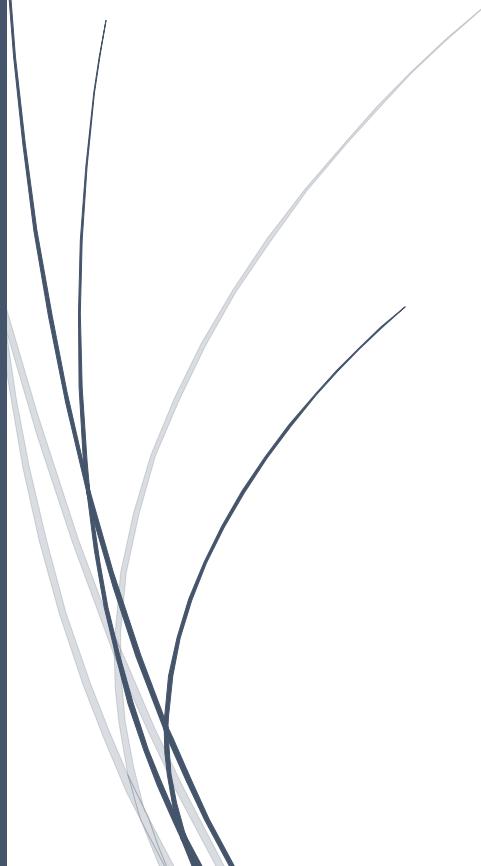


1/5/2024

PMH

LAB SESSION 2: UI DESIGN &
CREATING FLUTTER APPS



To: Mr Shiam Beeharry
BY: KRITIKA BISSESSUR

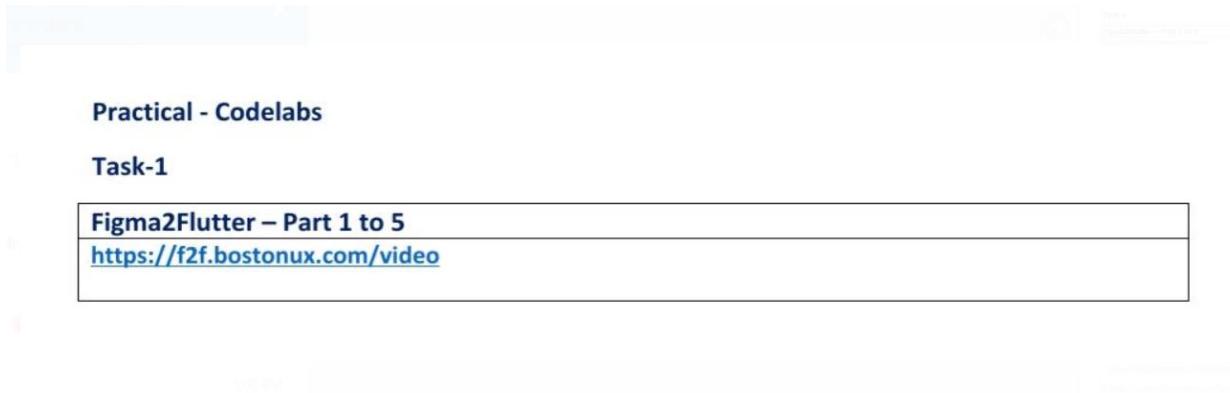
Contents

1	Task 1	1
1.1	Brief Overview.....	1
1.2	Navigation.....	1
1.2.1	Steps for navigation	1
1.2.2	Final Output	7
1.3	Popups & Images	8
1.3.1	Steps for popups & images	8
1.3.2	Final output.....	13
1.4	Menus	14
1.4.1	Steps for Menus	14
1.4.2	Final output.....	19
1.5	Vertical ListViews	20
1.5.1	Steps for ListViews	20
1.5.2	Final output.....	25
1.6	Horizontal ListViews.....	26
1.6.1	Steps for ListViews	26
1.6.2	Final output.....	30
2	Task 2	31
2.1	Brief Overview.....	31
2.2	Steps.....	31
2.2.1	Starter App.....	31
2.2.2	Modification in code	32
2.2.3	Typography and icons	33
2.2.4	New package	34
2.2.5	Importing the package	34
2.2.6	Dark Mode and bright Mode	35
2.2.7	Add adaptive design.....	36
2.2.8	Responsive layout	36
2.2.9	Whitespace	37
2.2.10	Adding motion & animation.....	38
2.2.11	Adding hover states	38
2.2.12	Play button.....	39



2.3	Final Output	39
3	Task 3	40
3.1	Brief Overview.....	40
3.2	Steps.....	41
3.3	Final Output	47
4	Task 4	49
4.1	Brief overview	49
4.2	Steps.....	49
4.3	Steps & explanation	54
4.4	Final Output	60

1 Task 1



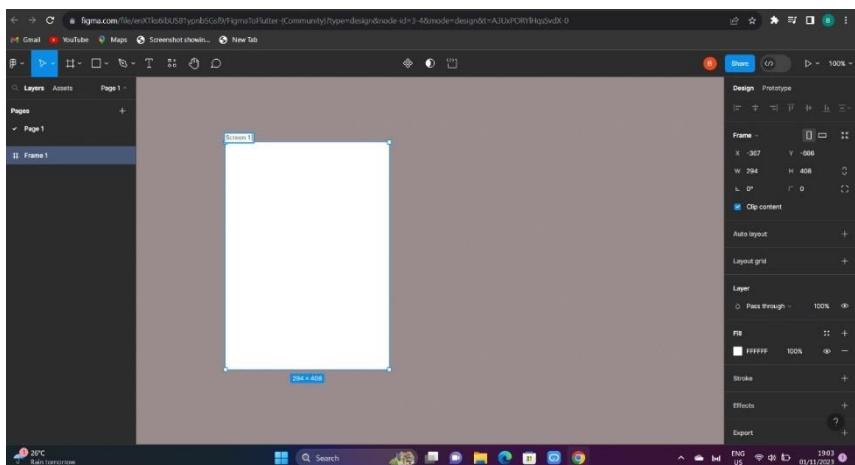
1.1 Brief Overview

The task involves converting a Figma project to a Flutter project. The Figma design includes elements like navigation, menus, and a horizontal list view. The goal is to replicate these design components in Flutter for a cohesive and functional mobile application.

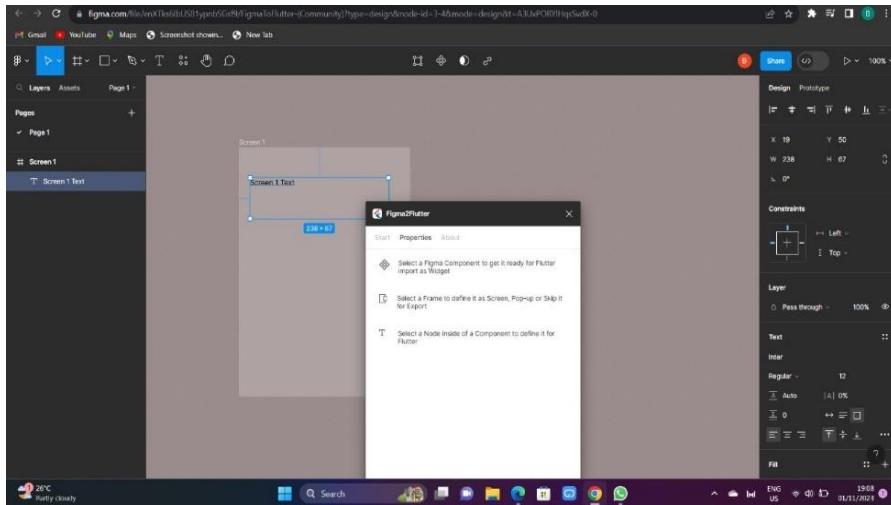
1.2 Navigation

1.2.1 Steps for navigation

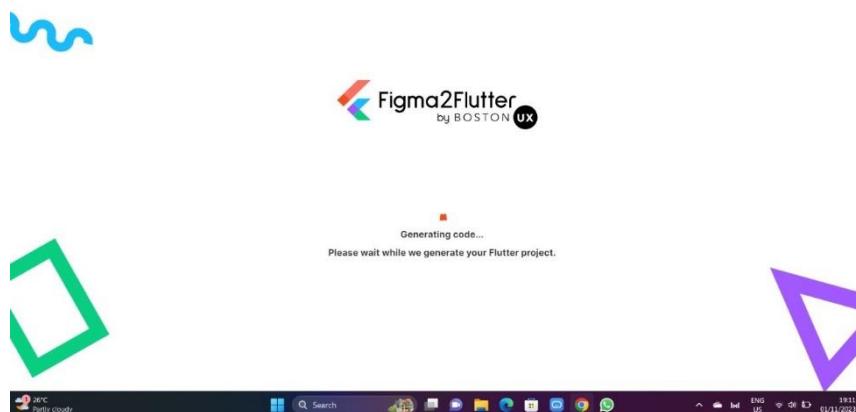
1. To begin, I will visit the link <https://f2f.bostonux.com/video>. Next, I will navigate through the Figma to Flutter project, which is a straightforward process. Firstly, I will create a Figma frame that will serve as the main screen. Then, I will proceed by adding text notes within the frame. Following that, I will define the background color.



- I'll kick off the process by selecting "Plugins" from the main toolbar and then opt for "Figma to Flutter". Following this, I'll carefully pinpoint the screen I wish to work on and duly label it as "Screen" within the user-friendly interface of the Figma to Flutter plugin. I'll finish the integration operation effortlessly by pressing "Export as Flutter Project" button.

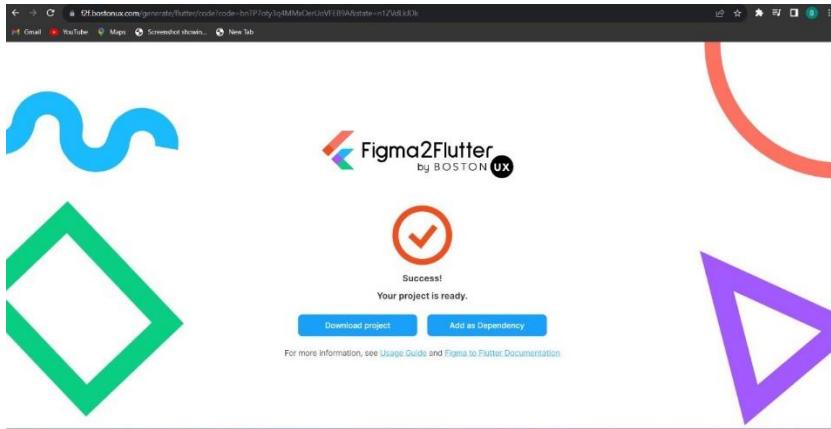


- Browser should automatically open with the figma tab. I'll press the Allow Access button to allow the plugin to generate code.

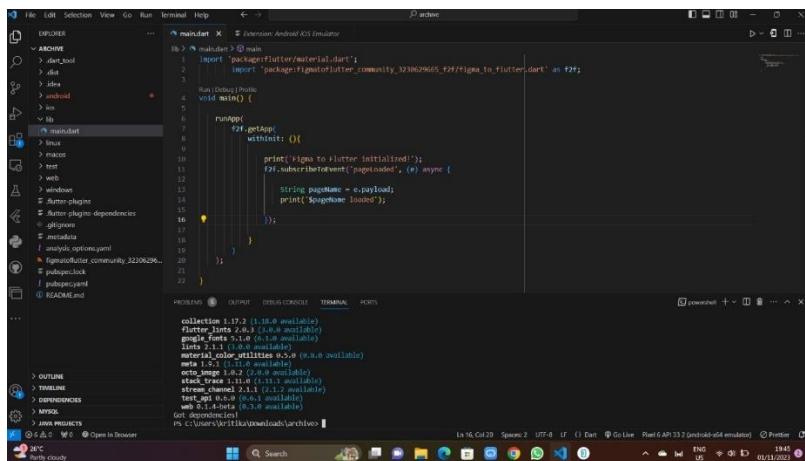




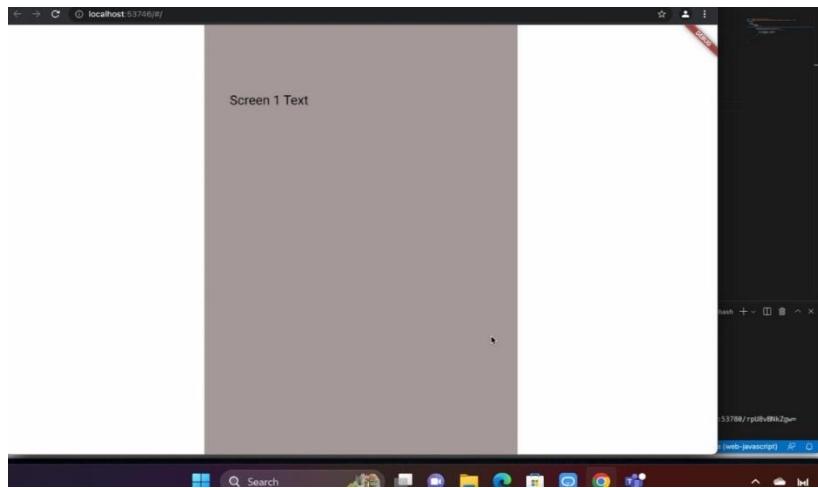
- Once the code is generated, I'll download it. My next step is to open the downloaded project named Archive in the Flutter environment.



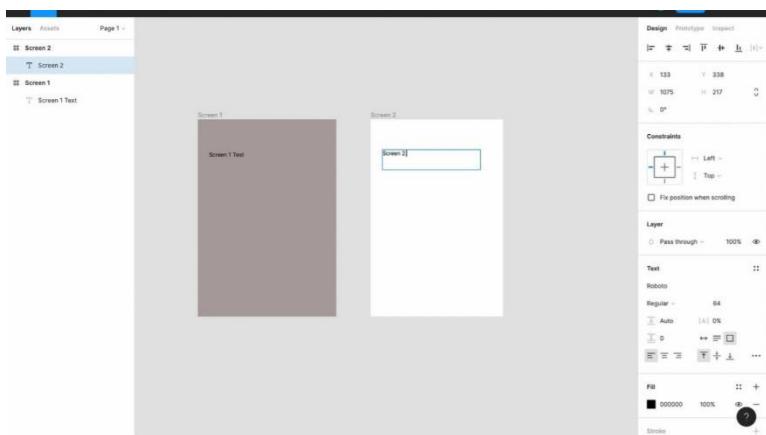
- This tutorial uses Visual Studio Code. In the Flutter console, I'll run the flutter pub get command to download all dependencies required to run the project. I only have to use this command once when running the project on the first time. Next, I'll run the flutter run command to start my project and then wait until it appears.

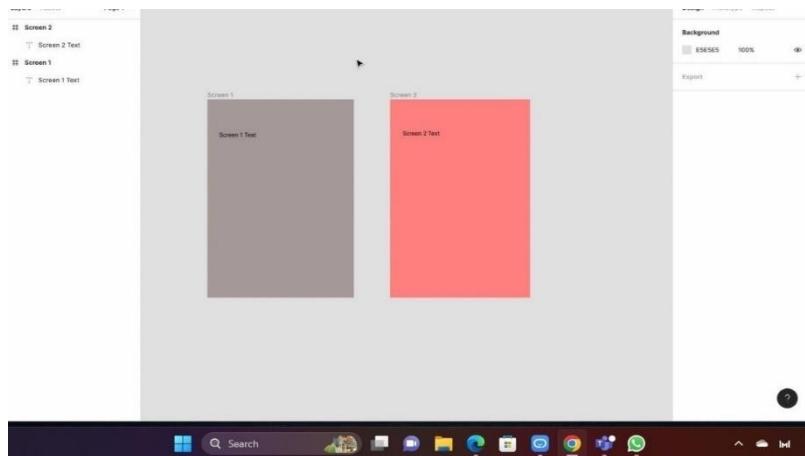


6. As you can see, it appeared while I run flutter run

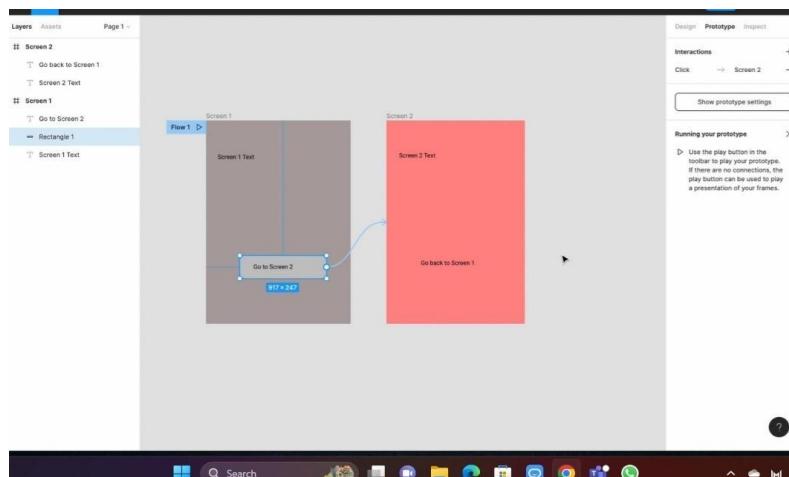


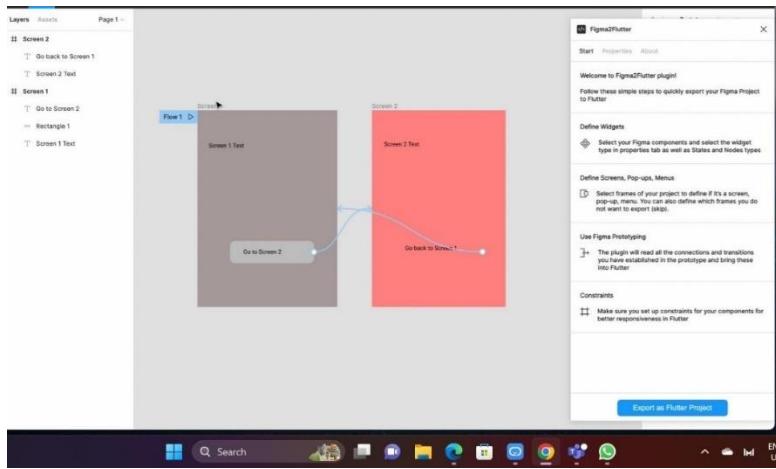
7. Subsequently, I will proceed to create a second frame, insert new text elements within it, and apply a suitable background color. Following this, I will proceed to design a text button in screen one, facilitating navigation to screen two. Finally, I'll create corresponding text in screen two to enable seamless navigation back to screen one.



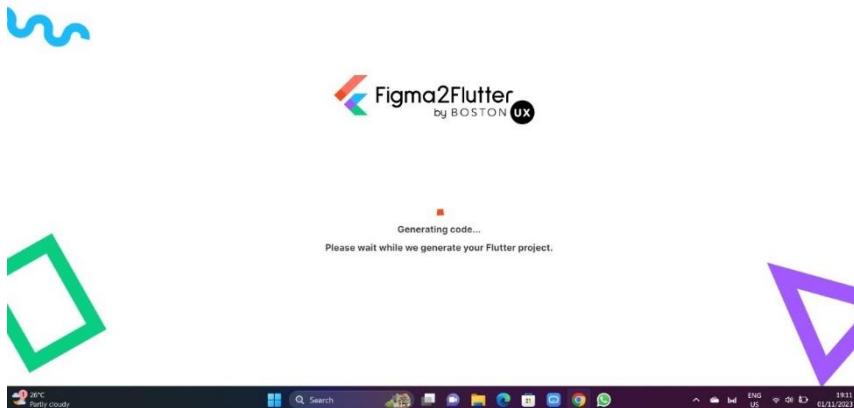


- Following the frame creation, I will properly label the newly created frame as "screen" using the options provided in the plugin menu. Then, from the Figma prototype tab, I'll seamlessly integrate navigation. Lastly, I'll add the button within my web browser.

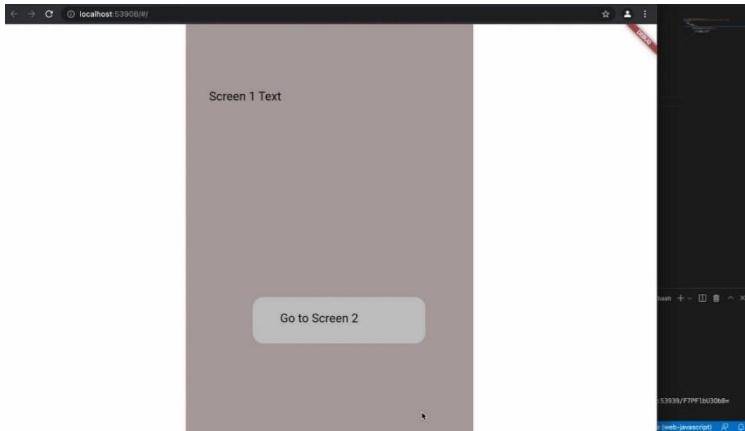




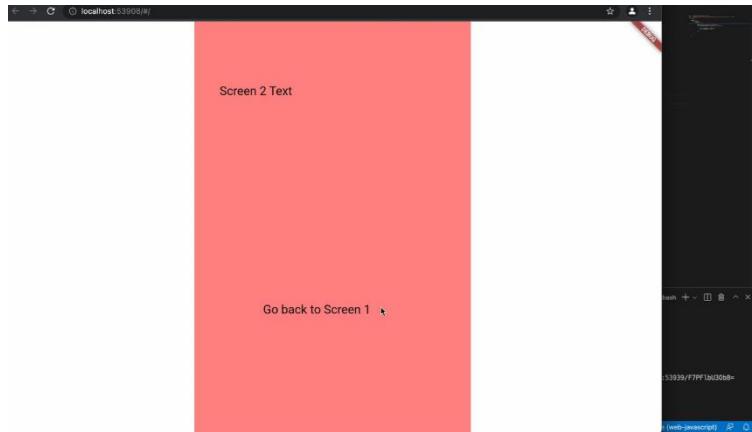
9. This time, there is no need to download the project again. I simply leave the browser and go to the Flutter console. One note, the download as dependency option is used for adding generated code to an existing project.



10. Thereafter, I will execute the "flutter packages upgrade" command, which is essential whenever modifications are made to a design. Upon making updates, I can verify the changes in the Flutter console log, ensuring that the project has been appropriately updated. As the final step, I will initiate the "flutter run" command. Upon navigating to localhost and clicking on screen one, followed by the transition to screen two, I will observe the color shift from gray to pink.



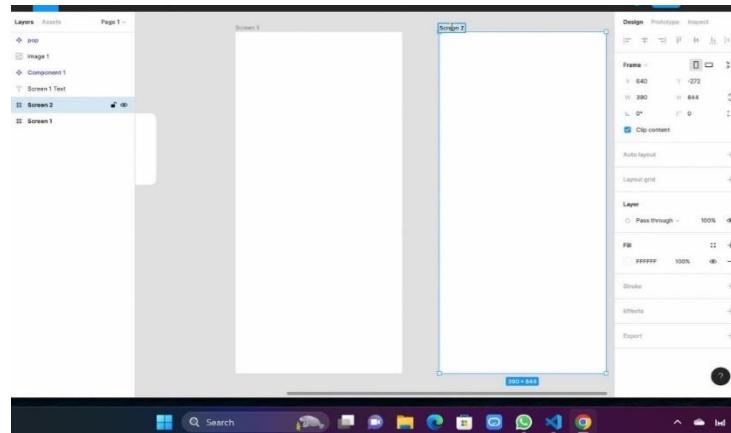
1.2.2 Final Output



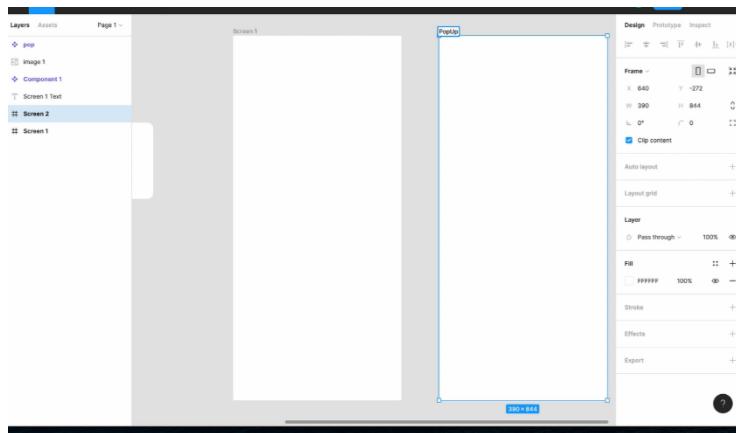
1.3 Popups & Images

1.3.1 Steps for popups & images

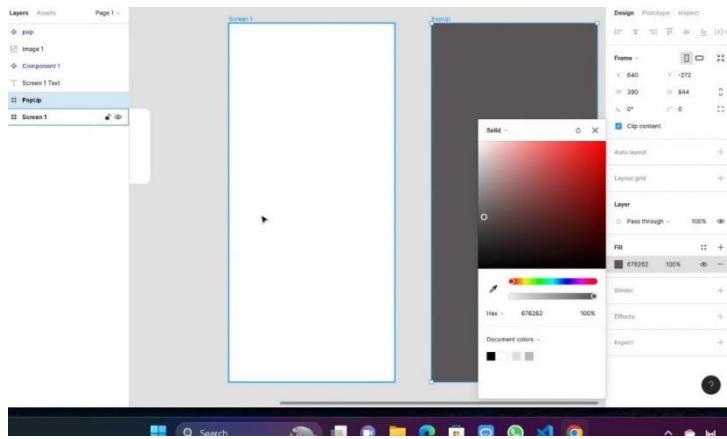
1. The first thing I'll do is draw a Figma frame. This will be my main screen.



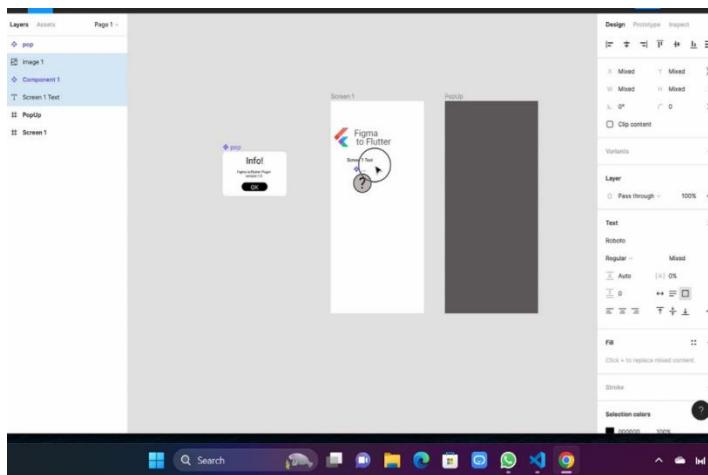
2. Next, I'll create another frame in the same way to serve as a pop-up.



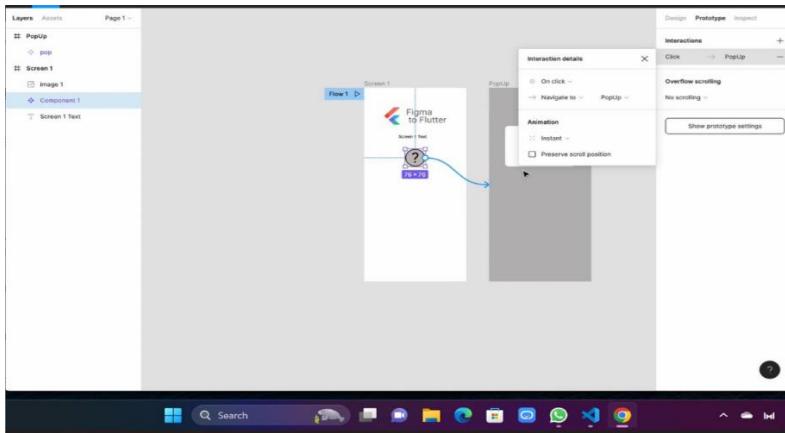
- I'll set the background color for the pop-up screen.



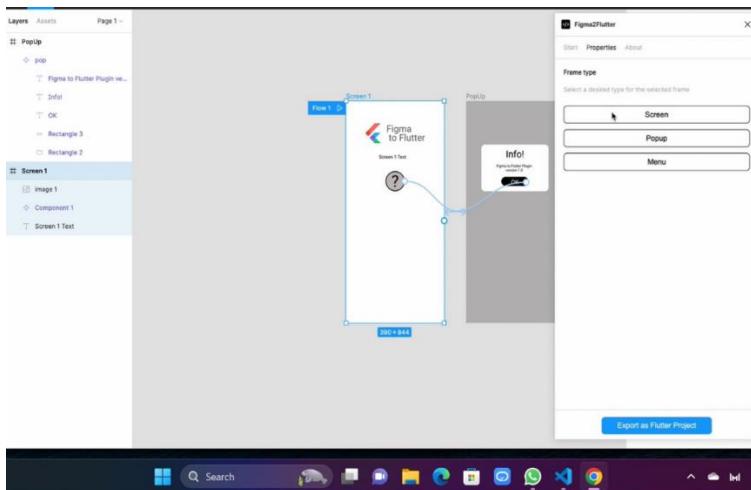
- I'll incorporate a few icons to prompt the pop-up. Additionally, on the second screen, I'll adjust the fill opacity to approximately 35% to achieve a visually appealing appearance.



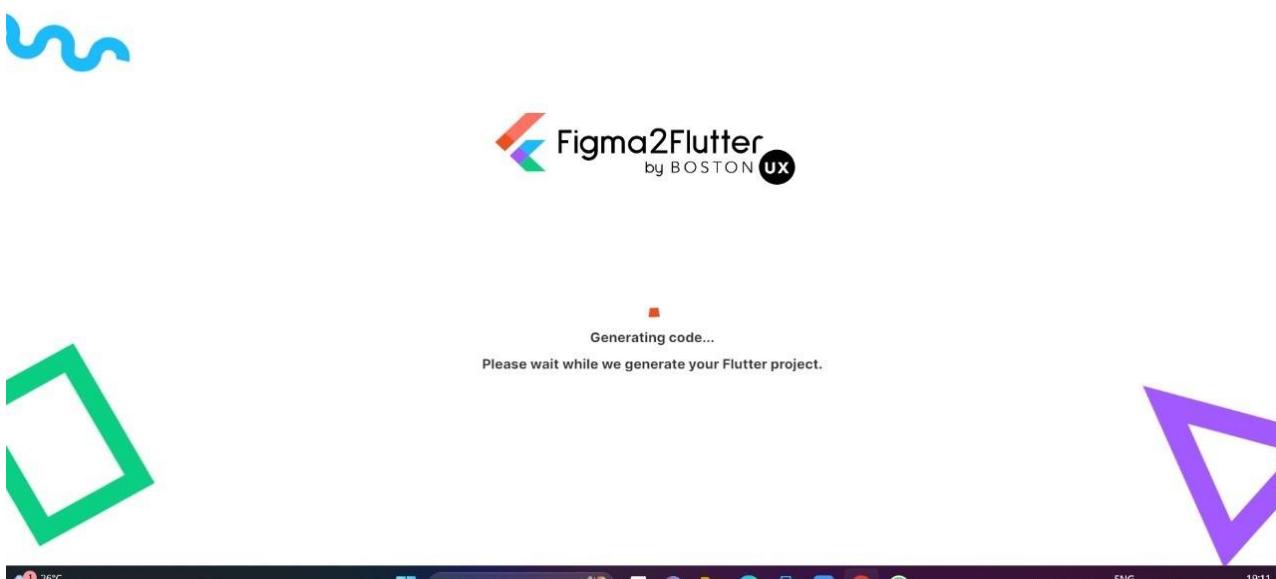
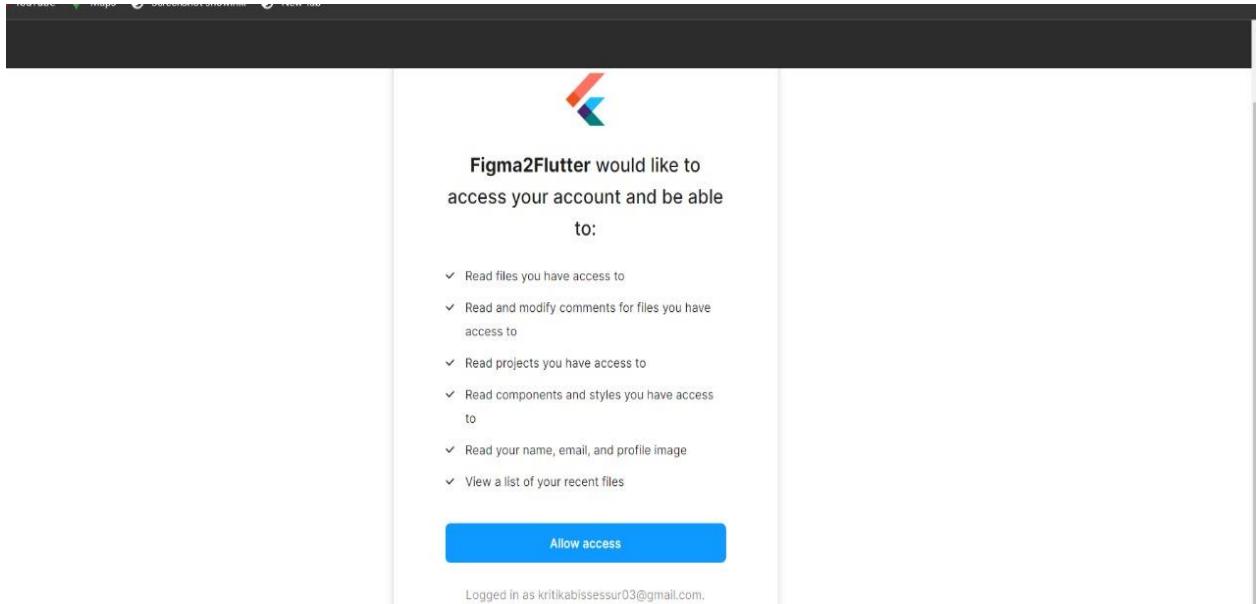
5. I'm ready to navigate to the prototype tab and establish connections between the icon and the message.



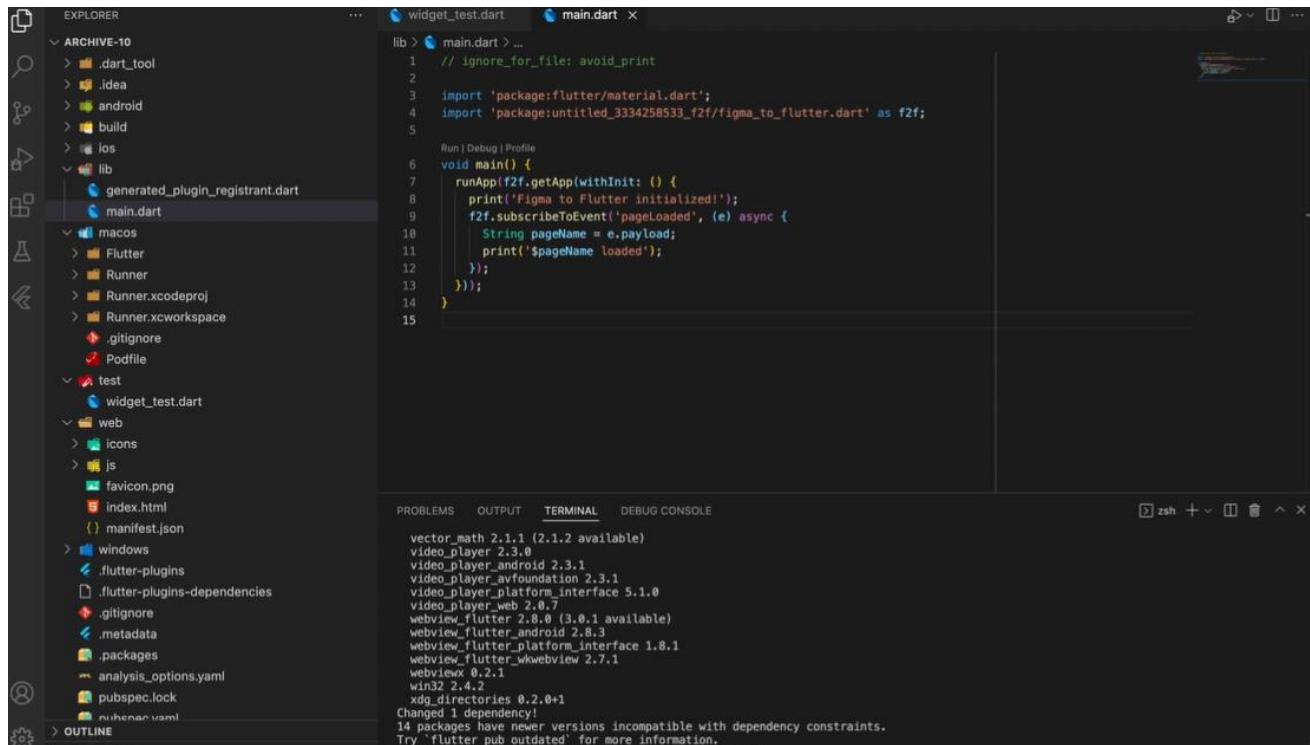
6. Next, I'll select Screen1 and specify the screen property using the Figma to Flutter plugin. To define the pop-up frame, I'll access the pop-up property in the Figma to Flutter plugin menu and then proceed to click the 'Export as Flutter Project' button.



7. Subsequently, my web browser should automatically open with the Figma tab. Upon this, I'll grant access by clicking the 'Allow Access' button to enable the plugin to generate the code.



8. Then I open visual code and proceed to run the 'flutter pub upgrade' command to update all dependencies.



The screenshot shows the Visual Studio Code interface with a Flutter project open. The Explorer sidebar on the left lists files and folders such as .dart_tool, .idea, android, build, ios, lib, main.dart, macos, Runner, Runner.xcodeproj, Runner.xcworkspace, .gitignore, Podfile, test, widget_test.dart, web, icons, js, favicon.png, index.html, manifest.json, windows, flutter-plugins, flutter-plugins-dependencies, .gitignore, .metadata, packages, analysis_options.yaml, pubspec.lock, and pubspec.yaml. The main editor area displays the main.dart file:

```

lib > main.dart ...
1 // ignore_for_file: avoid_print
2
3 import 'package:flutter/material.dart';
4 import 'package:untitled_3334258533_f2f/figma_to_flutter.dart' as f2f;
5
6 Run | Debug | Profile
7 void main() {
8   runApp(f2f.getApp(withInit: () {
9     print('Figma to Flutter initialized!');
10    f2f.subscribeToEvent('pageLoaded', (e) async {
11      String pageName = e.payload;
12      print('$pageName loaded');
13    });
14  }));
}

```

The Problems, Output, Terminal, and Debug Console tabs are visible at the bottom. The Terminal tab shows the output of the 'flutter pub upgrade' command:

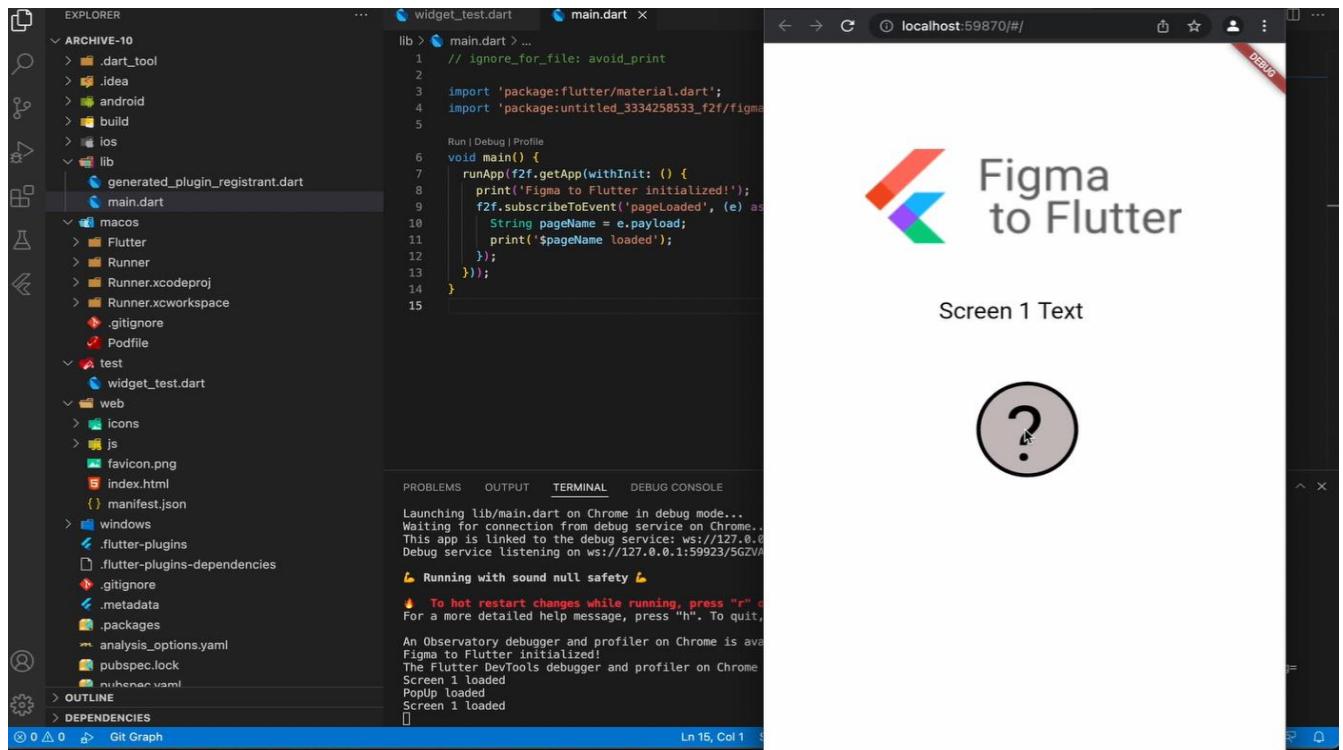
```

vector_math 2.1.1 (2.1.2 available)
video_player 2.3.0
video_player_android 2.3.1
video_player_avfoundation 2.3.1
video_player_platform_interface 5.1.0
video_player_web 2.0.7
webkit_embedding 2.0.0 (3.0.1 available)
webview_flutter_android 2.8.3
webview_flutter_platform_interface 1.8.1
webview_flutter_webview 2.7.1
webviewx 0.2.1
win32 2.4.2
xdg_directories 0.2.0+1
Changed 1 dependency!
14 packages have newer versions incompatible with dependency constraints.
Try 'flutter pub outdated' for more information.

```

1.3.2 Final output

I will press the Menu icon to preview the pop-up's appearance, keeping in mind the subtle visual contrast between menus and pop-ups. Additionally, I can verify in the console log that a particular screen or pop-up has been loaded, which provides confirmation each time a new screen or pop-up is displayed.



```

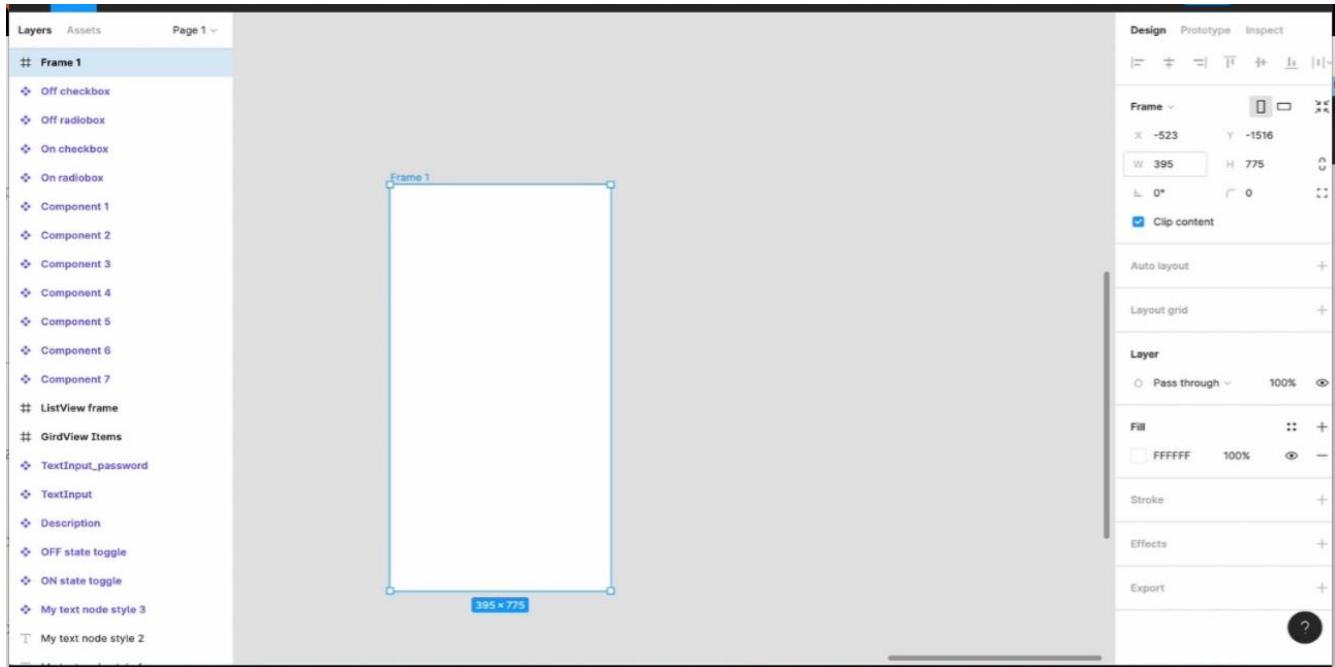
EXPLORER          widget_test.dart  main.dart
...               lib > main.dart > ...
ARCHIVE-10        lib > main.dart > ...
<                1 // ignore_for_file: avoid_print
>                2
>                3 import 'package:flutter/material.dart';
>                4 import 'package:untitled_3334258533_f2f/figma'
>                5
>                6 Run | Debug | Profile
>                7 void main() {
>                8   runApp(f2f.getApp(withInit: () {
>                9     print('Figma to Flutter initialized!');
>                10    f2f.subscribeToEvent('pageLoaded', (e) {
>                11      String pageName = e.payload;
>                12      print('$pageName loaded');
>                13    });
>                14  }));
>                15
PROBLEMS          OUTPUT      TERMINAL      DEBUG CONSOLE
Launching lib/main.dart on Chrome in debug mode...
Waiting for connection from debug service on Chrome...
This app is linked to the debug service: ws://127.0.0.1:59923/5GZVA
Debug service listening on ws://127.0.0.1:59923/5GZVA
Running with sound null safety
To hot restart changes while running, press "r"
For a more detailed help message, press "h". To quit,
An Observatory debugger and profiler on Chrome is available at:
Figma to Flutter initialized!
The Flutter DevTools debugger and profiler on Chrome
Screen 1 loaded
PopUp loaded
Screen 1 loaded
Screen 1 loaded
Ln 15, Col 1

```

1.4 Menus

1.4.1 Steps for Menus

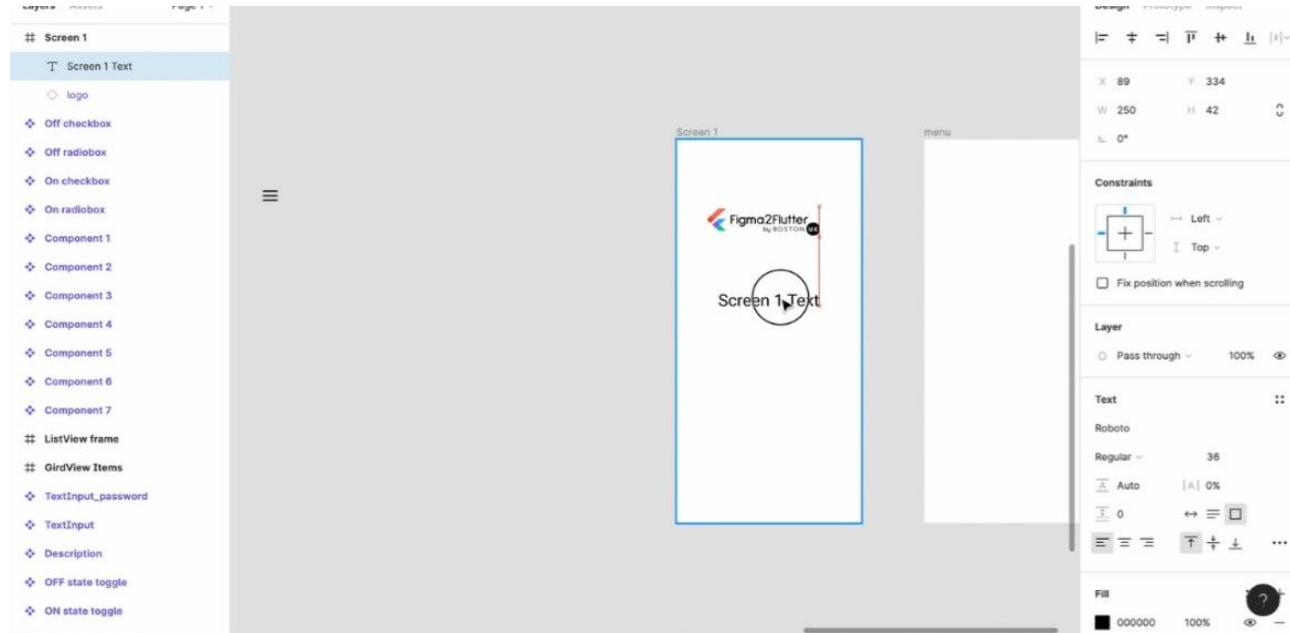
- I will first proceed by drawing a Figma frame. This will be the main screen.



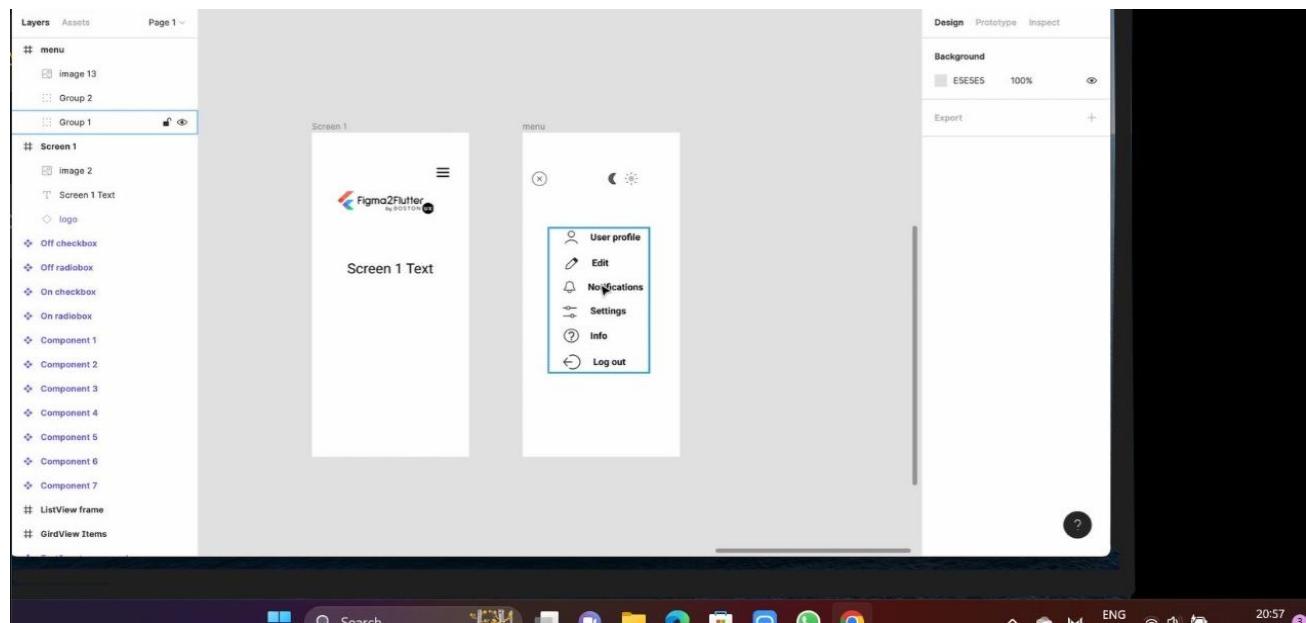
- Now, I will create a second frame the same as the first. That will be the menu.



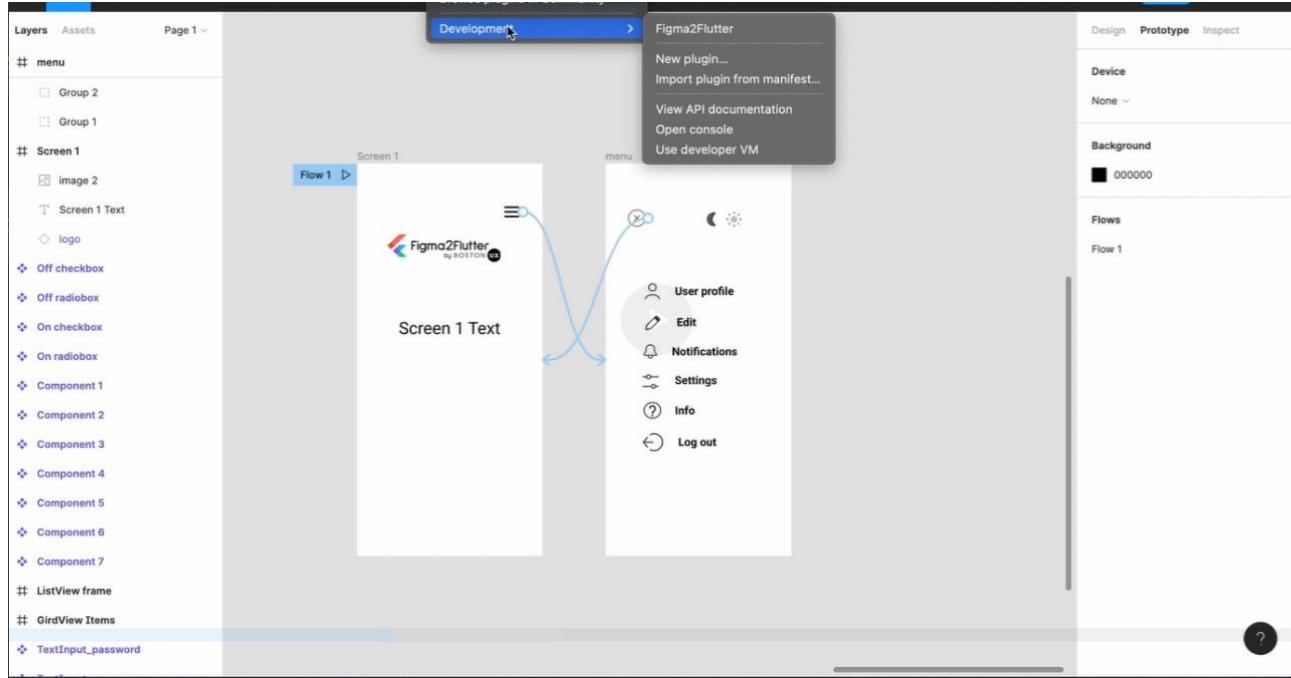
3. I will include text and images within my main screen.



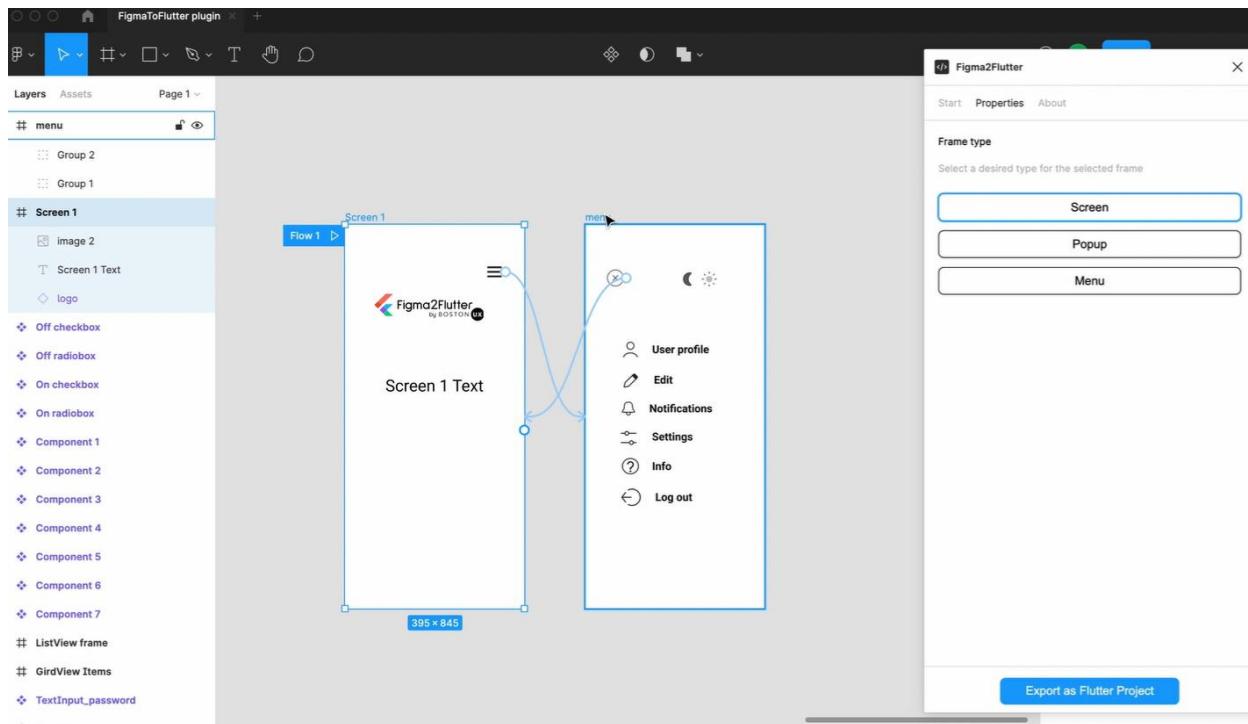
4. Additionally, I'll incorporate icons to activate the menu, along with some simple design elements.



- From the Figma Prototype tab, I add navigation for the icons for opening and closing the menu, then from the top bar, select Plugins, Figma to Flutter.



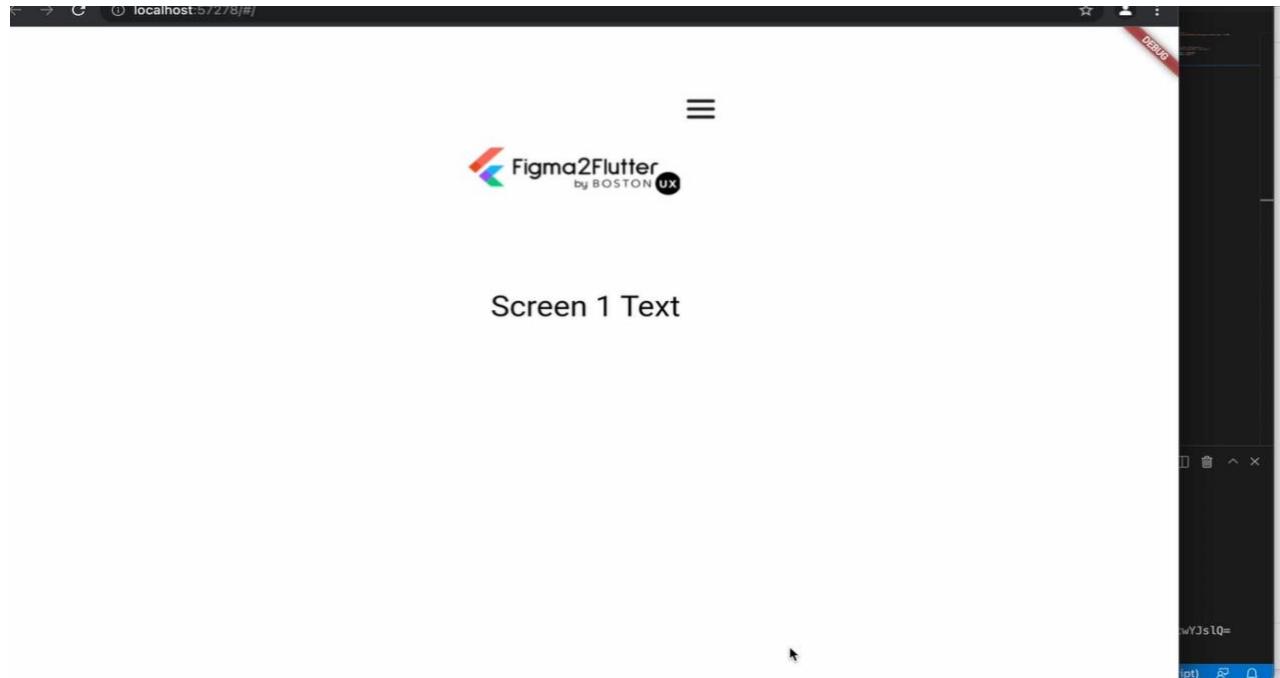
6. Select Screen1 and annotate the screen property in the Figma to Flutter plugin menu. For the menu frame,I select the Menu property in the Figma to Flutter plugin menu. Then I press the blue Export as Flutter Project button.



7. I then press the Allow Access button to let the plugin generate code.

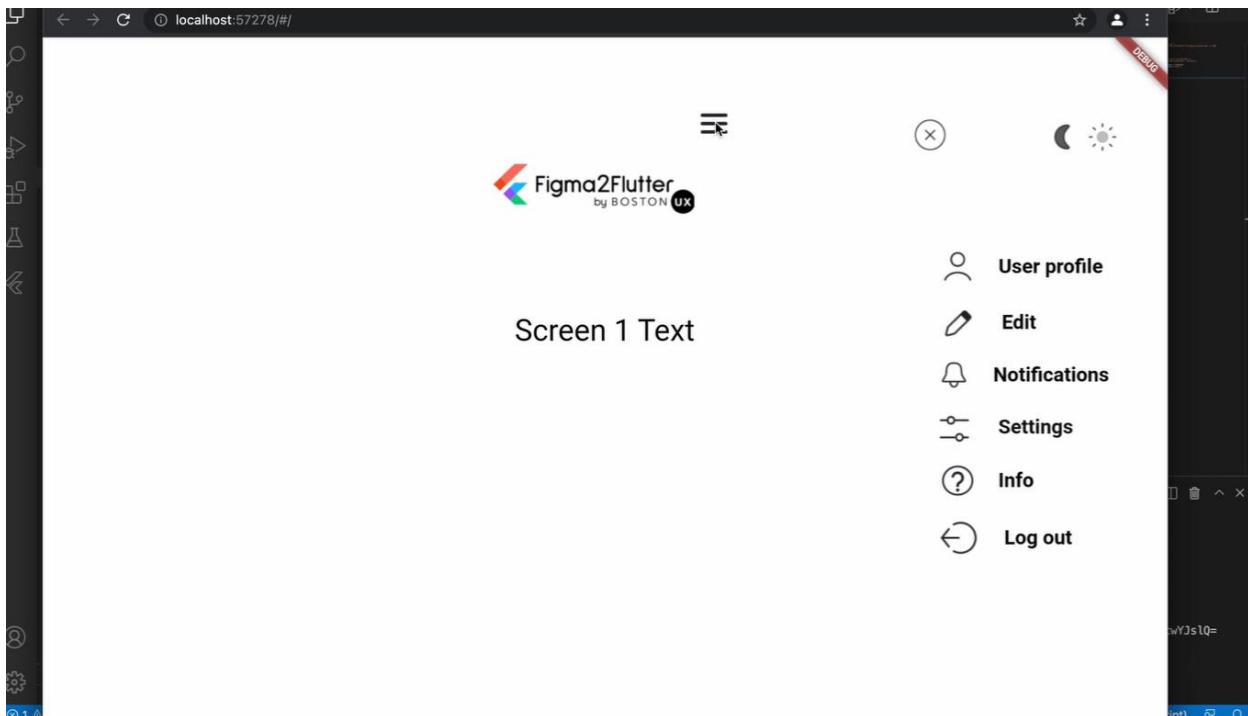


8. In the Flutter console, I run the 'flutter pub get' command to download all the dependencies required for running the project. As per as the video, It's important to note that this command needs to be used only once when initially running my project. Therefore i simply run the 'flutter packages upgrade' command.



1.4.2 Final output

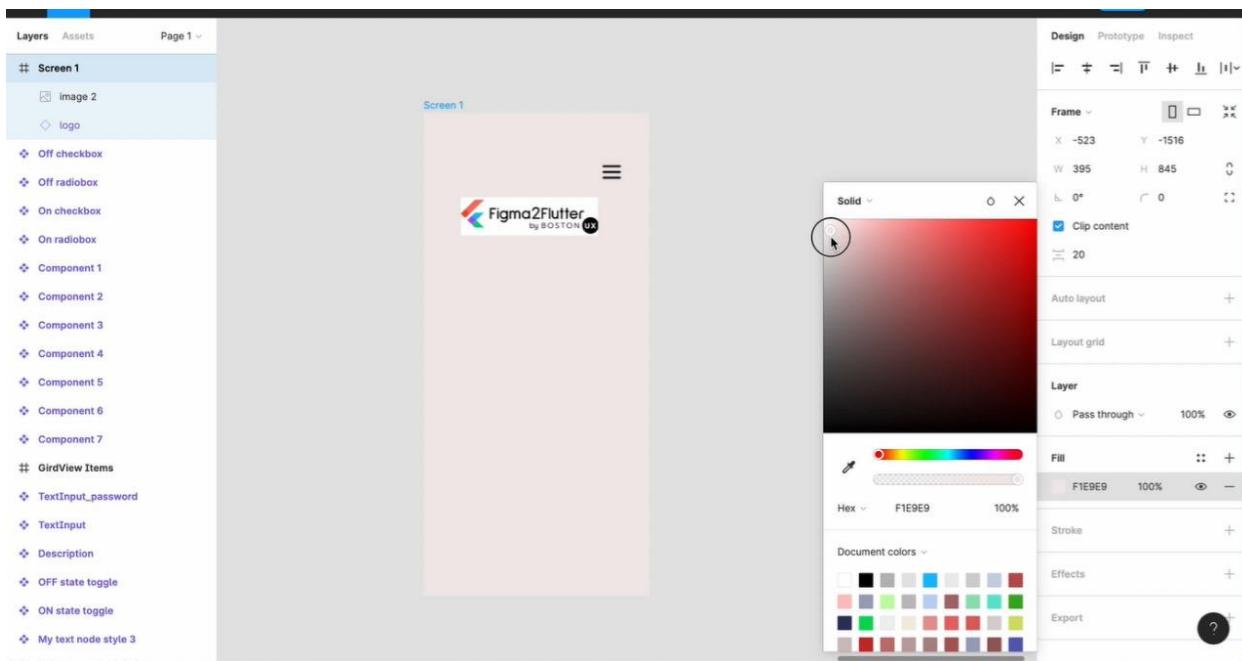
I need to wait until my project appears. Then, I play the icon and see how the menu appears. In the console log, I see confirmation that a single file has been created. In the console log, I see the confirmation that a specific screen or menu has been loaded. For instance, screen 1 loaded, or menu loaded, or a similar confirmation will appear every time a new screen or popup is displayed.



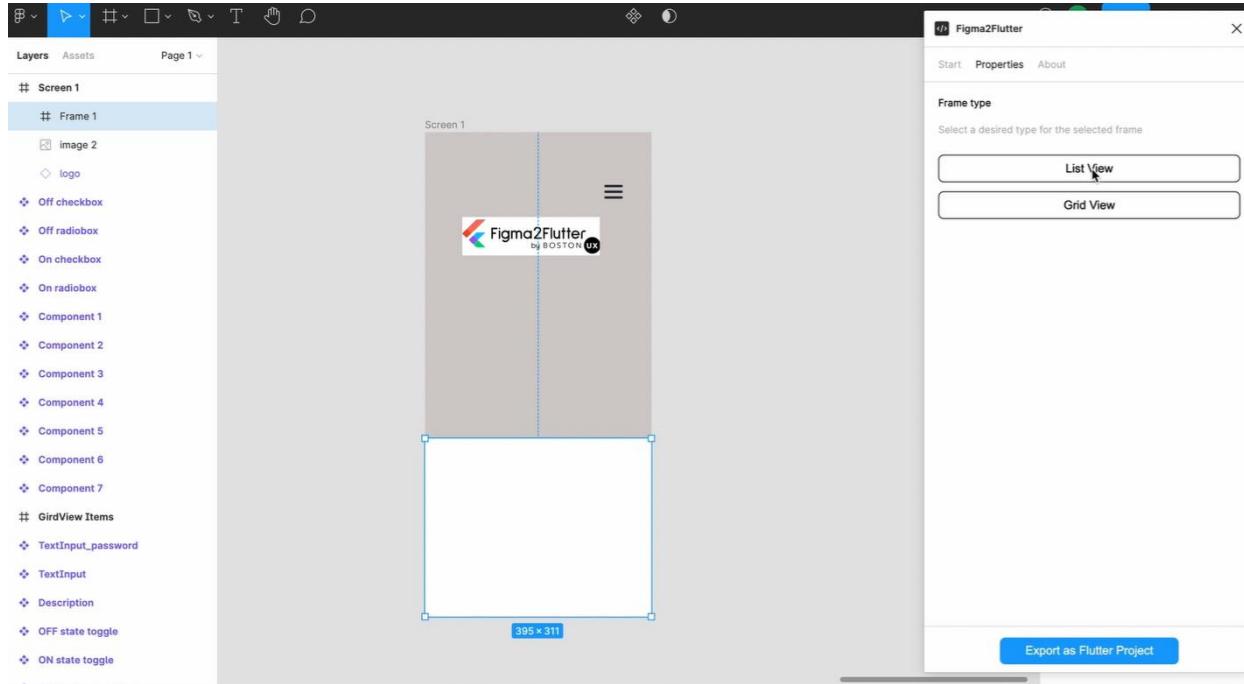
1.5 Vertical ListViews

1.5.1 Steps for ListViews

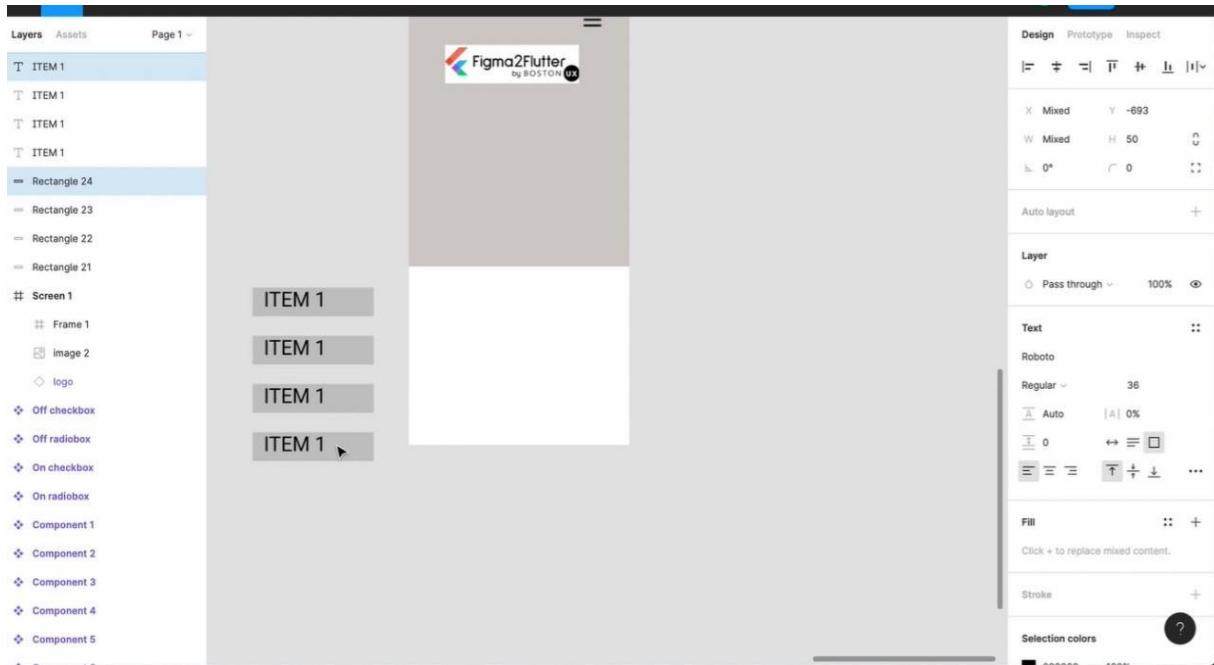
- Now let us explore how I proceeded for the part Vertical ListViews. First I draw a figma frame. This will be my main screen. I then set the background color and add some images

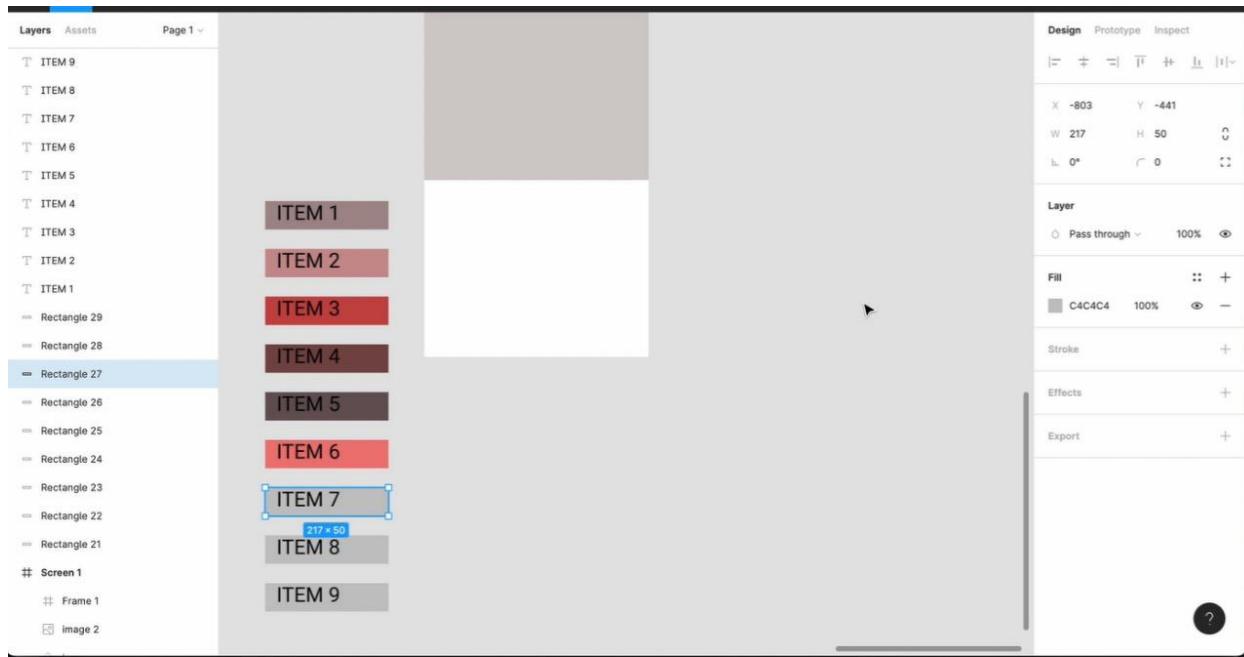


- Then, draw another frame inside the first frame. This will become a ListView field where items will be displayed as a scrollable list. I made sure that the frame is entirely within the screen frame. Otherwise, the plugin can't recognize it as a ListView field.

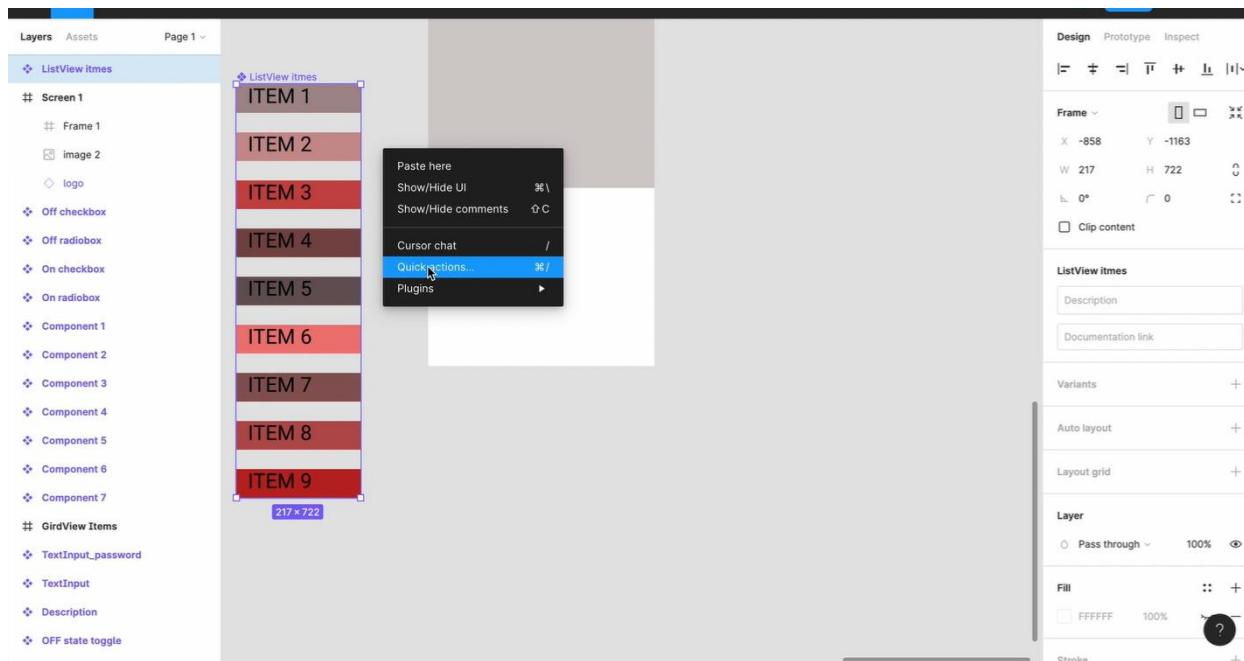


- Once I finished the past task, I created some items ; random rectangles with text and added different background colors.

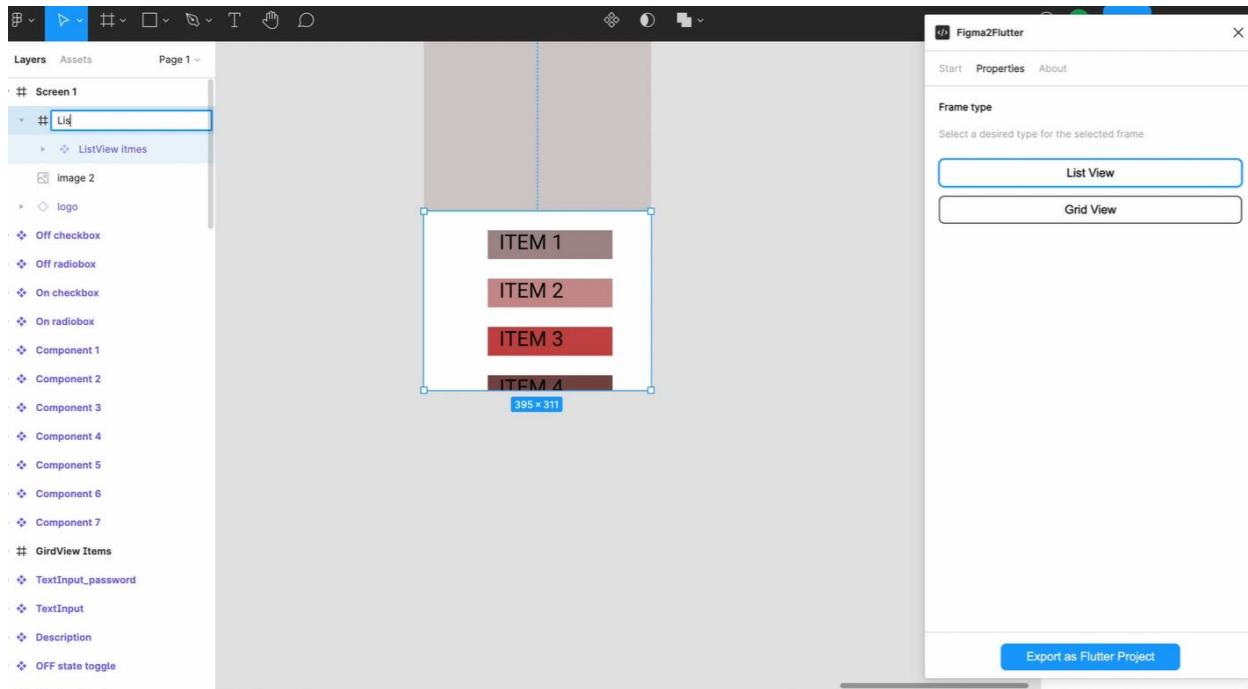




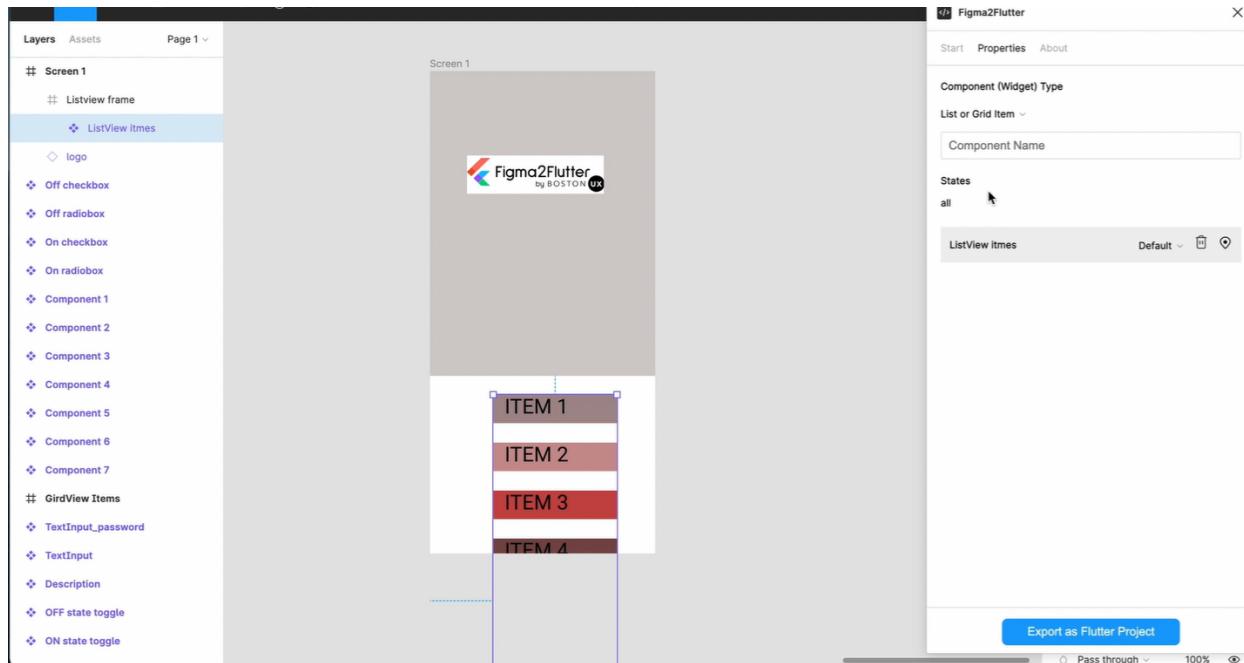
4. Here I created the components of my items and set the list name.



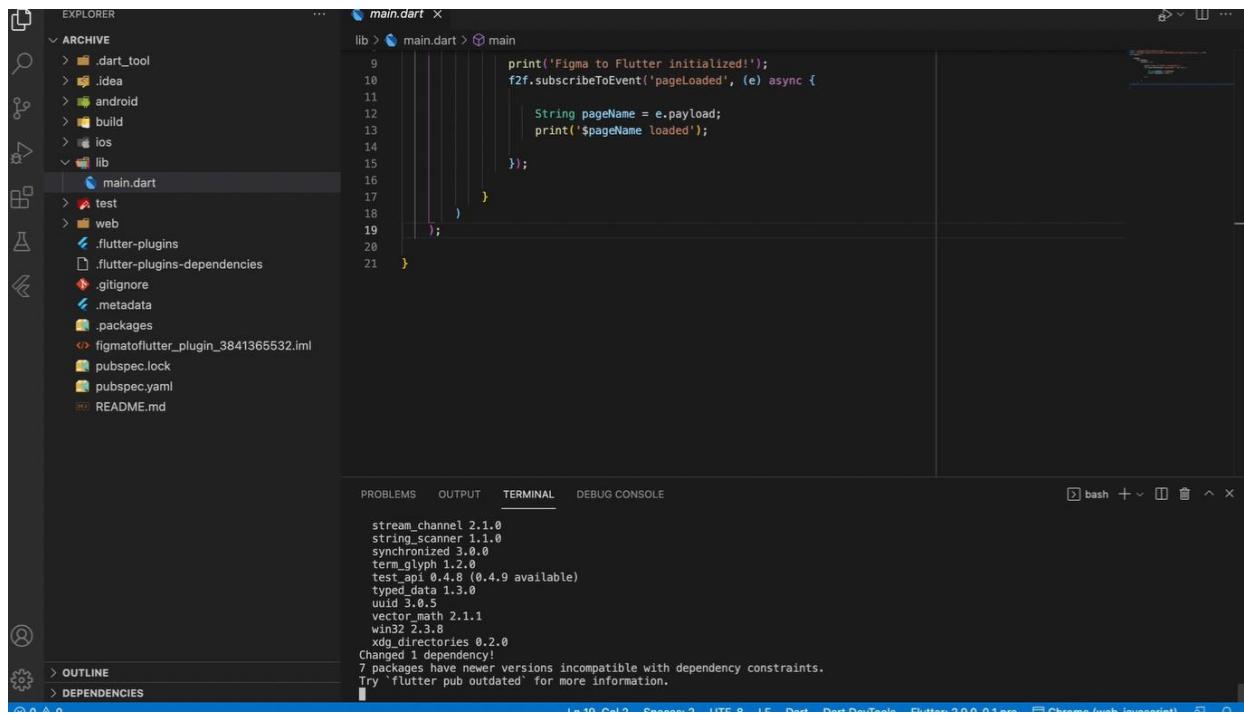
5. Next, select the newly created component, and from the plugin menu, press the Combine Selected Components as States button. From the drop-down menu, select List or Grid Item. Put our list items inside the ListView field.



6. I made sure that the selected components are exactly under the ListView field. Otherwise, the plugin can't read it as a scrollable list. Also, I made sure that my ListView frame is annotated as ListView in the plugin menu.I export the flutter project

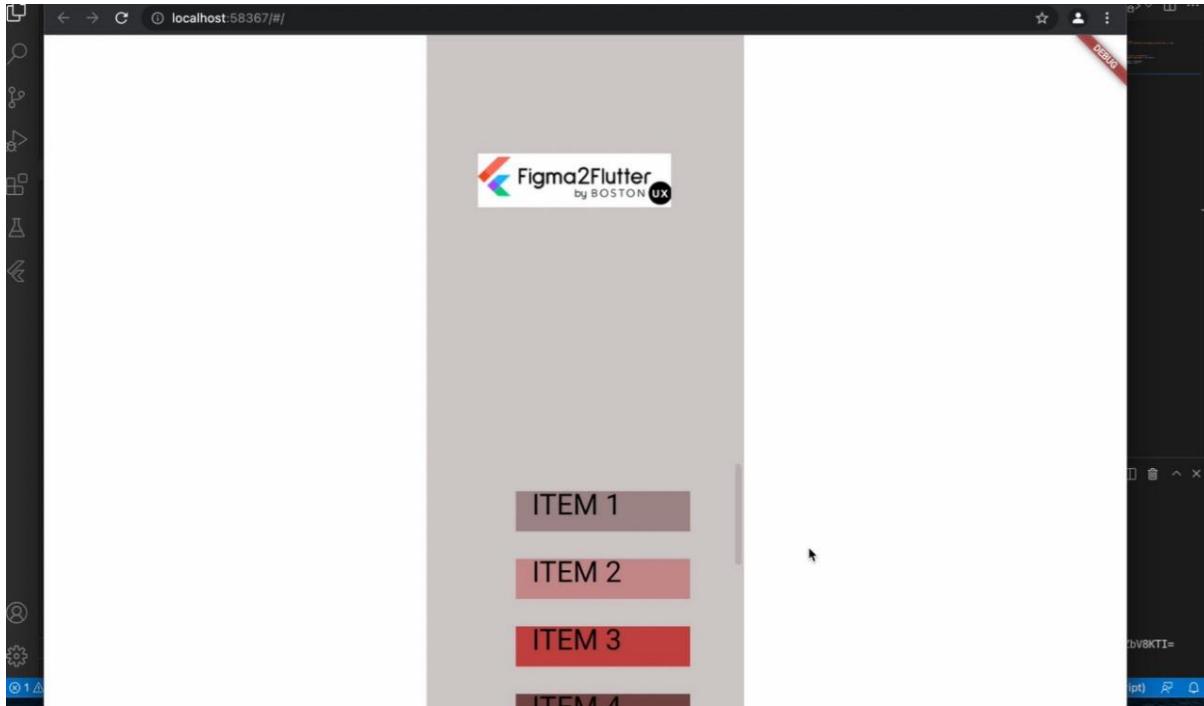


7. After exporting,i run the flutter-packages-upgrade command and wait until the dependencies are updated



1.5.2 Final output

As you can see, the type and code is exactly the same as my Figma design. I tried to scroll items up or down(vertically) and it worked perfectly!

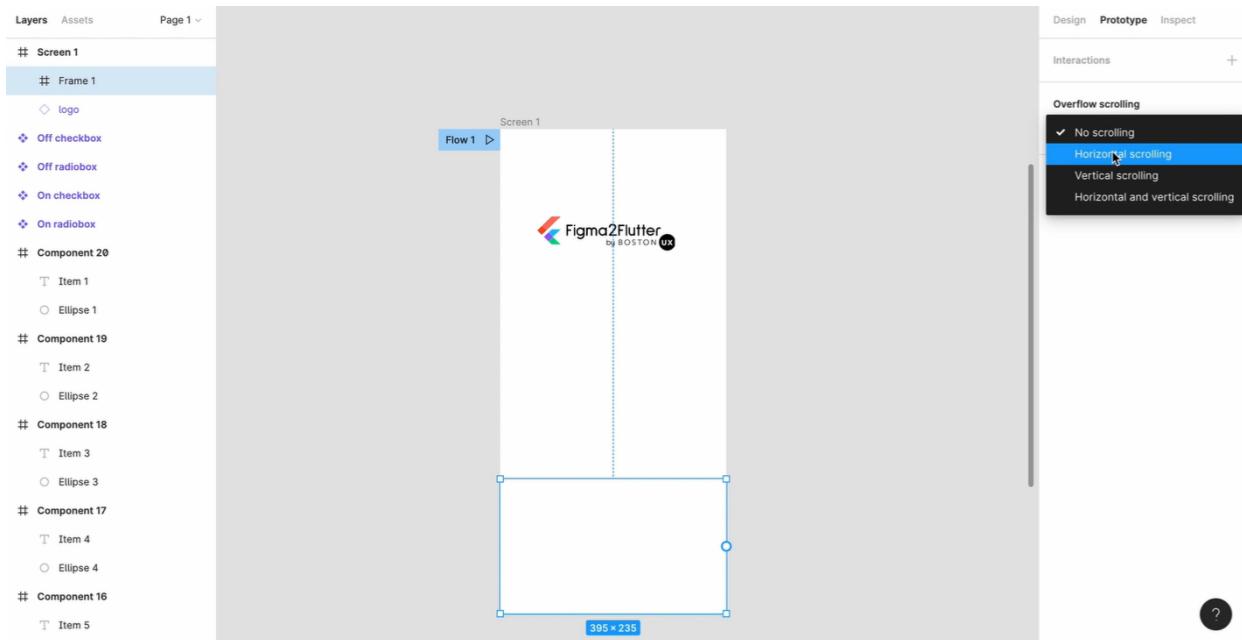


1.6 Horizontal ListView

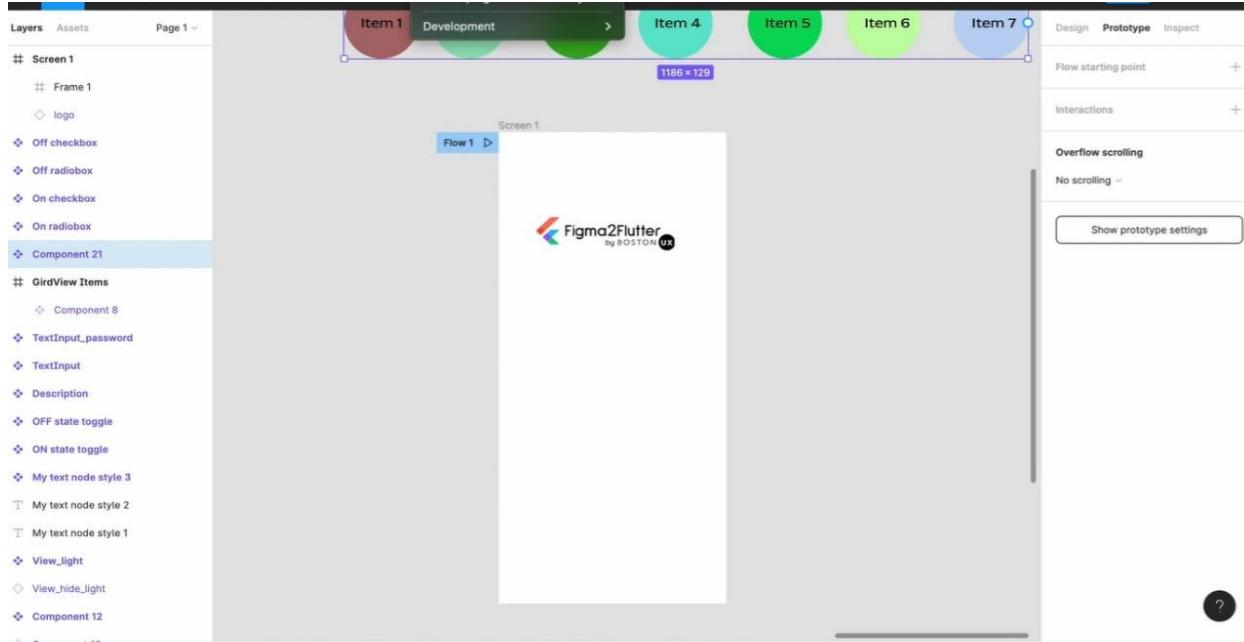
1.6.1 Steps for ListView

1. To begin with, I draw a new Figma frame .This is my main screen. Then draw another frame inside the first frame. This is my list view field where some items are displayed as a scrollable list.

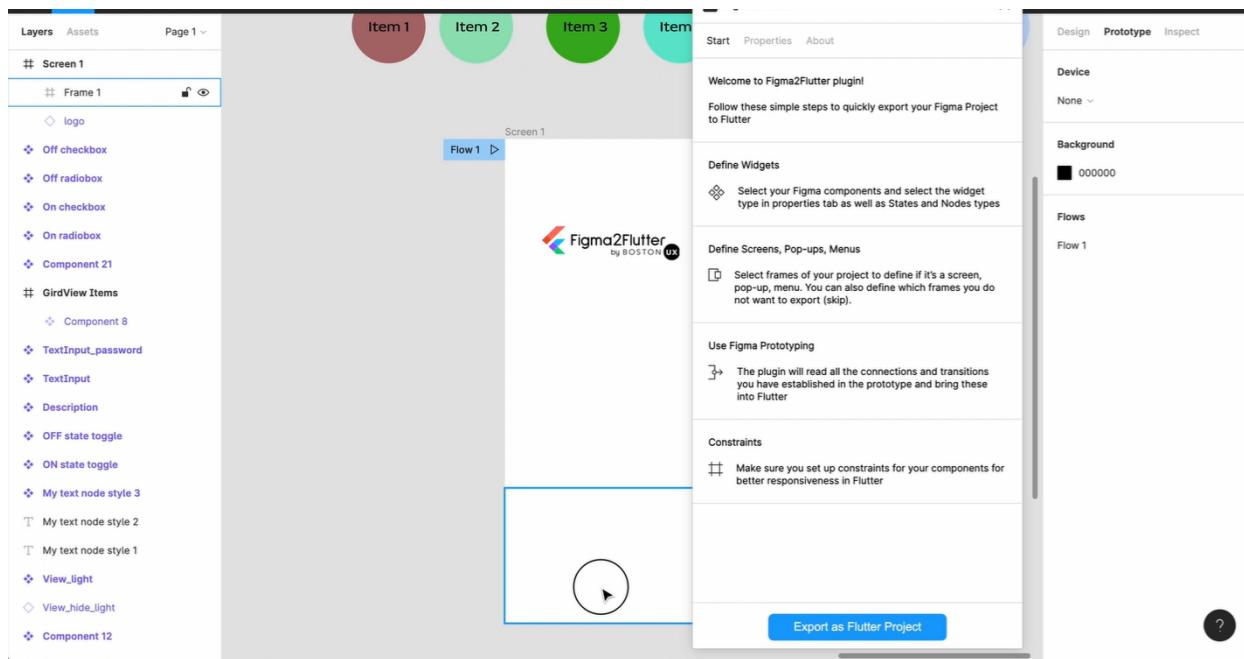
Note: It's important to ensure that your list view frame fits entirely within your original frame. Otherwise, the plugin won't recognize it as a list view field.



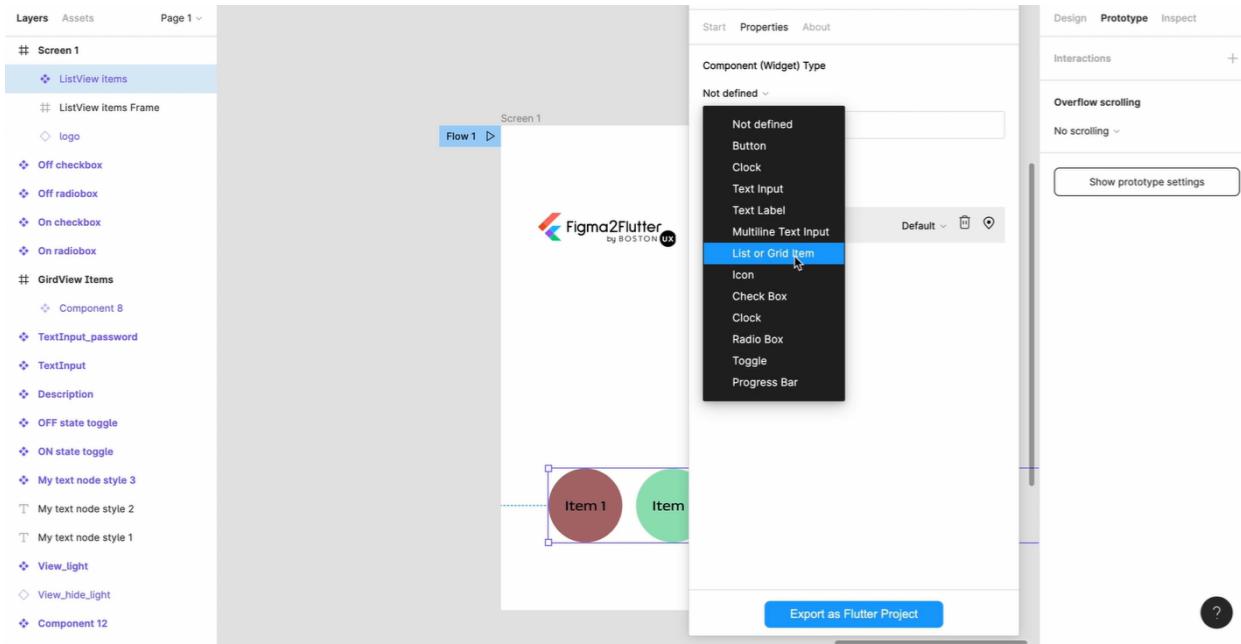
- I then create the components of my items and set the list name. Now, from the top bar, I select Plugins, Figma to Flutter.



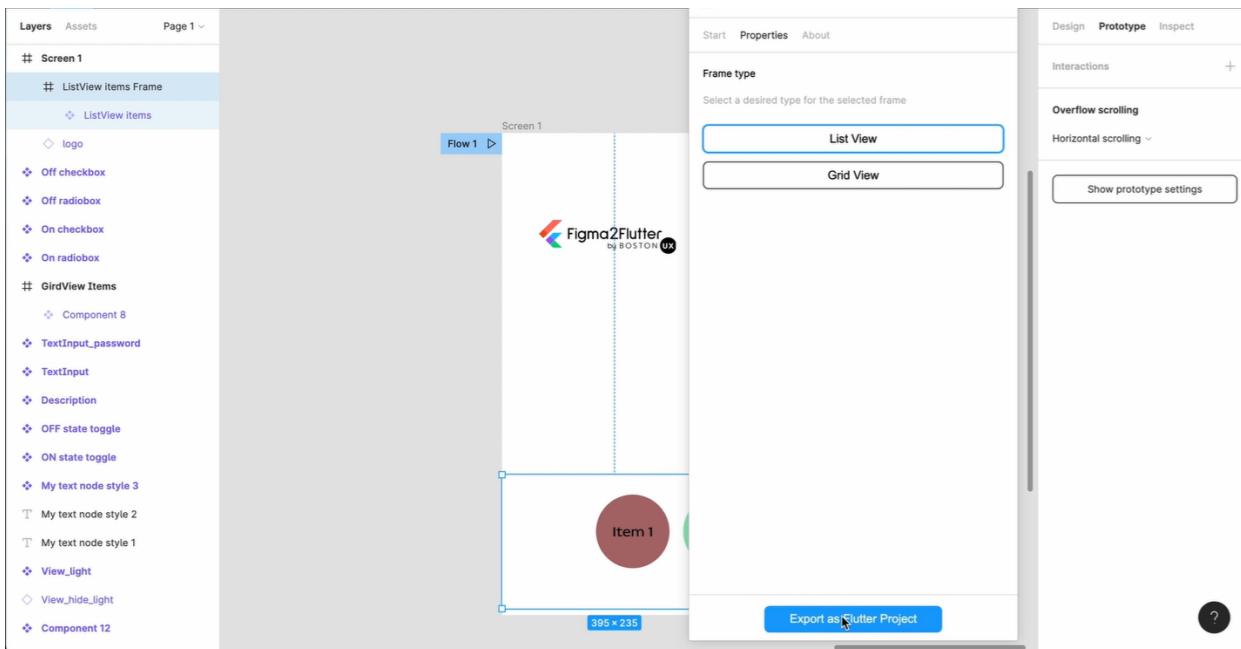
- I select my newly created component, and from the Plugin menu, I press the Combine Selected Components as States button.



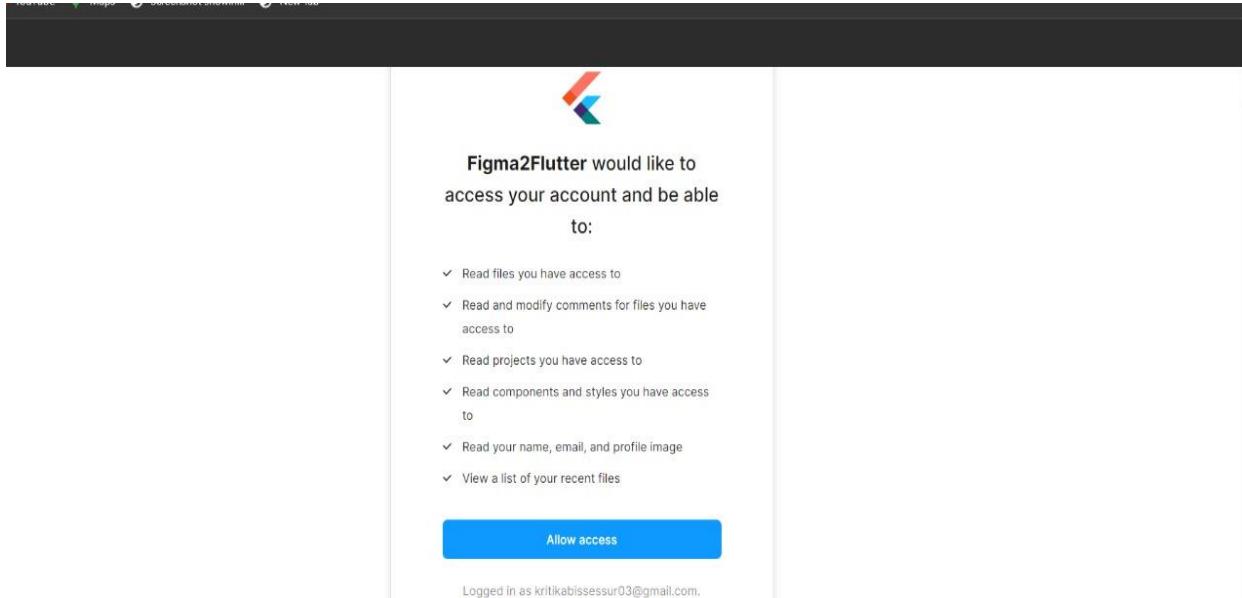
4. Next, I select List or Grid Item from the drop-down menu.



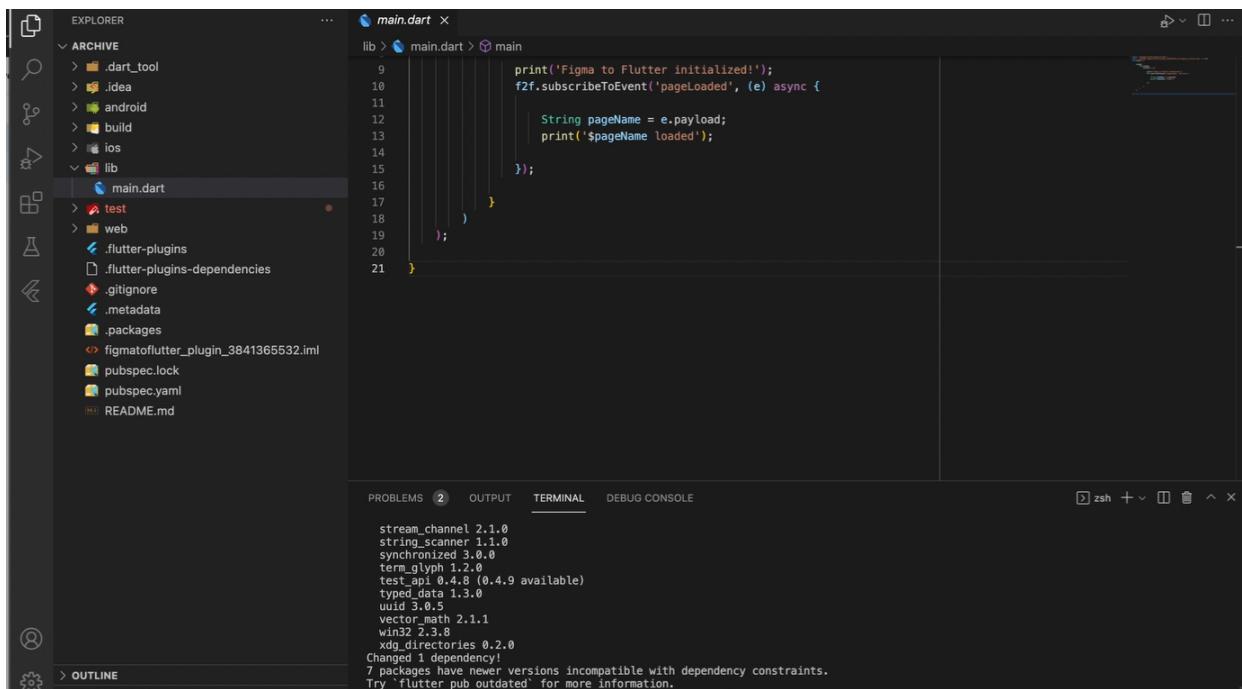
5. I put my list items inside the list view field. I make sure it is exactly within the list view field so the plugin can read it as a scrollable list. Subsequently, I press the export flutter project button.
Note: Make sure that your list view frame is annotated as List View in the Plugin menu.



6. I press the Allow Access button to let the plugin generate the image

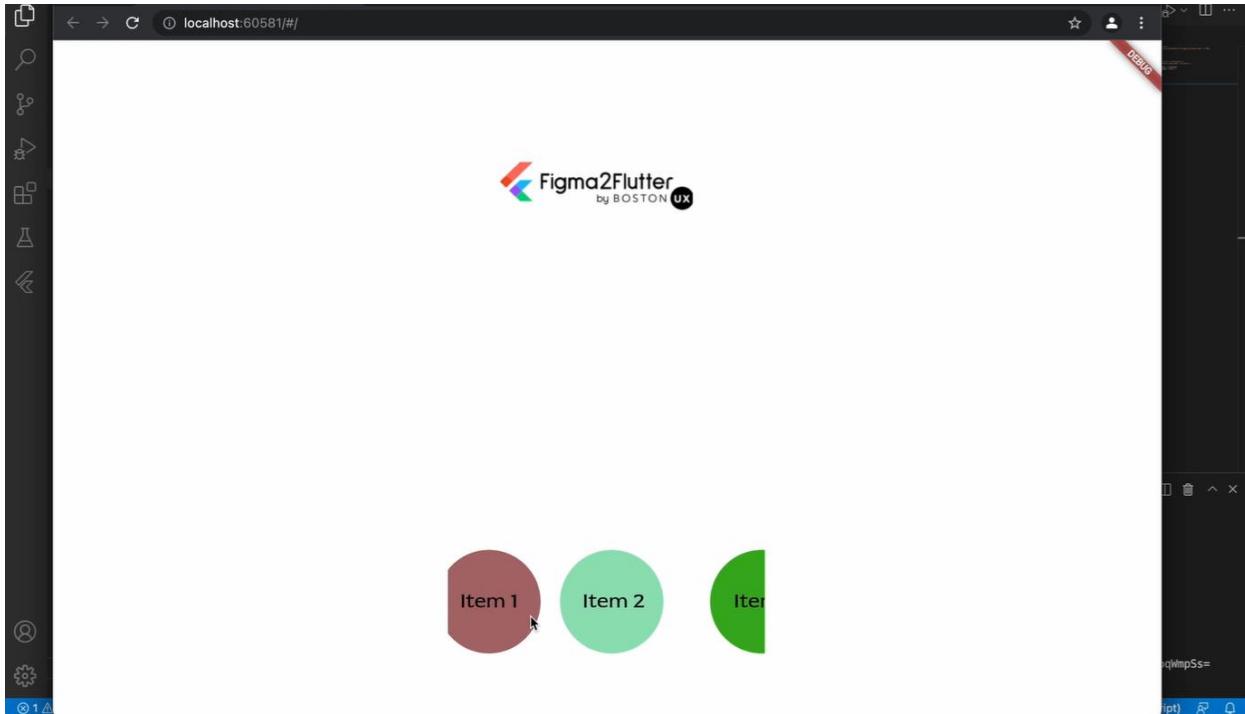


7. I just run the flutter-packages-upgrade command and wait until the dependencies are updated. Once they are, run the flutter-run command to start my project.



1.6.2 Final output

The open code is exactly as my figma design. Try to scroll items horizontally and see how it looks



2 Task 2

Task-2

Take your Flutter app from boring to beautiful

<https://codelabs.developers.google.com/codelabs/flutter-boring-to-beautiful#0>

Provide a documentation with a short description and screenshots for all the steps of the lab.

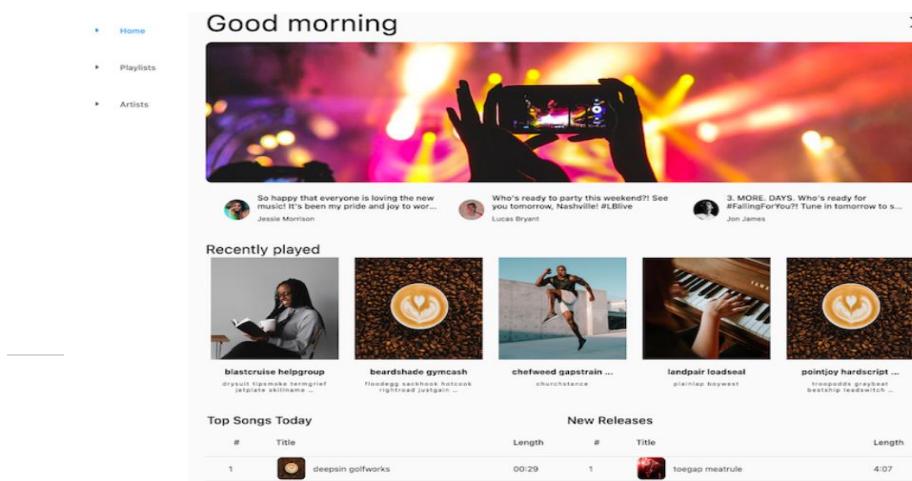
2.1 Brief Overview

In this codelab, participants will learn how to create an engaging and visually appealing Flutter application compatible across various platforms. The focus will be on crafting a user-friendly and aesthetically pleasing interface for 'MyArtist', a music player app designed to connect fans with their favorite musicians. The codelab will cover essentials like effective text design, color selection, widget customization, theme building, and the implementation of a dark mode for enhanced user comfort. Additionally, it will tackle the creation of an adaptive design that ensures the app's interface looks appealing on different screens and resolutions. Emphasis will also be placed on incorporating fluid motion within the app to create a dynamic user experience. This task aims to provide a solid foundation for building cross-platform applications that are not only functional but also visually striking.

2.2 Steps

2.2.1 Starter App

Right now when running the app, it looks like this. As from now, I'll start modifying as per the codelab



2.2.2 Modification in code

Next, I need to make some modifications. This file imports material.dart and implements a stateful widget using two classes:

The import statement makes the Material Components available.

The HomeScreenState class represents the entire page that is displayed.

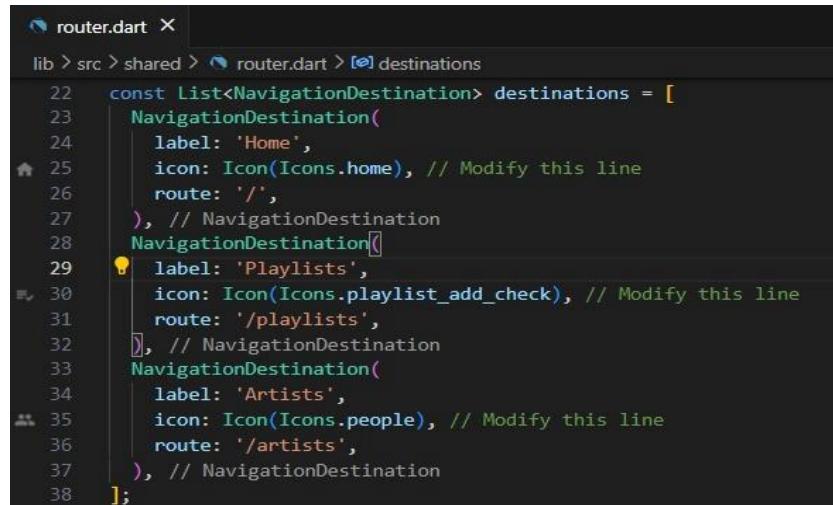
The _HomeScreenState class's build() method creates the root of the widget tree, which affects how all of the widgets in the UI are created.

```
home_screen.dart
lib > src > features > home > view > home_screen.dart > _HomeScreenState > build
1 // Copyright 2022 The Flutter Authors. All rights reserved.
2 // Use of this source code is governed by a BSD-style license that can
3 // be found in the LICENSE file.
4
5 import 'package:adaptive_components/adaptive_components.dart';
6 import 'package:flutter/material.dart';
7
8 import '../../../../../shared/classes/classes.dart';
9 import '../../../../../shared/extensions.dart';
10 import '../../../../../shared/providers/providers.dart';
11 import '../../../../../shared/views/views.dart';
12 import '../../../../../playlists/view/playlist_songs.dart';
13 import 'view.dart';
14
```

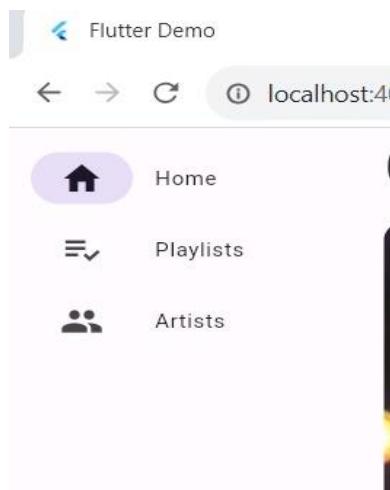
```
home_screen.dart
lib > src > features > home > view > home_screen.dart > _HomeScreenState > build
101 mainAxisAlignment: CrossAxisAlignment.start,
102 children: [
103   Padding(
104     padding: const EdgeInsets.all(2), // Modify this line
105     child: Text(
106       'Top Songs Today',
107       style: context.titleLarge,
108     ), // Text
109   ), // Padding
110   LayoutBuilder(
111     builder: (context, constraints) =>
112       PlaylistSongs(
113         playlist: newReleases,
114         constraints: constraints,
115       ), // PlaylistSongs
116     ), // LayoutBuilder
117   ],
118 ),
119 ), // Column
120 ), // Flexible
121 // Add spacer between tables
122 Flexible(
123   flex: 10,
124   child: Column(
125     mainAxisAlignment: MainAxisAlignment.start,
126     crossAxisAlignment: CrossAxisAlignment.start,
127     children: [
128       Padding(
129         padding: const EdgeInsets.all(2), // Modify this line
130         child: Text(
131           'New Releases',
132           style: context.titleLarge,
133         ), // Text
134       ), // Padding
135       LayoutBuilder(
136         builder: (context, constraints) =>
137           PlaylistSongs(
138             playlist: newReleases,
139             constraints: constraints,
140           ), // PlaylistSongs
141         ), // LayoutBuilder
142       ],
143     ), // Column
144   ), // Flexible
145 
```

2.2.3 Typography and icons

Here, I need to modify the icons shape in order to give it a more beautiful appearance and also change the text. As you can see my icons have changed to these new creative icons



```
router.dart X
lib > src > shared > router.dart > destinations
22 const List<NavigationDestination> destinations = [
23   NavigationDestination(
24     label: 'Home',
25     icon: Icon(Icons.home), // Modify this line
26     route: '/',
27   ), // NavigationDestination
28   NavigationDestination(
29     label: 'Playlists',
30     icon: Icon(Icons.playlist_add_check), // Modify this line
31     route: '/playlists',
32   ), // NavigationDestination
33   NavigationDestination(
34     label: 'Artists',
35     icon: Icon(Icons.people), // Modify this line
36     route: '/artists',
37   ), // NavigationDestination
38 ];
```



2.2.4 New package

From the command line, I pull in the google_fonts package. This also updates the pubspec file to add the fonts as an app dependency.

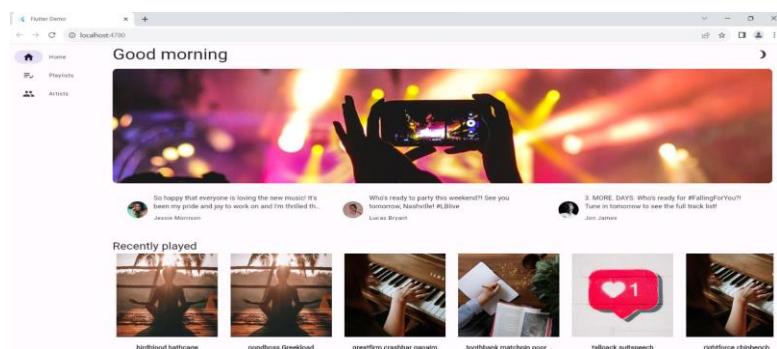
```
Application finished.
PS C:\Users\kriti\flutter-code-labs\boring_to_beautiful\step_01> flutter pub add google_fonts
Resolving dependencies...
  _fe_analyzer_shared 64.0.0 (65.0.0 available)
    analyzer 6.2.0 (6.3.0 available)
  + ffi 2.1.0
  + go_router 12.1.13 (13.0.1 available)
  + google_fonts 6.1.0
  + http 1.1.2
  js 0.6.7 (0.7.0 available)
  matcher 0.12.16 (0.12.16+1 available)
  material_color_utilities 0.5.0 (0.8.0 available)
  meta 1.10.0 (1.11.0 available)
  path 1.8.2 (1.8.3 available)
```

2.2.5 Importing the package

In lib/src/shared/extensions.dart, I import the new package and set the Montserrat TextTheme. After doing so, I notice the new NavigationRail icons along with text displayed in the Montserrat font.

```
// Add Google Fonts Package import
import 'package:google_fonts/google_fonts.dart'; // Add this line.
```

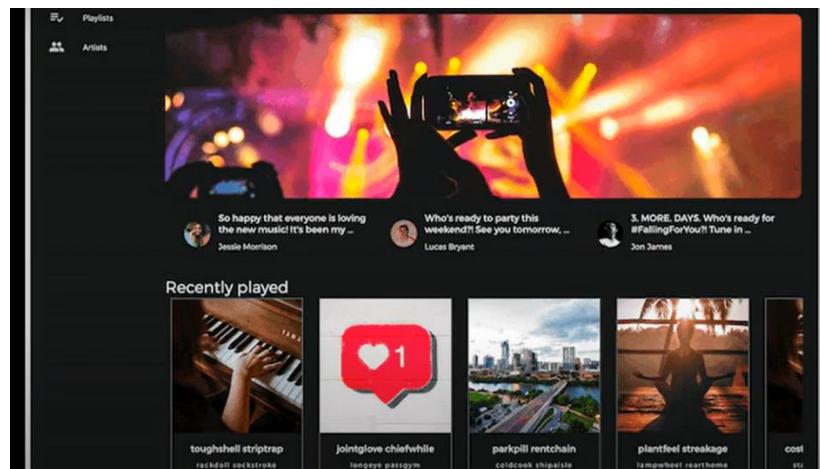
```
ThemeData get theme => theme.of(this),
TextTheme get textTheme => GoogleFonts.montserratTextTheme(theme.textTheme); // Modify this line
```



To use a particular color, access a color role on the colorScheme. I go to lib/src/shared/views/outlined_card.dart and give the OutlinedCard a border:

```
class _OutlinedCardState extends State<OutlinedCard> {
  @override
  Widget build(BuildContext context) {
    return MouseRegion(
      cursor: widget.clickable
        ? SystemMouseCursors.click
        : SystemMouseCursors.basic,
      child: Container(
        // Add from here...
        decoration: BoxDecoration(
          border: Border.all(
            color: Theme.of(context).colorScheme.outline,
            width: 1,
          ), // Border.all
        ), // BoxDecoration
        // ... To here.
      ), // Container
    ); // MouseRegion
  }
}
```

2.2.6 Dark Mode and bright Mode



2.2.7 Add adaptive design

The lib/src/shared/views/adaptive_navigation.dart file contains a navigation class where we can provide a list of destinations and content to render the body. Since I use this layout on multiple screens, there's a shared base layout to pass into each child. Navigation rails are good for desktop and large screens, but make the layout mobile friendly by showing a bottom navigation bar on mobile instead.

```
shared > views > adaptive_navigation.dart > AdaptiveNavigation
builder: (context, dimens) {
  // Tablet Layout
  if (dimens.maxWidth >= 600) { // Add this line
    return Scaffold(
      body: Row(
        children: [
          NavigationRail(
            extended: dimens.maxWidth >= 800,
            minExtendedWidth: 180,
            destinations: destinations
              .map((e) => NavigationRailDestination(
                icon: e.icon,
                label: Text(e.label),
              )) // NavigationRailDestination
              .toList(),
            selectedIndex: selectedIndex,
            onDestinationSelected: onDestinationSelected,
          ), // NavigationRail
          Expanded(child: child),
        ],
      ), // Row
    ); // Scaffold
  } // Add this line

  // Mobile Layout
  // Add from here...
  return Scaffold(
    body: child,
    bottomNavigationBar: NavigationBar(
      destinations: destinations,
      selectedIndex: selectedIndex,
      onDestinationSelected: onDestinationSelected,
    ), // NavigationBar
  ); // Scaffold
  // ... To here.
}, // LayoutBuilder
}
```

2.2.8 Responsive layout

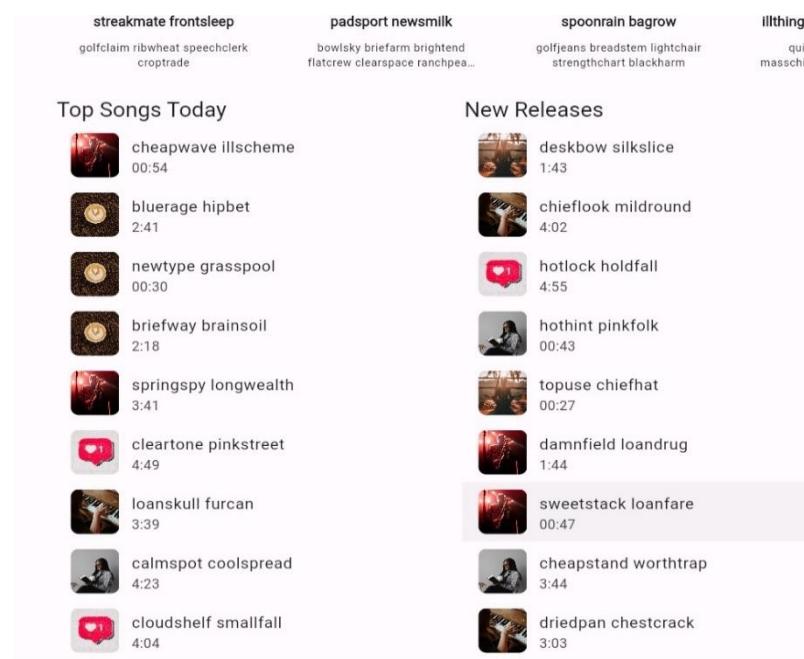
An adaptive layout needs two layouts: one for mobile, and a responsive layout for larger screens. The LayoutBuilder currently returns only a desktop layout. In lib/src/features/home/view/home_screen.dart build the mobile layout as a TabBar and TabBarView with 4 tabs.

```
builder: (context, constraints) {
  // Add from here...
  if (constraints.isMobile) {
    return DefaultTabController(
      length: 4,
      child: Scaffold(
        appBar: AppBar(
          centerTitle: false,
          title: const Text('Good morning'),
          actions: const [BrightnessToggle()],
          bottom: const TabBar(
            isScrollable: true,
            tabs: [
              Tab(text: 'Home'),
              Tab(text: 'Recently Played'),
              Tab(text: 'New Releases'),
              Tab(text: 'Top Songs'),
            ],
          ), // TabBar
        ), // AppBar
        body: LayoutBuilder(
          builder: (context, constraints) => TabBarView(
            children: [
              SingleChildScrollView(
                child: Column(
                  children: [
                    const HomeHighlight(),
                    HomeArtists(
                      artists: artists,
                      constraints: constraints,
                    ), // HomeArtists
                  ],
                ), // Column
              ), // SingleChildScrollView
              HomeRecent()
            ],
          ),
        ),
      ),
    );
  }
}
```

2.2.9 Whitespace

Whitespace is an important visual tool for my app, creating an organizational break between sections I simply wrapped a widget with a Padding object to add whitespace around that widget by increasing all of the padding values currently in lib/src/features/home/view/home_screen.dart to 35:

```
padding:  
|   padding: const EdgeInsets.all(35), // Modify this line  
|   child: Text(  
|     style: TextStyle(fontSize: 16, color: Colors.white),  
|     text: "I'm a  
|     Padding  
|     Widget!",  
|   ),  
| 
```



Section	Song	Artist	Duration
Top Songs Today	cheapwave	illscheme	00:54
	bluerage	hipbet	2:41
	newtype	grasspool	00:30
	briefway	brainsoil	2:18
	springspy	longwealth	3:41
	cleartone	pinkstreet	4:49
	loanskull	furcan	3:39
	calmspot	coolspread	4:23
	cloudshelf	smallfall	4:04
New Releases	deskbow	silkslice	1:43
	chieflook	mildround	4:02
	hotlock	holdfall	4:55
	hohint	pinkfolk	00:43
	topuse	chiefhat	00:27
	damnfield	loandrug	1:44
	sweetstack	loanfare	00:47
	cheapstand	worthtrap	3:44

2.2.10 Adding motion & animation

The `ThemeProvider` defines a `PageTransitionsTheme` with screen transition animations for mobile platforms (iOS, Android). Desktop users already get feedback from their mouse or trackpad click, so a page transition animation isn't needed.

```
final pageTransitionsTheme = const PageTransitionsTheme(
  builders: <TargetPlatform, PageTransitionsBuilder>{
    TargetPlatform.android: FadeUpwardsPageTransitionsBuilder(),
    TargetPlatform.iOS: CupertinoPageTransitionsBuilder(),
    TargetPlatform.linux: NoAnimationPageTransitionsBuilder(),
    TargetPlatform.macOS: NoAnimationPageTransitionsBuilder(),
    TargetPlatform.windows: NoAnimationPageTransitionsBuilder(),
  },
); // PageTransitionsTheme
```

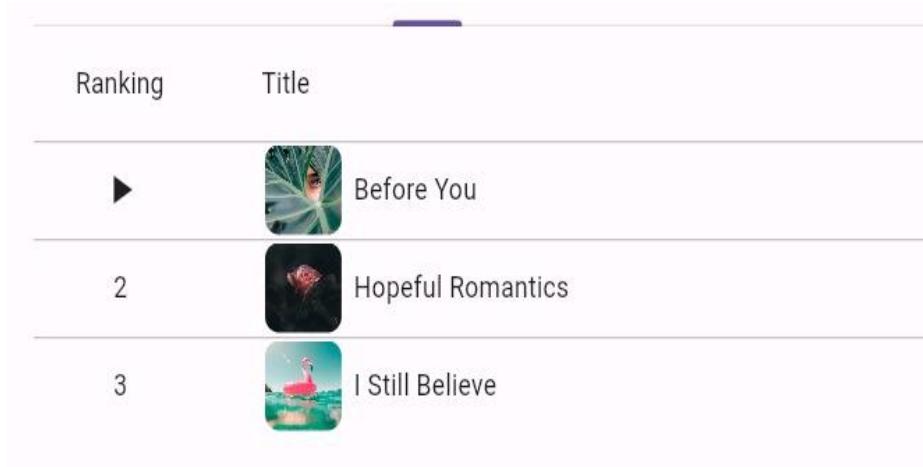
2.2.11 Adding hover states

One way to add motion to a desktop app is with hover states, where a widget changes its state (such as color, shape, or content), when the user hovers the cursor over it.

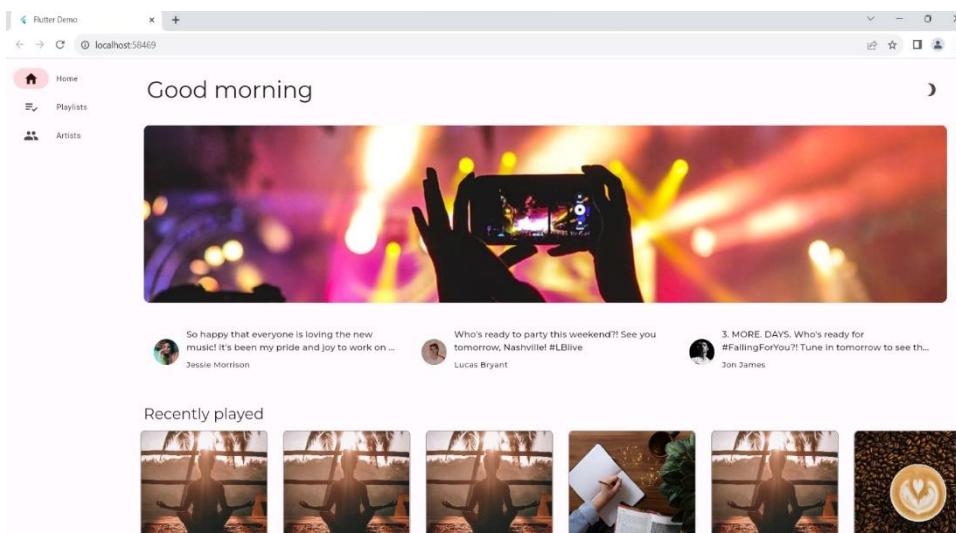
```
import 'package:flutter/material.dart';
class OutlinedCard extends StatefulWidget {
  const OutlinedCard({
    Key? key,
    required this.child,
    this.clickable = true,
  }) : super(key: key);
  final Widget child;
  final bool clickable;
  @override
  State<OutlinedCard> createState() => _OutlinedCardState();
}
class _OutlinedCardState extends State<OutlinedCard> {
  bool _hovered = false;
  @override
  Widget build(BuildContext context) {
    final borderRadius = BorderRadius.circular(_hovered ? 20 : 8);
    const animationCurve = Curves.easeInOut;
    return MouseRegion(
      onEnter: (_) {
        if (!widget.clickable) return;
        setState(() {
          _hovered = true;
        });
      },
      onExit: (_) {
        if (!widget.clickable) return;
        setState(() {
          _hovered = false;
        });
      },
      cursor: widget.clickable ? SystemMouseCursors.click : SystemMouseCursors.basic,
      child: AnimatedContainer(
        duration: kThemeAnimationDuration,
        curve: animationCurve,
        decoration: BoxDecoration(
          border: Border.all(
            color: _hovered ? Colors.red : Colors.black,
            width: 2,
          ),
        ),
        child: widget.child,
      ),
    );
  }
}
```

2.2.12 Play button

When I hot reload the app and then hover the cursor over the song number on either the Top Songs Today or the New Releases playlist.



2.3 Final Output



3 Task 3

1. <https://www.loginradius.com/blog/engineering/guest-post/authenticating-flutter-apps/>
2. <https://auth0.com/blog/get-started-with-flutter-authentication/>
3. <https://firebase.google.com/docs/auth/flutter/start>
4. <https://supabase.com/blog/flutter-authentication>
5. <https://wso2.com/asgardeo/docs/tutorials/auth-users-into-flutter-apps/#step-2-register-your-application-in-asgardeo>
6. <https://blog.openreplay.com/fingerprint-authentication-for-flutter-applications/>
7. <https://blog.codemagic.io/flutter-local-authentication-using-biometrics/>
8. <https://medium.flutterdevs.com/biometric-authentication-in-flutter-application-85a474507a46>

9. Authentication using a basic login form.

Provide a documentation with a short description and screenshots for all the steps of the

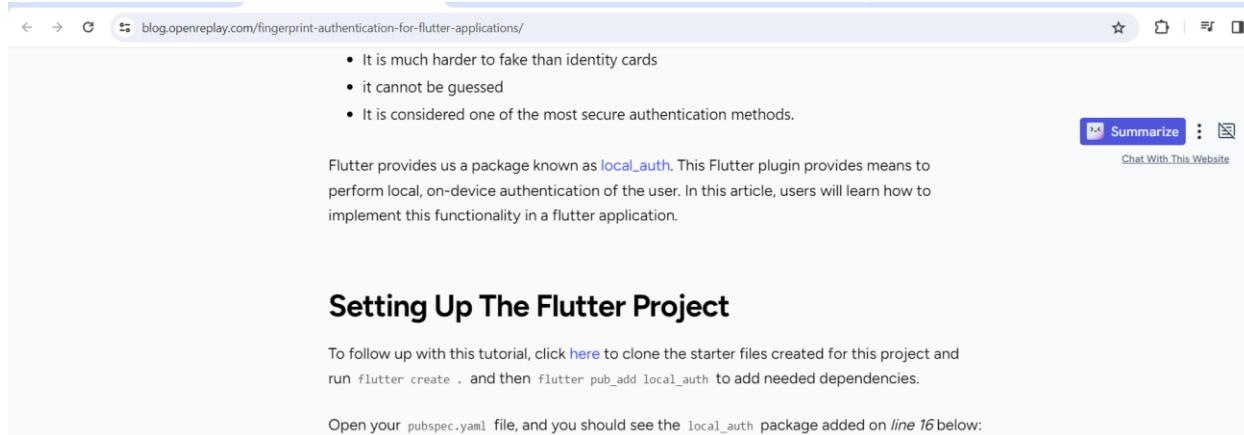
3.1 Brief Overview

(Part chosen: 6)

I have selected the article titled "Fingerprint Authentication for Flutter Applications" from <https://blog.openreplay.com/fingerprint-authentication-for-flutter-applications/> due to its comprehensive exploration of bolstering security in Flutter apps through biometric authentication. The content offers valuable perspectives on the practical implementation of fingerprint authentication, a robust and user-centric security measure. Given the prevailing emphasis on security in app development, adopting insights from this article aligns with current industry standards, contributing to a secure and seamless user experience within my Flutter applications.

3.2 Steps

- To initiate my journey with this tutorial, I'll click here to clone the provided starter files for my project. I'll execute 'flutter create .' to establish the groundwork and then add the necessary dependencies with a straightforward 'flutter pub' add 'local_auth'.



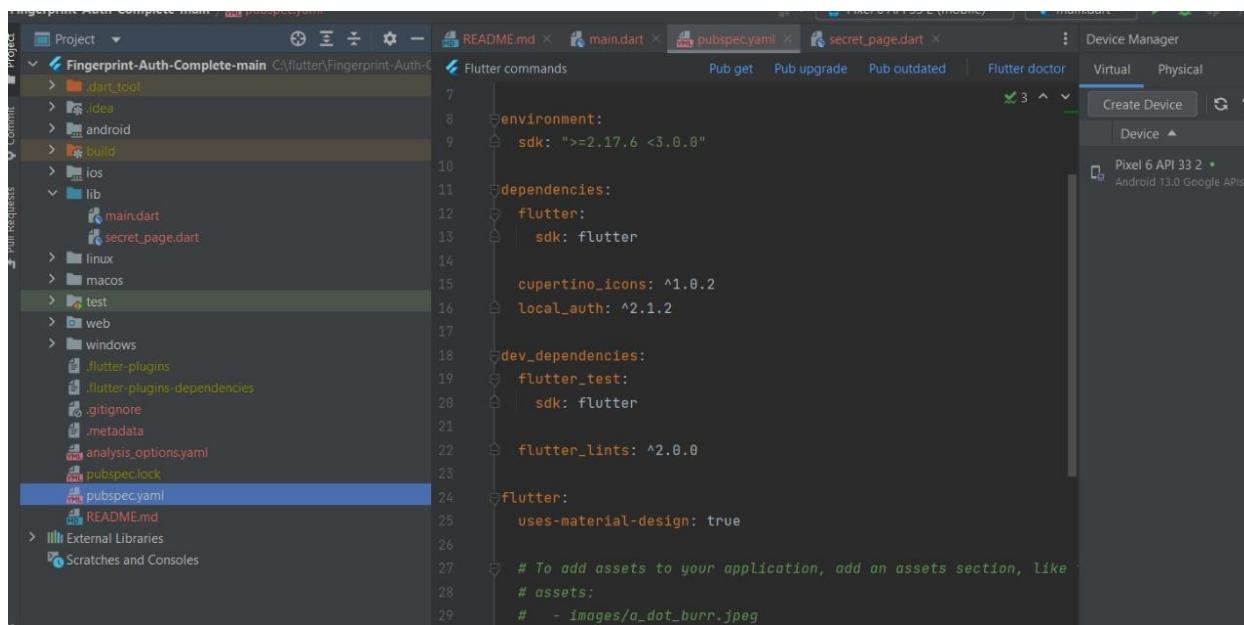
The screenshot shows a web browser displaying a blog post from blog.openreplay.com/fingerprint-authentication-for-flutter-applications/. The post highlights the security benefits of biometric authentication and introduces the `local_auth` package for Flutter. It includes instructions for cloning the project and adding dependencies via `flutter pub add local_auth`.

Setting Up The Flutter Project

To follow up with this tutorial, click [here](#) to clone the starter files created for this project and run `flutter create .` and then `flutter pub add local_auth` to add needed dependencies.

Open your `pubspec.yaml` file, and you should see the `local_auth` package added on [line 16](#) below:

- Next, I'll open my '`pubspec.yaml`' file. I'll notice the seamless integration of the '`local_auth`' package on line 16, ensuring my project is equipped with the fundamental capabilities of biometric authentication.



The screenshot shows an IDE (Android Studio) with the project structure on the left and the `pubspec.yaml` file content on the right. The `pubspec.yaml` file contains the following code:

```

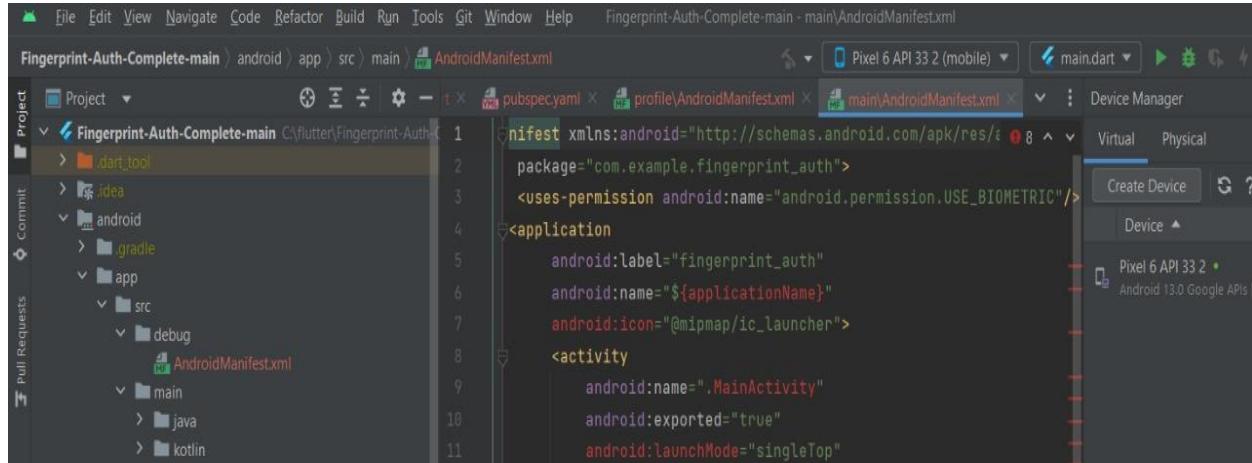
environment:
  sdk: ">=2.17.6 <3.0.0"
dependencies:
  flutter:
    sdk: flutter
  cupertino_icons: ^1.0.2
  local_auth: ^2.1.2
dev_dependencies:
  flutter_test:
    sdk: flutter
  flutter_lints: ^2.0.0
flutter:
  uses-material-design: true
# To add assets to your application, add an assets section, like:
# assets:
#   - images/a_dot_burr.jpeg

```

3. I'll proceed to the AndroidManifest.xml file located in android > app > src > main. Adding the code snippet below:

xml

<uses-permission android:name="android.permission.USE_BIOMETRIC"/>. This code signifies a crucial system permission that must be granted by the user for the seamless operation of the app.



```
1<manifest xmlns:android="http://schemas.android.com/apk/res/android"
2    package="com.example.fingerprint_auth">
3        <uses-permission android:name="android.permission.USE_BIOMETRIC"/>
4        <application
5            android:label="fingerprint_auth"
6            android:name="${applicationName}"
7            android:icon="@mipmap/ic_launcher">
8                <activity
9                    android:name=".MainActivity"
10                   android:exported="true"
11                   android:launchMode="singleTop"/>
```

4. Moving forward, I'll navigate to android > app > src > main > kotlin and open the MainActivity.kt file. I'll implement the following changes:

kotlin

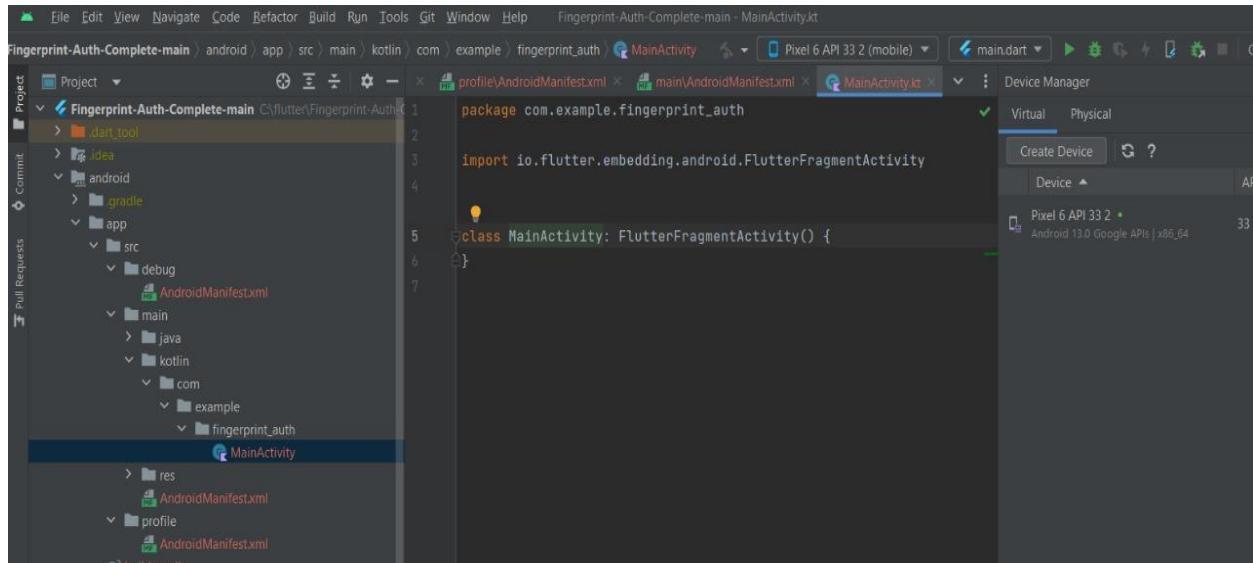
```
package com.example.fingerprint_auth

import io.flutter.embedding.android.FlutterFragmentActivity

class MainActivity: FlutterFragmentActivity() {

}
```

- ❖ This modification is essential because local_auth necessitates the use of a FragmentActivity instead of an Activity for optimal functionality.
- ❖ These meticulous steps contribute to the integration of biometric authentication seamlessly into my Flutter app, ensuring a secure and efficient user experience.

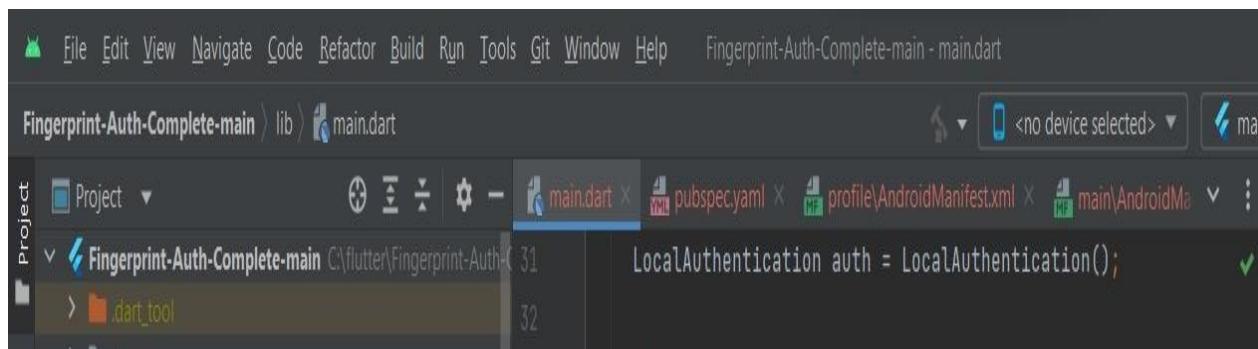


5. With the configuration completed, let's embark on creating our functionalities. I begin by importing the essential local_auth packages:

dart

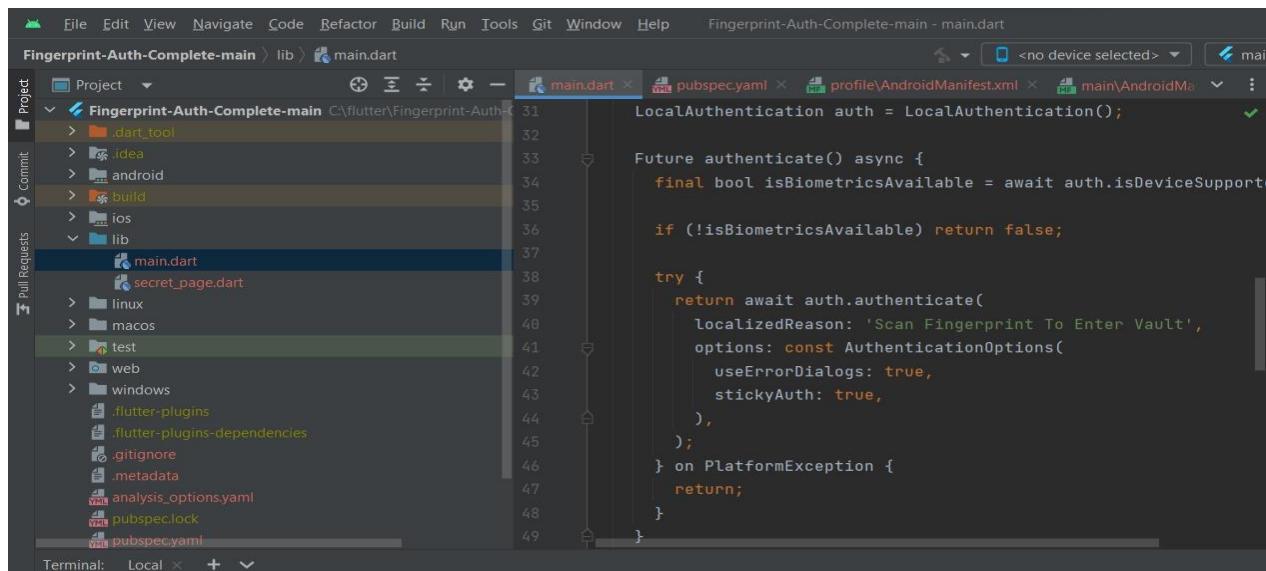
```
import 'package:local_auth/local_auth.dart';
```

6. Now, I instantiate my LocalAuthentication class:



```
File Edit View Navigate Code Refactor Build Run Tools Git Window Help Fingerprint-Auth-Complete-main - main.dart
Fingerprint-Auth-Complete-main > lib > main.dart
Project Fingerprint-Auth-Complete-main C:\flutter\Fingerprint-Auth-C 31
> .dart_tool 32
LocalAuthentication auth = LocalAuthentication(); 33
  
```

7. With that over, I create an asynchronous function to perform authentication when I click the button .



```
File Edit View Navigate Code Refactor Build Run Tools Git Window Help Fingerprint-Auth-Complete-main - main.dart
Fingerprint-Auth-Complete-main > lib > main.dart
Project Fingerprint-Auth-Complete-main C:\flutter\Fingerprint-Auth-C 31
> .dart_tool 32
> .idea 33
> android 34
> build 35
> ios 36
> lib 37
  > main.dart 38
  > secret_page.dart 39
> test 40
> web 41
> windows 42
> .flutter-plugins 43
> .flutter-plugins-dependencies 44
> .gitignore 45
> .metadata 46
> analysis_options.yaml 47
> pubspec.lock 48
> pubspec.yaml 49
LocalAuthentication auth = LocalAuthentication();
Future authenticate() async {
  final bool isBiometricsAvailable = await auth.isDeviceSupport
  if (!isBiometricsAvailable) return false;
  try {
    return await auth.authenticate(
      localizedReason: 'Scan Fingerprint To Enter Vault',
      options: const AuthenticationOptions(
        useErrorDialogs: true,
        stickyAuth: true,
      ),
    );
  } on PlatformException {
    return;
  }
}
  
```



8. The onPressed functionality checks authentication asynchronously. If successful, it smoothly navigates to SecretPage; otherwise, it presents a placeholder Container for a seamless user experience.

The screenshot shows the Android Studio interface with the project 'Fingerprint-Auth-Complete-main' open. The 'lib/main.dart' file is selected and displayed in the code editor. The code implements a check for authentication using an ElevatedButton's onPressed callback. If authentication is successful, it pushes a MaterialPageRoute to a 'SecretPage'. Otherwise, it displays a placeholder Container.

```
return Scaffold(
  appBar: AppBar(
    title: const Text('Fingerprint Auth'),
  ),
  body: Center(
    child: ElevatedButton(
      onPressed: () async {
        bool isAuthenticated = await authenticate();

        if (isAuthenticated) {
          Navigator.push(
            context,
            MaterialPageRoute(
              builder: (context) {
                return const SecretPage();
              },
            ),
          );
        }
      },
    ),
  ),
);
```

9. I made a modification to the color scheme, transitioning it from blue to purple. This change was a deliberate design choice aimed at infusing creativity into the visual elements of the application while maintaining a professional and polished appearance.(line 19)

The screenshot shows the Android Studio interface with the project 'Fingerprint-Auth-Complete-main' open. The 'lib/main.dart' file is selected and displayed in the code editor. A line of code at line 19 is highlighted with a purple rectangle, indicating the modification made to the theme data.

```
class MyApp extends StatelessWidget {
  const MyApp({Key? key}) : super(key: key);

  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'Fingerprint Authentication',
      debugShowCheckedModeBanner: false,
      theme: ThemeData(
        primarySwatch: Colors.purple,
      ),
      home: const MyHomePage(),
    );
}
```



10. Upon executing 'flutter run', the results are now visible.

The screenshot shows the Android Studio interface with the project 'Fingerprint-Auth-Complete-main'. The main.dart file is open in the code editor. The terminal at the bottom shows the command 'flutter run' being executed, followed by logs indicating the build process and the app running on a 'Pixel 6 API 33 2 (mobile)' device. The device screen displays a simple UI with a button labeled 'Click me'.

```
PS C:\flutter\Fingerprint-Auth-Complete-main> flutter run
Using hardware rendering with device sdk gphone x86_64. If you notice graphics artifacts, consider enabling software rendering with "--enable-software-rendering".
Launching lib/main.dart on sdk gphone x86_64 in debug mode...
Running Gradle task 'assembleDebug'...                                17.0s
v Built build\app\outputs\flutter-apk\app-debug.apk.
Syncing files to device sdk gphone x86_64...                          813ms

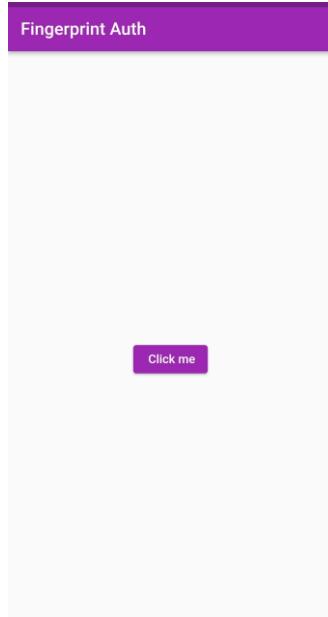
Flutter run key commands.
r Hot reload.
R Hot restart.
```

11. Various connected devices are visible, I utilized my mobile device (SM A705FN) to observe the app in action. The results are displayed below.

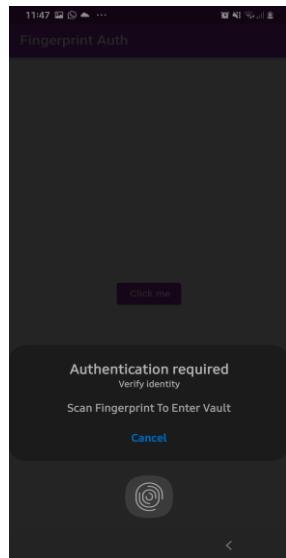
The screenshot shows the Android Studio interface with the project 'Fingerprint-Auth-Complete-main'. The terminal at the bottom lists the connected devices: SM A705FN (mobile), emulator-5554 (mobile), windows (emulator), and windows (physical). The device manager panel on the right shows the selected device as 'Pixel 6 API 33 2 (mobile)'.

```
PS C:\flutter\Fingerprint-Auth-Complete-main> flutter run
Connected devices:
SM A705FN (mobile)          * R58MB21057B    * android-arm64  * Android 9 (API 28)
emulator-5554 (mobile)        * emulator-5554   * android-x64   * Android 13 (API 33) (emulator)
windows (emulator)           * windows       * windows-x64  * Microsoft Windows [Version 10.0.22621.2428]
windows (physical)           * edge          * web-javascript * Microsoft Edge 119.0.2151.44
[1]: SM A705FN (R58MB21057B)
[2]: emulator-5554 (Android 13 | API 33)
[3]: windows (Windows)
```

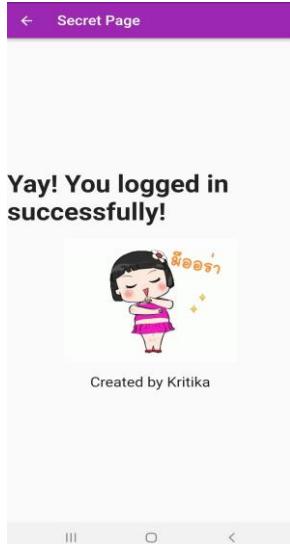
3.3 Final Output



I am asked to scan my fingerprint here



As you can see I added a GIF when the right fingerprint is entered, and it says that you logged in successfully.



4 Task 4

Task-4

Use ChatGPT to create a Flutter Mobile App

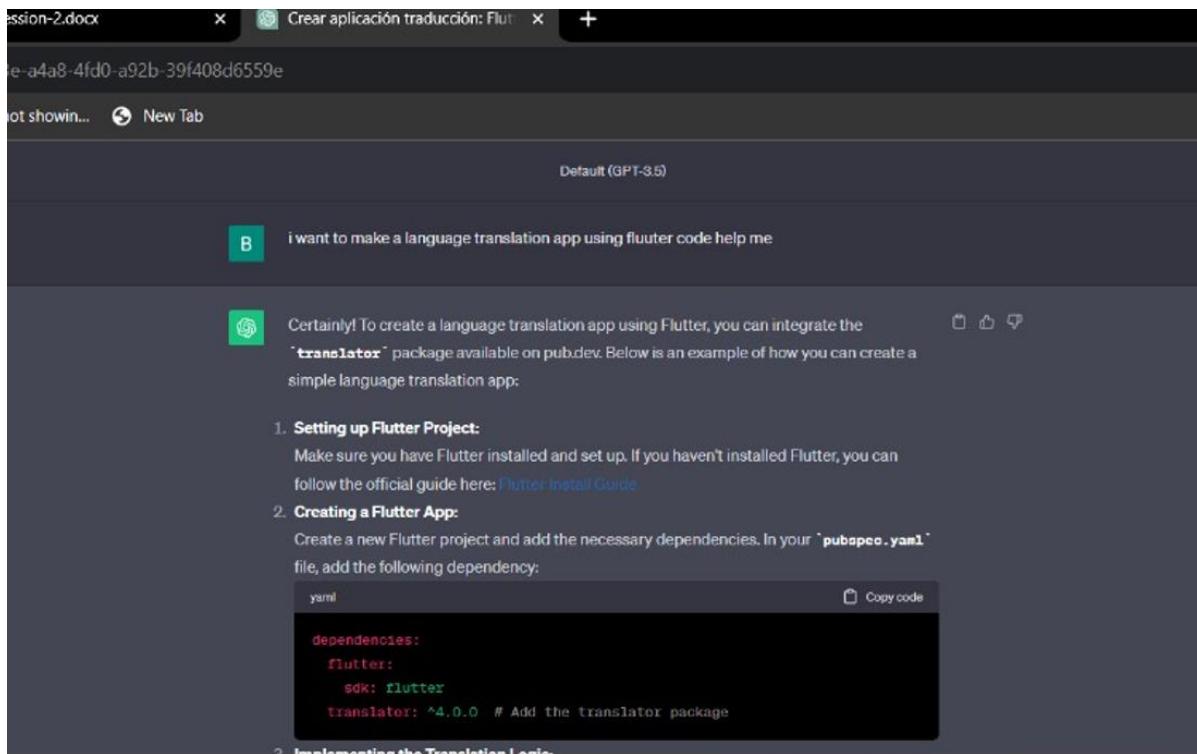
Provide a documentation with a short description and screenshots for all the steps of the lab.

4.1 Brief overview

The goal of this project was to create an application that allows users to easily translate text between different languages. This type of cross-lingual communication tool has many practical applications for international business, education and travel.

4.2 Steps

1. Commencing the development process, I engaged a conversation with a language model AI for comprehensive guidance, providing detailed specifications. This prompted the AI to furnish the essential dependencies required for the envisioned translation application. Subsequently, I was directed to include these dependencies in the pubsec.yaml file, a pivotal step in setting up the development environment.



A screenshot of a web browser window titled "Crear aplicación traducción: Flut". The browser has two tabs: "session-2.docx" and "Crear aplicación traducción: Flut". The main content area shows a conversation with ChatGPT:

B i want to make a language translation app using flutter code help me

G Certainly! To create a language translation app using Flutter, you can integrate the "translator" package available on pub.dev. Below is an example of how you can create a simple language translation app:

1. **Setting up Flutter Project:**
Make sure you have Flutter installed and set up. If you haven't installed Flutter, you can follow the official guide here: [Flutter Install Guide](#)
2. **Creating a Flutter App:**
Create a new Flutter project and add the necessary dependencies. In your `pubspec.yaml` file, add the following dependency:

```

dependencies:
  flutter:
    sdk: flutter
  translator: ^4.0.0 # Add the translator package
  
```

The browser also shows a "Copy code" button next to the code snippet.

2. Incorporating translator packages in a Flutter application empowers the seamless integration of multilingual capabilities, enabling users to effortlessly translate text between different languages. By leveraging these packages, developers can enhance user experiences by providing functionalities to input text, select source and target languages, and receive accurate translations. This integration facilitates the creation of user-friendly and globally accessible applications, catering to diverse linguistic preferences and fostering inclusive interactions.

3. Implementing the Translation Logic

Here is an example of how you can use the `'translator'` package to perform translations in your Flutter app:

```

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

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

class MyApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'Language Translator',
      theme: ThemeData(
        primarySwatch: Colors.blue,
      ),
      home: MyHomePage(),
    );
  }
}

class MyHomePage extends StatefulWidget {
  @override
  _MyHomePageState createState() => _MyHomePageState();
}

class _MyHomePageState extends State<MyHomePage> {
  final translator = GoogleTranslator();

  String _inputText = '';
  String _translatedText = '';

  void _translateText() {
    translator.translate(_inputText, to: 'es').then((result) {
      setState(() {
        _translatedText = result.text;
      });
    }).catchError((error) {
      print("Translation error: $error");
    });
  }

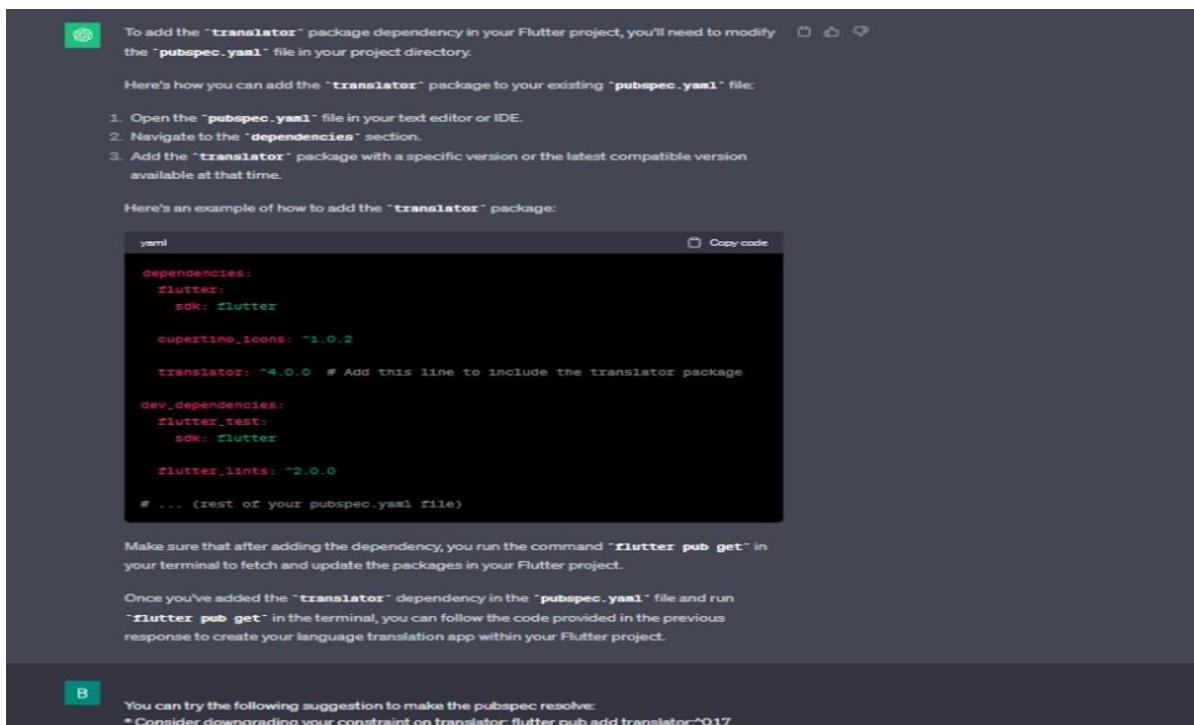
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: Text('Language Translator'),
      ),
      body: Padding(
        padding: EdgeInsets.all(16),
        child: Column(
          mainAxisAlignment: MainAxisAlignment.spaceEvenly,
          children: [
            TextField(
              decoration: InputDecoration(labelText: 'Enter text to translate'),
              onChanged: (text) {
                _inputText = text;
              },
            ),
            ElevatedButton(
              onPressed: _translateText,
              child: Text('Translate'),
            ),
            Text(
              _translatedText,
              style: TextStyle(fontSize: 18),
            ),
          ],
        ),
      ),
    );
  }
}

```

[Copy code](#)

[Regenerate](#)

3.



To add the `"translator"` package dependency in your Flutter project, you'll need to modify the `pubspec.yaml` file in your project directory.

Here's how you can add the `"translator"` package to your existing `pubspec.yaml` file:

1. Open the `pubspec.yaml` file in your text editor or IDE.
2. Navigate to the `dependencies` section.
3. Add the `"translator"` package with a specific version or the latest compatible version available at that time.

Here's an example of how to add the `"translator"` package:

```
yaml
dependencies:
  flutter:
    sdk: flutter

cupertino_icons: ^1.0.2

translator: ^4.0.0 # Add this line to include the translator package

dev_dependencies:
  flutter_test:
    sdk: flutter

flutter_lints: ^2.0.0

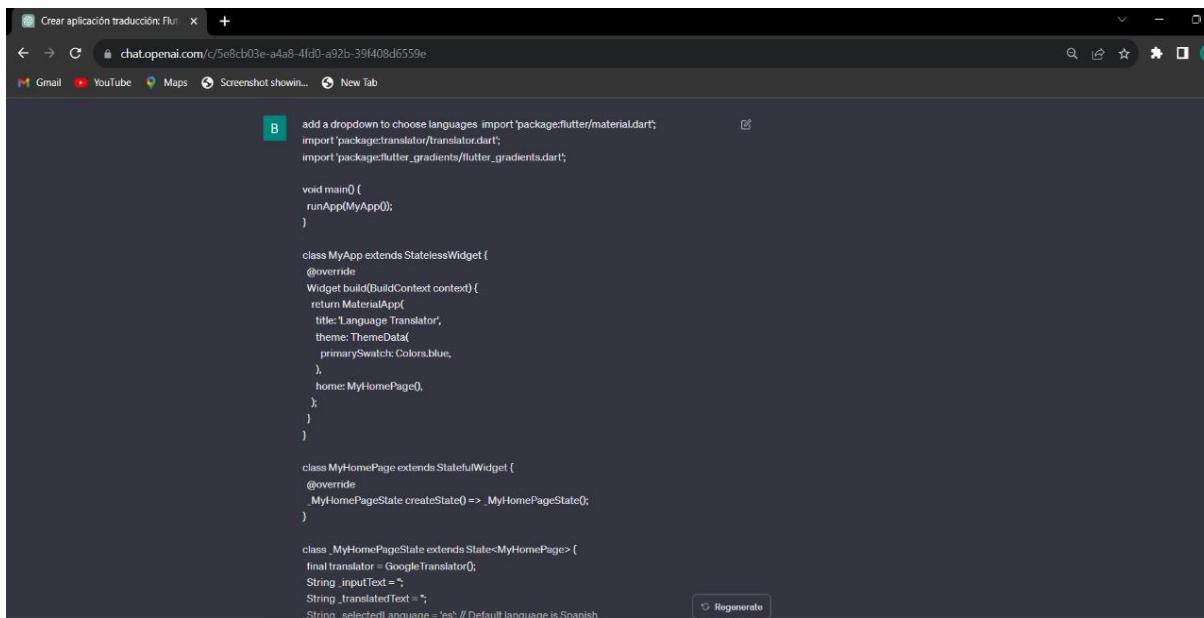
# ... (rest of your pubspec.yaml file)
```

Make sure that after adding the dependency, you run the command `flutter pub get` in your terminal to fetch and update the packages in your Flutter project.

Once you've added the `"translator"` dependency in the `pubspec.yaml` file and run `flutter pub get` in the terminal, you can follow the code provided in the previous response to create your language translation app within your Flutter project.

B You can try the following suggestion to make the pubspec resolve:
 * Consider downgrading your constraint on translator: flutter pub add translator:^0.17

- ### 4.
- Seeking to enhance the user experience, I sought guidance from the AI for implementing a language selection dropdown menu. Additionally, I requested guidance on extending the language repertoire to accommodate a broader range of options. The AI adeptly detailed the utilization of language maps, enriching my understanding of the intricate configuration.



B add a dropdown to choose languages import 'package:flutter/material.dart'; import 'package:translator/translator.dart'; import 'package:flutter_gradients/flutter_gradients.dart';

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

class MyApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'Language Translator',
      theme: ThemeData(
        primarySwatch: Colors.blue,
      ),
      home: MyHomePage(),
    );
  }
}

class MyHomePage extends StatefulWidget {
  @override
  _MyHomePageState createState() => _MyHomePageState();
}

class _MyHomePageState extends State<MyHomePage> {
  final translator = GoogleTranslator();
  String _inputText = '';
  String _translatedText = '';
  String _selectedLanguage = 'es'; // Default language is Spanish
```

```

    ),
    onProceeded: _translateText,
    child: Text('Translate'),
),
SizedBox(height: 20),
Text(
  'Translated Text',
  style: TextStyle(fontWeight: FontWeight.bold),
),
SizedBox(height: 10),
Text(
  _translatedText,
  style: TextStyle(fontStyle: FontStyle.italic),
),
),
),
),
);
}
)
}

```

In this updated code, a `'languageMap'` is defined to associate the language codes with their full names. The dropdown will now display the full names of the languages instead of their codes, making it more user-friendly. You can add more languages to the `'languageMap'` by following the same pattern if needed.

S ChatGPT can make mistakes. Consider checking important information.

Microsoft Edge window showing the generated Flutter code for a translation application. The code includes a `languageMap` for translating text between English, Spanish, French, German, Chinese, and Arabic. The window also shows a toolbar with icons for search, file operations, and system status.

Microsoft Outlook Personal Email | TP-Session-2.docx | Crear aplicación traducción: Flutter | +

chat.openai.com/c5e8cb03e-a4a8-4fd0-a92b-39f408d6559e

Gmail YouTube Maps Screenshot showing... New Tab

i want to add more languages import 'package:flutter/material.dart';
import 'package:translator/translator.dart';

```

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

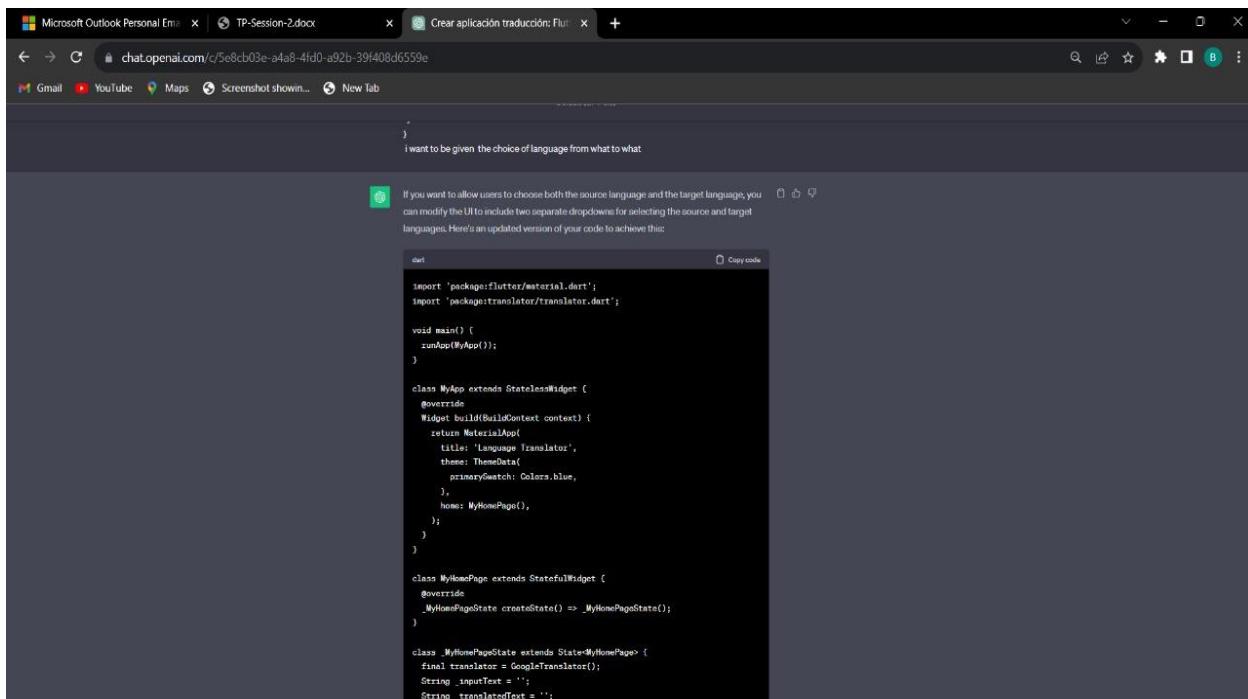
class MyApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'Language Translator',
      theme: ThemeData(
        primarySwatch: Colors.blue,
      ),
      home: MyHomePage(),
    );
  }
}

class MyHomePage extends StatefulWidget {
  @override
  _MyHomePageState createState() => _MyHomePageState();
}

class _MyHomePageState extends State<MyHomePage> {
  final translator = GoogleTranslator();
  String _inputText = '';
  String _translatedText = '';
  String _selectedLanguage = 'es'; // Default language is Spanish

  Map<String, String> languageMap = {
    'es': 'Spanish',
    'fr': 'French',
    'de': 'German',
    'zh': 'Chinese',
    'ar': 'Arabic',
    // Add more languages here if needed
  };

  void _translateText() {
    translator.translate(_inputText, to: _selectedLanguage).then((result) {
    
```



```

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

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

class MyApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'Language Translator',
      theme: ThemeData(
        primarySwatch: Colors.blue,
      ),
      home: MyHomePage(),
    );
  }
}

class MyHomePage extends StatefulWidget {
  @override
  _MyHomePageState createState() => _MyHomePageState();
}

class _MyHomePageState extends State<MyHomePage> {
  final translator = GoogleTranslator();
  String _inputText = '';
  String _translatedText = '';
}

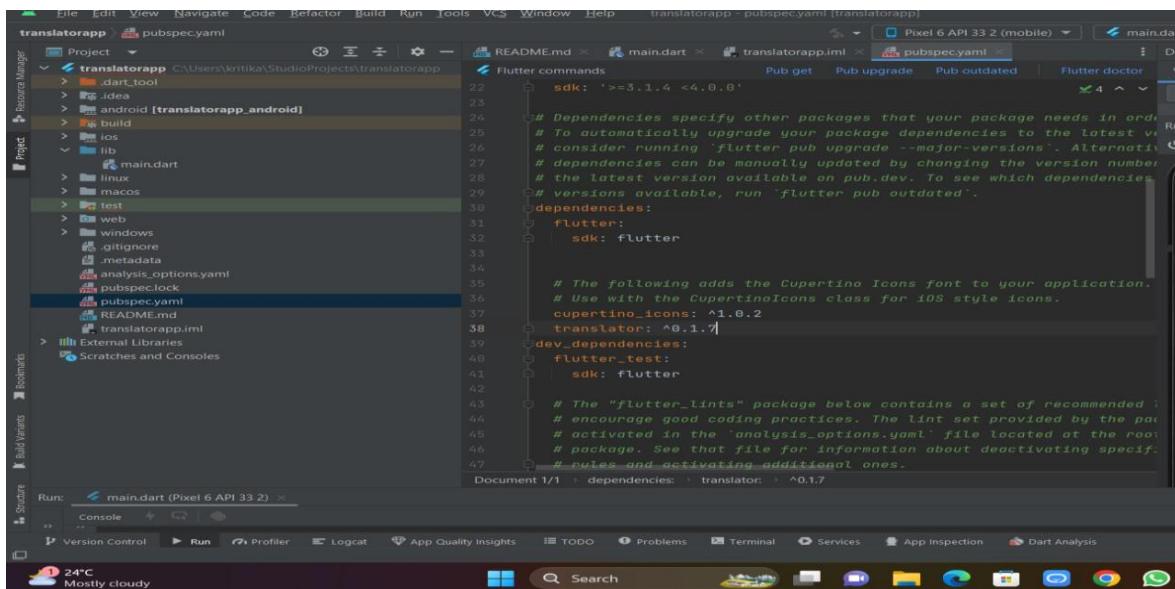
```

If you want to allow users to choose both the source language and the target language, you can modify the UI to include two separate dropdowns for selecting the source and target languages. Here's an updated version of your code to achieve this:

4.3 Steps & explanation

1. Importing the dependencies

The translator package in Flutter provides a convenient and powerful tool for integrating language translation features into your mobile applications. By incorporating the GoogleTranslator class, developers can seamlessly enable text translation capabilities, allowing users to communicate and understand content in multiple languages.



```

dependencies:
  flutter:
    sdk: flutter

  # The following adds the Cupertino Icons font to your application.
  # Use with the CupertinoIcons class for iOS style icons.
  cupertino_icons: ^1.0.2
  translator: ^0.1.7

  dev_dependencies:
    flutter_test:
      sdk: flutter

  # The "flutter_lints" package below contains a set of recommended lints
  # encourage good coding practices. The lint set provided by the package
  # activated in the "analysis_options.yaml" file located at the root of
  # package. See that file for information about deactivating specific
  # rules and activating additional ones.
  flutter_lints: ^0.1.0

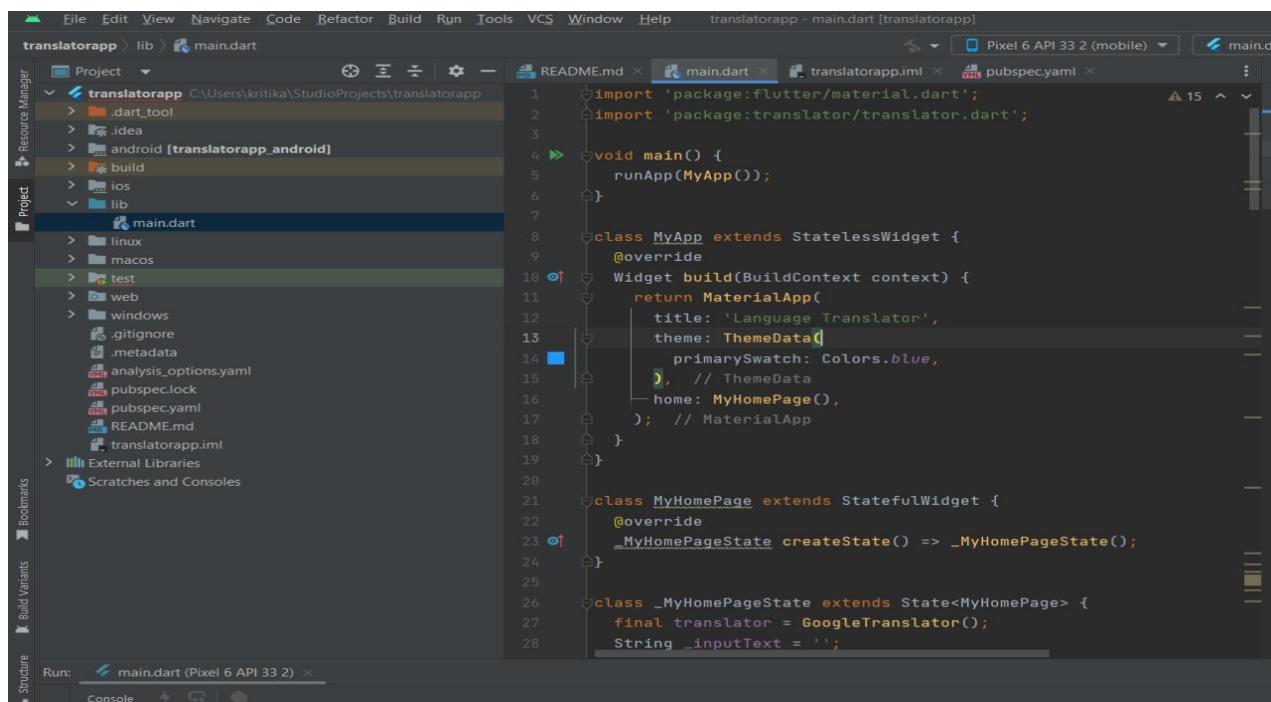
```

2. Configuring Default Values and Language Options

This Flutter code sets up a simple language translation app. The application consists of two screens: the main app (MyApp) and the home screen (MyHomePage). It uses the translator package to facilitate language translation functionality.

In the main method, the app is initiated by calling runApp(MyApp()). The main app (MyApp) is a stateless widget that defines the theme and the home screen for the application.

The home screen (MyHomePage) is a stateful widget that contains text input, dropdowns for selecting source and target languages, and a button to trigger translation. The translation is performed using the GoogleTranslator class from the translator package. Users can choose both the source and target languages from dropdown menus, and the translated text is displayed on the screen.



```

File Edit View Navigate Code Refactor Build Run Tools VCS Window Help translatorapp - main.dart [translatorapp]
translatorapp > lib > main.dart
Project Manager Project C:\Users\kritika\StudioProjects\translatorapp
> .dart_tool
> .idea
> android [translatorapp_android]
> build
> ios
lib
> main.dart
> linux
> macos
test
> web
> windows
> .gitignore
> metadata
> analysis_options.yaml
> pubspec.lock
> pubspec.yaml
> README.md
> translatorapp.iml
External Libraries
Scratches and Consoles
Run: main.dart (Pixel 6 API 33 2) ×
Console

```

```

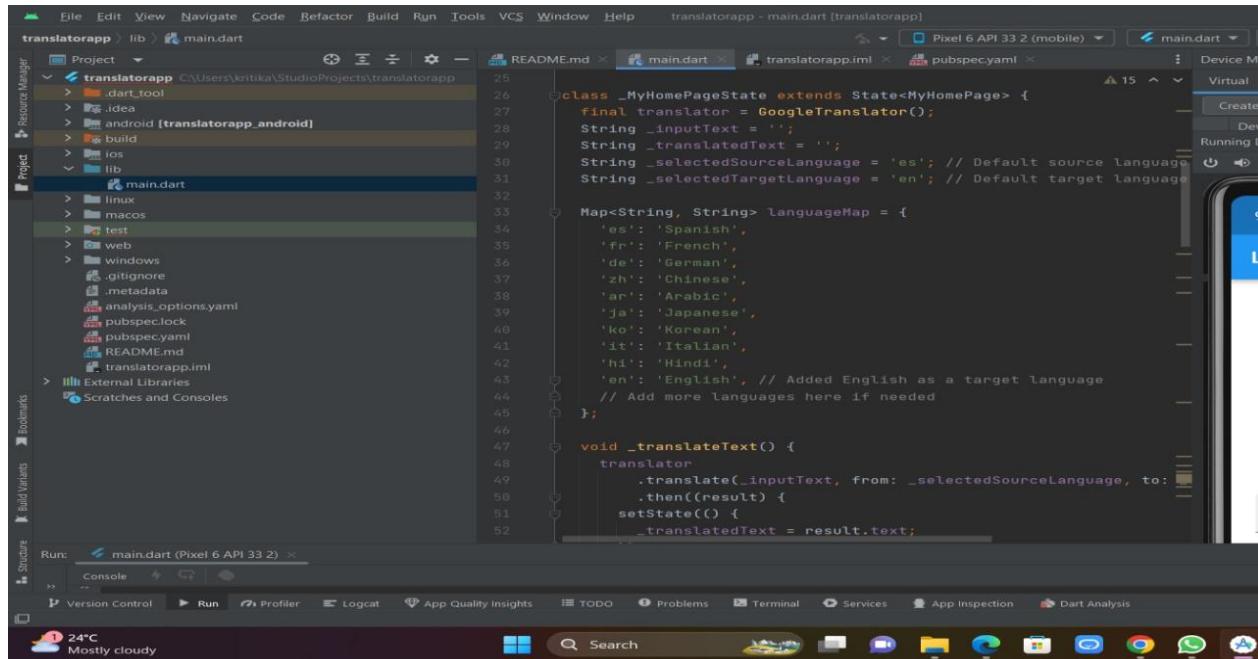
1 import 'package:flutter/material.dart';
2 import 'package:translator/translator.dart';
3
4 void main() {
5   runApp(MyApp());
6 }
7
8 class MyApp extends StatelessWidget {
9   @override
10  Widget build(BuildContext context) {
11    return MaterialApp(
12      title: 'Language Translator',
13      theme: ThemeData(
14        primarySwatch: Colors.blue,
15      ), // ThemeData
16      home: MyHomePage(),
17    ); // MaterialApp
18  }
19
20
21 class MyHomePage extends StatefulWidget {
22   @override
23   _MyHomePageState createState() => _MyHomePageState();
24 }
25
26 class _MyHomePageState extends State<MyHomePage> {
27   final translator = GoogleTranslator();
28   String _inputText = '';

```

3. User Interface Definition

The code establishes default values for the source and target languages (Spanish and English, respectively) and includes a mapping of language codes to human-readable names. Developers can easily extend the language options by adding more key-value pairs to the languageMap variable.

This code provides a foundation for a language translation app, enabling developers to create applications that support multilingual communication and understanding.



```

class _MyHomePageState extends State<MyHomePage> {
    final translator = GoogleTranslator();
    String _inputText = '';
    String _translatedText = '';
    String _selectedSourceLanguage = 'es'; // Default source language
    String _selectedTargetLanguage = 'en'; // Default target language

    Map<String, String> languageMap = {
        'es': 'Spanish',
        'fr': 'French',
        'de': 'German',
        'zh': 'Chinese',
        'ar': 'Arabic',
        'ja': 'Japanese',
        'ko': 'Korean',
        'it': 'Italian',
        'hi': 'Hindi',
        'en': 'English', // Added English as a target language
        // Add more languages here if needed
    };

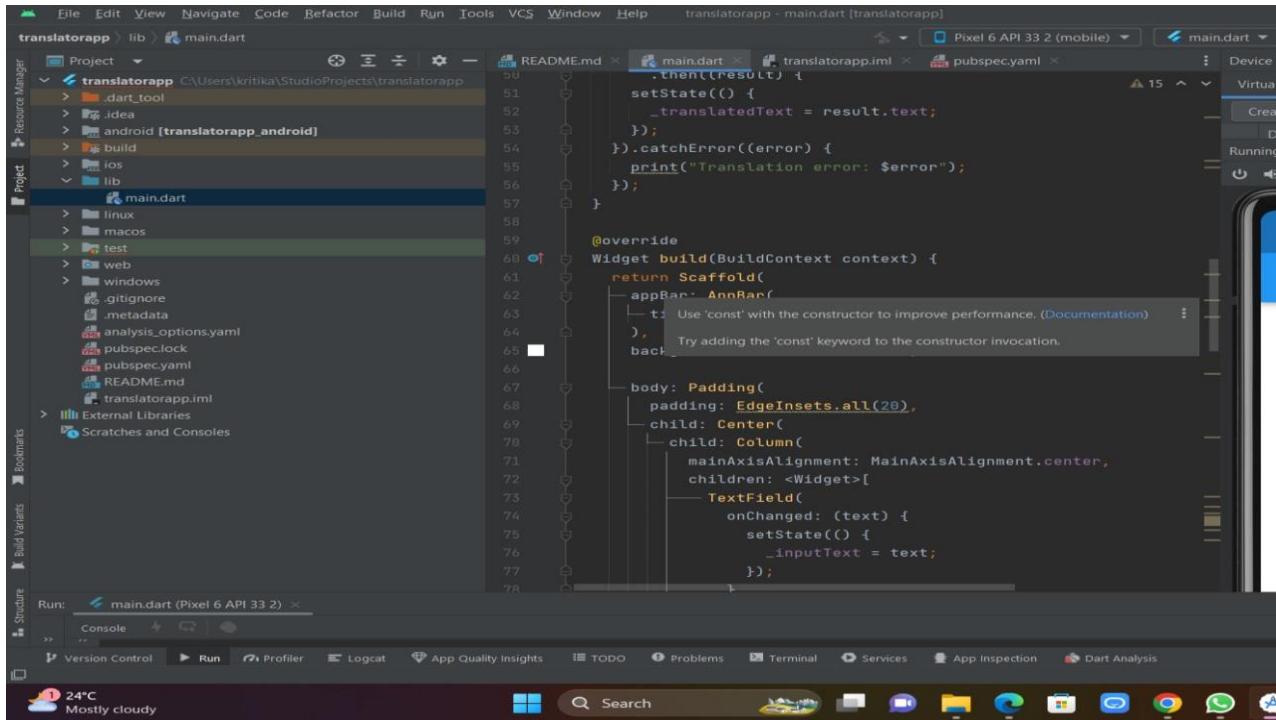
    void _translateText() {
        translator
            .translate(_inputText, from: _selectedSourceLanguage, to: _selectedTargetLanguage)
            .then((result) {
                setState(() {
                    _translatedText = result.text;
                });
            });
    }
}

```



4. Translation Process and UI Elements

This part of the Flutter code defines the user interface for the language translation app. The build method returns a Scaffold widget, providing the basic structure of the app. The AppBar at the top displays the app's title as "Language Translator." The background color is set to white.



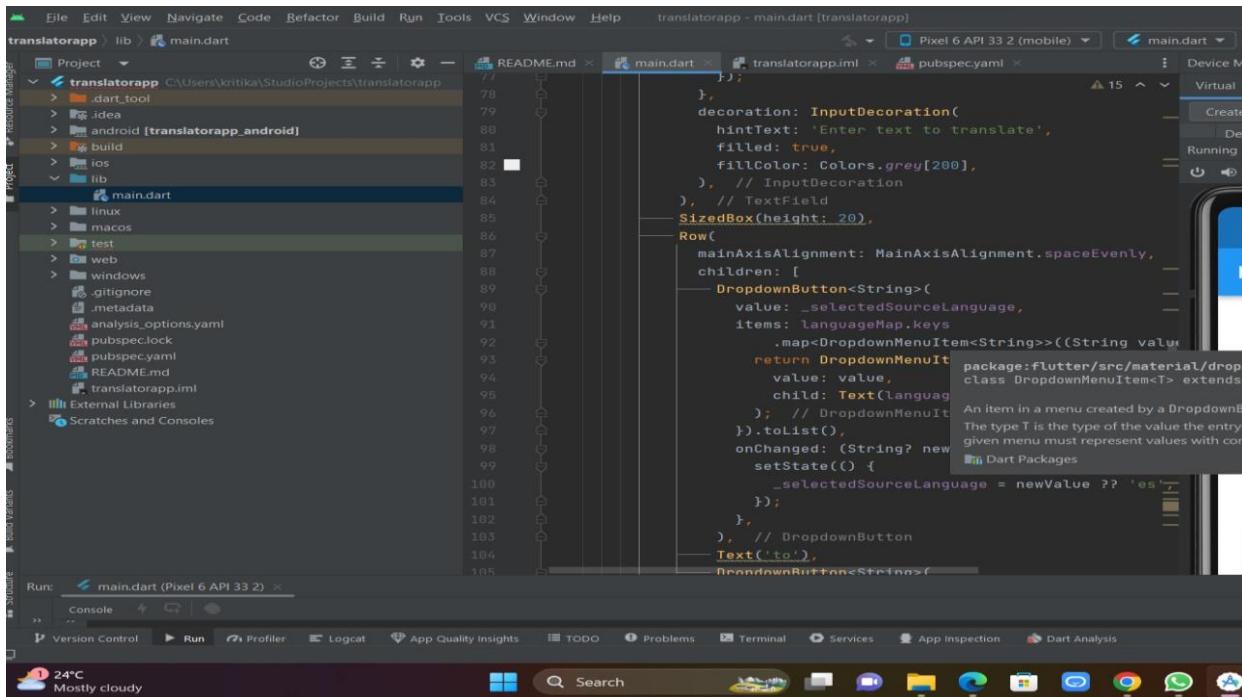
```
File Edit View Navigate Code Befactor Build Run Tools VCS Window Help translatorapp - lib > main.dart
translatorapp C:\Users\kritika\StudioProjects\translatorapp
> .dart_tool
> .idea
> android [translatorapp_android]
> build
> ios
lib
  main.dart
> linux
> macos
> test
> web
> windows
  .gitignore
  .metadata
  analysis_options.yaml
  pubspec.lock
  pubspec.yaml
  README.md
  translatorapp.iml
> External Libraries
Scratches and Consoles

main.dart
50   .then(result) {
51     setState(() {
52       _translatedText = result.text;
53     });
54   }).catchError((error) {
55     print("Translation error: $error");
56   });
57 }
58
59 @override
60 Widget build(BuildContext context) {
61   return Scaffold(
62     appBar: AppBar(
63       title: const Text('Language Translator'),
64     ),
65     body: Padding(
66       padding: EdgeInsets.all(20),
67       child: Center(
68         child: Column(
69           mainAxisAlignment: MainAxisAlignment.center,
70           children: [
71             TextField(
72               onChanged: (text) {
73                 setState(() {
74                   _inputText = text;
75                 });
76               });
77           ],
78         ),
79       ),
80     );
81   );
82 }

main.dart (Pixel 6 API 33 2) <> Console <> Version Control <> Run <> Profiler <> Logcat <> App Quality Insights <> TODO <> Problems <> Terminal <> Services <> App Inspection <> Dart Analysis
```

5. Enhancing User Interface

Within the body of the app, there's a padding of 20 units on all sides, and the main content is centered using the Center widget. The user interface includes a TextField for users to input the text they want to translate. The onChanged callback updates the internal state variable _inputText as the user types.



```

Row(
  mainAxisAlignment: MainAxisAlignment.spaceEvenly,
  children: [
    DropdownButton<String>(
      value: _selectedSourceLanguage,
      items: languageMap.keys
          .map<DropdownMenuItem<String>>((String value) {
        return DropdownMenuItem<String>(
          value: value,
          child: Text(languageMap[value]!));
      }).toList(),
      onChanged: (String? newValue) {
        setState(() {
          _selectedSourceLanguage = newValue ?? 'en';
        });
      },
    ),
    // DropdownButton
    TextField(
      controller: _inputTextController,
      decoration: InputDecoration(
        hintText: 'Enter text to translate',
        filled: true,
        fillColor: Colors.grey[200],
      ),
    ), // TextField
  ],
),
SizedBox(height: 20),

```

6. Initiating Translation with 'Translate' Button

Below the language selection section, there's an ElevatedButton labeled 'Translate.' This button triggers the _translateText method when pressed, initiating the translation process based on the user's input and language selections.



The screenshot shows the Android Studio interface with the main.dart file open. The code implements a dropdown menu for selecting source and target languages. It uses a `DropdownButton` for the source language and a `DropdownMenuItem` for the target language. The target language dropdown lists available languages from a `languageMap` key. The selected target language is stored in the `_selectedTargetLanguage` variable.

```
111     value: value,
112     child: Text(languageMap[value]!),
113   ); // DropdownMenuItem
114 ).toList(),
115 onChanged: (String? newValue) {
116   setState(() {
117     _selectedTargetLanguage = newValue ?? 'en';
118   });
119 },
120 ), // Row
121 SizedBox(height: 20),
122 ElevatedButton(
123   onPressed: _translateText,
124   child: Text('Translate'),
125 ), // ElevatedButton
126 SizedBox(height: 20),
127 Text(
128   'Translated Text:',
129   style: TextStyle(fontWeight: FontWeight.bold),
130 ), // Text
131 SizedBox(height: 10),
132 Text(
133   _translatedText,
134   style: TextStyle(fontStyle: FontStyle.italic),
135 ), // Text
136 ], // <Widget>[]
137 ), // Column
138 )
```

7. Improving User Interaction: Source and Target Language Selection

This segment of Flutter code enhances the language translation app's user interface. Following the source language dropdown, it introduces a 'to' label and a target language dropdown, allowing users to specify both source and target languages for translation. The selected target language is stored in the `_selectedTargetLanguage` variable.

The screenshot shows the Android Studio interface with the main.dart file open. The code has been updated to include a 'to' label and a second `DropdownButton` for the target language. The target language dropdown lists available languages from a `languageMap` key. The selected target language is stored in the `_selectedTargetLanguage` variable. The code also includes logic to update the translated text based on the selected languages.

```
101     value: value,
102     child: Text(languageMap[value]!),
103   ); // DropdownMenuItem
104 ).toList(),
105 onChanged: (String? newValue) {
106   setState(() {
107     _selectedTargetLanguage = newValue ?? 'en';
108   });
109 },
110 ), // Row
111 SizedBox(height: 20),
112 ElevatedButton(
113   onPressed: _translateText,
114   child: Text('Translate'),
115 ), // ElevatedButton
116 SizedBox(height: 20),
117 Text(
118   'Translated Text:',
119   style: TextStyle(fontWeight: FontWeight.bold),
120 ), // Text
121 SizedBox(height: 10),
122 Text(
123   _translatedText,
124   style: TextStyle(fontStyle: FontStyle.italic),
125 ), // Text
126 ], // <Widget>[]
127 ), // Column
128 )
```

4.4 Final Output

