Laboratorio3

Link al repositorio: https://github.com/TheMob009/3479A321_2023479014

Descripción de lo realizado:

En este laboratorio se trabajo con las estructuras y las dependencias de un proyecto Flutter, logrando lo siguiente:

-Gestión Branch en Git: se creó una nueva Branch “dependencias” a partir del repositorio actual.

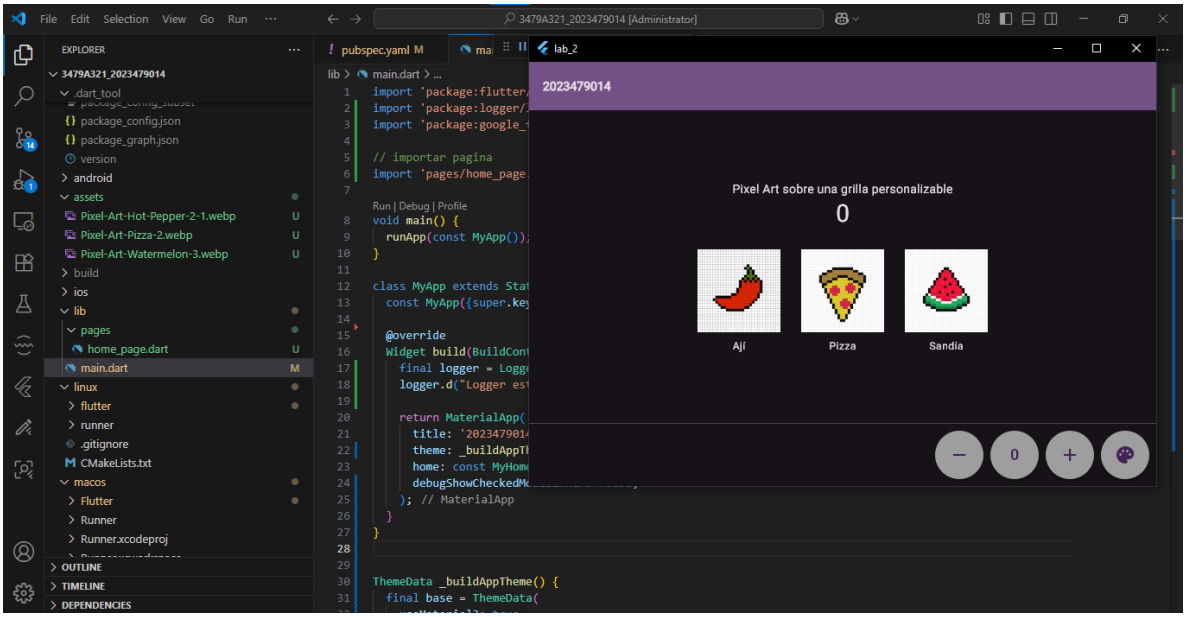
-Incorporación de Logger: se incorpora la herramienta Logger a las dependencias, para así registrar mensajes en consola.

-Manejo de Assets: se creo la carpeta assets, para utilizar distintos recursos durante la ejecución del programa, en este caso, imágenes.

-Visualización de Imágenes: se mostraron imágenes con Image.asset, organizadas en un Row dentro de un SingleChildScrollView.

-Personalización del tema: se incluyo una nueva dependencia, google_fonts, y se configuró el ThemeData, con nuevos colores y fuentes.

-Organización del proyecto: se creo una nueva carpeta, lib/pages/, y se movió la pantalla de Home actual a un nuevo archivo.



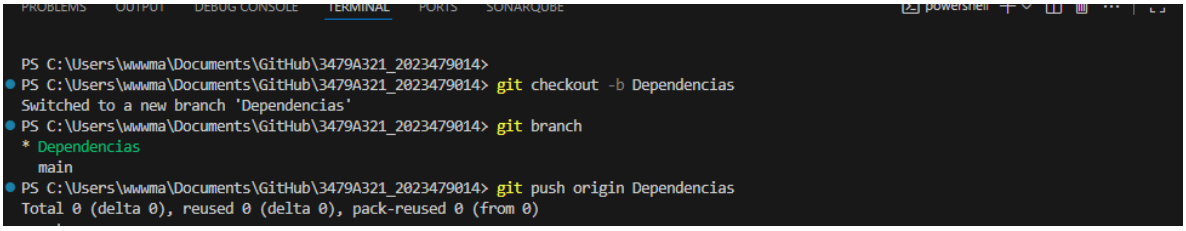
Como inicio de este laboratorio se nos solicita crear una nueva Branch de nuestro repositorio mediante comando de Git por consola, para esto ocupamos 3 comandos:

-git checkout -b NombreBranch

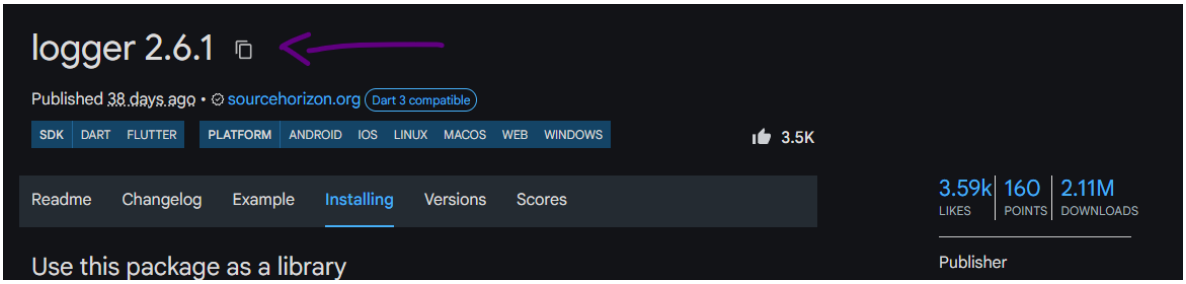
-git branch

-git push origin NombreBranch

En este caso llamaremos a esta nueva rama “Dependencias”



En el segundo apartado de este laboratorio se nos solicita incluir una herramienta para registrar los mensajes del registro. Esto lo logramos mediante la instalación de la herramienta/librería “Logger”, para instalar esta debemos dirigirnos aquí, y copiar el código que se encuentra en el tope de la página.



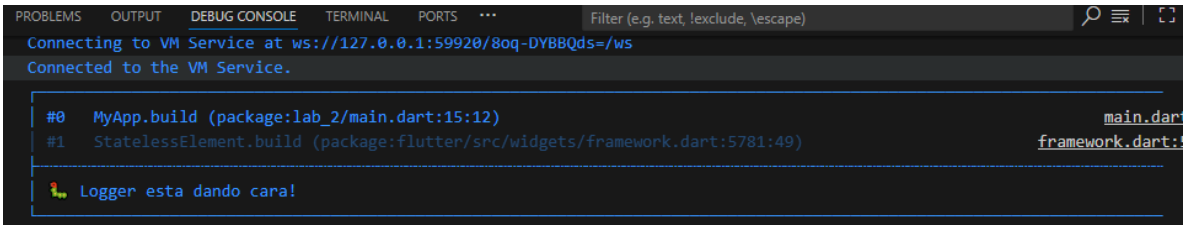
Con el código copiado, nos dirigimos a el archivo “pubspec.yaml” que se encuentra en nuestro repositorio. Dentro de este archivo, nos dirigimos a “dependencies” y pegamos debajo de cupertinoicons.

```yaml
environment:
  sdk: ^3.8.1

# Dependencies specify other packages that your package needs in order to work.
# To automatically upgrade your package dependencies to the latest versions
# consider running `flutter pub upgrade --major-versions`. Alternatively,
# dependencies can be manually updated by changing the version numbers below to
# the latest version available on pub.dev. To see which dependencies have newer
# versions available, run `flutter pub outdated`.
dependencies:
  flutter:
    sdk: flutter

  # The following adds the Cupertino Icons font to your application.
  # Use with the CupertinoIcons class for iOS style icons.
  cupertino_icons: ^1.0.8
  logger: ^2.6.1
```

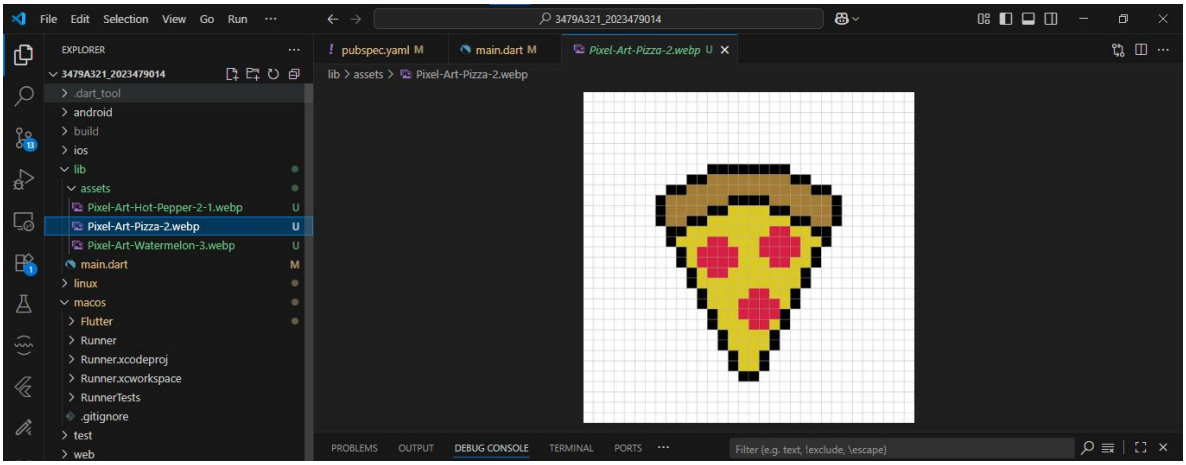
Guardamos el archivo y ya estará instalado el logger en este repositorio. Ahora nos dirigimos a nuestro main, y añadimos las siguientes líneas, esto importara el paquete de Logger, y luego la función principal junto a un mensaje para testear si ha funcionado:

main.dart > main

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

Run | Debug | Profile
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) {
    var logger = Logger();
    logger.d("Logger esta dando cara!");
```



Ahora, debemos de agregar imágenes como assets para usarlas luego en el programa, para esto debemos de crear una nueva carpeta llamada “Assets” dentro de la carpeta “lib”, luego dentro de esta pegamos las imágenes a usar, se nos entregó tres imágenes para este laboratorio:



Así como están no podremos utilizarlas, para arreglar eso nos dirigiremos nuevamente a “pubspec.yaml”, dentro de la sección dedicada a los paquetes de flutter, añadiremos las siguientes líneas:

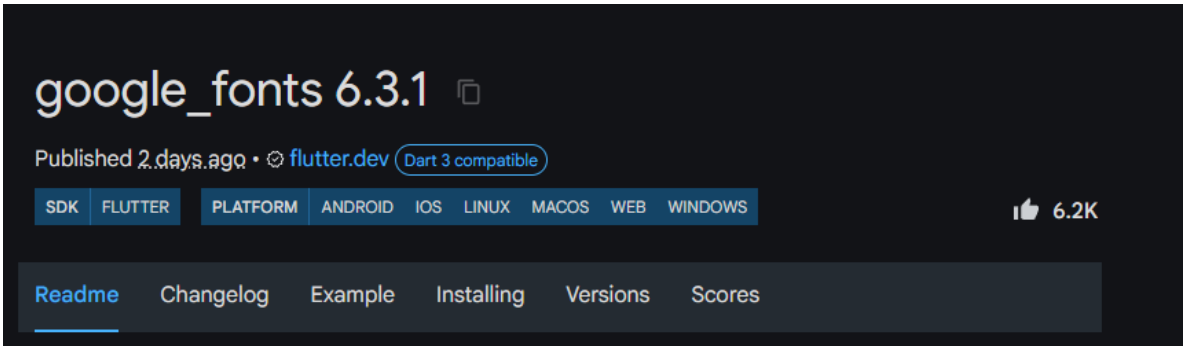
```
# The following section is specific to Flutter packages.
flutter:

  # The following line ensures that the Material Icons font is
  # included with your application, so that you can use the icons in
  # the material Icons class.
  uses-material-design: true


assets:
  -assets/Pixel-Art-Hot-Pepper-2-1.webp
  -assets/Pixel-Art-Pizza-2.webp
  -assets/Pixel-Art-Watermelon-3.webp
  # To add assets to your application, add an assets section, like this:
  # assets:
  #   - images/a_dot_burr.jpeg
  #   - images/a_dot_ham.jpeg
```

De esta forma ya las podremos ocupar en la aplicación, para hacer esto, hacemos uso de “Image.asset()”, dentro de los paréntesis pondremos el nombre de la imagen que queremos utilizar. Ahora, para mostrar las imágenes de forma horizontal debajo del texto central, tuve que crear un “Widget” para mostrar imágenes y luego hacer uso de SingleChildScrollView para mostrarlas horizontalmente:

```
// Funcion utilizada para mostrar las imagenes correctamente
Widget _assetThumb(String path, String label) {
  return Padding(
    padding: const EdgeInsets.symmetric(horizontal: 12),
    child: Column(
      children: [
        Image.asset(
          path,
          width: 96,
          height: 96,
          fit: BoxFit.contain,
          // Por si escribi mal la ruta:
          errorBuilder: (context, error, stack) => const Icon(Icons.broken_image, size: 96),
        ), // Image.asset
        const SizedBox(height: 8),
        Text(label),
      ],
    ), // Column
  ); // Padding
}
```

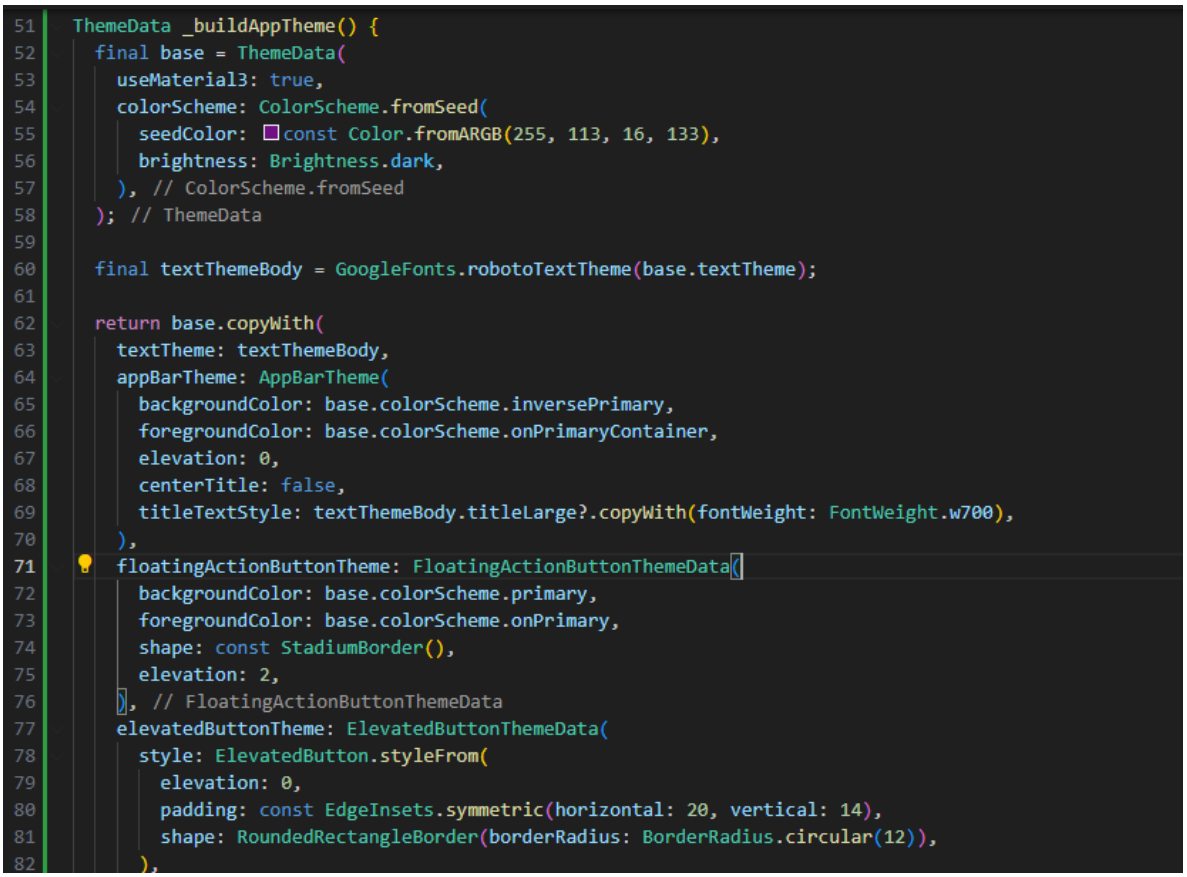
```
    ), // Text

    const SizedBox(height: 24), // Algo de espacio entre el texto y las imagenes

    // Carrusel horizontal de imágenes debajo del texto
    SingleChildScrollView(
      scrollDirection: Axis.horizontal,
      child: Row(
        children: [
          _assetThumb('assets/Pixel-Art-Hot-Pepper-2-1.webp', 'Ají'),
          _assetThumb('assets/Pixel-Art-Pizza-2.webp', 'Pizza'),
          _assetThumb('assets/Pixel-Art-Watermelon-3.webp', 'Sandía'),
        ],
      ), // Row
    ), // SingleChildScrollView
  ], // <Widget>[]
), // Column
), // Center
```

Ahora se nos dan dos alternativas, yo elegí la A, que consiste en usar un package para compartir colores y tipografías a través del Theme de la app. Primero, agregamos una nueva dependencia:




```yaml
dependencies:
  flutter:
    sdk: flutter

  # The following adds the Cupertino Icons font to your application.
  # Use with the CupertinoIcons class for iOS style icons.
  cupertino_icons: ^1.0.8
  logger: ^2.6.1
  google_fonts: ^6.3.1
```

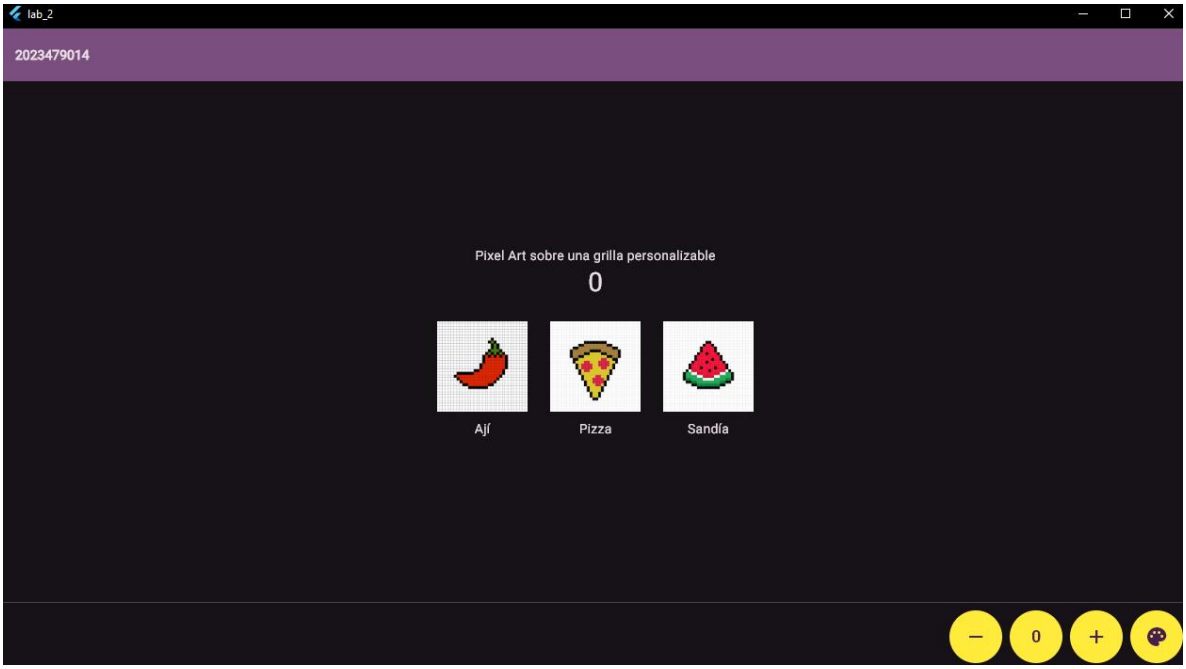
Ahora, siguiendo la documentación de flutter (y con ayuda de tutoriales), creamos un nuevo tema para nuestra aplicación:



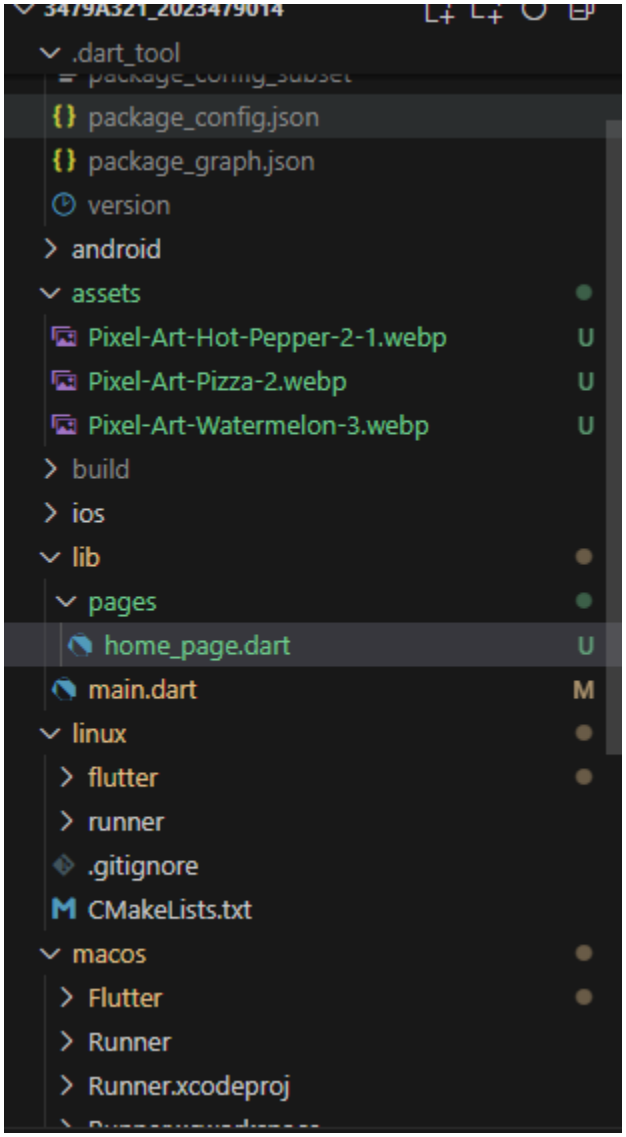
```dart
ThemeData _buildAppTheme() {
  final base = ThemeData(
    useMaterial3: true,
    colorScheme: ColorScheme.fromSeed(
      seedColor: const Color.fromARGB(255, 113, 16, 133),
      brightness: Brightness.dark,
    ), // ColorScheme.fromSeed
  ); // ThemeData

  final textThemeBody = GoogleFonts.robotoTextTheme(base.textTheme);

  return base.copyWith(
    textTheme: textThemeBody,
    appBarTheme: AppBarTheme(
      backgroundColor: base.colorScheme.inversePrimary,
      foregroundColor: base.colorScheme.onPrimaryContainer,
      elevation: 0,
      centerTitle: false,
      titleTextStyle: textThemeBody.titleLarge?.copyWith(fontWeight: FontWeight.w700),
    ),
    floatingActionButtonTheme: FloatingActionButtonThemeData(
      backgroundColor: base.colorScheme.primary,
      foregroundColor: base.colorScheme.onPrimary,
      shape: const StadiumBorder(),
      elevation: 2,
    ), // FloatingActionButtonThemeData
    elevatedButtonTheme: ElevatedButtonThemeData(
      style: ElevatedButton.styleFrom(
        elevation: 0,
        padding: const EdgeInsets.symmetric(horizontal: 20, vertical: 14),
        shape: RoundedRectangleBorder(borderRadius: BorderRadius.circular(12)),
      ),
```

```
      foregroundColor: base.colorScheme.onPrimary,
      shape: const StadiumBorder(),
      elevation: 2,
    ), // FloatingActionButtonThemeData
    elevatedButtonTheme: ElevatedButtonThemeData(
      style: ElevatedButton.styleFrom(
        elevation: 0,
        padding: const EdgeInsets.symmetric(horizontal: 20, vertical: 14),
        shape: RoundedRectangleBorder(borderRadius: BorderRadius.circular(12)),
      ),
    ), // ElevatedButtonThemeData
    cardTheme: const CardThemeData(
      elevation: 0,
      color: Colors.white,
      margin: EdgeInsets.all(12),
      shape: RoundedRectangleBorder(
        borderRadius: BorderRadius.all(Radius.circular(16)),
      ), // RoundedRectangleBorder
    ), // CardThemeData
    inputDecorationTheme: InputDecorationTheme(
      border: OutlineInputBorder(borderRadius: BorderRadius.circular(12)),
      filled: true,
      fillColor: base.colorScheme.surfaceContainerHighest,
    ), // InputDecorationTheme
  );
}
```

Así se ve el nuevo tema (buen Dark Mode para los ojitos):



Para finalizar el laboratorio, se nos solicita dejar la pantalla “Home” en un archivo aparte, esto considerando que en un futuro podriamos tener más de una pantalla, para esto, en el main, eliminamos la clase MyHomePage, y usamos un import hacia “pages/home_page.dart”, en donde dejaremos parte de lo creado:



pubspec.yaml M | main.dart M | home_page.dart U

lib > main.dart > ...

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

// importar pagina
import 'pages/home_page.dart';

Run | Debug | Profile
void main() {
  runApp(const MyApp());
}

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

  @override
  Widget build(BuildContext context) {
    final logger = Logger();
    logger.d("Logger está dando cara!");

    return MaterialApp(
      title: '2023479014',
      theme: _buildAppTheme(),
      home: const MyHomePage(title: '2023479014'),
      debugShowCheckedModeBanner: false,
    ); // MaterialApp
  }
}

ThemeData _buildAppTheme() {
```