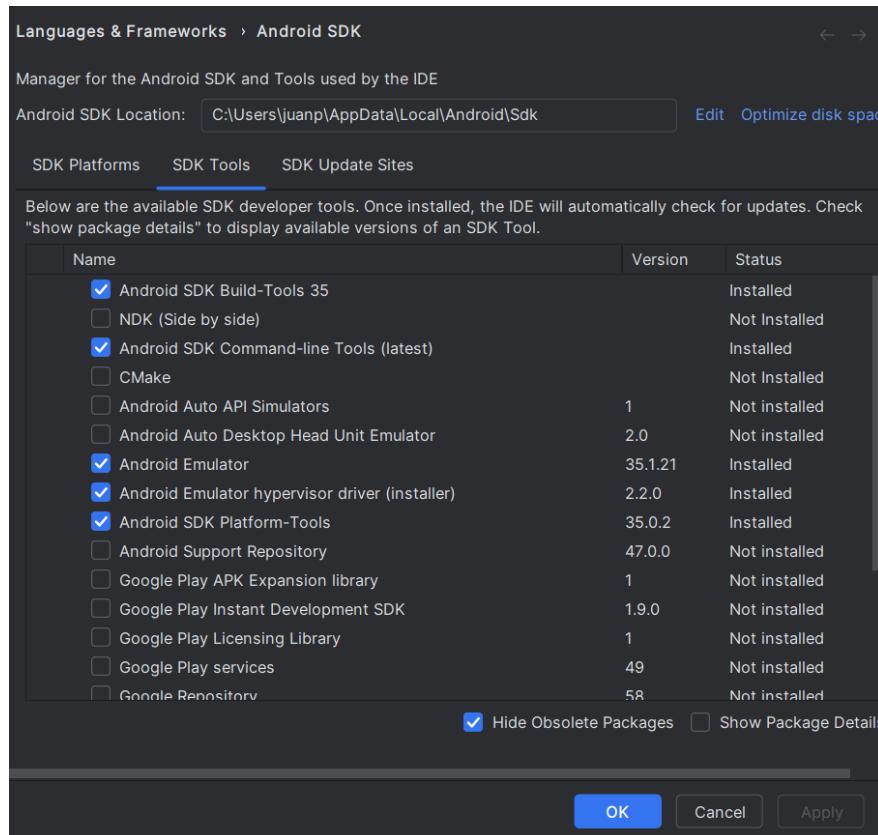
Cuando se nos haya creado, necesitamos ir primero de todo arriba a la derecha, a los settings para configurar el SDK.



Cuando le demos, se nos aparecerá una desplegable con muchas opciones, en nuestro caso vamos a primero seleccionar la opción de '**SDK Manager...**'.



Cuando estemos dentro, tendremos que seleccionar en el apartado de '**SDK Tools**' el '**Android SDK Command-line Tools (latest)**'. Una vez seleccionado, le daremos a ok, y se nos descargara.



3. Visual Studio Code

3.1 Introducción a Visual Studio Code

Visual Studio Code es un editor de código fuente desarrollado por Microsoft. Es ligero, potente, y ampliamente utilizado para programar en diversos lenguajes como Python, JavaScript, C++, entre otros. Su versatilidad lo hace ideal para estudiantes y profesionales.

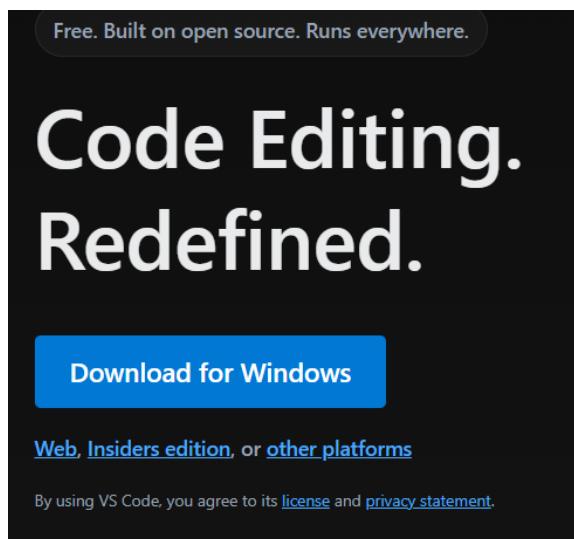
Visual Studio Code (también llamado VS Code) es un editor de código fuente desarrollado por Microsoft para Windows, Linux, macOS y Web. Incluye soporte para la depuración, control integrado de Git, resalutación de sintaxis, finalización inteligente de código, fragmentos y refactorización de código. También es personalizable, por lo que los usuarios pueden cambiar el tema del editor, los atajos de teclado y las preferencias. Es software gratuito y de código abierto, aunque la descarga oficial está bajo software privativo e incluye características personalizadas por Microsoft.

Visual Studio Code se basa en Electron, un framework que se utiliza para implementar Chromium y Node.js como aplicaciones para escritorio, que se ejecuta en el motor de diseño Blink. Aunque utiliza el framework Electron, el software no usa Atom y en su lugar emplea el mismo componente editor (Mónaco) utilizado en Visual Studio Team Services (anteriormente llamado Visual Studio Online).

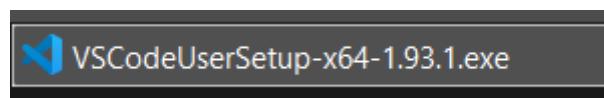
3.2 Descarga e Instalación

Para descargar e instalar el Visual, lo primero que vamos a hacer es ir a su página oficial y buscar el apartado de descargas, abajo tenéis un enlace que os lleva a su página web. Y la vamos a dar a '**Download**'.

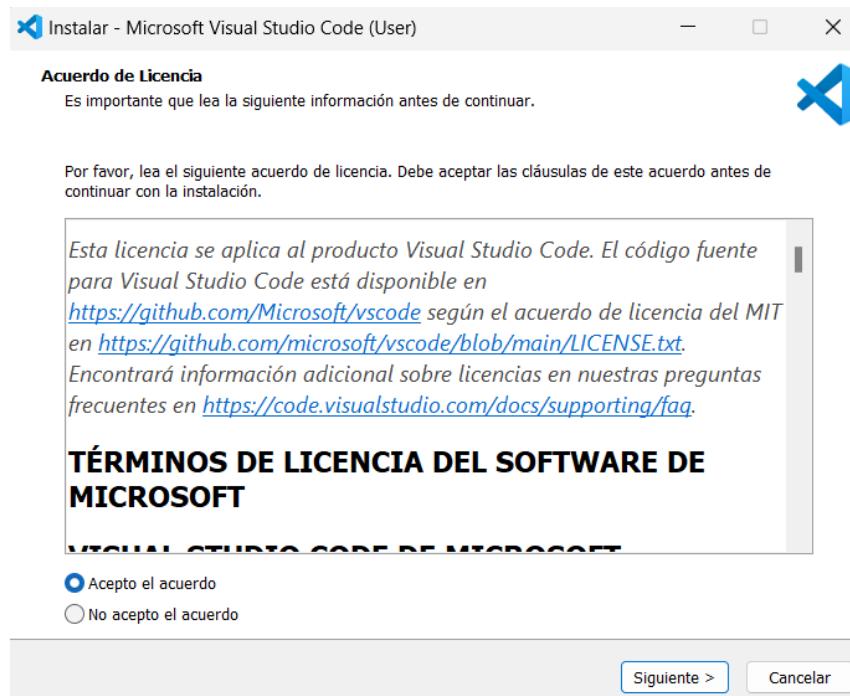
Enlace: <https://code.visualstudio.com/>



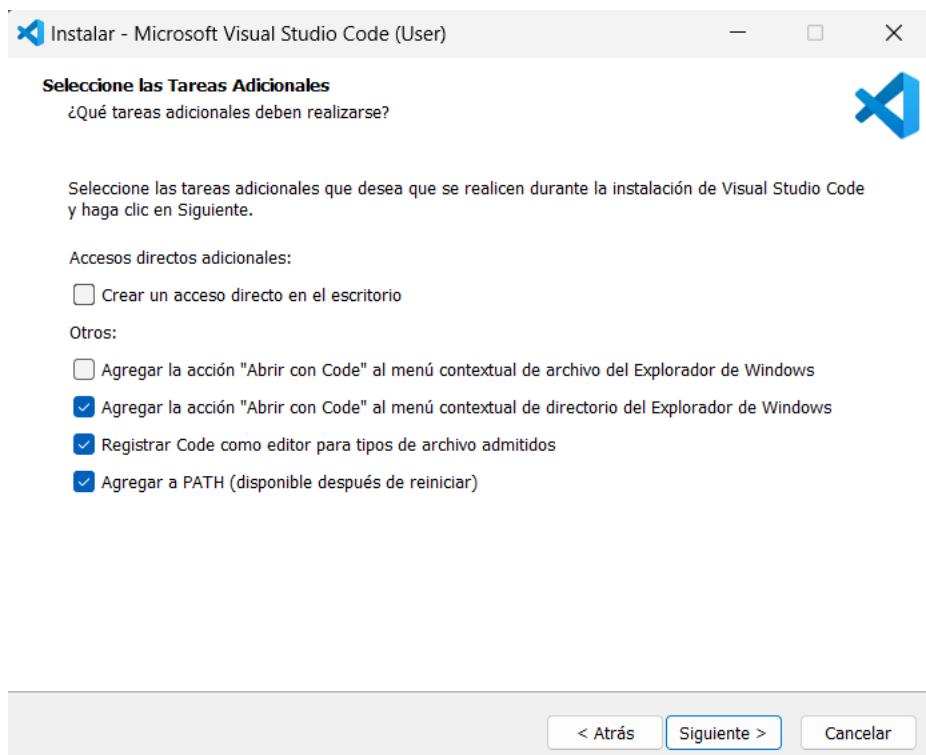
Se nos descargará automáticamente, cuando ya se haya bajado, lo ejecutaremos.



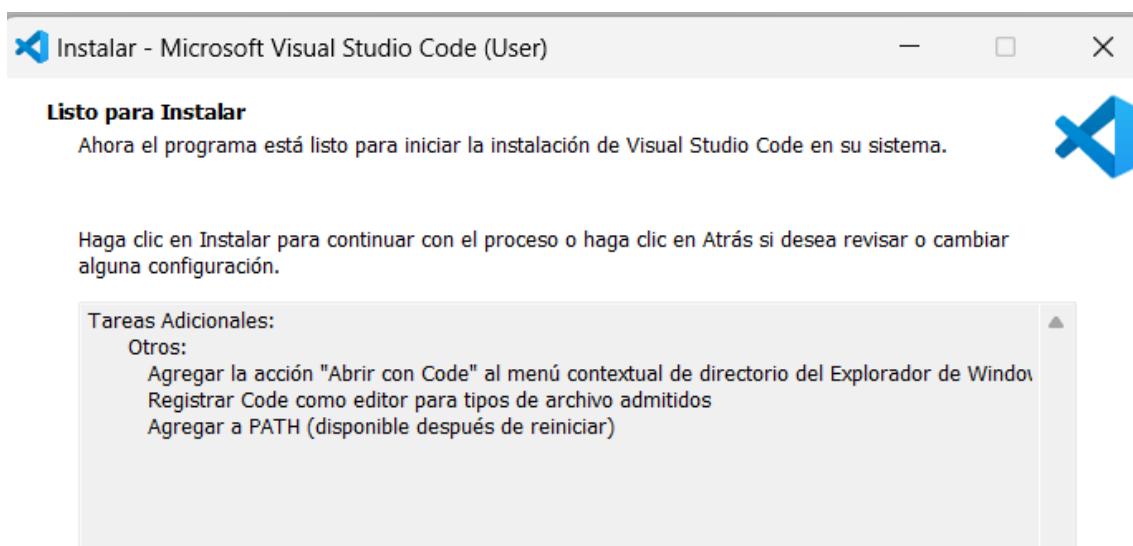
Después tendremos que darle a aceptar el acuerdo y siguiente.



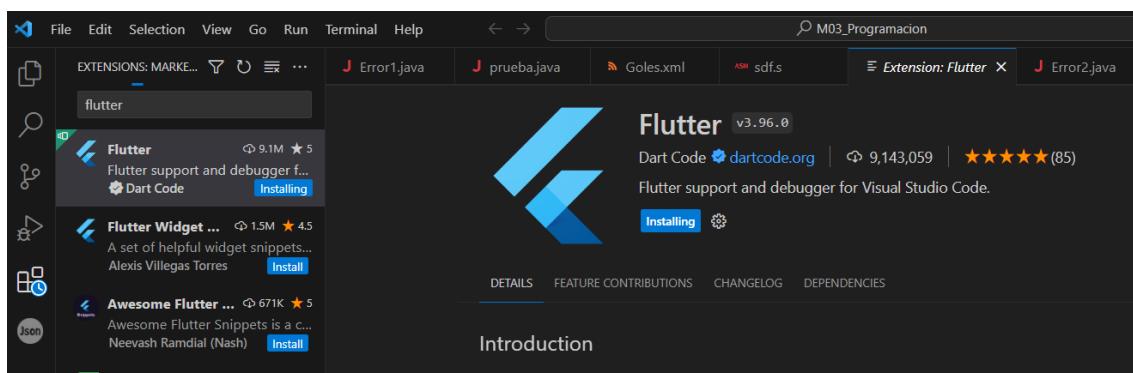
Seleccionamos las Tareas Adicionales, y le damos a siguiente.



Entonces ya podremos empezar la instalación.



Una vez acabada la instalación ya se nos abrirá el Visual, y lo primero que haremos será ir a la parte de la izquierda donde hay un logo de cuadrados, le daremos ahí y nos descargaremos el paquete de Flutter, lo mismo que hemos hecho con el Android, haremos con el Visual Studio.



Una vez instalados los paquetes nos volveremos al Flutter, y le daremos otra vez al comando:

'flutter doctor'

```
C:\Users\juanp>flutter doctor
Doctor summary (to see all details, run flutter doctor -v):
[✓] Flutter (Channel stable, 3.24.3, on Microsoft Windows [Versión 10.0.22631.4169], locale es-ES)
[✓] Windows Version (Installed version of Windows is version 10 or higher)
[!] Android toolchain - develop for Android devices (Android SDK version 35.0.0)
    ! Some Android licenses not accepted. To resolve this, run: flutter doctor --android-licenses
[✓] Chrome - develop for the web
[!] Visual Studio - develop Windows apps (Visual Studio Community 2019 16.11.36)
    X The current Visual Studio installation is incomplete.
        Please use Visual Studio Installer to complete the installation or reinstall Visual Studio.
[✓] Android Studio (version 2024.1)
[✓] Connected device (3 available)
[✓] Network resources

! Doctor found issues in 2 categories.

C:\Users\juanp>flutter doctor --android-licenses|
```

Ahora estaremos viendo que aun tenemos un problema, y para solucionarlo solo tendremos que escribir el comando siguiente:

'--android-licenses'

```
Accept? (y/N): y
All SDK package licenses accepted
```

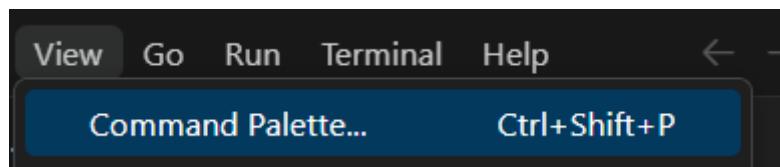
Entonces cuando se acaben de instalar las licencias, y volvemos a poner el comando de *flutter doctor* se nos tendrá que mostrar así, con todo correcto.

```
C:\Users\juanp>flutter doctor
Doctor summary (to see all details, run flutter doctor -v):
[✓] Flutter (Channel stable, 3.24.3, on Microsoft Windows [Version 10.0.22631.4169], locale es-ES)
[✓] Windows Version (Installed version of Windows is version 10 or higher)
[✓] Android toolchain - develop for Android devices (Android SDK version 35.0.0)
[✓] Chrome - develop for the web
[✓] Visual Studio - develop Windows apps (Visual Studio Community 2019 16.11.36)
[✓] Android Studio (version 2024.1)
[✓] Connected device (3 available)
[✓] Network resources

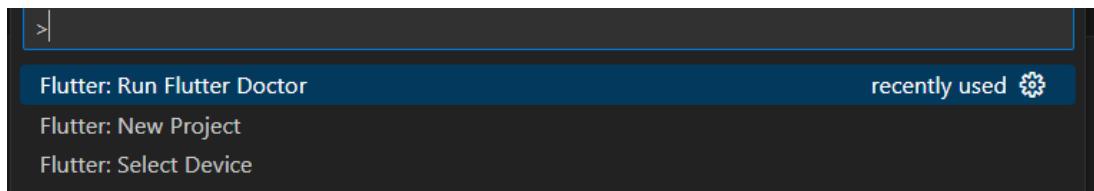
• No issues found!

C:\Users\juanp>
```

Y para asegurarnos de que esta bien instalado en el visual, para hacer eso, nos iremos a las opciones de arriba a la izquierda, nos situaremos en la opción de view, lo desplegamos y la primer de todas.



Entonces elegiremos la opción de '**Flutter: Run Flutter Doctor**' para que nos muestre que todo esta bien.



Y como podemos ver, todo está instalado correctamente.

```
[✓] Network resources
  • All expected network resources are available.

  • No issues found!
exit code 0
```

4. Nuestra primera Aplicación

4.1 Nuestra primera Aplicación con Android Studio

Si has seguido todos los pasos y los tienes correctos, asta este punto ahora será más fácil.

Ahora vamos a entrar al programa de Android Studio y vamos a crear un proyecto nuevo como se los he mostrado, y una vez creado ya tendremos el programa listo para subir y ejecutar.

The screenshot shows the Android Studio interface with the following details:

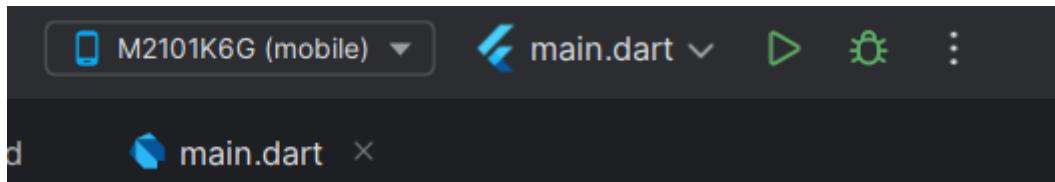
- Project Bar:** Shows the project name "p1" and the selected file "main.dart".
- File Explorer:** Lists the project structure including "lib", "test", "web", "ios", "build", "lib", "linux", "macos", and "windows".
- Code Editor:** Displays the content of "main.dart". The code initializes a Flutter application with a "MaterialApp" widget.
- Toolbars and Status Bar:** Includes standard Android Studio icons and status information like "11:29 CRLF UTF-8 2 spaces".

```
import 'package:flutter/material.dart';
void main() {
  runApp(const MyApp());
}

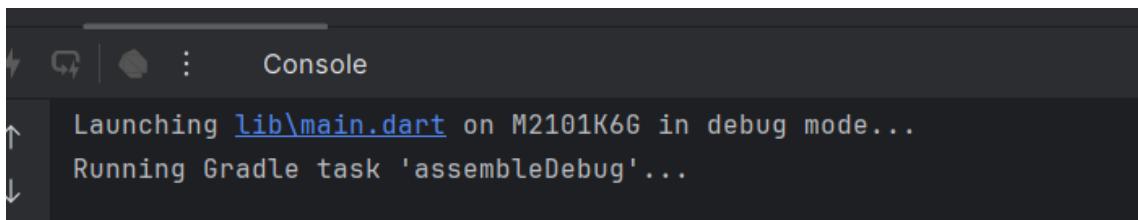
class MyApp extends StatelessWidget {
  const MyApp({super.key});

  // This widget is the root of your application.
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      debugShowCheckedModeBanner: false, //ESTO ES PARA ESCONDER LA ETIQUETA DE DEBUG
      title: 'Flutter Demo',
      theme: ThemeData(
        // This is the theme of your application.
        //
        // TRY THIS: Try running your application with "flutter run". You'll see
        // the application has a purple toolbar. Then, without quitting the app,
        // try changing the seedColor in the colorScheme below to Colors.green
        // and then invoke "hot reload" (save your changes or press the "hot
        // reload" button in a Flutter-supported IDE, or press "r" if you used
        // the command line to start the app).
        //
        // Notice that the counter didn't reset back to zero; the application
        // state is not lost during the reload. To reset the state, use hot
        // restart instead.
        //
      ),
    );
}
```

Para conectarse, simplemente le vamos a poner un cable a nuestro móvil des de nuestro portátil y cuando encuentre el móvil, arriba, se empezará a instalar todo, y ya podremos utilizar nuestra aplicación.



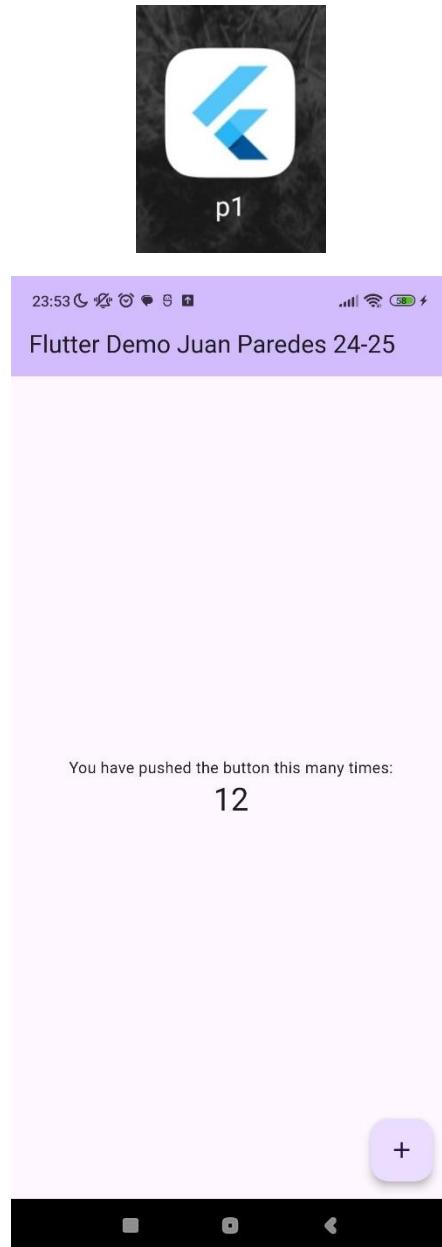
Ahora una vez que hemos verificado que esta conectado le daremos al botón de play que esta a la derecha, como podemos ver en la imagen de arriba. Y se nos empezara a instalarlos y descargarnos el programa.



Tendréis que estar atentos a vuestros móviles porque os saldrá un mensaje de instalar, y si no presiona para que se instale, tendréis que volver a darle al play.

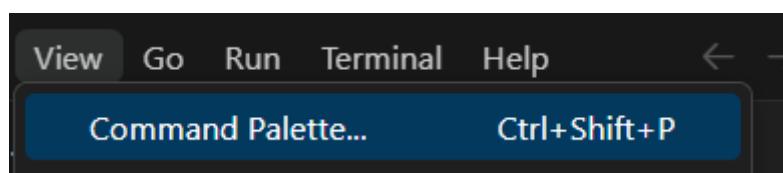


Entonces se os pondrá el programa en la pantalla de menú de vuestro móvil y ya podréis ejecutarlo cuando queráis.

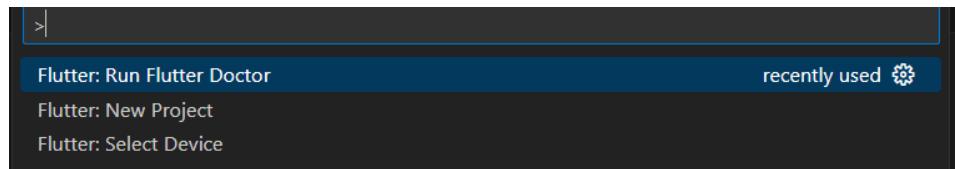


4.2 Mi primera Aplicación en Visual Studio Code

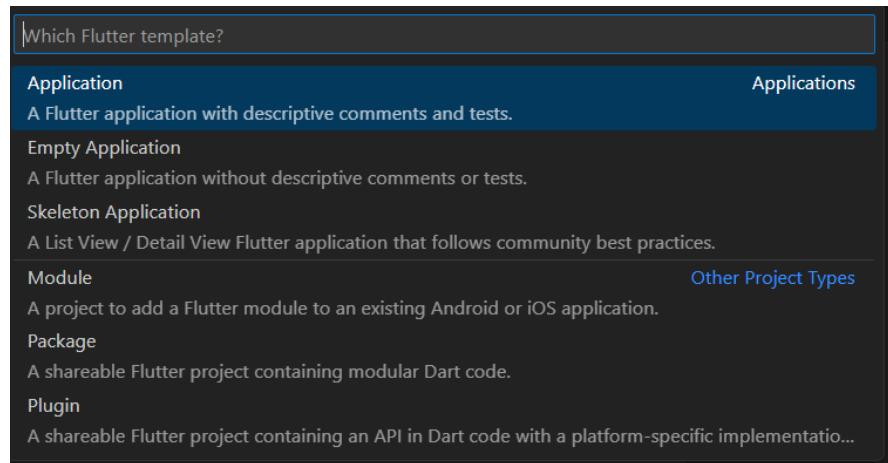
Con el Visual Studio Code no es muy diferente del de Android, lo primero que haremos será abrir el Visual, una vez dentro nos iremos arriba a la izquierda, seleccionaremos la opción de view, y la primera opción.



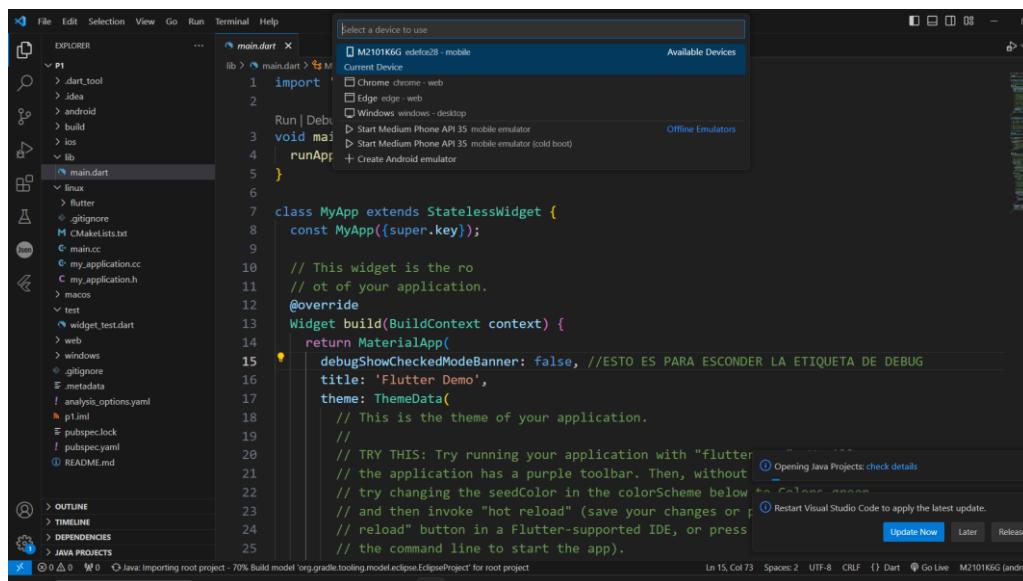
Una vez hecho eso, vamos a seleccionar la opción de nuevo proyecto con Flutter.



Y después seleccionaremos la opción de aplicación.



Y ya se nos mostrara todo el código.



Y lo mismo, que, con el punto anterior, nos conectaremos con el móvil, se nos conectará automáticamente con el móvil y ya podremos instalarnos el programa.

Y si queréis todo esto, os paso unos enlaces a mi GitHub para que lo podáis ver con mas detalle.

Enlace del código en Android Studio:

https://github.com/teropod234/AndroidStudio_flutter

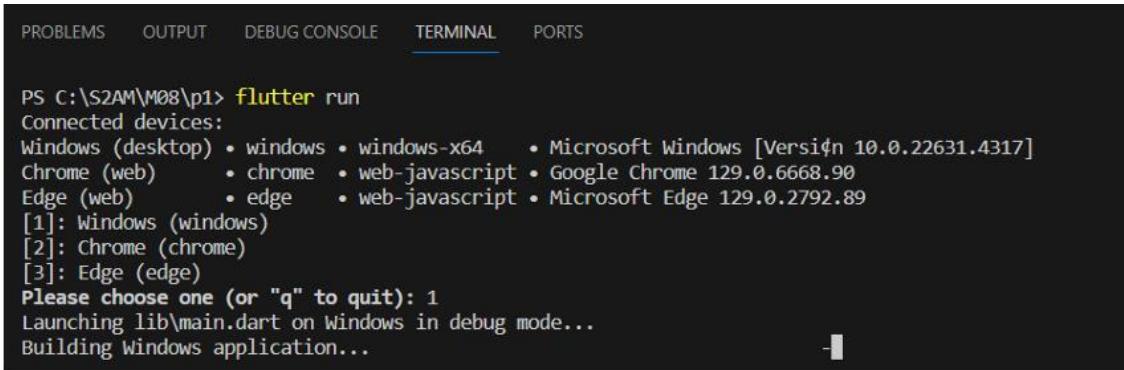
Enlace del código en Visual Studio Code:

https://github.com/teropod234/VisualStudioCode_flutter

5. Crear una aplicación por Windows con Flutter

Nos iremos al **visual studio**, y le daremos a la terminal y escribiremos *flutter run*.

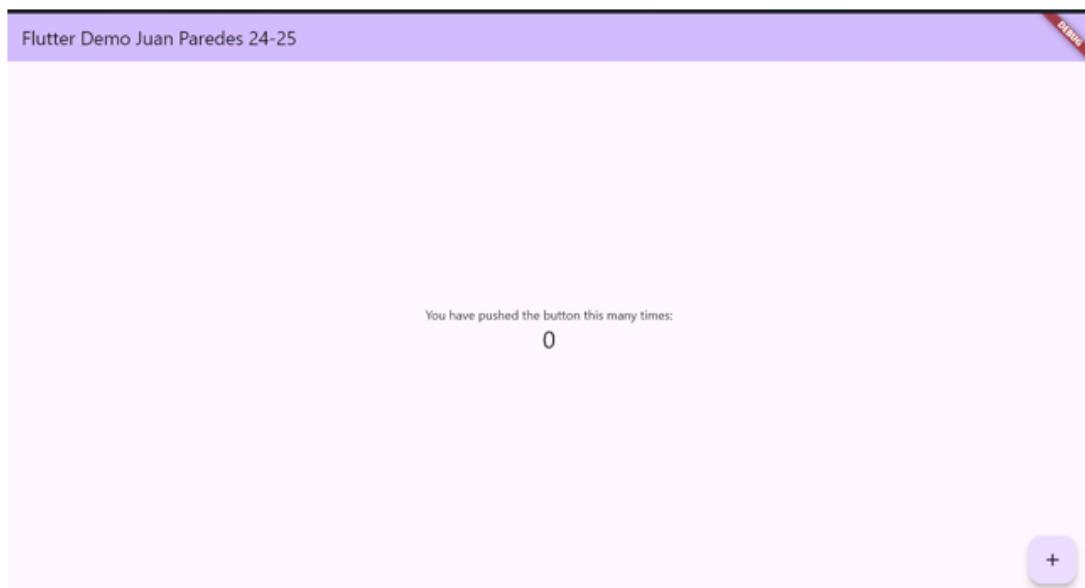
Después nos pedirá que seleccionamos una de las opciones para abrir la aplicación, y seleccionaremos la primera que es Windows, y empezara a cargar para abrir.



```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

PS C:\S2AM\M08\p1> flutter run
Connected devices:
Windows (desktop) • windows • windows-x64      • Microsoft Windows [Versión 10.0.22631.4317]
Chrome (web)       • chrome   • web-javascript • Google Chrome 129.0.6668.90
Edge (web)         • edge     • web-javascript • Microsoft Edge 129.0.2792.89
[1]: Windows (windows)
[2]: Chrome (chrome)
[3]: Edge (edge)
Please choose one (or "q" to quit): 1
Launching lib/main.dart on Windows in debug mode...
Building Windows application...
```

Cuando acabe de cargar, se nos mostrara la aplicación para empezar.

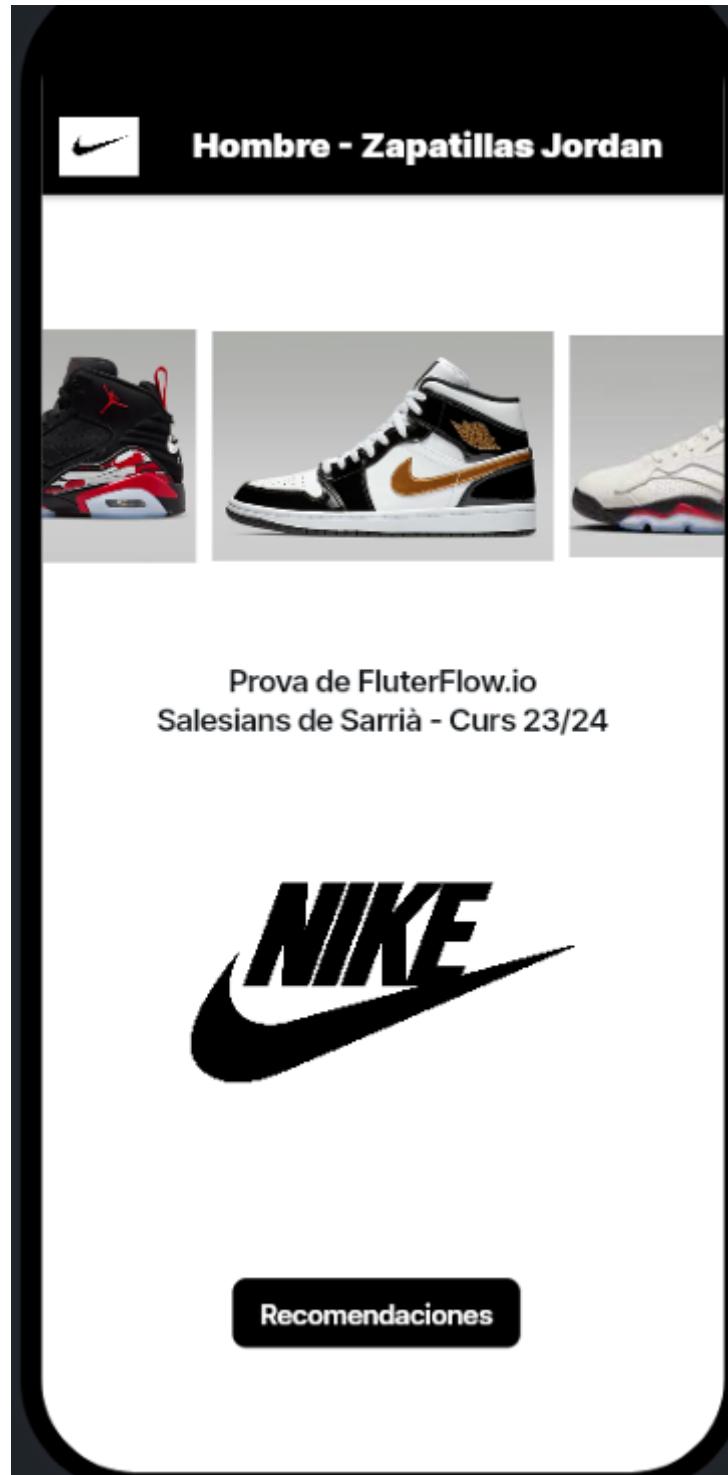


6. Crear y entender una UI con FlutterFlow.io

Mi diseño FlutterFlow.io:

[Home](#)

Si le das al logo se te abre una pestaña nueva donde te explico un poco sobre la historia de Nike, también puedes scrollar para abajo.



Historia de Nike

Si le das al botón de arriba a la izquierda, te volverá al menú. Y por último, si le das al botón que estará abajo del home, que pone 'Recomendaciones', te llevará a una pestaña donde te muestro varias tipos de marcas de Nike que te podrían gustar, donde también puedes scrollar para abajo.



Recomendaciones

Y pasa lo mismo con la pantalla de la historia de Nike, si le das al botón de arriba a la izquierda te vuelve al menú o home y también puedes desplazar la pantalla para abajo donde encontraras mas información.

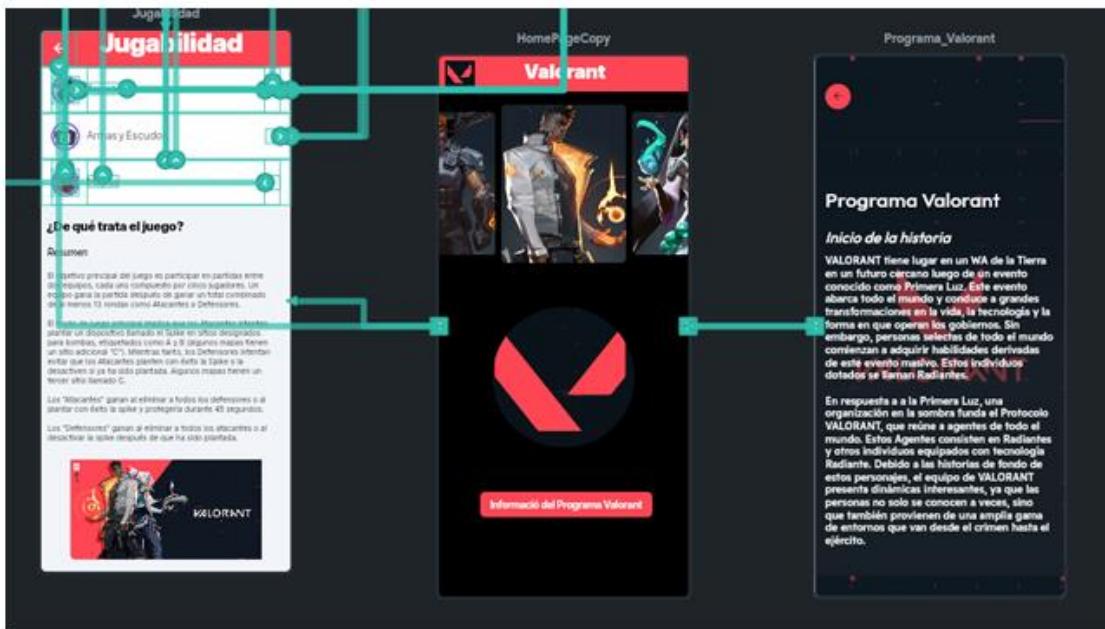


6.1 Mi FlutterFlow.io personalizado

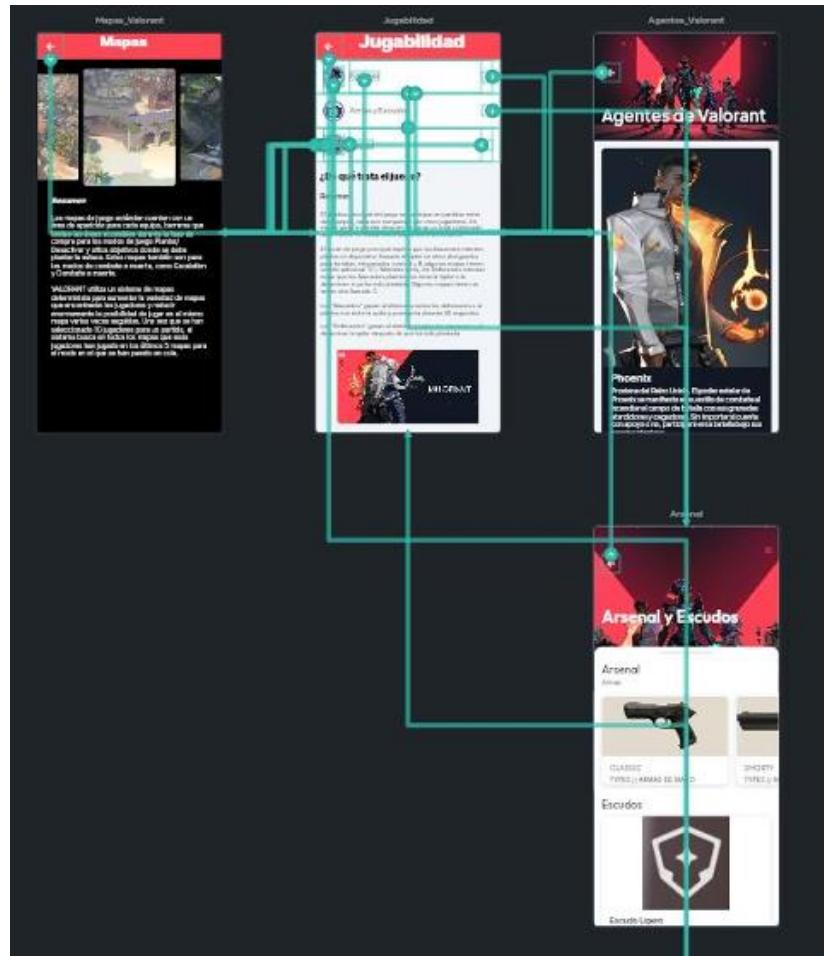
Para hacer mi flutterflow personalizado, lo que he hecho ha sido crear;

Mi pantalla Home tiene un carrousel de imágenes de unos agentes que puedes arrastrar por los lados. También si le das al logo que hay arriba a la izquierda, se te hará en grande la pantalla y la podrás ver en grande. El logo del centro es un enlace a otra pantalla que te

mostrará la jugabilidad del juego. Y el botón de abajo te mostrará la pantalla de la historia del Valorant.



Dentro de la página de Jugabilidad tengo una lista para una pantalla de agentes, donde si desplazas para abajo podrás ver varios contenedores con sus imágenes y textos, y si le das al botón de arriba a la izquierda, te vuelves a la página de jugabilidad. Después tenemos la opción de armas y escudos, donde tenemos una lista en horizontal donde se muestran las armas y abajo tenemos los escudos, y lo mismo con el botón de arriba a la izquierda nos volverá a la página de antes, y por último tenemos la opción de mapas, que es simplemente un carrousel de fotos y texto abajo, y el botón que te lleva a la página de jugabilidad, y en jugabilidad tenemos una lista y abajo texto con una imagen, con un botón que te devuelve a la pantalla de home.



7. Formularios

7.1 Formularios de Ejemplo

En esta práctica vamos a tener que hacer formularios, como podemos ver, he juntado toda la práctica en un solo proyecto. Como podemos observar tenemos 5 formularios, la primera opción son los formularios sencillos, y los 4 siguientes son los que tiene una temática en concreto.

Flutter Form Builder

- Formularis Exemple
- Form A
- Form B
- Form C
- Form D

Nos mostrara una pantalla donde podremos escribir texto que luego podremos ver en las opciones donde esta el cuadrado de abajo, habrá unas opciones donde podremos ver varias opciones.

← Formularis

Recuperar el valor d'un camp de text

BÚSQUEDA

Tr

Aquí podemos hay varias opciones, y cada uno hace una diferente.

Choose your option

SimpleDialog

AlertDialog

SnackBar

modalBottomSheet

Por ejemplo, la primera opción se nos mostrara una pantalla pequeña con lo que hemos escrito:

SimpleDialog

Hola, soy Juan

En la siguiente opción que es el AlertDialog se nos mostrara una pantalla con un botón para volver:

Texto escrito

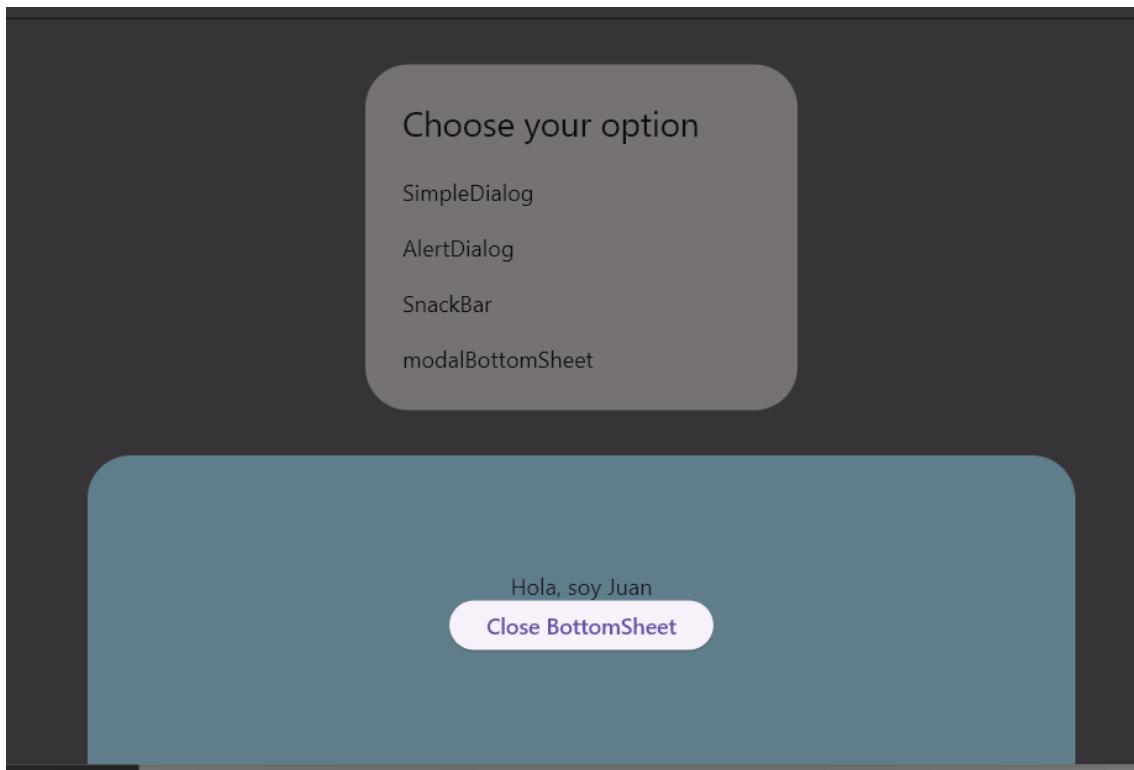
Hola, soy Juan

Volver

En la siguiente se nos mostrara una pantalla emergente des de abajo con el texto que hemos escrito previamente:



Y la última opción es otra pantalla que sale emergente des de abajo y hay otro botón donde puedes cerrar la pantalla emergente:



7.2 Formularios en FormBuilder

7.2.1 Formulario A

Ahora están los formularios utilizando el FormBuilder, en primera opción tenemos el Formulario A, donde tenemos un texto como título, y varias opciones y cuando hayas seleccionado todo, y la hayas dado al botón de submit se te mostrarán todas las opciones seleccionadas y escritas:

← Salesians Sarrià 24/25

Driving Form

Form example

Please provide the speed of vehicle?

Please provide the speed of vehicle

- above 40km/h
- below 40km/h
- 0km/h

Enter remarks:

Enter your remarks

Please provide the high speed of vehicle?

Select Option

Please provide the speed of vehicle past 1 hour?

Please select one or more option given below

- 20km/h
- 30km/h
- 40km/h
- 50km/h



Resumen de la Solicitud

Velocidad: below 40km/h

Opción: Hola

Rango de Velocidad: Medium

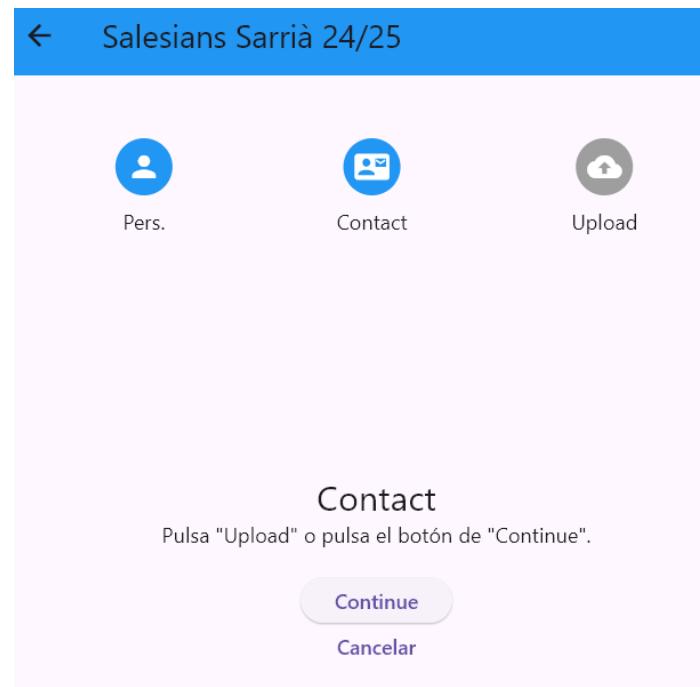
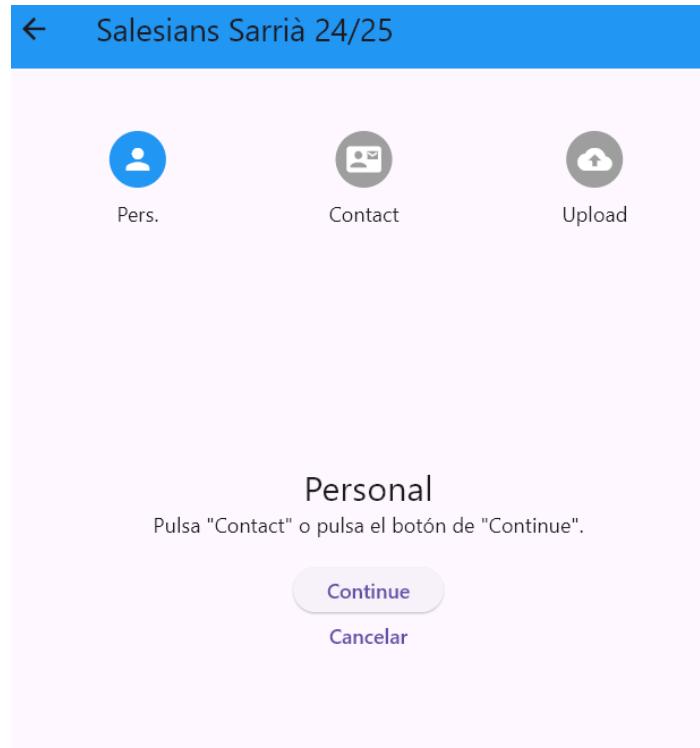
Múltiple Velocidad: [30km/h, 40km/h]

OK

7.2.2 Formulario B

En el formulario B es como un login en 3 pasos para verificar que hayas aceptado y leído como los términos, es un formato Wizzard, pero no necesitas hacer nada, simplemente tienes dos botones de continue y cancel, donde el continue es para continuar y el cancel es para tirar atrás, y por ultimo tienes varios textos que llenar para la información del email, y el mismo

botón de continue te mostrara la información que hayas rellenado y si le das a los iconos de arriba te lleva a la condición.



Pers. Contact Upload

Email

Email: ejemplo@gamil.com

Address: Ejemplo

Mobile No: 8888888888

Continue Cancelar

Resumen de Envío

Email: ejemplo@gamil.com

Address: Ejemplo

Mobile No: 8888888888

OK

7.2.3 Formulario C

En el formulario C tenemos una opción de elegir uno de las opciones, después hay para aceptar los términos, otro para escribir y que no se pase de 15 caracteres, para seleccionar un país y seleccionar una de las opciones, y lo mismo, al darle el botón de submit se nos muestra todas las respuestas y elecciones que has elegido:

← Salesians Sarrià 24/25

Choice chips:

F Flutter A Android C Chrome OS

I Accept the terms and conditions



Nombre Completo

0/15

País

Mejor lenguaje

- Option 1
- Option 2
- Option 3
- Option 4



Resumen de la Solicitud

Lenguaje elegido: Flutter
Nombre completo: Juan Paredse
País: Spain
Mejor lenguaje: Option 2
Términos aceptados: true

OK

7.2.4 Formulario D

Y por último en el último formulario tenemos para escribir un país y que se autocomplete, para seleccionar una fecha, el rango de fechas, la hora y una de selección múltiple, y por último tenemos el botón de submit que hace los mismo que los otros:

← Salesians Sarrià 24/25

Atrás

Countries

Select Date 

Date Range 

Select Time 

The language of my people

 HTML  CSS  Java  Dart  TypeScript

 Angular



Resumen de la Solicitud

País: Spain

Día: 2024-11-13

Rango de Fechas: 2024-11-21 a 2024-11-22

Hora: No seleccionado

Lenguajes: Dart, CSS

OK

Aquí podrás ver y ejecutar el código de los formularios:

<https://github.com/teropod24/Prova.git>

8 APIs

8.1 Introducción

Una API (Interfaz de Programación de Aplicaciones, por sus siglas en inglés: **Application Programming Interface**) es un conjunto de reglas, protocolos y herramientas que permite a

diferentes aplicaciones de software comunicarse entre sí. En esencia, una API actúa como un puente que conecta sistemas o servicios, permitiendo que intercambien datos y funcionalidades de manera estructurada y controlada.

Por ejemplo, cuando usas una aplicación de clima en tu teléfono, esta utiliza una API para obtener datos del servicio meteorológico y mostrarte la temperatura o el pronóstico. Las APIs son fundamentales en el desarrollo de software moderno porque simplifican la integración de diferentes sistemas, aceleran el desarrollo y permiten a los desarrolladores reutilizar funcionalidades existentes en lugar de crearlas desde cero.

En un trabajo introductorio, podrías mencionar los siguientes puntos clave:

1. **Propósito de una API:** Facilitar la comunicación entre aplicaciones, sistemas y servicios.
2. **Funcionamiento general:** Una aplicación envía una solicitud a través de la API y recibe una respuesta en un formato estándar, como JSON o XML.
3. **Ejemplos comunes:** Integración con redes sociales (Twitter, Facebook), mapas (Google Maps API), pagos (Stripe, PayPal) o servicios en la nube.
4. **Importancia:** Las APIs son esenciales en el ecosistema digital actual, promoviendo interoperabilidad, innovación y escalabilidad.



8.2 Mi API personalizada

8.2.1 Main

En esta práctica lo que tenemos que hacer es buscar un api cualquiera, donde nos podemos descargar el jsno o podemos utilizar directamente la url de la pagina web o la puedes creara tu mismo, en mi caso me he decantado por utilizar un api existente de series.

Aquí será para descargar los paquetes necesarios de Dart o llamar a otros archivos Dart utilizados en la aplicación. Estos paquetes incluyen los relacionados con Flutter (material.dart), temporizadores (async), y los archivos personalizados como el modelo de series, la lista de series y el formulario para agregar nuevas series.

```
main.dart -> my_app
import 'package:flutter/material.dart';
import 'dart:async';
import 'Series_model.dart';
import 'Series_list.dart';
import 'new_Series_form.dart';
//
```

Aquí empieza el código principal, donde se define el punto de entrada de la aplicación. Se utiliza la función main () para ejecutar el widget principal llamado MyApp.

```
Run | Debug | Profile
void main() => runApp(const MyApp());
```

En esta parte se configura el widget principal de la aplicación, llamado MyApp. Aquí establecemos:

- El título de la aplicación.
- El tema oscuro para toda la interfaz.
- La pantalla inicial (MyHomePage).

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

  @override
  Widget build(BuildContext context) {
    // Configuración del MaterialApp con un tema oscuro y desactivación del banner de depuración.
    return MaterialApp(
      debugShowCheckedModeBanner: false,
      title: 'My fav Series',
      theme: ThemeData(brightness: Brightness.dark),
      home: const MyHomePage(
        title: 'My fav Series',
      ), // MyHomePage
    ); // MaterialApp
  }
}
```

Aquí se define el widget MyHomePage, que será la pantalla principal de la aplicación. Este widget es un StatefulWidget, lo que significa que su estado puede cambiar durante la ejecución.

```
class MyHomePage extends StatefulWidget {
  final String title;
  const MyHomePage({super.key, required this.title});

  @override
  // Declaración del estado asociado a este widget.
  // El uso de "library_private_types_in_public_api" está silenciado aquí.
  _MyHomePageState createState() => _MyHomePageState();
}
```

En esta parte, se configura el estado para MyHomePage.

1. **Lista inicial:** Contiene las series favoritas predeterminadas.
2. **Formulario para agregar series:** La función `_showNewDigimonForm` abre una nueva pantalla para que el usuario pueda ingresar una nueva serie. Si se envía, la serie se agrega a la lista y se actualiza la interfaz.

```
class _MyHomePageState extends State<MyHomePage> {
  // Lista inicial de series favoritas con títulos y un identificador.
  List<Series> initialSeries =
  [
    Series('Juego de Tronos', '63741002e2c75d8744f80a50'),
    Series('Breaking Bad', '63741002e2c75d8744f80a51'),
    Series('Stranger Things', '63741002e2c75d8744f80a52')
  ];

  // Método para mostrar el formulario de nueva serie utilizando navegación.
  Future _showNewDigimonForm() async {
    Series? newDigimon = await Navigator.of(context).push(MaterialPageRoute(builder: (BuildContext context) {
      return const AddSeriesFormPage();
    })); // MaterialPageRoute
    // Si se devuelve una nueva serie del formulario, se añade a la lista.
    if(newDigimon != null){
      initialSeries.add(newDigimon);
      // Se actualiza el estado para reflejar los cambios en la UI.
      setState(() {});
    }
  }
}
```

Por último, aquí se define la estructura visual de la aplicación.

- **Barra superior:** Muestra el título y un botón para agregar series.
- **Cuerpo:** Contenedor con un fondo personalizado que muestra la lista de series centrada en la pantalla.

```
@override
Widget build(BuildContext context) {
  var key = GlobalKey<ScaffoldState>();
  return Scaffold(
    key: key,
    // Barra superior con título centrado y botón para agregar series.
    appBar: AppBar(
      title: Text(widget.title, style: const TextStyle(color: Colors.white)),
      centerTitle: true,
      backgroundColor: const Color(0xFF0B479E),
      actions: <Widget>[
        IconButton(
          icon: const Icon(Icons.add),
          onPressed: _showNewDigimonForm,
        ), // IconButton
      ], // <Widget>[]
    ), // AppBar
    body: Container(
      // Fondo de la aplicación con un color personalizado y lista de series en el centro.
      color: const Color.fromRGBO(255, 88, 111, 137),
      child: Center(
        child: DigimonList(initialSeries),
      ), // Center // Container
    ); // Scaffold
  }
}
```

8.2.2 Añadir una nueva serie

La clase AddSeriesFormPage representa una pantalla para agregar una nueva serie. Es un StatefulWidget, ya que su estado puede cambiar cuando el usuario interactúa con la aplicación.

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

  @override
  _AddDigimonFormPageState createState() => _AddDigimonFormPageState();
}
```

En el estado _AddDigimonFormPageState, se configura:

1. **Controlador de texto:** Para capturar el nombre de la serie que el usuario ingrese.
2. **Indicador de carga:** Un bool (isLoading) para mostrar un indicador mientras se procesa la solicitud.
3. **Lógica de envío:** La función submitPup verifica si el campo está vacío y busca el ID relacionado al nombre de la serie en la API. Si encuentra un ID, lo devuelve al widget anterior; de lo contrario, muestra un mensaje de error.

```
class _AddDigimonFormPageState extends State<AddSeriesFormPage> {
  TextEditingController nameController = TextEditingController();
  bool isLoading = false;

  void submitPup(BuildContext context) async {
    if (nameController.text.isEmpty) {
      ScaffoldMessenger.of(context).showSnackBar(const SnackBar(
        backgroundColor: Colors.redAccent,
        content: Text('You forgot to insert the series name'),
      )); // SnackBar
    } else {
      setState(() {
        isLoading = true; // Mostrar un indicador de carga
      });

      try {
        // Buscar el ID relacionado al nombre
        String? idTitle = await fetchIdByName(nameController.text);
        if (idTitle != null) {
          var newDigimon = Series(nameController.text, idTitle);
          Navigator.of(context).pop(newDigimon); // Regresar la nueva serie
        } else {
          ScaffoldMessenger.of(context).showSnackBar(const SnackBar(
            backgroundColor: Colors.redAccent,
            content: Text('Series not found. Please check the name.'),
          )); // SnackBar
        }
      } catch (error) {
        ScaffoldMessenger.of(context).showSnackBar(SnackBar(
          backgroundColor: Colors.redAccent,
          content: Text('Error: $error'),
        )); // SnackBar
      } finally {
        setState(() {
          isLoading = false; // Ocultar el indicador de carga
        });
      }
    }
  }
}
```

La función fetchIdByName realiza una solicitud a una API para obtener una lista de series y buscar una coincidencia exacta con el nombre ingresado. Si encuentra una coincidencia, devuelve el ID relacionado; si no, lanza un error.

```
Future<String?> fetchIdByName(String name) async {
    HttpClient http = HttpClient();
    try {
        var uri = Uri.https('peticiones.online', '/api/series');
        var request = await http.getUrl(uri);
        var response = await request.close();
        var responseBody = await response.transform(utf8.decoder).join();

        List<dynamic> data = json.decode(responseBody);
        // Buscar el ID por el nombre ingresado
        var matchedSeries = data.firstWhere(
            (item) => item["title"] as String).toLowerCase() == name.toLowerCase(),
           orElse: () => null,
        );

        return matchedSeries?["_id"];
    } catch (exception) {
        throw 'Failed to fetch data: $exception';
    }
}
```

En esta parte se construye la interfaz de usuario.

- **Barra superior:** Muestra el título de la página.
- **Cuerpo:** Contiene:
 1. Un campo de texto para ingresar el nombre de la serie.
 2. Un botón para enviar los datos, que muestra un indicador de carga mientras se procesa

```
@override
Widget build(BuildContext context) {
    return Scaffold(
        appBar: AppBar(
            title: const Text('Add a new serie'),
            backgroundColor: const Color(0xFF0B479E),
        ), // AppBar
        body: Container(
            color: const Color(0xFFABCDEF),
            child: Padding(
                padding: const EdgeInsets.symmetric(vertical: 8.0, horizontal: 32.0),
                child: Column(children: [
                    Padding(
                        padding: const EdgeInsets.only(bottom: 8.0),
                        child: TextField(
                            controller: nameController,
                            style: const TextStyle(color: Colors.black),
                            decoration: const InputDecoration(
                                labelText: 'Series Name',
                                labelStyle: TextStyle(color: Colors.black),
                            ), // InputDecoration
                        ), // TextField
                    ), // Padding
                    Padding(
                        padding: const EdgeInsets.all(16.0),
                        child: Builder(
                            builder: (context) {
                                return ElevatedButton(
                                    onPressed: isLoading ? null : () => submitPup(context),
                                    child: isLoading
                                        ? const CircularProgressIndicator(
                                            color: Colors.white,
                                        ) // CircularProgressIndicator
                                        : const Text('Submit Series'),
                                ); // ElevatedButton
                            },
                        ),
                    ),
                ],
            ),
        ),
    );
}
```

8.2.3 Detalles de las cartas

El widget DigimonCard representa la tarjeta visual para mostrar la información básica de una serie y permite acceder a más detalles al tocarla. Es un StatefulWidget porque su estado (como las imágenes) cambia dinámicamente

```
class DigimonCard extends StatefulWidget {
    final Series digimon;

    const DigimonCard(this.digimon, {super.key});

    @override
    // ignore: library_private_types_in_public_api, no_logic_in_create_state
    _DigimonCardState createState() => _DigimonCardState(digimon);
}
```

El estado `_DigimonCardState` se encarga de:

1. Almacenar la serie asociada y la URL de su imagen.
2. Inicializar el estado y cargar la imagen de la serie al inicio.
3. Actualizar dinámicamente la interfaz según el estado de la tarjeta

```
class _DigimonCardState extends State<DigimonCard> {
    Series digimon;
    String? renderUrl;

    _DigimonCardState(this.digimon);

    @override
    void initState() {
        super.initState();
        renderDigimonPic();
    }
}
```

Se utiliza un Hero para animar la transición de la imagen al detalle, combinando:

1. **placeholder:** Una imagen circular genérica que aparece mientras se carga la imagen real.
2. **digimonAvatar:** Muestra la imagen real una vez cargada.
Se usa AnimatedCrossFade para cambiar entre el placeholder y la imagen real con un efecto suave

```
@override
void initState() {
    super.initState();
    renderDigimonPic();
}

Widget get digimonImage {
    var digimonAvatar = Hero(
        tag: digimon,
        child: Container(
            width: 100.0,
            height: 100.0,
            decoration: BoxDecoration(
                shape: BoxShape.circle,
                image: DecorationImage(
                    fit: BoxFit.cover,
                    image: NetworkImage(renderUrl ?? ''),
                ), // DecorationImage
            ), // BoxDecoration
        ), // Container
    ); // Hero

    var placeholder = Container(
        width: 100.0,
        height: 100.0,
        decoration: const BoxDecoration(
            shape: BoxShape.circle,
            gradient: LinearGradient(
                begin: Alignment.topLeft,
                end: Alignment.bottomRight,
                colors: [
                    Colors.black54,
                    Colors.black,
                    Color.fromARGB(255, 84, 110, 122),
                ],
            ),
        ),
    );
}
```

La función renderDigimonPic solicita la URL de la imagen desde la clase Series y actualiza el estado de la tarjeta

```
void renderDigimonPic() async {
    await digimon.getImageUrl();
    if (mounted) {
        setState(() {
            renderUrl = digimon.imageUrl;
        });
    }
}
```

El widget digimonCard muestra la información básica:

1. **Título de la serie.**
2. **Canal asociado.**
3. **Puntuación.**

Se usa un diseño con bordes redondeados y un color de fondo suave

```
Widget get digimonCard {
    return Positioned(
        right: 0.0,
        child: SizedBox(
            width: 290,
            height: 115,
            child: Card(
                shape: RoundedRectangleBorder(
                    borderRadius: BorderRadius.circular(15.0),
                ), // RoundedRectangleBorder
                color: const Color(0xFFFF8F8F),
                child: Padding(
                    padding: const EdgeInsets.only(top: 8, bottom: 8, left: 64),
                    child: Column(
                        mainAxisAlignment: MainAxisAlignment.start,
                        mainAxisSize: MainAxisSize.spaceAround,
                        children: <Widget>[
                            // Mostrar el título de la serie
                            Text(
                                widget.digimon.title ?? 'Sin título',
                                style: const TextStyle(
                                    color: const Color(0xFF000600),
                                    fontSize: 20.0,
                                    fontWeight: FontWeight.bold,
                                ), // TextStyle
                            ), // Text
                            // Mostrar el canal
                            Text(
                                ' Media: ${widget.digimon.channel ?? 'Sin canal'}',
                                style: const TextStyle(
                                    color: const Color(0xFF555555),
                                    fontSize: 14.0,
                                ), // TextStyle
                            ), // Text
                        ],
                    ),
                ),
            ),
        ),
    );
}
```

La función showDigimonDetailPage navega hacia la página de detalles y asegura que cualquier cambio en los datos se refleje al regresar

```
void showDigimonDetailPage() {
    Navigator.of(context)
        .push(
            MaterialPageRoute(builder: (context) => DigimonDetailPage(digimon)))
        .then((_) {
    setState(() {
        // Esto asegura que la tarjeta se reconstruya con los nuevos valores.
    });
});
}
```

El método build organiza los elementos visuales:

1. Tarjeta de información.
2. Imagen de la serie, colocada sobre la tarjeta.
3. Funcionalidad de toque para abrir detalles

```
@override
Widget build(BuildContext context) {
    return InkWell(
        onTap: () => showDigimonDetailPage(),
        child: Padding(
            padding: const EdgeInsets.symmetric(horizontal: 16.0, vertical: 8.0),
            child: SizedBox(
                height: 115.0,
                child: Stack(
                    children: <Widget>[
                        digimonCard,
                        Positioned(top: 7.5, child: digimonImage),
                    ],
                ),
            ),
        ),
    );
}
```

8.2.4 Detalles de las páginas

Esta clase gestiona la visualización dinámica de los detalles de la serie y permite al usuario actualizar la puntuación.

Propiedades principales:

- digimonAvarterSize: Tamaño del avatar de la serie.
- _sliderValue: Controla la puntuación seleccionada por el usuario

```
class _DigimonDetailPageState extends State<DigimonDetailPage> {
    final double digimonAvarterSize = 150.0;
    double _sliderValue = 10.0;
```

Slider para Ajustar la Puntuación

El widget addYourRating muestra un Slider para que el usuario ajuste la puntuación y un botón para enviarla.

- **Slider:** Permite seleccionar un valor entre 0 y 10.
- **Botón:** Actualiza la puntuación en el modelo

```
Widget get addYourRating {
  return Column(
    children: <Widget>[
      Container(
        padding: const EdgeInsets.symmetric(vertical: 16.0, horizontal: 16.0),
        child: Row(
          mainAxisAlignment: MainAxisAlignment.spaceBetween,
          children: <Widget>[
            Flexible(
              flex: 1,
              child: Slider(
                activeColor: const Color(0xFF0B479E),
                min: 0.0,
                max: 10.0,
                value: _sliderValue,
                onChanged: (newRating) {
                  setState(() {
                    _sliderValue = newRating;
                  });
                },
              ),
            ), // Slider
          ], // Flexible
          Container(
            width: 50.0,
            alignment: Alignment.center,
            child: Text(
              '${_sliderValue.toInt()}',
              style: const TextStyle(color: Colors.black, fontSize: 25.0),
            ), // Text
          ), // Container
        ], // <Widget>[]
      ), // Row
    ], // Container
    submitRatingButton,
  ); // <Widget>[]
}
```

Actualiza la puntuación seleccionada en el modelo de datos

```
void updateRating() {
  setState(() {
    widget.digimon.rating = _sliderValue.toInt();
  });
}
```

El widget digimonImage muestra la imagen de la serie. Utiliza un Hero para animar transiciones

```
Widget get digimonImage {
  return Hero(
    tag: widget.digimon,
    child: Container(
      height: digimonAvarterSize,
      width: digimonAvarterSize,
      decoration: BoxDecoration(
        shape: BoxShape.circle,
        image: DecorationImage(
          fit: BoxFit.cover,
          image: NetworkImage(widget.digimon.imageUrl ?? ""),
        ), // DecorationImage
      ), // BoxDecoration
    ), // Container
  ); // Hero
}
```

El widget rating muestra la puntuación actual de la serie de forma visual y numérica

```
Widget get rating {
  return Row(
    mainAxisAlignment: MainAxisAlignment.center,
    children: <Widget>[
      const Icon(
        Icons.star,
        size: 40.0,
        color: Colors.yellow,
      ), // Icon
      Text(
        '${widget.digimon.rating}/10',
        style: const TextStyle(color: Colors.black, fontSize: 30.0),
      ), // Text
    ], // <Widget>[]
  ); // Row
}
```

El widget digimonProfile muestra:

1. Imagen de la serie.
2. Nombre, creador, canal y fechas.
3. La puntuación actual

```
Widget get digimonProfile {
  return Container(
    padding: const EdgeInsets.symmetric(vertical: 32.0),
    decoration: const BoxDecoration(
      color: Color(0xFFABCAED),
    ), // BoxDecoration
    child: Column(
      crossAxisAlignment: CrossAxisAlignment.center,
      children: <Widget>[
        digimonImage,
        const SizedBox(height: 10),
        Text(
          widget.digimon.name,
          style: const TextStyle(color: Colors.black, fontSize: 32.0),
        ), // Text
        const SizedBox(height: 5),
        Text(
          'Created by ${widget.digimon.creator ?? "Unknown"}',
          style: const TextStyle(color: Colors.black54, fontSize: 18.0),
        ), // Text
        const SizedBox(height: 5)
      ],
    ),
  );
}
```

Construye la interfaz de la página.

- **AppBar:** Muestra el título de la serie.
- **ListView:** Contiene el perfil de la serie y el widget para agregar una puntuación

```
@override
Widget build(BuildContext context) {
  return Scaffold(
    backgroundColor: const Color(0xFFABCAED),
    appBar: AppBar(
      backgroundColor: const Color(0xFF0B479E),
      title: Text('Serie: ${widget.digimon.title ?? "Unknown"}'),
    ), // AppBar
    body: ListView(
      children: <Widget>[digimonProfile, addYourRating],
    ), // ListView
  ); // Scaffold
}
```

8.2.5 Listas de Series

En un StatelessWidget, el método build es el encargado de crear la interfaz de usuario. En este caso, llama al método privado _buildList para crear y devolver un ListView con los elementos de la lista de series. El método privado _buildList devuelve un ListView.builder, que es un widget muy eficiente para mostrar listas largas de elementos. Utiliza un constructor de elementos bajo demanda:

- **itemCount:** Número total de elementos en la lista (series.length).
- **itemBuilder:** Función que se ejecuta para crear cada elemento de la lista. Toma el contexto y el índice (int), y en este caso devuelve un widget DigimonCard para cada serie.

```
@override
Widget build(BuildContext context) {
  return _buildList(context);
}

ListView _buildList(BuildContext context) {
  return ListView.builder(
    itemCount: series.length,
    // ignore: avoid_types_as_parameter_names
    itemBuilder: (context, int) {
      return DigimonCard(series[int]);
    },
  ); // ListView.builder
```

8.2.6 Conectarse a mi api

La clase Series representa los datos de una serie, con varias propiedades opcionales como título, canal, creador, imagen, etc., y algunas propiedades requeridas como el nombre (name) y el ID de la serie (idtitle).

```
class Series {
  String? title;
  String? channel;
  String? creator;
  String? imageUrl;
  String? dates;
  final String name;
  String? idtitle;
  int rating = 10;
```

Este es un método asíncrono que se encarga de obtener la URL de la imagen de la serie, así como otros detalles relacionados a la serie desde una API externa.

1. **Chequeo de imagen existente:** Si ya existe un valor en imageUrl, simplemente se retorna y no hace nada.
2. **Solicitud HTTP:** Si no existe la imagen, se realiza una solicitud HTTP a la API (<https://peticiones.online/api/series>) para obtener los detalles de las series.
3. **Procesamiento de la respuesta:** La respuesta de la API se convierte en una lista de objetos JSON, luego se busca el objeto que coincide con el idtitle de la serie.

4. **Asignación de datos:** Si se encuentra la serie, se asignan sus detalles (título, creador, canal, fechas, imagen) a las propiedades de la clase.
5. **HttpClient:** Se utiliza para realizar una solicitud GET a la API y obtener los datos en formato JSON
6. **json.decode:** Convierte la respuesta de la API en un formato que se puede manejar en Dart
7. **firstWhere:** Busca el primer elemento que coincide con el idtitle, y si no lo encuentra, devuelve null

```
Series(this.name, this.idtitle);

Future getImageUrl() async {
  if (imageUrl != null) {
    return;
  }

  HttpClient http = HttpClient();
  try {
    title = name.toLowerCase();
    var uri = Uri.https('peticiones.online', '/api/series');
    var request = await http.getUrl(uri);
    var response = await request.close();
    var responseBody = await response.transform(utf8.decoder).join();

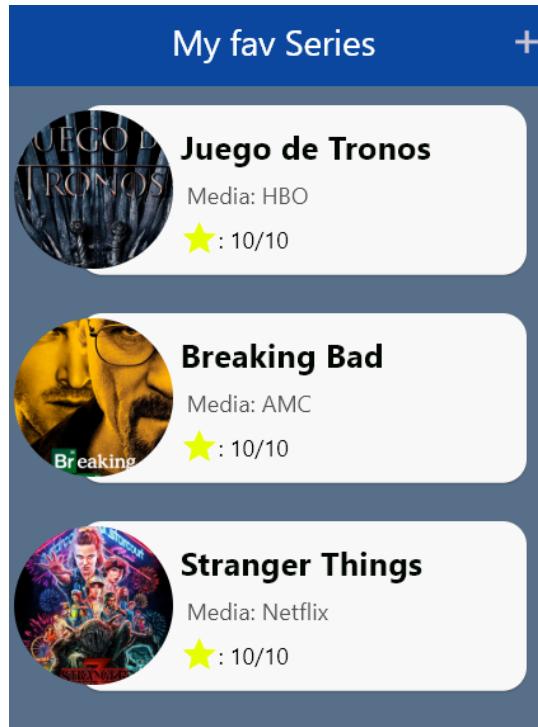
    List<dynamic> data = json.decode(responseBody);

    // Buscamos el elemento con el idtitle correspondiente.
    var matchedSeries = data.firstWhere(
      (item) => item["_id"] == idtitle,
      orElse: () => null,
    );

    if (matchedSeries != null) {
      imageUrl = matchedSeries["image"];
      title = matchedSeries["title"];
      creator = matchedSeries["creator"];
      channel = matchedSeries["channel"];
      dates = matchedSeries["dates"];
    }
    //print(levelDigimon);
  } catch (exception) {
    //print(exception);
  }
}
```

8.3 Probar mi API

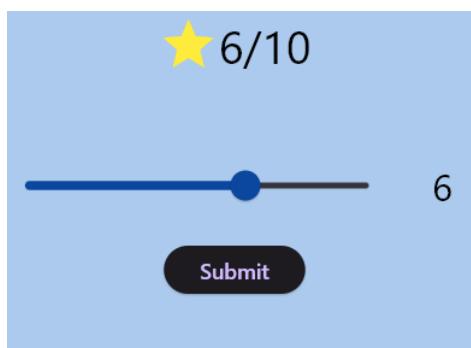
Ahora vamos a ver nuestra api mas de cerca, como podemos ver este es nuestro menú principal, donde ya están 3 series predefinidas como lo hemos preparado.



Después si nos vamos a una, como por ejemplo la primera de todos que es Juegos de Tronos, lo que podremos ver es que tiene el nombre de la serie, sus creadores, su fecha de estreno y por último donde poder ver esta serie. También puedes cambiar la valoración a tu antojo y siempre que después le des a submit, se te guardara esa nueva valoración.



Como puedes ver la valoración al cambiarla también se cambia en la información tanto des de dentro como de afuera.



Y afuera, en la página principal también se ha cambiado correctamente.



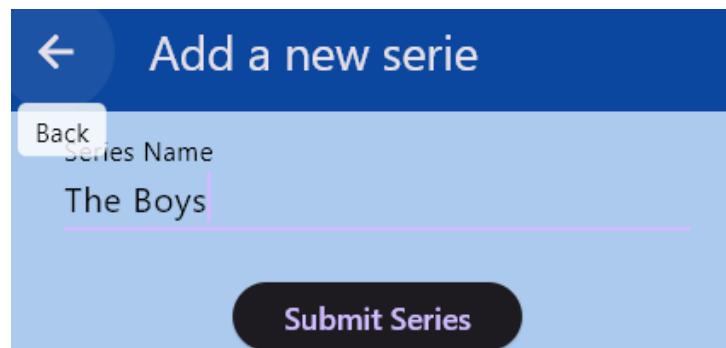
También tenemos la opción de añadir una nueva serie siempre que la api la tenga.



Si intentamos escribir una que no existe o que ya está nos aparecerá una alerta abajo diciendo que no se ha encontrado.



Entonces insertamos el nombre bien.



Y ya se nos aparecerá la nueva serie en la página principal con toda su información.



También hay un video para que veas cómo funciona:



Y el enlace al github para que puedas utilizar mi Api o aplicarla a otro api:

<https://github.com/teropod24/API.git>

9 Api para mostrar Actores

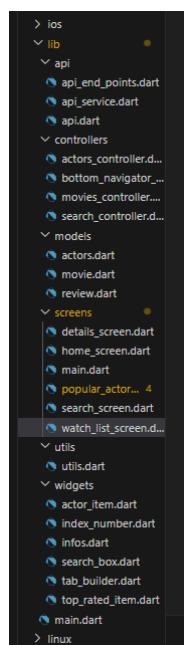
9.1 Introducción al ejercicio

Lo que primer vamos a hacer es ver que nos esta pidiendo en este enunciado;

A partir de l'exemple de Yousssof de les "Movies-App" volem fer dues activitats:

- Primera, es tracta d'estudiar bé el codi en Dart per entendre com s'estan carregant els valors de les pel·lícules a partir de la API.

En el primer parte nos dice que estudiemos bien el código que no ha pasado para poder ejercer el ejercicio a lo que nos demandan. Pues vamos a ver cómo está estructurado el código y vamos a analizarlo para entender bien que hace el código.



Una vez estudiado y analizado el código vamos a la segunda parte que nos dice:

- Un cop entès, passareu a modificar el codi per a obtenir una aplicació transformada per vosaltres:
 - Dona't d'alta a la API de TMDB. [The Movie Database API \(themoviedb.org\)](https://www.themoviedb.org)
 - Modifica la App per a que mostri una **llista d'actors** en la pantalla principal. Quan polsem a sobre d'un actor o actriu podrem veure una plana de detall amb la descripció i la seva informació relacionada.
 - AMPLIACIÓ 1: Afageix en la plana del detall de l'actor, en la part de sota, **una llista amb les pel·lícules on ha participat l'actor seleccionat**. Si es pulsa a sobre d'una pel·lícula, es veuen totes les dades que hi fan referència.
 - AMPLIACIÓ 2: Mitjançant un menú, fes que l'usuari pugui triar **escollir pel·lícules o sèries**, i ens mostri una llista amb les pel·lícules o sèries. Si després n'escollem alguna ens mostrarà totes les dades relacionades.

Com sempre cal lluir una **bona documentació funcional** que expliqui com funciona la App i quins canvis heu fet per tal d'aconseguir les modificacions proposades. Recordeu que és necessari comentar també el codi.

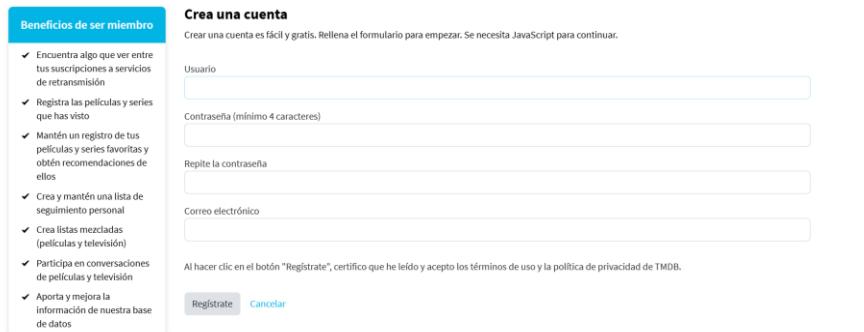
Es valorarà el fet d'afegir **un vídeo** on es mostri el comportament de la App.

Ahora nos está diciendo que nos creamos una nueva cuenta en el api de TMDB y que modifiquemos el código para que sea para una lista de actores.

9.2 Modificación del Código

9.2.1 Primer paso dar de alta

Primero vamos a acceder a la página web de '**TMDB**', y vamos a darle a unirse, después vamos a tener que llenar todos estos campos, y una vez acabado, nos llegara un correo para confirmar la verificación.



Beneficios de ser miembro

- ✓ Encuentra algo que ver entre tus suscripciones a servicios de retransmisión
- ✓ Registra las películas y series que has visto
- ✓ Mantén un registro de tus películas y series favoritas y obtén recomendaciones de ellos
- ✓ Crea y mantén una lista de seguimiento personal
- ✓ Crea listas mezcladas (películas y televisión)
- ✓ Participa en conversaciones de películas y televisión
- ✓ Aporta y mejora la información de nuestra base de datos

Crea una cuenta
Crear una cuenta es fácil y gratis. Rellena el formulario para empezar. Se necesita JavaScript para continuar.

Usuario

Contraseña (mínimo 4 caracteres)

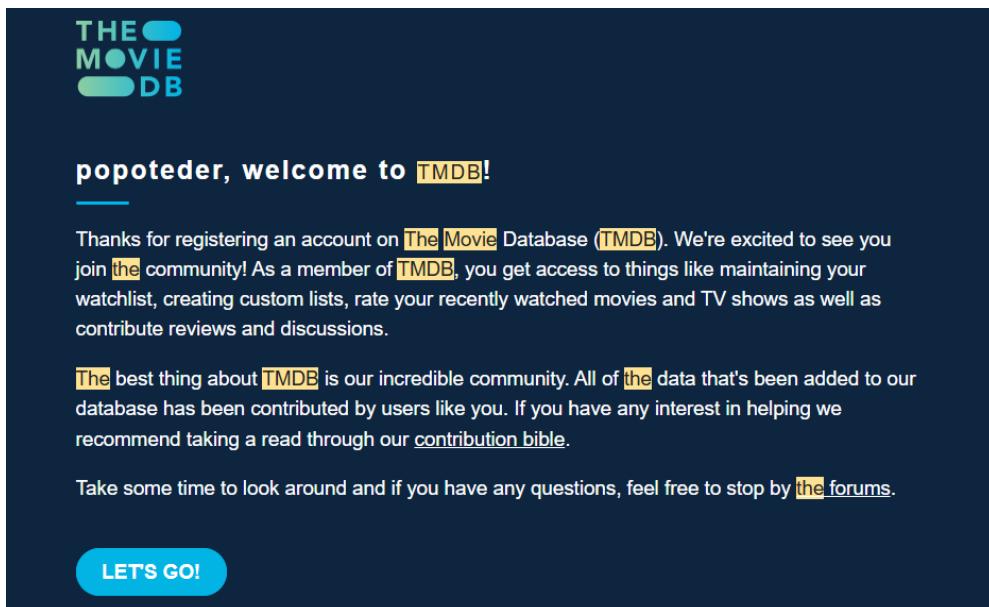
Repite la contraseña

Correo electrónico

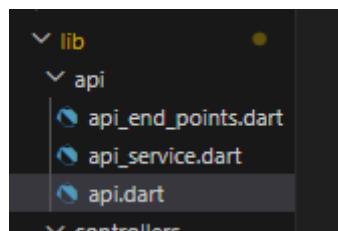
Al hacer clic en el botón "Regístrate", certifico que he leído y acepto los términos de uso y la política de privacidad de TMDB.

Regístrate [Cancelar](#)

Cuando te llegue el correo simplemente tenemos que darle a lets go y ya nos dará nuestra api Key que vamos a tener que utilizar.



Cuando la tengamos, nos iremos al archivo que se llama '**api.dart**' que esta dentro de la carpeta de '**Api**'.

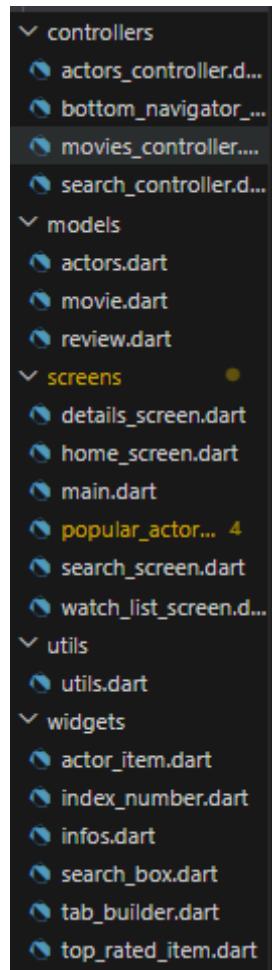


Una vez dentro lo que podemos ver que ya explicaremos más adelante es que tenemos una constante estática que se llama **apiKey**, pues ahí cambiaremos el api que esta puesta por la nuestra.

```
api > api.dart > ...
1 class Api {
2   static const baseUrl = "https://api.themoviedb.org/3/";
3   static const imageBaseUrl = "https://image.tmdb.org/t/p/original/";
4   static const apiKey = "c563dfe11b9b27c56da72a253d1bf23d";
5 }
```

9.2.2 Copiar de los archivos del Programa

Vale, una vez hecho el primer paso, vamos a empezar a modificar el código, lo primero que vamos a hacer es copiar varios archivos ya existentes en las mismas carpetas y le vamos a cambiar los nombres simplemente. Para eso te voy a decir que archivos tienes que copiar, tienes que copiar; **movies_controller**, **movies**, **details_screen**, **infos**. Tienes que copiar todos estos archivos y renombrarlos en la misma carpeta en donde este. Te preguntaras para que sirve todo esto, esto sirve para cuando queramos hacer la parte de solo actores, ya tendremos todo lo necesario, si has mirado el código, sabrás que estos son los archivos mas activos para las películas.



9.2.4 Añadir lógica a los archivos de actores

9.2.4.1 Carpeta Api

Vale, ahora que tenemos todo preparado para empezar, vamos a irnos primero a la carpeta de api, el archivo **api_end_points**:

La clase ApiEndPoints almacena **URLs o rutas de servicios API** (interfaces para obtener datos) que una aplicación puede usar para comunicarse con un servidor y obtener información.

- **Objetivo principal:** Organizar las rutas de forma clara y centralizada, para que el código sea fácil de usar y mantener.
- **Por qué es útil:** Si necesitas cambiar alguna de las rutas, solo tienes que modificar esta clase en lugar de buscar y reemplazar en todo el proyecto.

```
pi > api_end_points.dart > ...
1  class ApiEndPoints {
2    static const products = "products";
3    static const popularMovies = "movie/popular";
4    static const upcomingMovies = "movie/upcoming";
5    static const getGenreList = "genre/movie/list";
6    static const getActors = "person/popular";
7 }
```

Vale, una vez hecho esto, nos iremos al siguiente archivo para que funcione bien el api, ya tenemos nuestro endpoint, pero necesitamos algo que llame al api y que nos devuelva el dato, para eso esta el archivo que vamos a modificar ahora que es el **api_Service** y lo que vamos a hacer es añadir todas estas nuevas funciones dentro del archivo;

¿Qué hace? Obtiene una lista de actores populares desde la API. Crea una URL con la base de la API (*Api.baseUrl*), el endpoint (*ApiEndPoints.getActors*), y la clave de API (*Api.apiKey*). Realiza una solicitud HTTP con `http.get(url)`.

Si el servidor responde con éxito (código 200):

- Decodifica la respuesta JSON.
- Extrae la lista de actores desde el campo **results**.
- Convierte cada actor de JSON a un objeto Actor usando `Actor.fromJson()`.

Si no responde correctamente, lanza un error.

```
static Future<List<Actor>> getPopularActors() async {
  final url = Uri.parse(
    '${Api.baseUrl}${ApiEndPoints.getActors}?api_key=${Api.apiKey}&language=en-US&page=1');
  final response = await http.get(url);

  if (response.statusCode == 200) {
    final data = json.decode(response.body);
    if (data == null || data['results'] == null) {
      throw Exception('Empty or invalid data received');
    }
    final actors = (data['results'] as List)
      .map((actor) => Actor.fromJson(actor))
      .toList();
    return actors;
  } else {
    throw Exception(
      'Failed to load data. Status code: ${response.statusCode}');
  }
}
```

¿Qué hace? Obtiene el reparto (actores) de una película específica. Crea una URL específica para los créditos de la película usando el movield. Realiza una solicitud HTTP. Si el servidor responde correctamente, decodifica la lista de actores del campo **cast** y convierte cada actor en un objeto Actor.

```
static Future<List<Actor>> getMovieCast(int movieId) async {
    try {
        final url = Uri.parse(
            '${Api.baseUrl}movie/$movieId/credits?api_key=${Api.apiKey}');
        final response = await http.get(url);

        if (response.statusCode == 200) {
            final data = json.decode(response.body);
            final cast = (data['cast'] as List)
                .map((actor) => Actor.fromJson(actor))
                .toList();
            return cast;
        } else {
            throw Exception('Failed to load movie cast');
        }
    } catch (e) {
        print('Error while fetching cast: $e');
        throw Exception('Failed to load movie cast');
    }
}
```

¿Qué hace? Obtiene los detalles completos de un actor dado su actorId, crea la URL con el actorId, realiza la solicitud HTTP y si tiene éxito, convierte la respuesta JSON en un objeto Actor.

```
static Future<Actor> getActorDetailsById(int actorId) async {
    final url =
        '${Api.baseUrl}person/$actorId?api_key=${Api.apiKey}&language=en-US';

    try {
        final response = await http.get(Uri.parse(url));
        if (response.statusCode == 200) {
            final data = json.decode(response.body);
            return Actor.fromJson(data);
        } else {
            throw Exception("Failed to load actor details");
        }
    } catch (e) {
        print("Error: $e");
        throw Exception("Failed to fetch actor details");
    }
}
```

¿Qué hace? Obtiene una lista de películas en las que ha participado un actor usando su actorId, construye una URL con el actorId para obtener las películas, realiza la solicitud HTTP, decodifica la lista de películas desde el campo **cast** y convierte cada película en un objeto **Movie**.

```
static Future<List<Movie>> getActorMovies(int actorId) async {
    final url =
        '${Api.baseUrl}person/$actorId/movie_credits?api_key=${Api.apiKey}&language=en-US';

    try {
        final response = await http.get(Uri.parse(url));
        if (response.statusCode == 200) {
            final data = json.decode(response.body);
            final movies = (data['cast'] as List)
                .map((movie) => Movie.fromMap(movie))
                .toList();
            return movies;
        } else {
            throw Exception("Failed to load actor movies");
        }
    } catch (e) {
        print("Error: $e");
        throw Exception("Failed to load actor movies");
    }
}
```

9.2.4.2 Carpeta Controllers

Ahora nos situamos en la carpeta de controllers y seleccionamos el archivo que acá copiado y vamos a modificar el código para que funcione como queremos, para eso vamos a modificar el código y vamos a poner esto:

Cuando este controlador se inicializa (gracias a `onInit`), llama a la función `fetchPopularActors()`.

1. `fetchPopularActors()`:

- Muestra un indicador de carga (`isLoading`).
- Llama a la API para obtener los actores populares.
- Llena `actorsList` con los actores obtenidos.
- Apaga el indicador de carga cuando termina.

Si la app está observando `isLoading` o `actorsList`, cualquier cambio en estas variables actualizará automáticamente la interfaz.

```
> controllers > actors_controller.dart > ActorsController > fetchPopularActors
1 import 'package:get/get.dart';
2 import 'package:movies/api/api_service.dart';
3 import 'package:movies/models/actors.dart';
4
5 class ActorsController extends GetxController {
6   var isLoading = true.obs;
7   var actorsList = <Actor>[].obs;
8
9   @override
10  void onInit() {
11    super.onInit();
12    fetchPopularActors();
13  }
14
15  Future<void> fetchPopularActors() async {
16    try {
17      isLoading(true);
18      final actors = await ApiService.getPopularActors();
19      actorsList.assignAll(actors);
20    } finally {
21      isLoading(false);
22    }
23  }
24}
25
```

9.2.4.3 Carpeta Models

Ahora vamos a seguir modificando los archivos copiados, y para eso vamos a seleccionar el siguiente archivo que es el de los **Models**:

Esta clase convierte datos JSON en objetos Actor que puedes usar fácilmente en tu app.

Propiedades: Representan toda la información del actor (nombre, fecha de nacimiento, películas, etc.).

Constructor: Sirve para crear un actor manualmente.

fromJson: Convierte un JSON (como el que obtienes de una API) en un objeto Actor.

_getGender: Traduce el número de género del JSON en texto comprensible ("Male", "Female").

```

models > actors.dart > Actor
1  class Actor {
2    final int id;
3    final String name;
4    final String profilePath;
5    final String? biography;
6    final String? dateOfBirth;
7    final String? placeOfBirth;
8    final String? gender;
9    final String? knownForDepartment;
10   final List<String>? movieTitles; // Puedes agregar una lista de títulos de películas
11   final String? deathDay; // Para manejar actores que ya no están vivos
12
13  Actor({
14    required this.id,
15    required this.name,
16    required this.profilePath,
17    this.biography,
18    this.dateOfBirth,
19    this.placeOfBirth,
20    this.gender,
21    this.knownForDepartment,
22    this.movieTitles,
23    this.deathDay,
24  });
25
26  factory Actor.fromJson(Map<String, dynamic> json) {
27    return Actor(
28      id: json['id'] ?? 0,
29      name: json['name'] ?? 'Unknown',
30      profilePath: json['profile_path'] ?? '',
31      biography: json['biography'] ?? '',
32      dateOfBirth: json['birthday'] ?? '',
33      placeOfBirth: json['place_of_birth'] ?? '',
34      gender: _getGender(json['gender']), // Usamos una función para convertir el número
35      knownForDepartment: json['known_for_department'] ?? '',
36      movieTitles: json['known_for'] != null
37        ? List<String>.from(json['known_for'].map((movie) => movie['title'] ?? '')) 
38        : null,
39      deathDay: json['deathday'] ?? '',
40    ); // Actor
41  }
42
43  // Función para convertir el número de género a texto (1 = mujer, 2 = hombre)
44  static String? _getGender(int? gender) {
45    if (gender == null) return null;
46    if (gender == 1) return 'Female';
47    if (gender == 2) return 'Male';
48    return 'Unknown';
49  }
50 }
```

9.2.4.4 Carpeta Screens

Vale, como esta parte es muy extensa, el código estará dividido;



Aquí está el código completo:

```
1  import 'package:fade_shimmer/fade_shimmer.dart';
2  import 'package:flutter/material.dart';
3  import 'package:get/get.dart';
4  import 'package:movies/api/api.dart';
5  import 'package:movies/api/api_service.dart';
6  import 'package:movies/models/actors.dart';
7  import 'package:movies/screens/details_screen.dart';
8  import 'package:movies/models/movie.dart';
9  import 'package:movies/controllers/movies_controller.dart';
10
11 class PopularActorsScreen extends StatelessWidget {
12   final int actorId;
13
14   const PopularActorsScreen({
15     Key? key,
16     required this.actorId,
17   }) : super(key: key);
18
19   @override
20   Widget build(BuildContext context) {
21     return SafeArea(
22       child: Scaffold(
23         body: SingleChildScrollView(
24           child: Column(
25             children: [
26               // Header con botón de volver y título
27               Padding(
28                 padding: const EdgeInsets.only(left: 24, right: 24, top: 34),
29                 child: Row(
30                   mainAxisAlignment: MainAxisAlignment.spaceBetween,
31                   crossAxisAlignment: CrossAxisAlignment.center,
32                   children: [
33                     IconButton(
34                       tooltip: 'Back to home',
35                       onPressed: () => Get.back(),
36                       icon: const Icon(
37                         Icons.arrow_back_ios,
38                         color: Colors.white,
39                       ), // Icon
40                     ), // IconButton
41                     const Text(
42                       'Actor Detail',
43                       style: TextStyle(),
44                       fontWeight: FontWeight.w400,
45                       fontSize: 24,
46                     ), // TextStyle
47                     ), // Text
48                   ],
49                 ), // Row
50               ), // Padding
51               const SizedBox(height: 40),
52               // Imagen y nombre del actor
53               SizedBox(
54                 height: 330,
55                 child: Stack(
56                   children: [
57                     FutureBuilder<Actor?>(
58                       future: ApiService.getActorDetailsById(actorId),
59                       builder: (_, snapshot) {
60                         if (snapshot.connectionState ==
61                           ConnectionState.waiting) {
62                           return const Center(
63                             child: CircularProgressIndicator()); // Center
64                         }
65
66                         if (snapshot.hasData) {
67                           final actor = snapshot.data!; // Obtener el actor
68                           return SizedBox(
69                             height: 330,
```

```
final actor = snapshot.data!; // Obtener el actor
return SizedBox(
  height: 330,
  child: Stack(
    children: [
      // Contenedor para la imagen de fondo o cualquier otra
      ClipRRect(
        borderRadius: const BorderRadius.only(
          bottomLeft: Radius.circular(16),
          bottomRight: Radius.circular(16),
        ), // BorderRadius.only
        child: Image.network(
          Api.imageBaseUrl + actor.profilePath,
          width: Get.width,
          height: 250,
          fit: BoxFit.cover,
          loadingBuilder: (_, __, ___) {
            if (___ == null) return ___;
            return FadeShimmer(
              width: Get.width,
              height: 250,
              highlightColor: const Color(0xff22272F),
              baseColor: const Color(0xff20252d),
            ); // FadeShimmer
          },
        ),
        errorBuilder: (_, __, ___) => const Align(
          alignment: Alignment.center,
          child: Icon(
            Icons.broken_image,
            size: 250,
          ), // Icon
        ), // Align
      ), // Image.network
    ], // ClipRRect
    // Aquí colocamos el IconButton en la parte superior
    Positioned(
      top: 30, // Ajusta esta posición según sea necesario
      right: 24, // Ajusta el margen para alejar el botón del icono
      child: IconButton(
        tooltip: 'Save this actor to your list',
        onPressed: () {
          // Llamar al controlador para guardar al actor
          Get.find<MoviesController>()
            .addToActorList(actor);
        },
        icon: Obx(() {
          // Aquí verificamos si el actor está en la lista
          final isSaved =
            Get.find<MoviesController>()
              .isActorInList(actor);
          return Icon(
            isSaved
              ? Icons.bookmark
              : Icons
                  .bookmark_outline, // Cambiar el icono
            color: Colors.white,
            size: 33,
          ); // Icon
        }), // Obx
      ), // IconButton
    ), // Positioned
    // Aquí puedes agregar más elementos encima de la imagen
    Positioned(
      top: 255, // Ajusta la posición para el nombre del actor
      left: 155,
```



```
        ), // Text
    ), // Padding
    const Divider(
        color: Colors.grey, thickness: 1), // Divider
    Padding(
        padding: const EdgeInsets.all(16.0),
        child: Text(
            'Date: ${actor.dateOfBirth?.isEmpty ?? true ? 'No biography available' : actor.dateOfBirth!}'),
            style: const TextStyle(
                fontSize: 24,
            ), // TextStyle
        ), // Text
    ), // Padding
    const Divider(
        color: Colors.grey, thickness: 1), // Divider
    Padding(
        padding: const EdgeInsets.all(16.0),
        child: Text(
            'Gender: ${actor.gender}',
            style: const TextStyle(
                fontSize: 24,
            ), // TextStyle
        ), // Text
    ), // Padding
    const Divider(
        color: Colors.grey, thickness: 1), // Divider
    Padding(
        padding: const EdgeInsets.all(16.0),
        child: Text(
            'Department: ${actor.knownForDepartment}',
            style: const TextStyle(
                fontSize: 24,
            ), // TextStyle
        ), // Text
    ), // Padding
    const Divider(
        color: Colors.grey, thickness: 1), // Divider
    Padding(
        padding: const EdgeInsets.symmetric(
            horizontal: 16.0, vertical: 8.0), // EdgeInsets.symmetric
        child: Text(
            'Biography: ${actor.biography?.isEmpty ?? true ? 'No biography available' : actor.biography!}'),
            style: const TextStyle(
                fontSize: 24,
            ), // TextStyle
        ), // Text
    ), // Padding
    ],
), // Column
); // SingleChildScrollView
}

return const Center(
    child: Text('Error loading actor details')); // Center
},
), // FutureBuilder

FutureBuilder<List<Movie>?>(
    future: ApiService.getActorMovies(actorId),
    builder: (_, snapshot) {
        if (snapshot.connectionState ==
            ConnectionState.waiting) {
            return const Center(
                child: CircularProgressIndicator()); // Center
        }

        if (snapshot.hasData) {
```

```
future: Apiservice.getActorMovies(actorId),
builder: (_, snapshot) {
  if (snapshot.connectionState ==
      ConnectionState.waiting) {
    return const Center(
      child: CircularProgressIndicator()); // Center
  }

  if (snapshot.hasData) {
    return snapshot.data!.isEmpty
        ? const Center(
            child: Text('No movies found')) // Center
        : ListView.builder(
            itemCount: snapshot.data!.length,
            itemBuilder: (_, index) {
              final movie = snapshot.data![index];
              return ListTile(
                leading: movie.posterPath != null &&
                    movie.posterPath!
                        .isNotEmpty
                ? ClipRRect(
                    borderRadius:
                        BorderRadius.circular(
                            8), // BorderRadius.circular(
                    child: Image.network(
                      Api.imageUrl +
                          movie.posterPath!,
                      width:
                          50, // Ajusta el tamaño de la imagen
                      height: 75,
                      fit: BoxFit.cover,
                    ), // Image.network
                ) // ClipRRect
                : const Icon(Icons.movie,
                    size:
                        50), // Icono en caso de no tener imagen
                title: Text(
                  movie.title.isNotEmpty
                  ? movie.title
                  : 'No title available', // Si no hay título
                style: const TextStyle(
                  color: Colors
                      .white), // Para que el texto sea blanco
                ), // Text
                subtitle: Text(
                  movie.releaseDate ??
                  'No release date available', // Si no tiene fecha de lanzamiento
                style: const TextStyle(
                  color: Colors
                      .white), // Para que el texto sea blanco
                ), // Text
                onTap: () {
                  Get.to(() => DetailsScreen(
                    movie:
                        movie)); // Navega a la pantalla de detalles
                },
              );
            });
  }
}

return const Center(
  child: Text('Error loading movies')) // Center
},
), // FutureBuilder
,
```

9.2.4.4.1 Explicación del código

Ahora vamos a explicar que hace cada cosa del código que te acabo de pasar en fotos, vamos a empezar por él;

Estructura de la Pantalla

- **PopularActorsScreen** es una pantalla que recibe el actorId como parámetro para mostrar información específica sobre el actor.
- El constructor permite pasar el actorId cuando se navega a esta pantalla.

```
class PopularActorsScreen extends StatelessWidget {
    final int actorId;

    const PopularActorsScreen({
        Key? key,
        required this.actorId,
    }) : super(key: key);
```

SafeArea y Scaffold

- **SafeArea**: Asegura que los elementos de la pantalla no se superpongan con áreas sensibles como las barras de estado o navegación.
- **Scaffold**: Es la estructura base de la pantalla, donde se coloca el contenido principal.
- **SingleChildScrollView**: Permite que el contenido se desplace si es demasiado largo para la pantalla.
- Esta sección muestra el encabezado con un botón para volver a la pantalla anterior y un título que dice "Actor Detail".
- El botón utiliza Get.back() para navegar hacia atrás cuando se presiona.

```
@override
Widget build(BuildContext context) {
    return SafeArea(
        child: Scaffold(
            body: SingleChildScrollView(
                child: Column(
                    children: [
                        // Header con botón de volver y título
                        Padding(
                            padding: const EdgeInsets.only(left: 24, right: 24, top: 34),
                            child: Row(
                                mainAxisAlignment: MainAxisAlignment.spaceBetween,
                                crossAxisAlignment: CrossAxisAlignment.center,
                                children: [
                                    IconButton(
                                        tooltip: 'Back to home',
                                        onPressed: () => Get.back(),
                                        icon: const Icon(
                                            Icons.arrow_back_ios,
                                            color: Colors.white,
                                        ), // Icon
                                    ), // IconButton
                                    const Text(
                                        'Actor Detail',
                                        style: TextStyle(
                                            fontWeight: FontWeight.w400,
                                            fontSize: 24,
                                        ), // TextStyle
                                    ), // Text
                                ],
                            ), // Row
                        ), // Padding
                    ],
                ),
            ),
        ),
    );
}
```

Imagen y Nombre del Actor

- Usamos un **FutureBuilder** para obtener los detalles del actor desde la API. Si los datos están disponibles, se muestran.
- La imagen del actor se carga de manera optimizada con un **FadeShimmer** mientras se carga, y se muestra un ícono en caso de error de carga.

- También hay un botón de "guardar" que permite añadir el actor a una lista, y su estado cambia según si el actor ya está guardado o no.

```

        ...
        SizedBox(
          height: 330,
          child: Stack(
            children: [
              FutureBuilder<Actor?>(
                future: ApiService.getActorDetailsById(actorId),
                builder: (_, snapshot) {
                  if (snapshot.connectionState ==
                      ConnectionState.waiting) {
                    return const Center(
                      child: CircularProgressIndicator());
                  }
                }
              ),
              if (snapshot.hasData) {
                final actor = snapshot.data!;
                return SizedBox(
                  height: 330,
                  child: Stack(
                    children: [
                      // Contenedor para la imagen de fondo o cualquier otro
                      ClipRect(
                        borderRadius: const BorderRadius.only(
                          bottomLeft: Radius.circular(16),
                          bottomRight: Radius.circular(16),
                        ),
                        child: Image.network(
                          Api.imageUrl + actor.profilePath,
                          width: Get.width,
                          height: 250,
                          fit: BoxFit.cover,
                          loadingBuilder: (_, __, ___) {
                            if (__ == null) return ___;
                            return FadeShimmer(
                              width: Get.width,
                              height: 250,
                              highlightColor: const Color(0xff22272f),
                              baseColor: const Color(0xff20252d),
                            );
                          },
                        ),
                      ],
                    ],
                  ),
                );
              }
            ],
          ),
        );
      }
    }
  }
}

```

Tabs (Pestañas de Información y Películas)

- **TabBar**: Aquí se definen las pestañas que el usuario puede seleccionar. En este caso, "About Actor" y "Movies".
- **TabBarView**: Se encargará de mostrar el contenido correspondiente a cada tab. Uno muestra información sobre el actor y el otro muestra las películas en las que ha participado.

```

const SizedBox(height: 25),
Padding(
  padding: const EdgeInsets.all(24),
  child: DefaultTabController(
    length: 3,
    child: Column(
      mainAxisSize: MainAxisSize.min,
      children: [
        const TabBar(
          indicatorWeight: 4,
          indicatorSize: TabBarIndicatorSize.label,
          indicatorColor: Color.fromARGB(255, 255, 255, 255),
          tabs: [
            Tab(text: 'About Actor'),
            Tab(text: 'Movies'),
          ],
        ),
        SizedBox(
          height: 400,
          child: TabBarView(
            children: [
              // About Actor Tab
              FutureBuilder<Actor?>(
                future: ApiService.getActorDetailsById(actorId),
                builder: (_, snapshot) {
                  if (snapshot.connectionState ==
                      ConnectionState.waiting) {
                    return const Center(
                      child: CircularProgressIndicator());
                  }
                }
              );
            ],
          ),
        ),
      ],
    ),
  ),
)

```

Información del Actor (Sobre el Actor)

- En esta parte, se muestra información sobre el actor, como su nombre, fecha de nacimiento, género y biografía.
- Si los datos no están disponibles o ocurre un error, se muestra un mensaje adecuado.

```
FutureBuilder<Actor?>(
    future: ApiService.getActorDetailsById(actorId),
    builder: (_, snapshot) {
        if (snapshot.connectionState ==
            ConnectionState.waiting) {
            return const Center(
                child: CircularProgressIndicator()); // Center
        }
        //....
    }
);
```

Películas del Actor

En este **FutureBuilder**, se obtiene la lista de películas del actor y se muestran en una lista. Si el actor tiene películas asociadas, se muestran en tarjetas con el título, la fecha de lanzamiento y la imagen de póster. Si no, se muestra un mensaje indicando que no hay películas.

```
FutureBuilder<List<Movie>?>(
    future: ApiService.getActorMovies(actorId),
    builder: (_, snapshot) {
        if (snapshot.connectionState ==
            ConnectionState.waiting) {
            return const Center(
                child: CircularProgressIndicator()); // Center
        }
        //....
    }
);
```

9.2.4.5 Carpeta Widgets

Ahora vamos a seguir modificando los archivos copiados, y para eso vamos a seleccionar el siguiente y ultimo archivo que es el de los **Widgets**;

Este código define un widget **ActorItem** que muestra la foto (o un ícono en su lugar) y el nombre de un actor en una interfaz sencilla. El widget utiliza un **Column** para organizar la imagen y el nombre, y les agrega espacio alrededor con un **Padding**. Si el actor tiene una foto, se muestra en forma de círculo; si no tiene, se muestra un ícono genérico. El nombre del actor se muestra debajo de la imagen, centrado y con un estilo específico.

Este widget puede ser utilizado en una lista de actores, mostrando de forma visualmente atractiva la foto y el nombre de cada actor.

```
// actor_item.dart
import 'package:flutter/material.dart';
import 'package:movies/models/actors.dart';

class ActorItem extends StatelessWidget {
    final Actor actor;

    const ActorItem({Key? key, required this.actor}) : super(key: key);

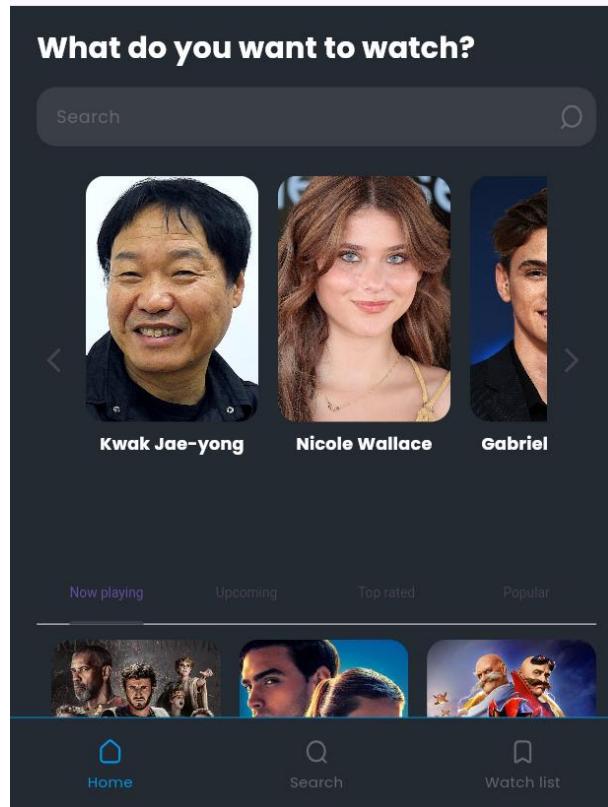
    @override
    Widget build(BuildContext context) {
        return Padding(
            padding: const EdgeInsets.all(8.0),
            child: Column(
                children: [
                    actor.profilePath.isNotEmpty
                        ? ClipOval(
                            child: Image.network(
                                'https://image.tmdb.org/t/p/w500/${actor.profilePath}',
                                width: 100,
                                height: 100,
                                fit: BoxFit.cover,
                            ), // Image.network
                        ) // ClipOval
                        : Icon(
                            Icons.person,
                            size: 100,
                            color: Colors.grey,
                        ), // Icon
                    const SizedBox(height: 8),
                    Text(
                        actor.name,
                        textAlign: TextAlign.center,
                        style: const TextStyle(fontSize: 12, fontWeight: FontWeight.w500),
                    ), // Text
                ],
            ), // Column
        ); // Padding
    }
}
```

9.3 Funcionamiento de la aplicación

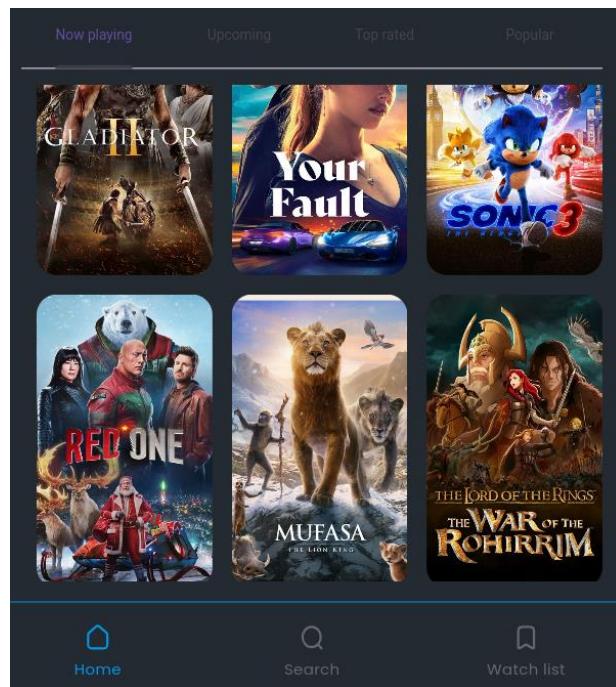
Ahora vamos a ver cómo funciona nuestra app, de manera que vamos a ver cómo queda:

Como podemos ver en el menú, tenemos un botón en la lista de actores donde si le damos clic en las flechas se moverá hacia la dirección en la que va la flecha y entonces se moverá la lista de los actores para que puedas ver mas actores.

Home



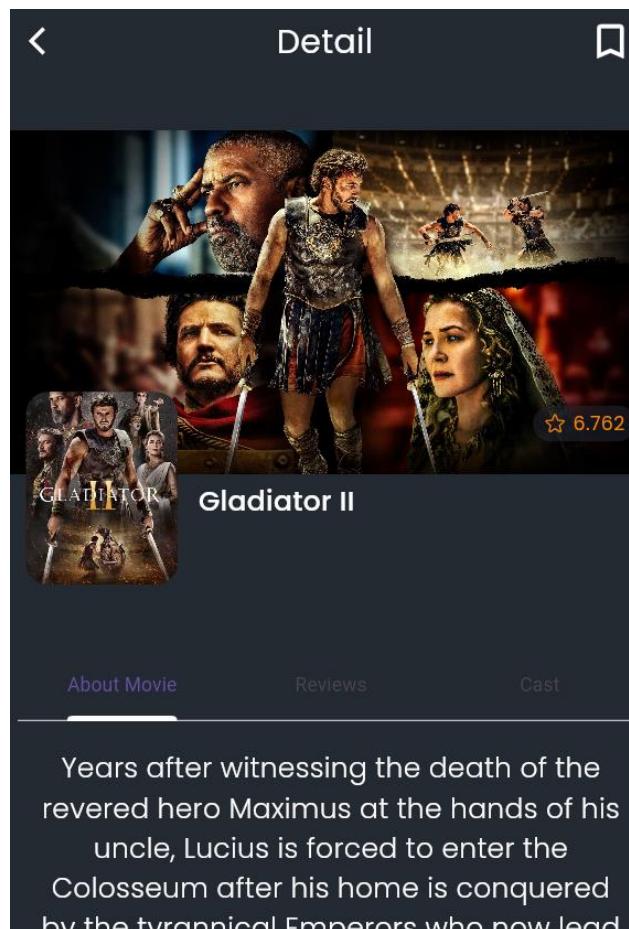
También podemos observar que tenemos varias opciones en el apartado de las pelis, donde podemos ver el las mas top, las populares, las que están en estreno y las que vendrán, también puedes bajar para observar mas películas.



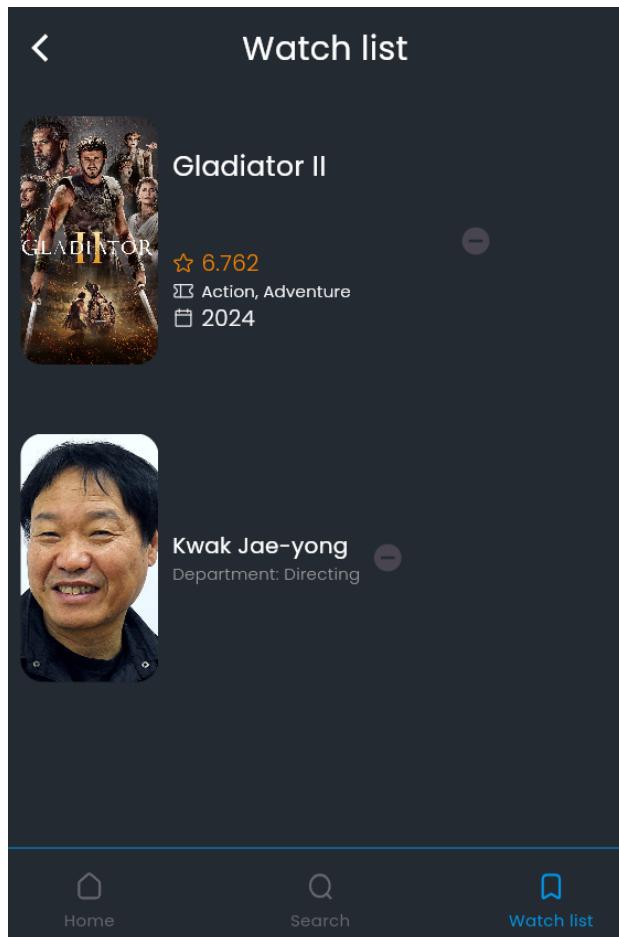
También si le damos a un actor, se nos aparecerá toda su información, donde también podemos ver las películas en las que ha estado, y también tiene el botón de guardar:



Y lo mismo pasa con las películas, si le das a una de las películas, podemos ver su información, los comentarios y los actores que hay en la película, y el mismo botón de guardar.



Y si nos vamos a la opción de guardar se nos aparecer un botón para eliminar el actor o la película de guardados.



También hay un video para que veas cómo funciona:



Y el enlace al GitHub para que puedas utilizar mi Api o aplicarla a otro api:
https://github.com/teropod24/Api_Actores.git

10. SupaBase

10.1 Introducción

Superbase es un programa de base de datos de escritorio para usuarios finales que comenzó en Commodore 64 y se trasladó desde allí a varios sistemas operativos a lo largo de más de 20 años. También incluía un lenguaje de programación para automatizar tareas orientadas a bases de datos y, en versiones posteriores, incluía diseñadores de formularios e informes WYSIWYG, así como capacidades de programación más sofisticadas.

Superbase se ha utilizado para tareas muy básicas de usuario final, pero su verdadera fortaleza reside en la capacidad de programadores relativamente inexpertos para crear aplicaciones

complejas. Estas aplicaciones suelen construirse con el tiempo a medida que surgen las necesidades. Los tipos de aplicaciones abarcan desde sistemas de contabilidad, paquetes ERP / MRP, sistemas de información empresarial, sistemas de control de producción y productos complejos similares hasta sistemas muy básicos de gestión de contactos o listas de miembros.

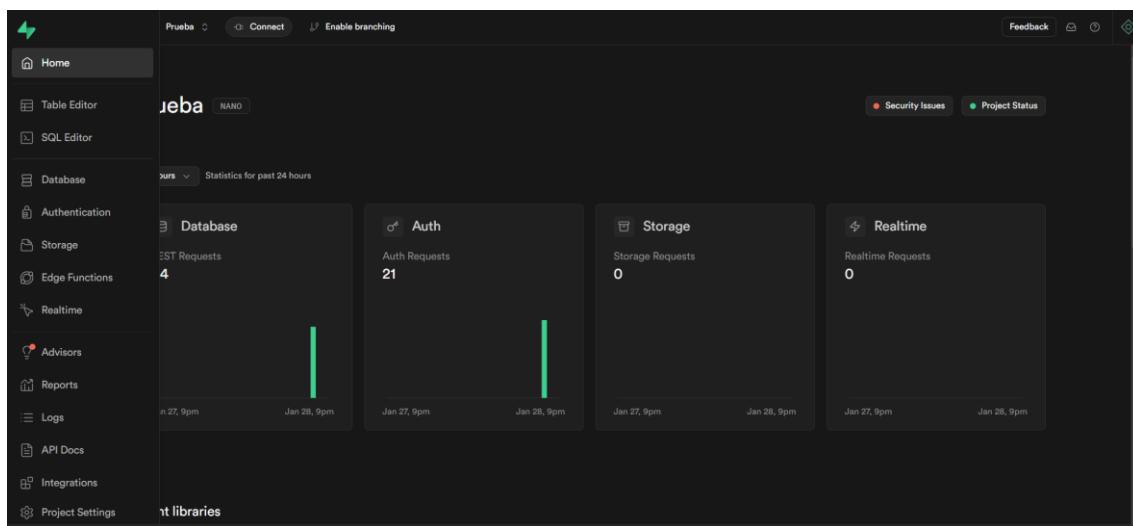


10.1.1 Crear una cuenta de SupaBase

Vamos a crear una cuenta de SupaBase porque luego lo vamos a necesitar para la práctica, para eso primero vamos a la pagina web y le damos a crear una nueva cuenta, podemos hacerlo directamente des de este link:

<https://supabase.com/dashboard/sign-in?returnTo=%2Fprojects>

Y si podemos utilizar la cuenta de la escuela o una personal, o incluso la del GitHub, la que prefieras. Y una vez dentro no tendréis nada, pero las opciones de la izquierda serán claves para más adelante para la práctica que tendremos que hacer. Entonces ya tendremos una cuenta de SupaBase.



10.2 Introducción al Ejercicio

En este caso vamos a utilizar el ‘**SupaBase**’ para guardar datos en la nube y de ahí poder recoger esos datos para validar o mostrar, etc. Para eso vamos a ver el ejercicio que nos plantea esta práctica:

Activitats:

La nostra cinquena aplicació en Flutter:

A partir dels exemples mostrats en el PPT hauràs de crear una nova App i pujar-la al teu Github.

Recorda documentar àmpliament la seva estructura i el seu funcionament.

S'agrairà si pots fer un vídeo mostrar la funcionalitat de la nova App.

- Has de fer servir la base de dades del SupaBase, i com a mínim crear **dues taules** que estiguin relacionades. Pots escollir guardar la informació del que et vingui més de gust, i evidentment les taules han de ser diferents de les dels exemples mostrats.
- Has de permetre **l'autenticació** dels usuaris.

Ampliacions:

- Afegir en la càrrega una finestra d'splash i un logo "xulo".
- Guardar informació d'imatges, documents, etc ... en el servei d'Storage del SupaBase, que estiguï relacionada amb els usuaris autènticats.
- Enviar notificacions als mòbils dels usuaris que estiguin adscrits al servei de subscripcions Realtime.
- Fer servir un tsvector i implementar una cerca per text.
- Utilitzar PowerSync per fer funcionar l'applicació quan està desconnectada de la xarxa de dades.

Nos dice de utilizar los ejemplos del ppt para crear una nueva app, y después crear dos nuevas tablas que estén relacionadas y que los usuarios nuevos se autentifiquen por correo para validar.

10.3 DBeaver

DBeaver es una aplicación de software cliente de SQL y una herramienta de administración de bases de datos. Para las bases de datos relacionales, utiliza la interfaz de programación de aplicaciones (API) JDBC para interactuar con las bases de datos a través de un controlador JDBC. Para otras bases de datos (NoSQL) utiliza controladores de bases de datos propietarios. Proporciona un editor que soporta el autocompletado de código y el resaltado de sintaxis. Proporciona una arquitectura de plugins (basada en la arquitectura de plugins de Eclipse) que permite a los usuarios modificar gran parte del comportamiento de la aplicación para proporcionar funcionalidad o características específicas de la base de datos que son independientes de la base de datos. Esta es una aplicación de escritorio escrita en Java y basada en la plataforma Eclipse.

La Community Edition (CE) de DBeaver es un software libre y de código abierto que se distribuye bajo la Apache License. Una edición empresarial de código cerrado de DBeaver se distribuye bajo una licencia comercial.



10.3.1 Preparación del DBeaver con SupaBase

Para poder conectarnos con el DBeaver con el SupaBase primero tendremos que descargarnos el DBeaver des de la página oficial:

<https://dbeaver.io/>

Download

DBeaver Community 24.3.3

Released on January 19th 2024 ([Milestones](#)).

It is free and open source ([license](#)).

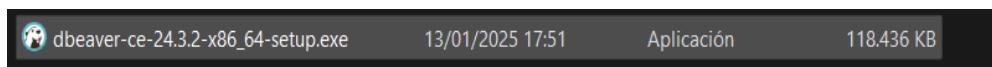
Also you can get it from the [GitHub mirror](#).

[System requirements](#).

Windows

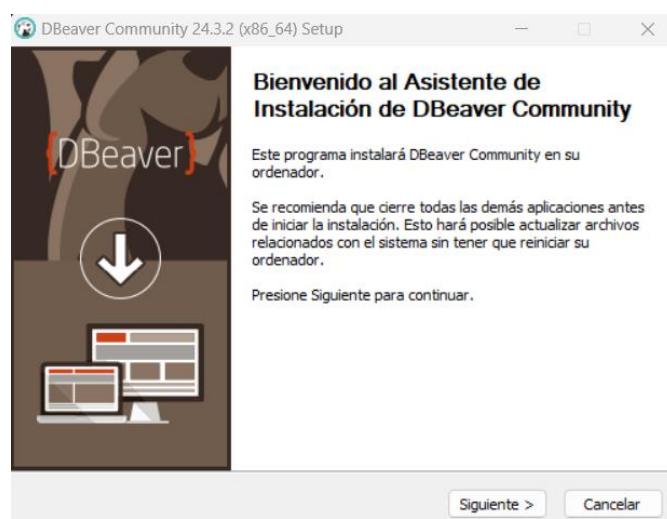
- [Windows \(installer\)](#)
- [Windows \(zip\)](#)
- [Chocolatey \(choco installdbeaver\)](#)
- [Install from Microsoft Store](#)

En este caso vamos a descargarnos la opción de Windows, pero vosotros elegir el que mejor os vaya. Y al darle a descargar, vamos a empezar la instalación del programa.

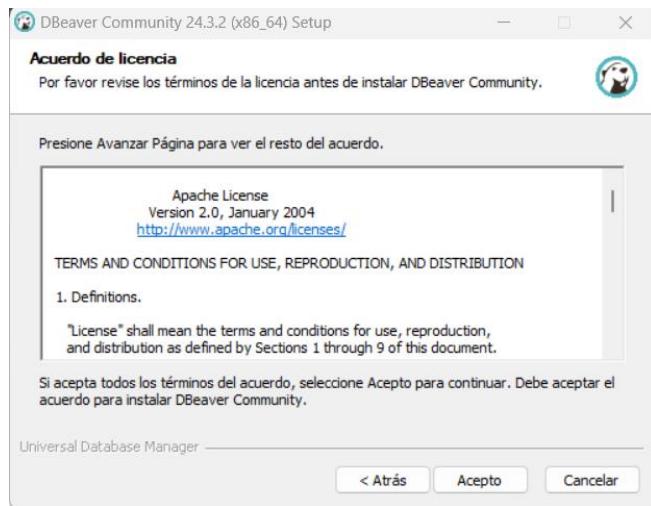


10.3.1.1 Instalación del DBeaver

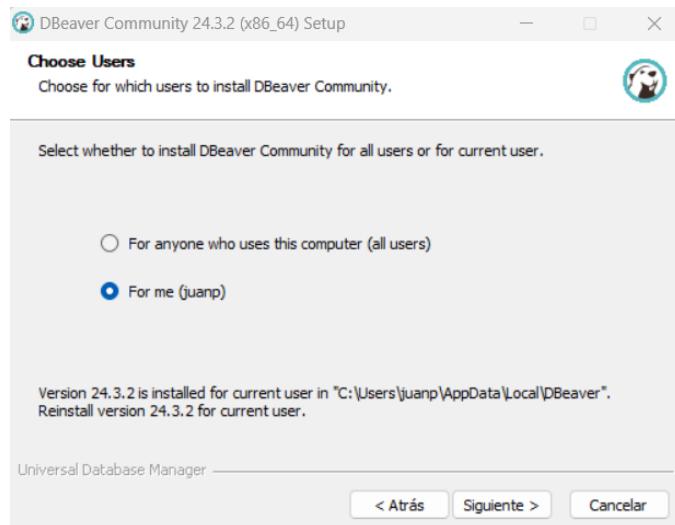
Cuando le demos doble clic al exe se nos abrirá la pantalla para empezar la instalación, donde podremos ver una pequeña información del DBeaver.



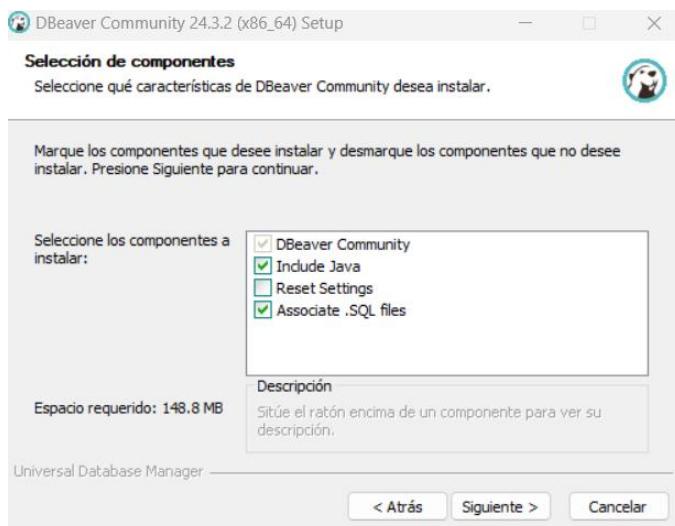
Simplemente le damos a siguiente, y le damos a aceptar las licencias.



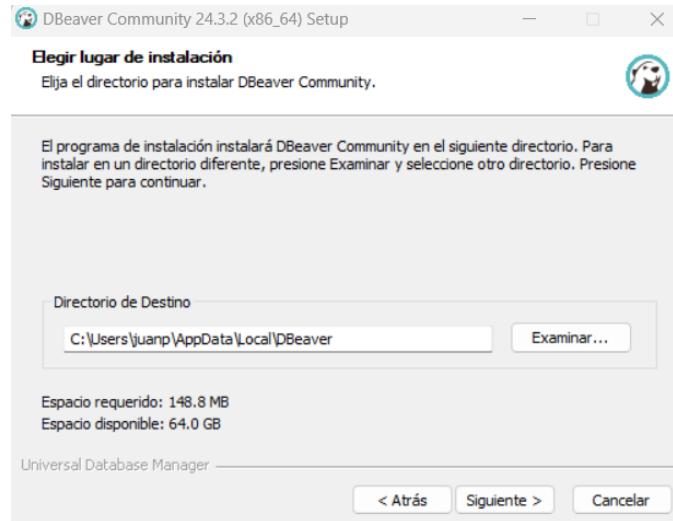
Después nos mostrara dos opciones, podemos dejar la que esta predeterminada y le damos a siguiente.



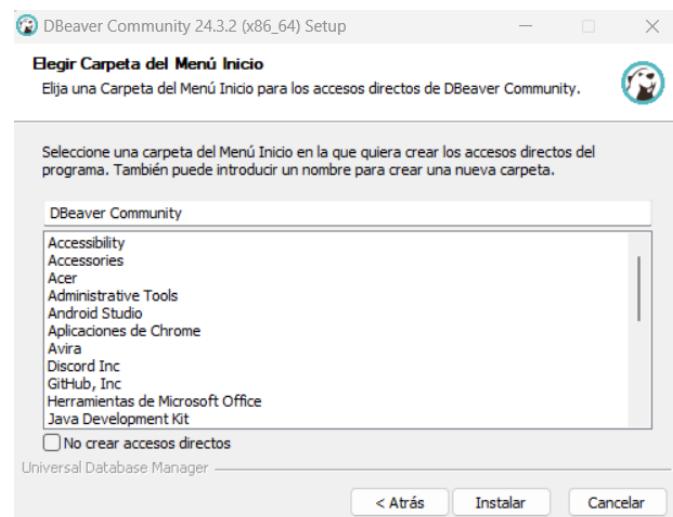
Aquí se nos aparecerá mas opciones, en mi caso voy a añadir la opción de SQL, no es necesario pero no está de mal tenerlo, y después le damos a siguiente.



Aquí es simplemente ponerle la ruta de instalación.



Y al final se nos mostrará todo lo que tenemos y una vez confirmado todo, le damos a instalar.

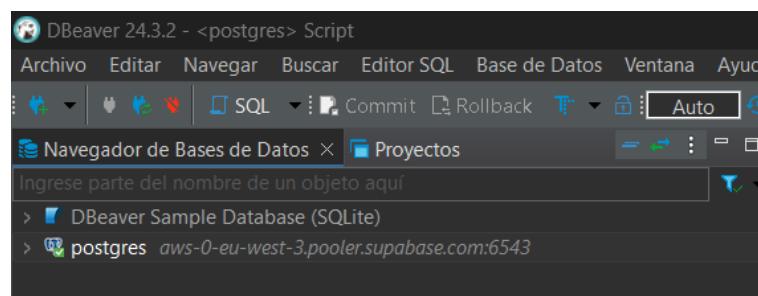


Entonces cuando acabe de instalar ya tendremos el DBeaver descargado para utilizar.

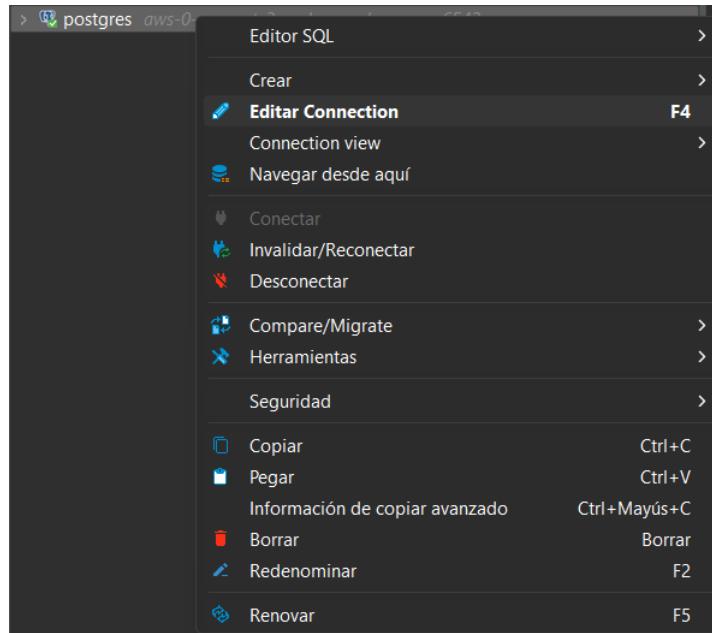
10.3.1.2 Conexión del DBeaver con SupaBase

Antes de continuar deberemos tener una cuenta de SupaBase como hemos dicho previamente antes de acceder a este paso, sino vuelve al apartado del SupaBase para poder crear la cuenta.

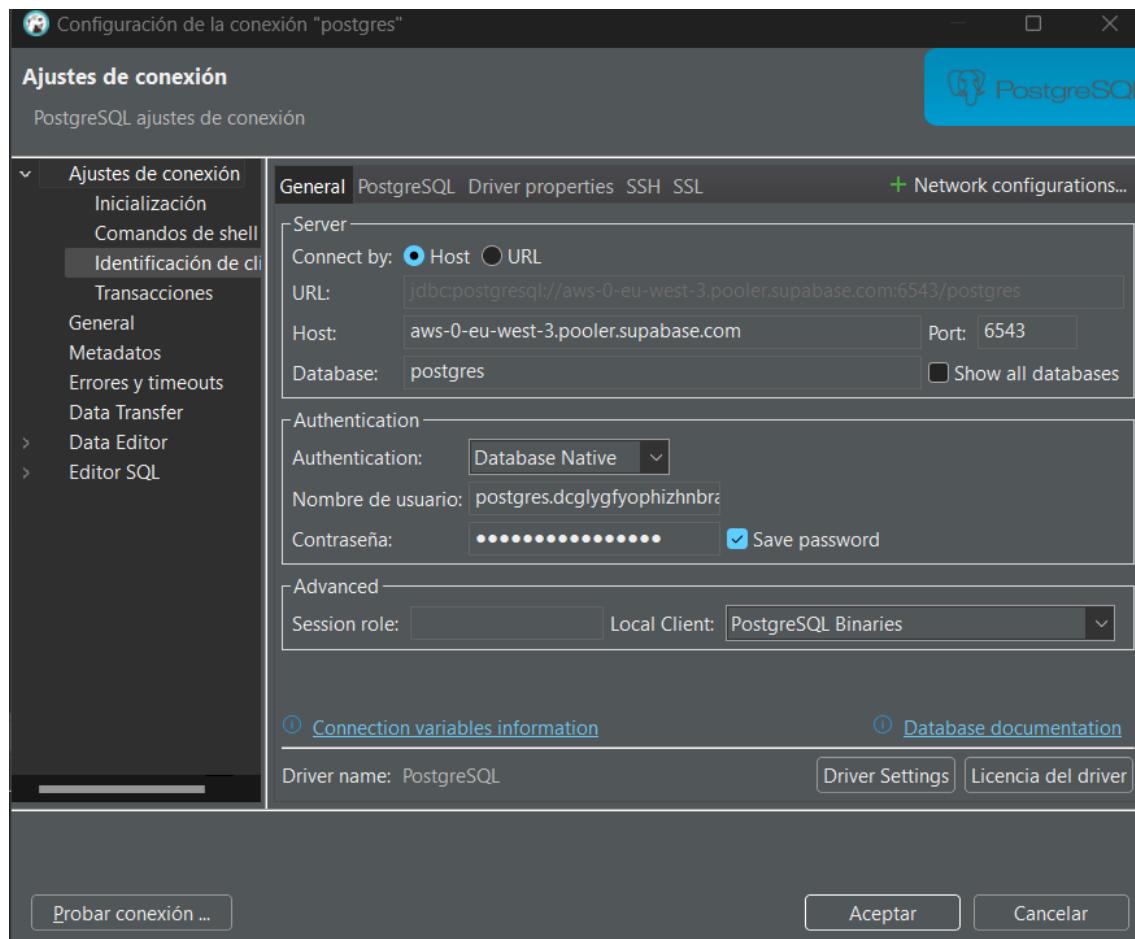
Vale, entonces si ya tienes una cuenta de SupaBase y ya tienes instalado el DBeaver, si iniciamos el Dbeaver, podremos ver a la izquierda una opción para poder conectarnos a la base de datos, esto es parecido al **SQL server**. Entonces vamos a desplegar la opción de '**postgres**'.



Entonces le damos a clic derecho a donde postgres, para poder conectarnos con la base de datos del SupaBase, y cuando le hayamos dado podremos ver varias opciones pero la que nos interesa es la que pone '**Editar Connection**'.



Entonces se nos abrirá una nueva pestaña donde aquí tendremos que poner todos los datos del SupaBase. En '**Connect By**' vamos a dejar la opción de *Host*, entonces ahora nos iremos a nuestra cuenta de SupaBase.



Configuración de la conexión "postgres"

Ajustes de conexión

PostgreSQL ajustes de conexión

General PostgreSQL Driver properties SSH SSL + Network configurations...

Server

Connect by: Host URL

URL: jdbc:postgresql://aws-0-eu-west-3.pooler.supabase.com:6543/postgres

Host: aws-0-eu-west-3.pooler.supabase.com Port: 6543

Database: postgres Show all databases

Authentication

Authentication: Database Native

Nombre de usuario: postgres.dcglygfphizhnbr

Contraseña: ***** Save password

Advanced

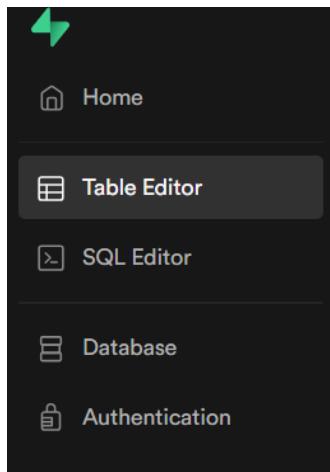
Session role: Local Client: PostgreSQL Binaries

Connection variables information Database documentation

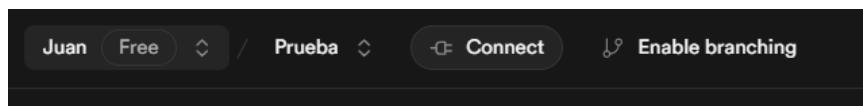
Driver name: PostgreSQL Driver Settings Licencia del driver

Probar conexión ... Aceptar Cancelar

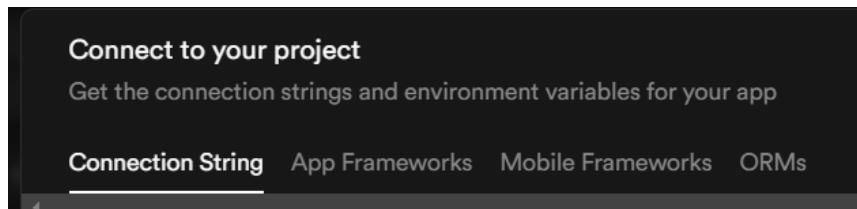
En las opciones de la izquierda nos vamos a la que pone *Table Editor*.



Entonces una vez dentro, arriba en el centro habrá una opción que pone *connect*, pues le damos y se nos abrirá una pestaña con mucha información.



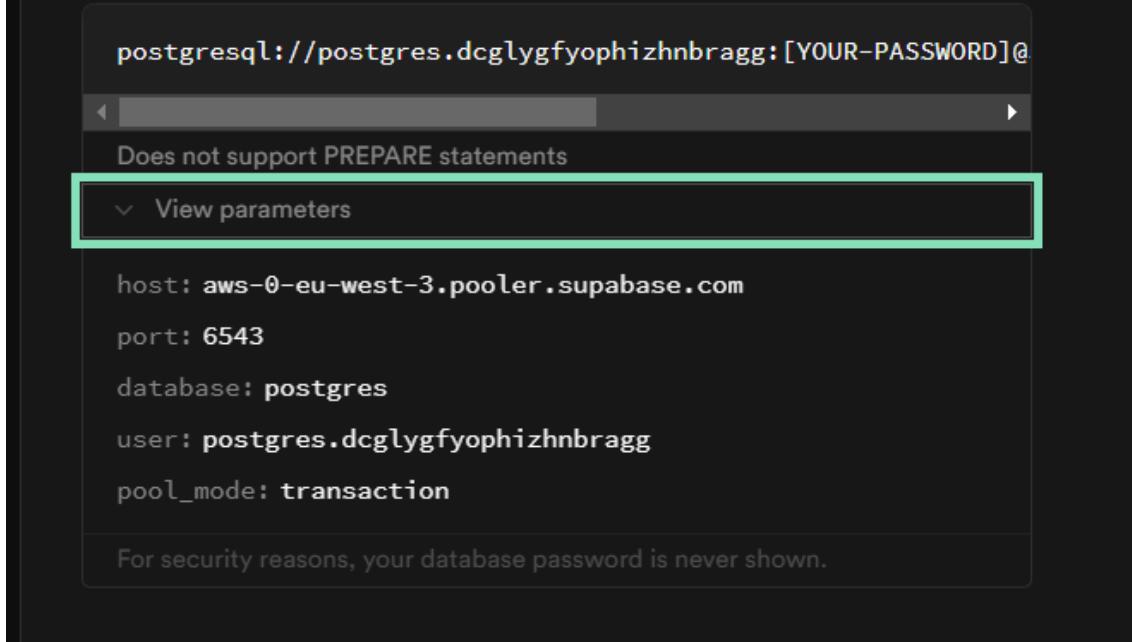
Entonces nos tendremos que situar en el apartado que esta en la imagen de abajo, que pone '**Connection String**'.



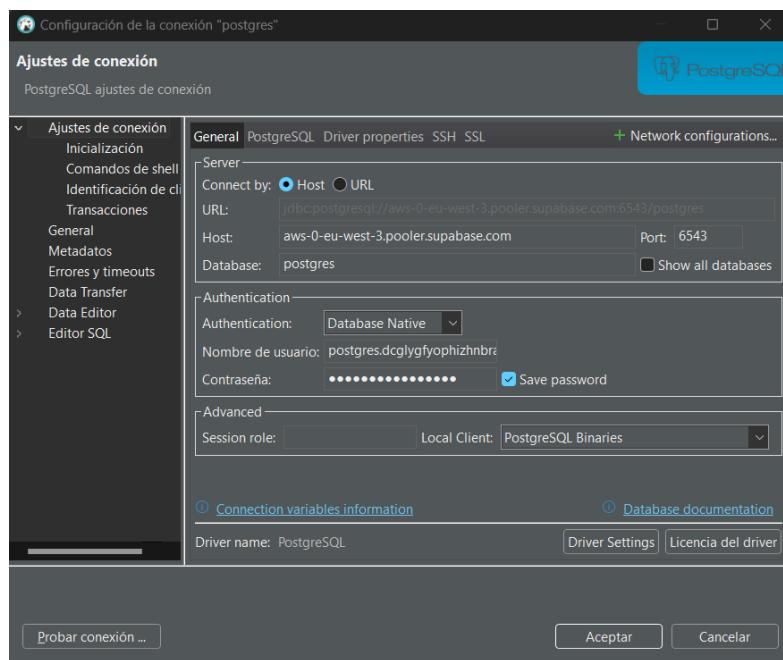
Entonces si nos vamos un poco mas abajo hay unas opciones y la que nos interesa es la que dice '**Transaction Pooler**' y si nos fijamos un poco abajo hay un desplegable que poner '**View Parameters**' le damos y hay aparecerá toda la información que tendremos que llenar en el Dbeaver para la conexión.

Transaction pooler

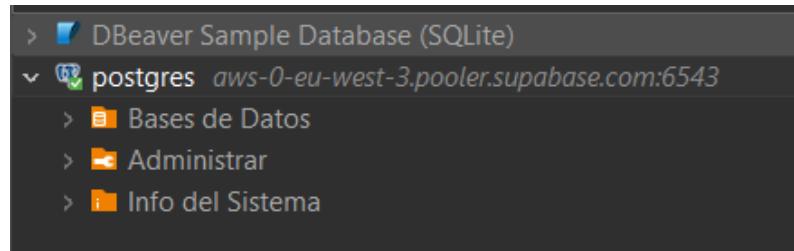
Ideal for stateless applications like serverless functions where each interaction with Postgres is brief and isolated.



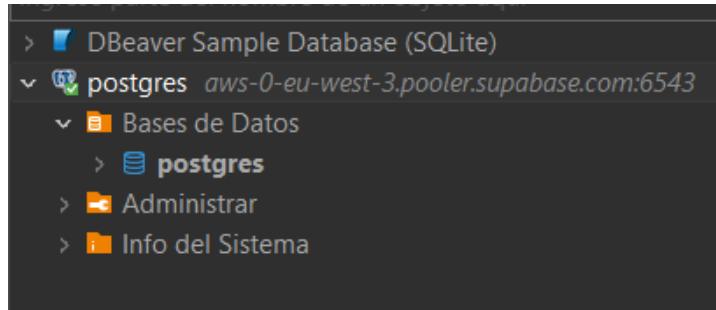
Y finalmente una vez que hemos completado todos los campos necesarios le damos a aceptar.



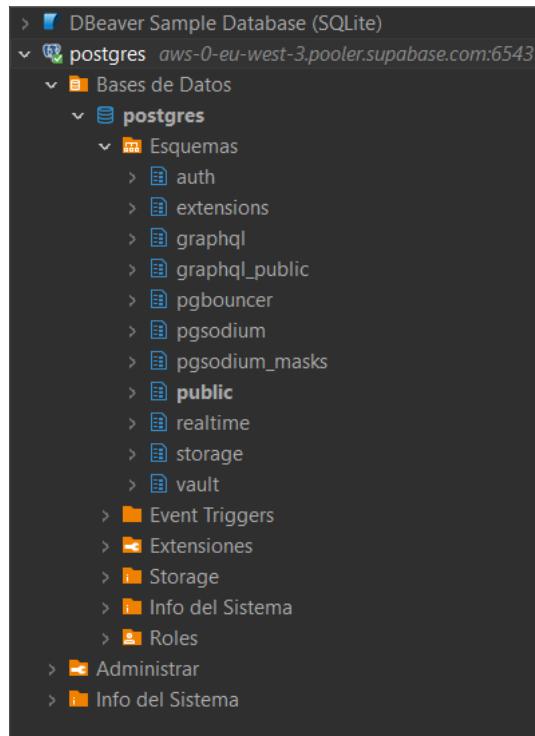
Después nos volvemos al postgres y le damos a desplegar la opción de '**Base de Datos**'.



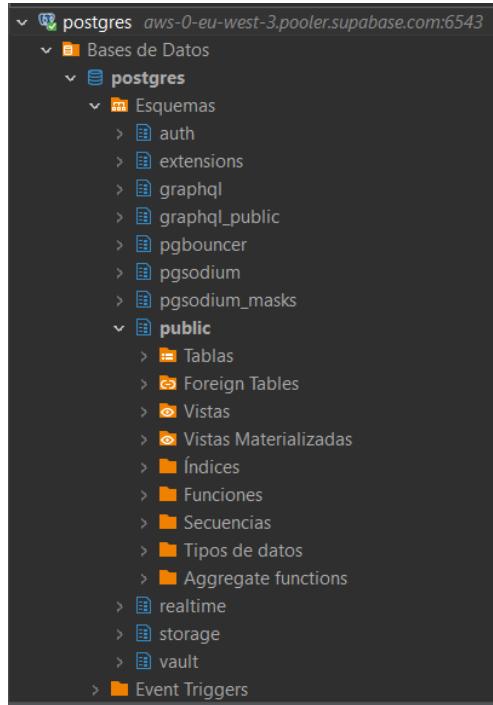
Lo mismo, pero ahora otra vez el postgres.



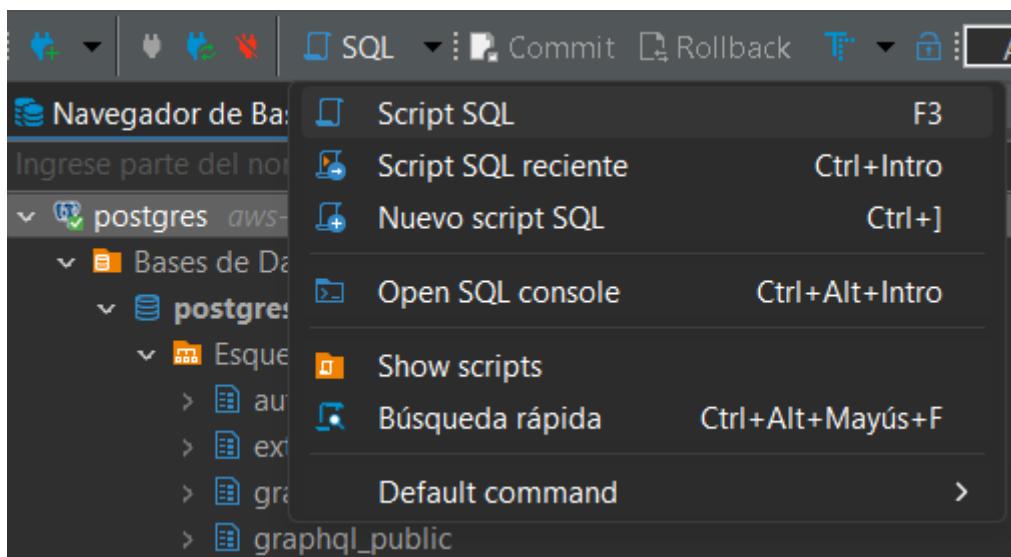
Ahora nos iremos a la opción que pone '**public**'.



Y aquí en tablas de momento no vamos a tener nada porque no hemos creado ninguna, entonces lo que vamos a hacer primero es la conexión de datos con el SupaBase.



Entonces después de todo esto, vamos a crear las tablas que necesitamos para la práctica, para eso le vamos a dar a la opción que haya arriba que dice SQL y después le damos a '**Script SQL'**



Entonces cuando le demos, se nos abrirá una pantalla a la derecha para poder escribir los scripts, y como nos había dicho y mostrado el profesor tenemos dos tablas una de user para usuarios y otro que se llama note que son para las notas, pero como no había dicho que modificáramos las tablas, sino que las añadimos, en nuestro caso vamos a añadir dos campos que también estaría bien. Vamos a añadir dos campos nuevos, uno para la edad de la película y otro para el género. Y el script final quedaría algo como esto:

```
CREATE TABLE public."Peliculas" (
```

```
    id bigint NOT NULL,
```

```
user_id bigint,  
title character varying,  
description text,  
created_at timestamp without time zone DEFAULT now(),  
edad bigint NOT null,  
genero character varying  
);  
  
ALTER TABLE public."Peliculas" OWNER TO postgres;  
  
ALTER TABLE public."Peliculas" ALTER COLUMN id ADD GENERATED BY DEFAULT AS IDENTITY  
(  
    SEQUENCE NAME public.notes_id_seq  
    START WITH 1  
    INCREMENT BY 1  
    NO MINVALUE  
    NO MAXVALUE  
    CACHE 1  
);  
  
CREATE TABLE public.users (  
    id bigint NOT NULL,  
    uid uuid DEFAULT gen_random_uuid(),  
    email character varying,  
    name character varying,  
    created_at timestamp without time zone DEFAULT now()  
);  
  
ALTER TABLE public.users OWNER TO postgres;  
COMMENT ON TABLE public.users IS 'users';  
  
ALTER TABLE public.users ALTER COLUMN id ADD GENERATED BY DEFAULT AS IDENTITY (  
    SEQUENCE NAME public.users_id_seq  
    START WITH 1  
    INCREMENT BY 1  
    NO MINVALUE
```

NO MAXVALUE

CACHE 1

);

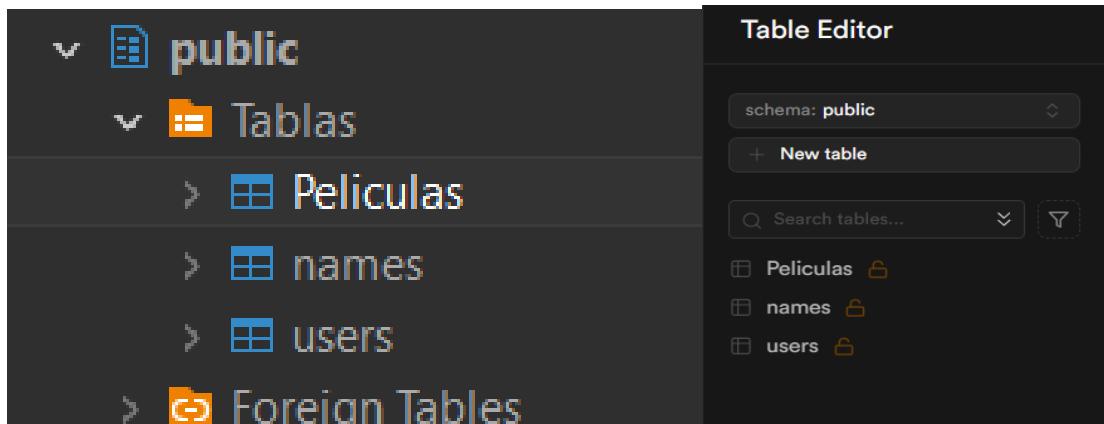
ALTER TABLE public."Peliculas"

ADD COLUMN user_id **bigint**,

ADD CONSTRAINT fk_peliculas_users **FOREIGN KEY** (user_id) **REFERENCES** public.users(id) **ON DELETE SET NULL;**

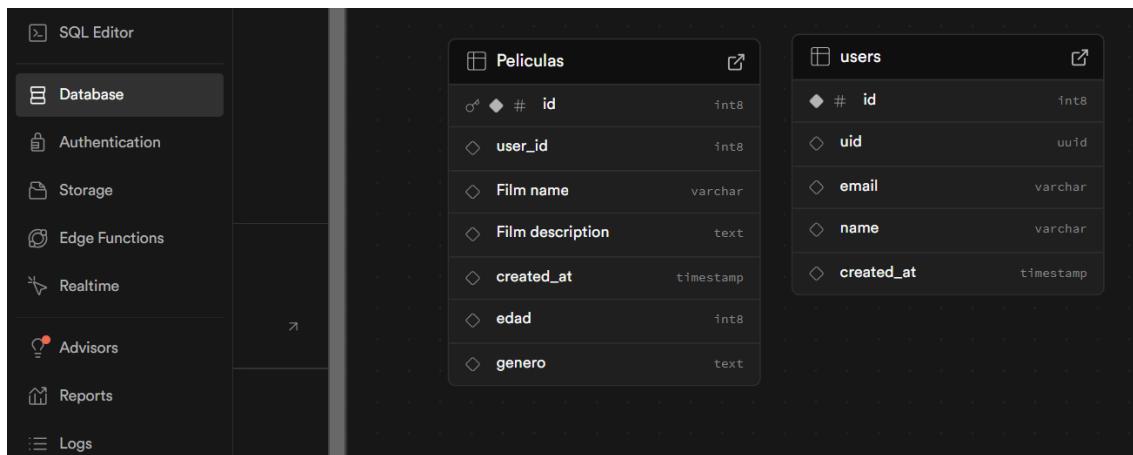
```
•CREATE TABLE public."Peliculas" (
    id bigint NOT NULL,
    user_id bigint,
    title character varying,
    description text,
    created_at timestamp without time zone DEFAULT now(),
    edad bigint NOT null,
    genero character varying
);
ALTER TABLE public."Peliculas" OWNER TO postgres;
•ALTER TABLE public."Peliculas" ALTER COLUMN id ADD GENERATED BY DEFAULT AS IDENTITY (
    SEQUENCE NAME public.notes_id_seq
    START WITH 1
    INCREMENT BY 1
    NO MINVALUE
    NO MAXVALUE
    CACHE 1
);
•CREATE TABLE public.users (
    id bigint NOT NULL,
    uid uuid DEFAULT gen_random_uuid(),
    email character varying,
    name character varying,
    created_at timestamp without time zone DEFAULT now()
);
ALTER TABLE public.users OWNER TO postgres;
COMMENT ON TABLE public.users IS 'users';
•ALTER TABLE public.users ALTER COLUMN id ADD GENERATED BY DEFAULT AS IDENTITY (
    SEQUENCE NAME public.users_id_seq
    START WITH 1
    INCREMENT BY 1
    NO MINVALUE
    NO MAXVALUE
    CACHE 1
);
```

Entonces después de ejecutar y renovar las tablas se nos debería mostrar las tablas que hemos creado con todos sus campos, y lo mismo pasa en el SupaBase, si nos vamos a Table Editor veremos que se han creado con todos sus campos.



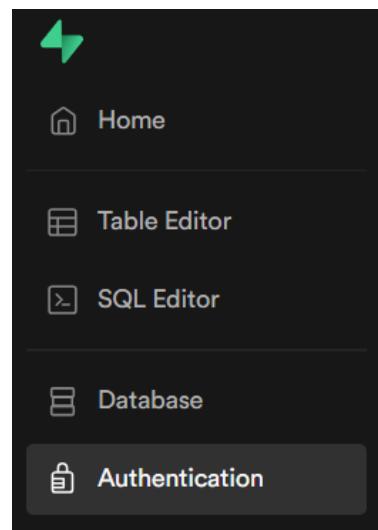
The screenshot shows the Supabase Table Editor interface. On the left, a tree view of the database schema 'public' is displayed, containing 'Tablas' (Tables) with 'Peliculas', 'names', and 'users', and a 'Foreign Tables' section. On the right, a 'Table Editor' panel is open for the 'public' schema, showing three tables: 'Peliculas', 'names', and 'users'. Each table has its columns listed: 'Peliculas' has columns 'id' (int8), 'user_id' (int8), 'Film name' (varchar), 'Film description' (text), 'created_at' (timestamp), 'edad' (int8), and 'genero' (text); 'names' has columns 'id' (int8) and 'uid' (uuid); 'users' has columns 'id' (int8), 'uid' (uuid), 'email' (varchar), 'name' (varchar), and 'created_at' (timestamp).

O también lo podemos ver en la opción



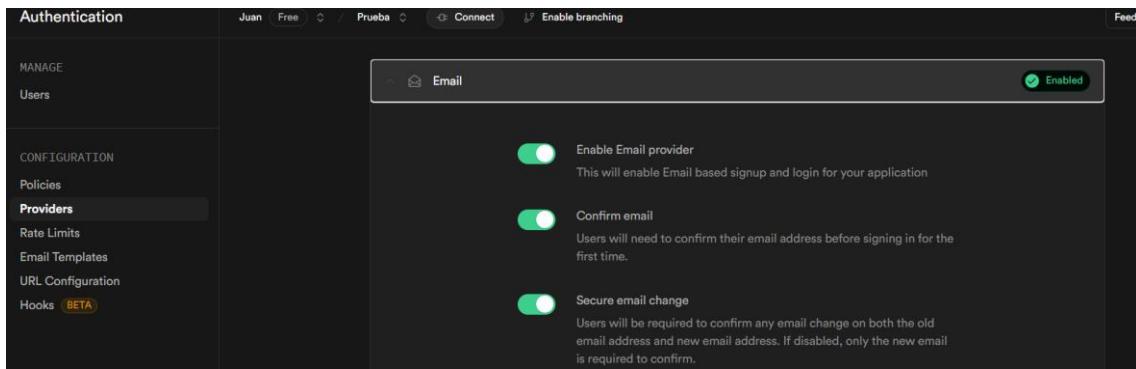
The screenshot shows the Supabase dashboard. On the left, a sidebar menu is open with options: SQL Editor, Database (selected), Authentication, Storage, Edge Functions, Realtime, Advisors, Reports, and Logs. In the main central area, two tables are displayed: 'Peliculas' and 'users'. The 'Peliculas' table has columns: id (int8), user_id (int8), Film name (varchar), Film description (text), created_at (timestamp), edad (int8), and genero (text). The 'users' table has columns: id (int8), uid (uuid), email (varchar), name (varchar), and created_at (timestamp).

Después si volvemos al SupaBase nos vamos a la parte de la izquierda nos iremos a la opción que poner '**Authentication**'. Nos vamos a la opción que pone '**Provider**'.



The screenshot shows the Supabase Authentication section in the sidebar. The sidebar includes icons for Home, Table Editor, SQL Editor, Database, and Authentication (which is highlighted with a dark background).

Aquí dentro podremos ver que la opción de email esta activa y que varias de sus opciones están activas, esto es para que en el código cuando lo iniciemos haga una segunda verificación/autentificación por vía email, y se nos enviara un email para confirmar ese usuario, pero para eso lo veremos más adelante cuando expliquemos el funcionamiento del programa.



10.4 Explicación del Código

10.4.1 Explicación del Ejemplo1 del PPT

Ahora vamos a explicar el código del ejemplo 1 del PowerPoint que nos ha mostrado el profesor, para eso primero tendremos que haber hecho todo lo anterior para poder utilizar la app del ejemplo 1, y en este caso, el script para las tablas será diferente que como hemos hecho anteriormente, lo que haremos será descargarnos un CSV con muchos nombres y los vamos a exportar a nuestra base, para eso primero tendremos que descargarnos el CSV.



Una vez descargado, lo que tendremos que hacer es irnos a nuestro DBeaver y entonces vamos a escribir este script de la misma forma que el anterior, pero esta vez escribiremos este script:

```

CREATE TABLE names (
    id serial primary key,
    fname text,
    lname text
);
ALTER TABLE
    names
ADD COLUMN
    fts tsvector generated always as (to_tsvector('english', fname || ' ' || lname)) stored;
CREATE INDEX name_fts ON names USING gin (fts);

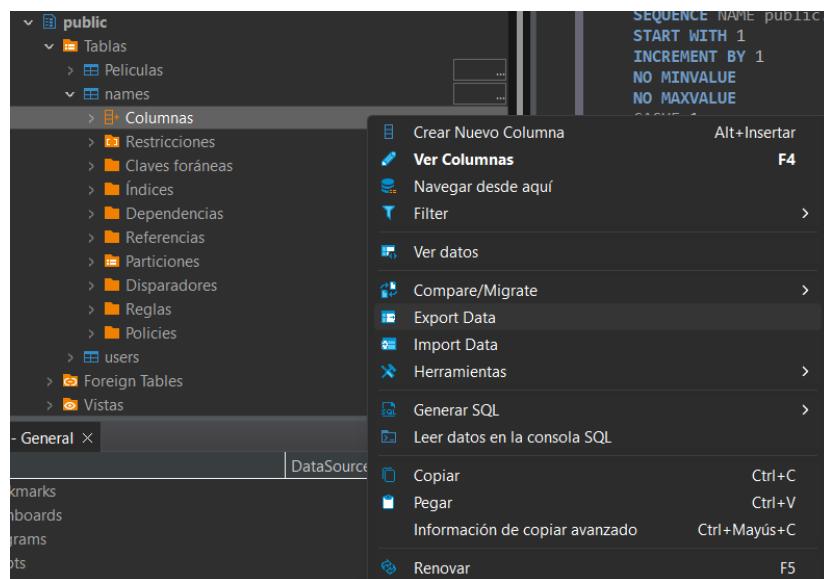
```

```

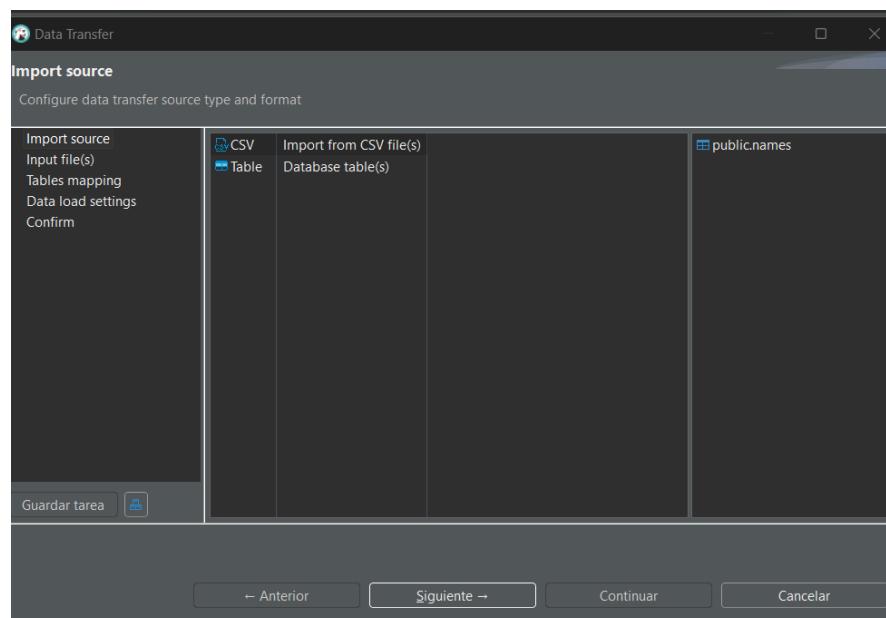
•CREATE TABLE names (
    id serial primary key,
    fname text,
    lname text
);
•ALTER TABLE
    names
ADD COLUMN
    fts tsvector generated always as (to_tsvector('english', fname || ' ' || lname)) stored;
CREATE INDEX name_fts ON names USING gin (fts);
|

```

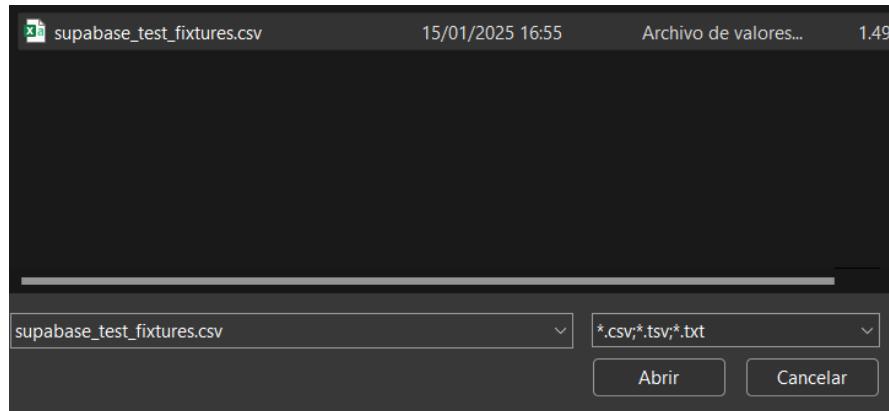
Una vez creado, lo que haremos será irnos a nuestra tabla y darle al desplegable y clic derecho a la opción que pone '**columnas**' y se nos mostrar muchas opciones, pero la que nos interesa en este caso será la que pone '**Import Data**'.



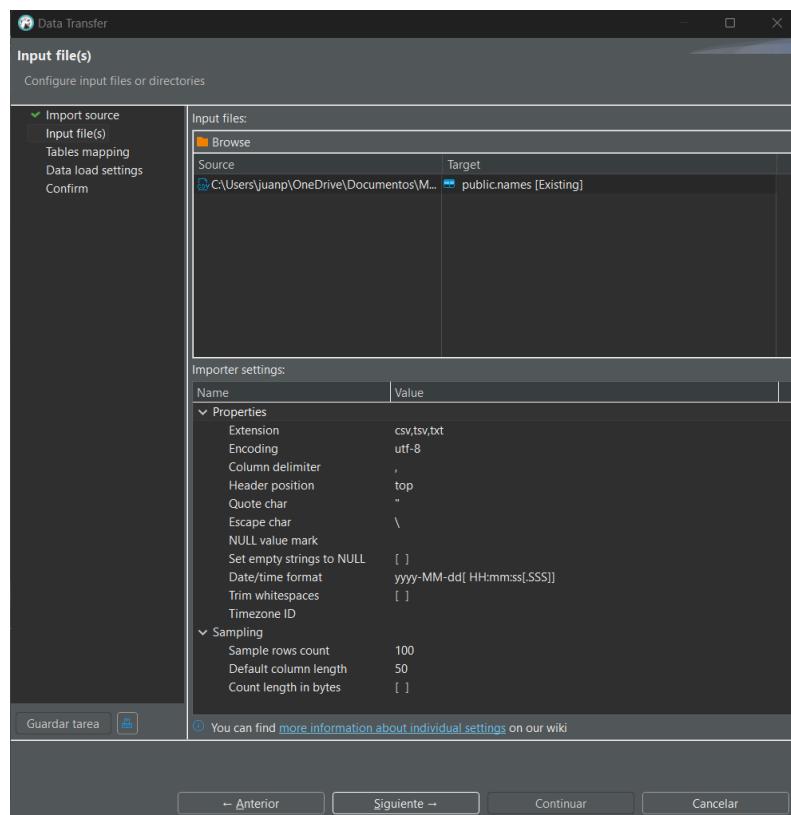
Entonces se nos abrirá una pestaña para seleccionar el formato que queremos importar a nuestra tabla, en este caso seleccionaremos la opción de CSV, y le damos a siguiente.



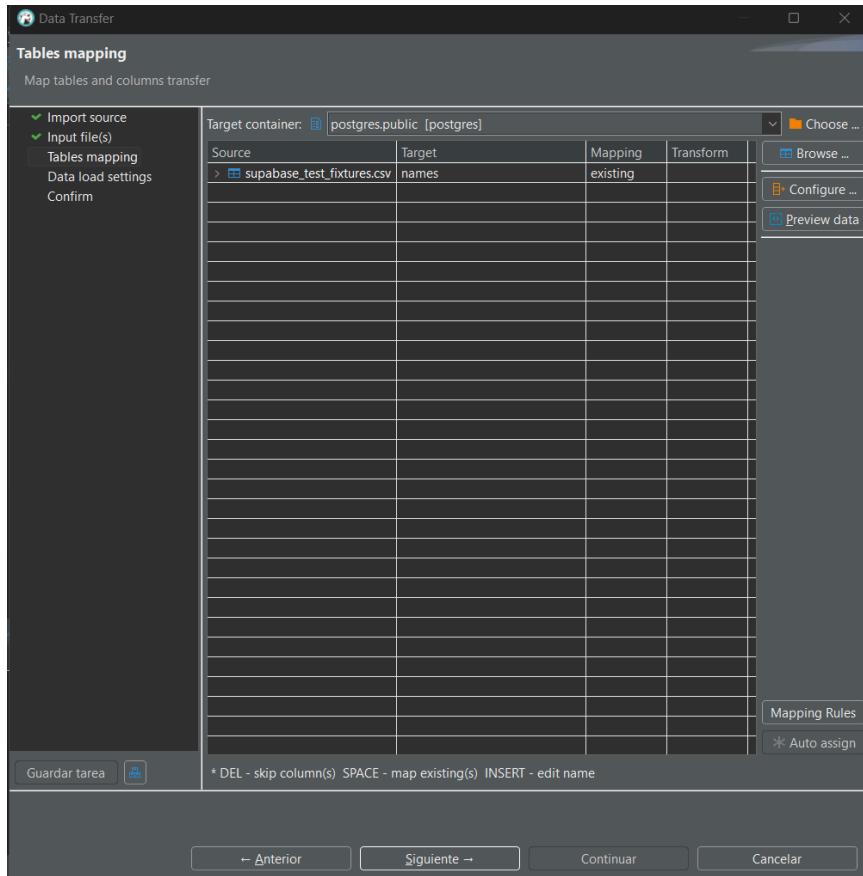
Entonces una vez dado al botón de siguiente, se nos abrirá el explorador de archivos para seleccionar el CSV que queremos importar, lo único que tendremos que hacer es buscarlo y darle a abrir.



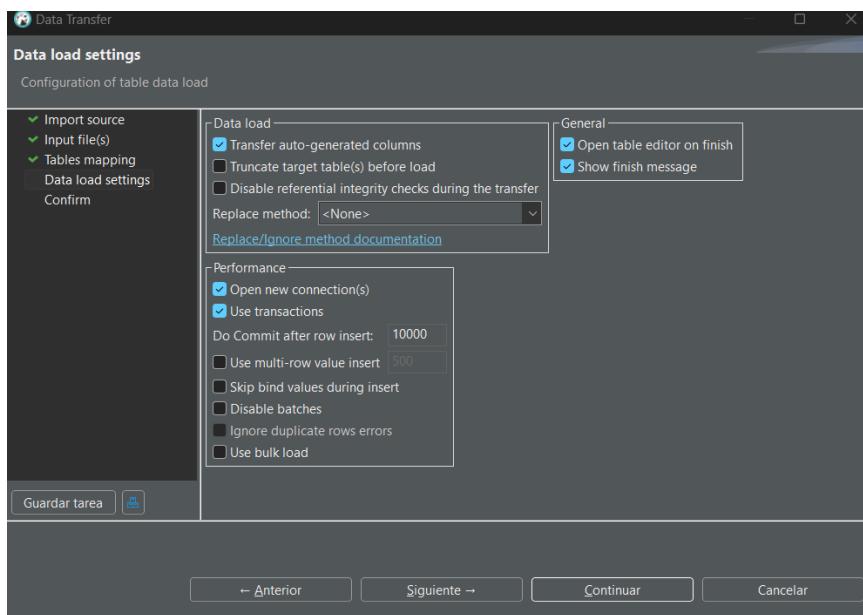
Cuando le demos a abrir se nos mostrara esta pestaña donde simplemente le tenemos que dar a siguiente.



Se nos mostrara esta pestaña y le damos a siguiente.



Y por último le vamos a dar a continuar y ya lo tendremos



Entonces vamos a desplegar la tabla de names, y le vamos a dar clic derecho a la opción de;

Columnas → Ver columnas, y arriba a nos aparecerá varias opciones, la que nos interesa es la que pone datos para ver lo que nos ha importado.

Ingrese parte del nombre de un objeto aquí

- Esquemas
 - auth
 - extensions
 - graphql
 - graphql_public
 - pgbouncer
 - pgsodium
 - pgsodium_masks
- public
 - Tablas
 - Películas
 - names
 - Columnas
 - id** (serial4)
 - fname (text)
 - lname (text)
 - fts (tvector)
 - Restricciones
 - Claves foráneas
 - Índices
 - Dependencias
 - Referencias
 - Particiones
 - Disparadores
 - Reglas

Project - General

Name	Data
Bookmarks	
Dashboards	
Diagrams	
Scripts	

names

Propiedades Datos Diagrama ER

Enter a SQL expression to filter results (use Ctrl+Space)

	id	fname	lname	fts
1	1	Karolina	Ehrman	'ehrman':2 'karolina':1
2	2	Giustina	Roarke	'giustina':1 'roark':2
3	3	Janecka	Sekofski	'janecka':1 'sekofski':2
4	4	Susette	Blake	'blake':2 'susett':1
5	5	Sidoney	Kosey	'kosey':2 'sidone':1
6	6	Kerrin	Lucienne	'kerrin':1 'lucien':2
7	7	Misha	Borrell	'borrel':2 'misha':1
8	8	Fredericka	Aloise	'alois':2 'fredericka':1
9	9	Helsa	Sikorski	'helsa':1 'sikorsk':2
		Vacuva	'vacuva':2	
		Laverne	'kalina':1 'laver':2	
		Carbo	'carbo':2 'gretal':1	
		Arvo	'arvo':2 'wendil':1	
		Behre	'behr':2 'caryl':1	
		Marsden	'aerelia':1 'marsden':2	
		Bonilla	'bonilla':2 'imojean':1	
		Azeria	'azeria':2 'iolet':1	
		Abbot	'abbot':1 'robinia':1	
		Waite	'bett':1 'wait':2	
		Mott	'mott':2 'nyssa':1	
		Guildroy	'guildroy':2 'suzett':1	
		Seumas	'lore':1 'seuma':2	
		LaRue	'laru':2 'marrina':1	
		Merriott	'jorri':1 'merriott':2	
		Thunell	'deirdr':1 'thunel':2	
		Pauly	'dulc':1 'paull':2	
		Gavrila	'gavrila':2 'madil':1	
		McGrody	'jordan':1 'mcgrod':2	
		Sasnett	'sasnett':2 'willetta':1	
		Stover	'barbi':1 'stover':2	
		Earlie	'darcie':1 'earli':2	

Renovar

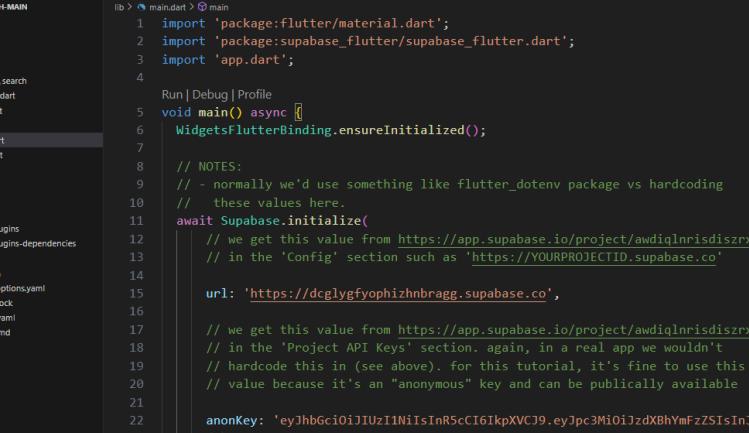
Record

Renovar Save Cancel

Y ahora nos descargamos el proyecto que nos ha pasado el profesor o lo podéis descargar de mi GitHub, donde también estará el CSV por si no os deja descargarlo des del documento y un video del funcionamiento de la app:

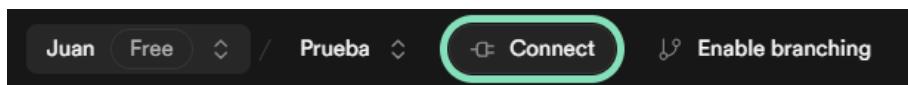
https://github.com/teropod24/SupaBase_search.git

Ahora abrimos el visual studio o el Android, y lo que tendremos que hacer es irnos al main y vamos a modificar las dos variables que se llaman '*url*' y la de '*anonKey*'.

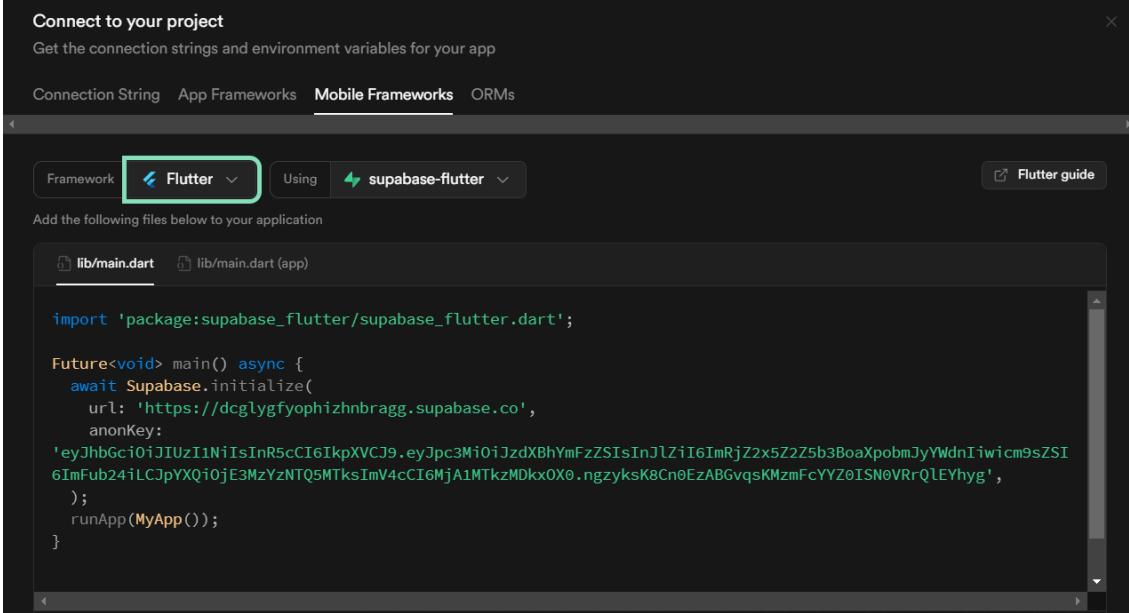


```
lib > main.dart > main
1 import 'package:flutter/material.dart';
2 import 'package:supabase_flutter/supabase_flutter.dart';
3 import 'app.dart';
4
5 void main() async {
6   WidgetsFlutterBinding.ensureInitialized();
7
8   // NOTES:
9   // - normally we'd use something like flutter_dotenv package vs hardcoding
10  // these values here.
11  await Supabase.initialize(
12    // we get this value from https://app.supabase.io/project/awd1qlnr1sdiszrxxwmi/settings/
13    // in the 'Config' section such as 'https://YOURPROJECTID.supabase.co'
14
15    url: 'https://dcglygfyzphiznbbragg.supabase.co',
16
17    // we get this value from https://app.supabase.io/project/awd1qlnr1sdiszrxxwmi/settings/
18    // in the 'Project API Keys' section. again, in a real app we wouldn't
19    // hardcode this in (see above). for this tutorial, it's fine to use this
20    // value because it's an "anonymous" key and can be publicly available
21
22    anonKey: 'eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVJC9.eyJpc3MiOiJzdXBhYmFzZSIsInJlZiI6ImRjZ2x5Z
23
24    debug: true);
25
26 runApp(const App());
```

Y para hacer eso nos volvemos al SupaBase, donde pone '**connect**' que esta al centro arriba.



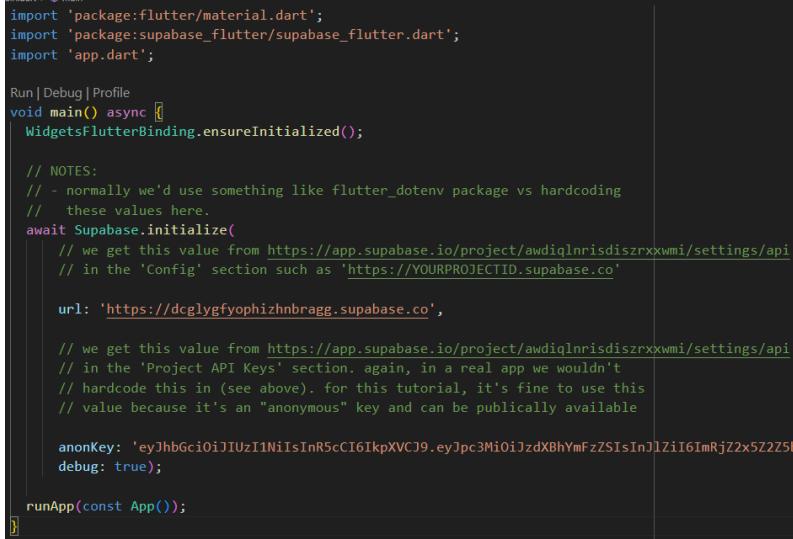
Entonces una vez dentro nos vamos a la opción de '**Mobile Framework**' y dentro vamos a seleccionar en el apartado de '**Framework**' seleccionamos la opción de '**Flutter**' y entonces ya tendremos las calles que tendremos que poner en nuestro código.



```
lib/main.dart
import 'package:supabase_flutter/supabase_flutter.dart';

Future<void> main() async {
  await Supabase.initialize(
    url: 'https://dcglygfophizhnbragg.supabase.co',
    anonKey:
    'eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpc3MiOiJzdXBhYmFzZSIiNjZiI6ImRjZ2x5Z2Z5b3BoaXpobmJyYWdnIiwicm9sZSI6ImFub24iLCJpYXQiOjE3MzYzNTQ5MTksImV4cCI6MjA1MTkzM0x0.0ngzyksK8Cn0EzABGvqsKMzmFcYYZ0ISN0VRrQLEYhyg',
  );
  runApp(MyApp());
}
```

Vale, ahora vamos a explicar un poco el código, en la parte del main, se inicializa **Flutter** y se conecta con **Supabase**, un backend en la nube. La API URL y la clave anónima (anonKey) están **hardcodeadas** (lo ideal sería usar variables de entorno). Se lanza la aplicación con runApp (const App ()).



```
main.dart
import 'package:flutter/material.dart';
import 'package:supabase_flutter/supabase_flutter.dart';
import 'app.dart';

Run | Debug | Profile
void main() async [
  WidgetsFlutterBinding.ensureInitialized();

  // NOTES:
  // - normally we'd use something like flutter_dotenv package vs hardcoding
  // these values here.
  await Supabase.initialize(
    // we get this value from https://app.supabase.io/project/awdiglnrisdisrzxxwmi/settings/api
    // in the 'Config' section such as 'https://YOURPROJECTID.supabase.co'

    url: 'https://dcglygfophizhnbragg.supabase.co',

    // we get this value from https://app.supabase.io/project/awdiglnrisdisrzxxwmi/settings/api
    // in the 'Project API Keys' section. again, in a real app we wouldn't
    // hardcode this in (see above). for this tutorial, it's fine to use this
    // value because it's an "anonymous" key and can be publicly available

    anonKey: 'eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpc3MiOiJzdXBhYmFzZSIiNjZiI6ImRjZ2x5Z2Z5b3BoaXpobmJyYWdnIiwicm9sZSI6ImFub24iLCJpYXQiOjE3MzYzNTQ5MTksImV4cCI6MjA1MTkzM0x0.0ngzyksK8Cn0EzABGvqsKMzmFcYYZ0ISN0VRrQLEYhyg',
    debug: true,
  );

  runApp(const App());
]
```

En el style.dart se definen estilos globales, como fuentes (GoogleFonts), colores (Colors.grey.shade900) y tamaños de texto. Estos estilos se aplican a toda la aplicación.

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

final TextStyle placeholderTextFieldStyle =
    TextStyle(color: Colors.grey.shade400);

final appTheme = ThemeData(
  appBarTheme: AppBarTheme(
    color: Colors.grey.shade900,
    elevation: 0,
    titleTextStyle: TextStyle(
      fontSize: 26.0, fontFamily: GoogleFonts.rubik().fontFamily), // Text
      brightness: Brightness.light,
      primaryColor: Colors.grey.shade900,
      fontFamily: GoogleFonts.outfit().fontFamily,
      // fontFamily: 'Open sans',
      textTheme: TextTheme(
        displayLarge: const TextStyle(fontSize: 36.0),
        displayMedium: const TextStyle([
          fontSize: 22.0, fontWeight: FontWeight.bold, color: Colors.white],
        bodyLarge: const TextStyle([
          fontSize: 26.0, fontWeight: FontWeight.bold, letterSpacing: 1.5),
        bodySmall: TextStyle(
          fontSize: 18.0,
          fontWeight: FontWeight.bold,
          color: Colors.grey.shade400), // TextStyle // TextTheme
      ); // ThemeData
```

En la app.dart, se realiza el MaterialApp que define la estructura general. Usa el **tema global** y la pantalla de inicio es Search (), la cual permite buscar usuarios.

```
app.dart > ...
import 'package:flutter/material.dart';
import 'screens/search/search.dart';
import 'style.dart';

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

  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'Super Search',
      home: const Search(),
      theme: appTheme,
    ); // MaterialApp
  }
}
```

En tile.dart, el tile es un **widget reutilizable** para mostrar cada resultado de la búsqueda. Usa **gradientes aleatorios** para los fondos de las tarjetas.

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

class Tile extends StatelessWidget {
  final String title;

  const Tile(this.title, {super.key});

  @override
  Widget build(BuildContext context) {
    return Stack(children: [
      Container(
        decoration: BoxDecoration(
          gradient: LinearGradient(
            begin: Alignment.topRight,
            end: Alignment.bottomLeft,
            colors: [
              Colors.primaries[Random().nextInt(Colors.primaries.length)],
              Colors.primaries[Random().nextInt(Colors.primaries.length)],
            ],
          )), // LinearGradient // BoxDecoration // Container
      Padding(
        padding: const EdgeInsets.all(10),
        child: Column(
          mainAxisAlignment: MainAxisAlignment.end,
          crossAxisAlignment: CrossAxisAlignment.stretch,
          children: [
            Text(
              title.toUpperCase(),
              style: Theme.of(context).textTheme.displayMedium,
              textAlign: TextAlign.end,
            ), // Text
          ], // Column // Padding
      ), // Stack
    ]);
  }
}
```

En la pantalla de búsqueda de search.dart , el search es un StatefulWidget porque maneja cambios de estado en tiempo real. Muestra un **campo de texto** para buscar usuarios y una lista de resultados. Cada vez que el usuario escribe en el buscador, se ejecuta _onSearchFieldChanged(). Esta función consulta Supabase con _searchUsers(), que usa **búsqueda de texto completo (full-text search)** en Postgres. Los resultados se muestran en tarjetas (Tile).

```
// ignore_for_file: deprecated_member_use

import 'package:flutter/material.dart';
import 'package:supabase_flutter/supabase_flutter.dart';
import 'package:supersearch/screens/search/tile.dart';
import 'package:supersearch/style.dart';

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

  @override
  State<Search> createState() => _SearchState();
}

class _SearchState extends State<Search> {
  List<String>? _results;
  String _input = '';

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: const Text('Search Users'), backgroundColor: Colors.red,
      ), // AppBar
      body: Column(children: [
        Padding(
          padding: const EdgeInsets.symmetric(horizontal: 10),
          child: TextFormField(
            style: Theme.of(context).textTheme.bodyLarge,
            onChanged: _onSearchFieldChanged,
            autocorrect: false,
            autofocus: true,
            decoration: InputDecoration(
              hintText: "Name",
              hintStyle: placeholderTextfieldStyle,
              border: InputBorder.none,
              focusedBorder: InputBorder.none,
              enabledBorder: InputBorder.none,
            ),
          ),
        ),
      ],
    );
  }

  void _onSearchFieldChanged(String value) {
    setState(() {
      _input = value;
      _results = null;
    });
  }

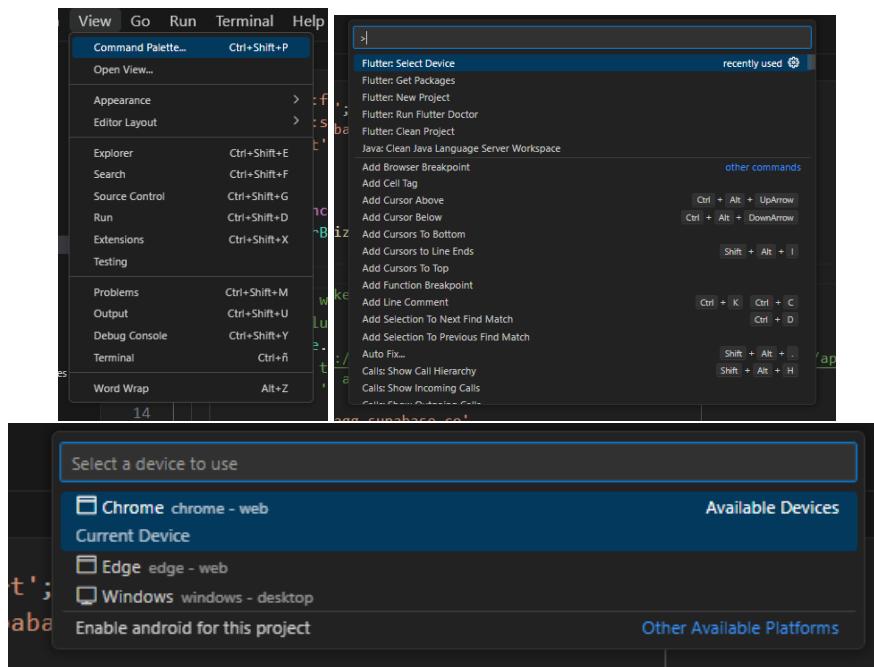
  void _onSearchResults(List<String> results) {
    setState(() {
      _results = results;
    });
  }
}
```

Entonces si iniciamos ahora el programa lo que podemos ver es que hay un buscador de nombres, que serán los que tenemos dentro de la tabla de *name*.

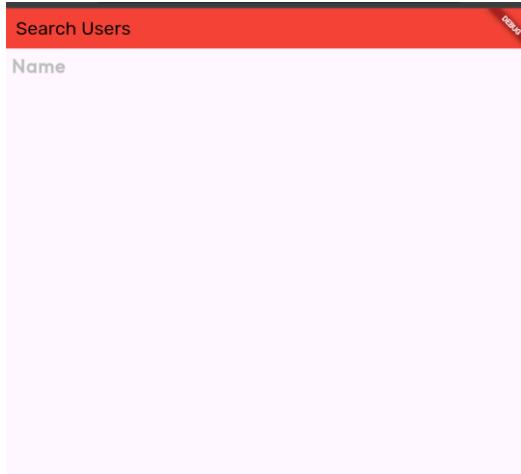
→ ¡Recordatorio, para este caso que funcione el programa con el SupaBase, tendremos que iniciar el programa con el Chrome o el Edge, porque con Windows no te deja! ←

Para seleccionar el modo en que se ejecuta el programa lo podemos hacer en el visual le damos a la opción que tenemos arriba de:

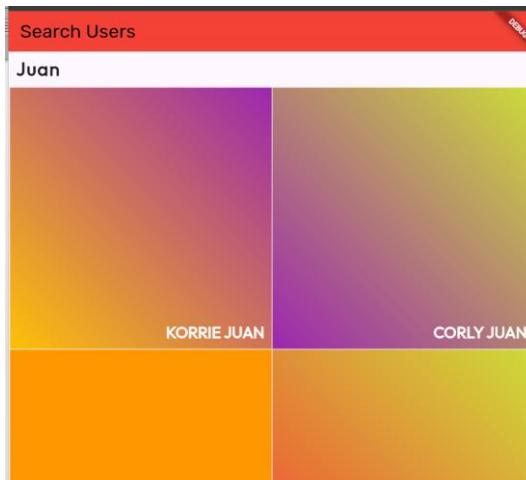
view → Command Palette... → Flutter: Select Device → Chrome/Edge



Entonces se nos mostrara asi, como podemos ver solo hay un buscador por nombres, y si escribimos un nombre automaticamente se nos buscara uno que se parezca.



En este caso hemos puesto Juan por probar y se nos a mostrado varios resultados.



Si no, aquí hay un video de cómo funciona.



Y aquí está el GitHub para poder descargarse el proyecto, con el CSV y el video.

https://github.com/teropod24/SupaBase_search.git

10.4.2 Explicación del Ejemplo2 del PPT

Ahora vamos a explicar el código del ejemplo 2 del PowerPoint que nos ha mostrado el profesor, para eso primero tendremos que haber hecho todo lo anterior para poder utilizar la app del ejemplo 2, vale, pero en este caso lo que tendremos que hacer es crear las dos tablas que hay en el proyecto, que son las de '**user**' y '**Películas**', la de películas es porque tiene que ser diferente al del proyecto, y le añadiremos dos campos más, y tendrán una clave primaria y foránea. Y el script que tendremos que ejecutar esta mas arriba, pero te lo dejo aquí abajo igualmente.

```
CREATE TABLE public."Peliculas" (
    id bigint NOT NULL,
    user_id bigint,
    title character varying,
    description text,
    created_at timestamp without time zone DEFAULT now(),
    edad bigint NOT null,
    genero character varying
);

ALTER TABLE public."Peliculas" OWNER TO postgres;

ALTER TABLE public."Peliculas" ALTER COLUMN id ADD GENERATED BY DEFAULT AS IDENTITY
(
    SEQUENCE NAME public.notes_id_seq
        START WITH 1
        INCREMENT BY 1
        NO MINVALUE
        NO MAXVALUE
        CACHE 1
);

CREATE TABLE public.users (
    id bigint NOT NULL,
    uid uuid DEFAULT gen_random_uuid(),
    email character varying,
    name character varying,
    created_at timestamp without time zone DEFAULT now()
);

ALTER TABLE public.users OWNER TO postgres;

COMMENT ON TABLE public.users IS 'users';

ALTER TABLE public.users ALTER COLUMN id ADD GENERATED BY DEFAULT AS IDENTITY (
    SEQUENCE NAME public.users_id_seq
        START WITH 1
);
```

INCREMENT BY 1

NO MINVALUE

NO MAXVALUE

CACHE 1

);

ALTER TABLE public."Peliculas"

ADD COLUMN user_id bigint,

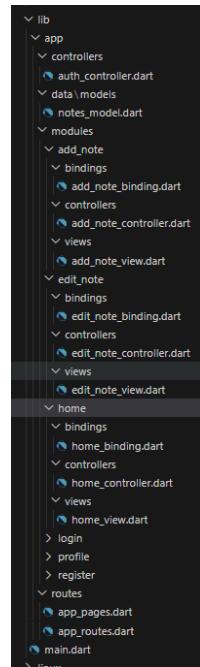
ADD CONSTRAINT fk_peliculas_users FOREIGN KEY (user_id) REFERENCES public.users(id) ON DELETE SET NULL;

```

CREATE TABLE public."Peliculas" (
    id bigint NOT NULL,
    user_id bigint,
    title character varying,
    description text,
    created_at timestamp without time zone DEFAULT now(),
    edad bigint NOT null,
    genero character varying
);
ALTER TABLE public."Peliculas" OWNER TO postgres;
ALTER TABLE public."Peliculas" ALTER COLUMN id ADD GENERATED BY DEFAULT AS IDENTITY (
    SEQUENCE NAME public.notes_id_seq
    START WITH 1
    INCREMENT BY 1
    NO MINVALUE
    NO MAXVALUE
    CACHE 1
);
CREATE TABLE public.users (
    id bigint NOT NULL,
    uid uuid DEFAULT gen_random_uuid(),
    email character varying,
    name character varying,
    created_at timestamp without time zone DEFAULT now()
);
ALTER TABLE public.users OWNER TO postgres;
COMMENT ON TABLE public.users IS 'users';
ALTER TABLE public.users ALTER COLUMN id ADD GENERATED BY DEFAULT AS IDENTITY (
    SEQUENCE NAME public.users_id_seq
    START WITH 1
    INCREMENT BY 1
    NO MINVALUE
    NO MAXVALUE
    CACHE 1
);
ALTER TABLE public."Peliculas"
    ADD COLUMN user_id bigint,
    ADD CONSTRAINT fk_peliculas_users FOREIGN KEY (user_id) REFERENCES public.users(id) ON DELETE SET NULL;

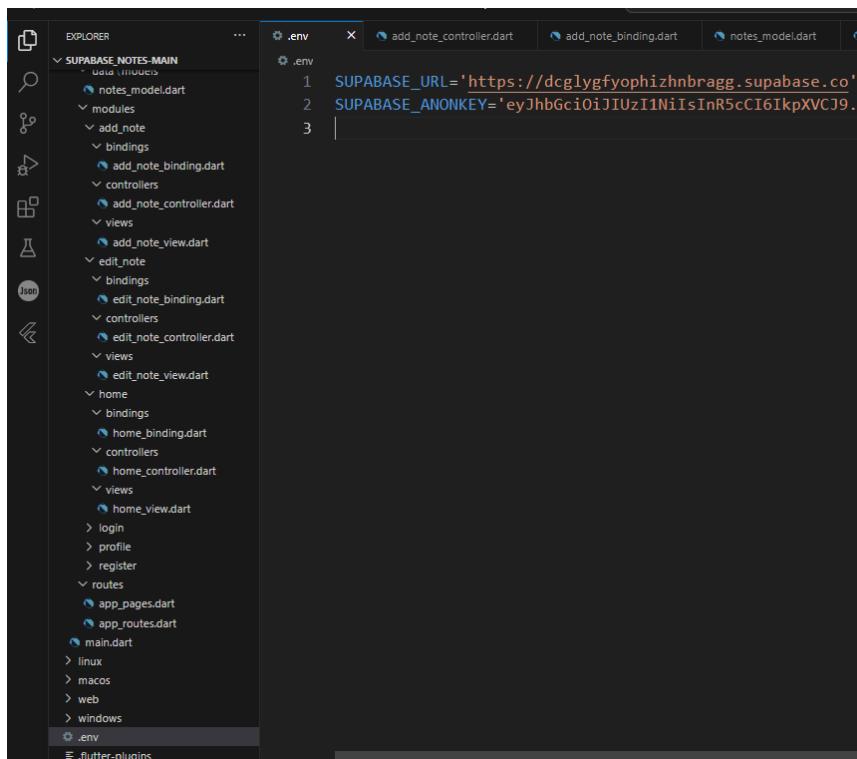
```

Entonces ahora tendremos que descargarnos el proyecto que nos ha dejado de ejemplo el profesor, pero te dejo mi GitHub para que puedas descargar el proyecto directamente y también tiene un video de cómo funciona la aplicación. Si entramos al visual studio, y abrimos el proyecto podemos ver que tenemos muchos archivos, pero los que realmente nos interesa son los de **edit_not**, **add_not** y el **home**, los demás no son necesariamente importantes que hagamos cambios.



Entonces ahora para poner las claves para la conexión con el SupaBase está dentro del archivo que se llama '**.env**' y hacemos lo mismo que previamente, nos iremos a nuestro SupaBase, nos vamos a:

connect → Mobile Framework → Framework → Flutter



Vale, ahora vamos a explicar un poco el código, en la parte del main, se inicializa una aplicación Flutter que usa **Supabase** para la autenticación y la gestión de rutas. Primero, carga las variables de entorno para configurar la conexión con Supabase. Luego, verifica si hay una sesión activa: si el usuario ha iniciado sesión, lo lleva a la pantalla principal; si no, muestra la pantalla de inicio de sesión.

```

3 import 'package:flutter/material.dart';
4 import 'package:flutter_dotenv/flutter_dotenv.dart';
5 import 'package:get/get.dart';
6 import 'package:supabase_notes/app/controllers/auth_controller.dart';
7 import 'package:supabase_flutter/supabase_flutter.dart';
8
9 import 'app/routes/app_pages.dart';
0
Run | Debug | Profile
1 void main() async {
2   WidgetsFlutterBinding.ensureInitialized();
3   await dotenv.load(); //wait load env
4
5   String supaUri = dotenv.get('SUPABASE_URL'); //get env key
6   String supaAnon = dotenv.get('SUPABASE_ANONKEY');
7
8   Supabase supaProvider = await Supabase.initialize(
9     url: supaUri,
10    anonKey: supaAnon,
11  );
12
13   final authC = Get.put(AuthController(), permanent: true);
14   runApp(
15     GetMaterialApp(
16       title: "Application",
17       initialRoute: supaProvider.client.auth.currentUser == null
18         ? Routes.LOGIN
19         : Routes.HOME, //cek login current user
20       getPages: AppPages.routes,
21       debugShowCheckedModeBanner: false,
22     ),
23   );
24 }

```

Este controlador maneja la sesión del usuario, permitiendo el cierre automático de sesión después de una hora de inactividad. Utiliza un temporizador que se cancela y reinicia cada vez que se realiza una acción. Si el usuario está inactivo por más de 3600 segundos (1 hora), la sesión se cierra automáticamente y redirige a la pantalla de inicio de sesión.

```

import 'package:get/get.dart';
import 'package:supabase_notes/app/routes/app_pages.dart';
import 'dart:async';

import 'package:supabase_flutter/supabase_flutter.dart';

class AuthController extends GetxController {
  Timer? authTimer;
  SupabaseClient client = Supabase.instance.client;
  Future<void> autoLogout() async {
    if (authTimer != null) {
      authTimer!.cancel();
    }

    authTimer = Timer(const Duration(seconds: 3600), () async {
      await client.auth.signOut();
      Get.offAllNamed(Routes.LOGIN);
    }); // after login will run auto logout after 1 hours // Tim
  }

  Future<void> resetTimer() async {
    if (authTimer != null) [
      authTimer!.cancel();
      authTimer = null;
    ]
  }
}

```

En esta parte, lo que podemos ver es un modelo de datos que representa las películas, con campos como *id*, *userId*, *title*, *description*, *edad* y *género*. Tiene métodos para convertir datos

de un formato JSON a un objeto Peliculas y viceversa, permitiendo manejar fácilmente las respuestas de la base de datos.

```
class Peliculas {
    int? id;
    int? userId;
    String? title;
    String? description;
    String? createdAt;
    int? edad;
    String? genero;

    Peliculas({this.id, this.userId, this.title, this.description, this.createdAt, this.edad, this.genero});

    Peliculas.fromJson(Map<String, dynamic> json) {
        id = json['id'];
        userId = json['user_id'];
        title = json['Film name'];
        description = json['film description'];
        createdAt = json['created_at'];
        edad = json['edad'];
        genero = json['genero'];
    }

    Map<String, dynamic> toJson() {
        final data = <String, dynamic>();
        data['id'] = id;
        data['user_id'] = userId;
        data['Film name'] = title;
        data['film description'] = description;
        data['created_at'] = createdAt;
        data['edad'] = edad;
        data['genero'] = genero;
        return data;
    }

    static List<Peliculas> fromJsonList(List? data) {
        if (data == null || data.isEmpty) return [];
        return data.map((e) => Peliculas.fromJson(e)).toList();
    }
}
```

Este código de aquí abajo es una clase que se encarga de enlazar el controlador AddNoteController con la vista correspondiente. Usando el método Get.lazyPut, asegura que el controlador se cree solo cuando sea necesario, lo cual optimiza el rendimiento de la app.

```
import 'package:get/get.dart';

import '../controllers/add_note_controller.dart';

class AddNoteBinding extends Bindings {
    @override
    void dependencies() {
        Get.lazyPut<AddNoteController>(
            () => AddNoteController(),
        );
    }
}
```

Este controlador gestiona la lógica para agregar una nueva película a la base de datos. Recibe los datos de los campos de texto, obtiene el userId del usuario autenticado y los inserta en la tabla de películas en Supabase. También maneja el estado de carga mientras se realiza la operación.

```
import 'package:flutter/cupertino.dart';
import 'package:get/get.dart';
import 'package:supabase_flutter/supabase_flutter.dart';

class AddNoteController extends GetxController {
    RxBool isLoading = false.obs;
    RxBool isHidden = true.obs;
    TextEditingController titleC = TextEditingController();
    TextEditingController descC = TextEditingController();
    TextEditingController edad = TextEditingController();
    TextEditingController genero = TextEditingController();
    SupabaseClient client = Supabase.instance.client;

    Future<bool> addNote() async {
        if (titleC.text.isNotEmpty && descC.text.isNotEmpty) {
            isLoading.value = true;
            List<dynamic> res = await client
                .from("users")
                .select("id")
                .match({"uid": client.auth.currentUser!.id});
            Map<String, dynamic> user = (res).first as Map<String, dynamic>;
            int id = user["id"]; //get and match user id before insert to db
            await client.from("Películas").insert({
                "user_id": id, //insert data with user id as foreign key
                "created_at": DateTime.now().toIso8601String(),
                "Film name": titleC.text,
                "Film description": descC.text,
                "edad" : int.parse(edad.text),
                "genero" : genero.text,
            });
            return true;
        } else {
            return false;
        }
    }
}
```

Es la interfaz de usuario que permite al usuario ingresar los detalles de una nueva película (nombre, descripción, edad, género). Los campos son gestionados por el AddNoteController, y el botón de agregar película muestra un estado de carga mientras se guarda la información. Cuando se agrega con éxito, actualiza la lista de películas y regresa a la vista anterior.

```
// ignore_for_file: must_be_immutable

import 'package:flutter/material.dart';
import 'package:get/get.dart';
import 'package:supabase_notes/app/modules/home/controllers/home_controller.dart';
import '../controllers/add_note_controller.dart';

class AddNoteView extends GetView<AddNoteController> {
    HomeController homeC = Get.find();

    AddNoteView({super.key}); // get controller from another controller
    @override
    Widget build(BuildContext context) {
        return Scaffold(
            appBar: AppBar(
                title: const Text('Add Film'),
                centerTitle: true,
            ), // AppBar
            body: ListView(
                padding: const EdgeInsets.all(20),
                children: [
                    TextField(
                        controller: controller.titleC,
                        decoration: const InputDecoration(
                            labelText: "Film name",
                            border: OutlineInputBorder(),
                        ), // InputDecoration
                    ), // TextField
                    const SizedBox(
                        height: 25,
                    ), // SizedBox
                    TextField(
                        controller: controller.descC,
                        decoration: const InputDecoration(
                            labelText: "Film description",
                            border: OutlineInputBorder(),
                        ), // InputDecoration
                    ), // TextField
                    const SizedBox(
                        height: 25,
                    ), // SizedBox
                    TextField(
                        controller: controller.edad,
                        decoration: const InputDecoration(
                            labelText: "Film age",
                        ),
                    ), // TextField
                ],
            ),
        );
    }
}
```

Esta clase es responsable de enlazar el controlador EditNoteController con la vista de edición de notas. Utiliza Get.lazyPut para crear el controlador solo cuando se necesite, lo que optimiza el rendimiento de la aplicación al evitar la creación innecesaria de objetos.

```
> modules > edit_note > bindings > edit_note_binding.dart > ...
import 'package:get/get.dart';

import '../controllers/edit_note_controller.dart';

class EditNoteBinding extends Bindings {
    @override
    void dependencies() {
        Get.lazyPut<EditNoteController>(
            () => EditNoteController(),
        );
    }
}
```

El controlador maneja la lógica para editar una película en la base de datos. Permite modificar campos como el nombre, la descripción, la edad y el género de una película. Si los campos no están vacíos, realiza una actualización en la base de datos y, durante este proceso, muestra un estado de carga.

```
import 'package:flutter/cupertino.dart';
import 'package:get/get.dart';
import 'package:supabase_flutter/supabase_flutter.dart';

class EditNoteController extends GetxController {
    RxBool isLoading = false.obs;
    RxBool isHidden = true.obs;
    TextEditingController titleC = TextEditingController();
    TextEditingController descC = TextEditingController();
    TextEditingController edad = TextEditingController();
    TextEditingController gender = TextEditingController();
    SupabaseClient client = Supabase.instance.client;

    Future<bool> editNote(int id) async {
        if (titleC.text.isNotEmpty && descC.text.isNotEmpty) {
            isLoading.value = true;
            await client
                .from("Peliculas")
                .update({ "Film name": titleC.text, "Film description": descC.text,
                    "edad": int.parse(edad.text), "genero": gender.text }).match({
                    "id": id,
                });
            return true;
        } else {
            return false;
        }
    }
}
```

Es la interfaz donde el usuario puede editar los detalles de una película existente. Los campos se llenan con los datos actuales de la película, y el usuario puede modificarlos. Al presionar el botón de "Editar", los cambios se guardan en la base de datos. También tiene un estado de carga mientras se realiza la actualización.

```
import 'package:flutter/material.dart';
import 'package:get/get.dart';
import 'package:supabase_notes/app/data/models/note_model.dart';
import 'package:supabase_notes/app/modules/home/controllers/home_controller.dart';

import '../controllers/edit_note_controller.dart';

class EditNoteView extends GetView<EditNoteController> {
    Peliculas note = Get.arguments;
    HomeController homeC = Get.find();

    EditNoteView({super.key});
    @override
    Widget build(BuildContext context) {
        controller.titleC.text = note.title!;
        controller.descC.text = note.description!;
        controller.edad.text = note.edad!.toString();
        controller.gender.text = note.genero!;

        return Scaffold(
            appBar: AppBar(
                title: const Text('Edit Film'),
                centerTitle: true,
            ),
            body: ListView(
                padding: const EdgeInsets.all(20),
                children: [
                    TextField(
                        controller: controller.titleC,
                        decoration: const InputDecoration(
                            labelText: "Film name",
                            border: OutlineInputBorder(),
                        ),
                    ),
                    const SizedBox(
                        height: 25,
                    ),
                    TextField(
                        controller: controller.descC,
                        decoration: const InputDecoration(
                            labelText: "Film description",
                            border: OutlineInputBorder(),
                        ),
                    ),
                    const SizedBox(

```

Este Binding se encarga de injectar el HomeController en la vista correspondiente. Al usar Get.lazyPut, asegura que el controlador se cree solo cuando la vista lo necesite, mejorando el rendimiento al evitar instancias innecesarias del controlador.

```
import 'package:get/get.dart';

import '../controllers/home_controller.dart';

class HomeBinding extends Bindings {
  @override
  void dependencies() {
    Get.lazyPut<HomeController>(
      () => HomeController(),
    );
  }
}
```

Este controlador maneja la obtención y eliminación de las películas del usuario desde la base de datos. Primero obtiene el userId del usuario actual y luego carga todas las películas asociadas a ese userId. También tiene la capacidad de eliminar una película y actualizar la lista.

```
import 'package:get/get.dart';
import 'package:supabase_notes/app/data/models/notes_model.dart';
import 'package:supabase_flutter/supabase_flutter.dart';

class HomeController extends GetxController {
  RxList allNotes = List<Peliculas>.empty().obs;
  SupabaseClient client = Supabase.instance.client;

  Future<void> getAllNotes() async {
    List<dynamic> res = await client
      .from("users")
      .select("id")
      .match({"uid": client.auth.currentUser!.id});
    Map<String, dynamic> user = (res).first as Map<String, dynamic>;
    int id = user["id"]; //get user id before get all notes data
    var notes = await client.from("Películas").select().match(
      {"user_id": id}, //get all notes data with match user id
    );
    List<Peliculas> notesData = Peliculas.fromJsonList((notes as List));
    allNotes(notesData);
    allNotes.refresh();
  }

  Future<void> deleteNote(int id) async {
    await client.from("Películas").delete().match(
      {"id": id,
    });
    getAllNotes();
  }
}
```

Es la pantalla principal de la aplicación donde se muestran todas las películas asociadas al usuario. Utiliza un FutureBuilder para cargar las notas y un Obx para actualizar la vista en tiempo real cuando se agregan o eliminan películas. Además, permite editar o eliminar películas y navegar a otras secciones como el perfil del usuario.

```
import 'package:flutter/material.dart';
import 'package:get/get.dart';
import 'package:supabase_notes/app/data/models/notes_model.dart';
import 'package:supabase_notes/app/routes/app_pages.dart';

import '../controllers/home_controller.dart';

class HomeView extends GetView<HomeController> {
  const HomeView({super.key});

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: const Text('HOME'),
        centerTitle: true,
        actions: [
          IconButton(
            onPressed: () async {
              Get.toNamed(Routes.PROFILE);
            },
            icon: const Icon(Icons.person),
          ) // IconButton
        ],
      ), // AppBar
      body: FutureBuilder<
        Future<List<Película>>(
          future: controller.getAllNotes(),
          builder: (context, snapshot) {
            if (snapshot.connectionState == ConnectionState.waiting) {
              return const Center(child: CircularProgressIndicator());
            }
            return Obx(() => controller.allNotes.isEmpty
              ? const Center(
                  child: Text("NO FILM"),
                ) // Center
              : ListView.builder(
                  itemCount: controller.allNotes.length,
                  itemBuilder: (context, index) {
                    Película note = controller.allNotes[index];
                    return ListTile(
                      onTap: () => Get.toNamed(
                        Routes.EDIT_NOTE,
                        arguments: note,
                      ),
                      leading: CircleAvatar(
                        child: Text("t${note.id}"),
                      ), // CircleAvatar
                    );
                  },
                );
          },
        ),
      ),
    );
  }
}
```

Esta clase define las rutas y las vistas de la aplicación utilizando el paquete get. Establece la ruta inicial de la aplicación y configura cada página, asociando las rutas con las vistas y los bindings correspondientes para la gestión de dependencias.

```
import 'package:get/get.dart';

import 'package:supabase_notes/app/modules/add_note/bindings/add_note_binding.dart';
import 'package:supabase_notes/app/modules/add_note/views/add_note_view.dart';
import 'package:supabase_notes/app/modules/edit_note/bindings/edit_note_binding.dart';
import 'package:supabase_notes/app/modules/edit_note/views/edit_note_view.dart';
import 'package:supabase_notes/app/modules/home/bindings/home_binding.dart';
import 'package:supabase_notes/app/modules/home/views/home_view.dart';
import 'package:supabase_notes/app/modules/login/bindings/login_binding.dart';
import 'package:supabase_notes/app/modules/login/views/login_view.dart';
import 'package:supabase_notes/app/modules/profile/bindings/profile_binding.dart';
import 'package:supabase_notes/app/modules/profile/views/profile_view.dart';
import 'package:supabase_notes/app/modules/register/bindings/register_binding.dart';
import 'package:supabase_notes/app/modules/register/views/register_view.dart';

part 'app_routes.dart';

class AppPages {
  AppPages._();

  static const INITIAL = Routes.HOME;

  static final routes = [
    GetPage(
      name: _Paths.HOME,
      page: () => const HomeView(),
      binding: HomeBinding(),
    ), // GetPage
    GetPage(
      name: _Paths.LOGIN,
      page: () => LoginView(),
      binding: LoginBinding(),
    ), // GetPage
    GetPage(
      name: _Paths.REGISTER,
      page: () => const RegisterView(),
      binding: RegisterBinding(),
    ), // GetPage
    GetPage(
      name: _Paths.PROFILE,
      page: () => ProfileView(),
      binding: ProfileBinding(),
    ), // GetPage
    GetPage(
      name: _Paths.ADD_NOTE,
      page: () => AddNoteView(),
      binding: AddNoteBinding(),
    ), // GetPage
  ];
}
```

Routes es una clase que contiene constantes para las rutas de la aplicación. Estas constantes se usan en otras partes del código para hacer referencia a las rutas de manera consistente, evitando errores por cadenas de texto mal escritas.

```
part of 'app_pages.dart';
// DO NOT EDIT. This is code generated via package:get_cli/get_cli.dart

abstract class Routes {
  Routes._();

  static const HOME = _Paths.HOME;
  static const LOGIN = _Paths.LOGIN;
  static const REGISTER = _Paths.REGISTER;
  static const PROFILE = _Paths.PROFILE;
  static const ADD_NOTE = _Paths.ADD_NOTE;
  static const EDIT_NOTE = _Paths.EDIT_NOTE;
}

abstract class _Paths {
  static const HOME = '/home';
  static const LOGIN = '/login';
  static const REGISTER = '/register';
  static const PROFILE = '/profile';
  static const ADD_NOTE = '/add-note';
  static const EDIT_NOTE = '/edit-note';
}
```

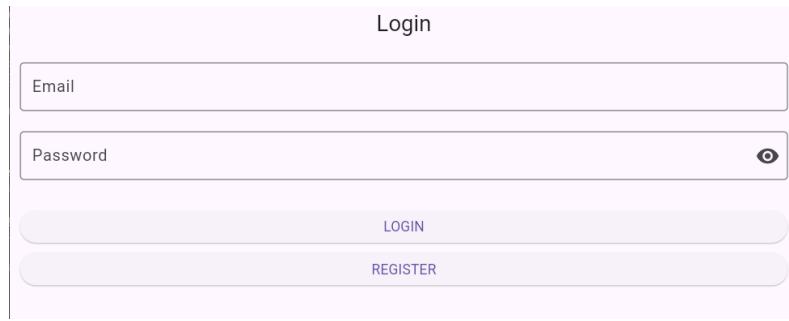
Y el main inicializa una aplicación Flutter que usa **Supabase** para la autenticación y **GetX** para la gestión de estado y navegación. Carga variables de entorno desde un archivo .env para obtener las credenciales de Supabase. Luego, configura Supabase con la URL y clave anónima,

e instancia un controlador de autenticación (AuthController). Finalmente, inicia la aplicación con **GetMaterialApp**, determinando si mostrar la pantalla de **inicio de sesión** o **home**, según el estado del usuario.

```

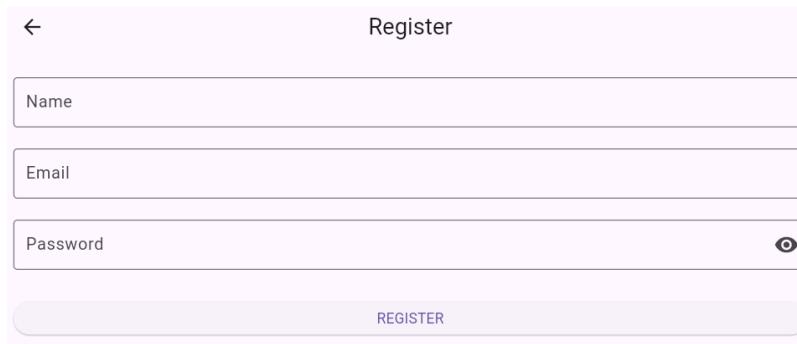
1 // ignore_for_file: unused_local_variable
2
3 import 'package:flutter/material.dart';
4 import 'package:flutter_dotenv/flutter_dotenv.dart';
5 import 'package:get/get.dart';
6 import 'package:supabase_notes/app/controllers/auth_controller.dart';
7 import 'package:supabase_flutter/supabase_flutter.dart';
8
9 import 'app/routes/app_pages.dart';
10
11 Run | Debug | Profile
12 void main() async {
13   WidgetsFlutterBinding.ensureInitialized();
14   await dotenv.load(); //wait load env
15
16   String supaUri = dotenv.get('SUPABASE_URL'); //get env key
17   String supaAnon = dotenv.get('SUPABASE_ANONKEY');
18
19   Supabase supaProvider = await Supabase.initialize(
20     url: supaUri,
21     anonKey: supaAnon,
22   );
23
24   final authC = Get.put(AuthController(), permanent: true);
25   runApp(
26     GetMaterialApp(
27       title: "Application",
28       initialRoute: supaProvider.client.auth.currentUser == null
29         ? Routes.LOGIN
30         : Routes.HOME, //cek login current user
31       getPages: AppPages.routes,
32       debugShowCheckedModeBanner: false,
33     ),
34   );
35 }
```

Entonces si iniciamos el programa lo primero que tendremos es un login para acceder o registrarse, como no tenemos ninguna cuenta de momento le daremos a registrarse.



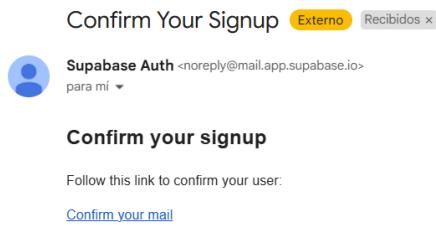
The image shows a mobile application's login screen. At the top, it says "Login". Below that are two input fields: one for "Email" and one for "Password" with an eye icon to show/hide the text. At the bottom are two buttons: "LOGIN" and "REGISTER".

Entonces, dentro tendrás que completar un par de campos incluidos un correo real y una contraseña de mínimo 6 caracteres.

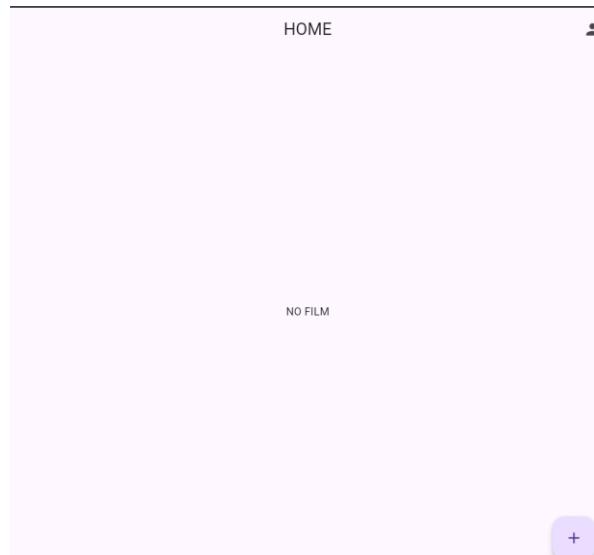


The image shows a mobile application's register screen. At the top, there is a back arrow icon and the word "Register". Below that are three input fields: "Name", "Email", and "Password" with an eye icon. At the bottom is a single "REGISTER" button.

Entonces cuando nos hayamos registrado nos tendremos que ir a nuestro correo que hemos puesto y nos llegara un correo de confirmación, simplemente confirmas y después nos vamos otra vez al login



Entonces una vez confirmado el correo, cuando entremos en el programa podremos ver que hay un botón para añadir películas y otro para ver la información de tu perfil.



Entonces si le damos al botón de abajo a la derecha, vamos a añadir una nueva película, y dentro tendremos que llenar los campos necesarios.



Add Film

Film name

Film description

Film age

Film gender

Add note

Entonces una vez añadida la película, ahora podremos editara o borrara des de la página principal.



Aquí te deja un video para que puedas ver cómo funciona el programa.



Video_Peliculas.mp4

Y si no, aquí te deja un enlace de mi GitHub de este proyecto donde te puedes descargar todo el material para esta explicación incluido el video:

https://github.com/teropod24/SupaBase_Add_Films.git