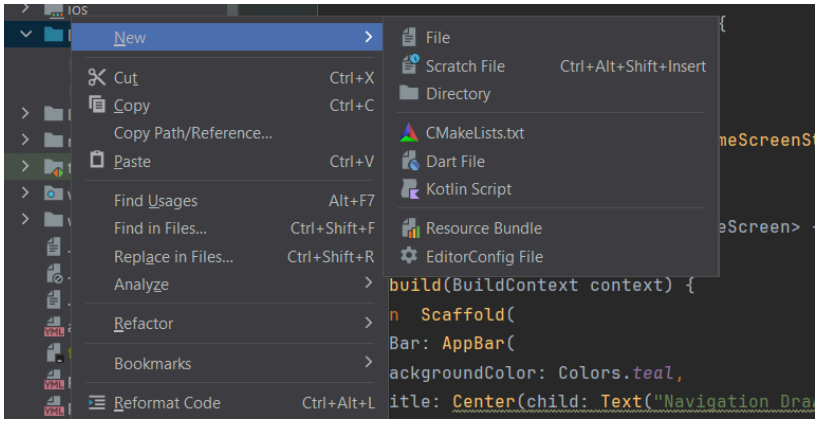
chapter 5

creating a new file and using it in main.dart
right click on lib → new → dart file



then write stf in new file and and create a stateful widget by the name HomeScreen()
import it in main.dart

// home_screen.dart

```
import 'package:flutter/material.dart';
//allows to create widgets


//creating a widget HomeScreen
class HomeScreen extends StatefulWidget {
 const HomeScreen({super.key});

 @override
 State<HomeScreen> createState() => _HomeScreenState();
}

class _HomeScreenState extends State<HomeScreen> {
 @override
 Widget build(BuildContext context) {
  return  Scaffold(
   appBar: AppBar(
    backgroundColor: Colors.teal,
    title: Center(child: Text("Navigation Drawer", style: TextStyle(color: Colors.white),)),
   ),
   body: Column(
    mainAxisAlignment: MainAxisAlignment.center
    ,crossAxisAlignment: CrossAxisAlignment.center,
    children: [
     //text button has a property onpressed that accept
    TextButton(
     onPressed: (){

     },
     child: Text('Screen 1'),
    )
```

```
      ],
    )
   );
  }
}
```

//main.dart

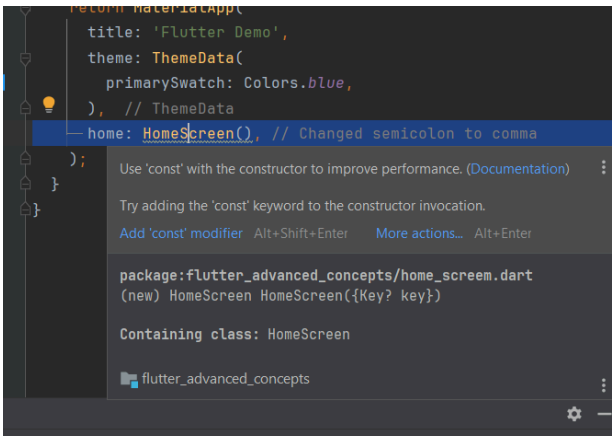
```
import 'package:flutter/material.dart';
import 'package:flutter_advanced_concepts/home_screem.dart';

void main() {
 runApp(const MyApp());  //app runs from here

}
//then it comes to this widget where it gets material app
class MyApp extends StatelessWidget {
 const MyApp({super.key});

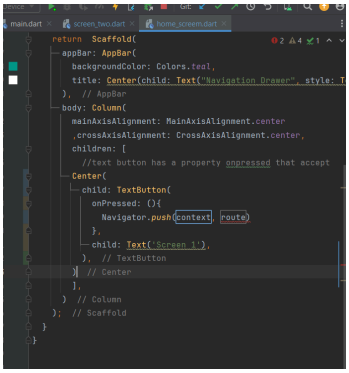
 @override
 Widget build(BuildContext context) {
  return MaterialApp(
    title: 'Flutter Demo',
    theme: ThemeData(
      primarySwatch: Colors.blue,
    ),
    home: HomeScreen(), // Changed semicolon to comma
   );
 }
}
```

also call the home_screen dart file widget “homeScreen()” in **home:**
for importing the correct file path of new home_screen file hover over HomeScreen() and copy the path directly



How to navigate to screen two after pressing?

pass Navigator.push(___,___)



then=>

```
crossAxisAlignment: CrossAxisAlignment.center,
hildren: [
 //text button has a property onpressed that accept
enter(
child: TextButton(
 onPressed: (){
   Navigator.push(context, MaterialPageRoute(builder: (context)=> ScreenTwo()));
  },
 child: Text('Screen 1'),
),  // TextButton
 // Center
,
// Column
/ Scaffold
```

navigator.push() vs navigator.pop()

In Flutter, **`Navigator.push()`** and **`Navigator.pop()`** are used to manage the navigation stack, allowing you to move between different screens or pages in an app. **`Navigator.push()`** is used to add a new route (screen) onto the navigation stack, effectively displaying a new page on top of the current one. This is typically used when navigating forward in the app, like opening a new screen. On the other hand, **`Navigator.pop()`** is used to remove the topmost route from the navigation stack, which takes the user back to the previous screen. This is commonly used when you want to close the current screen and return to the previous one, essentially navigating backward.

Another method is **routes** for moving from one screen to another(routing, easy way of shifting towards any screen)
There are two primary methods of routing in Flutter:

TYPES OF ROUTING

1. **Named Routing:**

- Named routing involves defining routes with unique string identifiers and managing navigation using these names. In this approach, you define a map of route names to widget builders in the `MaterialApp` widget's `routes` property. This method is convenient for managing complex navigation flows, as routes are centrally defined and can be easily referenced by name throughout the app.

MaterialApp(

```
  initialRoute: '/',
  routes: {
    '/': (context) => HomeScreen(),
    '/screen2': (context) => ScreenTwo(),
  },
);


// Navigating to another screen
Navigator.pushNamed(context, '/screen2');
```

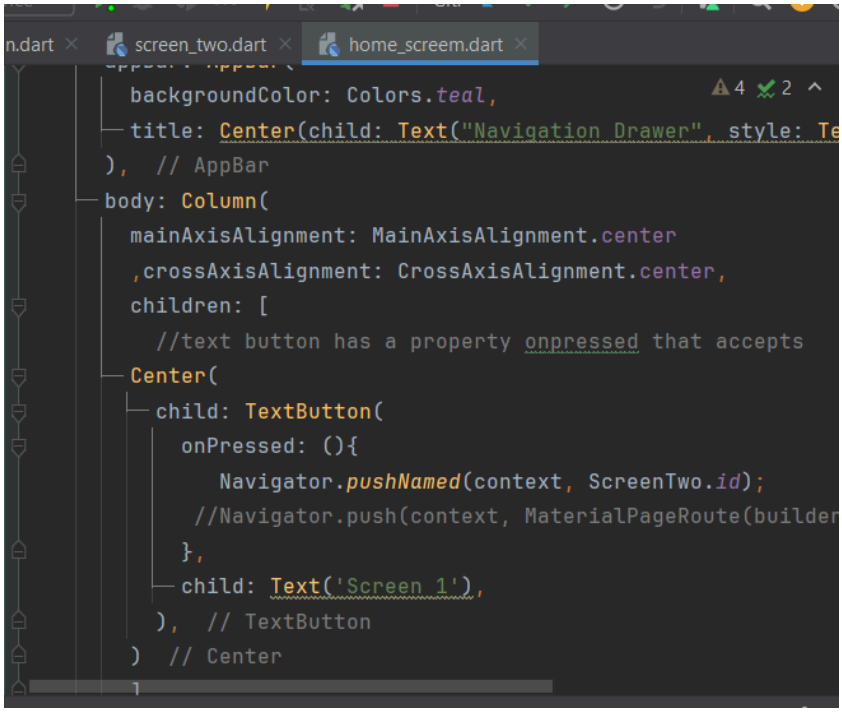
2. **Direct Routing (Imperative Routing):**

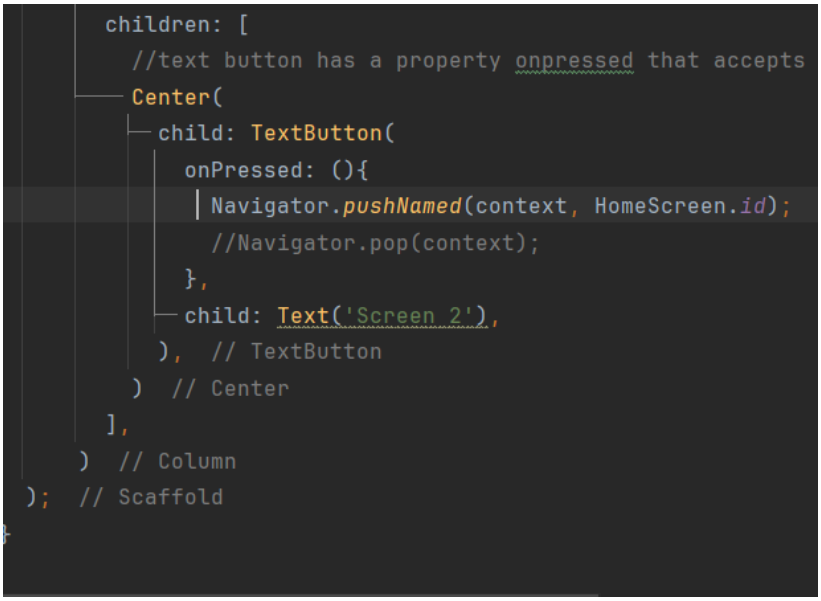
- Direct routing, also known as imperative routing, uses `Navigator` methods such as `push()`, `pop()`, and `pushReplacement()` to manage the navigation stack. This approach allows you to pass data between screens and offers more control over the navigation process. However, it can become cumbersome in large apps due to the need to manage routes individually within widgets.

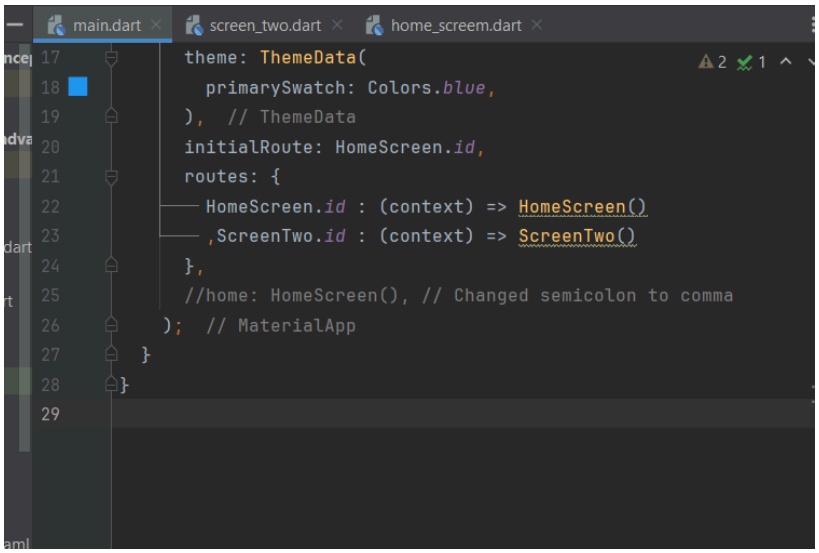
```
// Navigating to another screen
Navigator.push(context, MaterialPageRoute(builder: (context) => ScreenTwo()));


// Going back to the previous screen
Navigator.pop(context);
```

NAMED ROUTING:



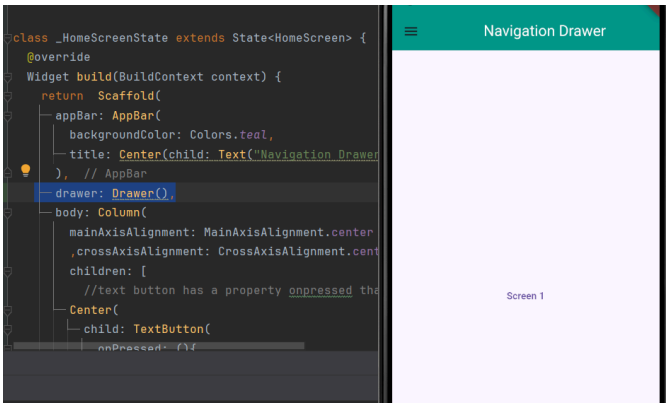


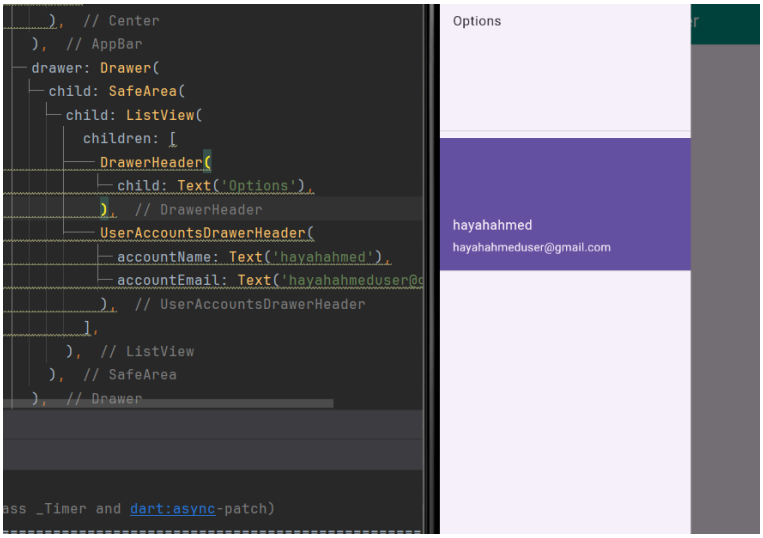


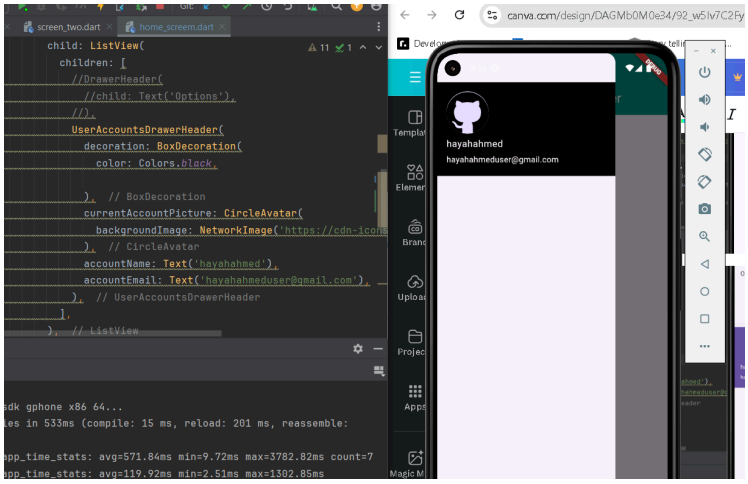
Drawer widget

The Drawer widget in Flutter is used to create a slide-in menu that typically contains navigation links or other options for users. It is often used in conjunction with the Scaffold

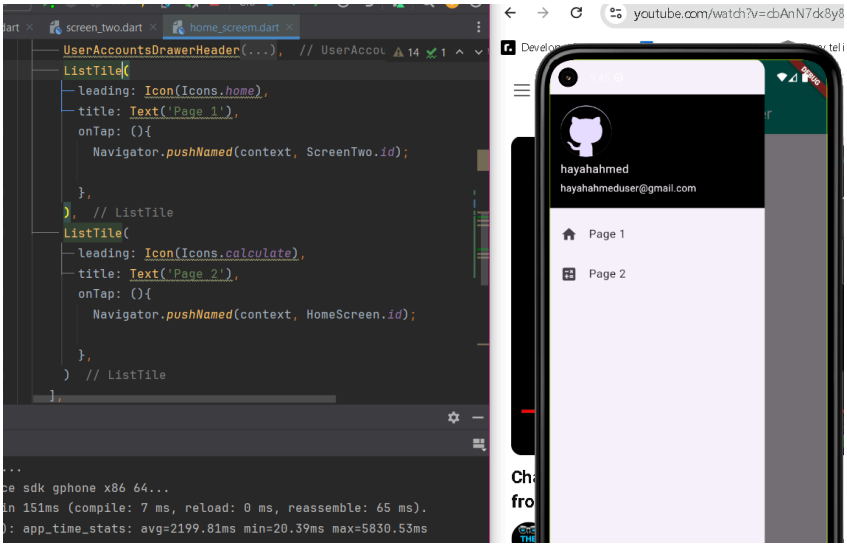
widget, where it provides an accessible way for users to navigate through different sections of an app.







using list tile and on press property which helps in routing/navigation



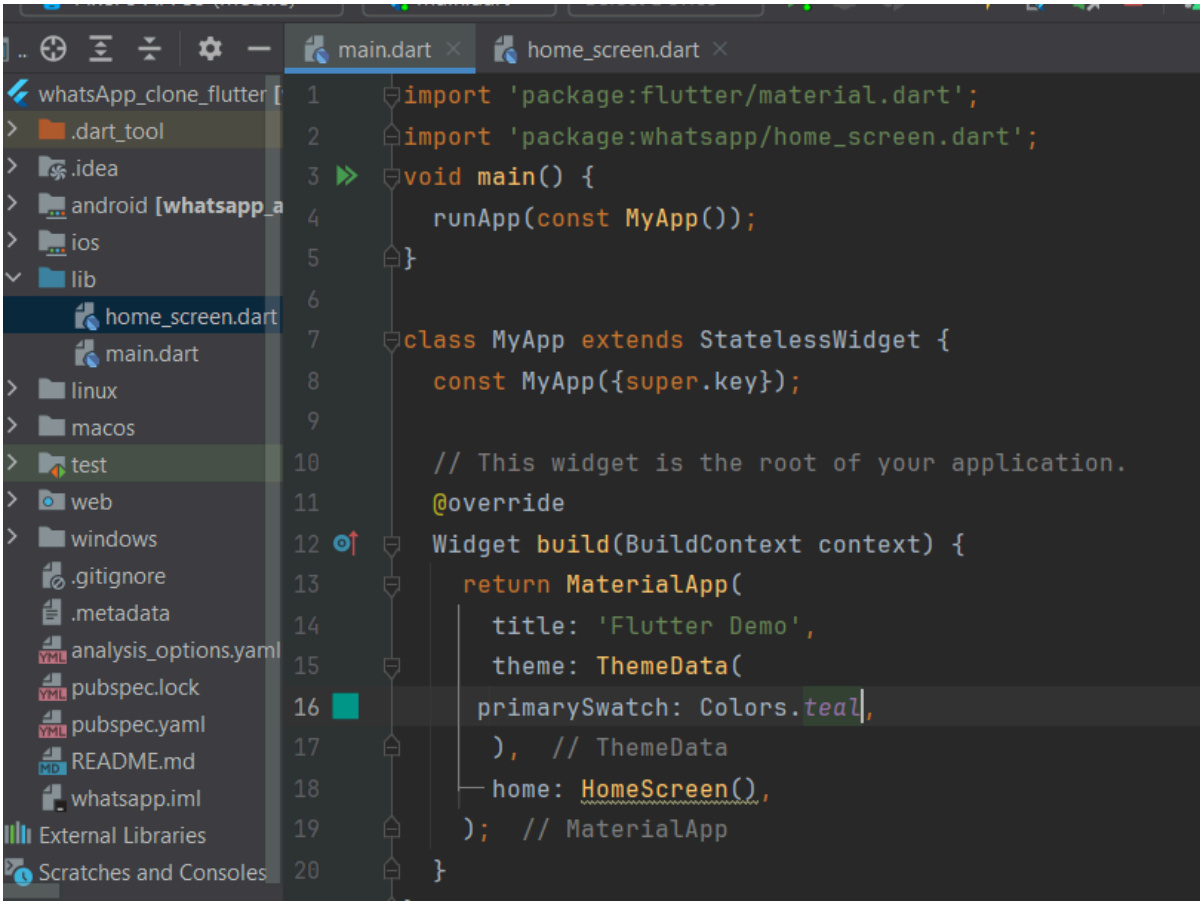
chapter 6

BUILDING WHATSAPP UI ON FLUTTER

Step 1=> Creating separate page for homescreen and linking it with main.dart

imports in main.dart

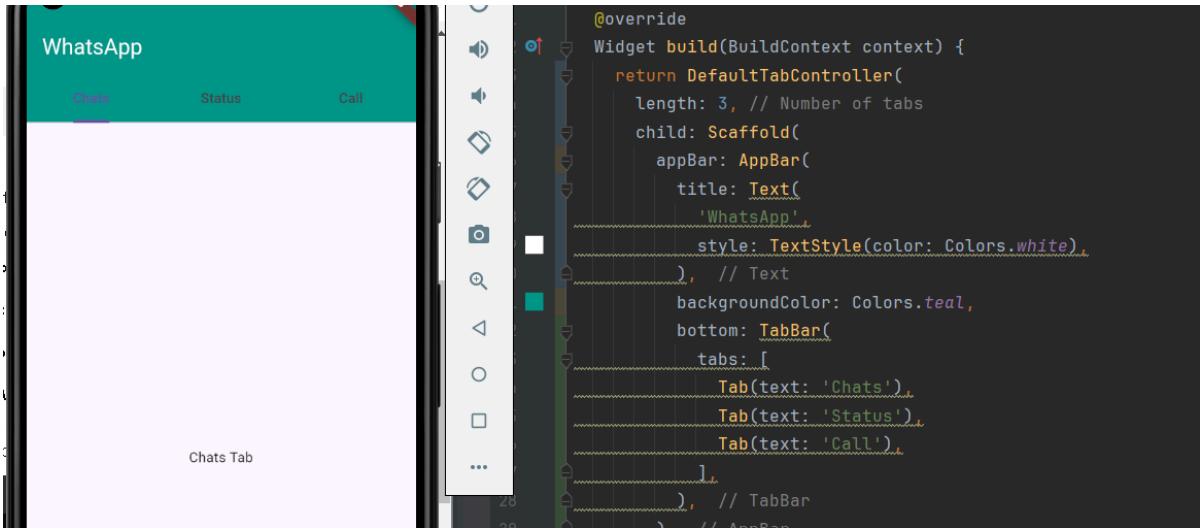
```
import 'package:flutter/material.dart';
import 'package:whatsapp/home_screen.dart';
```



Designing Home_Screen.dart

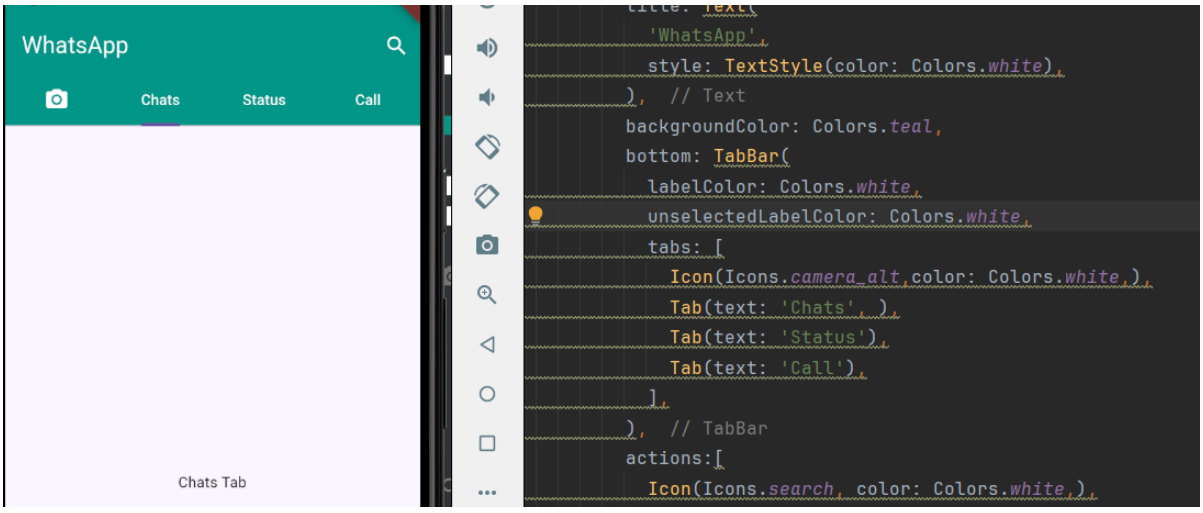
The Scaffold widget in Flutter has several key properties for creating a layout:

- **appBar**: This property defines the top app bar of the screen. It is a widget that typically contains the title and actions for the app. The appBar property can be customized with a variety of child widgets, including AppBar.
- **AppBar**: Inside the AppBar, you can set the title property to define the main text or widget displayed in the app bar. The AppBar also has a bottom property, which is used to add a widget such as TabBar at the bottom of the app bar.
- **bottom**: This property of the AppBar is used to add widgets like TabBar, which allows for creating a tabbed interface. The TabBar widget typically contains multiple Tab widgets, each representing a different tab in the interface.
- **body**: This property of the Scaffold defines the main content area of the screen. To display different content for each tab, use the TabBarView widget. The TabBarView accepts a list of widgets via its children property, where each widget corresponds to the content displayed when a tab is selected.



Changed the tab bar color to white & added search bar

search bar is added by using actions property



This setup allows the application to present additional options to the user in a compact and visually accessible way,

In the provided Flutter code snippet and screenshot, three options—**New group**, **Settings**, and **Logout**—are displayed in a popup menu within the app. Here's how each aspect of the

code works to achieve this:

1. **PopupMenuButton Widget:**

- **Definition:** The PopupMenuButton is a widget in Flutter that displays a menu when tapped.
- **Icon:** It is associated with an icon using icon: const Icon(Icons.more_vert_outlined, color: Colors.white) which is the three vertical dots icon (more_vert_outlined), commonly used to represent additional options in mobile apps.
- **Trigger:** Tapping this icon triggers the popup menu to appear.

2. **itemBuilder:**

- **Function:** This is a required parameter of the PopupMenuButton. It returns a list of PopupMenuItem widgets that will be shown in the popup menu.
- **Implementation:** In the code, the itemBuilder function creates three PopupMenuItemwidgets, each representing a different menu option:
  - **New group**: PopupMenuItem(value: 0, child: Text("New group"))
  - **Settings**: PopupMenuItem(value: 1, child: Text("Settings"))
  - **Logout**: PopupMenuItem(value: 2, child: Text("Logout"))
- **Text Widgets:** Each PopupMenuItem contains a Text widget that specifies the label (e.g., "New group").

3. **Menu Display Logic:**

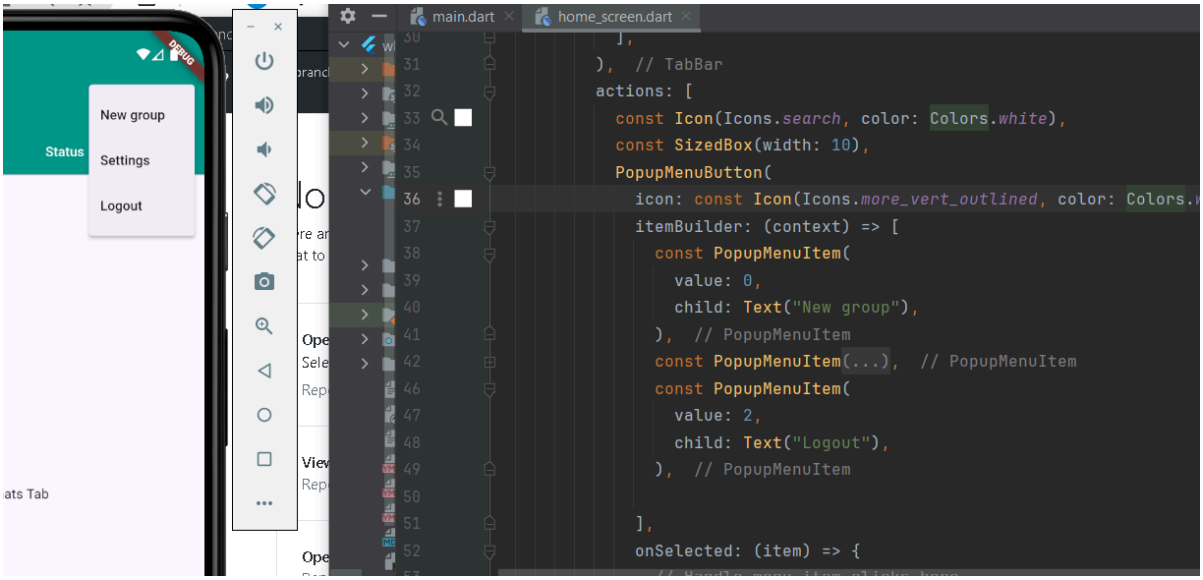
- **PopupMenuItems:** These items are automatically displayed in a column format within the popup when the user taps the icon. Each item is listed vertically with a slight padding and margin to create separation between the options.
- **Styling:** The popup menu appears with a light background and a slight shadow to make it stand out against the rest of the UI, as seen in the screenshot.

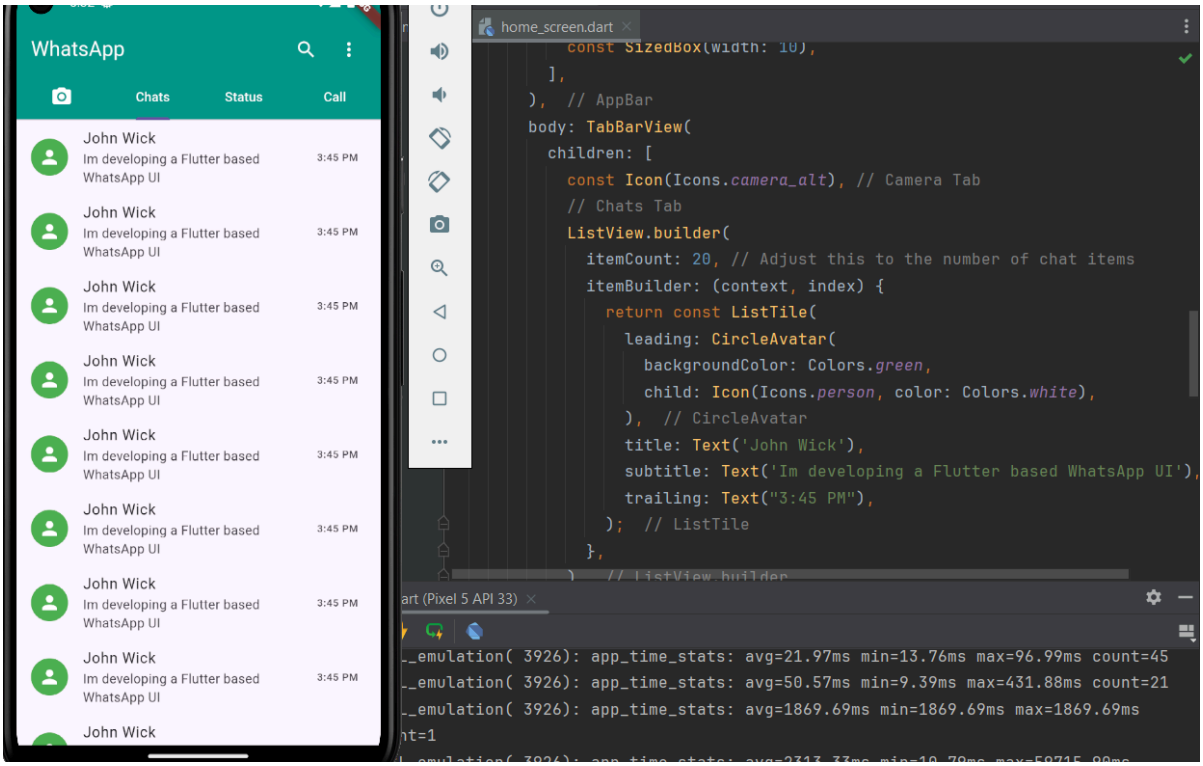
4. **Action Handling:**

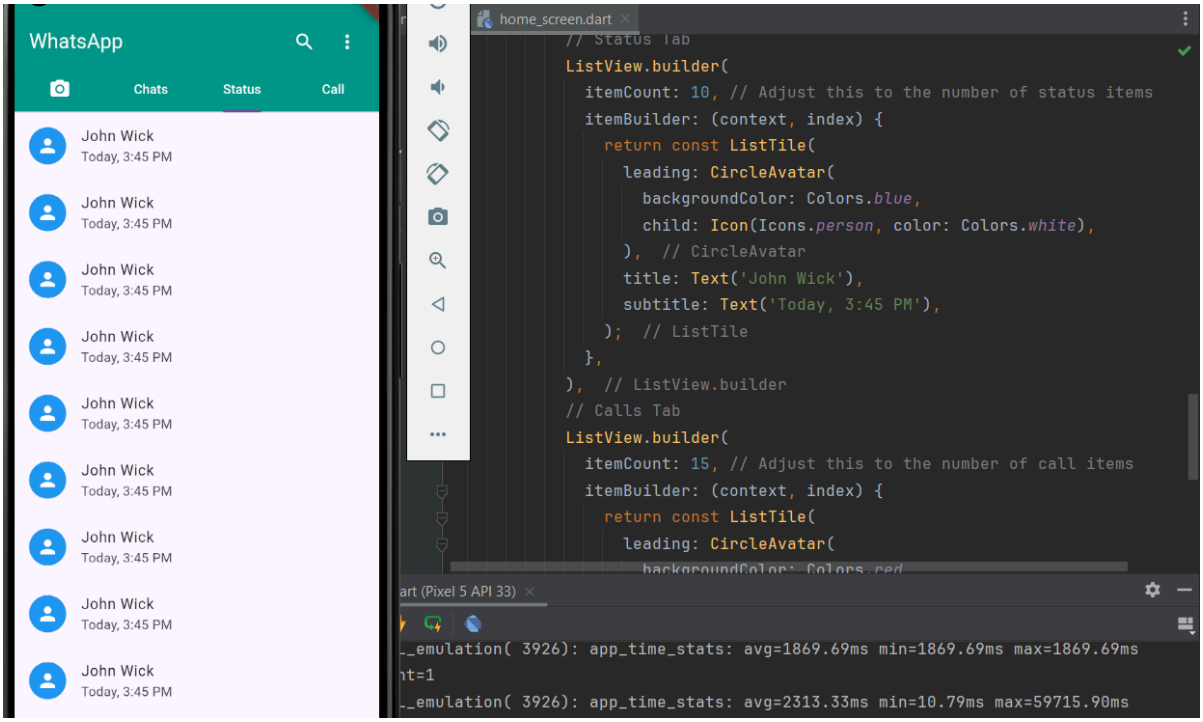
- **onSelected:** This property is not fully shown in the snippet but is typically used to handle the actions based on the user's selection from the popup menu.
- **Value Assignment:** Each PopupMenuItem has a value parameter that helps identify which item was selected. For example, value: 0 for "New group," value: 1 for "Settings," and value: 2 for "Logout."

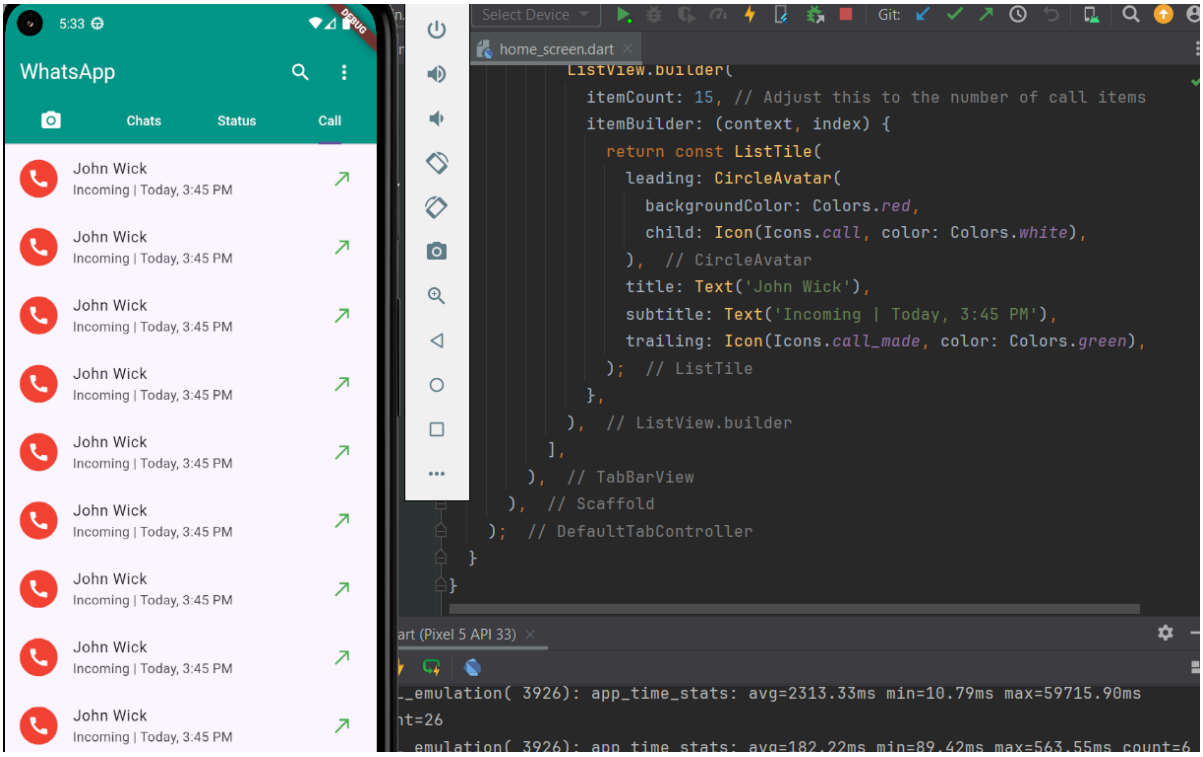
5. **Positioning:**

- **Context of Display:** The PopupMenuButton is placed within the actions array of the AppBar, so it appears on the top-right corner of the screen, as is standard for menus triggered by an options icon.


DEBUG
Status
New group
Settings
Logout
main.dart
home_screen.dart
), // TabBar
actions: [
const Icon(Icons.search, color: Colors.white),
const SizedBox(width: 10),
PopupMenuButton(
icon: const Icon(Icons.more_vert_outlined, color: Colors.
itemBuilder: (context) => [
const PopupMenuItem(
value: 0,
child: Text("New group"),
), // PopupMenuItem
const PopupMenuItem(...), // PopupMenuItem
const PopupMenuItem(
value: 2,
child: Text("Logout"),
), // PopupMenuItem
],
onSelected: (item) => {


WhatsApp
Chats
Status
Call
John Wick
Im developing a Flutter based WhatsApp UI
3:45 PM
home_screen.dart
const SizedBox(width: 10),
], // AppBar
body: TabBarView(
children: [
const Icon(Icons.camera_alt), // Camera Tab
// Chats Tab
ListView.builder(
itemCount: 20, // Adjust this to the number of chat items
itemBuilder: (context, index) {
return const ListTile(
leading: CircleAvatar(
backgroundColor: Colors.green,
child: Icon(Icons.person, color: Colors.white),
), // CircleAvatar
title: Text('John Wick'),
subtitle: Text('Im developing a Flutter based WhatsApp UI'),
trailing: Text("3:45 PM"),
); // ListTile
},
art (Pixel 5 API 33)
_emulation(3926): app_time_stats: avg=21.97ms min=13.76ms max=96.99ms count=45
_emulation(3926): app_time_stats: avg=50.57ms min=9.39ms max=431.88ms count=21
_emulation(3926): app_time_stats: avg=1869.69ms min=1869.69ms max=1869.69ms





chapter 7

Simulators and Emulators

- **iOS Simulator**: A tool provided by Xcode to test iOS apps on a simulated iOS device.
- **Android Emulator**: A virtual device provided by Android Studio to test Android apps on a simulated Android device.

Pub.dev Overview

- **What is Pub.dev?**

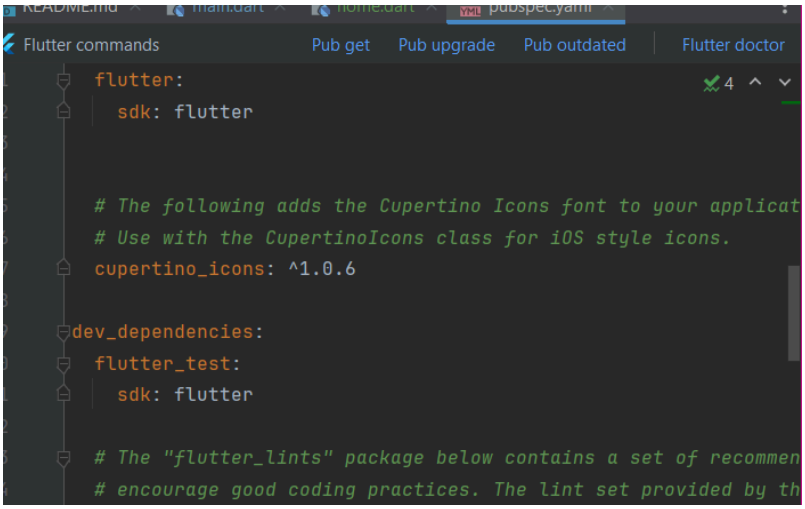
- **Definition**: Pub.dev is an online repository and community for Dart and Flutter packages.
- **Purpose**: It allows developers to upload, discover, and use packages to simplify development tasks.
- **Management**: It is designed, launched, and managed by Google.

Dependencies/Packages:

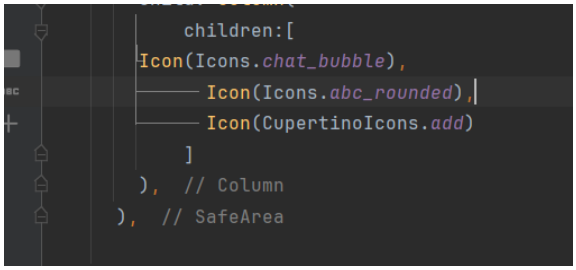
- **Definition**: Dependencies or packages are libraries that provide additional functionality to your application. They are defined in the pubspec.yaml file.
- **Usage**: These packages can be fetched and included in your code to extend your app's features. For example, the cupertino_icons package provides iOS-style icons, which can be replaced with apple_icons if needed.

pubspec.yaml File

- **Purpose**: The pubspec.yaml file is used to define your project's dependencies, packages, and other configurations.
- **Dependencies Section**: Lists the packages your project depends on. It specifies the package name and version constraints.

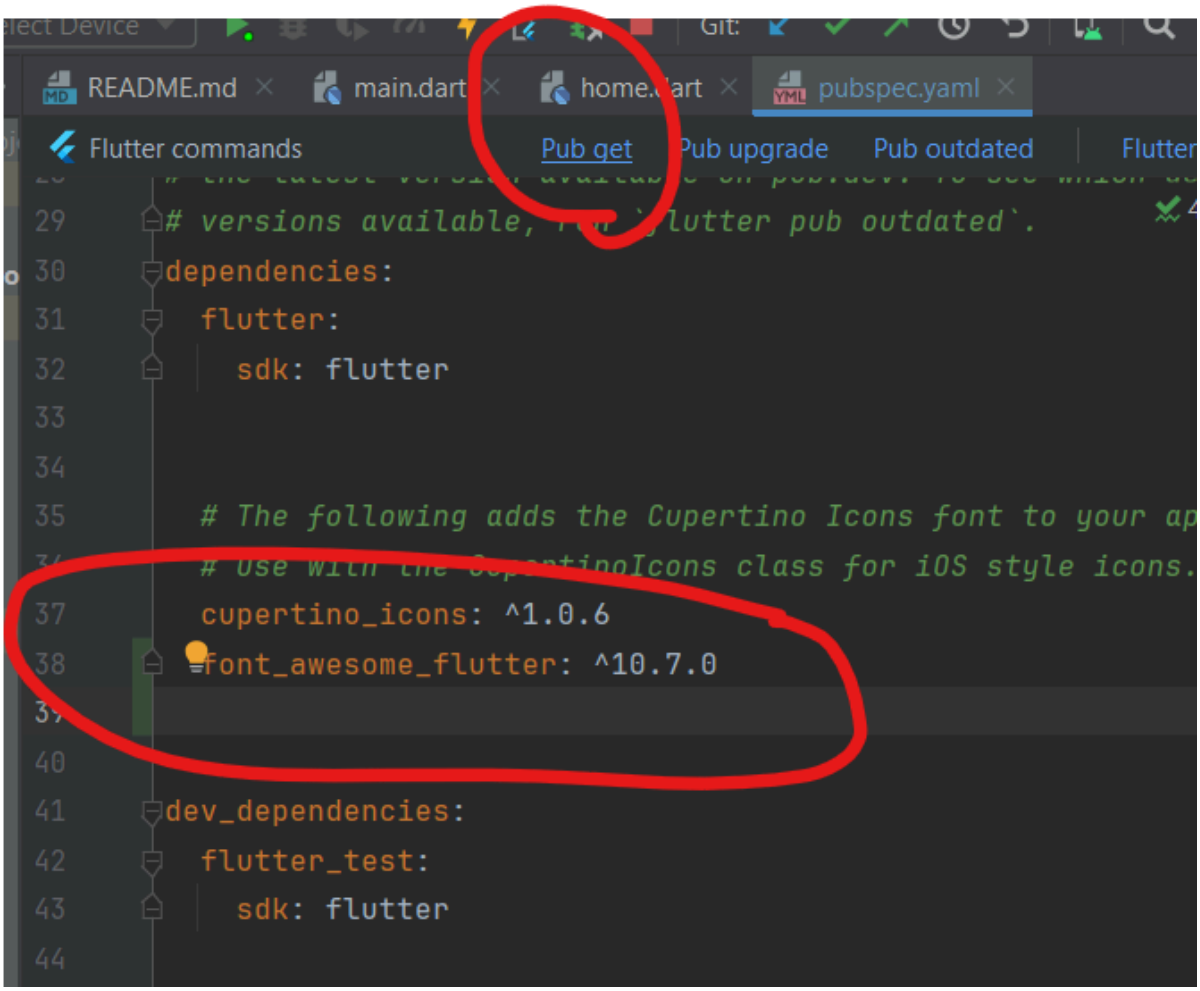


How to use Cupertino Icons

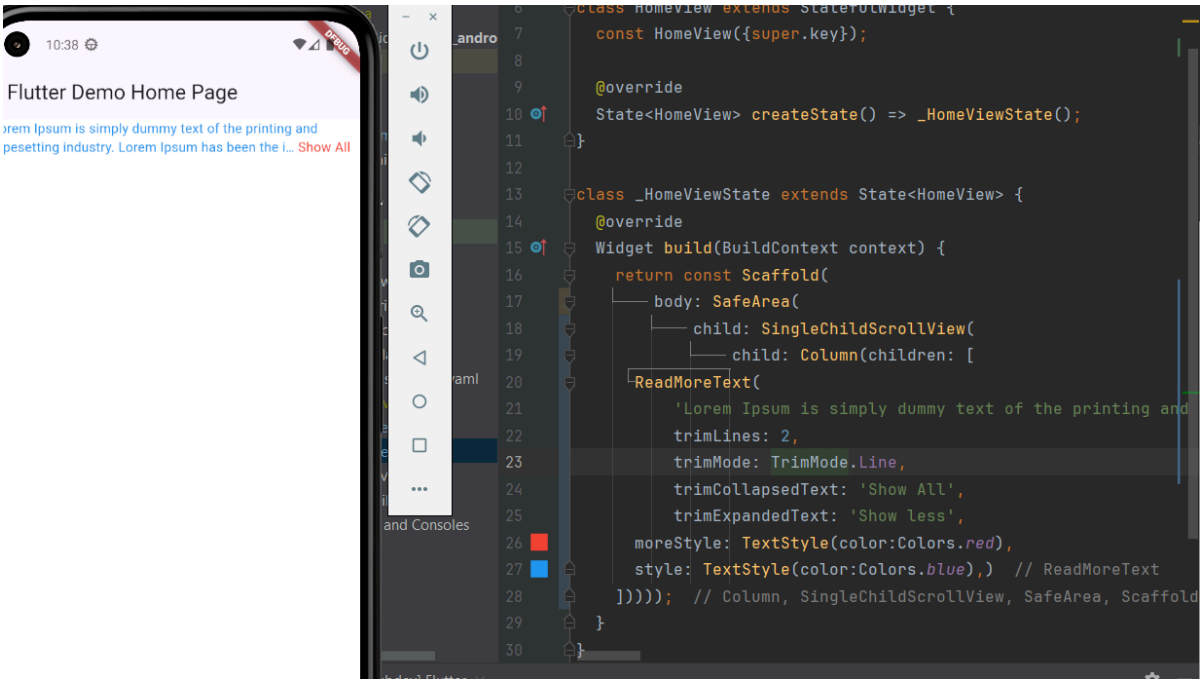


Using font_awesome

pub.dev



Readmore package:

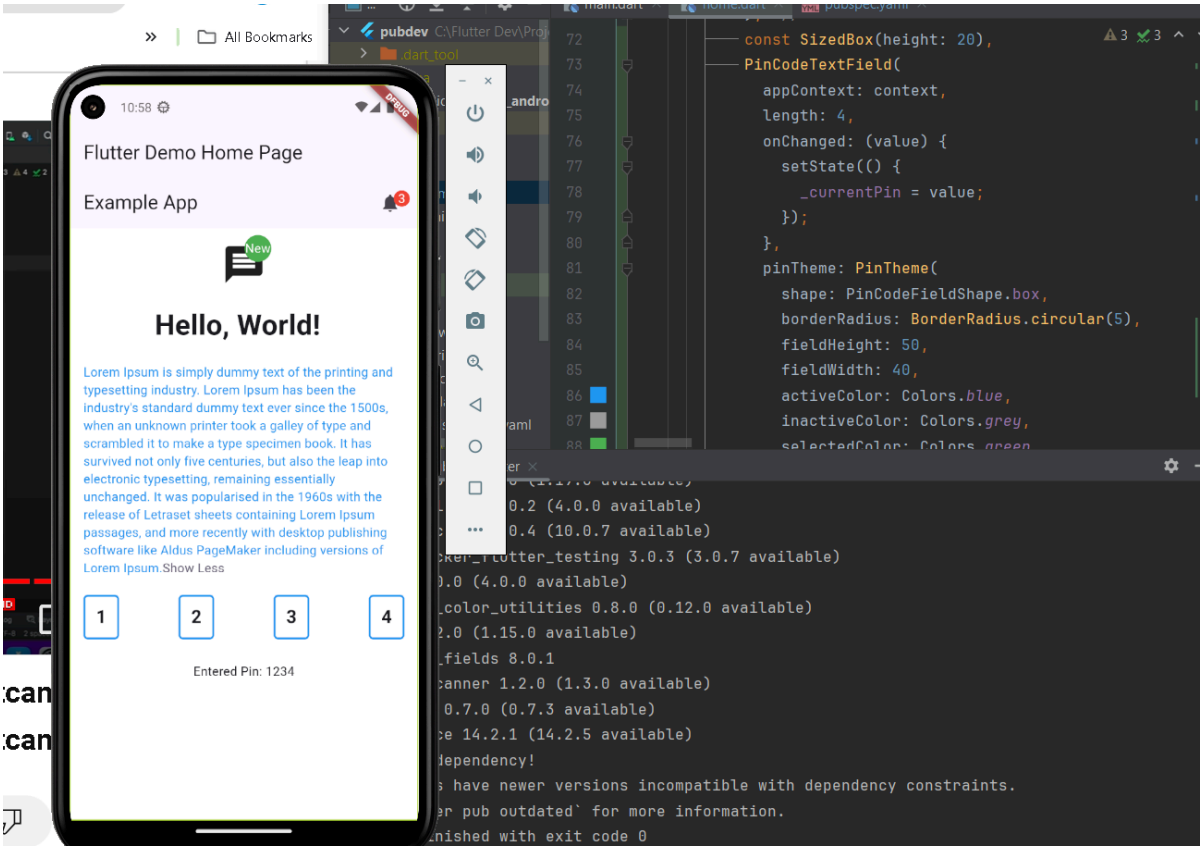


badge package:

animated text package

https://pub.dev/documentation/animated_text_kit/latest/

pincode package



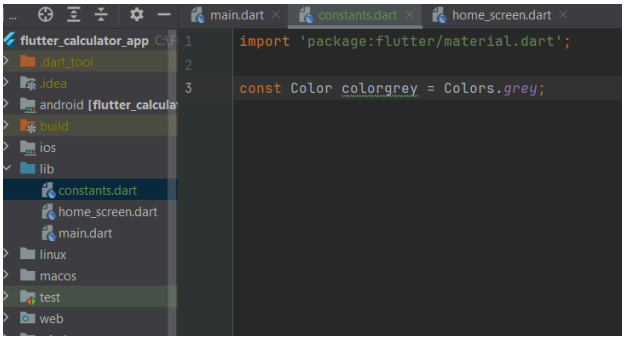
chapter 8

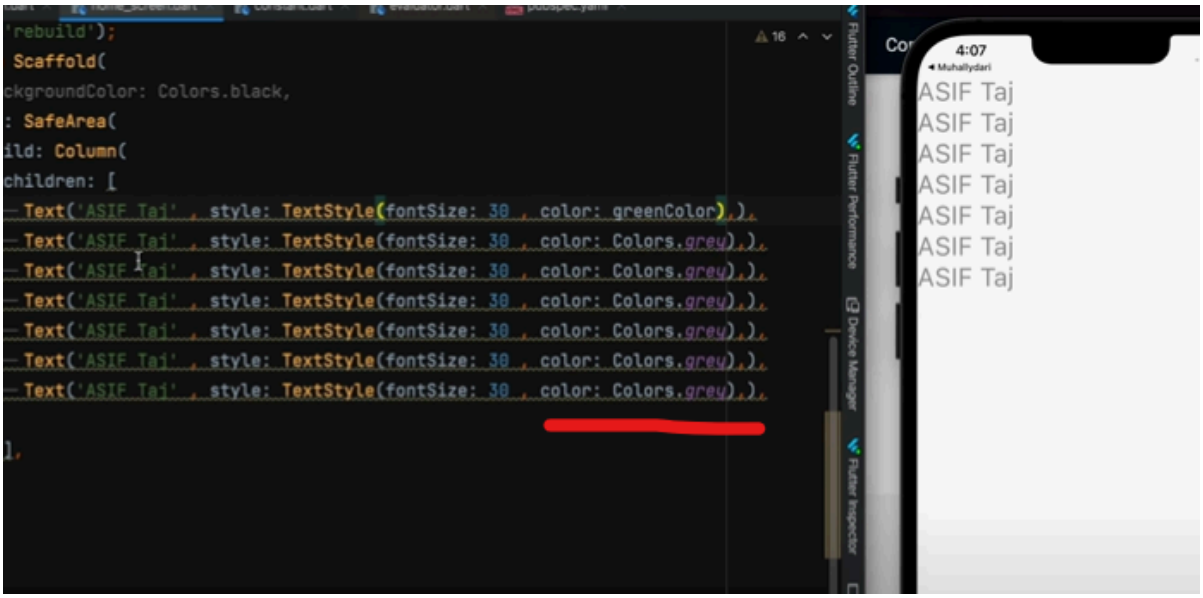
print() => prints on the console
variable declaration:
int num = 8;
if statement syntax:
if(num > num1){
print('greater')
}

Pre-requsites

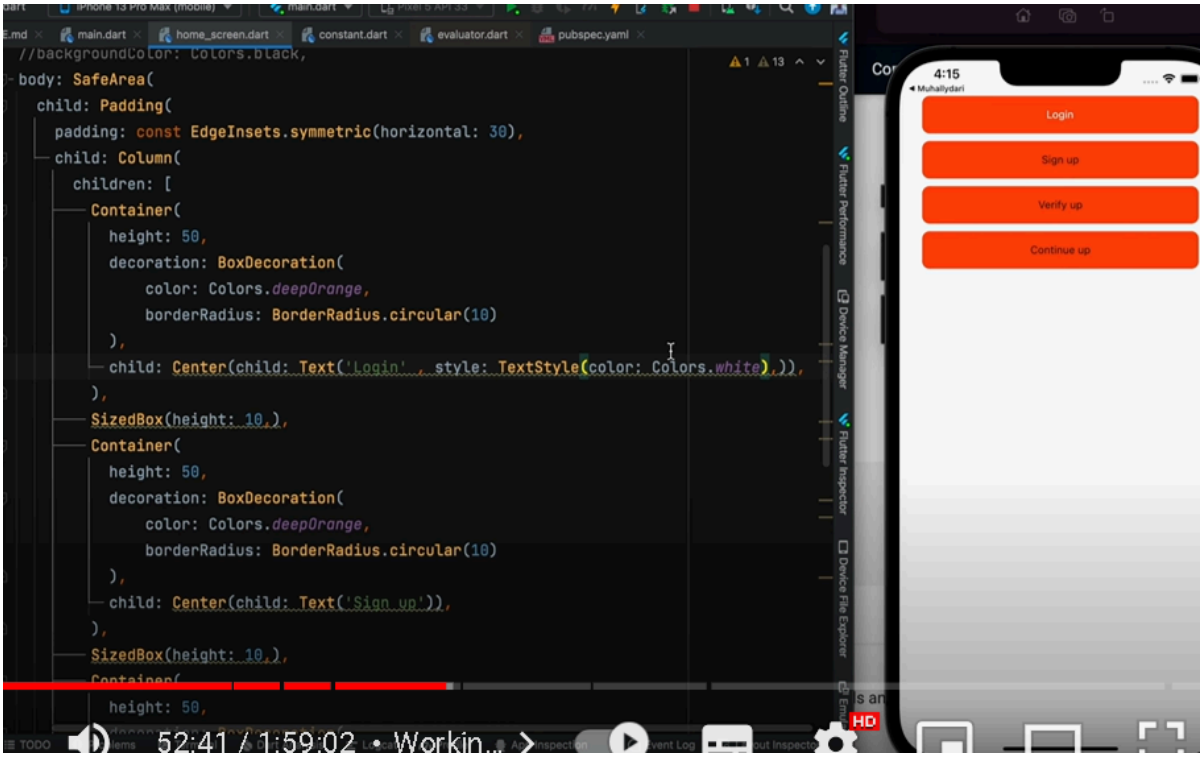
What are constants?

To eliminate the repetition of Colors.grey, you can create a new file, such as constants.dart, to define a constant for the color. For instance, you could define const myGrey = Colors.grey; in constants.dart and then import this file wherever needed. In your widget code, replace instances of Colors.grey with myGrey. This approach centralizes the color definitions, making your code cleaner and easier to maintain.
rightclick on lib-> then create new dart file





How to create reusable components?

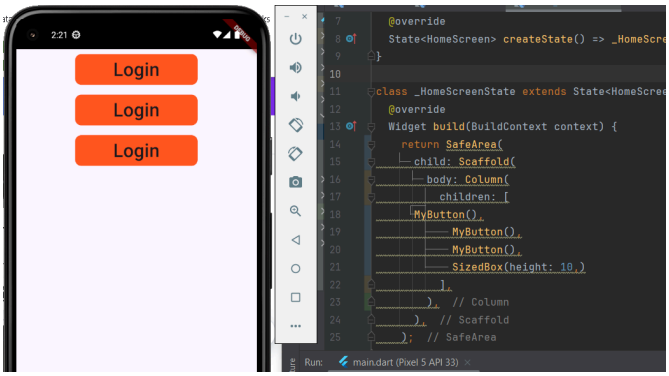


To eliminate repetition and promote reusability, we created a reusable component by defining a stateful widget named MyButton within the home.dart file. This widget can be called multiple times within the HomeScreen, allowing us to create consistent button instances throughout the app without duplicating code. By encapsulating the button's design and behavior in a single widget, we streamlined the code and made future modifications easier to manage.

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

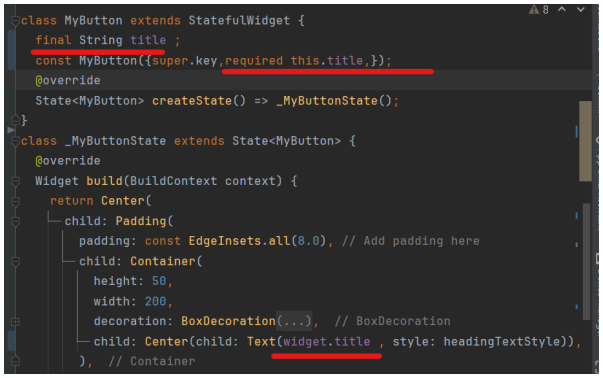
  @override
  State<MyButton> createState() => _MyButtonState();
}

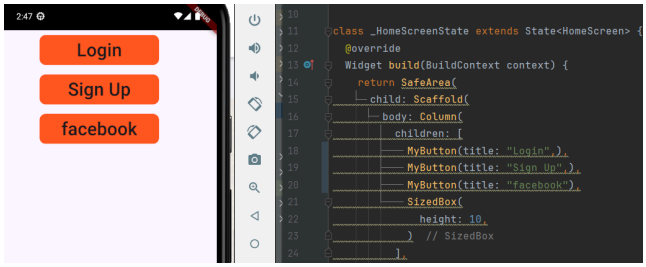
class _MyButtonState extends State<MyButton> {
  @override
  Widget build(BuildContext context) {
    return Center(
      child: Padding(
        padding: const EdgeInsets.all(8.0),  // Add padding here
        child: Container(...),  // Container
      ),  // Padding
    );  // Center
  }
```



If you want to change only the text while keeping the rest of the button's design the same, you can follow these steps: In the MyButton class, define a title variable using final String title; The final keyword ensures that this variable is immutable once assigned. In the MyButton constructor, include required this.title, which maps the input title to the widget's title property. In the Text widget, replace the static text with widget.title to dynamically

display the provided title. Finally, when calling MyButton, you can specify different titles, to reuse the same button design with different text. MyButton(title: "Login",),
MyButton(title: "Sign Up",),
MyButton(title: "facebook"),





Building the Calculator UI

In this project, the first step was to build a reusable component named `MyButton` in a separate Dart file. The purpose of this component was to create consistent, stylized buttons that would be used throughout the calculator interface. The `MyButton` widget was defined with a few key properties: `title`, `color`, and `onPress`.

The `title` property was a required string that defined the label displayed on the button, while `color` was an optional property that allowed customization of the button's background color, with a default value of `Color(0xffa5a5a5)`. The `onPress` property was also required and accepted a callback function, which would be executed whenever the button was pressed.

The button itself was constructed using an `InkWell` widget, which provided the ripple effect when the button was tapped. Inside the `InkWell`, a `Container` was used to define the size and shape of the button, specifically setting the shape to a circle and the height to 80 pixels. The text label was centered within the button using a `Center` widget, with the text style set to white and a font size of 20 pixels. This reusable component ensured that all buttons in the calculator would have a uniform appearance and behavior.

After creating the `MyButton` widget, the next step was to integrate these buttons into the `HomeScreen` widget, which served as the main interface of the calculator. The `HomeScreen` was a stateful widget, holding two key variables: `userInput` and `answer`, which stored the current input and the calculated result, respectively.

The `build` method of `HomeScreen` was designed to construct the UI, consisting of a `Column` with two main sections. The first section was an `Expanded` widget containing the display area for `userInput` and `answer`, aligned to the bottom-right of the screen using `Align` and `TextAlign.right`. This ensured that the numbers and results appeared in a typical calculator layout, with the input shown above the result.

The second section of the `Column` was another `Expanded` widget, which contained the grid of buttons representing digits, operators, and actions like `AC` (clear), `DEL` (delete), and `=` (equals). Each button was implemented using the previously defined `MyButton` component. For example, the `AC` button was programmed to clear both `userInput` and `answer`, resetting the calculator's state, while the `DEL` button removed the last character from the `userInput` string.

The logic for handling mathematical operations was implemented in the `equalPress` method, which utilized the `math_expressions` package. When the `=` button was pressed, the `equalPress` method was called to parse and evaluate the mathematical expression stored in `userInput`. The result was then displayed in the `answer` field.

Additionally, the `addOperator` method was implemented to handle the input of operators like `+`, `-`, `*`, `/`, and `%`. This method ensured that operators were not duplicated consecutively in the input, replacing the last operator with the new one if necessary.

Overall, the project was approached methodically by first building reusable components, integrating them into the UI, and then applying the necessary logic to create a functional calculator app. Each part of the code was designed to be modular and maintainable, making it easier to expand or modify the app in the future.

my_button.dart

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

class MyButton extends StatefulWidget {
 final String title;
 final Color color;
 final VoidCallback onPress;

 const MyButton({
  super.key,
  required this.title,
  this.color = const Color(0xffa5a5a5),
  required this.onPress,
 });
```

```
  @override
  State<MyButton> createState() => _MyButtonState();
}

class _MyButtonState extends State<MyButton> {
  @override
  Widget build(BuildContext context) {
    return Expanded(
      child: InkWell(
        onTap: widget.onPress, // Fixed the access to the onPress callback
        child: Container(
          height: 80,
          decoration: BoxDecoration(
            shape: BoxShape.circle,
            color: widget.color,
          ),
          child: Center(
            child: Text(
              widget.title,
              style: const TextStyle(fontSize: 20, color: Colors.white),
            ),
          ),
        ),
      ),
    );
  }
}
```

home_screen.dart

```
import 'package:flutter/material.dart';
import 'package:flutter_calculator_app/components/my_button.dart';
import 'package:math_expressions/math_expressions.dart';

class HomeScreen extends StatefulWidget {
  HomeScreen({super.key});
  var userInput = '';
  var answer = '';

  @override
  State<HomeScreen> createState() => _HomeScreenState();
}

class _HomeScreenState extends State<HomeScreen> {
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      backgroundColor: Colors.black,
      body: SafeArea(
        child: Padding(
          padding: const EdgeInsets.symmetric(horizontal: 10),
          child: Column(
            children: [
              Expanded(
                child: Padding(
```

```
    padding: const EdgeInsets.symmetric(vertical: 20),
    child: Align(
     alignment: Alignment.bottomRight,
     child: Column(
      mainAxisAlignment: MainAxisAlignment.end,
      crossAxisAlignment: CrossAxisAlignment.end,
      children: [
       Text(
        widget.userInput.toString(),
        style: TextStyle(fontSize: 30, color: Colors.white),
        textAlign: TextAlign.right,
       ),
       SizedBox(height: 10), // Space between userInput and answer
       Text(
        widget.answer.toString(),
        style: TextStyle(fontSize: 40, color: Colors.white, fontWeight: FontWeight.bold),
        textAlign: TextAlign.right,
       ),
      ],
     ),
    ),
   ),
  ),
 ),
 Expanded(
  flex: 2,
  child: Column(
   children: [
    Padding(
     padding: const EdgeInsets.only(bottom: 10.0),
     child: Row(
      children: [
       MyButton(
        title: 'AC',
        onPress: () {
         widget.userInput = '';
         widget.answer = '';
         setState(() {});
        },
       ),
       MyButton(
        title: '+/-',
        onPress: () {
         widget.userInput += '+/-';
         setState(() {});
        },
       ),
       MyButton(
        title: '%',
        onPress: () {
         addOperator('%');
        },
       ),
       MyButton(
        title: '/',
        color: Color(0xffffa00a),
        onPress: () {
```

```
          addOperator('/');
        },
      ),
    ],
  ),
),
Padding(
  padding: const EdgeInsets.only(bottom: 10.0),
  child: Row(
    children: [
      MyButton(
        title: '7',
        onPress: () {
          widget.userInput += '7';
          setState(() {});
        },
      ),
      MyButton(
        title: '8',
        onPress: () {
          widget.userInput += '8';
          setState(() {});
        },
      ),
      MyButton(
        title: '9',
        onPress: () {
          widget.userInput += '9';
          setState(() {});
        },
      ),
      MyButton(
        title: '*',
        color: Color(0xffffa00a),
        onPress: () {
          addOperator('*');
        },
      ),
    ],
  ),
),
Padding(
  padding: const EdgeInsets.only(bottom: 10.0),
  child: Row(
    children: [
      MyButton(
        title: '4',
        onPress: () {
          widget.userInput += '4';
          setState(() {});
        },
      ),
      MyButton(
        title: '5',
        onPress: () {
          widget.userInput += '5';
```

```
          setState(() {});
        },
      ),
      MyButton(
        title: '6',
        onPress: () {
          widget.userInput += '6';
          setState(() {});
        },
      ),
      MyButton(
        title: '-',
        color: Color(0xffffa00a),
        onPress: () {
          addOperator('-');
        },
      ),
    ],
  ),
),
Padding(
  padding: const EdgeInsets.only(bottom: 10.0),
  child: Row(
    children: [
      MyButton(
        title: '1',
        onPress: () {
          widget.userInput += '1';
          setState(() {});
        },
      ),
      MyButton(
        title: '2',
        onPress: () {
          widget.userInput += '2';
          setState(() {});
        },
      ),
      MyButton(
        title: '3',
        onPress: () {
          widget.userInput += '3';
          setState(() {});
        },
      ),
      MyButton(
        title: '+',
        color: Color(0xffffa00a),
        onPress: () {
          addOperator('+');
        },
      ),
    ],
  ),
),
Row(
```

```
                children: [
                  MyButton(
                    title: '0',
                    onPress: () {
                      widget.userInput += '0';
                      setState(() {});
                    },
                  ),
                  MyButton(
                    title: '.',
                    onPress: () {
                      if (!widget.userInput.endsWith('.')) {
                        widget.userInput += '.';
                        setState(() {});
                      }
                    },
                  ),
                  MyButton(
                    title: 'DEL',
                    onPress: () {
                      if (widget.userInput.isNotEmpty) {
                        widget.userInput = widget.userInput
                            .substring(0, widget.userInput.length - 1);
                        setState(() {});
                      }
                    },
                  ),
                  MyButton(
                    title: '=',
                    color: Color(0xffffa00a),
                    onPress: () {
                      equalPress();
                      setState(() {});
                    },
                  ),
                ],
              ),
            ],
          ),
        ),
      ],
    ),
  ),
 );
}

void addOperator(String operator) {
 if (widget.userInput.isNotEmpty) {
  String lastChar = widget.userInput[widget.userInput.length - 1];
  if ('+-*/%'.contains(lastChar)) {
   widget.userInput =
       widget.userInput.substring(0, widget.userInput.length - 1);
  }
 }
 widget.userInput += operator;
```

```
    setState(() {});
  }

  void equalPress() {
    try {
      Parser p = Parser();
      Expression expression = p.parse(widget.userInput);
      ContextModel contextModel = ContextModel();

      double eval = expression.evaluate(EvaluationType.REAL, contextModel);
      widget.answer = eval.toString();
    } catch (e) {
      widget.answer = 'Error';
    }
    setState(() {});
  }
}
```

main.dart

```
import 'package:flutter/material.dart';
import 'package:flutter_calculator_app/home_screen.dart';
void main() {
  runApp(const MyApp());
}

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

  // This widget is the root of your application.
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'Flutter Demo',
      color: Colors.black,
      home: HomeScreen(),
    );
  }
}
```

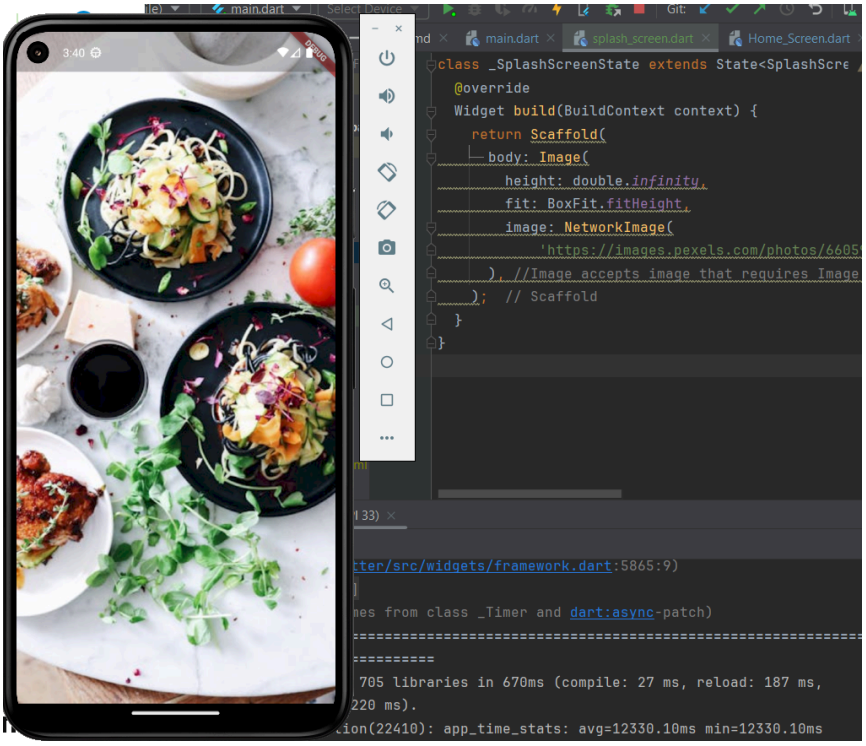
chapter 9

what is splash screen?

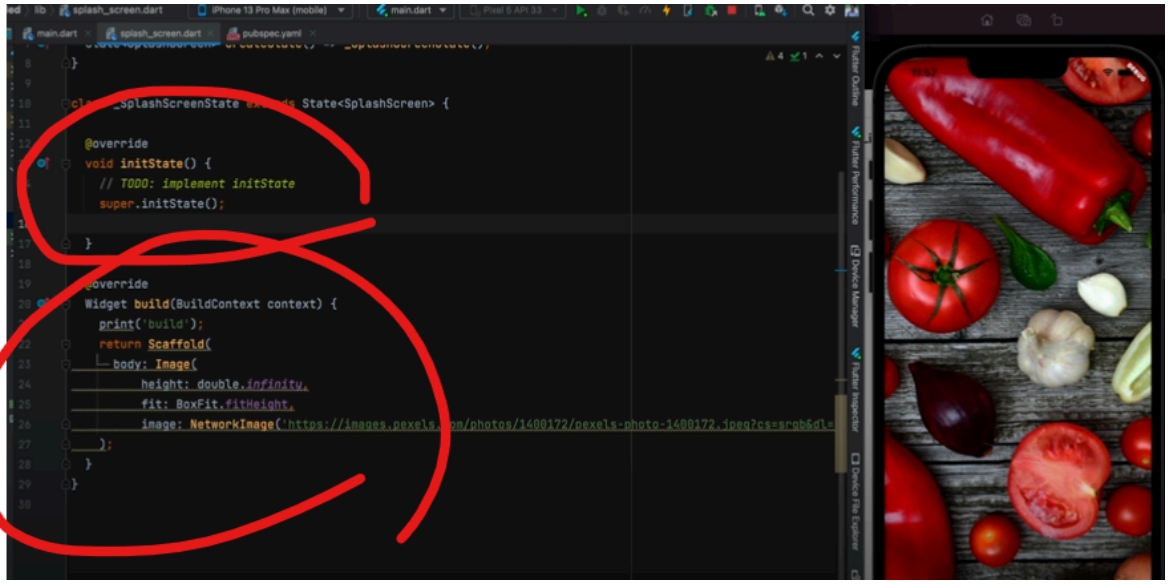
A splash screen is the initial screen that appears when an app is launched. It serves as a transitional display that is shown while the app is loading its content or initializing its components.

Fit full screen

inorder for image to fit full screen, we will use height and fit properties



if we write “init” we will create another function that if written before existing function, we execute first.



We created a splash screen in Flutter by using a `StatefulWidget` called `SplashScreen`, where the image from a network URL is displayed to cover the entire screen. The `Timer` function, from the `dart:async` package, is used in the `initState` method to delay the navigation to the `loginScreen` widget by 6 seconds using `Navigator.pushReplacement`. The `Scaffold` widget is used to set up the splash screen structure, with the `Image` widget set to `BoxFit.cover` to ensure the image covers the entire screen.

```
import 'package:flutter/material.dart';
import 'package:multi_role_base_app/Home_Screen.dart'; // Import the HomeScreen widget
import 'dart:async';
import 'package:multi_role_base_app/login_screen.dart';// Import dart:async for Timer

class SplashScreen extends StatefulWidget {
```

```
  @override
  _SplashScreenState createState() => _SplashScreenState();
}

class _SplashScreenState extends State<SplashScreen> {
  @override
  void initState() {
    super.initState();
    // Navigate to the home screen after a delay
    Timer(Duration(seconds: 6), () {
      Navigator.pushReplacement(
        context,
        MaterialPageRoute(builder: (context) => loginScreen()),
      );
    });
  }

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      body: Image(
        image: NetworkImage(
          'https://images.pexels.com/photos/6605903/pexels-photo-6605903.jpeg?
auto=compress&cs=tinysrgb&w=400'),
        fit: BoxFit.cover, // This makes the image cover the entire screen
        height: double.infinity,
        width: double.infinity,
      ),
    );
  }
}
```

InkWell

InkWell in Flutter is a widget that provides a simple way to add touch feedback to a widget, making it respond to tap gestures. It creates a "ripple" effect when the user taps on the widget, giving visual feedback that a touch interaction occurred.

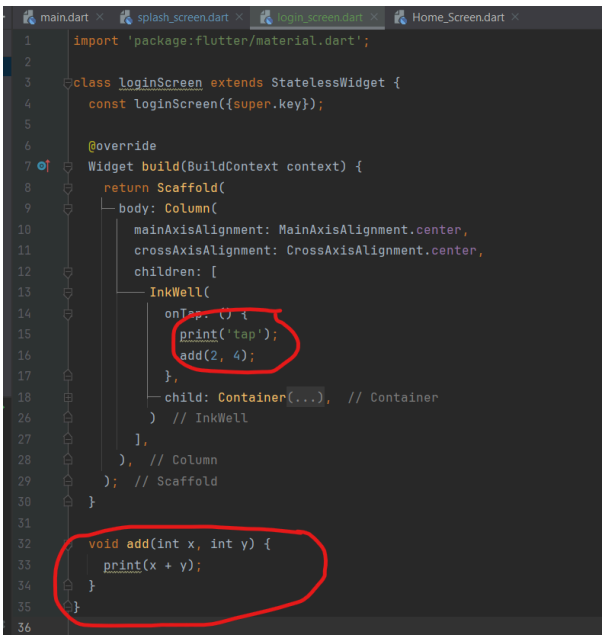
Asynchronous Functions

Asynchronous functions in Dart are functions that allow you to perform non-blocking operations, such as fetching data from the internet, reading a file, or performing a long computation, without freezing the main thread of your application. These functions are declared using the async keyword and typically return a Future, which represents a value that will be available at some point in the future.

Types of Functions in Dart:

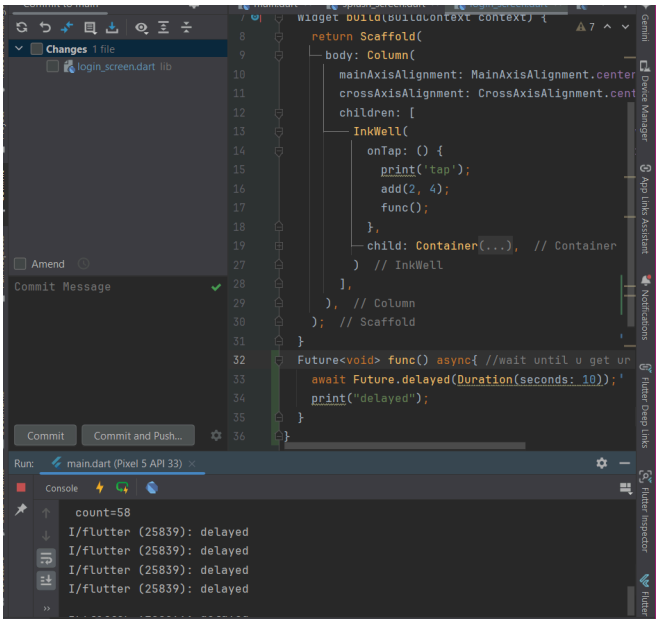
1. **Parameterized Function**: A function that takes one or more parameters as input.
2. **Void Function**: A function that returns no value (void return type).
3. **Future Function**: A function that returns a Future object, usually for asynchronous operations.
4. **Named Function**: A function with named parameters, allowing arguments to be passed by name rather than position.

Parameterized function



Future function

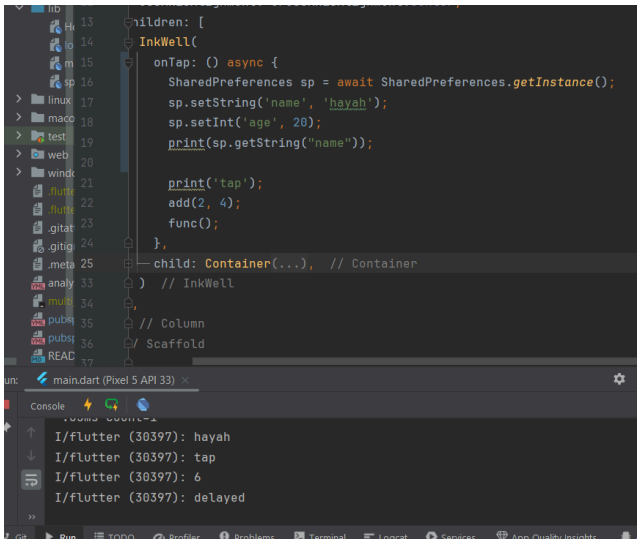
In the provided Flutter code, the `func` function is defined as `async`, meaning it returns a `Future` and can use the `await` keyword to pause its execution. When the `InkWell` widget is tapped, the `func` function is called, which initiates an asynchronous operation. Inside `func`, `await` is used with `Future.delayed(Duration(seconds: 10))`, which delays the execution of the next line by 10 seconds. This means the `print("delayed")` statement will only execute after the delay. The `await` keyword pauses the execution of `func` until the `Future.delayed` completes, allowing other operations to run concurrently during this waiting period. Meanwhile, the `add` function, which simply prints the sum of two integers, executes synchronously and immediately prints the result without any delay.



Shared Preferences

In the given Flutter code, shared preferences are used to store simple key-value pairs persistently on the device, allowing data to be saved and retrieved even after the app is closed. The `SharedPreferences` instance is obtained asynchronously using `SharedPreferences.getInstance()`. The `setString` and `setInt` methods store a string and an

integer (e.g., a user's name and age). The stored values can be retrieved using the `getString` method, as seen with `sp.getString("name")`, which prints the stored name. The button in the UI triggers this data storage and retrieval process when tapped, demonstrating how to persist and access user data easily in a Flutter application.



Building a multi role base app

Controllers Overview:

1. **TextEditingController**:
   - This class is used to control a text field. It allows you to retrieve the text input from the user, modify the text programmatically, and listen for changes in the text field.
2. If you wanted to print the email, password, and age entered by the user when the button is pressed, you could add this to the onTap function of the button:

onTap: () {

print('Email: ${emailcontroller.text}');

print('Password: ${passwordcontroller.text}');

print('Age: ${agecontroller.text}');

},’

Home_Screen.dart

This file defines the HomeScreen widget, which is the main screen users see after logging in.

Key Components:

1. **HomeScreen Widget**
   - StatefulWidget: The HomeScreen is a StatefulWidget because it needs to manage dynamic data (like the user's email and age) that can change over time.

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

 @override
 State<HomeScreen> createState() => _HomeScreenState();
}
```

_HomeScreenState Class

- **State Management**: This class holds the state for HomeScreen, including the user's email and age.
- **initState()**: This is called when the widget is first created. It's used here to load the user's data from SharedPreferences.
- **loadData() Method**: This method retrieves the user's email and age from SharedPreferences and updates the state, so the UI reflects the loaded data.

```
class _HomeScreenState extends State<HomeScreen> {
 String email = '', age = '';

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

 loadData() async {
  SharedPreferences sp = await SharedPreferences.getInstance();
  setState(() {
   email = sp.getString('email') ?? '';
   age = sp.getString('age') ?? '';
  });
 }
```

build() Method

- **UI Construction**: This method defines the UI of the HomeScreen. It includes a column that displays the user's email, age, and a logout button.
- **Logout Functionality**: The InkWell widget wraps the logout button. When tapped, it clears SharedPreferences (removing saved user data) and navigates back to the LoginScreen.

```
@override
Widget build(BuildContext context) {
 return Scaffold(
  body: Column(
   mainAxisAlignment: MainAxisAlignment.center,
   crossAxisAlignment: CrossAxisAlignment.center,
   children: [
    const Text('HOME SCREEN'),
    const SizedBox(height: 20),
    Text('Email: $email'),
    Text('Age: $age'),
    const SizedBox(height: 40),
    InkWell(
     onTap: () async {
      SharedPreferences sp = await SharedPreferences.getInstance();
      sp.clear();
      Navigator.pushReplacement(
       context,
       MaterialPageRoute(builder: (context) => const LoginScreen()),
      );
     },
     child: Container(
```

```
          height: 50,
          width: double.infinity,
          color: Colors.green,
          child: const Center(
            child: Text('Logout', style: TextStyle(color: Colors.white)),
          ),
        ),
      ),
    ],
  ),
 );
}
```

login_screen.dart

This file defines the LoginScreen widget, which is the screen where users enter their login information.

Key Components:

1. **LoginScreen Widget**
   - This is a StatefulWidget because it manages form fields (email, password, age) that can change based on user input.

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

@override
_LoginScreenState createState() => _LoginScreenState();
}
```

_LoginScreenState Class

- **TextEditingController**: These controllers are used to manage the text input fields for email, password, and age.
- **build() Method**: This constructs the UI of the login screen, which includes text fields for email, password, and age, and a button that handles the login process.

```
class _LoginScreenState extends State<LoginScreen> {
final TextEditingController passwordController = TextEditingController();
final TextEditingController emailController = TextEditingController();
final TextEditingController ageController = TextEditingController();
```

Login Logic

- **SharedPreferences**: When the user taps the login button, the app saves the user's information in SharedPreferences and sets the islogin flag to true. This information is then used to determine if the user should be taken directly to the HomeScreen in the future.
- **Navigation**: After storing the user's data, the app navigates to the HomeScreen.

```
InkWell(
onTap: () async {
SharedPreferences sp = await SharedPreferences.getInstance();
sp.setString('email', emailController.text.toString());
sp.setString('password', passwordController.text.toString());
```

```
sp.setString('age', ageController.text.toString());
sp.setBool('islogin', true);

Navigator.pushReplacement(
context,
MaterialPageRoute(builder: (context) => const HomeScreen()),
);
},
child: Container(
height: 50,
width: double.infinity,
color: Colors.green,
child: const Center(
child: Text('Click', style: TextStyle(color: Colors.white)),
),
),
),
```

splash_screen.dart

This file defines the SplashScreen widget, which is the initial screen users see when the app launches.

Key Components:

1. **SplashScreen Widget**
   - This is a StatefulWidget because it manages a timed delay before navigating to the appropriate screen.
   - ```
     class SplashScreen extends StatefulWidget {
     @override
     _SplashScreenState createState() => _SplashScreenState();
     }
     ```

_SplashScreenState Class

- **checkLoginStatus() Method**: This method checks if the user is already logged in by retrieving the islogin value from SharedPreferences.
- **Timer**: After a delay (6 seconds), the app decides whether to show the HomeScreen (if logged in) or the LoginScreen (if not logged in).

```
void checkLoginStatus() async {
SharedPreferences sp = await SharedPreferences.getInstance();
bool isLogin = sp.getBool('islogin') ?? false;

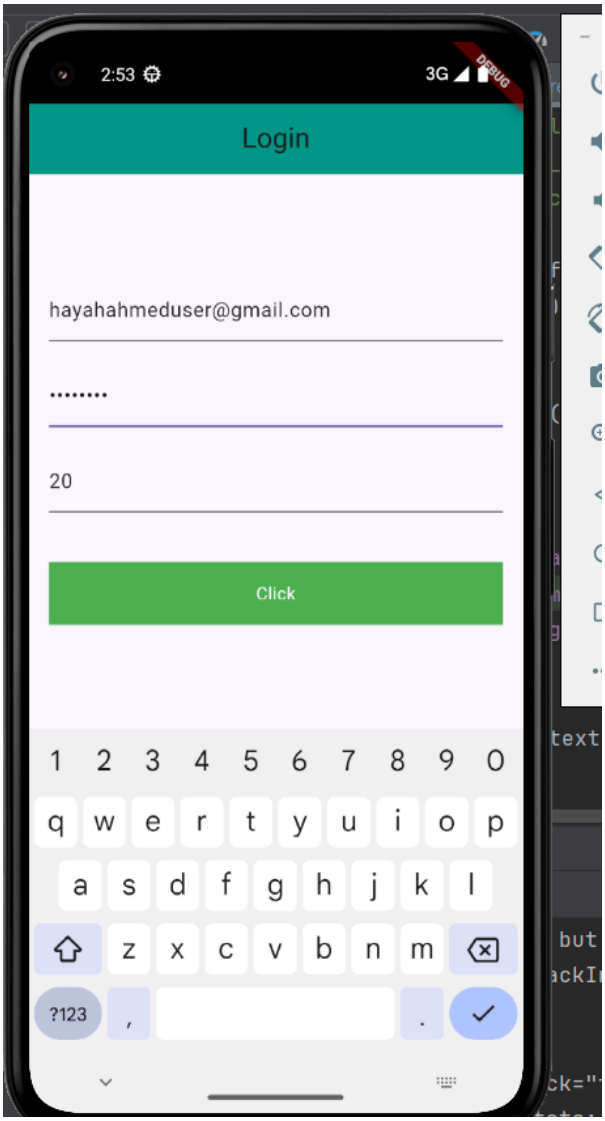
Timer(const Duration(seconds: 6), () {
if (isLogin) {
Navigator.pushReplacement(
context,
MaterialPageRoute(builder: (context) => const HomeScreen()),
);
} else {
Navigator.pushReplacement(
context,
MaterialPageRoute(builder: (context) => const LoginScreen()),
);
```
```

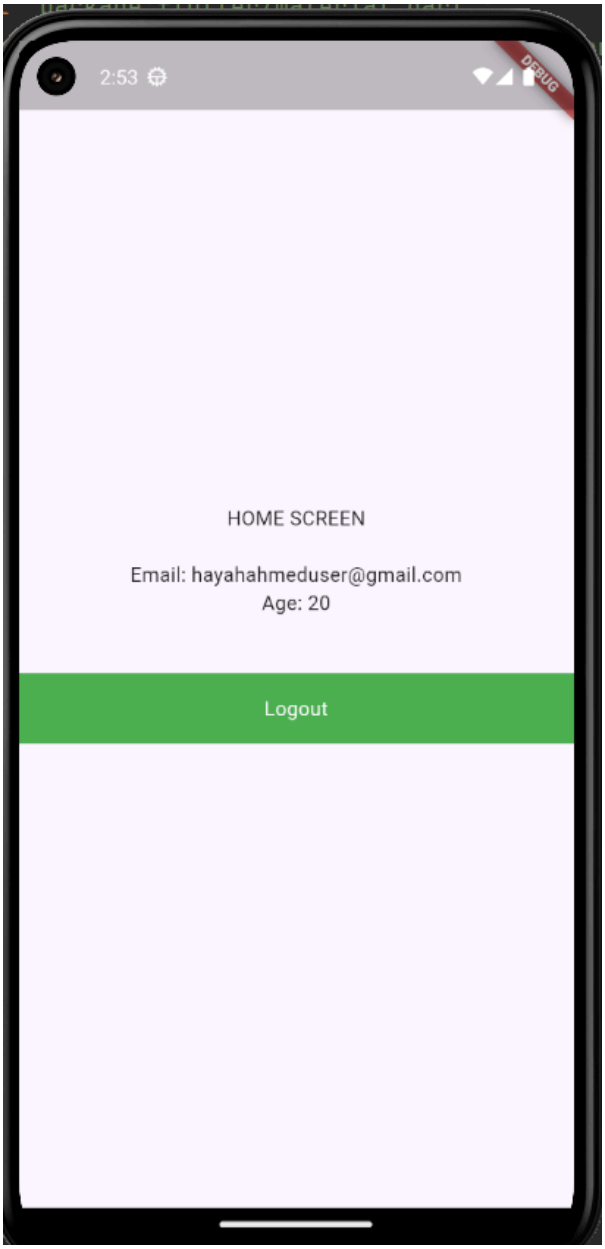
```
}
});
}
```

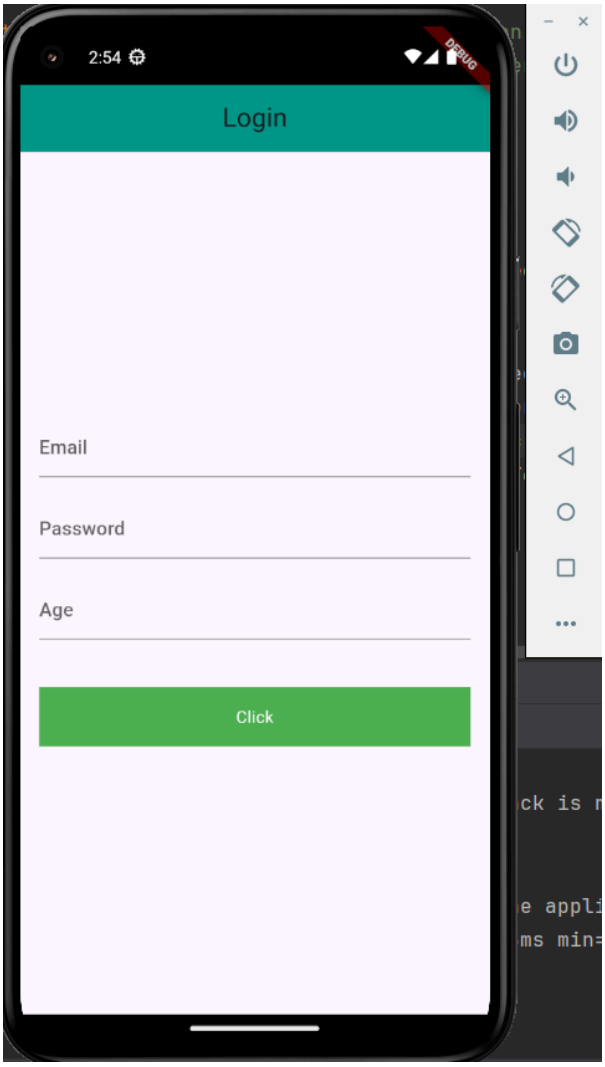
build() Method

- **UI Construction**: Displays an image covering the entire screen, serving as the splash screen while the app determines the login status.

```
@override
Widget build(BuildContext context) {
return Scaffold(
body: const Image(
image: NetworkImage(
'https://images.pexels.com/photos/6605903/pexels-photo-6605903.jpeg?auto=compress&cs=tinysrgb&w=400'),
fit: BoxFit.cover,
height: double.infinity,
width: double.infinity,
),
);
}
```


2:53
3G
DEBUG
Login
hayahahmeduser@gmail.com
20
Click


2:53
DEBUG
HOME SCREEN
Email: hayahahmeduser@gmail.com
Age: 20
Logout


2:54
DEBUG
Login
Email
Password
Age
Click