UNIVERSITÉ DES MASCAREIGNES

SAVOIR, C'EST POUVOIR

Université de Limoges

LAB SESSION 3: MULTI-SCREENS DESIGN USING FLUTTER

To: Mr Shiam Beehary
From: Kritika Bissessur

Contents

1 Task 1

Task-1

Adding a Home Screen widget to your Flutter App
https://codelabs.developers.google.com/flutter-home-screen-widgets#0 **Provide a documentation with a short description and screenshots for all the steps of the lab.**

1.1 Brief Overview

In this task, I have created home Screen widgets for a Flutter app, which will be compatible with both Android and iOS platforms. These widgets will pull data from the Flutter app and display it to the users. I'll also incorporate text using font assets shared from the Flutter app, and include an image of a rendered Flutter widget within the widgets. These widgets will serve as miniature versions of the app, providing users with a view into the app's information without the need to open the app itself. On Android, the widgets will be displayed on the home screen, while on iOS, they can be added to the home screen, lock screen, or the today view.

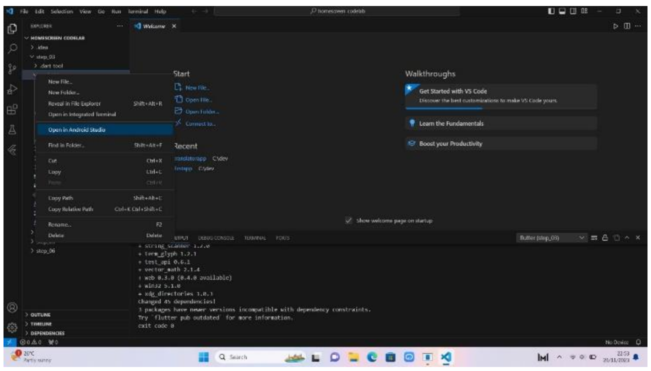
1.2 Steps

1. Add a basic Home Screen widget

First, I will add the Home Screen widget using the native platform tooling.

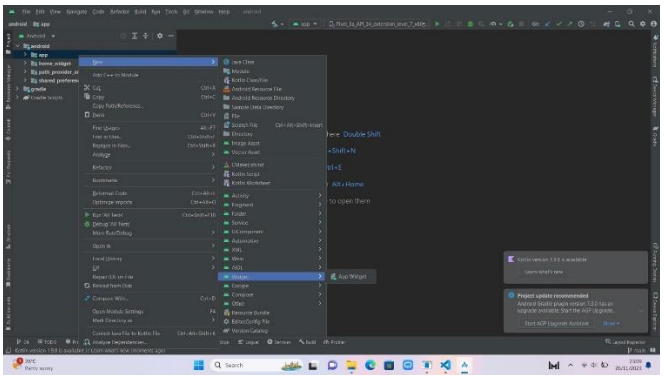
2. Creating a Basic Android Widget

- To add a Home Screen widget in Android, open the project's build file in Android Studio. I can find this file at android/build.gradle. Alternatively, right click on the android folder from VSCode and select Open in Android Studio.



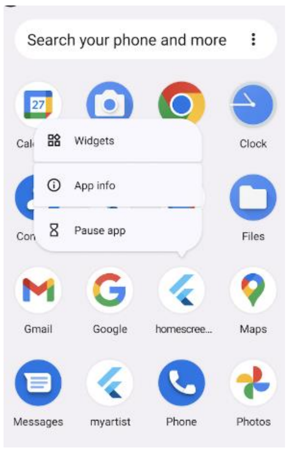
```
+ string_scanner 1.2.0
+ term_glyph 1.2.1
+ test_api 0.6.1
+ vector_math 2.1.4
! web 0.3.0 (0.4.0 available)
+ win32 5.1.0
+ xdg_directories 1.0.3
Changed 45 dependencies!
3 packages have newer versions incompatible with dependency constraints.
Try `flutter pub outdated` for more information.
exit code 0
```

- After the project builds,I locate the app directory in the top-left corner then add my new Home Screen widget to this directory. Right click the directory, select New -> Widget -> App Widget.



3. Debug and test the sample widget

Now, when I run my application,I can see the Home Screen widget. After building the app,I navigate to the application selection screen of my Android device, and long-press the icon for this Flutter project. I then select Widgets from the popup menu.



4. Send data from my Flutter app to my Home Screen widget

Having created the basic Home Screen widget, I am now able to personalize it by updating it to showcase a headline and summary for a news article. The example screenshot below illustrates the Home Screen widget presenting both the headline and summary.

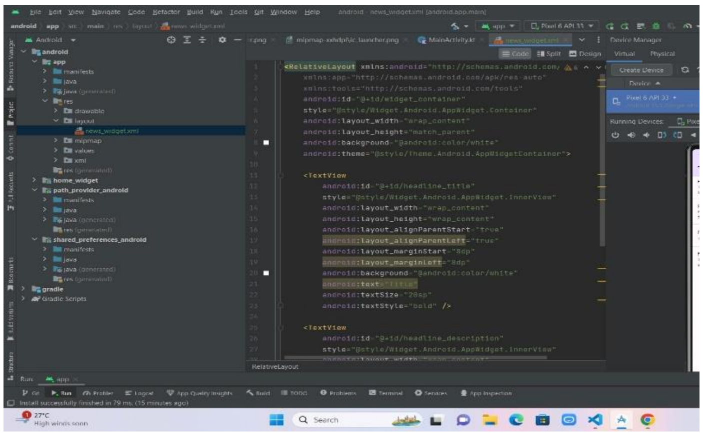
Update the headline and description values

To add the functionality to update my Home Screen widget from my Flutter app, I navigate to the lib/home_screen.dart file then Replace the contents of the file with the following code. Then, replace `<MY APP GROUP>` with the identifier for my App Group.

Call the updateHeadline function when the floatingActionButton is pressed as shown:

Add the Home Screen widget XML.

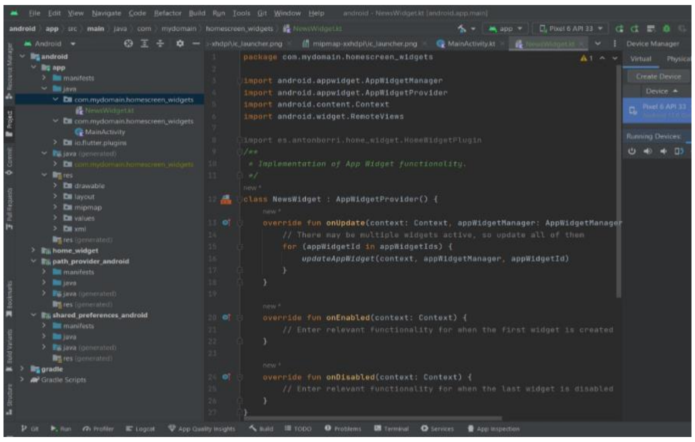
In Android Studio, I will update the files generated in the previous step. Open the res/layout/news_widget.xml file. It defines the structure and layout of my home screen widget. Select "Code" in the top right-hand corner and replace the contents of that file with the following code:



```
<RelativeLayout xmlns:android="http://schemas.android.com/
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/widget_container"
    style="@style/Widget.Android.AppWidget.Container"
    android:layout_width="wrap_content"
    android:layout_height="match_parent"
    android:background="@android:color/white"
    android:theme="@style/Theme.Android.AppWidgetContainer">

    <TextView
        android:id="@+id/headline_title"
        style="@style/Widget.Android.AppWidget.InnerView"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentStart="true"
        android:layout_alignParentLeft="true"
        android:layout_marginStart="8dp"
        android:layout_marginLeft="8dp"
        android:background="@android:color/white"
        android:text="Title"
        android:textSize="20sp"
        android:textStyle="bold" />

    <TextView
        android:id="@+id/headline_description"
        style="@style/Widget.Android.AppWidget.InnerView"
```

Update NewsWidget functionality

Here,I open the NewsWidget.kt Kotlin source code file. This file contains a generated class called NewsWidget that extends the AppWidgetProvider class. I will add my modifications here.



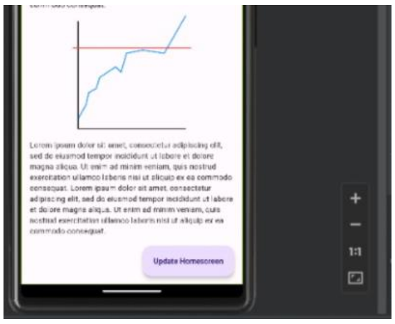
```kotlin
package com.mydomain.homescreen_widgets

import android.appwidget.AppWidgetManager
import android.appwidget.AppWidgetProvider
import android.content.Context
import android.widget.RemoteViews

import es.antonborri.home_widget.HomeWidgetPlugin
/**
 * Implementation of App Widget functionality.
 */
class NewsWidget : AppWidgetProvider() {
    override fun onUpdate(context: Context, appWidgetManager: AppWidgetManager
        // There may be multiple widgets active, so update all of them
        for (appWidgetId in appWidgetIds) {
            updateAppWidget(context, appWidgetManager, appWidgetId)
        }
    }

    override fun onEnabled(context: Context) {
        // Enter relevant functionality for when the first widget is created
    }

    override fun onDisabled(context: Context) {
        // Enter relevant functionality for when the last widget is disabled
    }
}
```

The NewsWidget class contains three methods from its superclass. I'll modify the onUpdate method. Android calls this method for widgets at fixed intervals.

Note: Android requests that the widgets "update themselves" by calling

5. Test the updates

In this section,I test the app to make sure that my Home Screen widgets update with new data. To update the data,I use the Update Home Screen FloatingActionButton on the article pages. My Home Screen widget should update with the article title.

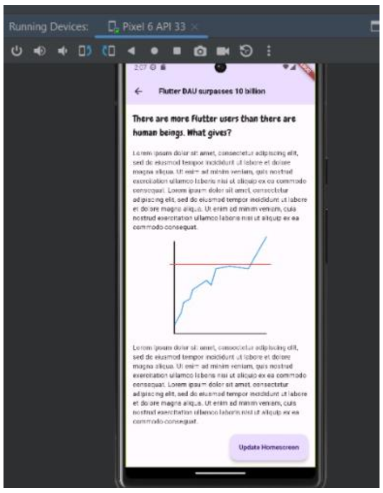


Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat.

6. Rendering Flutter widgets as an image

This widget provides a greater challenge than the text that I've displayed on the homescreen. It's far easier to display the Flutter chart as an image rather than try to recreate it using native UI components.

1.3 Final Output



There are more Flutter users than there are human beings. What gives?

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat.

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat.

2 Task 2

Task-2

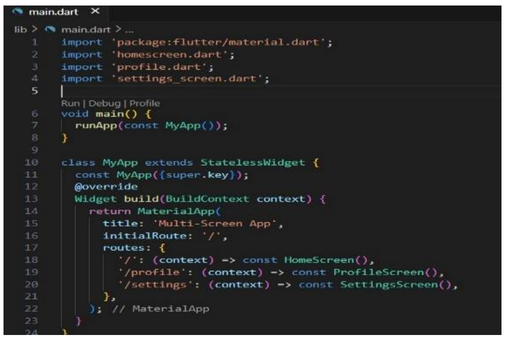
Use the given dart files to complete the mutli-screen application in Flutter using the routing and tabular navigation.
Provide a documentation with a short description and screenshots for all the steps of the lab.

2.1 Brief overview

The task involves creating a Flutter app with a modular and organized structure, featuring a home screen, a profile screen, and a settings screen, each defined in separate Dart files. The home screen provides a welcoming interface with a text message and a floating action button for navigating to the profile screen. Additionally, the app incorporates a tab-based navigation system, implemented through TabBar and TabBarView components, enabling users to switch between tabs, each associated with unique content. The overall objective is to build an intuitive and user-friendly mobile application with a multi-screen laIt and seamless tab-based navigation.

2.2 App Main code

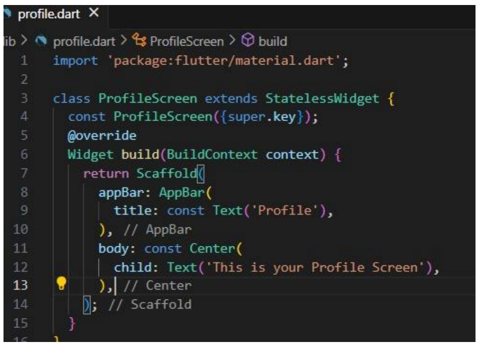
This Flutter code defines a multi-screen app with three screens: HomeScreen, ProfileScreen, and SettingsScreen. The main method initializes the app by running the MyApp widget. MyApp is a StatelessWidget that builds a MaterialApp, the root widget of the application. The MaterialApp sets the app's title and defines initial and named routes for navigation. The initialRoute is set to '/', indicating that HomeScreen will be the first screen. The routes map associates route names with corresponding screen widgets. In this case, '/profile' leads to ProfileScreen, and '/settings' leads to SettingsScreen. Each screen is likely implemented in separate Dart files. This code structure enables easy navigation between screens in the app.



```dart
import 'package:flutter/material.dart';
import 'homescreen.dart';
import 'profile.dart';
import 'settings_screen.dart';

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

class MyApp extends StatelessWidget {
  const MyApp({super.key});
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'Multi-Screen App',
      initialRoute: '/',
      routes: {
        '/': (context) => const HomeScreen(),
        '/profile': (context) => const ProfileScreen(),
        '/settings': (context) => const SettingsScreen(),
      },
    ); // MaterialApp
  }
}
```

2.3 ProfileScreen Widget

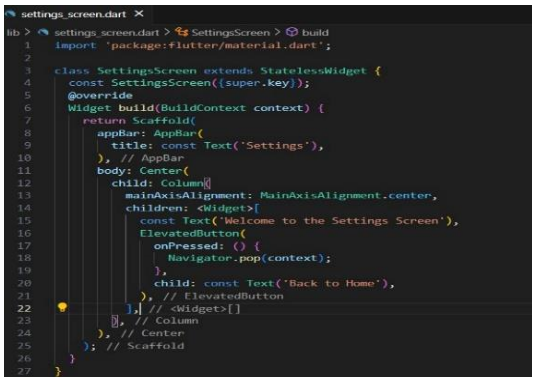
This Flutter code defines the ProfileScreen widget, representing a screen in the multi-screen app. The ProfileScreen is a stateless widget that inherits from StatelessWidget. It includes a Scaffold widget, providing a basic app structure with an AppBar at the top and a body section. The AppBar displays the title 'Profile', and the body contains a centered text widget with the message 'This is my Profile Screen'. This code is part of a larger Flutter application, likely with navigation from other screens to the ProfileScreen. The simplicity of this widget makes it suitable for displaying profile-related information within the app.



```
profile.dart ✕
lib > profile.dart > ProfileScreen > build
import 'package:flutter/material.dart';

class ProfileScreen extends StatelessWidget {
  const ProfileScreen({super.key});
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: const Text('Profile'),
      ), // AppBar
      body: const Center(
        child: Text('This is your Profile Screen'),
      ), // Center
    ); // Scaffold
  }
```

2.4 SettingsScreen Widget

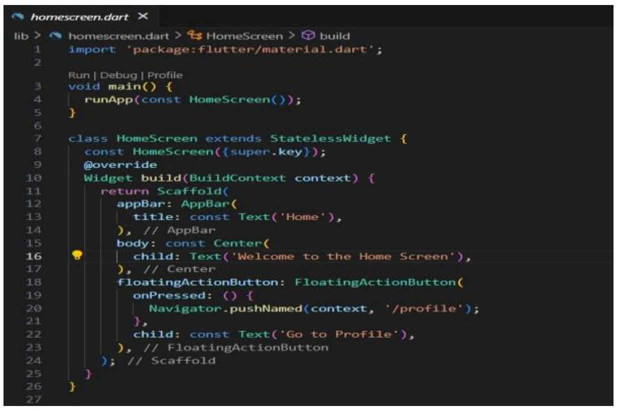
This Flutter code defines the SettingsScreen widget, serving as a screen within a multi-screen app. The SettingsScreen is implemented as a stateless widget and utilizes the Scaffold widget to structure the app. The screen features an AppBar at the top with the title 'Settings'. The body of the screen is a Center widget containing a Column with two child widgets: a text widget welcoming the user to the Settings Screen and an elevated button labeled 'Back to Home'. When the button is pressed, it triggers the Navigator.pop(context) method, allowing the user to navigate back to the previous screen (likely the HomeScreen). This code snippet exemplifies a simple settings screen with a clear and accessible navigation option to return to the main screen of the application.



```
settings_screen.dart ✕
lib > settings_screen.dart > SettingsScreen > build
import 'package:flutter/material.dart';

class SettingsScreen extends StatelessWidget {
  const SettingsScreen({super.key});
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: const Text('Settings'),
      ), // AppBar
      body: Center(
        child: Column(
          mainAxisAlignment: MainAxisAlignment.center,
          children: <Widget>[
            const Text('Welcome to the Settings Screen'),
            ElevatedButton(
              onPressed: () {
                Navigator.pop(context);
              },
              child: const Text('Back to Home'),
            ), // ElevatedButton
          ], // <Widget>[]
        ), // Column
      ), // Center
    ); // Scaffold
  }
}
```

2.5 HomeScreen Widget

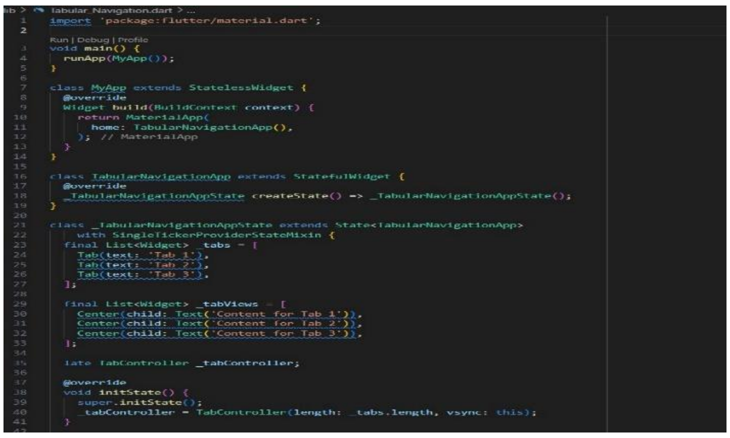
The HomeScreen widget serves as the primary screen in a Flutter application, offering a welcoming interface. The top app bar, generated by the AppBar widget, features the title 'Home.' The screen's body, enclosed within the Scaffold widget, contains a centered text message: 'Welcome to the Home Screen.' Additionally, a floating action button is provided with the label 'Go to Profile.' When pressed, this button triggers a navigation event, directing users to the '/profile' route. Overall, the HomeScreen creates a simple and navigable entry point for users, encouraging them to explore other parts of the application, particularly the profile screen.



```
homescreen.dart ×
lib > homescreen.dart > HomeScreen > build
import 'package:flutter/material.dart';

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

class HomeScreen extends StatelessWidget {
  const HomeScreen({super.key});
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: const Text('Home'),
      ), // AppBar
      body: const Center(
        child: Text('Welcome to the Home Screen'),
      ), // Center
      floatingActionButton: FloatingActionButton(
        onPressed: () {
          Navigator.pushNamed(context, '/profile');
        },
        child: const Text('Go to Profile'),
      ), // FloatingActionButton
    ); // Scaffold
  }
}
```

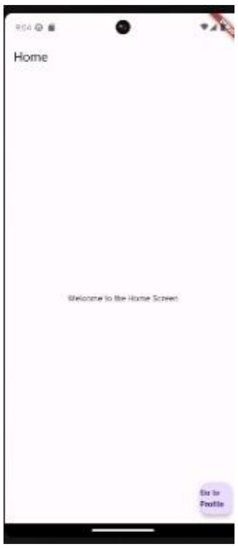
2.6 Tabular Naviagtion

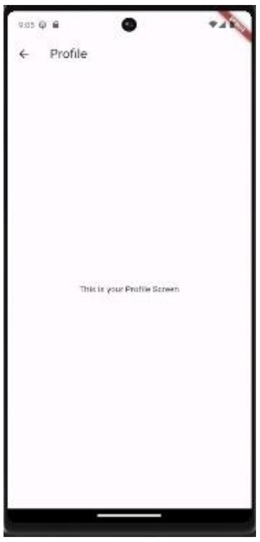
The TabularNavigationApp is implemented across multiple files (tab_navigation.dart, home_screen.dart, profile.dart, and settings_screen.dart). It features a tab-based navigation system using the TabBar and TabBarView components. The tabs ('Tab 1', 'Tab 2', 'Tab 3') in the app correspond to different screens with distinct content. The content for each tab is defined in the _tabViews list within the _TabularNavigationAppState class. Users can navigate between tabs, each associated with a specific screen, creating a seamless and organized user experience.

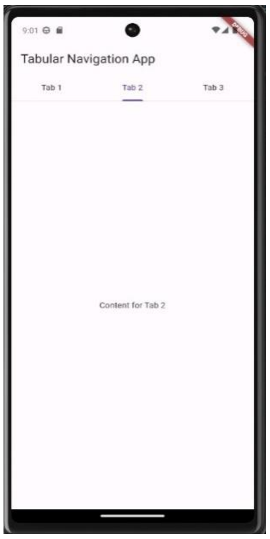


```
lib > Tabular_Navigation.dart > ...
import 'package:flutter/material.dart';

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

class MyApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      home: TabularNavigationApp(),
    ); // MaterialApp
  }
}

class TabularNavigationApp extends StatefulWidget {
  @override
  _TabularNavigationAppState createState() => _TabularNavigationAppState();
}

class _TabularNavigationAppState extends State<TabularNavigationApp>
    with SingleTickerProviderStateMixin {
  final List<Widget> _tabs = [
    Tab(text: 'Tab 1'),
    Tab(text: 'Tab 2'),
    Tab(text: 'Tab 3'),
  ];

  final List<Widget> _tabViews = [
    Center(child: Text('Content for Tab 1')),
    Center(child: Text('Content for Tab 2')),
    Center(child: Text('Content for Tab 3')),
  ];

  late TabController _tabController;

  @override
  void initState() {
    super.initState();
    _tabController = TabController(length: _tabs.length, vsync: this);
  }
```

2.7 Final Output







In summary, the app provides a multi-screen Flutter application with tab-based navigation, allowing users to switch between different screens, each offering unique content and functionality.

3 Task 3

Task-3

Adding AdMob ads to a Flutter app
(I) https://codelabs.developers.google.com/codelabs/admob-ads-in-flutter#0 (ii) https://codelabs.developers.google.com/codelabs/admob-inline-ads-in-flutter#0 **Provide a documentation with a short description and screenshots for all the steps of the lab.**

3.1 Brief overview

This codelab is a comprehensive guide for implementing AdMob ads in a Flutter app. It consists of two main parts: the first involving the addition of a banner, interstitial, and rewarded ad to an app named Awesome Drawing Quiz, which allows players to guess the name of the drawing. The second part revolves around implementing an AdMob inline banner and AdMob native inline ads in a Flutter app using the Google Mobile Ads plugin for Flutter.

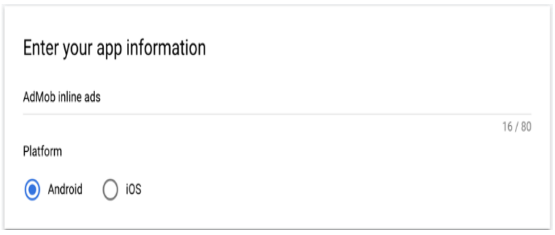
Throughout the codelab, i learned how to configure the Google Mobile Ads AdMob plugin, implement various ad formats including banner, interstitial, rewarded ads, as well as inline banner and native inline ads in their Flutter app. Key takeaways include understanding the setup and integration process for the AdMob plugin in Flutter, and gaining practical knowledge of ad implementation within the app.

3.2 Steps

1. Set up the AdMob app and ad units

Set up for Android

To set up for Android, I need to add an Android app and create ad units.



Enter your app information

AdMob inline ads

16 / 80

Platform

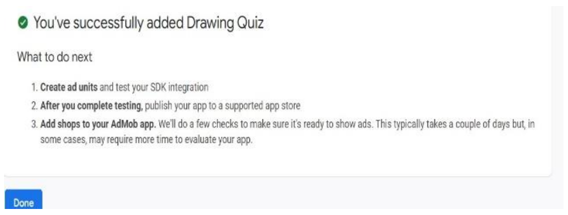
[✓ Android] iOS

Add an Android app

In the AdMob console, click ADD APP from the Apps menu.

When I'm asked ‘Have I publishedmy app on Google Play or the App Store?’, click NO.

Enter Awesome Drawing Quiz in the app name field, and select Android as the platform.



You've successfully added Drawing Quiz

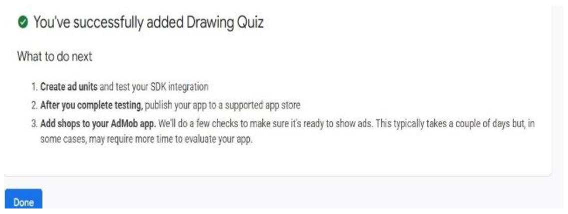
What to do next

1. **Create ad units** and test your SDK integration
2. **After you complete testing,** publish your app to a supported app store
3. **Add shops to your AdMob app.** We'll do a few checks to make sure it's ready to show ads. This typically takes a couple of days but, in some cases, may require more time to evaluate your app.

Done

Create ad units

To start adding ad units to AdMob:

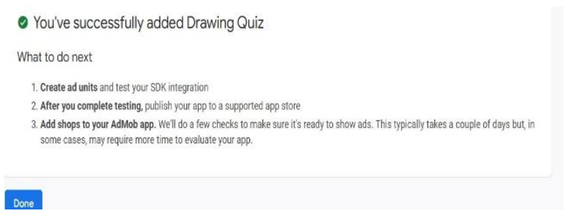


You've successfully added Drawing Quiz

What to do next

1. **Create ad units** and test your SDK integration
2. **After you complete testing,** publish your app to a supported app store
3. **Add shops to your AdMob app.** We'll do a few checks to make sure it's ready to show ads. This typically takes a couple of days but, in some cases, may require more time to evaluate your app.

Done

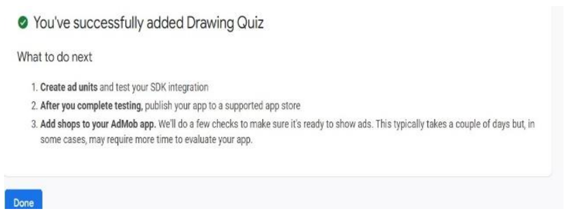


You've successfully added Drawing Quiz

What to do next

1. **Create ad units** and test your SDK integration
2. **After you complete testing,** publish your app to a supported app store
3. **Add shops to your AdMob app.** We'll do a few checks to make sure it's ready to show ads. This typically takes a couple of days but, in some cases, may require more time to evaluate your app.

Done



You've successfully added Drawing Quiz

What to do next

1. **Create ad units** and test your SDK integration
2. **After you complete testing,** publish your app to a supported app store
3. **Add shops to your AdMob app.** We'll do a few checks to make sure it's ready to show ads. This typically takes a couple of days but, in some cases, may require more time to evaluate your app.

Done

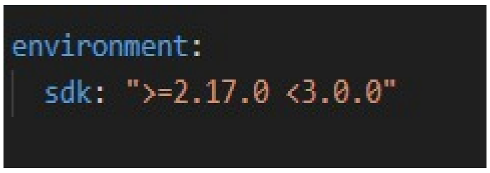
Select Awesome Drawing Quiz from the Apps menu in the AdMob console.

Click the Ad units menu.

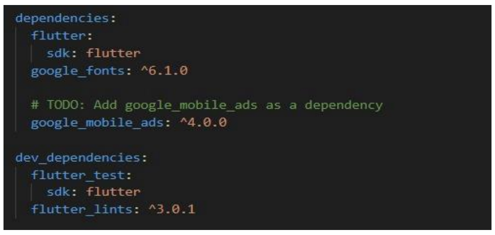
2. Add the Google Mobile Ads Flutter plugin

Add the Google Mobile Ads plugin as a dependency

To access the AdMob APIs from the AdMob inline ads project, i add google_mobile_ads as a dependency to the pubspec.yaml file located at the root of the project.

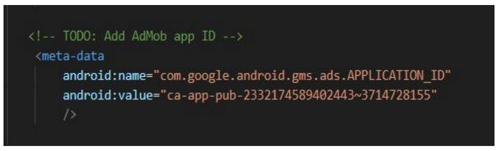


```yaml
environment:
  sdk: ">=2.17.0 <3.0.0"
```



```yaml
dependencies:
  flutter:
    sdk: flutter
  google_fonts: ^6.1.0

  # TODO: Add google_mobile_ads as a dependency
  google_mobile_ads: ^4.0.0

dev_dependencies:
  flutter_test:
    sdk: flutter
  flutter_lints: ^3.0.1
```

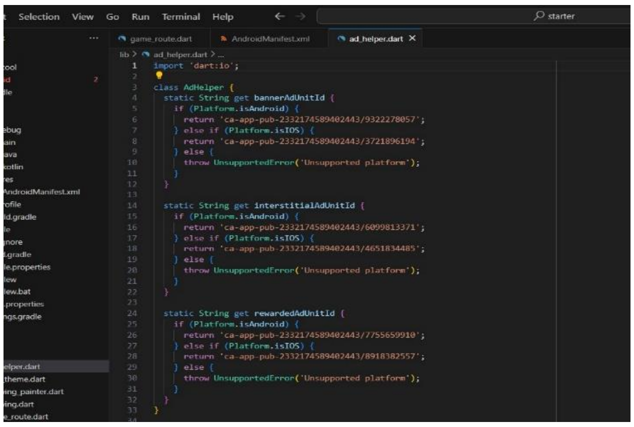
I then update AndroidManifest.xml (Android)



```xml
<!-- TODO: Add AdMob app ID -->
<meta-data
    android:name="com.google.android.gms.ads.APPLICATION_ID"
    android:value="ca-app-pub-2332174589402443~3714728155"
    />
```

3. Add a helper class for ads

Here,I create a new file named ad_helper.dart under the lib directory. Then, implement the AdHelper class, which provides an AdMob app ID and ad unit IDs for Android and iOS.

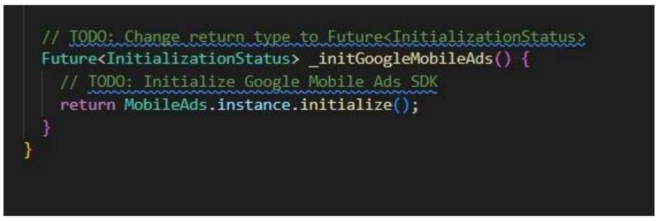


Selection View Go Run Terminal Help
starter
game_route.dart
AndroidManifest.xml
ad_helper.dart
lib > ad_helper.dart > ...

```dart
import 'dart:io';

class AdHelper {
  static String get bannerAdUnitId {
    if (Platform.isAndroid) {
      return 'ca-app-pub-2332174589402443/9322278057';
    } else if (Platform.isIOS) {
      return 'ca-app-pub-2332174589402443/3721896194';
    } else {
      throw UnsupportedError('Unsupported platform');
    }
  }

  static String get interstitialAdUnitId {
    if (Platform.isAndroid) {
      return 'ca-app-pub-2332174589402443/6099813371';
    } else if (Platform.isIOS) {
      return 'ca-app-pub-2332174589402443/4651834485';
    } else {
      throw UnsupportedError('Unsupported platform');
    }
  }

  static String get rewardedAdUnitId {
    if (Platform.isAndroid) {
      return 'ca-app-pub-2332174589402443/7755659910';
    } else if (Platform.isIOS) {
      return 'ca-app-pub-2332174589402443/8918382557';
    } else {
      throw UnsupportedError('Unsupported platform');
    }
  }
}
```

4. Initialize the Google Mobile Ads SDK

Before loading ads, I need to initialize the Google Mobile Ads SDK. Open the lib/home_route.dart file, and modify _initGoogleMobileAds() to initialize the SDK before the home page is loaded.

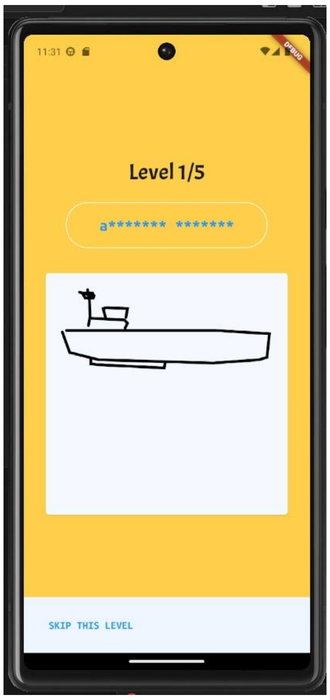


```
  // TODO: Change return type to Future<InitializationStatus>
  Future<InitializationStatus> _initGoogleMobileAds() {
    // TODO: Initialize Google Mobile Ads SDK
    return MobileAds.instance.initialize();
  }
}
```

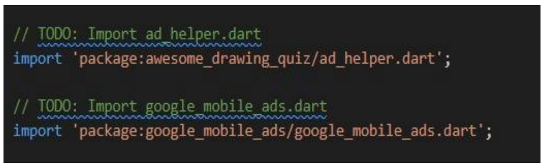
Note that I need to change the return type of the _initGoogleMobileAds() method from Future<dynamic> to Future<InitializationStatus> to get the SDK initialization result after it completes.

5. Add a banner ad

In this section, I display a banner ad at the top of the game screen, as shown in the following screenshot.

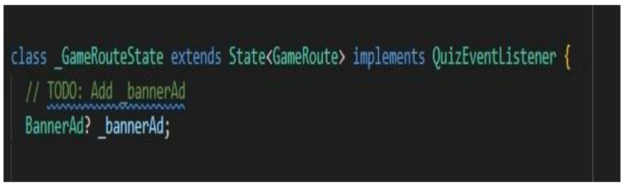


Here,I open the lib/game_route.dart file, and I will import ad_manager.dart. Additionally, I will import the ad_helper.dart and google_mobile_ads.dart by adding the following lines:



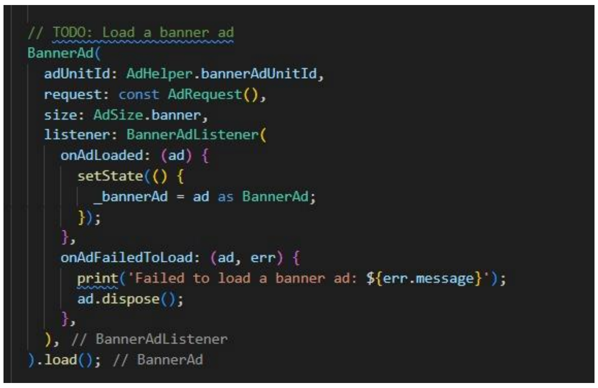
```
// TODO: Import ad_helper.dart
import 'package:awesome_drawing_quiz/ad_helper.dart';

// TODO: Import google_mobile_ads.dart
import 'package:google_mobile_ads/google_mobile_ads.dart';
```

In the _GameRouteState class, i then add the following members for a banner ad.



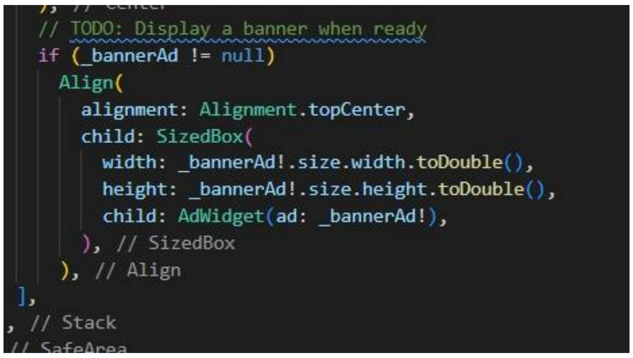
```
class _GameRouteState extends State<GameRoute> implements QuizEventListener {
  // TODO: Add _bannerAd
  BannerAd? _bannerAd;
```

In the initState() method, I create and load a BannerAd for the 320x50 banner (AdSize.banner). **Note that an ad event listener is configured to update the UI (setState()) when an ad is loaded .**



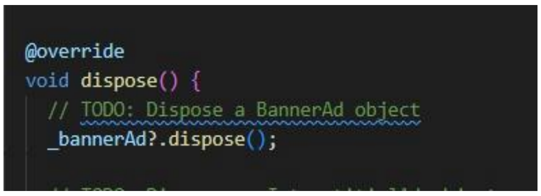
```
// TODO: Load a banner ad
BannerAd(
  adUnitId: AdHelper.bannerAdUnitId,
  request: const AdRequest(),
  size: AdSize.banner,
  listener: BannerAdListener(
    onAdLoaded: (ad) {
      setState(() {
        _bannerAd = ad as BannerAd;
      });
    },
    onAdFailedToLoad: (ad, err) {
      print('Failed to load a banner ad: ${err.message}');
      ad.dispose();
    },
  ), // BannerAdListener
).load(); // BannerAd
```

Then,I modify the build() method to display a banner ad when available.



```
  // TODO: Display a banner when ready
  if (_bannerAd != null)
    Align(
      alignment: Alignment.topCenter,
      child: SizedBox(
        width: _bannerAd!.size.width.toDouble(),
        height: _bannerAd!.size.height.toDouble(),
        child: AdWidget(ad: _bannerAd!),
      ), // SizedBox
    ), // Align
],
, // Stack
// SafeArea
```

I then release the resource associated with the BannerAd object by calling the BannerAd.dispose() method in the dispose() callback method.

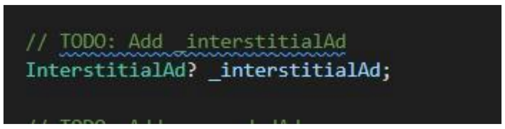


```
@override
void dispose() {
  // TODO: Dispose a BannerAd object
  _bannerAd?.dispose();
```

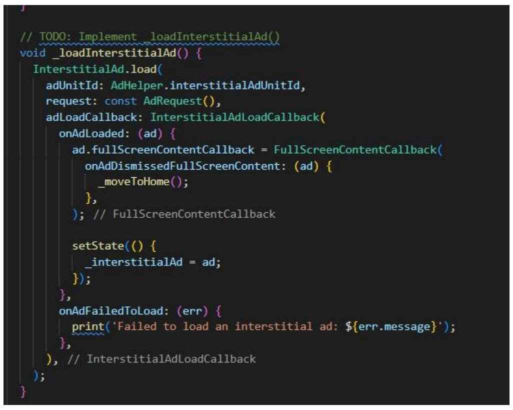
6. Add an interstitial ad

In this section, I display an interstitial ad after the game (5 levels in total) finishes.

I open the lib/game_route.dart file and within the _GameRouteState class, I add the following members and methods to incorporate an interstitial ad.



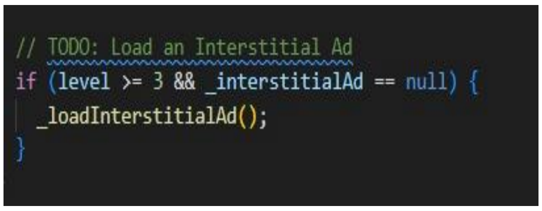
```dart
// TODO: Add _interstitialAd
InterstitialAd? _interstitialAd;
```



```dart
// TODO: Implement _loadInterstitialAd()
void _loadInterstitialAd() {
  InterstitialAd.load(
    adUnitId: AdHelper.interstitialAdUnitId,
    request: const AdRequest(),
    adLoadCallback: InterstitialAdLoadCallback(
      onAdLoaded: (ad) {
        ad.fullScreenContentCallback = FullScreenContentCallback(
          onAdDismissedFullScreenContent: (ad) {
            _moveToHome();
          },
        ); // FullScreenContentCallback

        setState(() {
          _interstitialAd = ad;
        });
      },
      onAdFailedToLoad: (err) {
        print('Failed to load an interstitial ad: ${err.message}');
      },
    ), // InterstitialAdLoadCallback
  );
}
```

In this codelab, an interstitial ad is displayed after a user completes 5 levels. To minimize unnecessary ad requests, request an ad when a user reaches level 3.

In the onNewLevel() method, I add the following lines.

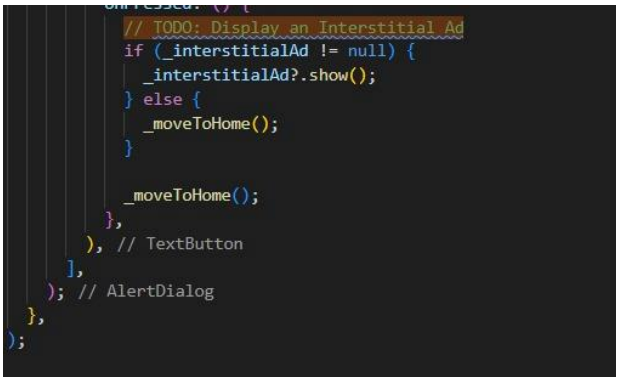


```dart
// TODO: Load an Interstitial Ad
if (level >= 3 && _interstitialAd == null) {
  _loadInterstitialAd();
}
```

When a game finishes, the game score dialog is displayed. When a user closes the dialog, it routes a user to the home screen of the Awesome Drawing Quiz.

Because interstitial ads should be displayed between screen transitions, we show the interstitial ad when a user clicks the CLOSE button.

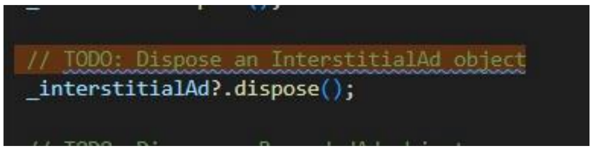
Here,I modify the onGameOver() method as follows.



```
      // TODO: Display an Interstitial Ad
      if (_interstitialAd != null) {
        _interstitialAd?.show();
      } else {
        _moveToHome();
      }

      _moveToHome();
    },
  ), // TextButton
 ],
); // AlertDialog
},
);
```

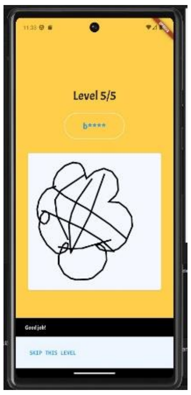
Note that an ad event listener is configured to check whether the ad is ready (onAdLoaded() and onAdFailedToLoad()) and to display the app's home screen when the ad is closed (onAdDismissedFullScreenContent())

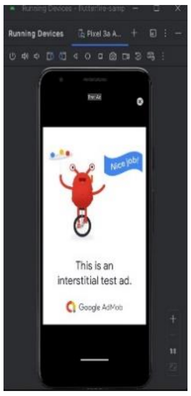
Release the resource associated with the InterstitialAd object by calling the InterstitialAd.dispose() method in the dispose() callback method.



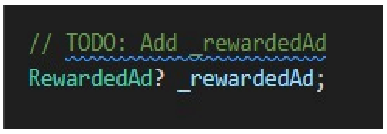
```
// TODO: Dispose an InterstitialAd object
_interstitialAd?.dispose();
```

When I run the project,I notice that the interstitial add

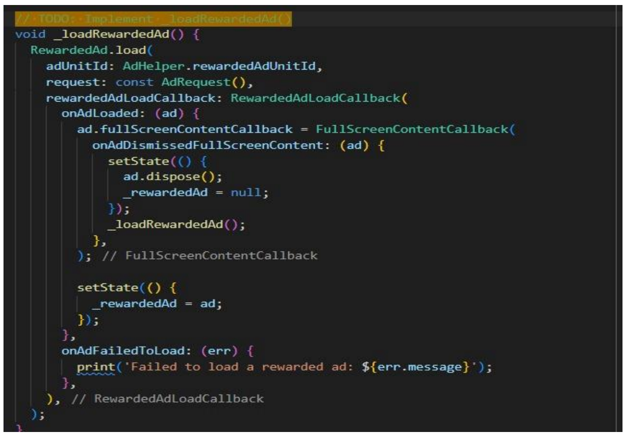




I open the lib/game_route.dart file and, within the _GameRouteState class, I add the following members and methods to implement an interstitial ad.



```
// TODO: Add _rewardedAd
RewardedAd? _rewardedAd;
```

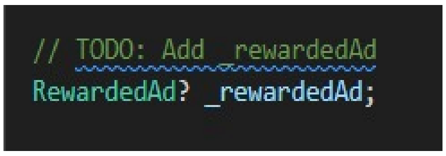


```dart
// TODO: Implement _loadRewardedAd()
void _loadRewardedAd() {
  RewardedAd.load(
    adUnitId: AdHelper.rewardedAdUnitId,
    request: const AdRequest(),
    rewardedAdLoadCallback: RewardedAdLoadCallback(
      onAdLoaded: (ad) {
        ad.fullScreenContentCallback = FullScreenContentCallback(
          onAdDismissedFullScreenContent: (ad) {
            setState(() {
              ad.dispose();
              _rewardedAd = null;
            });
            _loadRewardedAd();
          },
        ); // FullScreenContentCallback

        setState(() {
          _rewardedAd = ad;
        });
      },
      onAdFailedToLoad: (err) {
        print('Failed to load a rewarded ad: ${err.message}');
      },
    ), // RewardedAdLoadCallback
  );
}
```

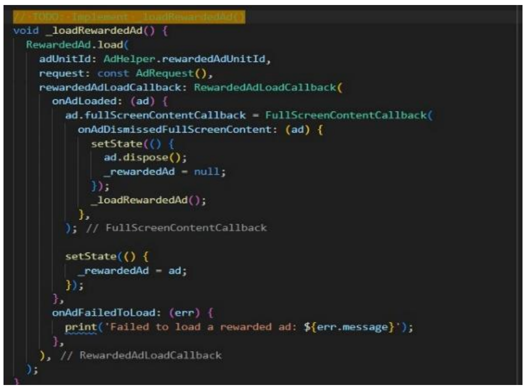
7. Add a rewarded ad

In this section, I will add a rewarded ad which gives a user an additional hint as a reward.

Open the lib/game_route.dart file
In the _GameRouteState class, add members for a rewarded ad, and implement _loadRewardedAd() method. Note that it loads another rewarded ad when the ad is closed (onAdDismissedFullScreenContent) to cache the ad as early as possible.

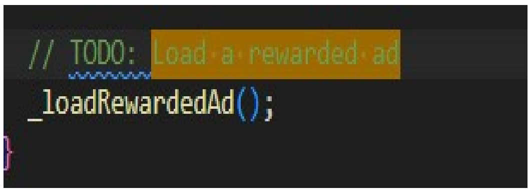


```dart
// TODO: Add _rewardedAd
RewardedAd? _rewardedAd;
```



```dart
// TODO: Implement _loadRewardedAd()
void _loadRewardedAd() {
  RewardedAd.load(
    adUnitId: AdHelper.rewardedAdUnitId,
    request: const AdRequest(),
    rewardedAdLoadCallback: RewardedAdLoadCallback(
      onAdLoaded: (ad) {
        ad.fullScreenContentCallback = FullScreenContentCallback(
          onAdDismissedFullScreenContent: (ad) {
            setState(() {
              ad.dispose();
              _rewardedAd = null;
            });
            _loadRewardedAd();
          },
        ); // FullScreenContentCallback

        setState(() {
          _rewardedAd = ad;
        });
      },
      onAdFailedToLoad: (err) {
        print('Failed to load a rewarded ad: ${err.message}');
      },
    ), // RewardedAdLoadCallback
  );
}
```

Call _loadRewardedAd() from initState() method to request a rewarded ad when the game starts.

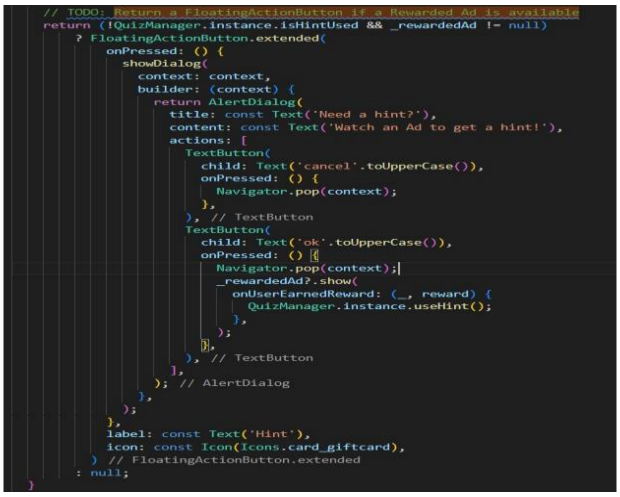


```
  // TODO: Load a rewarded ad
  _loadRewardedAd();
}
```

Allow users to watch a rewarded ad by clicking the floating action button. The button shows only if a user hasn't used a hint at the current level and a rewarded ad is loaded.
Modify the _buildFloatingActionButton() method, as follows, to display the floating action button. Note that returning null hides the button from the screen.

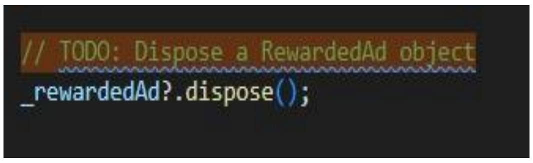
Note that onUserEarnedReward is the most important ad event in a rewarded ad. It's triggered when a user becomes eligible to receive a reward (for example., finished watching a video).

In this codelab, the QuizManager.instance.useHint() method is called from the callback, which reveals one more character in the hint string. The app reloads a rewarded ad in the onAdClosed callback to make sure the ad is ready as early as possible.



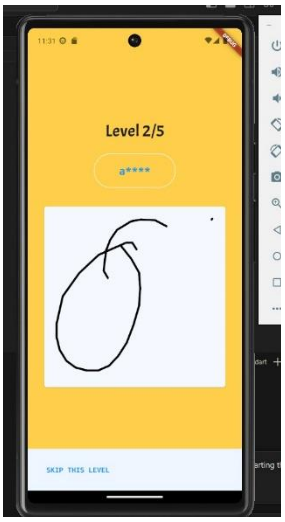
```
    // TODO: Return a FloatingActionButton if a Rewarded Ad is available
    return (!QuizManager.instance.isHintUsed && _rewardedAd != null)
        ? FloatingActionButton.extended(
            onPressed: () {
              showDialog(
                context: context,
                builder: (context) {
                  return AlertDialog(
                    title: const Text('Need a hint?'),
                    content: const Text('Watch an Ad to get a hint!'),
                    actions: [
                      TextButton(
                        child: Text('cancel'.toUpperCase()),
                        onPressed: () {
                          Navigator.pop(context);
                        },
                      ), // TextButton
                      TextButton(
                        child: Text('ok'.toUpperCase()),
                        onPressed: () {
                          Navigator.pop(context);
                          _rewardedAd?.show(
                            onUserEarnedReward: (_, reward) {
                              QuizManager.instance.useHint();
                            },
                          );
                        },
                      ), // TextButton
                    ],
                  ); // AlertDialog
                },
              );
            },
            label: const Text('Hint'),
            icon: const Icon(Icons.card_giftcard),
          ) // FloatingActionButton.extended
        : null;
  }
```

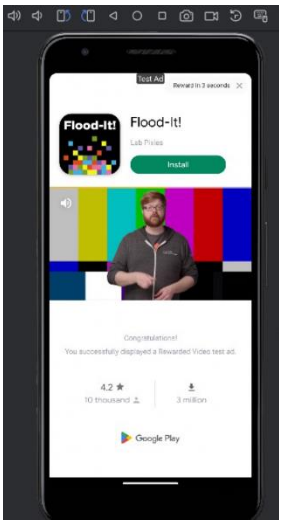
Release the resource associated with the RewardedAd object by calling the RewardedAd.dispose() method in the dispose() callback method.



```
// TODO: Dispose a RewardedAd object
_rewardedAd?.dispose();
```

In this section, I added a rewarded ad which gives a user an additional hint as a reward.





3.3 Brief Overview (part 2)

In this codelab, I have learnt how to integrate AdMob banner and native inline ads into a Flutter app. The process involved utilizing the Google Mobile Ads plugin for Flutter to seamlessly incorporate these ads within my application. By following this guide, I have gained practical experience in implementing both AdMob inline banner and native inline ads, enabling me to effectively monetize my Flutter app while providing a seamless and non-intrusive ad experience.

3.4 Steps (part 2)

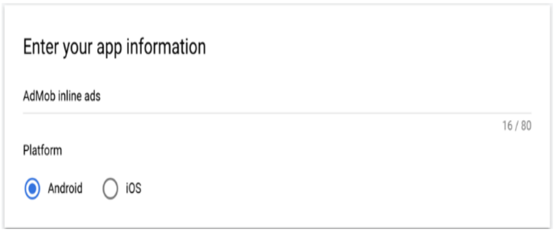
1. Set up AdMob app and ad units

Add an Android app

In the AdMob console, click ADD APP from the Apps menu.

When I're asked Have I published my app on Google Play or the App Store?, click NO.

Enter AdMob inline ads in the app name field, and select Android as the platform.



Enter your app information

AdMob inline ads

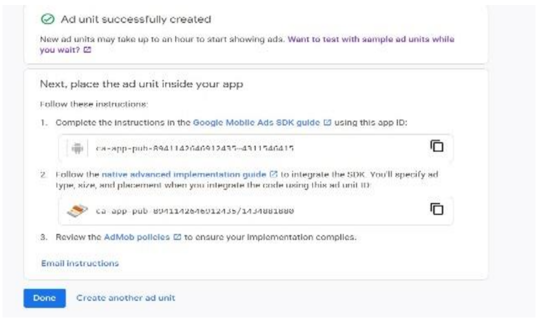
16 / 80

Platform

[✓ Android] iOS

To set up for Android, I need to add an Android app and create ad units:

- native



Ad unit successfully created

New ad units may take up to an hour to start showing ads. Want to test with sample ad units while you wait?

Next, place the ad unit inside your app

Follow these instructions:

1. Complete the instructions in the Google Mobile Ads SDK guide using this app ID:

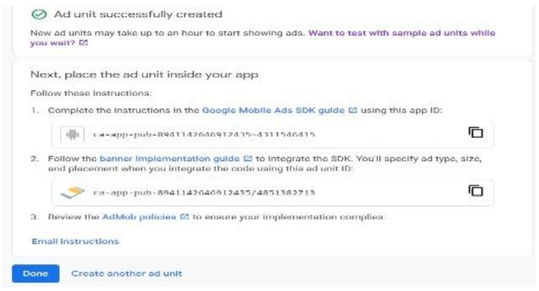
   ca-app-pub-8941142646912435~4311546415

2. Follow the native advanced implementation guide to integrate the SDK. You'll specify ad type, size, and placement when you integrate the code using this ad unit ID:

   ca-app-pub-8941142646912435/1434881880

3. Review the AdMob policies to ensure your implementation complies.

- banner



Ad unit successfully created

New ad units may take up to an hour to start showing ads. Want to test with sample ad units while you wait?

Next, place the ad unit inside your app

Follow these instructions:

1. Complete the instructions in the Google Mobile Ads SDK guide using this app ID:

   ca-app-pub-8941142646912435~4311546415

2. Follow the banner implementation guide to integrate the SDK. You'll specify ad type, size, and placement when you integrate the code using this ad unit ID:

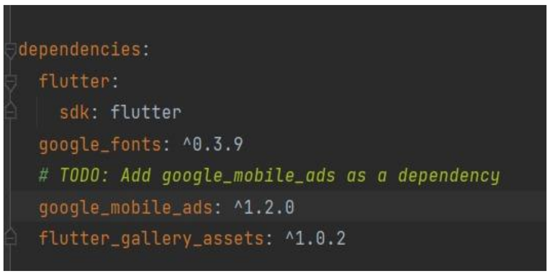
   ca-app-pub-8941142646912435/4851382713

3. Review the AdMob policies to ensure your implementation complies.

2. Add the Google Mobile Ads Flutter plugin

Add the Google Mobile Ads plugin as a dependency

To access the AdMob APIs from the AdMob inline ads project, I add google_mobile_ads as a dependency to the pubspec.yaml file located at the root of the project.

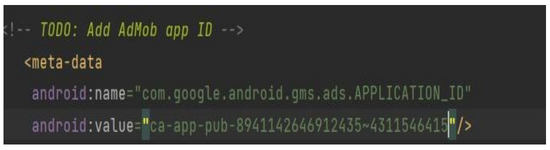


```
dependencies:
  flutter:
    sdk: flutter
  google_fonts: ^0.3.9
  # TODO: Add google_mobile_ads as a dependency
  google_mobile_ads: ^1.2.0
  flutter_gallery_assets: ^1.0.2
```

Update AndroidManifest.xml (Android)

Open the android/app/src/main/AndroidManifest.xml file in Android Studio.

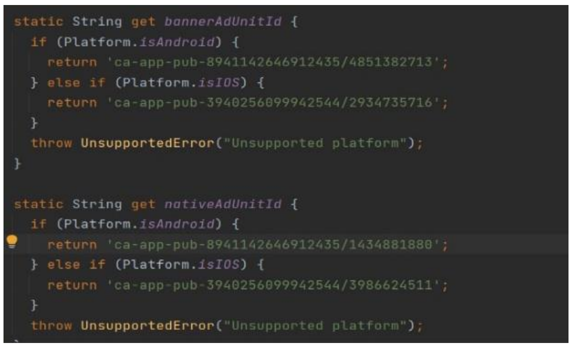
I add my AdMob app ID by adding a <meta-data> tag with the name com.google.android.gms.ads.APPLICATION_ID.



```
<!-- TODO: Add AdMob app ID -->
  <meta-data
   android:name="com.google.android.gms.ads.APPLICATION_ID"
   android:value="ca-app-pub-8941142646912435~4311546415"/>
```

3. Add a helper class for ads

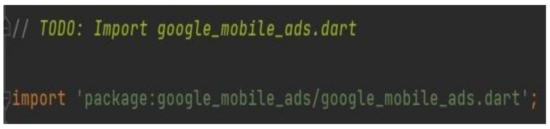
I create a new file named ad_helper.dart under the lib directory. In this file, I implement the AdHelper class, which serves as a utility class providing the AdMob app ID and ad unit IDs for both Android and iOS platforms.



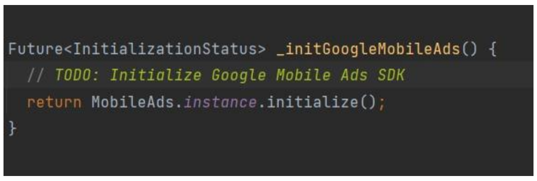
```
static String get bannerAdUnitId {
  if (Platform.isAndroid) {
    return 'ca-app-pub-8941142646912435/4851382713';
  } else if (Platform.isIOS) {
    return 'ca-app-pub-3940256099942544/2934735716';
  }
  throw UnsupportedError("Unsupported platform");
}

static String get nativeAdUnitId {
  if (Platform.isAndroid) {
    return 'ca-app-pub-8941142646912435/1434881880';
  } else if (Platform.isIOS) {
    return 'ca-app-pub-3940256099942544/3986624511';
  }
  throw UnsupportedError("Unsupported platform");
```

4. Initialize the Google Mobile Ads SDK

Before loading ads, I need to initialize the Google Mobile Ads SDK. Open the lib/home_page.dart file, and modify _initGoogleMobileAds() to initialize the SDK before the home page is loaded.



```
// TODO: Import google_mobile_ads.dart

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

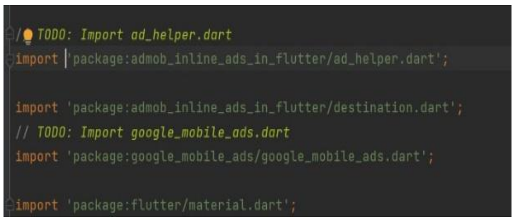


```
Future<InitializationStatus> _initGoogleMobileAds() {
  // TODO: Initialize Google Mobile Ads SDK
  return MobileAds.instance.initialize();
}
```

Note that I changed the return type of the _initGoogleMobileAds() method from Future<dynamic> to Future<InitializationStatus> to get the SDK initialization result after it completes.

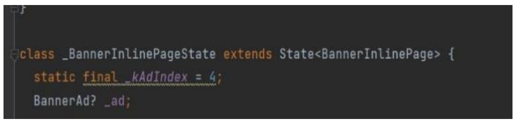
5. Add a banner ad

- Here,I Import ad_helper.dart and google_mobile_ads.dart by adding the following lines:

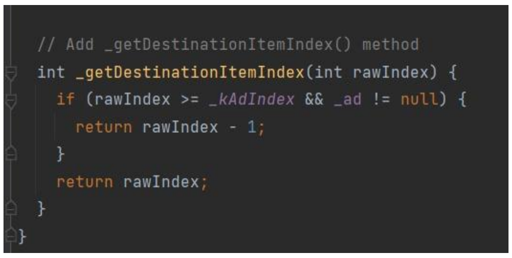


```
// TODO: Import ad_helper.dart
import 'package:admob_inline_ads_in_flutter/ad_helper.dart';

import 'package:admob_inline_ads_in_flutter/destination.dart';
// TODO: Import google_mobile_ads.dart
import 'package:google_mobile_ads/google_mobile_ads.dart';

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

- In the _BannerInlinePageState class, add the following members and methods for a banner ad.

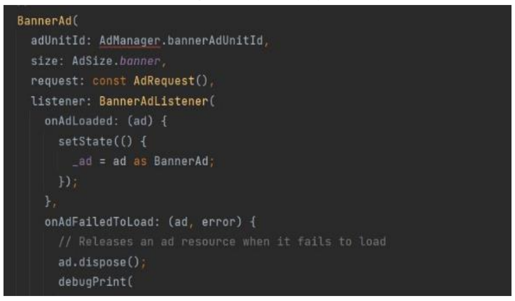


```
}

class _BannerInlinePageState extends State<BannerInlinePage> {
  static final _kAdIndex = 4;
  BannerAd? _ad;
```



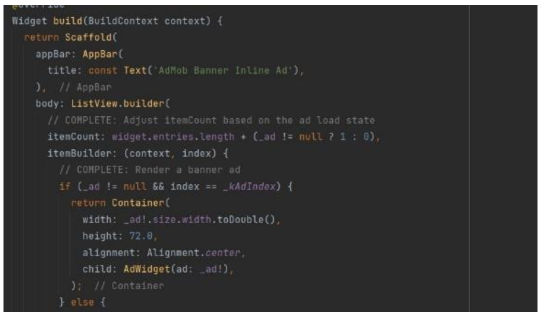
```
  // Add _getDestinationItemIndex() method
  int _getDestinationItemIndex(int rawIndex) {
    if (rawIndex >= _kAdIndex && _ad != null) {
      return rawIndex - 1;
    }
    return rawIndex;
  }
}
```

- In the initState() method, create and load a BannerAd for the 320x50 banner (AdSize.banner). Note that an ad event listener is configured to update the UI (setState()) when an ad is loaded.



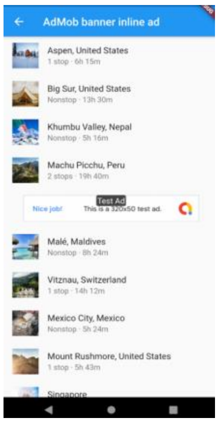
```
BannerAd(
  adUnitId: AdManager.bannerAdUnitId,
  size: AdSize.banner,
  request: const AdRequest(),
  listener: BannerAdListener(
    onAdLoaded: (ad) {
      setState(() {
        _ad = ad as BannerAd;
      });
    },
    onAdFailedToLoad: (ad, error) {
      // Releases an ad resource when it fails to load
      ad.dispose();
      debugPrint(
```

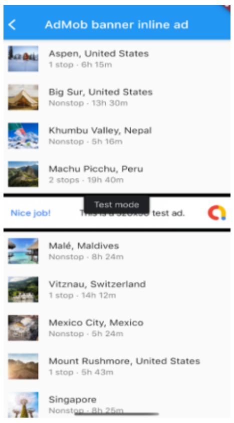
I then update itemCount, to count a banner ad entry, and update itemBuilder, to render a banner ad at the ad index (_kAdIndex) when the ad is loaded and the code to use the _getDestinationItemIndex() method to retrieve an index for the content item.



```
Widget build(BuildContext context) {
  return Scaffold(
    appBar: AppBar(
      title: const Text('AdMob Banner Inline Ad'),
    ), // AppBar
    body: ListView.builder(
      // COMPLETE: Adjust itemCount based on the ad load state
      itemCount: widget.entries.length + (_ad != null ? 1 : 0),
      itemBuilder: (context, index) {
        // COMPLETE: Render a banner ad
        if (_ad != null && index == _kAdIndex) {
          return Container(
            width: _ad!.size.width.toDouble(),
            height: 72.0,
            alignment: Alignment.center,
            child: AdWidget(ad: _ad!),
          ); // Container
        } else {
```

When I run the app,this is the following output I obtain





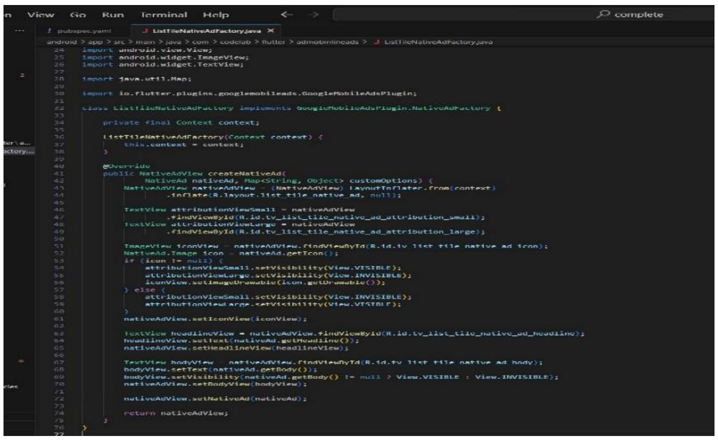
6. Add a native ad

- Create a native ad laIt

1. With the Android project opened, right-click **app** from the project pane in Android Studio, and select **New > Android Resource File** from the context menu.
2. In the **New Resource File** dialog, enter list_tile_native_ad.xml as the file name.
3. Select **LaIt** as the resource type, and enter com.google.android.gms.ads.nativead.NativeAdView as a root element.
4. Click **OK** to create a new layout file.

- Create the ListTileNativeAdFactory class
- In the Project pane, right-click the com.codelab.flutter.admobinlineads package, and select New > Java Class.
- Enter ListTileNativeAdFactory as the name, and select **Class** from the list.
    - After the New Class dialog appears, leave everything empty, and click OK.
    - I'll see that the ListTileNativeAdFactory class is created in the package com.codelab.flutter.admobinlineads.

ListTileNativeAdFactory.java

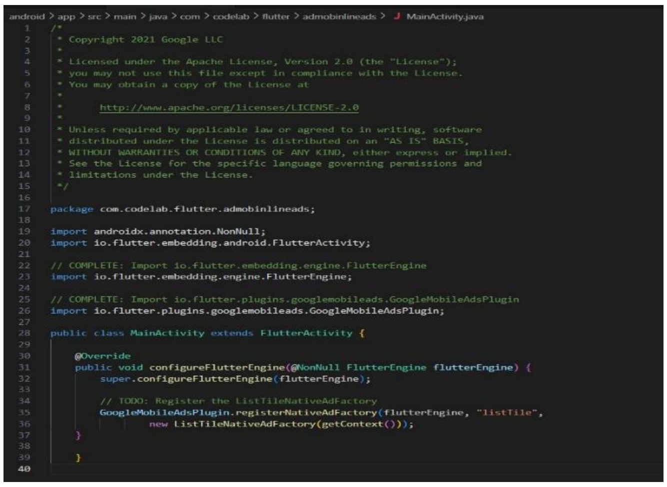

View Go Run Terminal Help
complete
pubspec.yaml
ListTileNativeAdFactory.java
android > app > src > main > java > com > codelab > flutter > admobinlineads > ListTileNativeAdFactory.java

```java
import android.view.View;
import android.widget.ImageView;
import android.widget.TextView;

import java.util.Map;

import io.flutter.plugins.googlemobileads.GoogleMobileAdsPlugin;

class ListTileNativeAdFactory implements GoogleMobileAdsPlugin.NativeAdFactory {

    private final Context context;

    ListTileNativeAdFactory(Context context) {
        this.context = context;
    }

    @Override
    public NativeAdView createNativeAd(
            NativeAd nativeAd, Map<String, Object> customOptions) {
        NativeAdView nativeAdView = (NativeAdView) LayoutInflater.from(context)
                .inflate(R.layout.list_tile_native_ad, null);

        TextView attributionViewSmall = nativeAdView
                .findViewById(R.id.tv_list_tile_native_ad_attribution_small);
        TextView attributionViewLarge = nativeAdView
                .findViewById(R.id.tv_list_tile_native_ad_attribution_large);

        ImageView iconView = nativeAdView.findViewById(R.id.iv_list_tile_native_ad_icon);
        NativeAd.Image icon = nativeAd.getIcon();
        if (icon != null) {
            attributionViewSmall.setVisibility(View.VISIBLE);
            attributionViewLarge.setVisibility(View.INVISIBLE);
            iconView.setImageDrawable(icon.getDrawable());
        } else {
            attributionViewSmall.setVisibility(View.INVISIBLE);
            attributionViewLarge.setVisibility(View.VISIBLE);
        }
        nativeAdView.setIconView(iconView);

        TextView headlineView = nativeAdView.findViewById(R.id.tv_list_tile_native_ad_headline);
        headlineView.setText(nativeAd.getHeadline());
        nativeAdView.setHeadlineView(headlineView);

        TextView bodyView = nativeAdView.findViewById(R.id.tv_list_tile_native_ad_body);
        bodyView.setText(nativeAd.getBody());
        bodyView.setVisibility(nativeAd.getBody() != null ? View.VISIBLE : View.INVISIBLE);
        nativeAdView.setBodyView(bodyView);

        nativeAdView.setNativeAd(nativeAd);

        return nativeAdView;
    }
}
```

MainActivity.java

- An instance of a NativeAdFactory should be registered to the GoogleMobileAdsPlugin before it can be used from the Flutter side.

- Open the MainActivity.java file, and override the configureFlutterEngine() method and the cleanUpFlutterEngine() method.
- Register the ListTileNativeAdFactory class with a unique string ID (listTile) in the configureFlutterEngine() method.

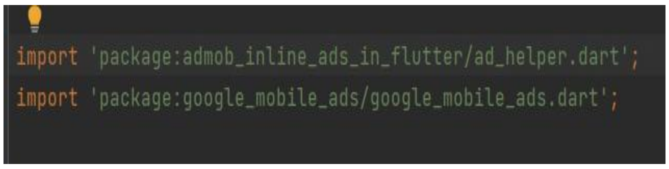


android > app > src > main > java > com > codelab > flutter > admobinlineads > MainActivity.java

```java
/*
 * Copyright 2021 Google LLC
 *
 * Licensed under the Apache License, Version 2.0 (the "License");
 * you may not use this file except in compliance with the License.
 * You may obtain a copy of the License at
 *
 *      http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 */

package com.codelab.flutter.admobinlineads;

import androidx.annotation.NonNull;
import io.flutter.embedding.android.FlutterActivity;

// COMPLETE: Import io.flutter.embedding.engine.FlutterEngine
import io.flutter.embedding.engine.FlutterEngine;

// COMPLETE: Import io.flutter.plugins.googlemobileads.GoogleMobileAdsPlugin
import io.flutter.plugins.googlemobileads.GoogleMobileAdsPlugin;

public class MainActivity extends FlutterActivity {

    @Override
    public void configureFlutterEngine(@NonNull FlutterEngine flutterEngine) {
        super.configureFlutterEngine(flutterEngine);

        // TODO: Register the ListTileNativeAdFactory
        GoogleMobileAdsPlugin.registerNativeAdFactory(flutterEngine, "listTile",
                new ListTileNativeAdFactory(getContext()));
    }

    }
```

7. Integrate the native ad with Flutter widgets

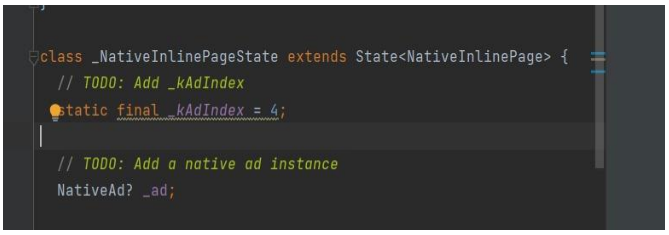
I should then open lib/native_inline_page.dart file. Then, import ad_helper.dart and google_mobile_ads.dart by adding the following lines:



```dart
import 'package:admob_inline_ads_in_flutter/ad_helper.dart';
import 'package:google_mobile_ads/google_mobile_ads.dart';
```

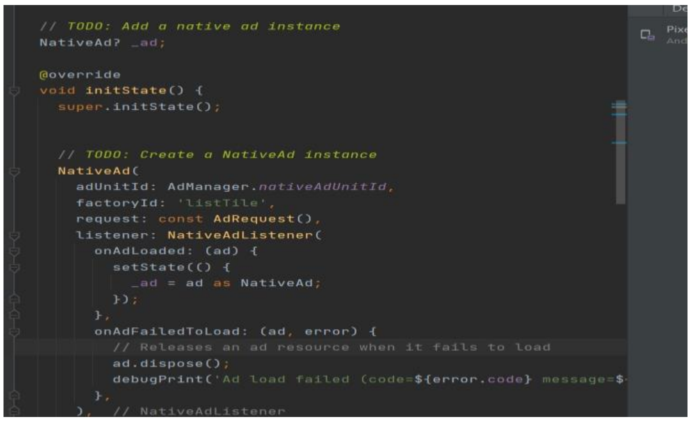
native_inline_page.dart

Note that _kAdIndex indicates the index where a banner ad will be displayed, and it's used to calculate the item index from the _getDestinationItemIndex() method.



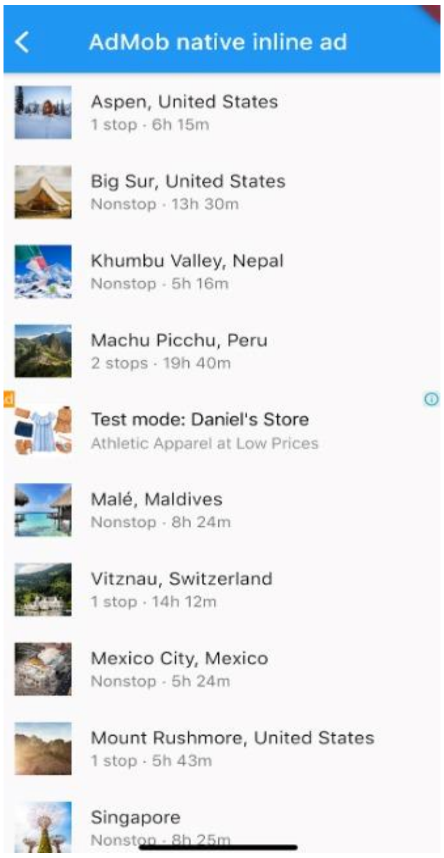
```
class _NativeInlinePageState extends State<NativeInlinePage> {
  // TODO: Add _kAdIndex
  static final _kAdIndex = 4;

  // TODO: Add a native ad instance
  NativeAd? _ad;
```

In the initState() method, create and load a NativeAd that uses ListTileNativeAdFactory to generate a native ad view.



```
// TODO: Add a native ad instance
NativeAd? _ad;

@override
void initState() {
  super.initState();


  // TODO: Create a NativeAd instance
  NativeAd(
    adUnitId: AdManager.nativeAdUnitId,
    factoryId: 'listTile',
    request: const AdRequest(),
    listener: NativeAdListener(
      onAdLoaded: (ad) {
        setState(() {
          _ad = ad as NativeAd;
        });
      },
      onAdFailedToLoad: (ad, error) {
        // Releases an ad resource when it fails to load
        ad.dispose();
        debugPrint('Ad load failed (code=${error.code} message=$
      },
    ), // NativeAdListener
```

3.5 Final Output



AdMob native inline ad

Aspen, United States
1 stop · 6h 15m

Big Sur, United States
Nonstop · 13h 30m

Khumbu Valley, Nepal
Nonstop · 5h 16m

Machu Picchu, Peru
2 stops · 19h 40m

Test mode: Daniel's Store
Athletic Apparel at Low Prices

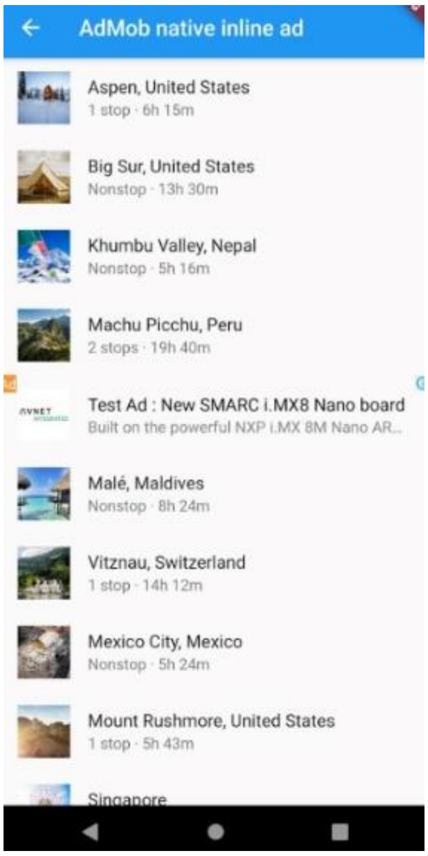
Malé, Maldives
Nonstop · 8h 24m

Vitznau, Switzerland
1 stop · 14h 12m

Mexico City, Mexico
Nonstop · 5h 24m

Mount Rushmore, United States
1 stop · 5h 43m

Singapore
Nonstop · 8h 25m



AdMob native inline ad

Aspen, United States
1 stop · 6h 15m

Big Sur, United States
Nonstop · 13h 30m

Khumbu Valley, Nepal
Nonstop · 5h 16m

Machu Picchu, Peru
2 stops · 19h 40m

Test Ad : New SMARC i.MX8 Nano board
Built on the powerful NXP i.MX 8M Nano AR...

Malé, Maldives
Nonstop · 8h 24m

Vitznau, Switzerland
1 stop · 14h 12m

Mexico City, Mexico
Nonstop · 5h 24m

Mount Rushmore, United States
1 stop · 5h 43m

Singapore

4 Task 4

Task-4

Building next generation UIs in Flutter
https://codelabs.developers.google.com/codelabs/flutter-next-gen-uis#0 **Provide a documentation with a short description and screenshots for all the steps of the lab.**

4.1 Brief Overview

This task involves creating the initial menu page for a post-apocalyptic sci-fi themed game. The page will feature a visually animated title achieved through a fragment shader that samples the text. Additionally, a difficulty menu will be implemented, incorporating color theme changes and multiple animations. An animated orb painted with a second fragment shader will also be featured. Towards the end of the codelab, a subtle particle effect will be added to introduce movement and visual interest to the page. Through this project, I gained practical experience in designing and implementing visually engaging elements essential for gaming interfaces, including the use of fragment shaders for text and orb animations, color theming, and the integration of subtle particle effects for enhanced user experience.

4.2 Steps

1. Download the starter code

In VS Code, click File > Open folder > codelabs-main > next-gen-uis > step_01 to open the starter project.

2. Run the starter app

- Here's what I see when I use Chrome as my deployment target:
- Open the lib/main.dart file and click The Play button from VSCode Start debugging. The app launches on my desktop operating system or in a Chrome browser.

3. Explore the starter app

In the starter app, I noticed the following:

The UI is ready for I to build.

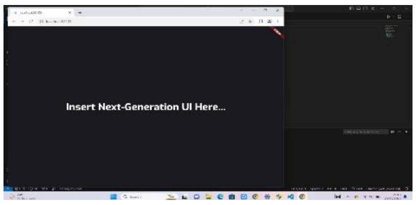
The assets directory has the art assets and two fragment shaders that I will use.

The pubspec.yaml file already lists the assets and a collection of pub packages that I will be utilizing.

The lib directory contains the obligatory main.dart file, an assets.dart file that lists the path of the art assets and fragment shaders, and a styles.dart file that lists the TextStyles and Colors I will use.

The lib directory also contains a common directory, which holds a handful of useful utilities that I will use in this codelab, and the orb_shader directory, which contains a Widget that will be used to display the orb with a vertex shader.

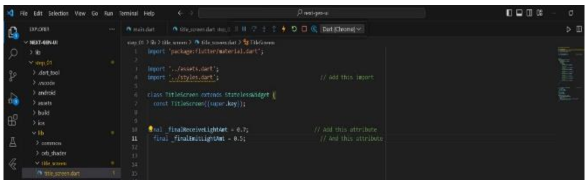
Here is what I will see once I start the app.



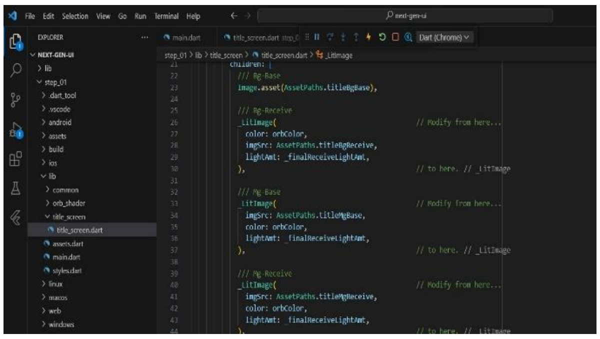
4. Paint the scene

Add assets to the scene

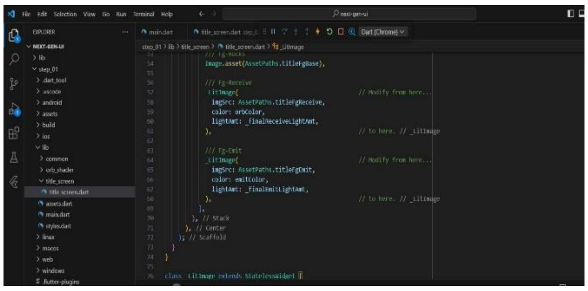
Here, I create a title_screen directory in my lib directory, and then I add a title_screen.dart file. I incorporate the following content into the file:



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

import '../assets.dart';
import '../styles.dart';                          // Add this import

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


final _finalReceiveLightAmt = 0.7;              // Add this attribute
  final _finalEmitLightAmt = 0.5;                // And this attribute
```

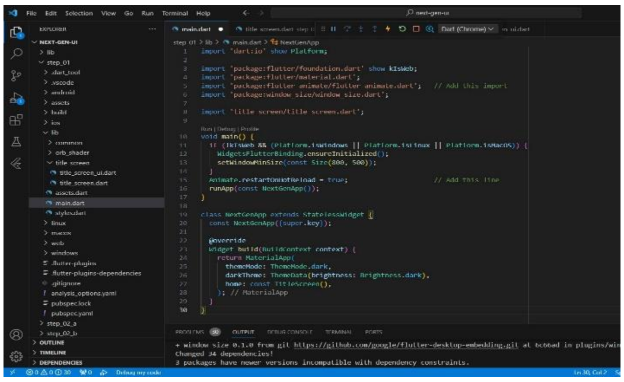


```dart
            children: [
              /// Bg-Base
              Image.asset(AssetPaths.titleBgBase),

              /// Bg-Receive
              _LitImage(                                    // Modify from here...
                color: orbColor,
                imgSrc: AssetPaths.titleBgReceive,
                lightAmt: _finalReceiveLightAmt,
              ),                                            // to here. // _LitImage

              /// Mg-Base
              _LitImage(                                    // Modify from here...
                imgSrc: AssetPaths.titleMgBase,
                color: orbColor,
                lightAmt: _finalReceiveLightAmt,
              ),                                            // to here. // _LitImage

              /// Mg-Receive
              _LitImage(                                    // Modify from here...
                imgSrc: AssetPaths.titleMgReceive,
                color: orbColor,
                lightAmt: _finalReceiveLightAmt,
              ),                                            // to here. // _LitImage
```



```dart
              /// Fg-Rocks
              Image.asset(AssetPaths.titleFgBase),

              /// Fg-Receive
              _LitImage(                                    // Modify from here...
                imgSrc: AssetPaths.titleFgReceive,
                color: orbColor,
                lightAmt: _finalReceiveLightAmt,
              ),                                            // to here. // _LitImage

              /// Fg-Emit
              _LitImage(                                    // Modify from here...
                imgSrc: AssetPaths.titleFgEmit,
                color: emitColor,
                lightAmt: _finalEmitLightAmt,
              ),                                            // to here. // _LitImage
            ],
          ), // Stack
        ), // Center
      ); // Scaffold
    }
  }

class _LitImage extends StatelessWidget {
```

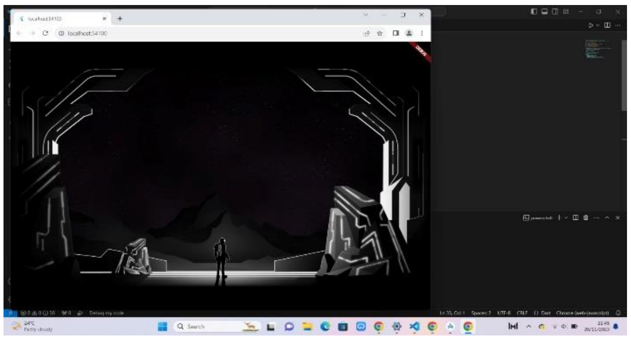
In the main.dart file, add the following content:



```
import 'dart:io' show Platform;

import 'package:flutter/foundation.dart' show kIsWeb;
import 'package:flutter/material.dart';
import 'package:flutter_animate/flutter_animate.dart';   // Add this import
import 'package:window_size/window_size.dart';

import 'title_screen/title_screen.dart';

void main() {
  if (!kIsWeb && (Platform.isWindows || Platform.isLinux || Platform.isMacOS)) {
    WidgetsFlutterBinding.ensureInitialized();
    setWindowMinSize(const Size(800, 500));
  }
  Animate.restartOnHotReload = true;                   // Add this line
  runApp(const NextGenApp());
}

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

  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      themeMode: ThemeMode.dark,
      darkTheme: ThemeData(brightness: Brightness.dark),
      home: const TitleScreen(),
    ); // MaterialApp
  }
}
```

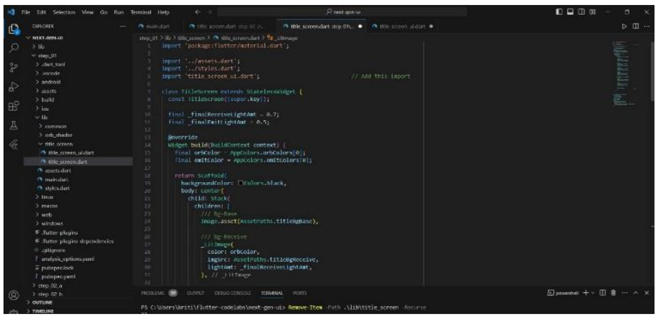
```
+ window_size 0.1.0 from git https://github.com/google/flutter-desktop-embedding.git at 6c66ad in plugins/win
Changed 34 dependencies!
3 packages have newer versions incompatible with dependency constraints.
```

This replaces the app's UI with the monochrome scene that the art assets create. Next, I color each layer.



Add an image coloring utility

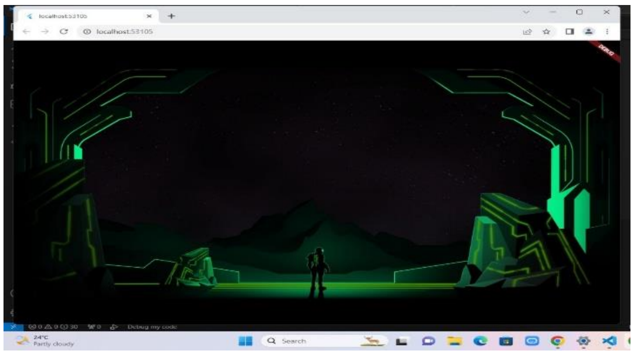
Paint in color by modifying the title_screen.dart file, as follows:



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

import '../assets.dart';
import '../styles.dart';
import 'title_screen_ui.dart';                          // Add this import

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

  final _finalReceiveLightAmt = 0.7;
  final _finalEmitLightAmt = 0.5;

  @override
  Widget build(BuildContext context) {
    final orbColor = AppColors.orbColors[0];
    final emitColor = AppColors.emitColors[0];

    return Scaffold(
      backgroundColor: Colors.black,
      body: Center(
        child: Stack(
          children: [
            /// Bg-Base
            Image.asset(AssetPaths.titleBgBase),

            /// Bg-Receive
            _LitImage(
              color: orbColor,
              imgSrc: AssetPaths.titleBgReceive,
              lightAmt: _finalReceiveLightAmt,
            ), // _LitImage
```

```
PS C:\Users\kriti\flutter-codelabs\next-gen-ui> Remove-Item -Path .\lib\title_screen -Recurse
```

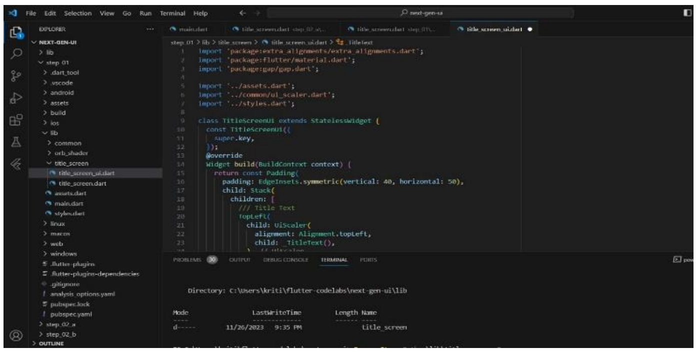
Here is the app again, this time with the art assets tinted green.



5. Add a UI

- Add a title

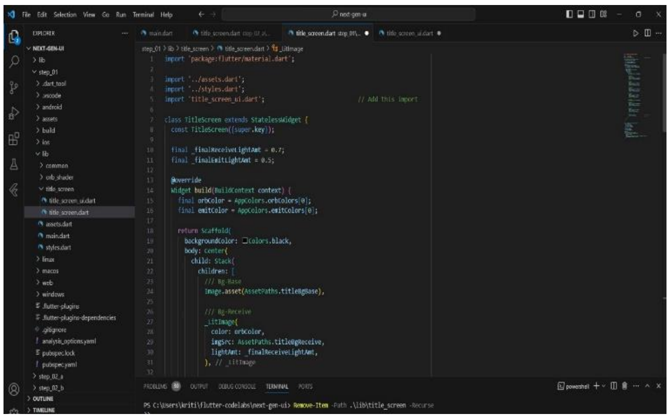
Create a title_screen_ui.dart file inside of the lib/title_screen directory and add the following content to the file:



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

import '../assets.dart';
import '../common/ui_scaler.dart';
import '../styles.dart';

class TitleScreenUi extends StatelessWidget {
  const TitleScreenUi({
    super.key,
  });
  @override
  Widget build(BuildContext context) {
    return const Padding(
      padding: EdgeInsets.symmetric(vertical: 40, horizontal: 50),
      child: Stack(
        children: [
          /// Title Text
          TopLeft(
            child: UiScaler(
              alignment: Alignment.topLeft,
              child: _TitleText(),
```

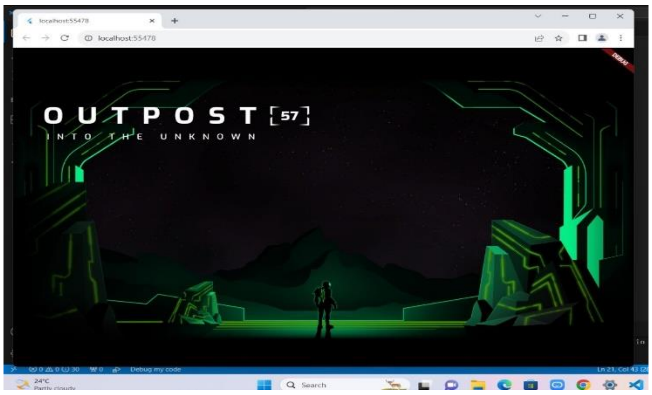
```
    Directory: C:\Users\kriti\flutter-codelabs\next-gen-ui\lib

Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
d-----        11/26/2023   9:35 PM                title_screen
```

- Update the lib/title_screen/title_screen.dart file, as follows:

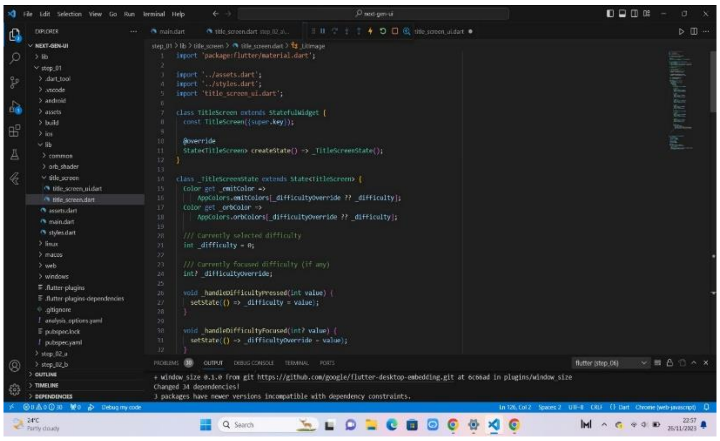


```
import 'package:flutter/material.dart';

import '../assets.dart';
import '../styles.dart';
import 'title_screen_ui.dart';                    // Add this import

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

  final _finalReceiveLightAmt = 0.7;
  final _finalEmitLightAmt = 0.5;

  @override
  Widget build(BuildContext context) {
    final orbColor = AppColors.orbColors[0];
    final emitColor = AppColors.emitColors[0];

    return Scaffold(
      backgroundColor: Colors.black,
      body: Center(
        child: Stack(
          children: [
            /// Bg-Base
            Image.asset(AssetPaths.titleBgBase),

            /// Bg-Receive
            _LitImage(
              color: orbColor,
              imgSrc: AssetPaths.titleBgReceive,
              lightAmt: _finalReceiveLightAmt,
            ), // _LitImage
```

```
PS C:\Users\kriti\flutter-codelabs\next-gen-ui> Remove-Item -Path .\lib\title_screen -Recurse
```

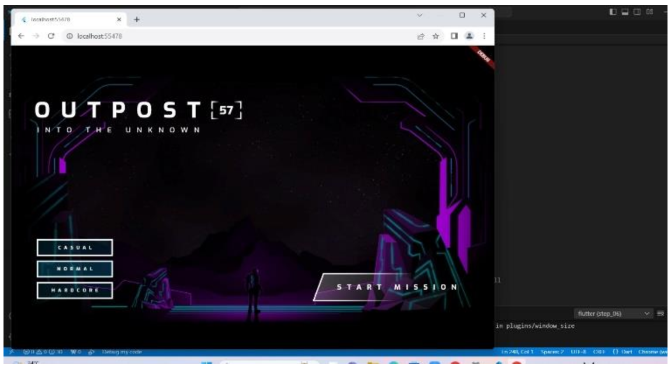


Now the following code should be added

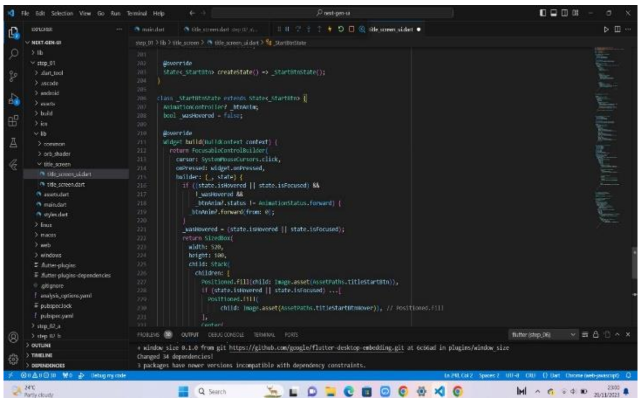


```
import 'package:flutter/material.dart';

import '../assets.dart';
import '../styles.dart';
import 'title_screen_ui.dart';

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

  @override
  State<TitleScreen> createState() => _TitleScreenState();
}

class _TitleScreenState extends State<TitleScreen> {
  Color get _emitColor =>
      AppColors.emitColors[_difficultyOverride ?? _difficulty];
  Color get _orbColor =>
      AppColors.orbColors[_difficultyOverride ?? _difficulty];

  /// Currently selected difficulty
  int _difficulty = 0;

  /// Currently focused difficulty (if any)
  int? _difficultyOverride;

  void _handleDifficultyPressed(int value) {
    setState(() => _difficulty = value);
  }

  void _handleDifficultyFocused(int? value) {
    setState(() => _difficultyOverride = value);
  }
```

```
+ window_size 0.1.0 from git https://github.com/google/flutter-desktop-embedding.git at 6c66ad in plugins/window_size
Changed 34 dependencies!
3 packages have newer versions incompatible with dependency constraints.
```

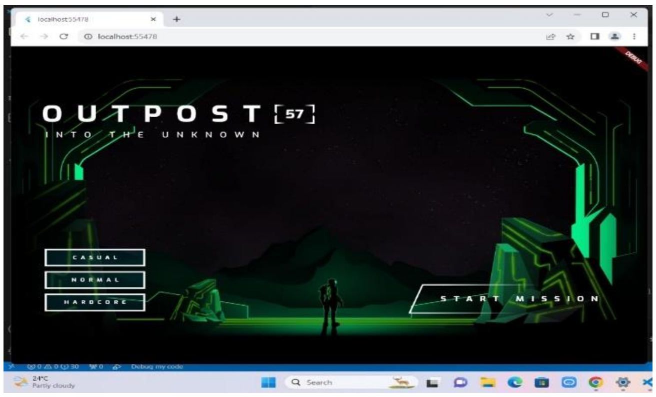


Add the following widget to implement the start button:



```
  @override
  State<_StartBtn> createState() => _StartBtnState();
}

class _StartBtnState extends State<_StartBtn> {
  AnimationController? _btnAnim;
  bool _wasHovered = false;

  @override
  Widget build(BuildContext context) {
    return FocusableControlBuilder(
      cursor: SystemMouseCursors.click,
      onPressed: widget.onPressed,
      builder: (_, state) {
        if ((state.isHovered || state.isFocused) &&
            !_wasHovered &&
            _btnAnim?.status != AnimationStatus.forward) {
          _btnAnim?.forward(from: 0);
        }
        _wasHovered = (state.isHovered || state.isFocused);
        return SizedBox(
          width: 520,
          height: 100,
          child: Stack(
            children: [
              Positioned.fill(child: Image.asset(AssetPaths.titleStartBtn)),
              if (state.isHovered || state.isFocused) ...[
                Positioned.fill(
                    child: Image.asset(AssetPaths.titleStartBtnHover)), // Positioned.fill
              ],
              Center(
```

```
+ window_size 0.1.0 from git https://github.com/google/flutter-desktop-embedding.git at 6c66ad in plugins/window_size
Changed 34 dependencies!
3 packages have newer versions incompatible with dependency constraints.
```



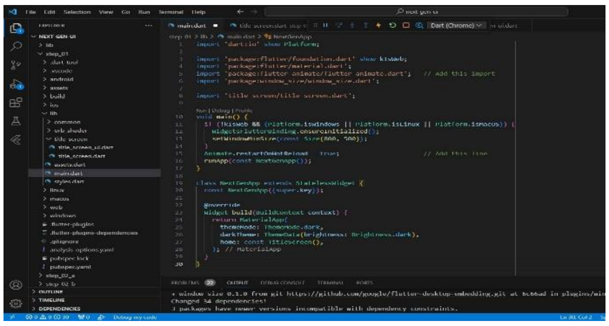
6. Add animation

In this step I animate the user interface and the color transitions for the art assets.

- Fade in the title

  In this step, I use multiple approaches to animate a Flutter app. One of the approaches is to use flutter_animate. Animations powered by this package can automatically replay whenever I hot reload my app in order to speed up development iterations.

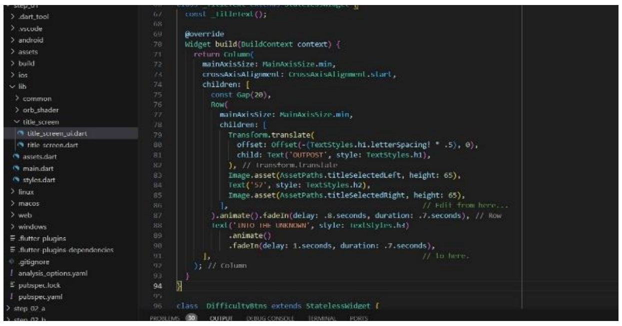
Modify the code in lib/main.dart, as follows:



```
import 'dart:io' show Platform;

import 'package:flutter/foundation.dart' show kIsWeb;
import 'package:flutter/material.dart';
import 'package:flutter_animate/flutter_animate.dart';   // add this import
import 'package:window_size/window_size.dart';

import 'title_screen/title_screen.dart';

void main() {
  if (!kIsWeb && (Platform.isWindows || Platform.isLinux || Platform.isMacOS)) {
    WidgetsFlutterBinding.ensureInitialized();
    setWindowMinSize(const Size(800, 500));
  }
  Animate.restartOnHotReload = true;                     // Add this line
  runApp(const NextGenApp());
}

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

  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      themeMode: ThemeMode.dark,
      darkTheme: ThemeData(brightness: Brightness.dark),
      home: const TitleScreen(),
    ); // MaterialApp
  }
}
```

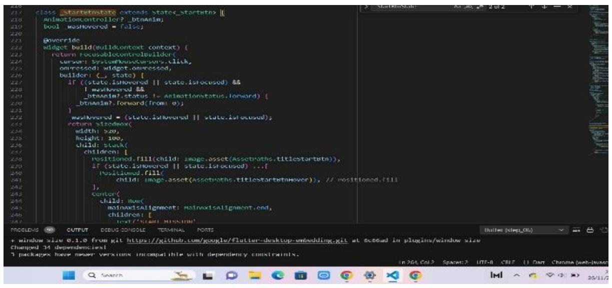
```
+ window_size 0.1.0 from git https://github.com/google/flutter-desktop-embedding.git at 6c66ad in plugins/window_size
Changed 34 dependencies!
3 packages have newer versions incompatible with dependency constraints.
```

- To take advantage of the flutter_animate package, I must import it. Add the import in lib/title_screen/title_screen_ui.dart, as follows:



```dart
  const _TitleText();

  @override
  Widget build(BuildContext context) {
    return Column(
      mainAxisSize: MainAxisSize.min,
      crossAxisAlignment: CrossAxisAlignment.start,
      children: [
        const Gap(20),
        Row(
          mainAxisSize: MainAxisSize.min,
          children: [
            Transform.translate(
              offset: Offset(-(TextStyles.h1.letterSpacing! * .5), 0),
              child: Text('OUTPOST', style: TextStyles.h1),
            ), // Transform.translate
            Image.asset(AssetPaths.titleSelectedLeft, height: 65),
            Text('57', style: TextStyles.h2),
            Image.asset(AssetPaths.titleSelectedRight, height: 65),
          ],                                                   // Edit from here...
        ).animate().fadeIn(delay: .8.seconds, duration: .7.seconds), // Row
        Text('INTO THE UNKNOWN', style: TextStyles.h3)
            .animate()
            .fadeIn(delay: 1.seconds, duration: .7.seconds),
      ],                                                       // to here.
    ); // Column
  }
}

class _DifficultyBtns extends StatelessWidget {
```

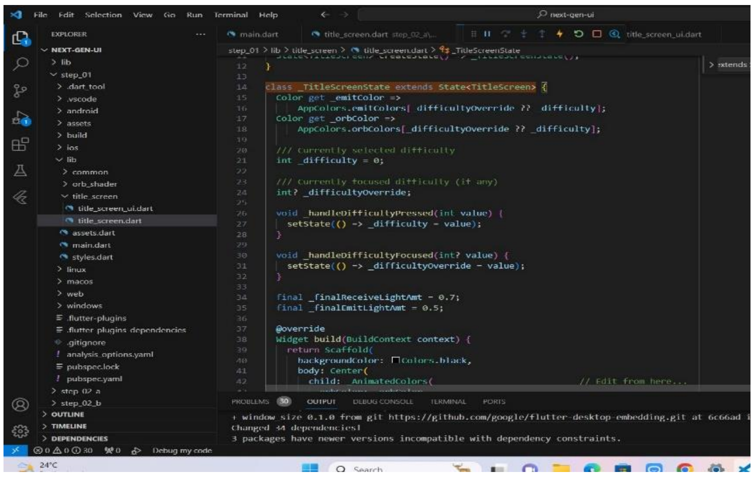
Fade in the start button



```dart
class _StartBtnState extends State<_StartBtn> {
  AnimationController? _btnAnim;
  bool _wasHovered = false;

  @override
  Widget build(BuildContext context) {
    return FocusableControlBuilder(
      cursor: SystemMouseCursors.click,
      onPressed: widget.onPressed,
      builder: (_, state) {
        if ((state.isHovered || state.isFocused) &&
            !_wasHovered &&
            _btnAnim?.status != AnimationStatus.forward) {
          _btnAnim?.forward(from: 0);
        }
        _wasHovered = (state.isHovered || state.isFocused);
        return SizedBox(
          width: 520,
          height: 100,
          child: Stack(
            children: [
              Positioned.fill(child: Image.asset(AssetPaths.titleStartBtn)),
              if (state.isHovered || state.isFocused) ...[
                Positioned.fill(
                    child: Image.asset(AssetPaths.titleStartBtnHover)), // Positioned.fill
              ],
              Center(
                child: Row(
                  mainAxisAlignment: MainAxisAlignment.end,
                  children: [
                    Text('START MISSION'
```

```
+ window_size 0.1.0 from git https://github.com/google/flutter-desktop-embedding.git at 6c66ad in plugins/window_size
Changed 34 dependencies!
3 packages have newer versions incompatible with dependency constraints.
```

Use the widget I just created to animate the colors of the lit images by updating the build method in _TitleScreenState, as follows:



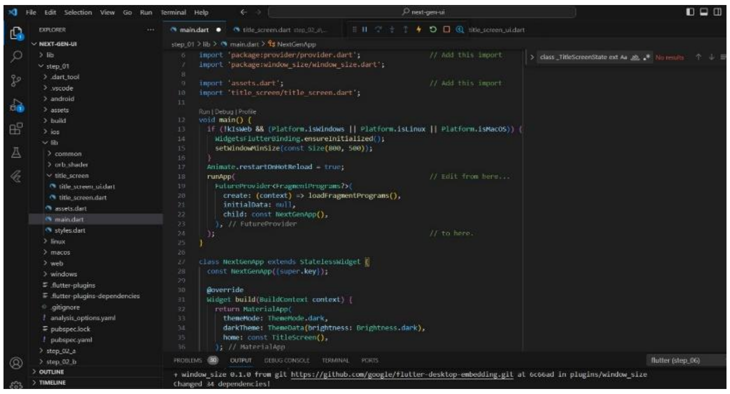
```dart
}

class _TitleScreenState extends State<TitleScreen> {
  Color get _emitColor =>
      AppColors.emitColors[_difficultyOverride ?? _difficulty];
  Color get _orbColor =>
      AppColors.orbColors[_difficultyOverride ?? _difficulty];

  /// Currently selected difficulty
  int _difficulty = 0;

  /// Currently focused difficulty (if any)
  int? _difficultyOverride;

  void _handleDifficultyPressed(int value) {
    setState(() => _difficulty = value);
  }

  void _handleDifficultyFocused(int? value) {
    setState(() => _difficultyOverride = value);
  }

  final _finalReceiveLightAmt = 0.7;
  final _finalEmitLightAmt = 0.5;

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      backgroundColor: Colors.black,
      body: Center(
        child: _AnimatedColors(                        // Edit from here...
```

```
+ window_size 0.1.0 from git https://github.com/google/flutter-desktop-embedding.git at 6c66ad i
Changed 34 dependencies!
3 packages have newer versions incompatible with dependency constraints.
```

7. Add fragment shaders

In this step I add fragment shaders to the app. First, I use a shader to modify the title to give it a more dystopian feel. Then, I add a second shader to create an orb that serves as the central focal point of the page.

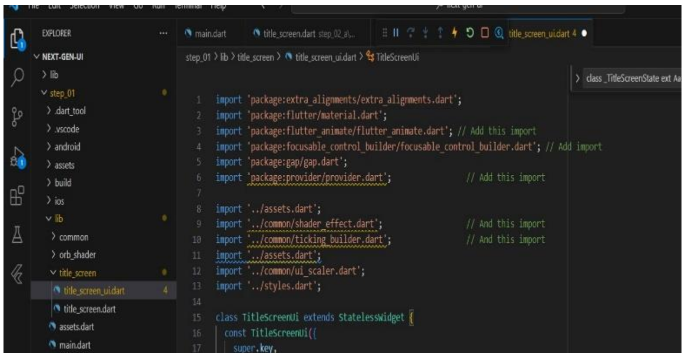
- Modify the code in lib/main.dart, as follows:



```dart
import 'package:provider/provider.dart';                // Add this import
import 'package:window_size/window_size.dart';

import 'assets.dart';                                   // Add this import
import 'title_screen/title_screen.dart';

void main() {
  if (!kIsWeb && (Platform.isWindows || Platform.isLinux || Platform.isMacOS)) {
    WidgetsFlutterBinding.ensureInitialized();
    setWindowMinSize(const Size(800, 500));
  }
  Animate.restartOnHotReload = true;
  runApp(                                               // Edit from here...
    FutureProvider<FragmentPrograms?>(
      create: (context) => loadFragmentPrograms(),
      initialData: null,
      child: const NextGenApp(),
    ), // FutureProvider
  );                                                    // to here.
}

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

  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      themeMode: ThemeMode.dark,
      darkTheme: ThemeData(brightness: Brightness.dark),
      home: const TitleScreen(),
    ); // MaterialApp
```

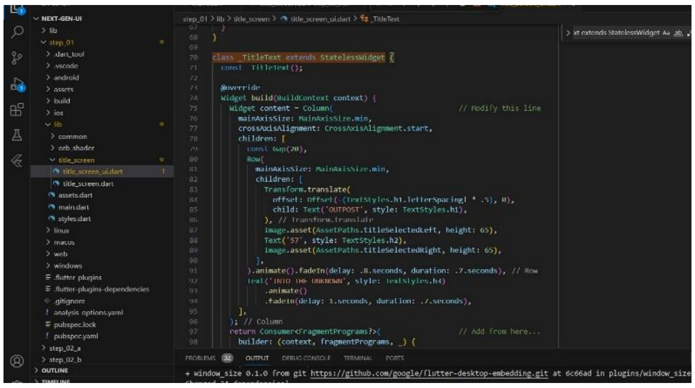
```
+ window_size 0.1.0 from git https://github.com/google/flutter-desktop-embedding.git at 6c66ad in plugins/window_size
Changed 34 dependencies!
```

To take advantage of the provider package, and the shader utilities included in step_01, I need to import them. Add new imports in lib/title_screen/title_screen_ui.dart, as follows:



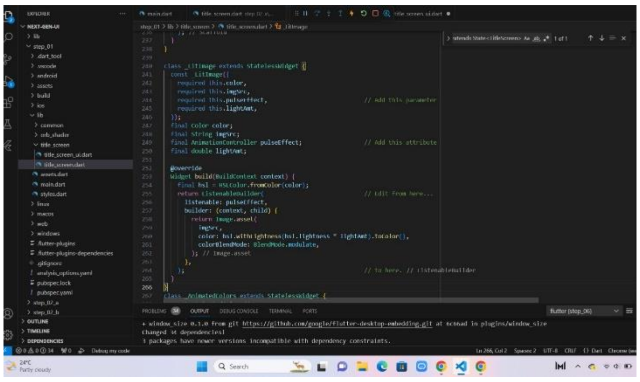
```dart
import 'package:extra_alignments/extra_alignments.dart';
import 'package:flutter/material.dart';
import 'package:flutter_animate/flutter_animate.dart'; // Add this import
import 'package:focusable_control_builder/focusable_control_builder.dart'; // Add import
import 'package:gap/gap.dart';
import 'package:provider/provider.dart';                 // Add this import

import '../assets.dart';
import '../common/shader_effect.dart';                   // And this import
import '../common/ticking_builder.dart';                 // And this import
import '../assets.dart';
import '../common/ui_scaler.dart';
import '../styles.dart';

class TitleScreenUi extends StatelessWidget {
  const TitleScreenUi({
    super.key,
```

- Distort the title with the shader by editing the _TitleText widget, as follows:



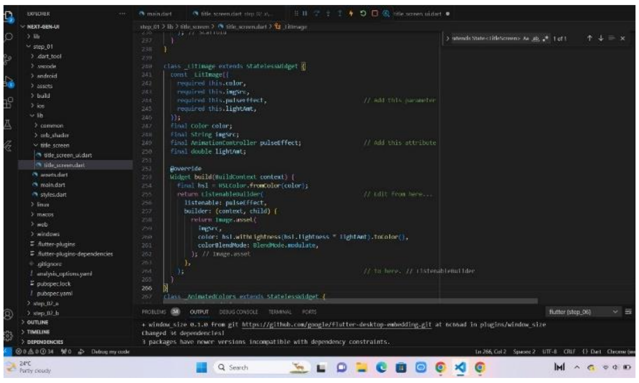
```dart
}

class _TitleText extends StatelessWidget {
  const _TitleText();

  @override
  Widget build(BuildContext context) {
    Widget content = Column(                             // Modify this line
      mainAxisSize: MainAxisSize.min,
      crossAxisAlignment: CrossAxisAlignment.start,
      children: [
        const Gap(20),
        Row(
          mainAxisSize: MainAxisSize.min,
          children: [
            Transform.translate(
              offset: Offset(-(TextStyles.h1.letterSpacing! * .5), 0),
              child: Text('OUTPOST', style: TextStyles.h1),
            ), // Transform.translate
            Image.asset(AssetPaths.titleSelectedLeft, height: 65),
            Text('57', style: TextStyles.h2),
            Image.asset(AssetPaths.titleSelectedRight, height: 65),
          ],
        ).animate().fadeIn(delay: .8.seconds, duration: .7.seconds), // Row
        Text('INTO THE UNKNOWN', style: TextStyles.h3)
            .animate()
            .fadeIn(delay: 1.seconds, duration: .7.seconds),
      ],
    ); // Column
    return Consumer<FragmentPrograms?>(                  // Add from here...
      builder: (context, fragmentPrograms, _) {
```

```
+ window_size 0.1.0 from git https://github.com/google/flutter-desktop-embedding.git at 6c66ad in plugins/window_size
```



```
    }); // Scaffold
  }
}

class _LitImage extends StatelessWidget {
  const _LitImage({
    required this.color,
    required this.imgSrc,
    required this.pulseEffect,                 // Add this parameter
    required this.lightAmt,
  });
  final Color color;
  final String imgSrc;
  final AnimationController pulseEffect;      // Add this attribute
  final double lightAmt;

  @override
  Widget build(BuildContext context) {
    final hsl = HSLColor.fromColor(color);
    return ListenableBuilder(                  // Edit from here...
      listenable: pulseEffect,
      builder: (context, child) {
        return Image.asset(
          imgSrc,
          color: hsl.withLightness(hsl.lightness * lightAmt).toColor(),
          colorBlendMode: BlendMode.modulate,
        ); // Image.asset
      },
    );                                         // to here. // ListenableBuilder
  }
}
class _AnimatedColors extends StatelessWidget {
```

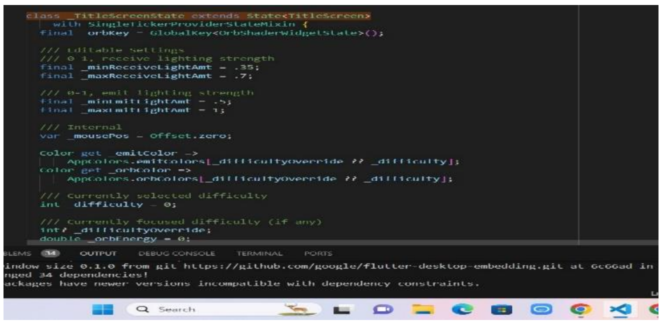
```
+ window_size 0.1.0 from git https://github.com/google/flutter-desktop-embedding.git at 6c66ad in plugins/window_size
Changed 34 dependencies!
3 packages have newer versions incompatible with dependency constraints.
```



```
    }); // Scaffold
  }
}

class _LitImage extends StatelessWidget {
  const _LitImage({
    required this.color,
    required this.imgSrc,
    required this.pulseEffect,                 // Add this parameter
    required this.lightAmt,
  });
  final Color color;
  final String imgSrc;
  final AnimationController pulseEffect;      // Add this attribute
  final double lightAmt;

  @override
  Widget build(BuildContext context) {
    final hsl = HSLColor.fromColor(color);
    return ListenableBuilder(                  // Edit from here...
      listenable: pulseEffect,
      builder: (context, child) {
        return Image.asset(
          imgSrc,
          color: hsl.withLightness(hsl.lightness * lightAmt).toColor(),
          colorBlendMode: BlendMode.modulate,
        ); // Image.asset
      },
    );                                         // to here. // ListenableBuilder
  }
}
class _AnimatedColors extends StatelessWidget {
```

```
+ window_size 0.1.0 from git https://github.com/google/flutter-desktop-embedding.git at 6c66ad in plugins/window_size
Changed 34 dependencies!
3 packages have newer versions incompatible with dependency constraints.
```

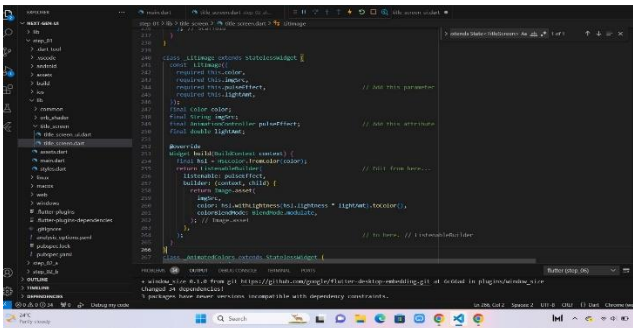
- Modify _TitleScreenState to match the following. Almost every part of the class is modified in some way.



```
class _TitleScreenState extends State<TitleScreen>
    with SingleTickerProviderStateMixin {
  final _orbKey = GlobalKey<OrbShaderWidgetState>();

  /// Editable Settings
  /// 0-1, receive lighting strength
  final _minReceiveLightAmt = .35;
  final _maxReceiveLightAmt = .7;

  /// 0-1, emit lighting strength
  final _minEmitLightAmt = .5;
  final _maxEmitLightAmt = 1;

  /// Internal
  var _mousePos = Offset.zero;

  Color get _emitColor =>
      AppColors.emitColors[_difficultyOverride ?? _difficulty];
  Color get _orbColor =>
      AppColors.orbColors[_difficultyOverride ?? _difficulty];

  /// Currently selected difficulty
  int _difficulty = 0;

  /// Currently focused difficulty (if any)
  int? _difficultyOverride;
  double _orbEnergy = 0;
```

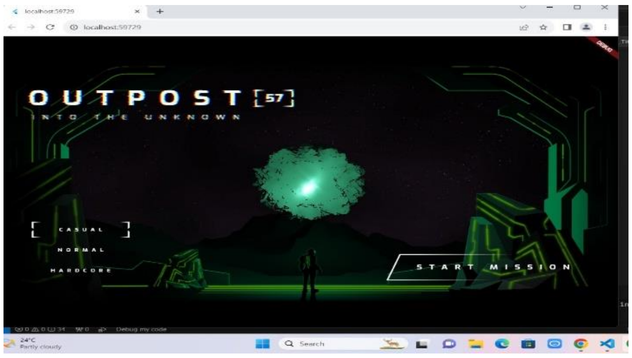
```
indow_size 0.1.0 from git https://github.com/google/flutter-desktop-embedding.git at 6c66ad in
nged 34 dependencies!
ackages have newer versions incompatible with dependency constraints.
```

- Modify _LitImage, as follows:



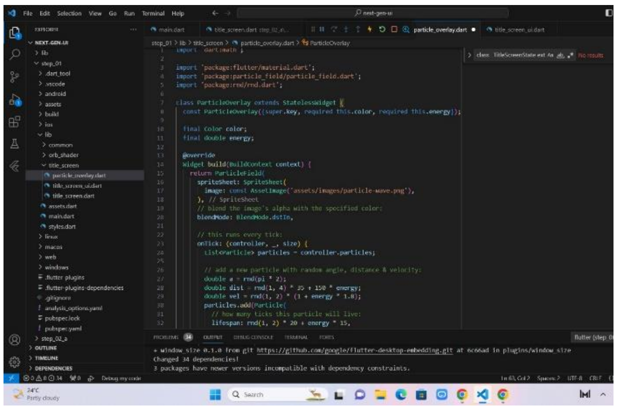
```
    }); // Scaffold
  }
}

class _LitImage extends StatelessWidget {
  const _LitImage({
    required this.color,
    required this.imgSrc,
    required this.pulseEffect,                 // Add this parameter
    required this.lightAmt,
  });
  final Color color;
  final String imgSrc;
  final AnimationController pulseEffect;      // Add this attribute
  final double lightAmt;

  @override
  Widget build(BuildContext context) {
    final hsl = HSLColor.fromColor(color);
    return ListenableBuilder(                  // Edit from here...
      listenable: pulseEffect,
      builder: (context, child) {
        return Image.asset(
          imgSrc,
          color: hsl.withLightness(hsl.lightness * lightAmt).toColor(),
          colorBlendMode: BlendMode.modulate,
        ); // Image.asset
      },
    );                                         // to here. // ListenableBuilder
  }
}
class _AnimatedColors extends StatelessWidget {
```

```
+ window_size 0.1.0 from git https://github.com/google/flutter-desktop-embedding.git at 6c66ad in plugins/window_size
Changed 34 dependencies!
3 packages have newer versions incompatible with dependency constraints.
```



8. Add particle animations

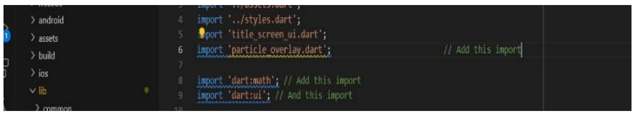
- In this step, I add particle animations to create a subtle pulsing movement to the app.



```
import 'dart:math';

import 'package:flutter/material.dart';
import 'package:particle_field/particle_field.dart';
import 'package:rnd/rnd.dart';

class ParticleOverlay extends StatelessWidget {
  const ParticleOverlay({super.key, required this.color, required this.energy});

  final Color color;
  final double energy;

  @override
  Widget build(BuildContext context) {
    return ParticleField(
      spriteSheet: SpriteSheet(
        image: const AssetImage('assets/images/particle-wave.png'),
      ), // SpriteSheet
      // blend the image's alpha with the specified color:
      blendMode: BlendMode.dstIn,

      // this runs every tick:
      onTick: (controller, _, size) {
        List<Particle> particles = controller.particles;

        // add a new particle with random angle, distance & velocity:
        double a = rnd(pi * 2);
        double dist = rnd(1, 4) * 35 + 150 * energy;
        double vel = rnd(1, 2) * (1 + energy * 1.8);
        particles.add(Particle(
          // how many ticks this particle will live:
          lifespan: rnd(1, 2) * 20 + energy * 15,
```

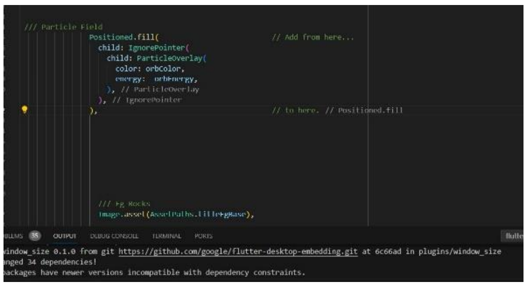
```
+ window_size 0.1.0 from git https://github.com/google/flutter-desktop-embedding.git at 6c66ad in plugins/window_size
Changed 34 dependencies!
3 packages have newer versions incompatible with dependency constraints.
```

- Modify the imports for lib/title_screen/title_screen.dart, as follows:



```
import '../styles.dart';
import 'title_screen_ui.dart';
import 'particle_overlay.dart';                    // Add this import

import 'dart:math'; // Add this import
import 'dart:ui'; // Add this import
```

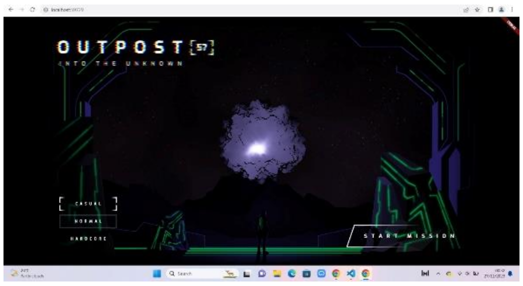
- Add the ParticleOverlay to the UI by modifying the build method of _TitleScreenState, as follows:



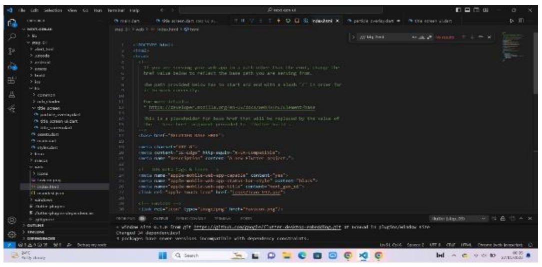
```
/// Particle Field
              Positioned.fill(                       // Add from here...
                child: IgnorePointer(
                  child: ParticleOverlay(
                    color: orbColor,
                    energy:  orbEnergy,
                  ), // ParticleOverlay
                ), // IgnorePointer
              ),                                     // to here. // Positioned.fill




                /// Fg Rocks
                Image.asset(AssetPaths.titleFgBase),
```

```
window_size 0.1.0 from git https://github.com/google/flutter-desktop-embedding.git at 6c66ad in plugins/window_size
anged 34 dependencies!
ackages have newer versions incompatible with dependency constraints.
```

The final result includes animations, fragment shaders, and particle effects—on multiple platforms!



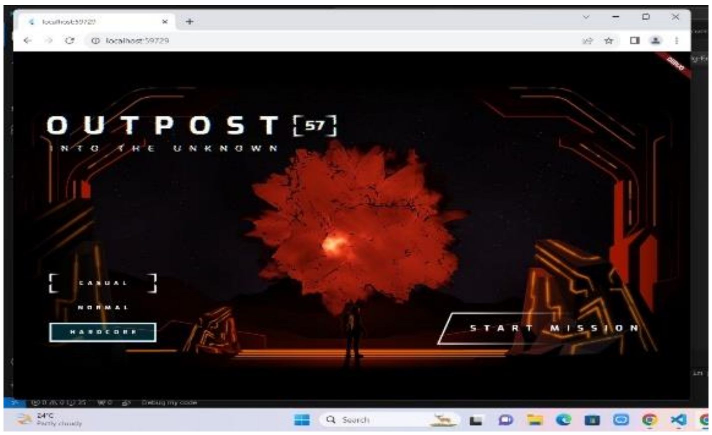
- Add particles everywhere—even the web
- Modify web/index.html, as follows:



```
<!DOCTYPE html>
<html>
<head>
  <!--
    If you are serving your web app in a path other than the root, change the
    href value below to reflect the base path you are serving from.

    The path provided below has to start and end with a slash "/" in order for
    it to work correctly.

    For more details:
    * https://developer.mozilla.org/en-US/docs/Web/HTML/Element/base

    This is a placeholder for base href that will be replaced by the value of
    the `--base-href` argument provided to `flutter build`.
  -->
  <base href="$FLUTTER_BASE_HREF">

  <meta charset="UTF-8">
  <meta content="IE=Edge" http-equiv="X-UA-Compatible">
  <meta name="description" content="A new Flutter project.">

  <!-- iOS meta tags & icons -->
  <meta name="apple-mobile-web-app-capable" content="yes">
  <meta name="apple-mobile-web-app-status-bar-style" content="black">
  <meta name="apple-mobile-web-app-title" content="next_gen_ui">
  <link rel="apple-touch-icon" href="icons/Icon-192.png">

  <!-- Favicon -->
  <link rel="icon" type="image/png" href="favicon.png"/>
```

```
+ window_size 0.1.0 from git https://github.com/google/flutter-desktop-embedding.git at [illegible] in plugins/window_size
Changed 34 dependencies!
4 packages have newer versions incompatible with dependency constraints.
```

4.3 Final Output



5 Task 5

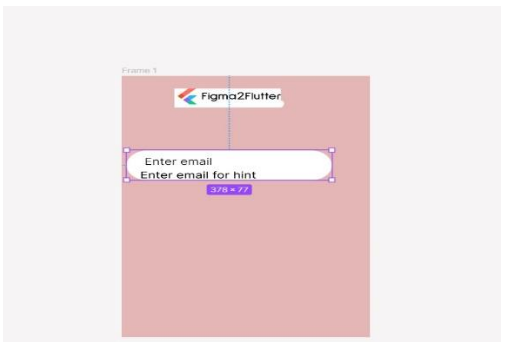
Task-5

Figma2Flutter – Part 6 to 10
https://f2f.bostonux.com/video **Provide a documentation with a short description and screenshots for all the steps of the lab.**

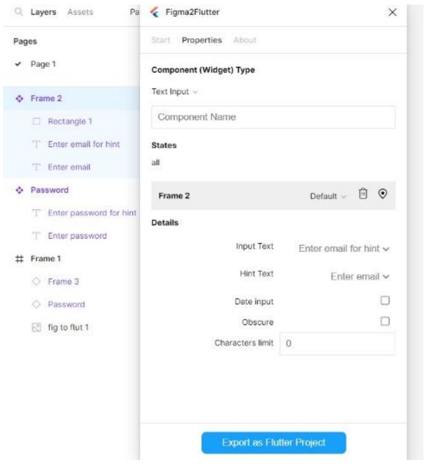
5.1 Brief Overview

5.2 Text Input Part 1

- Firstly,I draw a new Figma frame.This will be mymain screen. Set the background color and add some images. For the purpose of this tutorial, we'll create two text nodes. The first is for typing and the second is for the hint text.

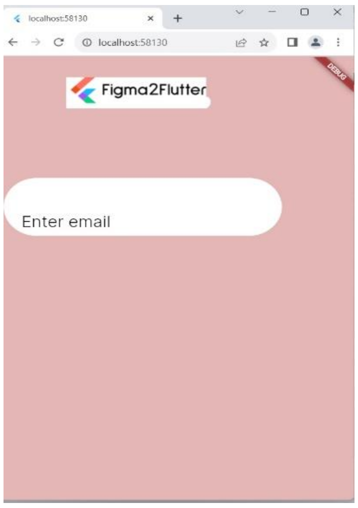


- Now, I select my newly created component. And from the Plugin menu, I press the Combine Selected Components at State button. From the drop-down menu, select Text Input. Below, in the Plugin menu, select the Input Text field for the first text node I created. Below, in the Plugin menu, I select the Hint Text field for the second text node, the one with the grayish color.



Component (Widget) Type
Text Input
Component Name
States
all
Frame 2 Default
Details
Input Text: Enter email for hint
Hint Text: Enter email
Date input ☐
Obscure ☐
Characters limit: 0

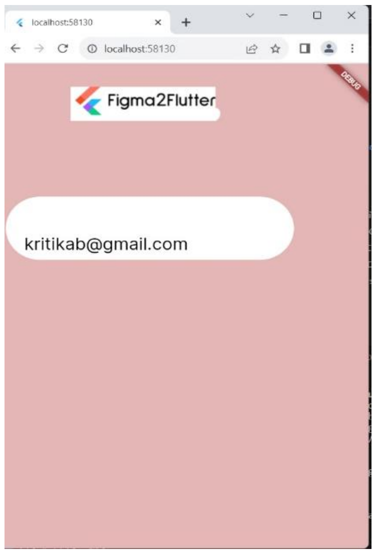
- Now, create an instance of my component and put it inside the screen one frame.As I can observe,the hint is hidden!



Enter email

Now when exporting my project into flutter, I should run flutter pub get for the dependencies.

5.2.1 Final Output

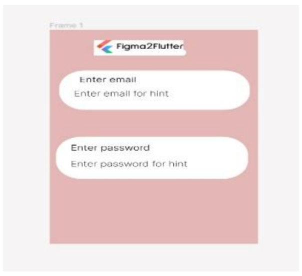


kritikab@gmail.com

5.3 Text Input Part 2

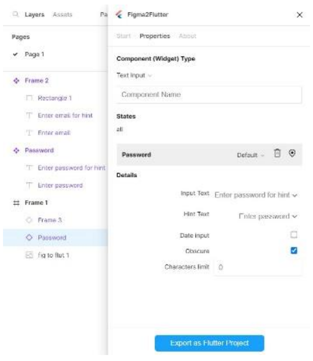
- I am using the existing frame.I'll then set the background color and add some images.

- It is very important that this component has at least one text node. For the purpose of this tutorial, we'll create two text nodes. The first is for typing and the second is for the hint text. The hint text node should have a grayish color for the text.



- Select Ir newly created component. And from the Plugin menu, press the Combine Selected Components at State button. From the drop-down menu, select Text Input. Below, in the Plugin menu, select the Input Text field for the first text node I created. Below, in the Plugin menu, select the Hint Text field for the second text node, the one with the grayish color.

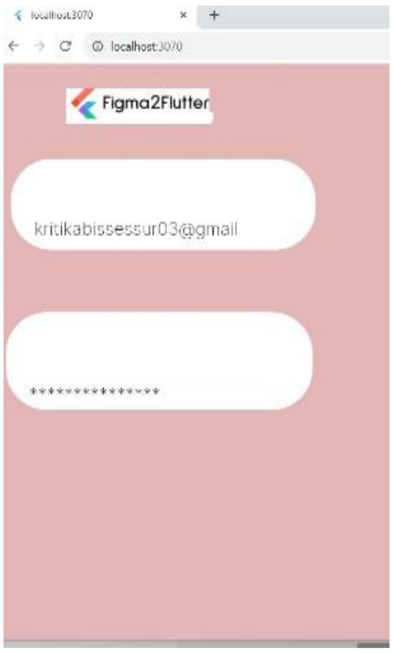
- Also, I clicked on obscure text since password is not supposed to be visible, it will be displayed in asterisk



Component (Widget) Type
Text Input
Component Name
States
all
Password Default
Details
Input Text Enter password for hint
Hint Text Enter password
Date input ☐
Obscure ☑
Characters limit 0

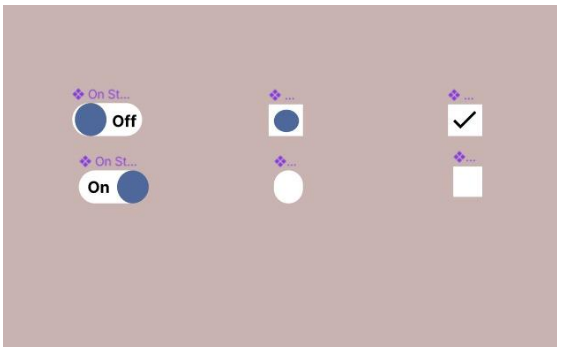
- I then click on the export button and run flutter pub to get the dependencies.

5.3.1 Final Output

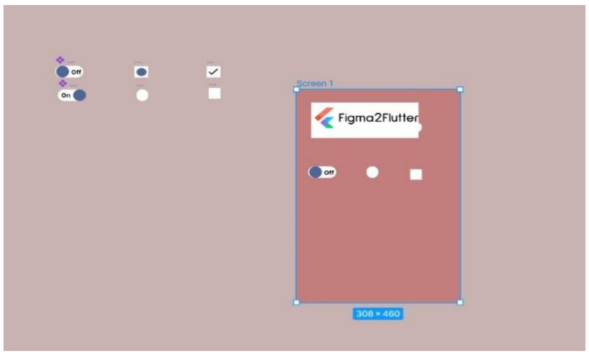


5.4 ToggleRadio CheckBox

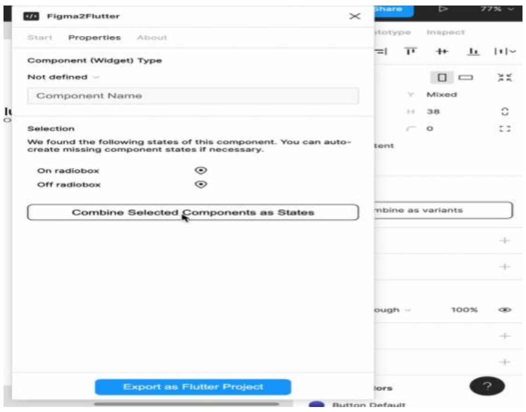
I am using the same figma frame,that is the existing one.And as from this I will create six components in three pairs.



I will need a pair for Toggle, for RadioBox, and for CheckBox. It's essential that this component has at least one text node. I then select the first pair of my newly created components, and from the Plugin menu, I press the Combine Selected Components as States button.

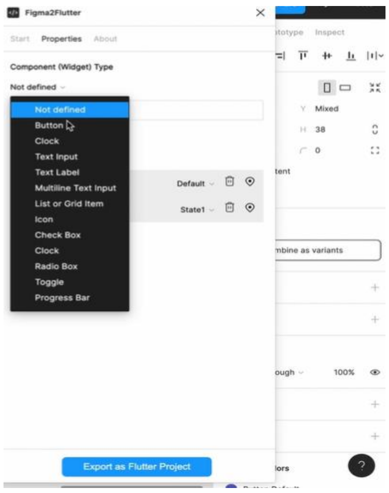


Select the first pair of my newly created components, and from the Plugin menu, press the combine Selected Components as States button.

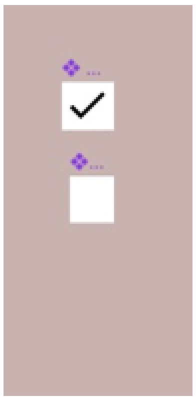


We found the following states of this component. You can auto-create missing component states if necessary.

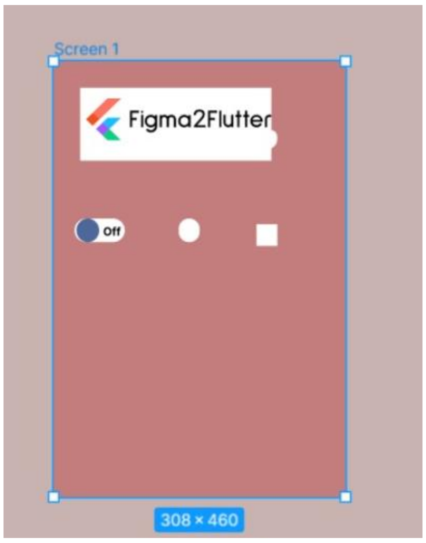
Next,i'll select toggle from the drop down menu,I then select the suitable states for my components.The on button will become active and the off button will become inactive



I then select the third pair of components and again,from the plugin menu,i'll press the combine selected elements as states.For the on checkbox,I have selected the on toggle and for the off checkbox,I selected the off toggle



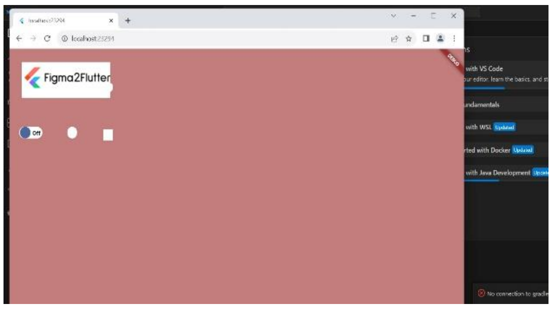
Next,I create an instance of each component and put it inside my screen

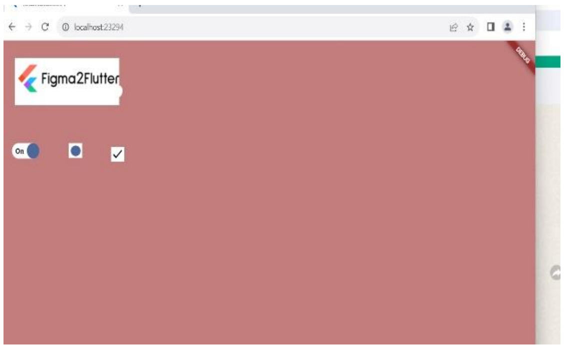


I export the project and then I run flutter pub get to obtain all the dependencies

5.4.1 Final Output

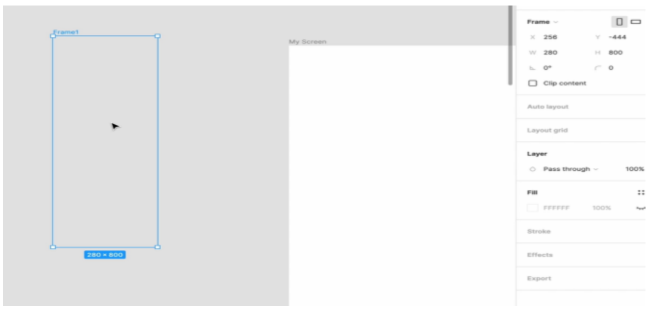
We can see the change when I press the toggle off,the active button and the checkbox





5.5 Sliders

Launch Figma and draw a figma frame to serve as the main screen. Name it "MyScreen."

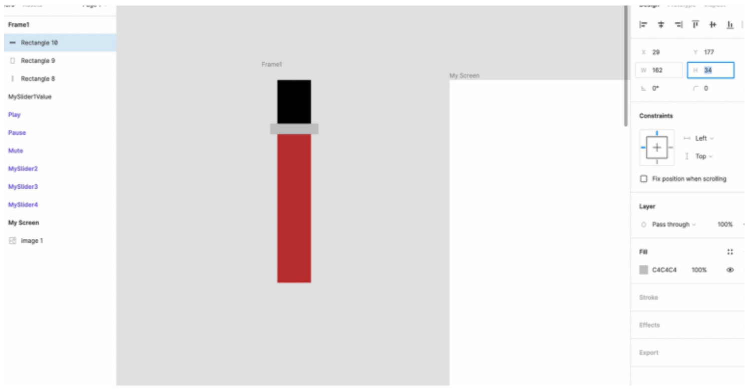


Create another frame within the main screen, which will represent the slider. Inside this frame, add three rectangles:

- First rectangle: Represents the slider length and should be colored appropriately for a visually pleasing appearance.

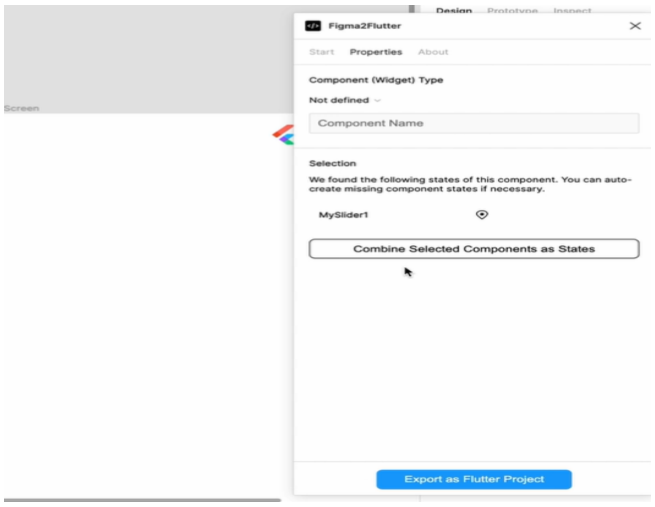
- Second rectangle: Represents the inactive part of the slider and should also be colored for contrast.

- Third rectangle: Represents the user's thumb and should be sized appropriately.

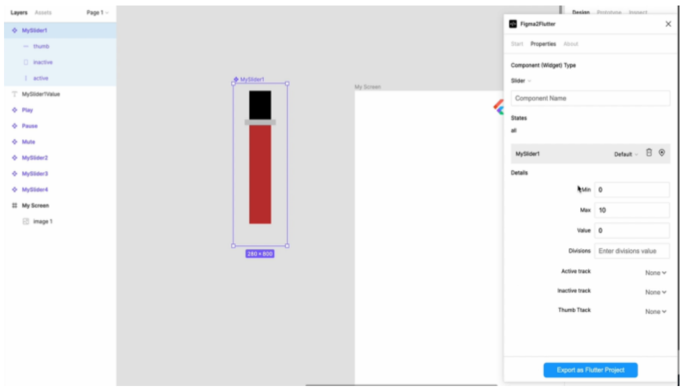


X 29, Y 177, W 162, H 34, 0°, 0; Pass through 100%; Fill C4C4C4 100%

Select the newly created component and click on the "Combine Selected Components as States" button from the plugin menu. Choose "Slider" from the dropdown.

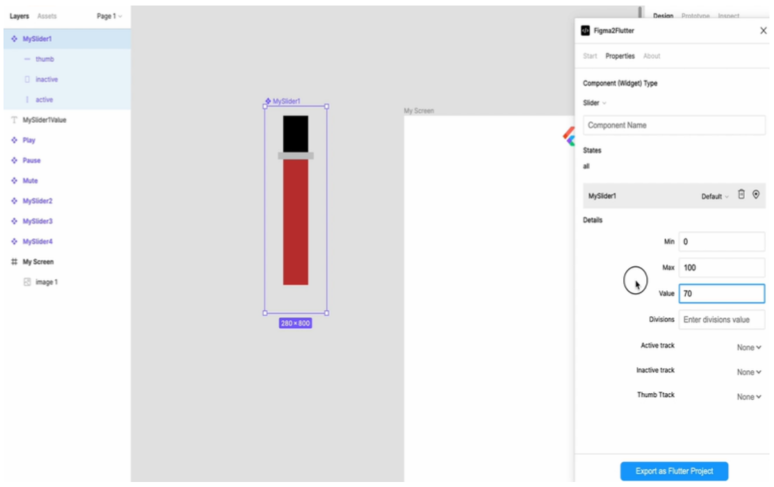


We found the following states of this component. You can auto-create missing component states if necessary.



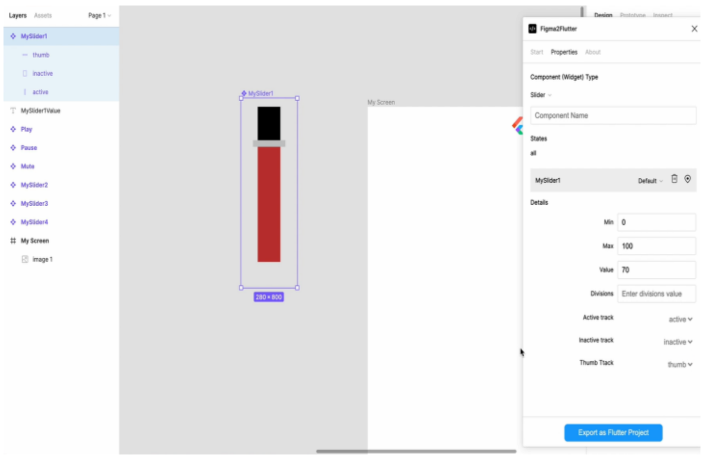
Min 0
Max 10
Value 0
Divisions Enter divisions value
Active track None
Inactive track None
Thumb Ttack None

Set the "Min Max Value" property under "Details" to define the range of the slider, and set the initial value to 70.



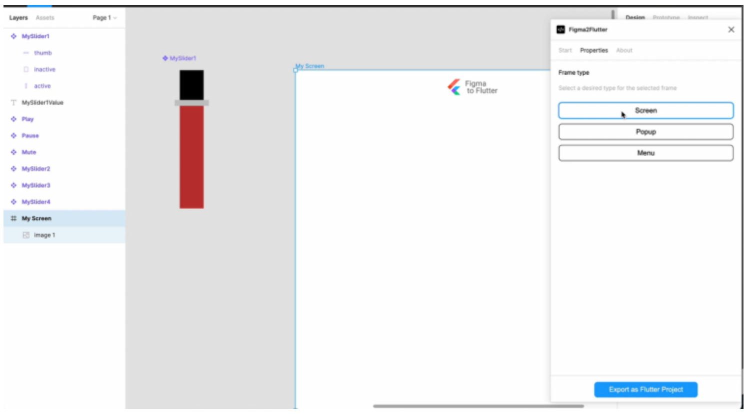
Component (Widget) Type
Slider
Component Name
States
all
MySlider1 Default
Details
Min 0
Max 100
Value 70
Divisions Enter divisions value
Active track None
Inactive track None
Thumb Ttack None

Configure the active, inactive, and thumb rectangles using the dropdown buttons in the plugin to associate them with the slider.

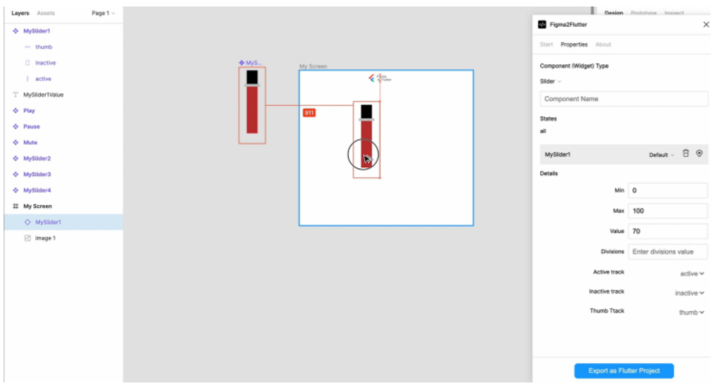


Component (Widget) Type
Slider
Component Name
States
all
MySlider1 Default
Details
Min 0
Max 100
Value 70
Divisions Enter divisions value
Active track active
Inactive track inactive
Thumb Ttack thumb

Annotate the "MyScreenFrame" as a screen from the plugin menu to designate it as the main screen.

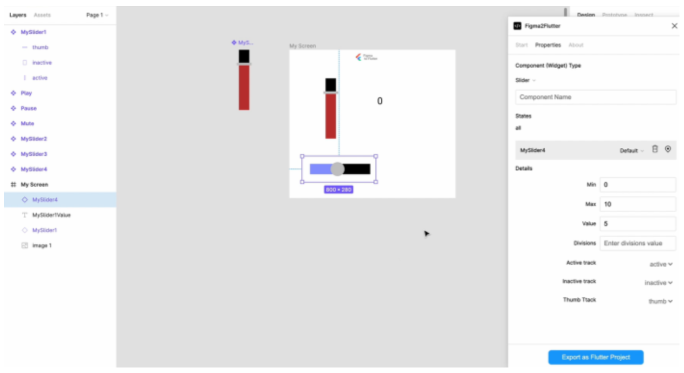


Create an instance of the slider component and place it within the screen frame.



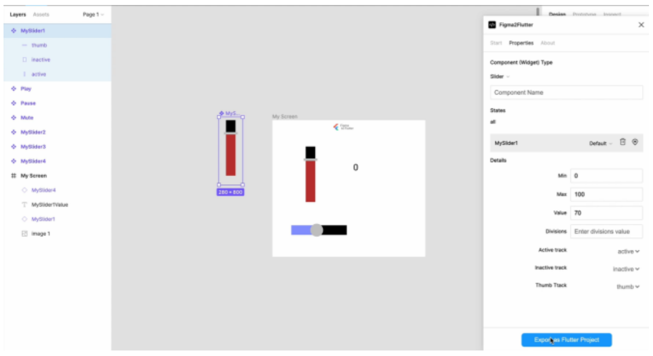
Component (Widget) Type
Slider
Component Name
States
all
MySlider1 Default
Details
Min 0
Max 100
Value 70
Divisions Enter divisions value
Active track active
Inactive track inactive
Thumb Ttack thumb

Add text nodes to display the current slider value and label them accordingly. This step is optional but can be helpful for displaying the slider value.



Component (Widget) Type
Slider
Component Name
States
all
MySlider4 Default
Details
Min 0
Max 10
Value 5
Divisions Enter divisions value
Active track active
Inactive track inactive
Thumb Ttack thumb

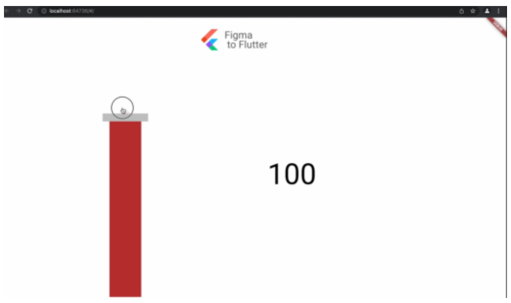
Click the "Export as Flutter Project" button in the Figma plugin and allow access to generate the code. Once the code is generated, download it.

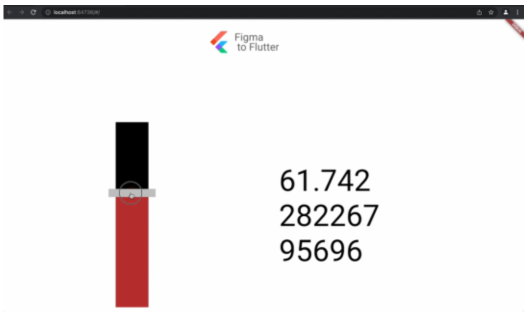


Component (Widget) Type
Slider
Component Name
States
all
MySlider1 Default
Details
Min 0
Max 100
Value 70
Divisions Enter divisions value
Active track active
Inactive track inactive
Thumb Ttack thumb

I then run the “flutter pub get" command in the Flutter console to download all dependencies.

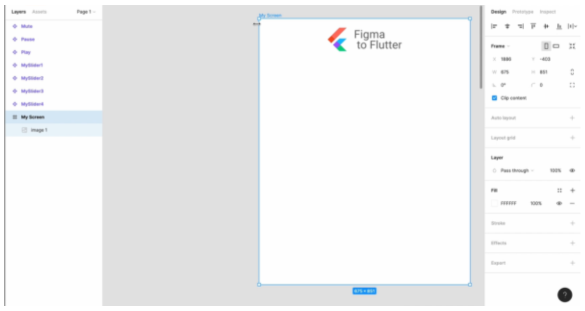
5.5.1 Final Output



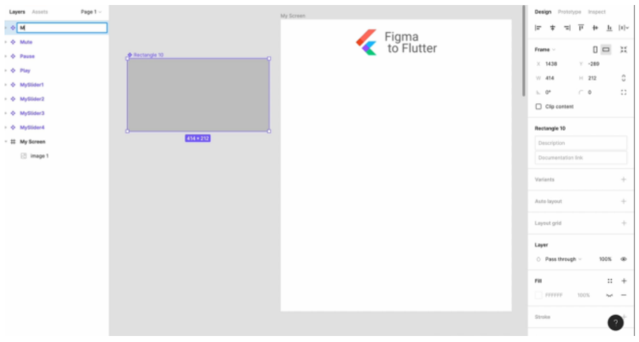


5.6 Video Component

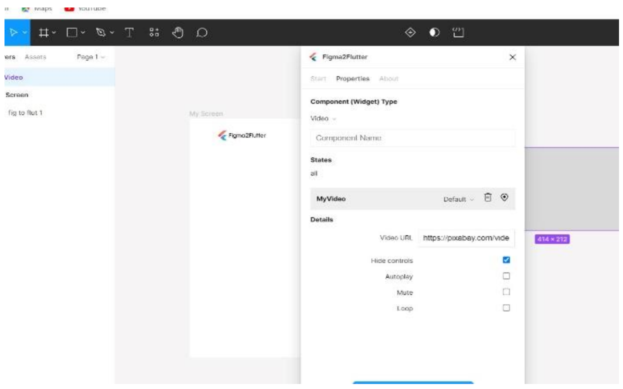
Create a screen design in Figma. This will be my main screen



I then draw a rectangle, creating a component and renaming it



I then select the newly created component, and from the Plugin menu, press the Combine Selected Components as States button. From the drop-down menu, select Video. Under Details, set the Video URL property



Figma2Flutter

Start Properties About

Component (Widget) Type

Video

Component Name

States

all

MyVideo Default

Details

Video URL https://pixabay.com/vide

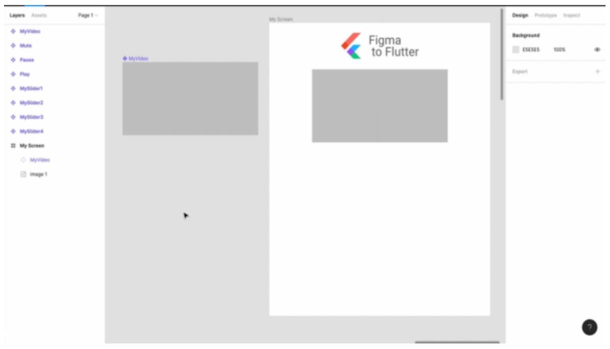
Hide controls [✓]

Autoplay ☐

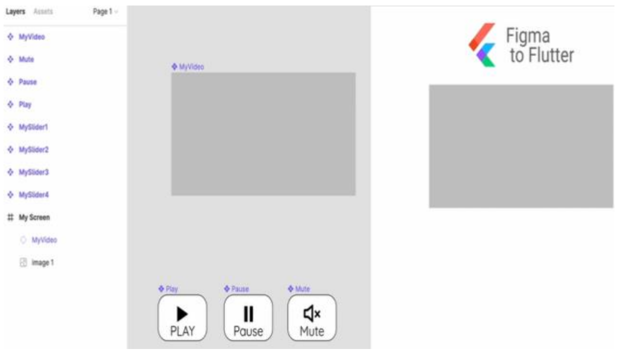
Mute ☐

Loop ☐

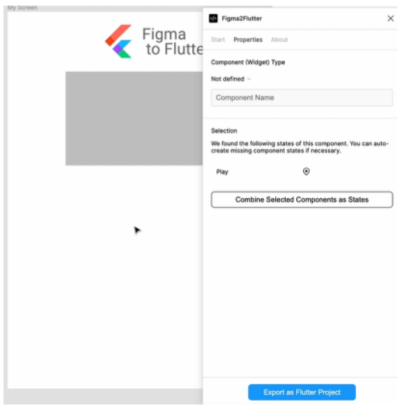
Now I create a Video component and make an instance of it and put it into the screen.



I then draw a few button components which include play, pause and mute



From the Plugin Menu, I then set these buttons as button components



Figma2Flutter

Start Properties About

Component (Widget) Type

Not defined

Component Name

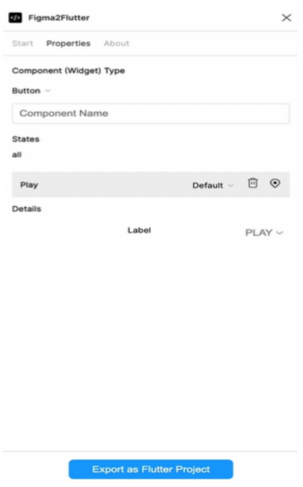
Selection

We found the following states of this component. You can auto-create missing component states if necessary.

Play

Combine Selected Components as States

Export as Flutter Project



Figma2Flutter

Start Properties About

Component (Widget) Type

Button

Component Name

States

all

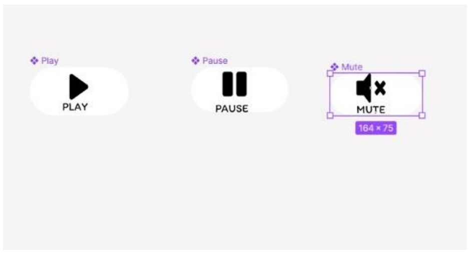
Play Default

Details

Label PLAY

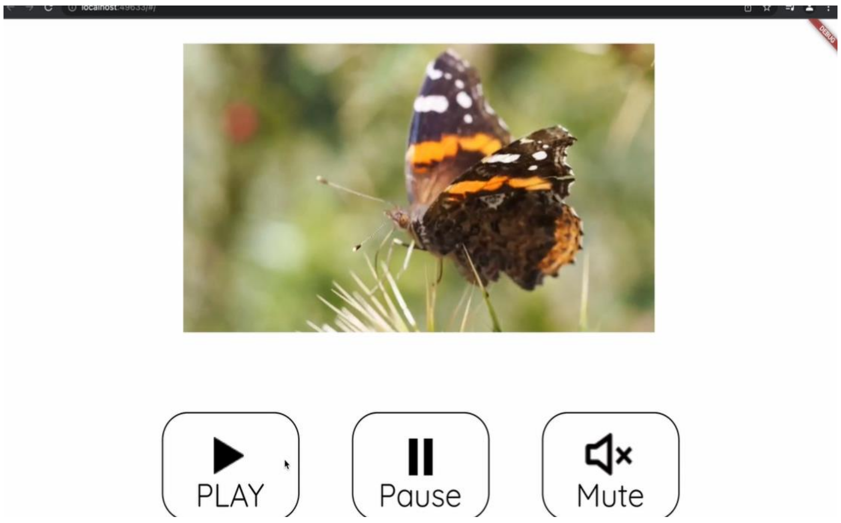
Export as Flutter Project

Then I create an instance of a button and put it inside the frame



I then export as a Flutter project and see how it works and run “flutter packages upgrade” commands

5.6.1 Final Output


PLAY
Pause
Mute