

PERÍODO 2022

PRÁCTICA DE LABORATORIO 6

**DESARROLLO DE APLICACIÓN
MÓVIL HÍBRIDA CON FLUTTER
USANDO CONEXIÓN A BASES
DE DATOS EXTERNAS**



**MSIG. ADRIANA COLLAGUAZO JARAMILLO
ITINERARIO DE APLICACIONES MÓVILES Y SERVICIOS TELEMÁTICOS
CARRERA DE INGENIERÍA EN TELEMÁTICA
FIEC ESPOL**

Objetivo de Aprendizaje: Desarrollar aplicaciones móviles híbridas sencillas considerando las características de la programación de dispositivos móviles.

Recursos: Android Studio, Firebase

Duración: 10 horas

INTRODUCCIÓN:

Flutter es un conjunto de herramientas de interfaz de usuario multiplataforma que está diseñado para permitir la reutilización de código en sistemas operativos como iOS y Android, al mismo tiempo que permite que las aplicaciones interactúen directamente con los servicios de la plataforma subyacente. El objetivo es permitir que los desarrolladores entreguen aplicaciones de alto rendimiento que se sientan naturales en diferentes plataformas, adoptando las diferencias donde existen mientras comparten la mayor cantidad de código posible.

Beneficios de Flutter

- Alta productividad al ser multiplataforma
- Desarrollo rápido y sencillo
- Compatibilidad
- De código abierto

Desventajas de Flutter

- Las aplicaciones creadas con Flutter tienden a ser pesadas.
- Pocas librerías de terceros

Limitaciones de Flutter

- Flutter está usando un lenguaje de programación llamado Dart.
- Complejidad limitada

ACTIVIDADES:

Paso 1: Instalar Flutter

Requerimientos

Antes de continuar hay que tener en cuenta los siguientes requerimientos para instalar Flutter.

Hardware

En primera instancia se recomienda un sistema con:

- Mínimo 8gb de ram.
- Espacio libre en disco: 6 gb mínimo.
- *Si posee tarjeta de video mucho mejor debido a que el emulador del dispositivo Android consume muchos recursos.

Software

Windows:

- Windows 7 SP1 o posterior (64 bits), x86-64
- Windows Powershell 5.0 o posterior. (Preinstalado en Windows 10)
- Git for Windows 2.x (Debe permitir ejecutar comandos "git" en el command prompt o powershell)

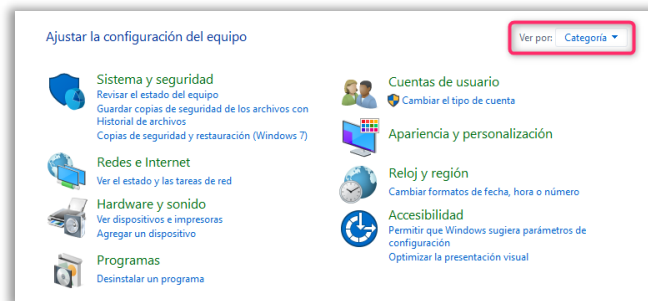
MacOS:

- MacOS (64-bit)
- Xcode (Incluye git, aunque puede ser instalado por separado)

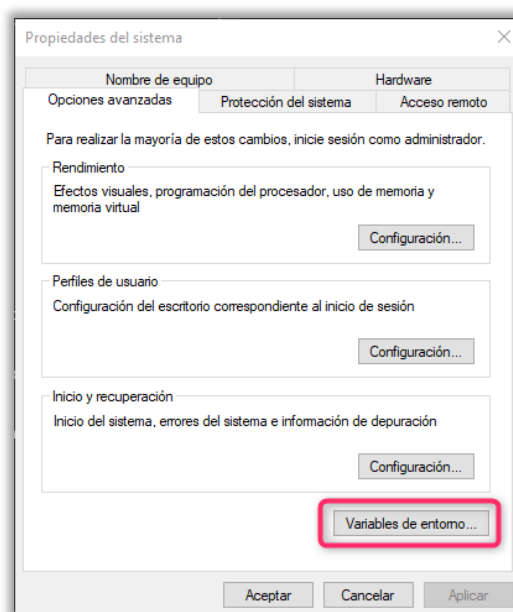
Linux:

- Linux (64-bit)

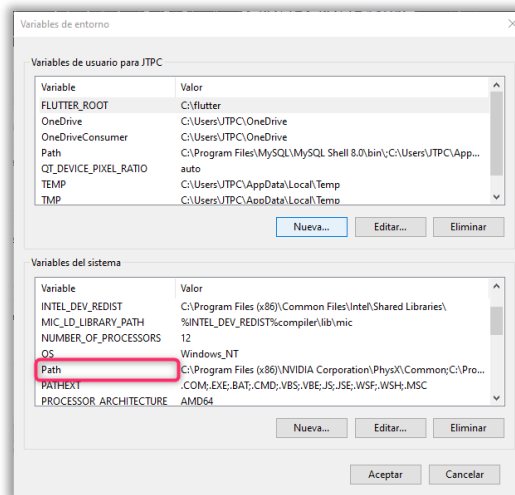
1. Descargar el archivo SDK en su última versión estable en el siguiente enlace, ahí podrán encontrar los diferentes archivos para cada tipo de sistema operativo.:
 - a. <https://flutter.dev/docs/development/tools/sdk/releases>
2. Extraemos el archivo zip.
 - a. En Windows podemos dar click derecho al archivo y damos en extraer todo.
 - b. En MacOS podemos dar click derecho al archivo y dar click en abrir, con esto de descomprimirá el archivo.
3. La carpeta extraída la alojamos en la siguiente localización. (En caso de no existir el directorio, crearlo).
 - a. En Windows: **C:\Users\<tu-nombre-de-usuario>\Documents.**
 - b. En Mac: **<Disco Principal>/Users/<tu-nombre-de-usuario>/development**
4. Debemos agregar el path a las variables del sistema:
 - a. Windows
 - i. En el inicio buscamos panel de control y lo abrimos.
 - ii. En la parte derecha superior seleccionamos Ver por: "Iconos grandes"
 - iii. Luego damos click en Sistema.



- iv. Dependiendo de la versión de Windows puede que se mantenga en el mismo panel de control o se inicie una nueva ventana de configuración.
- v. En esta nueva ventana buscaremos "Configuración avanzada del sistema", si se mantuvo en el panel de control aparecerá en un panel izquierda. En el otro caso de que se haya abierto la ventana de configuración esta opción está en el panel derecho.
- vi. Se abrirá la ventana de Propiedades del sistema, ahí daremos click en variables de entorno.



- vii. En el cuadro inferior de “Variables del sistema” buscaremos la variable llamada “Path” y le daremos doble click.

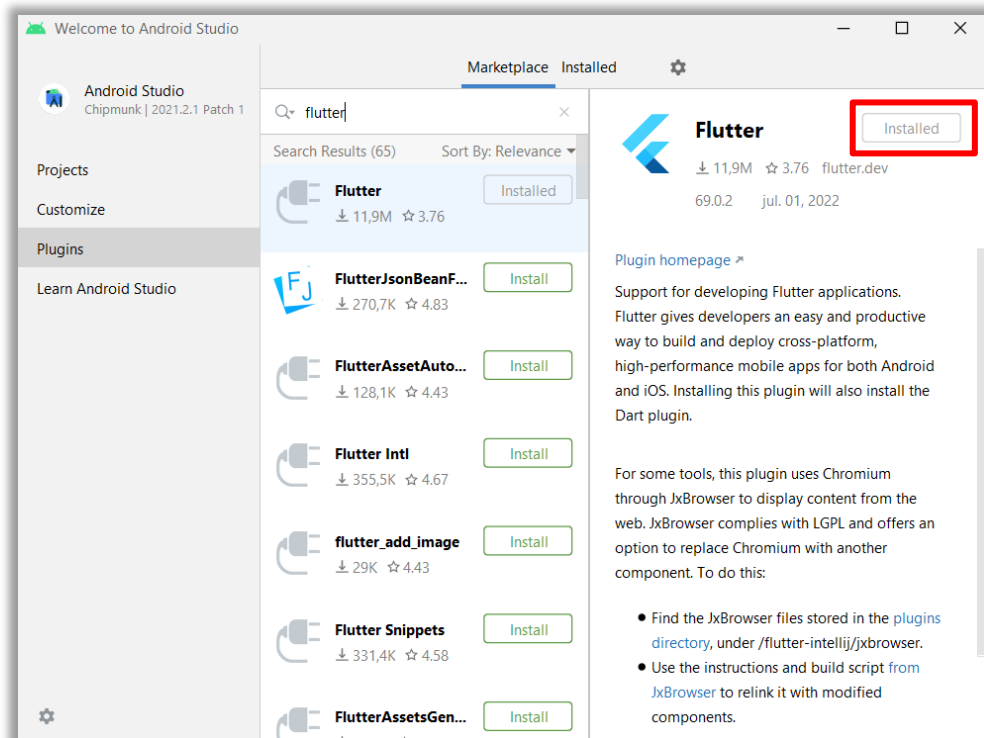


- viii. Se abrirá una nueva ventana, ahí presionaremos en el botón nuevo.
- ix. Escribiremos la ruta donde se alojó la carpeta de flutter. Ejemplo: **C:\Users\<tu-nombre-de-usuario>\Documents\flutter\bin**
- x. Daremos en aceptar en las ventanas hasta cerrarlas todas.
- b. MacOS.
- Abriremos una terminal.
 - Nos dirigiremos al directorio del usuario con el comando: **cd /Users/<tu-nombre-de-usuario>**
 - Una vez dentro, realizaremos el comando **vim .bash_profile**
 - Presionamos la tecla **i** para empezar a editar el archivo creado.
 - Escribiremos: **export PATH="\$PATH:/Users/<tu-nombre-de-usuario>/development/flutter/bin"** (el path agregado debe coincidir con el path donde se encuentra la carpeta de flutter alojada.)
 - Para salir de vim escribiremos **:.wq!**
 - Cerramos y volvemos a abrir la terminal.
5. Para verificar que hemos agregado correctamente flutter a nuestro path abriremos una instancia de cmd en Windows o una nueva terminal en MacOS.
6. Escribimos el comando **flutter --version**, deberá aparecer la versión actualmente instalada de flutter, en caso de error verificar los pasos anteriores.
7. Una vez verificado, ejecutamos el siguiente comando: **flutter doctor**
8. En caso de pedir aceptar algunas opciones de firmas y datos lo hacemos.

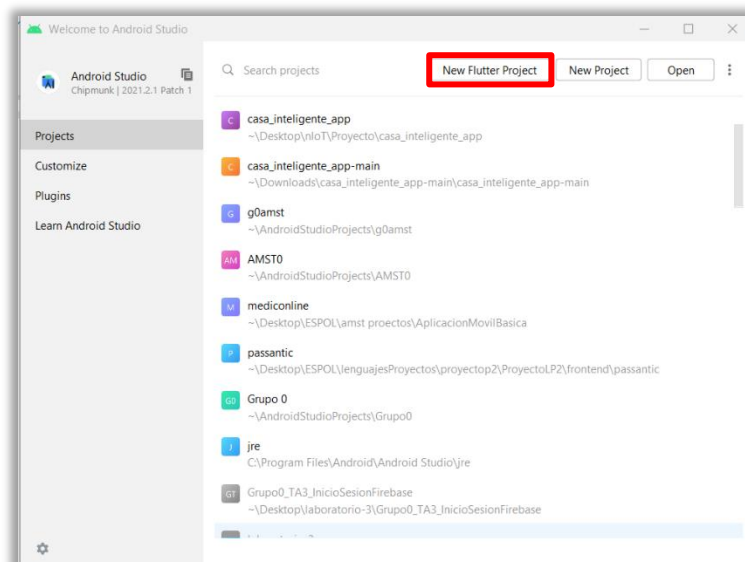
Paso 2: Configuración de Android Studio

Para esto ya deben tener instalado Android Studio y actualizado a la última versión.

- Abriremos Android Studio.
- Abriremos **configure > settings**.
- En la parte derecha seleccionaremos la pestaña **Plugins**.
- Buscaremos el Marketplace el nombre el plugin llamado **“flutter”**
- Una vez encontrado presionaremos en el icono de **Install**, para instalar el plugin.



6. En caso de mostrarnos un mensaje de que existe dependencia con dart, le damos a **install** también.
7. Esperamos a que termine y se mostrará un botón de **reiniciar ide**, presionamos y esperamos.
8. Para verificar que la instalación fue correcta en la página de inicio nos aparecerá una nueva pestaña que dice: **"Create new flutter Project"**.

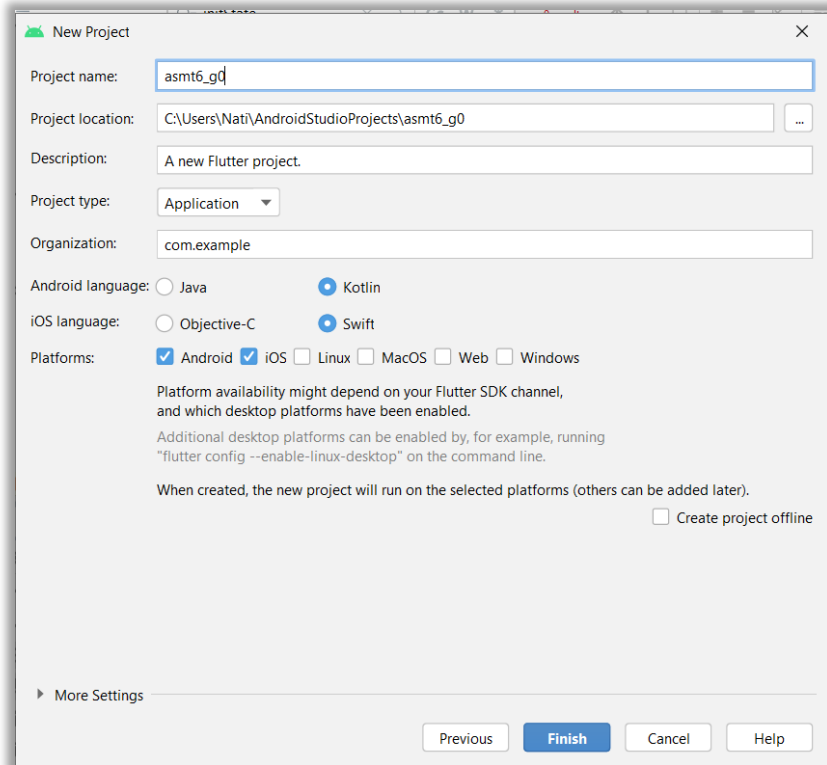


Paso 3: Crear nuevo proyecto

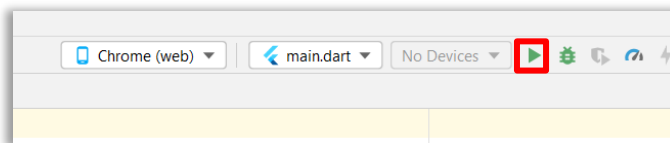
Crearemos un nuevo proyecto para familiarizarnos con los archivos de flutter.

1. Abriremos Android Studio.
2. Daremos click en **"Create new flutter project"**.
3. Donde dice Flutter SDK path deberíamos tener el directorio de nuestra carpeta de flutter, en caso de no tenerla:

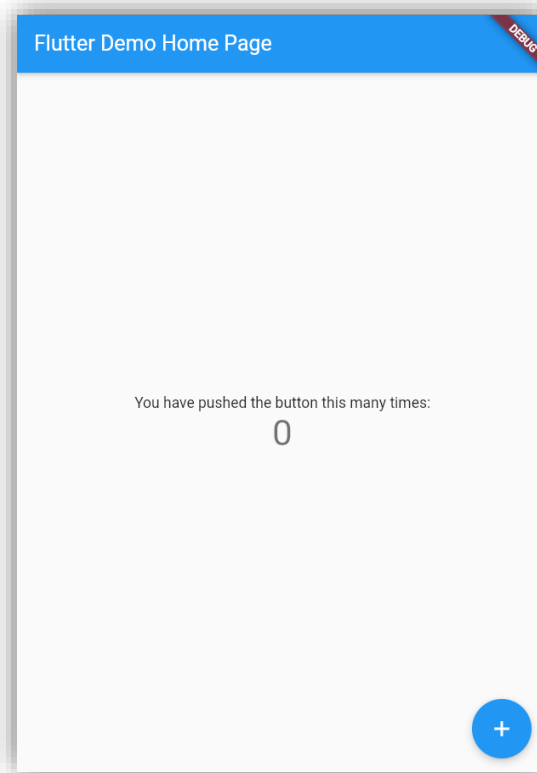
- a. Presionar en **los 3 puntos a la derecha**.
 - b. Buscar y seleccionar el directorio de la carpeta de flutter (Donde se la guardó cuando se instaló).
4. Le damos click en **Next**.
5. En la siguiente página podremos poner el nombre y la localización del proyecto, en nuestro caso llamaremos el proyecto **"amst_g0"**. Coloquen el nombre según el grupo.



6. Damos click en **Finish**.
7. Se nos abrirá la app de ejemplo que viene por defecto en flutter.
8. Para correrla debemos de tener un dispositivo Android conectado, o iniciado un dispositivo virtual.
9. Presionamos en el icono de run para que empiece a compilar la aplicación y automáticamente se abrirá en el dispositivo la aplicación.



10. Con esto se nos abra una aplicación que la genera flutter como demostracion. Eliminaremos todos los comentarios que encontramos en el archivo `./lib/main.dart`.



11. OJO* Algunos comentarios que son nombres de widgets como “MaterialAPP o ThemeData” se generan automaticamente por Android Studio para mejor facilidad de visualizacion de los componentes. Con esto estamos listos para el desarrollo de la app.

Paso 4: Desarrollo de la aplicación

En caso de tener más interés en el desarrollo de flutter, y tiene experiencia en el desarrollo en Android nativo, se recomienda visitar el siguiente enlace: <https://flutter.dev/docs/get-started/flutter-for/android-devs>. Aquí se explica a un programador de Android, cuáles son sus equivalentes en flutter. Así mismo encontrara para desarrolladores de IOS, web y más.

1. Primero debemos cambiar nuestra app de ser un StatelessWidget a un StatefulWidget, para esto debemos de modificar la clase dejándola de esta manera. Eliminar const del void main también.

```
1 class MyApp extends StatefulWidget {  
2   _AppState createState() => _AppState();  
3 }
```

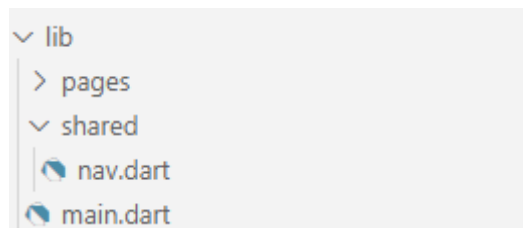
2. Consecutivamente eliminaremos la clase MyHomePage y la clase _MyHomePageState.

```
1 class MyHomePage extends StatefulWidget {  
2   ...  
3 }  
4  
5 class _MyHomePageState extends State<MyHomePage> {  
6   ...  
7 }
```

3. Crearemos una nueva clase llamada `_AppState` definida de la siguiente manera.

```
1 class _AppState extends State<MyApp> {
2   final GlobalKey<NavigatorState> navigatorKey =
3     new GlobalKey<NavigatorState>();
4
5   @override
6   void initState() {
7     super.initState();
8   }
9
10  @override
11  Widget build(BuildContext context) {
12    return MaterialApp(
13      title: 'Vacheck',
14      navigatorKey: navigatorKey,
15      initialRoute: 'nav',
16      routes: {
17        'nav': (context) => Nav(),
18      },
19      theme: ThemeData(
20        primaryColor: Colors.red[400],
21      ),
22    );
23  }
24 }
```

4. Aún falta crear nuestro componente `Nav`, que será el navegador entre las páginas. Para esto en la carpeta `lib` del proyecto crearemos una carpeta llamada `shared`. Y allí crearemos nuestro archivo `"nav.dart"` tal como se muestra a continuación.



5. Una vez creado el archivo, importaremos el paquete `material.dart` y definiremos la clase `Nav`:

```
1 import 'package:flutter/material.dart';
2
3 class Nav extends StatefulWidget {
4   @override
5   _NavState createState() => _NavState();
6 }
```

6. Continuaremos definiendo el `_NavState` de la página de la siguiente manera

```
1 class _NavState extends State<Nav> {
2
```



```

3   int _currentIndex = 1;
4
5   @override
6   Widget build(BuildContext context) {
7     return Scaffold(
8       appBar: AppBar(
9         title: Text('Vacheck'),
10      ),
11      body: _llamarPagina(_currentIndex),
12      bottomNavigationBar: _crearNavigationBar(),
13    );
14  }
15  Widget _crearNavigationBar() {
16    return BottomNavigationBar(
17      unselectedItemColor: Color.fromRGBO(168, 97, 93, 0.4),
18      iconSize: 30.0,
19      unselectedIconTheme: IconThemeData(size: 25.0),
20      currentIndex: _currentIndex,
21      onTap: (index) {
22        setState(() {
23          _currentIndex = index;
24        });
25      },
26      items: [
27        BottomNavigationBarItem(
28          icon: Icon(Icons.data_usage), label: 'Data'),
29        BottomNavigationBarItem(
30          icon: Icon(Icons.home), label: 'Home'),
31        BottomNavigationBarItem(
32          icon: Icon(Icons.bar_chart), label: 'Chart'),
33      ]);
34  }
35
36  Widget _llamarPagina(int paginaActual) {
37    switch (paginaActual) {
38      default:
39        return Text("Hola Mundo");
40    }
41  }
42 }

```

La primera variable current index, nos ayudara luego para poder usar la barra de navegación entre páginas.

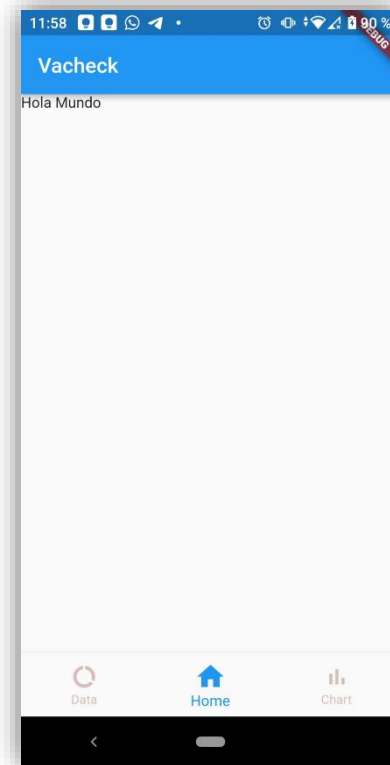
La función build es la que genera el widget que se muestra, el cual tiene título, barra de navegación y cuerpo. En el caso de la barra de navegación llamamos a la función _crearNavigationBar que crea una barra de navegación con 3 ítems, y que cambia el índice de la página en la que nos encontramos al presionar en el botón.

Por último, está la función _llamarPagina que se encarga de llamar a la página que el usuario seleccione, por ahora hemos dejado el switch en default para que en cualquier caso nos muestre siempre el mensaje Hola mundo.

7. Ahora debemos dirigirnos a el archivo main.dart para importa la clase Nav que necesitábamos.

```
1 import 'package:vacheck/shared/nav.dart';
```

8. Una vez hecho esto, podemos ejecutar la app para visualizar los avances (En caso de que siga teniendo la app ejecutándose desde el inicio del proyecto, por favor detenerla y volverla a iniciar, debido a que se hizo el cambio de estado a la aplicación)



9. Ahora crearemos la página de bienvenida super básica. Iremos al directorio “lib” y crearemos una nueva carpeta llamada “pages” y dentro crearemos un archivo llamado “home_page.dart”.
10. Dentro del archivo “home_page.dart” empezaremos a definir la clase HomePage

```
1 import 'package:flutter/material.dart';
2
3 class HomePage extends StatefulWidget{
4   @override
5   _HomePageState createState() => _HomePageState();
6 }
7
8 class _HomePageState extends State<HomePage>
9   with WidgetsBindingObserver {
10   @override
11   Widget build(BuildContext context){
12     return Container(
13       color: Colors.red[300],
14       padding: EdgeInsets.symmetric(horizontal: 18),
15       child: Column(
16         mainAxisAlignment: MainAxisAlignment.center,
17         children: [
18           Icon(Icons.airport_shuttle,color:
```

```

19 Colors.white, size: 190, ),
20 Text("Welcome",
21     style: TextStyle(
22         color: Colors.white,
23         fontSize: 60,
24         fontWeight:
25             FontWeight.bold
26     ),
27 ),
28 Text("Bienvenido a la app para visualizar
29 la informacion de temperatura de los camiones
30 que transportan las vacunas",
31     style: TextStyle(
32         color: Colors.white,
33         fontWeight: FontWeight.w300
34     )
35 ),
36 ],
37 );
38 }
39 }

```

En `_HomePageState` con la función `build`, crearemos el contenedor que ira en el body del widget principal. ***En caso de error en el texto de la línea 28 quitar los saltos de línea y dejarlo en una sola línea.***

11. Ahora esta página la llevaremos para que la clase `Nav` la use. Editaremos la función `_llamarPagina` para que retorne la instancia de la clase `HomePage`.

```

1 Widget _llamarPagina(int paginaActual)
2 {
3     switch (paginaActual) {
4         default:
5             return HomePage();
6     }
7 }

```

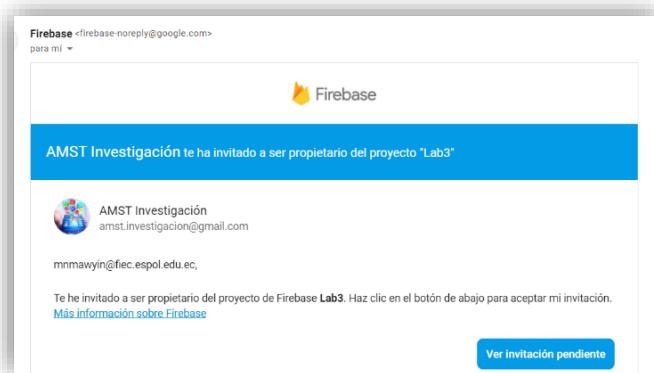
Recordar importar el archivo `home_page` para no tener errores, guardamos y ahora se nos muestra un mensaje de bienvenida simple.



12. Continuaremos ahora con la página de los datos, así mismo crearemos en la carpeta “pages” un nuevo archivo llamado “data.dart”.

Paso 5: Conexión con firebase

1. Se le ha dado permiso a cada líder de grupo de trabajo para el proyecto de firebase. Las invitaciones se encuentran en su correo proporcionado de Gmail.



2. Antes de continuar, revisar que la configuración de versiones de SDK en Android/app/build.gradle esté de la siguiente manera:

```
minSdkVersion 21
targetSdkVersion 31
```

3. En caso de no haber trabajado antes con flutter-firebase, se ejecuta el siguiente comando en el CMD para instalar Firebase CLI con npm:

```
npm install -g firebase-tools
```

Nota: De no tener npm puede ir al siguiente link para buscar otras alternativas de instalación: <https://firebase.google.com/docs/cli#windows-npm>

- Podemos acceder a Firebase con nuestra Cuenta de Google ejecutando el siguiente comando:

```
firebase login
```

- Para instalar la CLI de FlutterFire, ejecutamos el siguiente comando desde cualquier directorio:

```
dart pub global activate flutterfire_cli
```

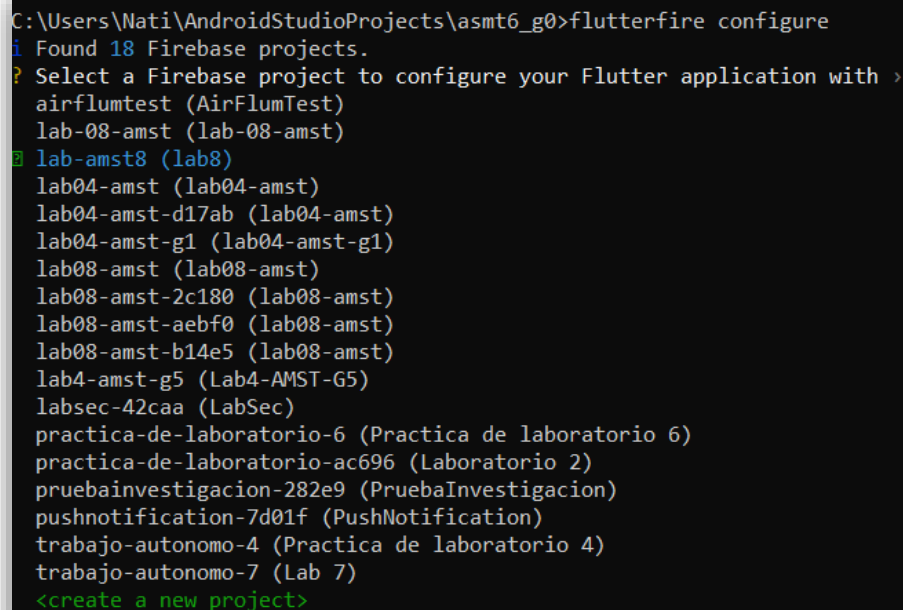
- Vamos a
- Desde el **directorio del proyecto de Flutter**, ejecutaremos el siguiente comando para iniciar el flujo de trabajo de configuración de la app:

```
flutterfire configure
```

Nota: En caso de problemas con este comando agregar:

`C:\Users*username*\AppData\Local\Pub\Cache\bin` a las variables del sistema y reiniciar el equipo

- Aparecerá una lista. Ahora debemos seleccionar el proyecto en el cual vamos a trabajar, que es aquel al que se les dio acceso en el paso 1.



```
C:\Users\Nati\AndroidStudioProjects\asmt6_g0>flutterfire configure
i Found 18 Firebase projects.
? Select a Firebase project to configure your Flutter application with >
airflumtest (AirFlumTest)
lab-08-amst (lab-08-amst)
lab-amst8 (lab8)
lab04-amst (lab04-amst)
lab04-amst-d17ab (lab04-amst)
lab04-amst-g1 (lab04-amst-g1)
lab08-amst (lab08-amst)
lab08-amst-2c180 (lab08-amst)
lab08-amst-aebf0 (lab08-amst)
lab08-amst-b14e5 (lab08-amst)
lab4-amst-g5 (Lab4-AMST-G5)
labsec-42caa (LabSec)
practica-de-laboratorio-6 (Practica de laboratorio 6)
practica-de-laboratorio-ac696 (Laboratorio 2)
pruebainvestigacion-282e9 (PruebaInvestigacion)
pushnotification-7d01f (PushNotification)
trabajo-autonomo-4 (Practica de laboratorio 4)
trabajo-autonomo-7 (Lab 7)
<create a new project>
```

- Luego pedirá que seleccionemos las plataformas que se van a configurar. Vienen por defecto Android y IOS que es lo que seleccionaremos. Dar enter para continuar.

```
C:\Users\Nati\AndroidStudioProjects\asmt6_g0>flutterfire configure
Found 18 Firebase projects.
Select a Firebase project to configure your Flutter application with - lab-amst8 (lab8)
Which platforms should your configuration support (use arrow keys & space to select)? - android, ios
Firebase android app com.example.asmt6_g0 registered.
Firebase ios app com.example.asmt6G0 is not registered on Firebase project lab-amst8.
Registered a new Firebase ios app on Firebase project lab-amst8.

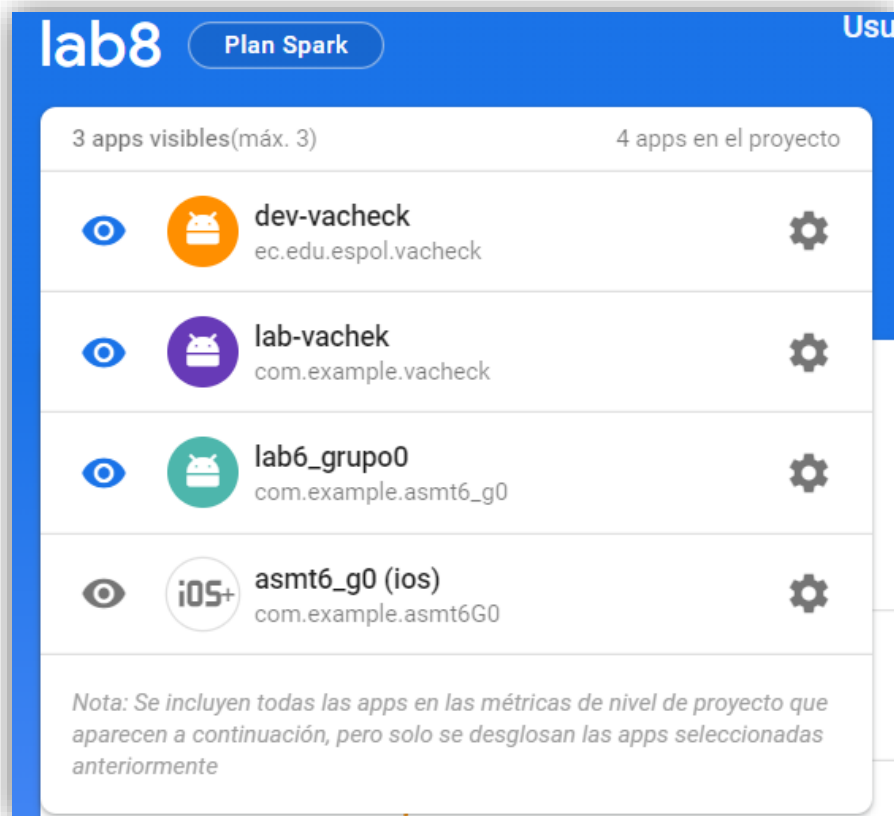
Firebase configuration file lib\firebase_options.dart generated successfully with the following Firebase apps:

Platform  Firebase App Id
android   1:72394538686:android:e8aebb0cab6686d47828ef
ios       1:72394538686:ios:69f3a89b876650867828ef

Learn more about using this file and next steps from the documentation:
> https://firebase.google.com/docs/flutter/setup

C:\Users\Nati\AndroidStudioProjects\asmt6_g0>
```

10. Para verificar nos dirigimos a la consola de firebase y podremos observar las apps creadas



11. Desde el directorio del proyecto de Flutter, ejecutemos el siguiente comando para instalar el complemento principal:

```
flutter pub add firebase_core
```

Podemos utilizar la terminal de Android studio.

12. Otra forma de instalar los plugins es agregarlos directamente al pubspec.yaml. En este caso nos dirigimos a ese archivo y agregamos lo siguiente:

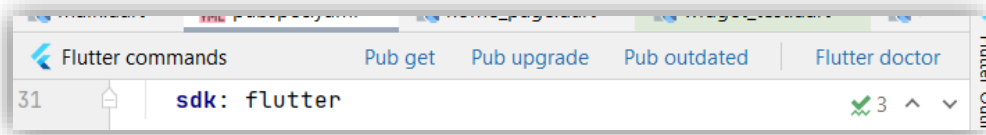
```
1 dependencies:
2   flutter:
3     sdk: flutter
4
5
```

```

6  #The following adds the Cupertino Icons font to your application.
7  #Use with the CupertinoIcons class for iOS style icons.
8  cupertino_icons: ^1.0.2
9  firebase_core: ^1.20.0
10 expandable: ^5.0.1
11 firebase_database: ^9.1.0
12 google_fonts: ^3.0.0
13 charts_flutter: ^0.12.0

```

Dar click a Pub get o ejecutarlo como comando



13. Ejecutamos nuevamente el comando **flutterfire configure** para asegurar de que la configuración de Firebase de la app de Flutter esté actualizada. Colocamos la misma configuración que en los pasos anteriores.
14. Ahora vamos al archivo lib/main.dart, donde importaremos el complemento principal de Firebase y el archivo de configuración que fue generado antes:

```

import 'package:firebase_core/firebase_core.dart';
import 'firebase_options.dart';

```

15. Además, en lib/main.dart, inicializamos Firebase con el objeto DefaultFirebaseOptions exportado por el archivo de configuración:

```

Future<void> main() async {
  WidgetsFlutterBinding.ensureInitialized();
  await Firebase.initializeApp();
  runApp(MyApp());
}

```

16. Corremos la aplicación para verificar que todo esté en orden. De todo salir bien, no saldrá ningún error y la pantalla se verá igual que antes de la configuración:



Paso 6: Continuación del desarrollo

1. Ahora si iremos a el archivo `./lib/pages/data.dart` para definir la clase.

```
import 'dart:developer';

import 'package:firebase_database/ui/firebase_animated_list.dart';
import 'package:flutter/material.dart';
import 'package:firebase_database/firebase_database.dart';

class Data extends StatefulWidget {
  @override
  _DataState createState() => _DataState();
}

class _DataState extends State<Data> {
  late final dataD;
  late DatabaseReference _dataref;
  @override
  void initState() {
    _dataref = FirebaseDatabase.instance.ref('data');
    super.initState();
  }

  @override
  Widget build(BuildContext context) {
    return Column(
      children: [
        Tab(
          child: Text("Data from database"),
        ),
        _crearListado(context),
      ],
    );
  }

  Widget _crearListado(BuildContext context) {
    return Flexible(child: FirebaseAnimatedList(
      query: _dataref,
      itemBuilder: (BuildContext context, DataSnapshot snapshot,
        Animation<double> animation, int index) {
        var id = ((snapshot.value as
dynamic) ["end_device_ids"] ["device_id"]).toString();
        var humedad = ((snapshot.value as
dynamic) ["uplink_message"] ["decoded_payload"] ["humedad"]).toString();
        var temperatura = ((snapshot.value as
dynamic) ["uplink_message"] ["decoded_payload"] ["temperatura"]).toString()
;
        var fecha = ((snapshot.value as
dynamic) ["received_at"]).toString();
        return Container(
          margin: EdgeInsets.symmetric(vertical: 5, horizontal: 10),
          decoration: BoxDecoration(
            color: Colors.blueGrey[50],
            borderRadius: BorderRadius.circular(5),
          ),
          child: ListTile(
            title: Text("ID: " + id,
              style: TextStyle(fontWeight: FontWeight.bold)),
            subtitle: Text("Humedad: "+humedad +"\n"
              + "Temperatura: " + temperatura
            ),
          ),
        );
      },
    ));
  }
}
```



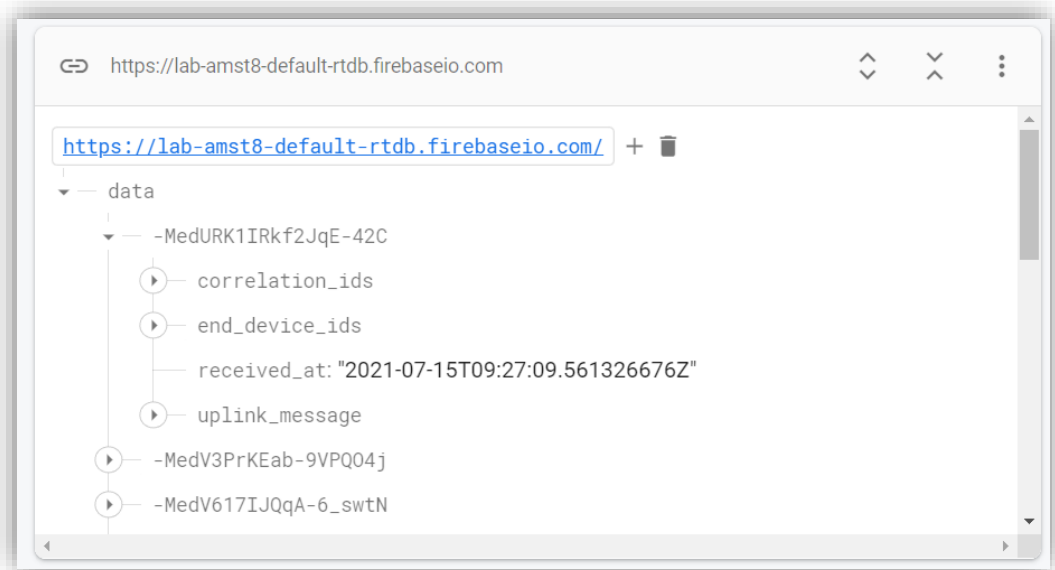
```

        trailing: Column(
            mainAxisAlignment: MainAxisAlignment.center,
            children: [
                Text("Fecha: " + fecha.split("T")[0]),
                Text("Hora: " + fecha.split(".")[0].split("T")[1]),
            ],
        ),
    ),
);
}

),
);
}
}
}

```

Dentro de esta clase inicializamos “_dataref” que es la referencia a la sección de datos que nos vamos a enfocar, en nuestro caso “data”, como podemos ver en la base de datos.



Luego tenemos una función build que construye la columna que se mostrara en pantalla, pero esta clase dentro de sus hijos llama a la función “_crearListado” que es la que realiza la magia. Dentro de esta función lo que hacemos es crear un widget de tipo Flexible, que tiene como hijo una FirebaseAuth, este widget permite conectarse a tiempo real a la base de datos de firebase, y mostrar los cambios que se hagan a tiempo real.

Dentro del itemBuilder de ese widget se ejecuta una función lambda que realiza la conexión hacia un query que es la referencia a la sección de la base, obteniendo los datos en una variable llamada snapshot desde donde saca la información como fecha, id, etc. Con esto generamos un container y alojamos la información de cómo queremos que se vea automáticamente para realizar el listado de todos los datos que reciba.

2. Ahora dentro de la clase `Nav()`, debemos agregar la clase `Data` para que cuando presionamos en la pestaña de datos se renderice esta página. Lo que haremos es agregar un nuevo caso en nuestro switch para llamar a la instancia de esta manera (Recordar importar el archivo `data.dart`):

```
1 Widget _llamarPagina(int paginaActual) {
2     switch (paginaActual) {
```

```

3      case 0:
4          return Data();
5      default:
6          return HomePage();
7      }
8  }

```

- Una vez realizado esto, guardar los archivos. Detener el proceso de debug y volver a ejecutar la app para que se compile la app con las nuevas dependencias. Al terminar si nos vamos a la pestaña de data veremos como ya obtiene los datos desde la base de datos.



- Ahora crearemos la página del gráfico, crearemos un nuevo archivo "chart.dart" dentro de ./lib/pages/
- Dentro añadiremos las siguientes líneas.

```

import 'package:flutter/material.dart';
import 'package:charts_flutter/flutter.dart' as charts;
import 'package:firebase_database/firebase_database.dart';

class Chart extends StatefulWidget {

  @override
  _ChartState createState() => _ChartState();
}

class _ChartState extends State<Chart> {
  late List<Registro> _data;
  late List<charts.Series<Registro,String>> _chardata;
  late DatabaseReference _dataref;

  void _makeData() {
    _chardata.add(
      charts.Series(

```

```

        domainFn: (Registro registro, _) =>
            registro.fecha + " " + registro.hora,
        measureFn: (Registro registro, _) =>
            registro.temperatura,
        id: 'Registros',
        data: _data,

    )
    );
}

@override
void initState() {
    final FirebaseDatabase database = FirebaseDatabase.instance;
    _dataref = database.reference().child("data");
    _data = <Registro>[];
    _chardata = <charts.Series<Registro, String>>[];
}

@override
Widget build(BuildContext context) {
    return Scaffold(
        body: _buildBody(context),
    );
}

Widget _buildBody(context) {
    return StreamBuilder(
        stream: _dataref.onValue,
        builder: (context, snapshot) {
            print(snapshot.data);
            if (!snapshot.hasData) {
                return LinearProgressIndicator();
            } else {
                List<Registro> registros = <Registro>[];
                Map data = (snapshot.data as dynamic).snapshot.value;
                for (Map childdata in data.values) {
                    var fecha =
                        childdata["received_at"].toString();
                    var temperatura =

childdata["uplink_message"]["decoded_payload"]["temperatura"].toString();
                    var humedad =

childdata["uplink_message"]["decoded_payload"]["humedad"].toString();
                    registros.add(Registro(fecha, temperatura, humedad));
                }
                return _buildChart(context, registros);
            }
        }
    );
}

Widget _buildChart(BuildContext context, List<Registro> registros) {
    _data = registros;
    _makeData();
    return Padding(
        padding: EdgeInsets.all(8.0),
        child: Container(

```

```

        child: Center(
          child: Column(
            children: <Widget>[
              Text("Diagrama de barras de temperaturas"),
              SizedBox(height: 10.0),
              Expanded(
                child: charts.BarChart(_chardata,
                  animate: true,
                  animationDuration: Duration(seconds: 1),
                  vertical: false,
                ),
              ),
            ],
          ),
        ),
      ),
    ),
  );
}

class Registro{
  String fecha='';
  String hora='';
  double temperatura=0;
  double humedad=0;

  Registro(fecha,temperatura,humedad){
    this.fecha = fecha.split("T")[0];
    this.hora = fecha.split(".")[0].split("T")[1];
    this.temperatura = double.parse(temperatura);
    this.humedad = double.parse(humedad);
  }
}

```

En esta clase definimos las listas donde se guardarán los datos, además de las referencias que son luego inicializadas.

La función `_makeData` lo que hace es definir la configuración del gráfico y define como se usara la data que se entrega.

Luego se define la función `_buildBody` que es llamada por la función `_build` para crear lo que se mostrara en el dispositivo. Se crea un widget tipo `StreamBuilder`, que desde información obtenida desde una base de datos genera lo que sea que le indiquemos. Recibimos los datos desde `firebase` y verificamos que lleguen correctamente, y con información para luego obtener cada dato y enviarlo a la siguiente función.

`_BuildChart` lo que hace es ya indicar a `dart` como tiene que crear el `chart` en el dispositivo.

Por último, agregamos una inner class que nos ayuda a manejar la información de los registros, y entregarlos ordenados al módulo de `charts` de `flutter`.

6. Finalmente necesitamos agregar esta página al `switch` de nuestra barra de navegación, por lo que la agregamos como se muestra a continuación.

```

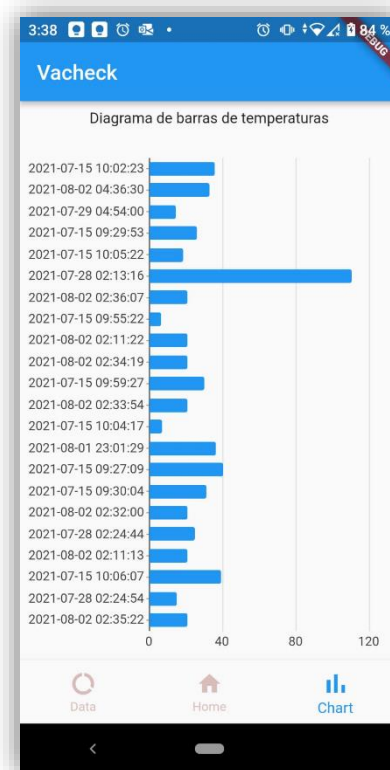
1  Widget _llamarPagina(int paginaActual) {
2    switch (paginaActual) {
3      case 0:
4        return Data();
5      case 2:

```

```

6      return Chart();
7      default:
8      return HomePage();
9  }
10 }
```

7. Guardamos los cambios, y podremos visualizar los datos en nuestra aplicación sin problema.



Nota: Les podría aparecer el siguiente error, pero no es motivo de preocupación por este laboratorio, puesto que viene directo de la librería que se está utilizando. Referirse a <https://docs.flutter.dev/development/tools/sdk/release-notes/release-notes-3.0.0> para mas información.

```

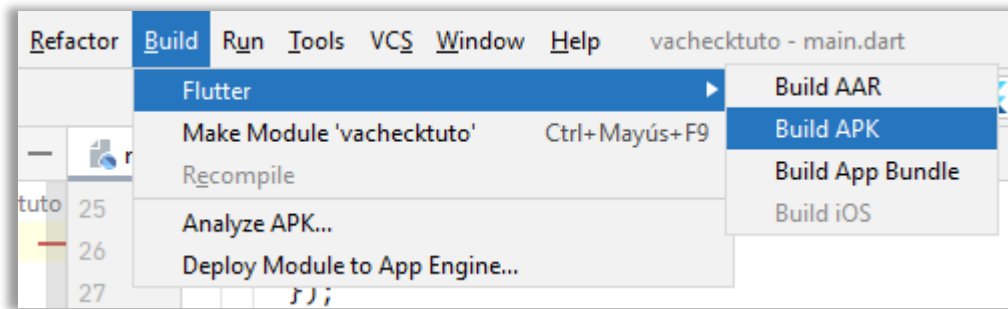
../../Documents/flutter/.pub-cache/hosted/pub.dartlang.org/charts_flutter-0.12
.0/lib/src/chart_container.dart:205:27: Warning: Operand of null-aware
operation '!' has type 'SchedulerBinding' which excludes null.
- 'SchedulerBinding' is from 'package:flutter/src/scheduler/binding.dart'
(' ../../Documents/flutter/packages/flutter/lib/src/scheduler/binding.dart').
  if (!SchedulerBinding.instance!.hasScheduledFrame) {
                                ^
```

Paso 7: Compilación de la aplicación

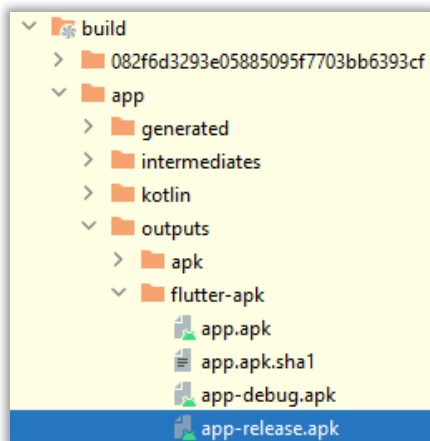
A continuación, realizaremos la compilación de la aplicación para los 2 principales sistemas móviles.

Android

1. Para compilar para Android, desde Android Studio en la parte superior encontraremos la opción Build, seleccionaremos esa opción, luego seleccionamos flutter y por último Build APK.



2. Se abrirá un terminal donde automáticamente iniciará la compilación de la aplicación, esperamos hasta que finalice.
3. La apk la encontraremos en el directorio, `./build/app/outputs/flutter-apk/app-release.apk` como se muestra a continuación:

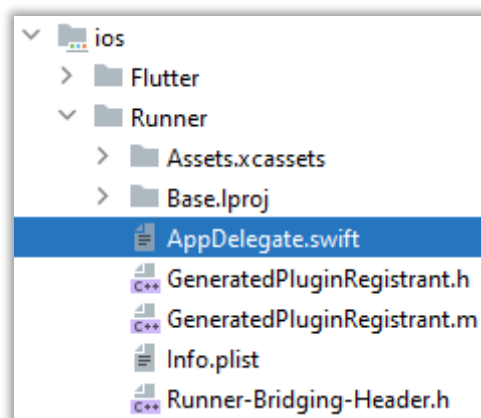


4. Esta apk la moveremos dentro de nuestro dispositivo móvil y lo instalaremos. En caso de error de instalación, activar la instalación de terceros en la configuración. Esto se da debido a que la aplicación no está debidamente firmada por una cuenta asociada a la play store.

IOS

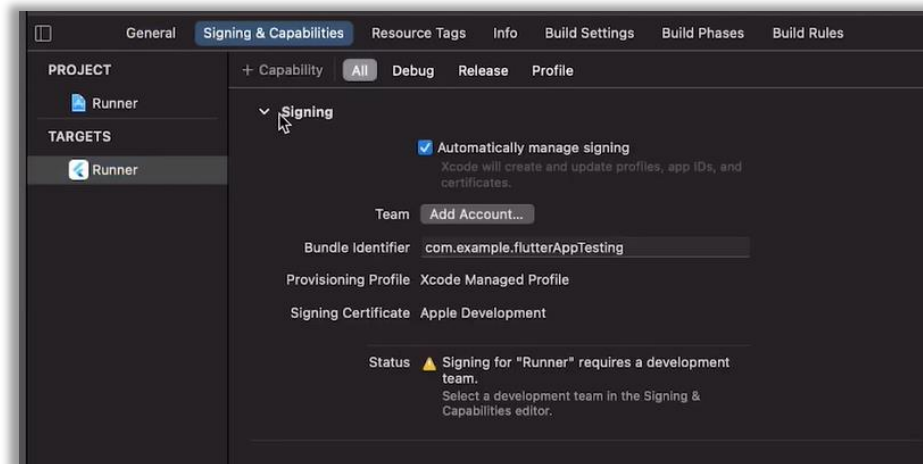
Para compilar nuestra aplicación para IOS necesitamos obligatoriamente tener una computadora macOS, debido a que necesitamos el programa xCode para poder compilarla. Esto es debido a que flutter usa los componentes nativos de IOS para compilar la aplicación.

1. Abrimos nuestro proyecto en Android Studio, desde ahí buscamos el siguiente archivo: `./ios/Runner/AppDelegate.swift`

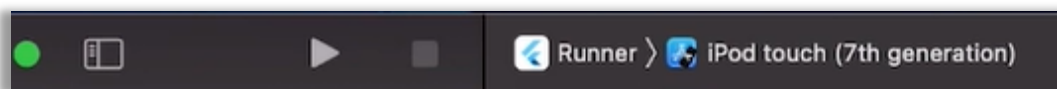


2. Una vez ahí, veremos a la derecha arriba una opción que dice [Open iOS module in Xcode](#), la seleccionamos y esta exportará y conectará a xcode con nuestro proyecto para compilarlo. Esperaremos un momento hasta que xCode se abra.
3. Una vez abierto, en deployment info asegurarnos que esté en la versión 14.1 de IOS.

4. Dentro de la tira de opciones encontraremos una pestaña llamada "Signing & capabilities"
5. En la sección que dice "Team" deberemos agregar nuestra cuenta de desarrollador* (Probar con icloud).



6. *En caso de querer ver la aplicación en nuestro dispositivo iphone*: Una vez realizado esto, podemos conectar nuestro dispositivo iphone a nuestra macOS, en la parte superior debemos seleccionar nuestro dispositivo y luego darle en el botón de play.



7. *Para compilar la aplicación*: En la parte superior en la opción "product", buscaremos "destinations", y escogeremos en "any los Devices (arm64)".
8. Iremos nuevamente a la opción "product" y escogemos la opción archive. Con eso se empezará a generar un archivo, esperaremos hasta que finalice.
9. Una vez finalizado nos mostrara el archivo en una ventana de archivos. A la derecha nos mostrara dos botones, escogeremos el que dice "Distribute App"
10. Seleccionamos el método iOS App Store y presionamos en Next.
11. Seleccionamos el destino en Export y presionamos en Next.
12. En App Store distribution opios, dejamos igual y presionamos Next.
13. En Re-sign Runner, seleccionamos "Automatically manage signing" y presionamos next.
14. Por último, esperaremos y presionaremos el botón export.
15. Seleccionamos a donde quieren ser exportados nuestros archivos y damos en el botón Export.
16. Nos dirigimos a la carpeta creada y ahí encontraremos los archivos para la instalación. El más importante es el archivo .ipa que lo usaremos para instalar en nuestros dispositivos iphone.

ADICIONALES

Para una guía adicional, pueden consultar el siguiente link: <https://github.com/OERS-libraries/laboratorio6>, donde se encuentra el código fuente de la práctica desarrollada.

TAREA DESAFIO

1. Investigar

- a. ¿Qué otros frameworks tienen características similares a flutter?
- b. ¿De qué otras manera pueden mostrar datos de forma gráfica?

FORMATO DE LA PRÁCTICA

El trabajo autónomo será desarrollado en el siguiente formato:

- Nombre del archivo: AMST_Práctica de Laboratorio A_Grupo B_Apellido1_Apellido2_Apellido3
- (*) Siendo A el número de la práctica y B el número del grupo

- Nombre de la materia
- Título de la práctica: Ejemplo: Trabajo Autónomo A - Tema
- Nombre de la profesora
- Número de grupo
- Nombres/Apellidos de los integrantes del grupo que hayan desarrollado el trabajo
- Fecha de inicio y fin del trabajo
- Resultados de las actividades planteadas: Explicación de las actividades ejecutadas, incluyendo las imágenes del proceso. Además, incluir el enlace del repositorio del proyecto en Github y el archivo ejecutable (apk) de la aplicación móvil.
- Conclusiones y Recomendaciones: Respecto a lo aprendido durante el desarrollo del trabajo.
- Referencias bibliográficas: Colocar los documentos, enlaces web o libros consultados.