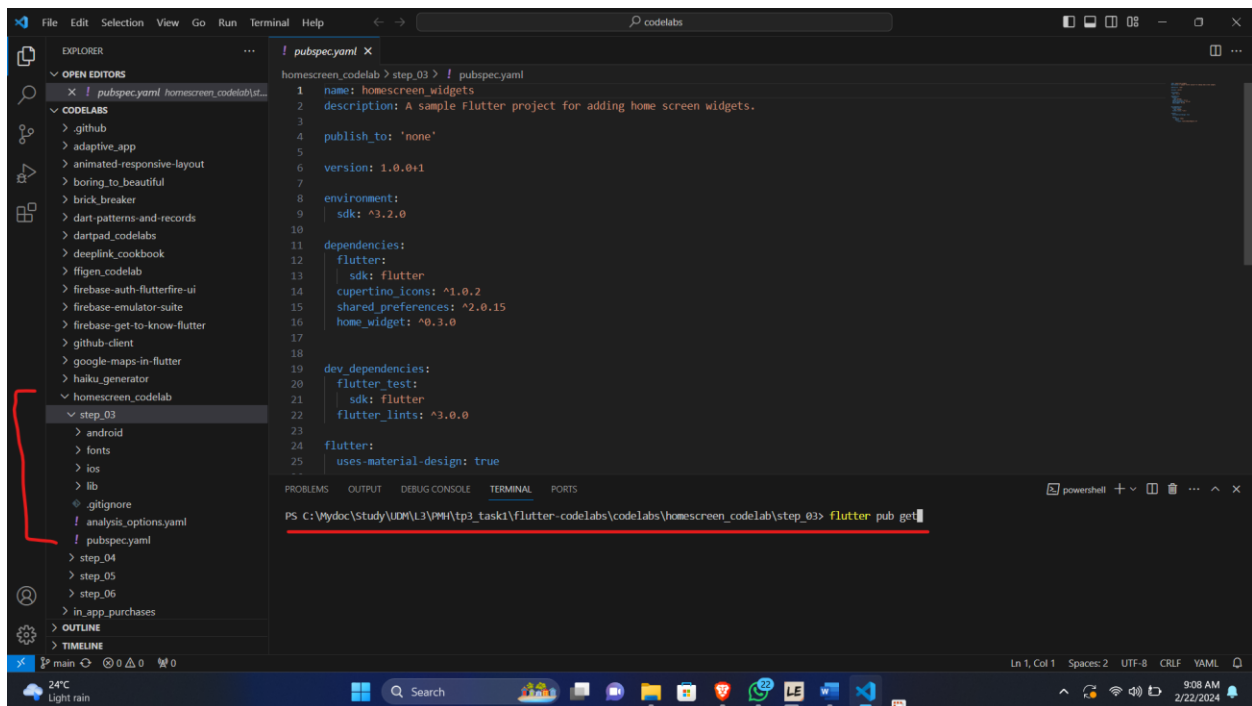


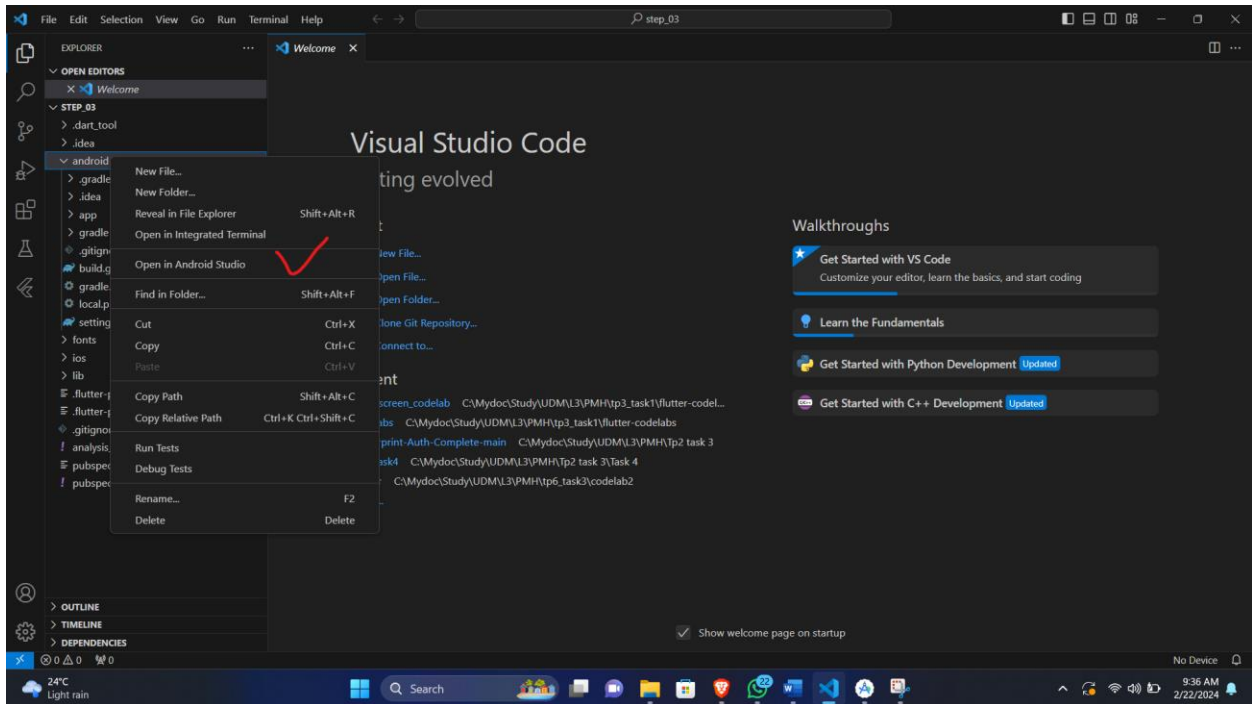
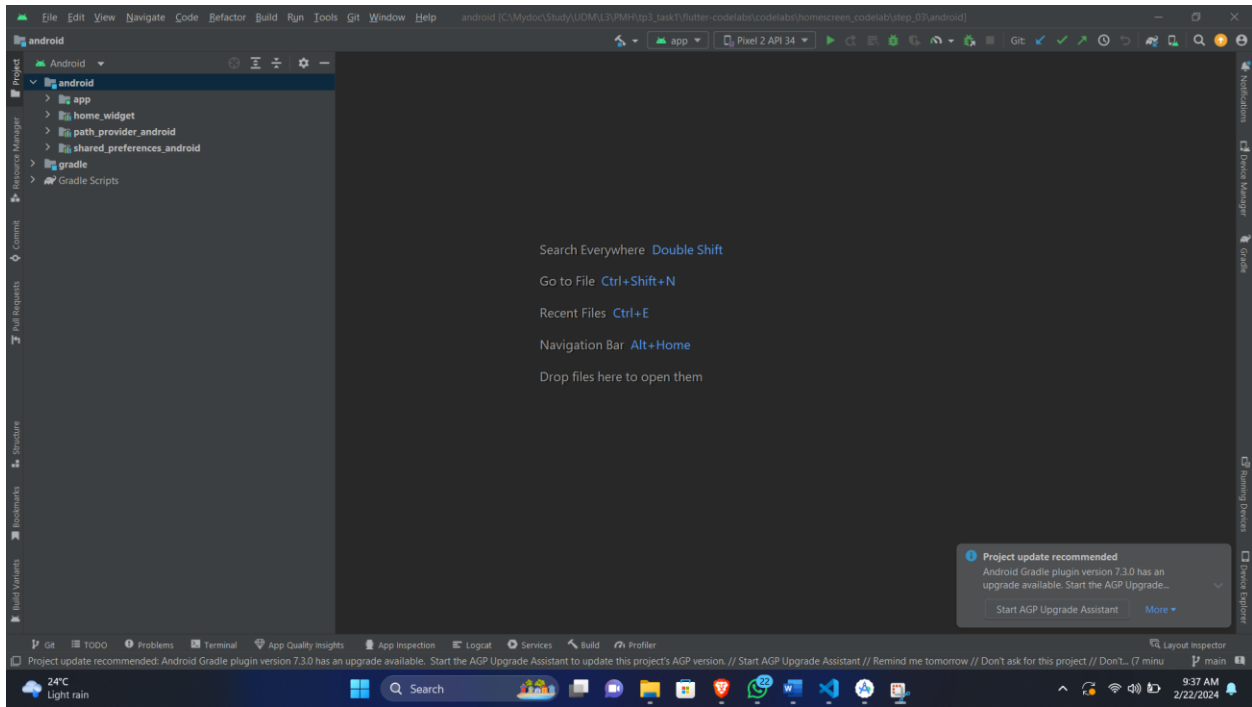
RAKOTONDRAMANANA A.A. LAFATRA

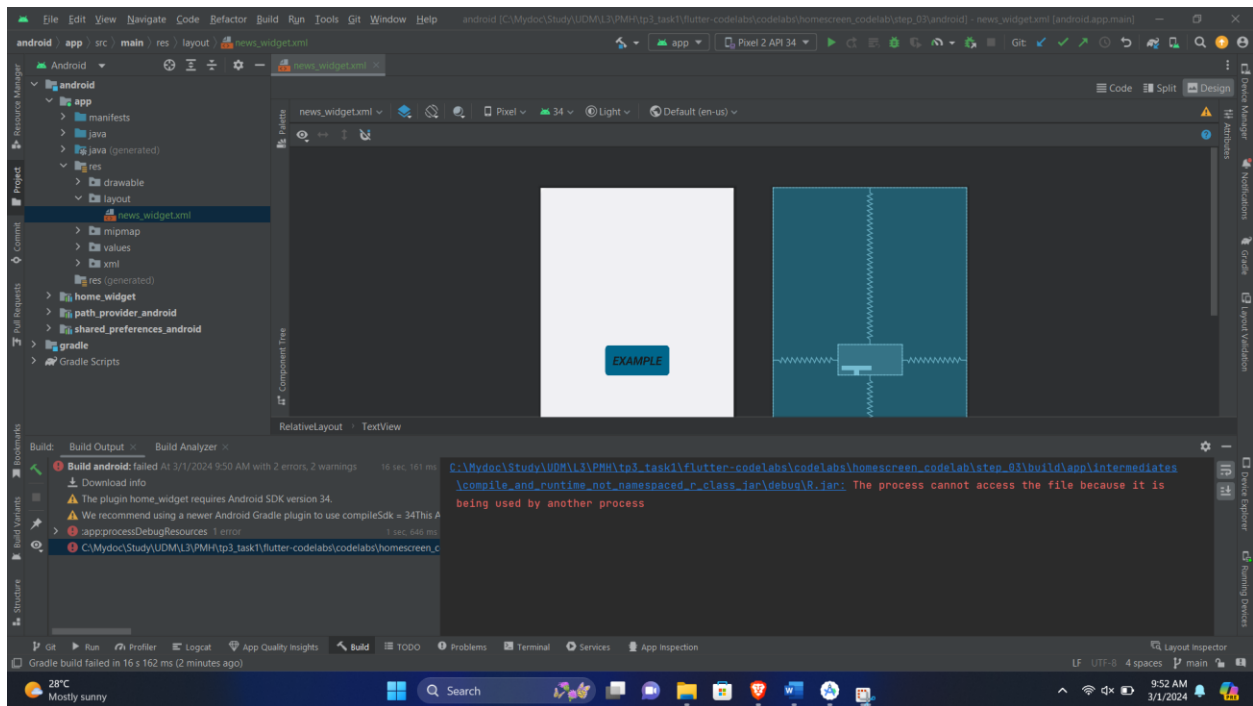
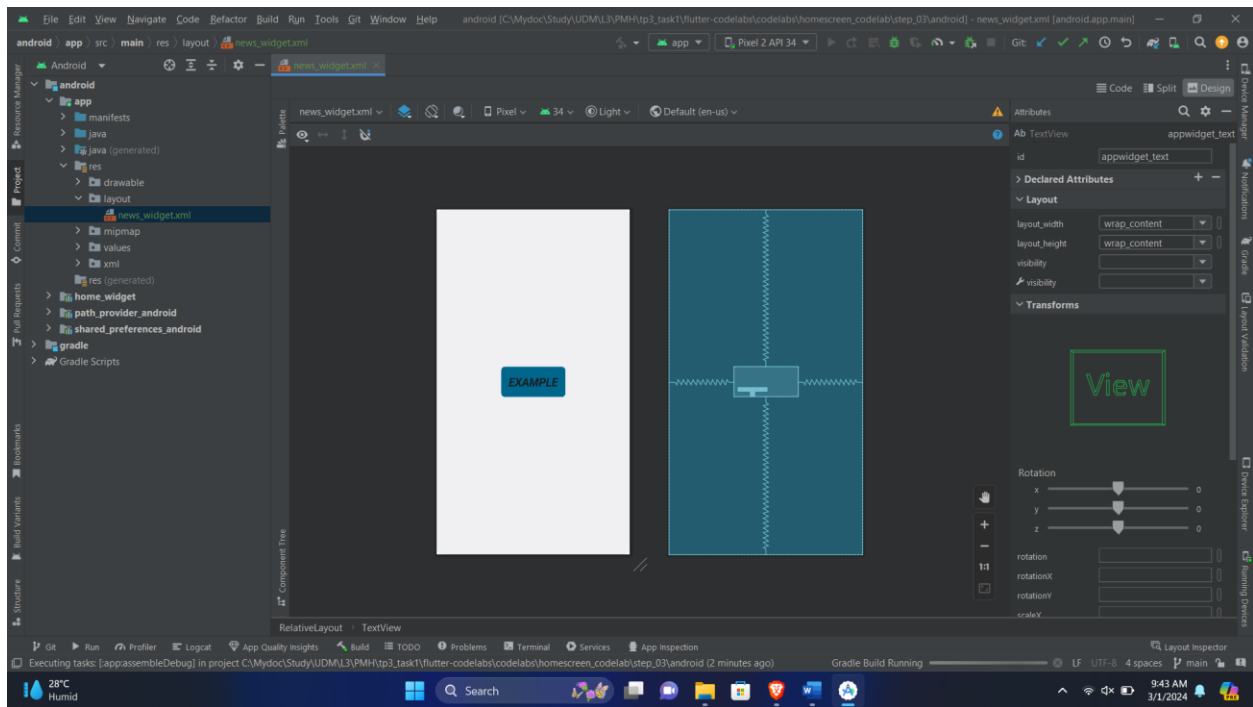
INFO 3 – UNIVERSITE DES MASCAREIGNES.

TP session 3: Multi-Screens Design using Flutter

Task-1: Adding a Home Screen widget to your Flutter App







Lab Documentation: Adding Home Screen Widget in Flutter

Step 1: Download Project from GitHub

1. Open your command line interface.
2. Clone the GitHub repository into a new directory named "flutter-codelabs" using the following command:

```
git clone https://github.com/flutter/codelabs.git flutter-codelabs
```

Step 2: Open the Starter App in your Preferred IDE

1. Navigate to the "homescreen_codelab" directory in the cloned repository.
2. Open the "step_03" directory in your preferred Integrated Development Environment (IDE).

Step 3: Install Packages

1. Open a terminal within your IDE or use the command line.
2. Navigate to the "homescreen_codelab/step_03" directory.
3. Run the following command to retrieve project dependencies:

```
flutter pub get
```

Step 4: Creating a Basic Android Widget

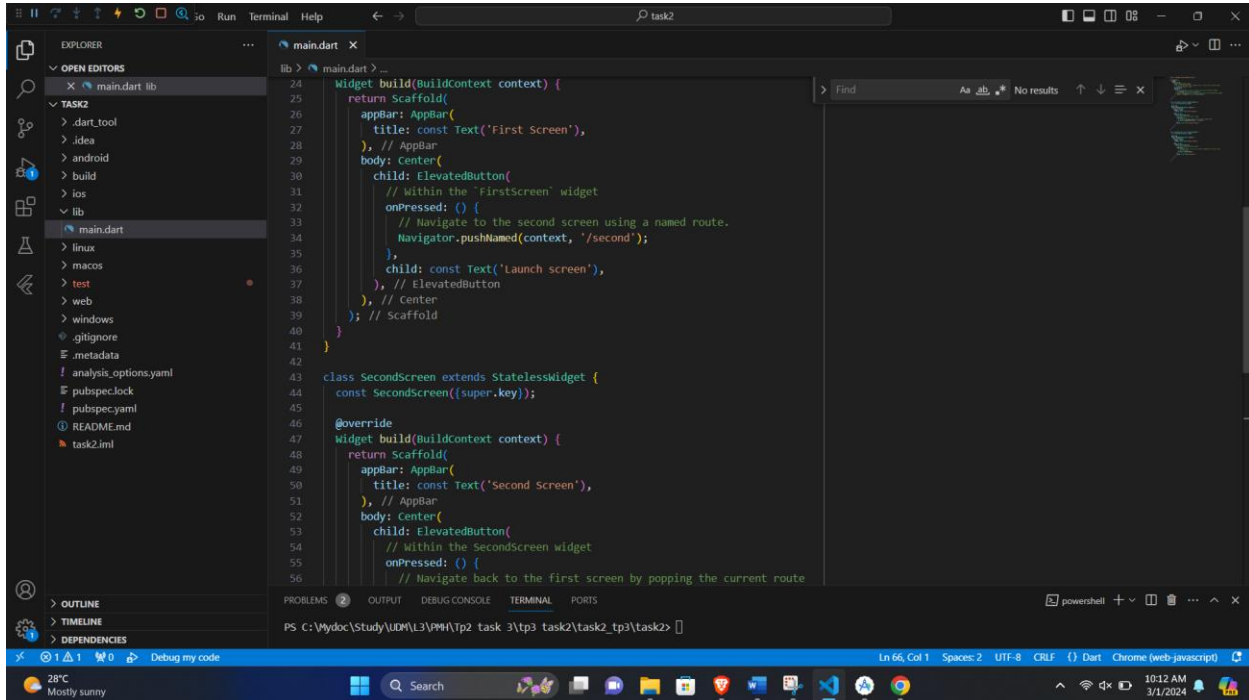
1. Open Android Studio.
2. Open the build file located at "android/build.gradle" in Android Studio.
 - Alternatively, right-click on the "android" folder in VSCode and select "Open in Android Studio."
3. After the project builds, locate the "app" directory in the top-left corner of Android Studio.
4. Right-click on the "app" directory, select "New" -> "Widget" -> "App Widget."

Step 5: Configuring the Home Screen Widget

1. Android Studio displays a new form for configuring the Home Screen widget.
2. Add basic information for the Home Screen widget:
 - Set the "Class Name" box to "NewsWidget."
 - Set the "Minimum Width (cells)" dropdown to "3."

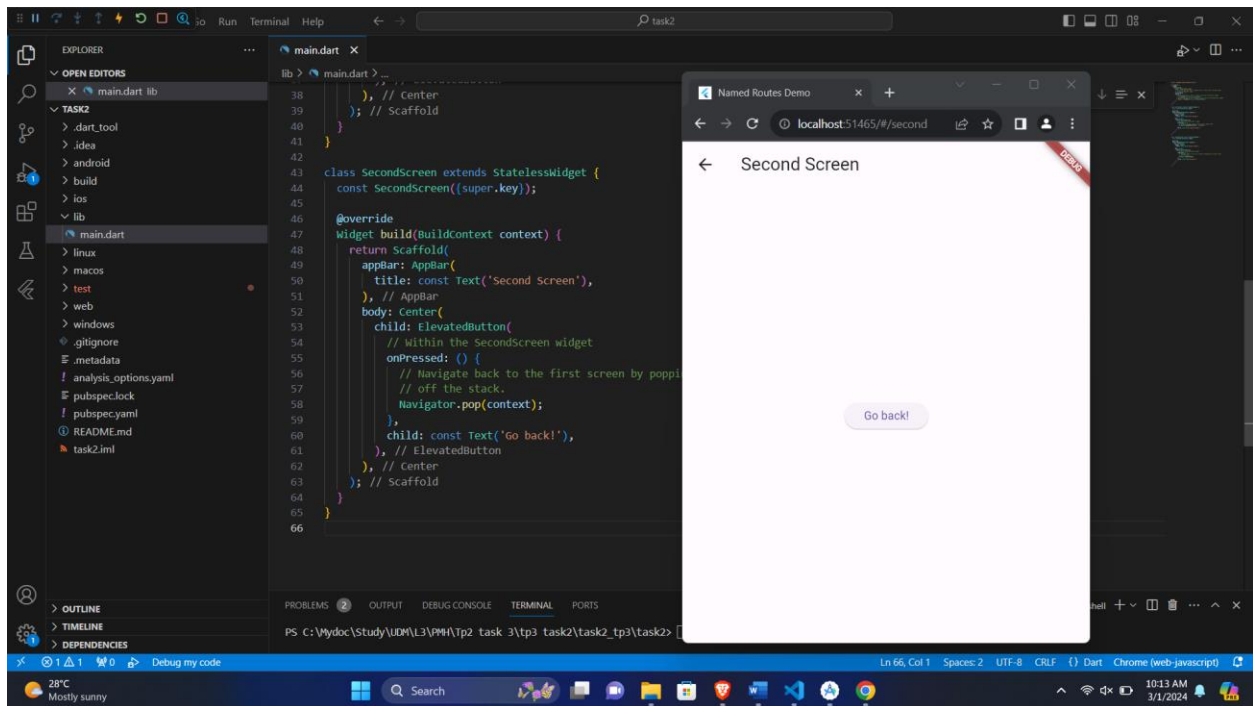
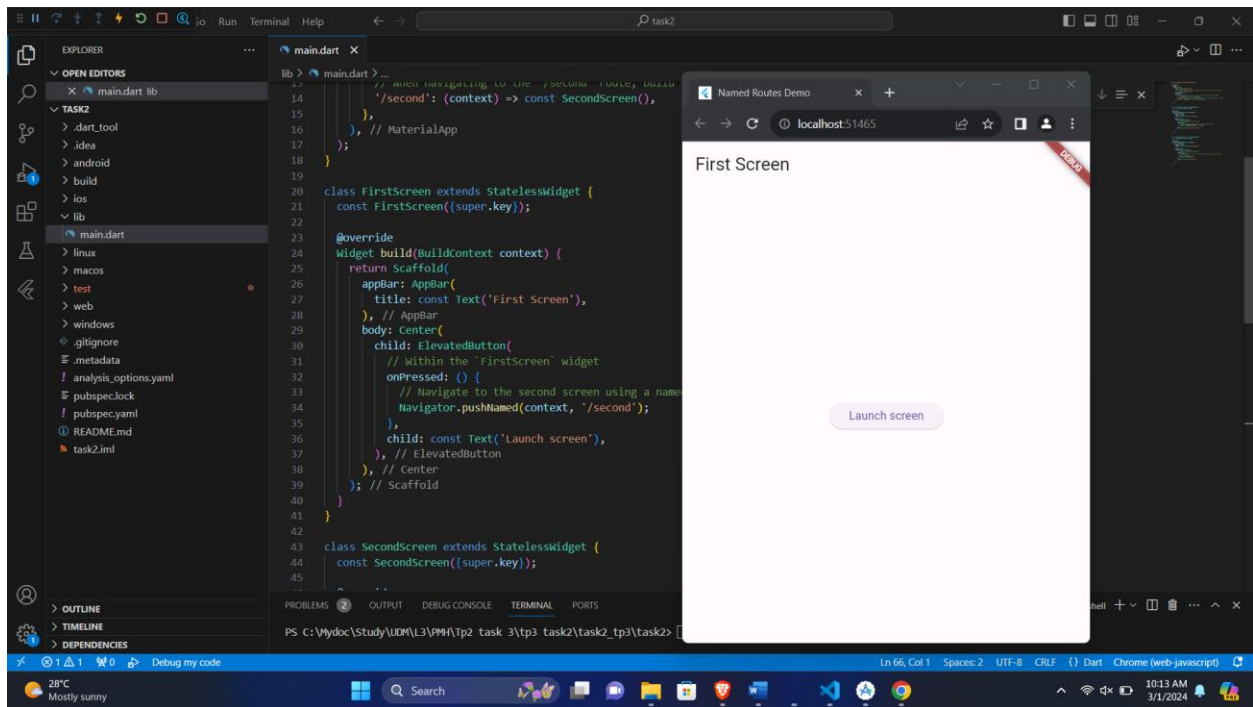
- Set the "Minimum Height (cells)" dropdown to "3."

Task-2: Use the given dart files to complete the mutli-screen application in Flutter using the routing and tabular navigation.



The screenshot shows an IDE with the following Dart code in `main.dart`:

```
24 widget build(BuildContext context) {  
25   return Scaffold(  
26     appBar: AppBar(  
27       title: const Text('First Screen'),  
28     ), // AppBar  
29     body: Center(  
30       child: ElevatedButton(  
31         // within the 'FirstScreen' widget  
32         onPressed: () {  
33           // Navigate to the second screen using a named route.  
34           Navigator.pushNamed(context, '/second');  
35         },  
36         child: const Text('Launch screen'),  
37       ), // ElevatedButton  
38     ), // Center  
39   ); // Scaffold  
40 }  
41  
42  
43 class SecondScreen extends StatelessWidget {  
44   const SecondScreen({super.key});  
45  
46   @override  
47   widget build(BuildContext context) {  
48     return Scaffold(  
49       appBar: AppBar(  
50         title: const Text('Second Screen'),  
51       ), // AppBar  
52       body: Center(  
53         child: ElevatedButton(  
54           // within the SecondScreen widget  
55           onPressed: () {  
56             // Navigate back to the first screen by popping the current route
```



Lab Documentation: Creating Two Screens and Navigation in Flutter

Step 1: Create Two Screens

1. Create the first screen (**FirstScreen**):
 - Include a button to navigate to the second screen.
 - The button's `onPressed()` callback should use **`Navigator.pushNamed()`** to go to the second screen.
2. Create the second screen (**SecondScreen**):
 - Include a button to navigate back to the first screen.
 - The button's `onPressed()` callback should use **`Navigator.pop()`** to return to the first screen.

Step 2: Define Routes

1. Open the **`main.dart`** file.
2. In the **`MaterialApp`** constructor, define the **`initialRoute`** property to set the starting route.
3. Define named routes in the **`routes`** property, mapping route names to the corresponding screen widgets.

Example:

```
MaterialApp( initialRoute: '/', routes: { '/': (context) => FirstScreen(), '/second': (context) => SecondScreen(), }, )
```

Step 3: Navigate to the Second Screen

1. In the **`onPressed`** callback of the button in **`FirstScreen`**:

```
onPressed: () { Navigator.pushNamed(context, '/second'); }
```

This triggers navigation to the second screen using the defined route.

Step 4: Return to the First Screen

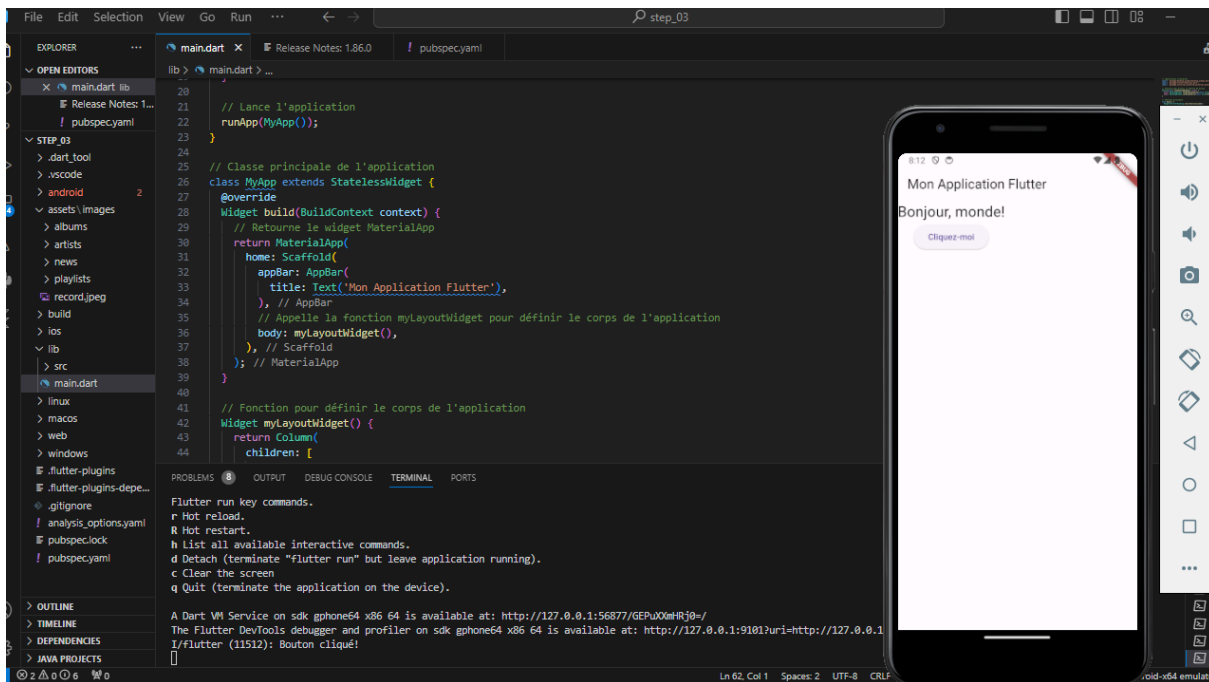
1. In the **`onPressed`** callback of the button in **`SecondScreen`**:

```
onPressed: () { Navigator.pop(context); }
```

This uses **`Navigator.pop()`** to navigate back to the first screen.

Task-3: Adding AdMob ads to a Flutter app

Ce code Flutter crée une application pour les environnements de bureau avec une fenêtre personnalisable. L'interface utilisateur est créée par la classe MyApp avec une barre d'applications et une fonction myLayoutWidget pour organiser un texte de salutation et un bouton dans une colonne. Un message est imprimé dans la console lorsque le bouton est cliqué. Pour les applications de bureau, la taille de la fenêtre est ajustée automatiquement.



The screenshot shows an IDE with the following components:

- EXPLORER:** Shows the project structure with folders like .dart_tool, .vscode, android, assets, build, ios, lib, src, linux, macos, web, windows, flutter-plugins, and flutter-plugins-depe....
- main.dart:** Contains the following code:


```

20 // Lance l'application
21 runApp(MyApp());
22
23
24 // Classe principale de l'application
25 class MyApp extends StatelessWidget {
26   @override
27   Widget build(BuildContext context) {
28     // Retourne le widget MaterialApp
29     return MaterialApp(
30       home: Scaffold(
31         appBar: AppBar(
32           title: Text('Mon Application Flutter'),
33         ), // AppBar
34         // Appelle la fonction myLayoutWidget pour définir le corps de l'application
35         body: myLayoutWidget(),
36       ), // Scaffold
37     ); // MaterialApp
38   }
39
40   // Fonction pour définir le corps de l'application
41   Widget myLayoutWidget() {
42     return Column(
43       children: [

```
- TERMINAL:** Shows the following output:

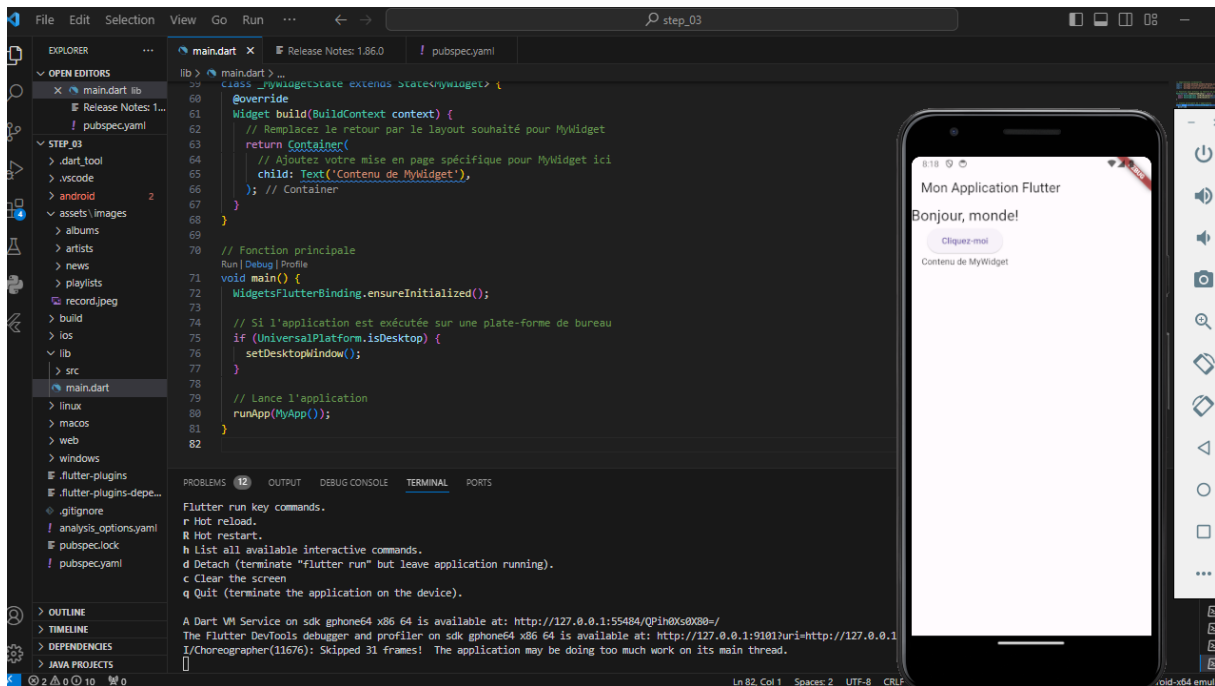

```

Flutter run key commands.
r Hot reload.
R Hot restart.
h List all available interactive commands.
d Detach (terminate "flutter run" but leave application running).
c Clear the screen
q Quit (terminate the application on the device).

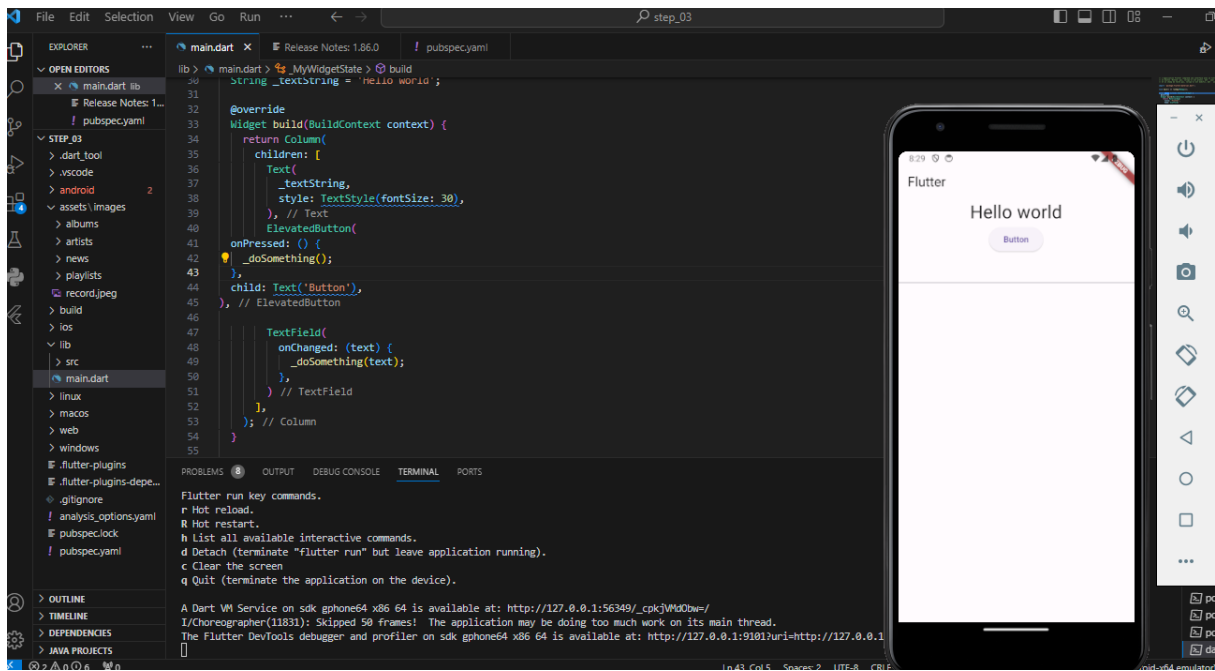
A Dart VM Service on sdk gphone64 x86_64 is available at: http://127.0.0.1:56877/GEpXXmRj9=/
The Flutter DevTools debugger and profiler on sdk gphone64 x86_64 is available at: http://127.0.0.1:9101/?uri=http://127.0.0.1:56877/GEpXXmRj9=/
I/flutter (11512): Bouton cliqué!

```
- Emulator:** Shows a mobile screen with the text "Mon Application Flutter" and "Bonjour, monde!" and a button labeled "Cliquez-moi".

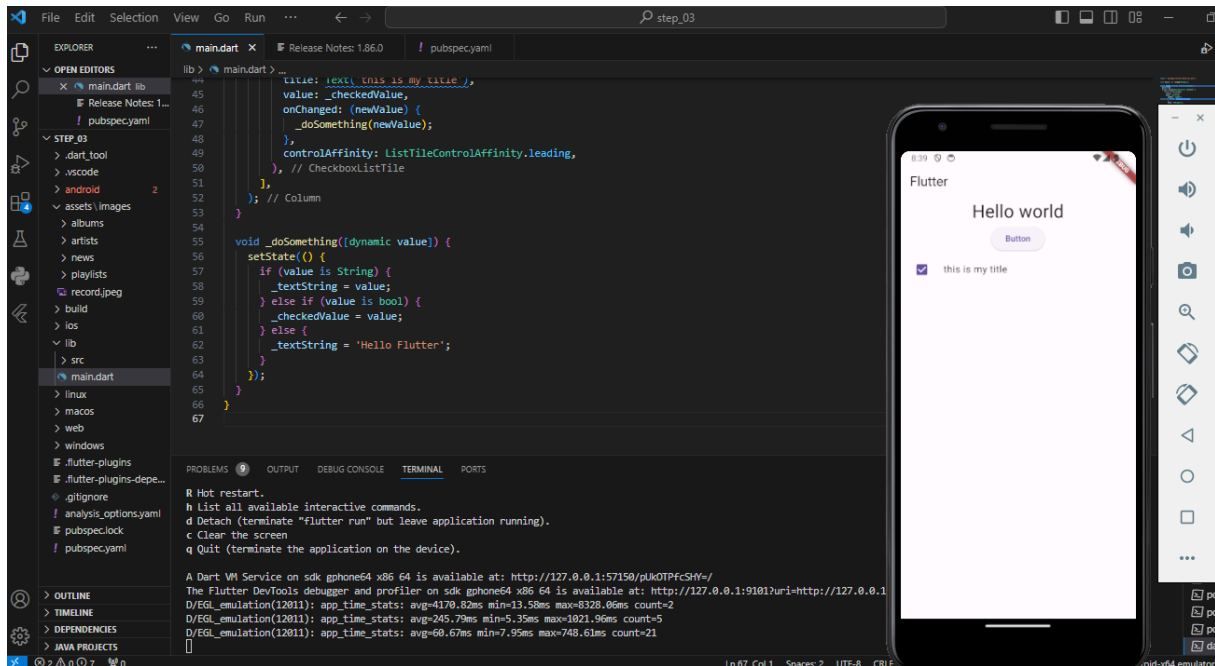
J'ai ajouté un nouveau widget appelé MyWidget, qui avait son propre état (_MyWidgetState) et une mise en page spécifique, ce qui a étendu l'application Flutter. Le widget MyWidget a été incorporé dans la structure principale de l'interface utilisateur de MyApp. Par conséquent, notre application comprend désormais une hiérarchie d'interface utilisateur plus complexe avec des éléments réutilisables pour une gestion efficace de l'état et de la mise en page.



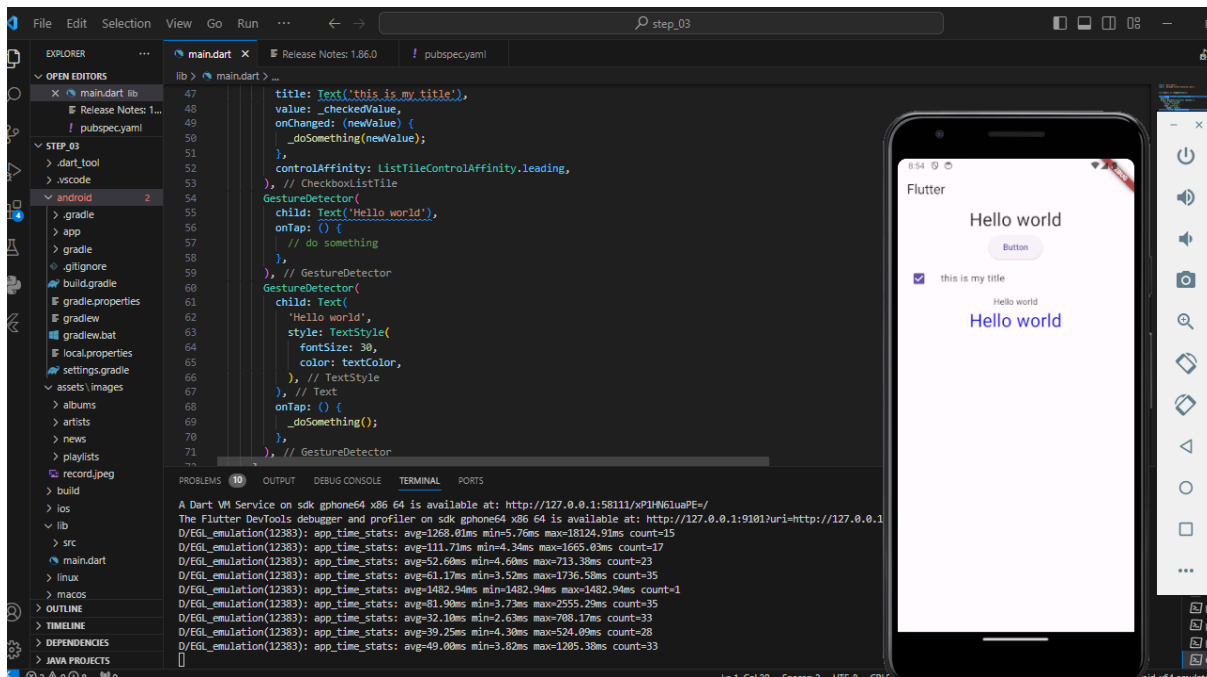
Un titre, un texte initial, un bouton et un champ de texte sont présents dans l'interface utilisateur de l'application Flutter. Lorsque le bouton est pressé, le texte est mis à jour dans le champ de texte et le texte est mis à jour en fonction de la saisie de l'utilisateur.



J'ai ajouté le `CheckboxListTile` dans `_MyWidgetState` avec la fonctionnalité correspondante. La méthode `_doSomething` a été mise à jour pour gérer la mise à jour du texte et la valeur de la case à cocher en fonction du type de valeur fourni.

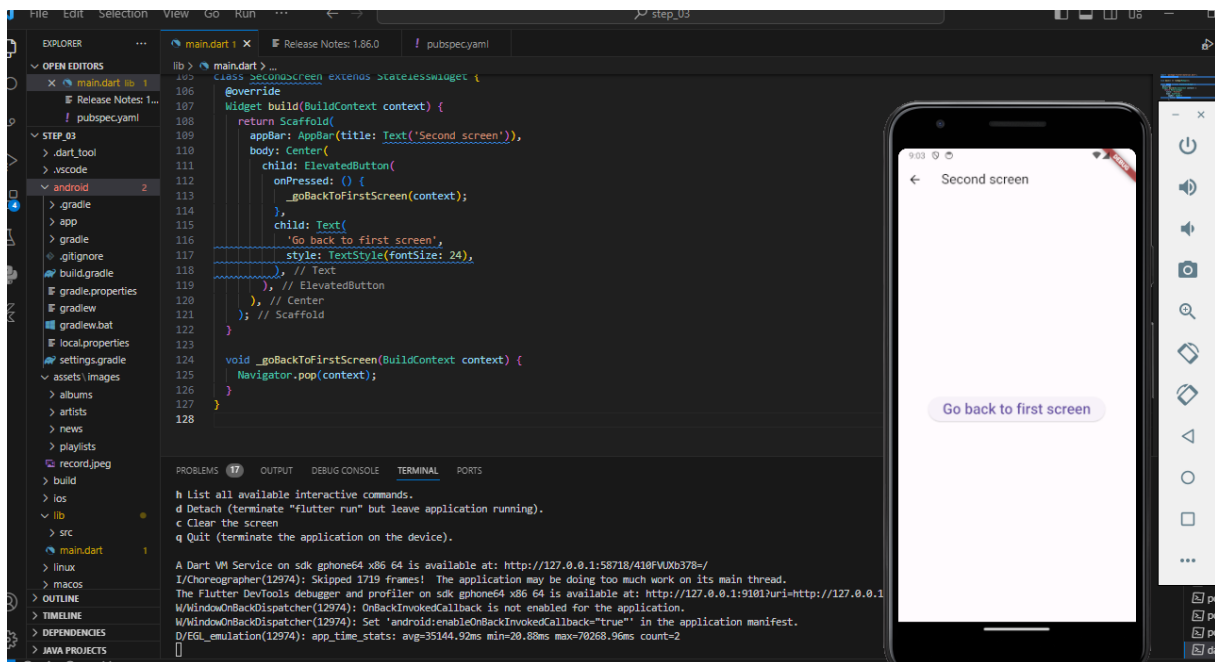
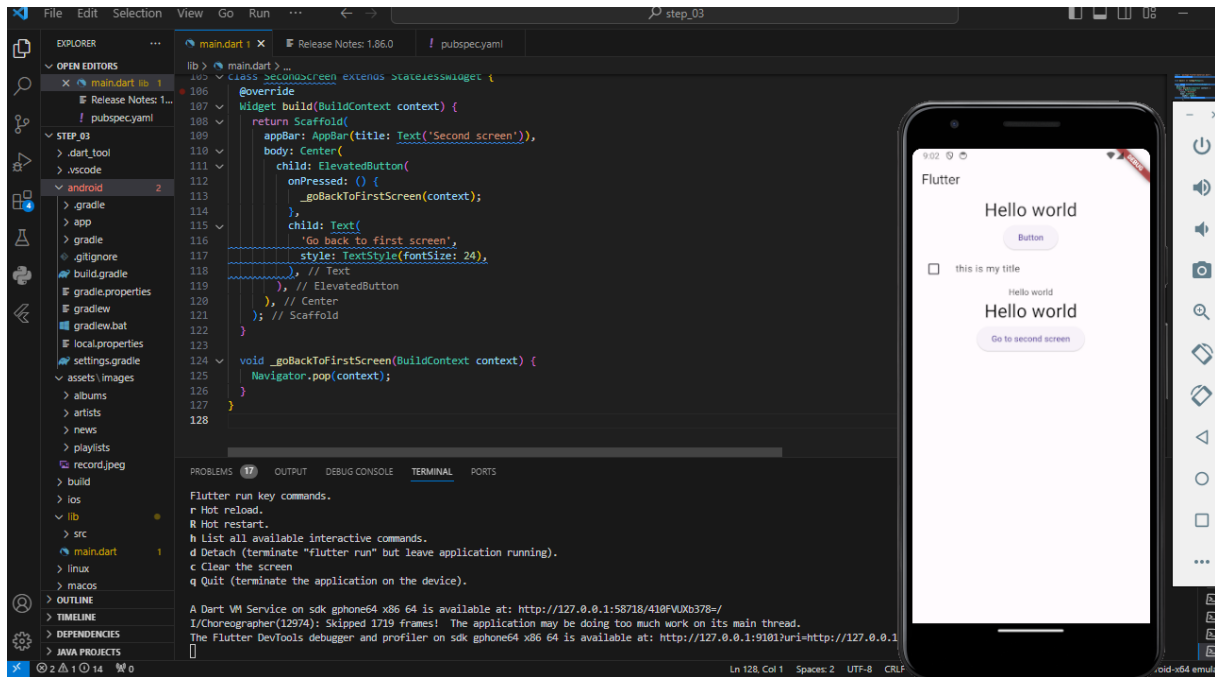


Cette application Flutter affiche du texte, un bouton, une case à cocher et réagit à certaines interactions de l'utilisateur en modifiant aléatoirement le texte, la valeur de la case à cocher et la couleur du texte. En réponse à ces interactions, la fonction principale `_doSomething` met à jour l'état de l'application.

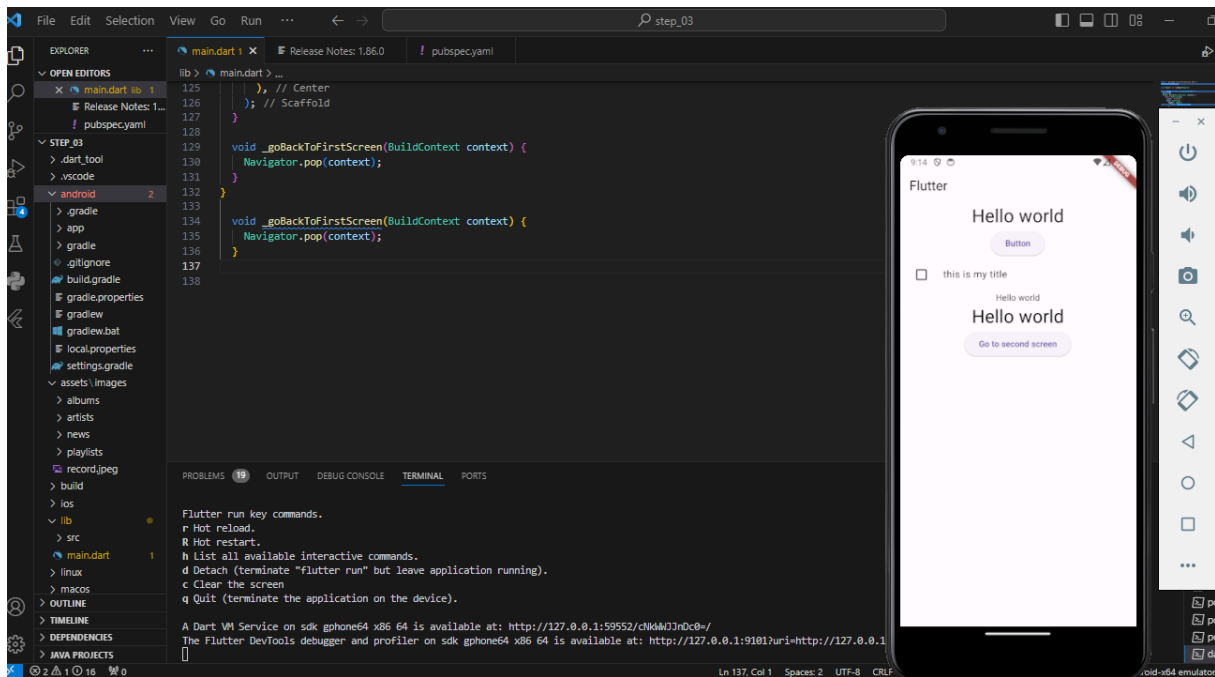


Avec ce prochain code , il y a une application à deux écrans (deux pages) :

- La première page (`MyWidget`) affiche du texte, un bouton élevé, une case à cocher, des gestes (taps) sur du texte, un changement aléatoire de la couleur du texte, et un bouton pour naviguer vers la deuxième page.
- La deuxième page (`SecondScreen`) affiche un bouton élevé qui, lorsqu'il est appuyé, ramène l'utilisateur à la première page.
- La navigation entre les pages est réalisée à l'aide de `Navigator.push` et `Navigator.pop`.

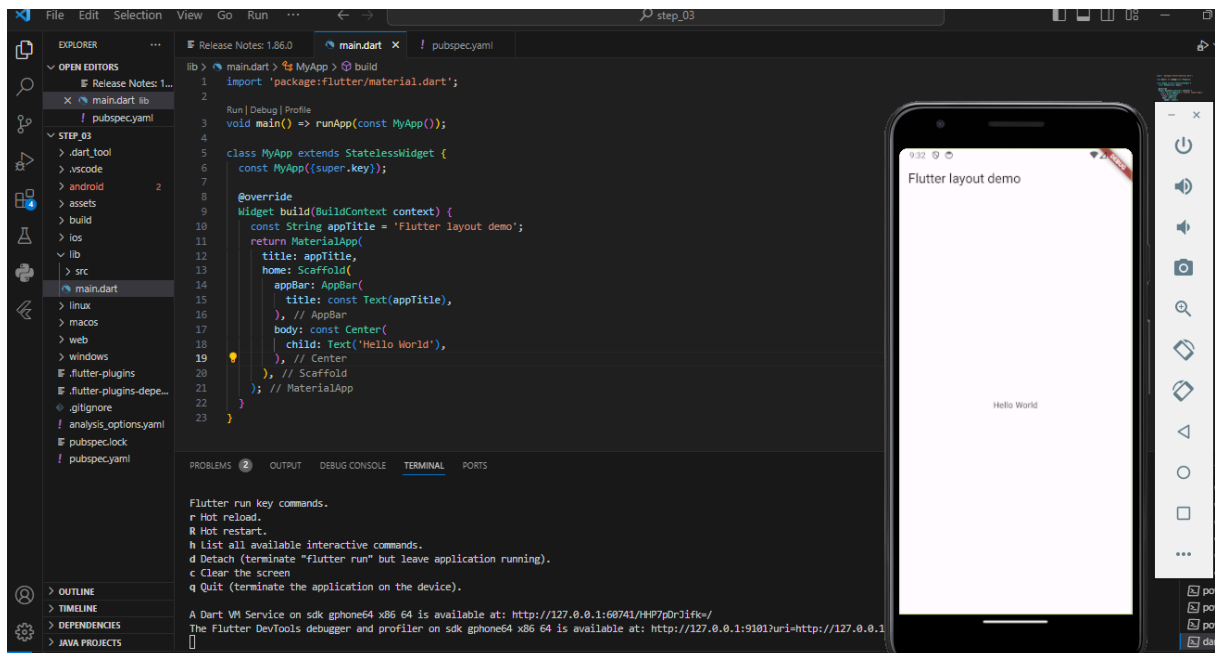


Ce prochain code Flutter crée une deuxième page ('SecondScreen') qui prend un texte en paramètre lors de son initialisation. Cette page affiche un bouton élevé, et lorsque ce bouton est pressé, elle utilise 'Navigator.pop' pour retourner à la première page. La classe 'SecondScreen' a été modifiée pour déclarer que le paramètre 'key' est requis ('required').

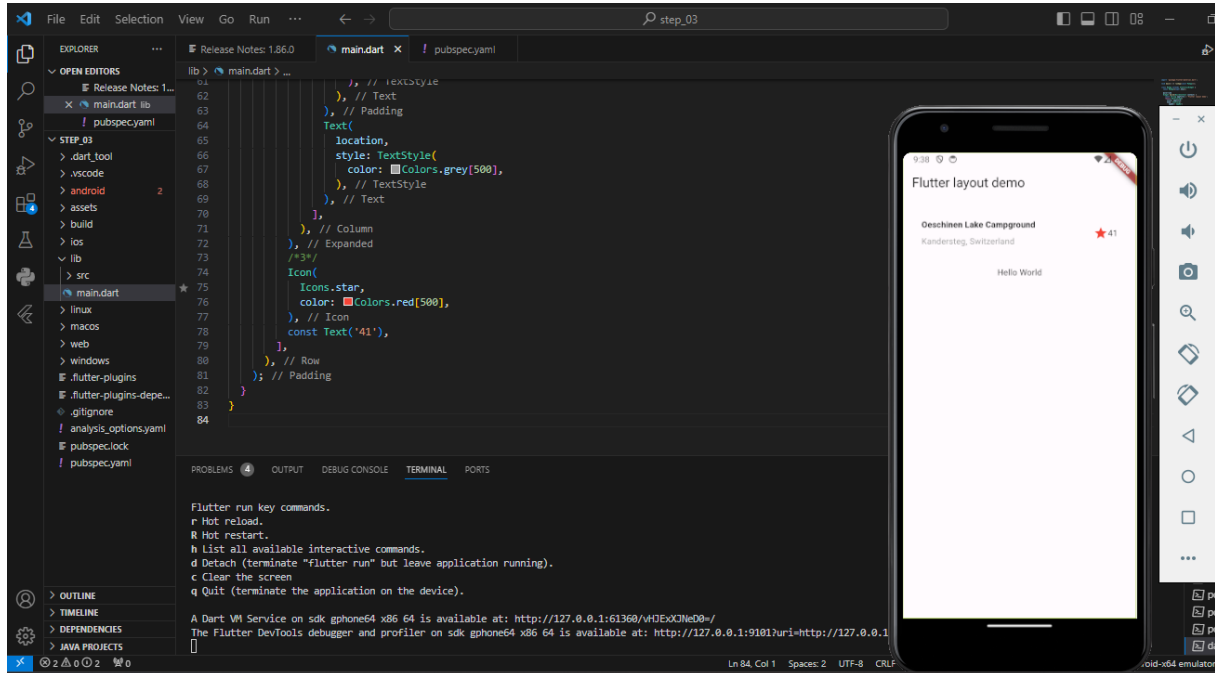


D. Build a Flutter layout

Ce code crée une application Flutter avec une fenêtre contenant une barre d'applications ayant le titre "Flutter layout demo" et un corps affichant le texte "Hello World" centré. L'output visuel sera une interface utilisateur simple avec ces éléments.

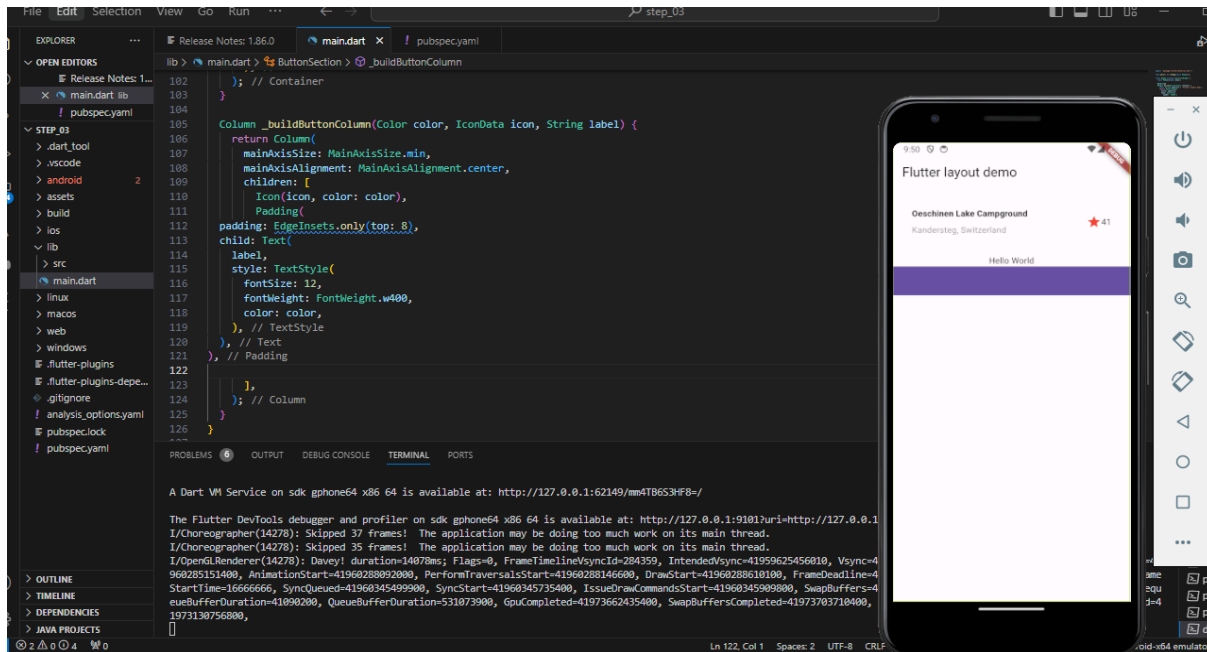


Avant le texte "Hello World", l'application affiche maintenant une section de titre avec le nom "Oeschinen Lake Campground" et l'emplacement "Kandersteg, Switzerland". La section de titre contient également le texte "41" et une icône d'étoile rouge.

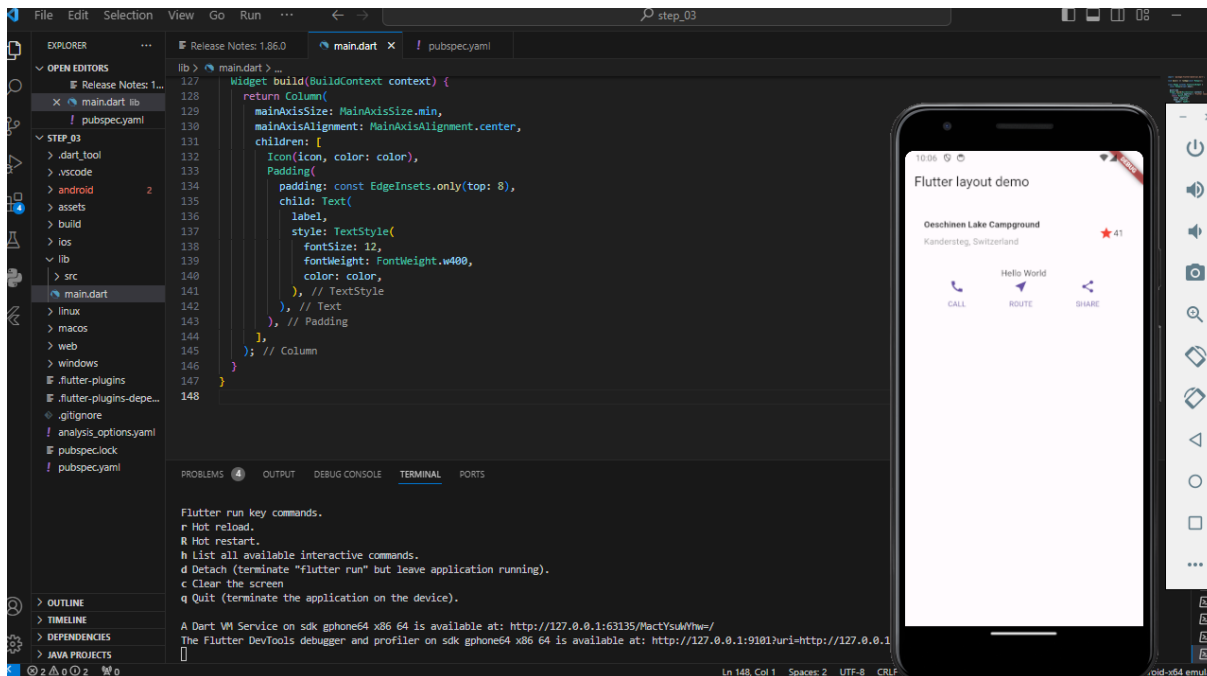


L'application affiche maintenant une section de boutons sous le texte "Hello World". Trois boutons, "CALL", "ROUTE" et "SHARE", sont présents dans cette section de boutons.

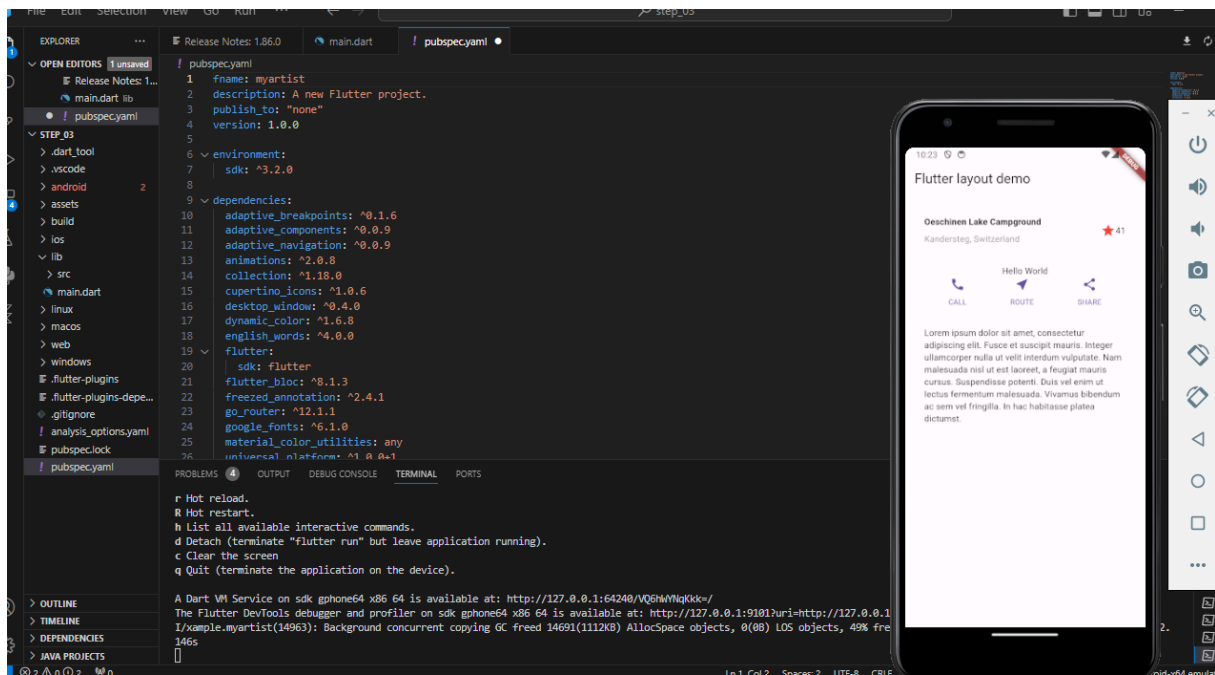
Avec cette modification, la classe ButtonSection utilise désormais la classe ButtonWithText récemment créée pour créer des boutons contenant des icônes et du texte. Après avoir intégré ces modifications, vous pouvez lancer votre application.



La structure générale de l'application est conservée avec ces modifications, car la classe ButtonSection utilise maintenant un SizedBox pour englober les boutons.



L'interface utilisateur a maintenant une nouvelle section de texte (TextSection) grâce à ces modifications.



Cette modification a ajouté une nouvelle section d'image à l'interface utilisateur.

