Flutter Notes

COURSE CONTENT

Introduction To Flutter & Dart

- Dart or Flutter ? Where to start
- Introduction to Flutter
- Prerequisite of installing Flutter

Install Flutter On Window

- Install Flutter for windows
- Install Java JDK
- Install android studio
- Install command line tools with sdks
- Install emulator

Install flutter on macbook

- Install flutter on Mac with intel chip and M1 with silicon chip
- Install android studio and android SDK
- Install command line tools for android
- Install emulator
- Install xcode
- Install cocoapod to for plugins conversion in future
- Run project on ios simulator

Running app on physical devices

- Creating our first project
- Run app on android physical device and virtual device
- Running app on physical iPhone device
- Run app on iPhone/iPad simulator

Flutter Chapter 1

- Creating our first project
- Material App, what is material app. Home, Text and centre widget with tree charts explanation
- What is material design
- Scaffolding the app, material colours and network image widge
- Assets image
- Chaning app icon
- Summing up what we have learned so far

Flutter Chapter 2 Login App UI

- This Chapter Tutorial
- Stateless widgets
- Hot reload and hot restart
- Single children widgets

- How to use material Icons widget
- How to use container widget with decorations
- Wrapping the widget
- What is layout
- Multiple Layout children widgets i.e column, row, and stack widgets
- Adding fonts into project
- Creating Login UI design
- TextForm Field widget
- Creating round button

Flutter Chapter 3 Lottery App Building app with states

- This Chapter Tutorial
- Creating Project
- Stateless widgets overview
- Stateful widget
- Understanding variable, data type and ternary operator
- Creating UI with stateless widget
- Difference between stateless and stateful widget
- Gesture Detector on tap or on pressed function
- Lottery App functionality implementation
- Random number
- Building states
- Understanding more about stateful widget with more explanation and practice
- Difference between stateless and stateful widget

Flutter Chapter 4 Tops 10 Widgets

- This Chapter Tutorial
- Top widgets that helps to build any kind of a APP UI
- Container widget
- Expanded widget
- List tile widget
- Circular Avatar widget
- Stacks widget
- List view builder widget
- Text Form Field widget
- RichText Widget widget
- SizedBox
- Divider
- Padding & Align widgets
- Image widget

Flutter Chapter 5 Building Building UI and understanding deep concepts and Navigation drawer

This Chapter Tutorial

- Creating Dart classes
- App Bar widget
- Drawer
- Drawer header
- Navigation
- Side Menu/ Drawer or Navigation drawer

- Code refactoring
- Navigation, Routing and navigation or moving between screen
- Building small navigation drawer app

Flutter Chapter 6 Building WhatsApp Ui

- This Chapter Tutorial
- Source code
- Understanding Tab Bar widgets
- How we can create different Tab
- Understanding models
- Creating chats ui and parsing data via model
- Creating status ui and parsing data via model
- Creating calls ui and parsing data via model
- Summing up what we have learned

Flutter Chapter 7 Understanding pub.dev

- This Chapter Tutorial
- Understanding the pub.dev
- Understanding about community
- Working with some of the packages on pub dev
- How we can use pub.dev
- How Fonts Awesome package helping community
- Read more package to show more and less text
- Badges packages to show text on Cart Icon
- Animated Text Kit package
- Pin Code Filed Package
- Animated splash screen package
- Summing up what we have learned

Flutter Chapter 8 Building calculator app

- This Chapter Tutorial
- Source Code
- What is data types, working with strings and substrings
- If else statement
- What is component
- What is constructor
- Creating component/ how to create components
- How to use reusable widget or code refactoring
- Building calculator app
- Creating logic for calculator
- Showing result for calculation
- Summing up what we have learn

Flutter Chapter 9 Shared Preference Multi Role Base App - Admin/User

- This Chapter Tutorial
- Splash Screen and init state function importance
- Async function
- Future functions and await keyword role
- Shared Preference and its importance
- Shared preference and its usage with example

- Sign up with role based and storing data in shared preference
- Login and checking user role
- Managing user session to check if he is login or not
- Summing up what we have learned

Flutter Chapter 10 Rest API Complete course

- This Chapter Tutorial
- I have created a full course on API integration in Flutter from scratch to advance level. If you are a new bee and looking for a flutter resource regarding API integration then you will find it interesting.

These videos are not just simple tutorials that i have created, i am sharing my experience in the field of app development and what type of mistakes i did during my learning phase and that you are not supposed to do. So, all of your questions that are shaking your mind will be answered here.

Here is what we will learn in this course:

- Introduction to APIS. Understanding of JSON Structure.
- What is Postman, how it helps us to understand the JSON Response.
- What is model, what are Plugins and how different Plugins help us to create models of our API JSON data.
- How we can parse JSON data via Model.
- GET APIS What are Get APIS
- What are different scenarios to handle Get API Integrate.
- Get APIS with Plugins Model and show data into List
- Integrate Get APIS with your own Model and show data into List Integrate
- Get APIS with without Model and show data into List Very Complex JSON practical example
- POST APIS, What is POST API How do Post APIS work
- Implement Login & Sign Up with with REST API in flutter
- Upload Single Image onto server via HTTP Request Example
- Upload Multiple images to server
- Upload data in arrays to server
- What is MVVM, what is the importance of learning MVVM architecture. Covering all the above concepts into a single app that we will create at the end of this course.

So what keeps you stopping to learn Flutter, let's start.

Flutter Chapter 11 Navigation & Routing

- This Chapter Tutorial
- In this chapter we will learn following concepts
- What is navigation
- How navigation actually work behind the door
- How understanding of stacks help you better understand navigation
- What is difference between navigation and routing
- Routing with 1 method
- Routing with second method and simplifying the code for better management

Flutter Chapter 12 Google Map

- This Chapter Tutorial

- Out of this course and what you will learn
- What is google map, why you to learn google map in mobile application development
- What google map api console, how we can set up our Flutter project with google map api console.
- Integrating google map with packages and understanding basic terminologies.
- How we can set up marker on google map
- How to set custom markers on google maps like careem and uber.
- How to get the user 's current location and show it on a marker.
- How to convert latlng to address and vice versa
- How to customise marker info windows like google map places.
- How to update camera position when camera is updating on google map.
- How to update user location when user location is changing.
- How to implement google map search places api.
- How to show network images or Avatar as marker on google map like Snapchat
- What is polygon and how we can add polygon on google map
- What is polyline and how we can add polyline on google map

Flutter Chapter 13 Local Storage Creating Shopping cart system

- This Chapter Tutorial
- Create a Shopping Cart System in Fuand what you will learn in this playlist.
- What is the local storage system and how does it work on the mobile in FLutter?
- What it is required to setup local storage in Flutter
- Create a project and a database for storing data.
- How we insert data into the database and how this actually works.
- Add product to cart
- Update cart
- Delete product from cart
- Check if product is already added to cart or not
- Calculating the total price for the products we have added and updating prices if we update or delete products etc.

At the end I will sum up the whole project, what we have learned and how we can utilise these concepts to build other similar real world applications.

Flutter Chapter 14 Provider State management

- This Chapter Tutorial
- Provider state management crash course from scratch to advanced level.
- Difference between stateful & stateless widgets
- Why you need to learn learn state management with example
- What is a provider, how does a provider work, why are we learning provider state management. What is change notifier, multi change notifier and consumer widget
- Example one, count and periodic timer example with set state and provider.
- Example two practice apps to understand provider state management a bit more in detail.
- Example three to make a simple app where we will list the product in a list and then make them favourite. After that we will display a list of favourite products that are marked as favourites from the list.
- Example 4 theme changer app with provider
- Example 5 Login API with provider
- Example 6 how to use stateless widget as stateful widget.
- Will cover up what we have learned and what we will do after that.

- Complete app with provider using MVVM architecture with rest APIs.

Flutter Chapter 15 Firebase

- This Chapter Tutorial
- Creating Project
- Setting up project with firebase for android and ios
- CRUD operation with Firebase real time database
- CRUD operation with Firestore
- Implementing Auth firebase
- Creating Blog application
- Summing up what we have learn

Flutter Firebase Notifications Chapter 16

- Flutter Firebase Notifications Roadmap
- Flutter Firebase Project Setup for android and ios
- How To Get Firebase Notification Permission
- What is FCM Token or Device Token
- Flutter Show notification when app is in Foregrounds, backgroudn or in terminated states
- Redirect User to specific screen when notifications is tap
- How to send notification from one device to anohter device

Flutter GetX State Management Chapter 17

Flutter Getx Road Map and what we will learn in this playlist

- Getx Roadmap
- Creating our Project and adding Getx Pacakages
- Get Material App (Why we used Get Material App)
- GetX as utilities
- GetX Snackbar
- Getx Dialog Alert
- Getx Bottom Sheet
- Getx Change Light & Dark Theme with GetX Flutter
- Getx Navigator & Routes (How to pass data between different screens)
- Get height & Get widh
- GetX Localization Change the Language of the App i.g Englihs, Urdu, Hindi

Installation:

chapter 2

- What is Material App Class?

All widgets are injected using which we are going to build our application.All functionalities, values, widgets. So apps are started using this. https://api.flutter.dev/flutter/material/MaterialApp-class.html

```
void main() {
  runApp(const MaterialApp());
}
```

Material App is a class that is given to RunApp as a parameter and has a property “home” to which a widget “Text” is assigned as value.

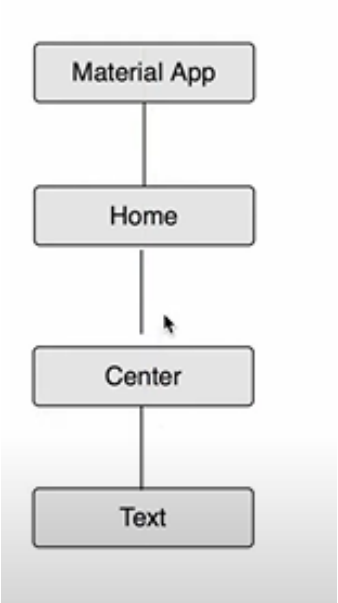
```
void main() {
  runApp(const MaterialApp(
    home: Text('Hello World')
  ));  // MaterialApp
}
```

- What is a Widget?

Anything that is displayed on the screen, be it text, icon, image.
https://docs.flutter.dev/ui/widgets/basics

- What is a Tree?

```
runApp(const MaterialApp(home: Center(child: Text("Hello World"))
```



Material App has a child Home , Home has a child Center widget and Center has a child Text widget.

Parent----Child------Child's child

```
runApp(const MaterialApp(
  home: Center(
    child: Text('Hello world'),
  ),  // Center
));  // MaterialApp
}
```

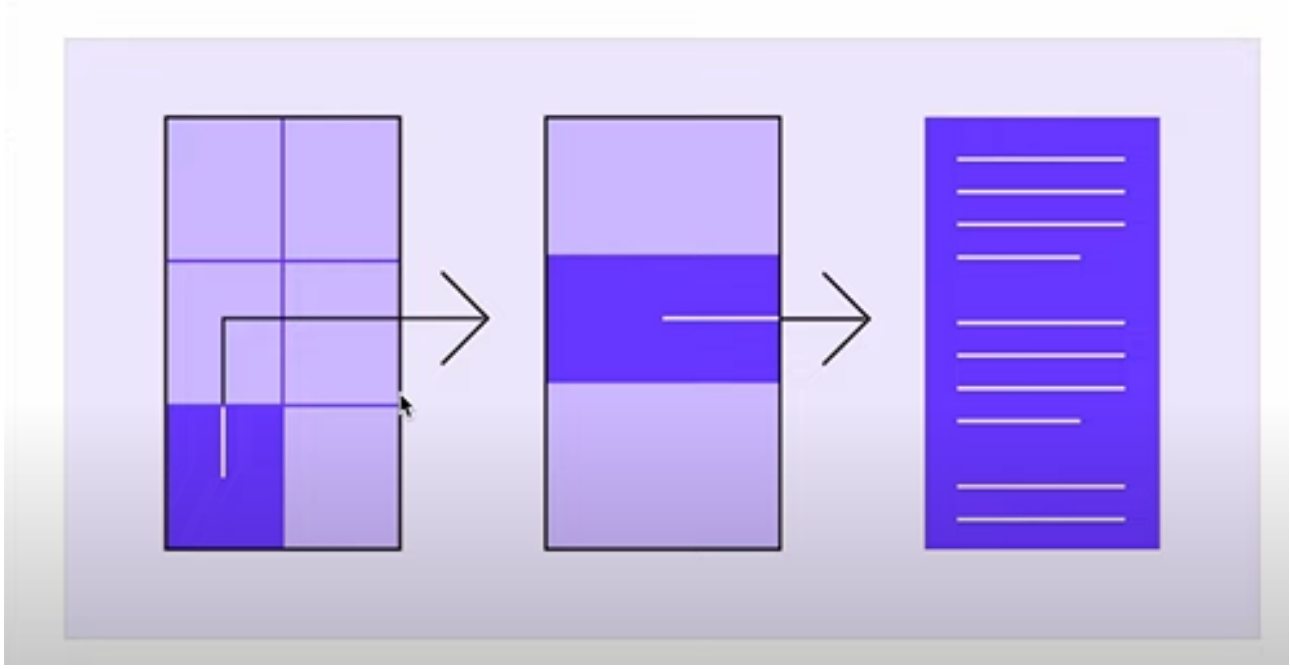
- What is Material Design?

Google's in-house language/design/systems for designing mobile applications. https://m3.material.io

- What is a Layout?

This whole screen/area is a layout.

Navigation:



Components:

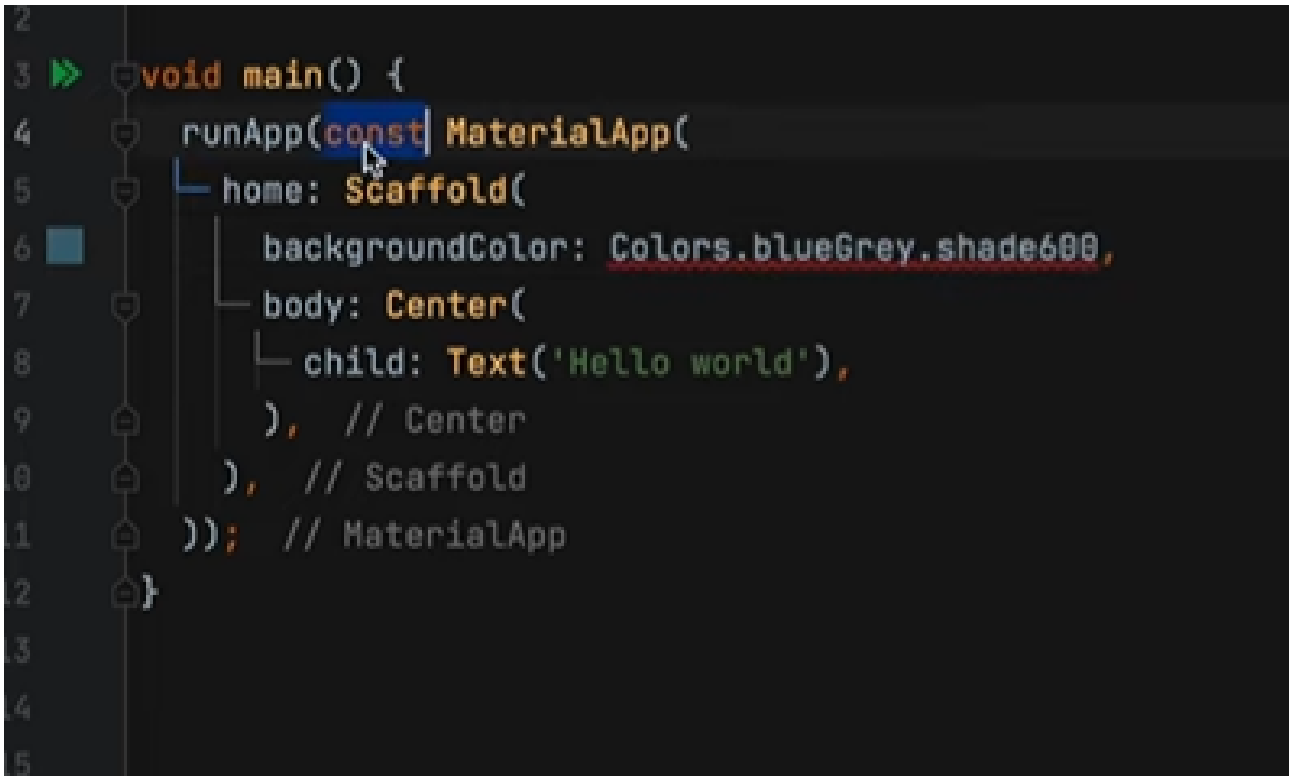
Flutter doesn't explicitly use the term **components** as distinct from widgets, components in a general sense can be thought of as composite widgets or custom widgets that encapsulate more complex functionality or UI patterns.

Scaffold widget, its properties and how it works.

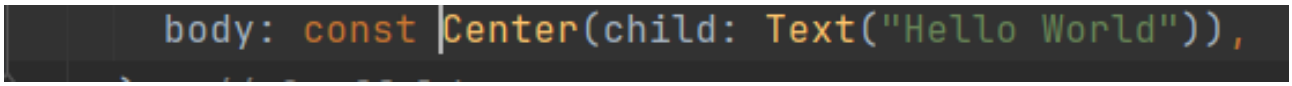
Scaffold occupies whole body of the screen. https://api.flutter.dev/flutter/material/Scaffold-class.html MaterialApp has a property home, which has the value Scaffold , that has the property body, background color.. etc

```dart
void main() {
  runApp(const MaterialApp(
    home: Scaffold(
      body: Center(child: Text("Hello World")),
    ),  // Scaffold
  ));  // MaterialApp
  }
```

This white area is basically the scaffold widget.



The backgroundcolor property of scaffold is giving error because blueGrey is no longer a constant value and is changed therefore we need to remove “const” keyword.

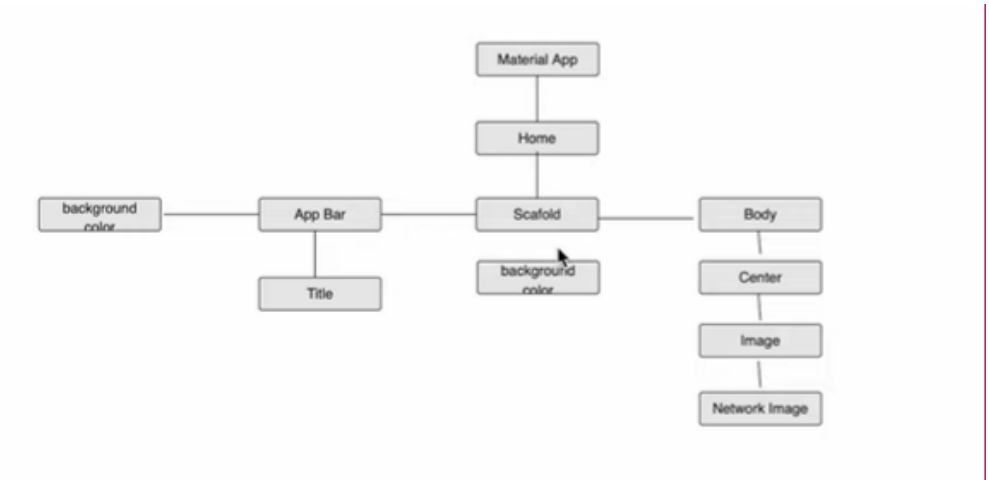


Here we added const keyword as the value is not changing at run time

Image Class

Widget that displays image. https://api.flutter.dev/flutter/widgets/Image-class.html

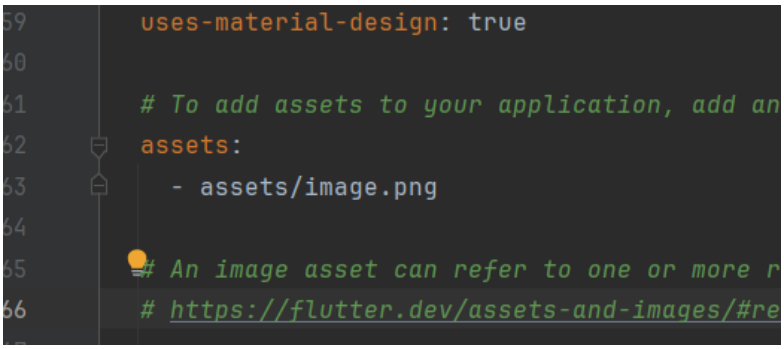
```dart
import 'package:flutter/material.dart';

void main() {
  runApp(MaterialApp(
    home: Scaffold(
      backgroundColor: Colors.pink.shade600,
      body: const Center(child: Image(image: NetworkImage('https://im
        appBar: AppBar(
        backgroundColor: Colors.pink.shade100,
        title: const Text("MY APP"),
        centerTitle: true,
      ),  // AppBar
    ),  // Scaffold
  ));  // MaterialApp
}
```



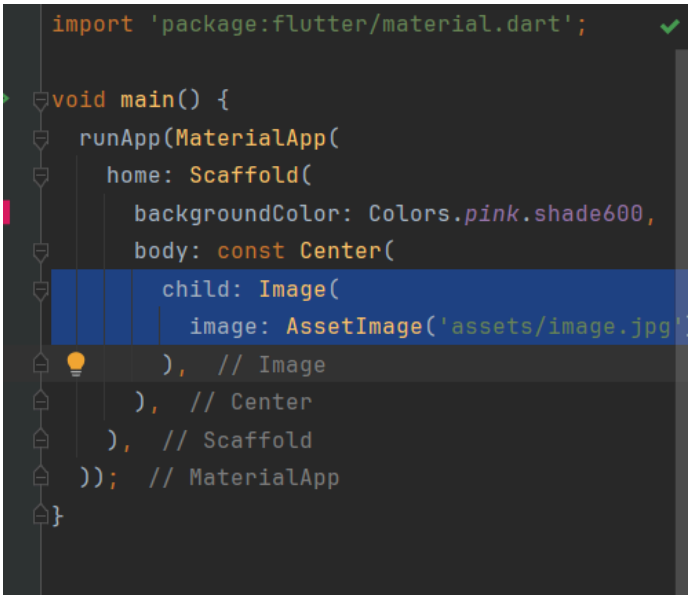
How to add/use/purpose assets?

external data that resides within your device.

- to use assets in frontend, go to **pubspec.yaml** file
- remove comments
- change path and file name
- 2 spaces for assets:
- 4 spaces for assets/image.png
- then go to **terminal** and write flutter pub get

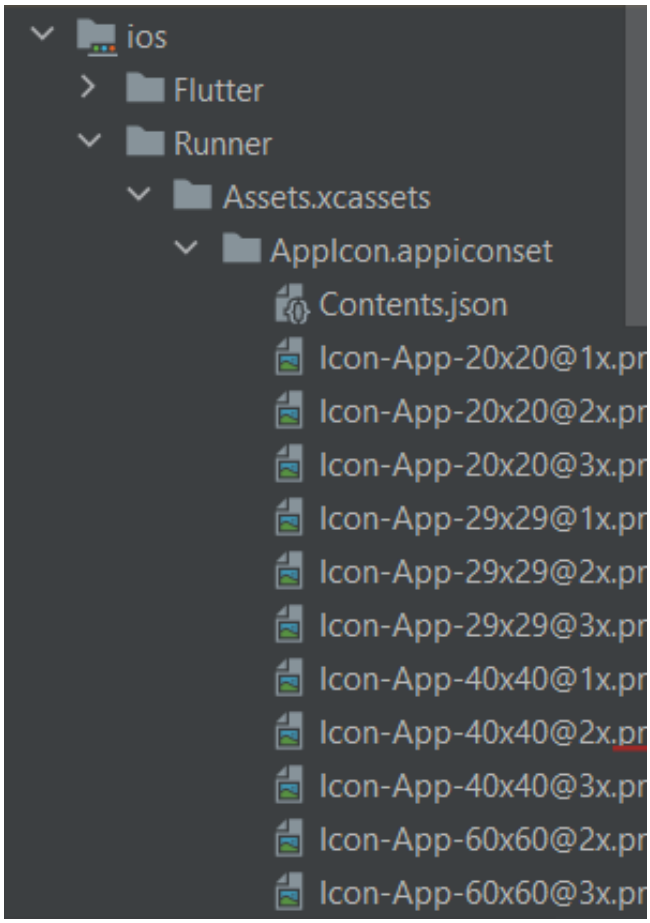


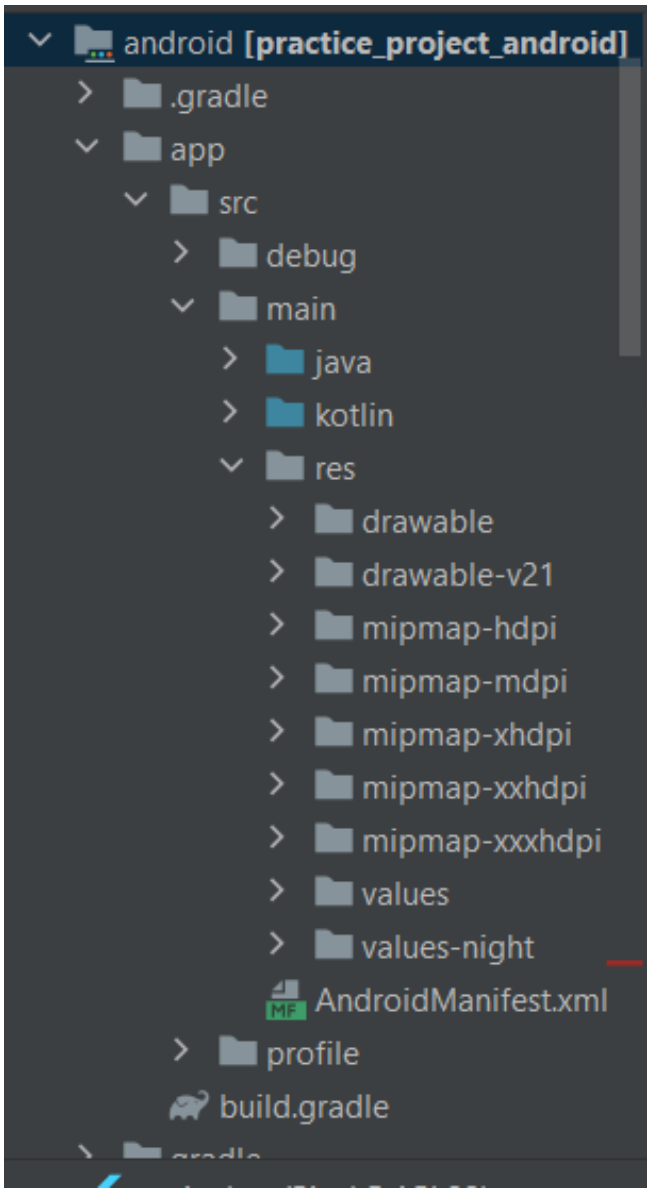

```
PS C:\Flutter Dev\Projects\practice_project> flutter pub get
Resolving dependencies...
Downloading packages...
```



What is APP ICON?

App icons automatically gets adjusted
by default theyre stored like this

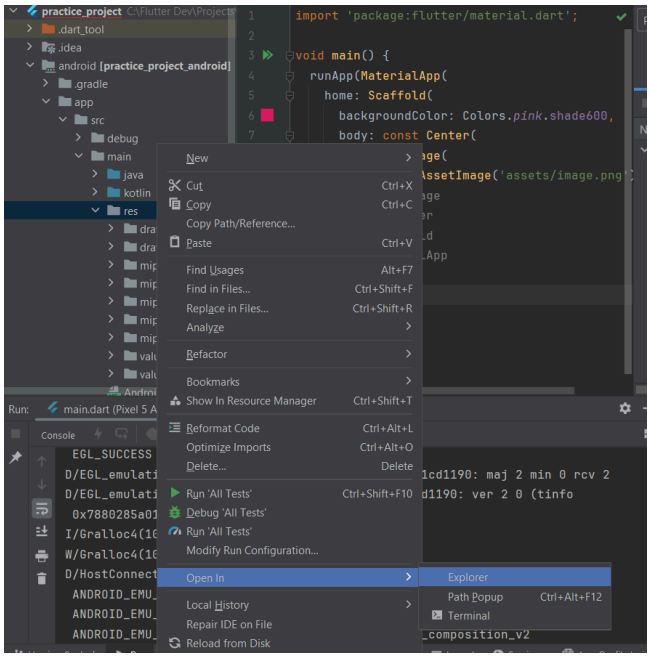

ios
Flutter
Runner
Assets.xcassets
AppIcon.appiconset
Contents.json
Icon-App-20x20@1x.pr
Icon-App-20x20@2x.pr
Icon-App-20x20@3x.pr
Icon-App-29x29@1x.pr
Icon-App-29x29@2x.pr
Icon-App-29x29@3x.pr
Icon-App-40x40@1x.pr
Icon-App-40x40@2x.pr
Icon-App-40x40@3x.pr
Icon-App-60x60@2x.pr
Icon-App-60x60@3x.pr



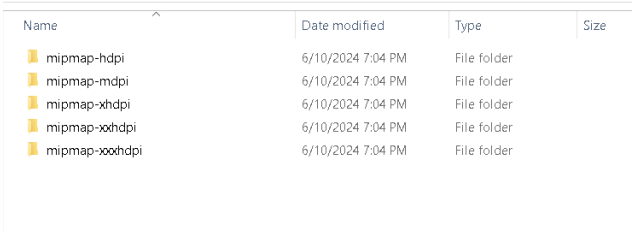
Inorder to change appicon we need to follow the following steps:

ANDROID

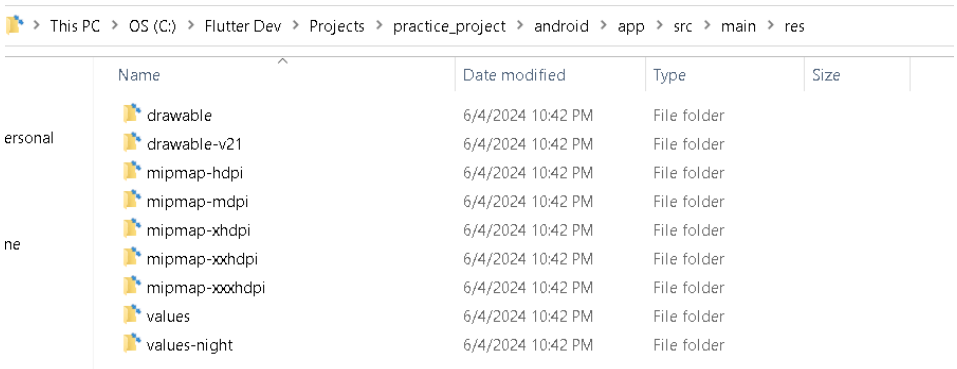
- [ ] To make icon size according to the platform go to appicon.co
- [ ] Then unzip the .zip file and store it in folder
- [ ] Go to Android-app-src-main-res-right click-open in-explorer.
- [ ] then copy all the files from folder which are in android folder and replace it with files in res



folder (unzipped)



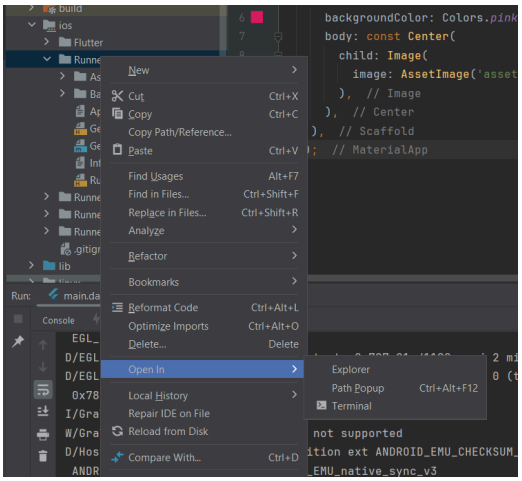
copy and paste to



IOS

- [ ] To make icon size according to the platform go to appicon.co
- [ ] Then unzip the .zip file and store it in folder
- [ ] Go to ios-Runner-right click-open in-explorer.

- [ ] then copy the Assets.xcassets in that unzipped folder and replace it with destination folder in ios



now run again

WHAT ARE STATELESS WIDGETS?

Stateless Widget is something that does not have a state. A state can be defined as “an imperative changing of the user interface” and, a widget is “an immutable description of the part of user interface”.

write stl and this will appear

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

  @override
  Widget build(BuildContext context) {
    return Container();
  }
}
```

The MyApp class is a Stateless Widget and Widget build is a method with BuildContext as a parameter that returns widgets. Every widget while entering the Widget build(Build Context context) has to override it in order to enter the widget tree. Some Examples of a stateless widget are Text, IconButton, AppBar, etc. Build Method is called inside a stateless widget in only three cases:

- Initially in the start when it is built for the very first time while starting the application.
- If the parent widget is changed
- If the inherited widget is changed

The control will enter the runApp() through the main. From there it will be redirected to the stateless widget of the MyApp class. The build method will be called the very first-time MyApp stateless widget is built. Every time a widget wants to enter the widget tree, it has to override the Widget build (BuildContext context)method. After that, the control goes to the MaterialApp widget and the rest of the code will be executed. Remember, every time a new widget enters the widget tree, it has to override the build method.

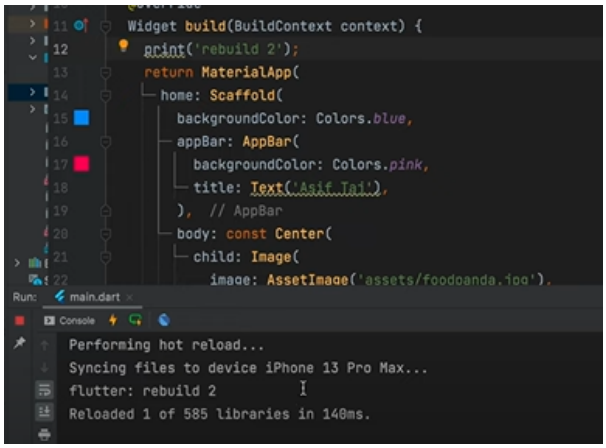
```dart
import 'package:flutter/material.dart';

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


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

  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      home: Scaffold(
        backgroundColor: Colors.pink.shade600,he
        body: const Center(
          child: Image(
            image: AssetImage('assets/image.png'),
          ),  // Image
        ),  // Center
      ),  // Scaffold
    );  // MaterialApp
  }
}
```

What is Hot Reload?

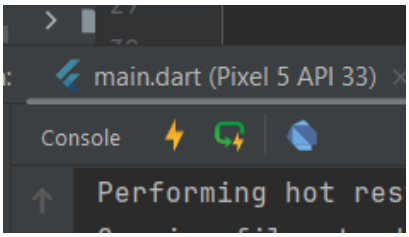
Flutter's hot reload feature helps you quickly and easily experiment, build UIs, add features, and fix bugs. Hot reload works by injecting updated source code files into the running Dart Virtual Machine (VM)

To see real time changes, instead of play button use hot reload

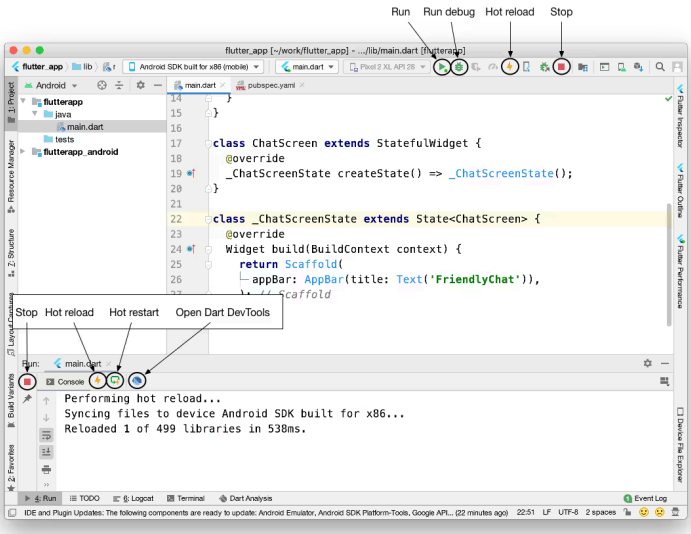
all the widgets inside Widget build() are rebuild after hot reload and main() se run nhi krtaa, it checks k screen k upar konsa widget run horha aur usko run krta



Hot Restart?

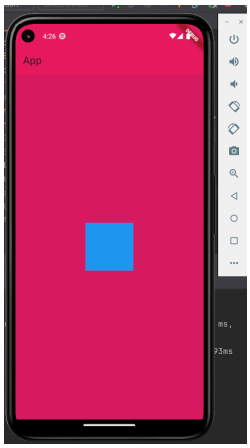


hot restart rebuilds the entire project



Container Widget:

its like a small box, and could be listed as a child of center.
pink area is layout which is build using scaffold
scaffold has two properties appbar and body



Single Layout child or Single child widget

Center() widget has a child Container(), Container() has a child Center() and it has a child Text()---> this hierarchy structure is **single layout child/single child widget.**

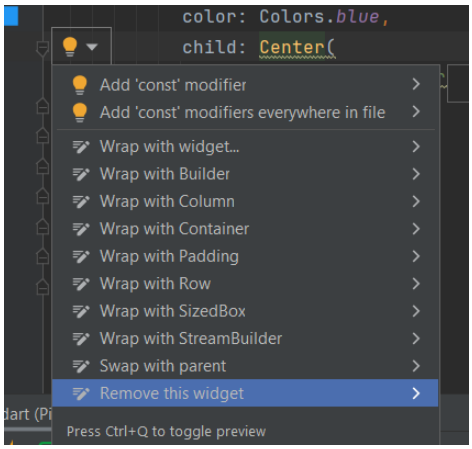
```
body:  Center(
  child: Container(
    height: 100,
    width: 100,
    color: Colors.blue,
    child: Center(
      child: Text('Container'),
    ),  // Center
  ),  // Container
),  // Center
),  // Scaffold
```

we wrapped text widget into a center widget

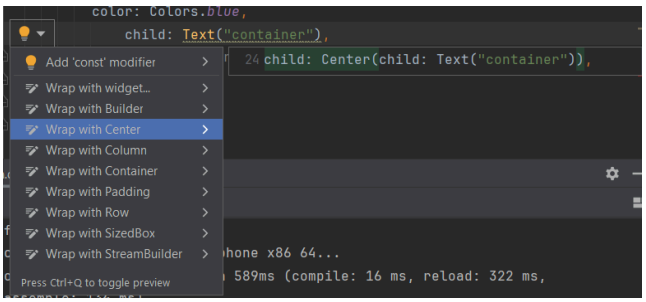
```
    ),  // AppBar
    body: Center(
      child: Container(
      height: 200,
      width: 200,
      color: Colors.blue,
        child: Center(
          child: Text("container"),
        )  // Center
      )  // Container
    ),  // Center
  ),  // Scaffold
);  // MaterialApp
}
```

A container widget can have only one child at a time, so the below implementation is wrong

Removing a widget:



wrapping a widget directly into center()



Multi Layout children or Multi children widget or Multi child layout widgets

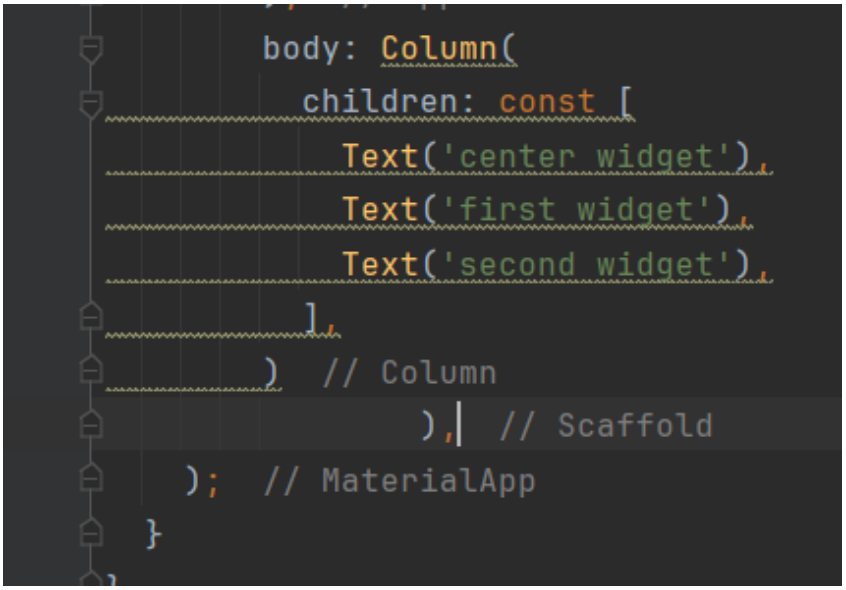
Important widgets: (whole flutter ui is based on these components)

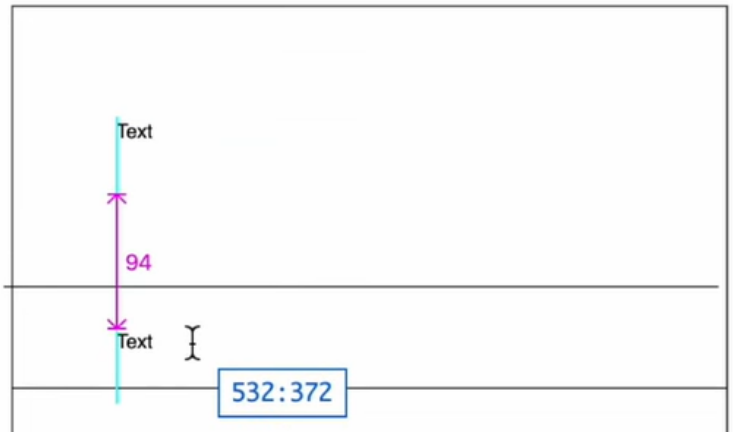
- Row widgets
- Column widgets
- Stack widgets

– widgets are divided into two categories: stateful & stateless

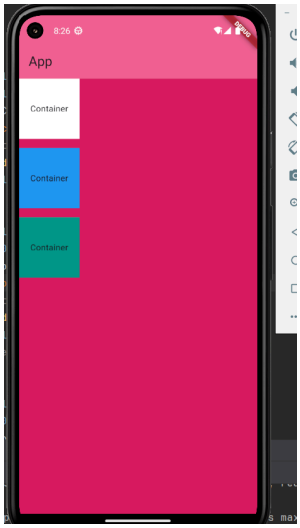
Column Widget:

scaffold has properties appbar and body
in body we could use column widget () inorder to have access to multiple children widgets
it has chidren property which accepts array of children widgets



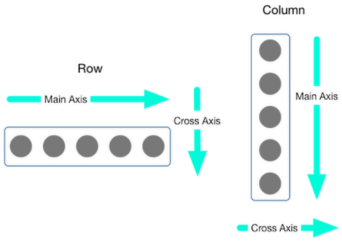


```
      ), // AppBar
      body: Column(
        children: [
          Container(
            height: 100,
             width: 100,
             color: Colors.white,
             child: const Center(child: Text('Container')),
          ), // Container
          const SizedBox(
            height: 15,
                      ), // SizedBox
          Container(
            height: 100,
            width: 100,
            color: Colors.blue,
            child: const Center(child: Text('Container')),
          ), // Container
          const SizedBox(
            height: 15,
          ), // SizedBox

          Container(
            height: 100,
            width: 100,
```

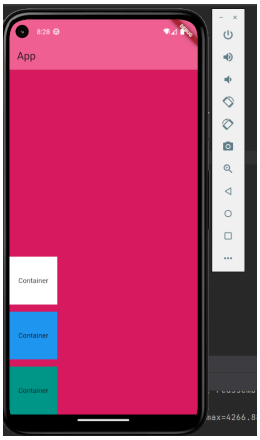


Main Axis vs Cross Axis

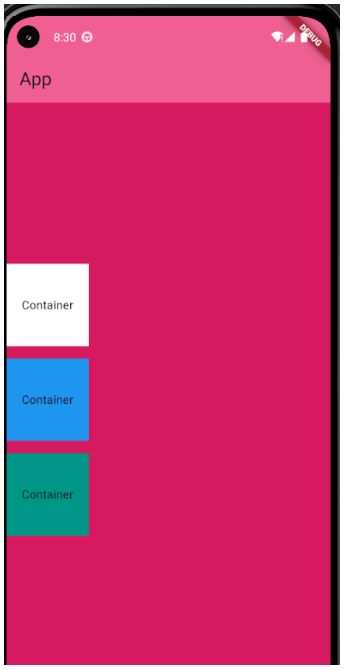
For a row, the main axis runs horizontally and the cross axis runs vertically.



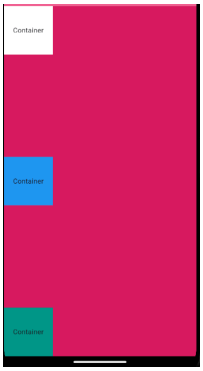

```
    body: Column(
      mainAxisAlignment: MainAxisAlignment.end,
      children: [
        Container(...), // Container
        const SizedBox(...), // SizedBox
        Container(...), // Container
        const SizedBox(...), // SizedBox

        Container(...), // Container
```



mainAxisAlignment: MainAxisAlignment.center,



mainAxisAlignment: MainAxisAlignment.spaceBetween,



crossAxisAlignment: CrossAxisAlignment.end,

SafeArea

wrapping widgets in SafeArea restricts it from going outside the layout

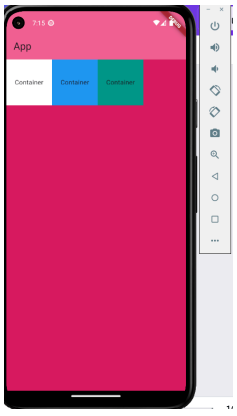
```
body: SafeArea(
  child: Column(
    mainAxisAlignment: MainAxisAlignment.spaceBetween,
```

```
children: [
 Container(
  height: 100,
   width: 100,
   color: Colors.white,
   child: const Center(child: Text('Container')),
 ),
 const SizedBox(
  height: 15,
```

Row Widget

arranges the children in row fromat

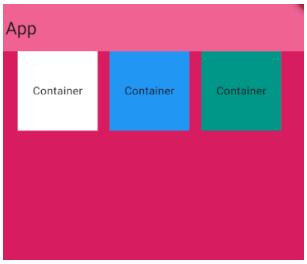
```
        backgroundColor: Colors.pink.shade300,
        title: const Text("App"),
      ),  // AppBar
      body: SafeArea(
        child: Row(
          mainAxisAlignment: MainAxisAlignment.start,
          crossAxisAlignment: CrossAxisAlignment.end,
          children: [
            Container(...),  // Container
            const SizedBox(...),  // SizedBox
            Container(...),  // Container
            const SizedBox(...),  // SizedBox
            Container(...),  // Container
          ],
        ),  // Row
      )),  // SafeArea, Scaffold
```



in Column Widget we were using height property of SixedBox widget for space between containers but in Row Widget we use width property

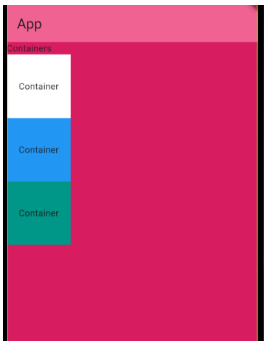
```
          child: Row(
            mainAxisAlignment: MainAxisAlignment.start,
            crossAxisAlignment: CrossAxisAlignment.end,
            children: [
              Container(...),  // Container
              const SizedBox(
                width: 15,
              ),  // SizedBox
              Container(...),  // Container
              const SizedBox(
                width: 15,
              ),  // SizedBox
              Container(...),  // Container
            ],
          ),  // Row
        )),  // SafeArea, Scaffold
    );  // MaterialApp
  }
```

mainAxisAlignment: MainAxisAlignment.center,
crossAxisAlignment: CrossAxisAlignment.end,



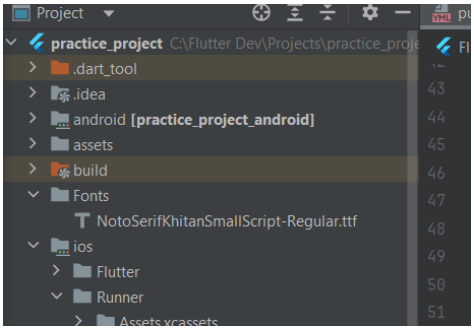
Row Widget inside Column Widget
to do that in the children array of column just simply write “Row()” and follow the same structure.

```
        body: SafeArea(
          child: Column(
            mainAxisAlignment: MainAxisAlignment.start,
            crossAxisAlignment: CrossAxisAlignment.start,
            children: [
              Row(
                children: [ Text("Containers")],
              ),  // Row
              Container(...),  // Container
              const SizedBox(...),  // SizedBox
              Container(...),  // Container
              const SizedBox(...),  // SizedBox
              Container(...),  // Container
            ],
          ),  // Column
        )),  // SafeArea, Scaffold
    );  // MaterialApp
  }
}
```



How to add fonts in workspace

download it from google fonts → create a new directory by right clicking on main directory and name it “fonts” → copy the .ttf file from zip folder to fonts directory



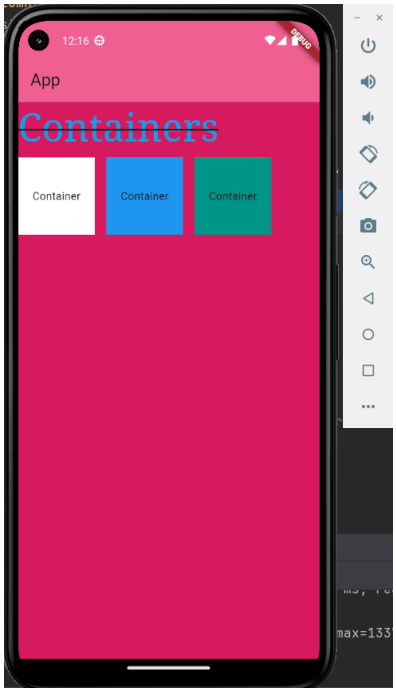
then go to pubspec.yaml and update it

```
fonts:
  - family: NotoSerifKhitanSmallScript
    fonts:
      - asset: fonts/NotoSerifKhitanSmallScript-Regular.ttf
```

make sure there is4 2 spacebar diff in “fonts”

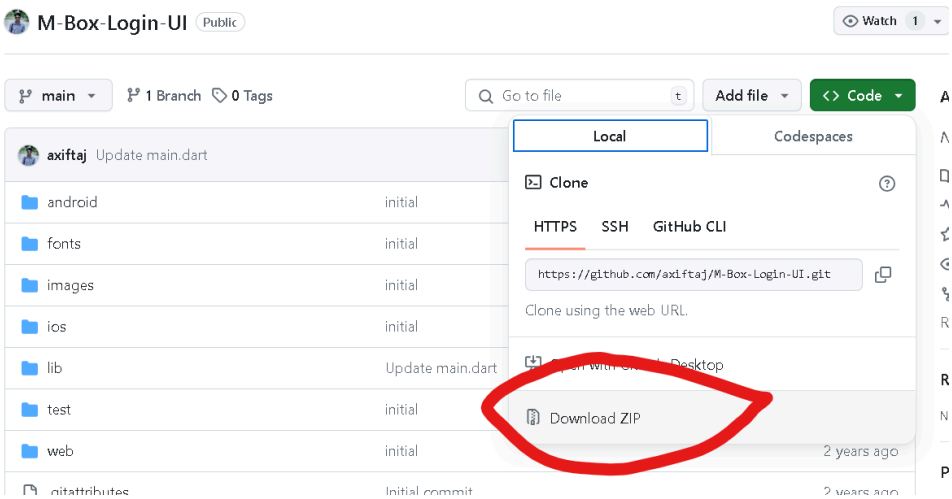
then go to “main.dart” and use it in your file

```
const Text(
 "Containers",
 style: TextStyle(
  fontFamily: 'NotoSerifKhitanSmallScript',
   decoration: TextDecoration.lineThrough,
  fontSize: 50,
 ),
),
```



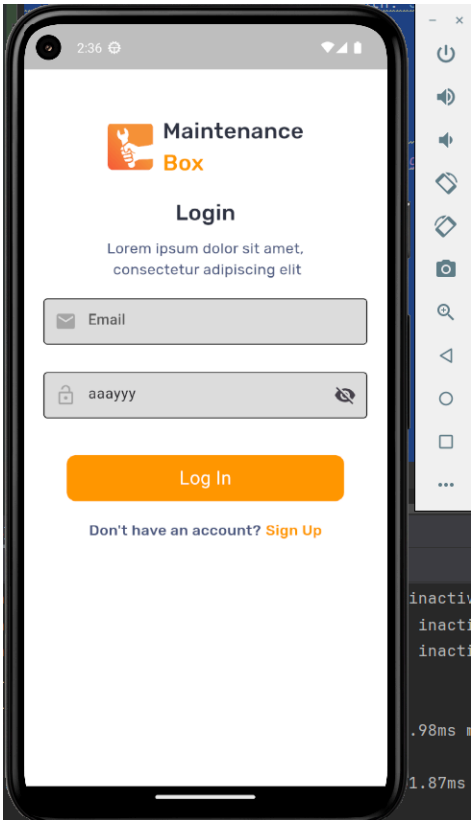
HOW TO CLONE PROJECT IN FLUTTER?

go to github and download zip file



extract it then open it in android studios
in main directory open terminal and type "**flutter pub get**" - *basically extract assets etc and updates yaml file*

LOGIN UI FROM SCRATCH



Import Statements

import 'package:flutter/cupertino.dart';

import 'package:flutter/material.dart';

- **Purpose:** These lines import the necessary Flutter packages.

  - `cupertino.dart`: Contains iOS-style widgets.

  - `material.dart`: Contains Material Design widgets, which are the most commonly used in Flutter for creating Android-style UIs.

The `main` Function

```dart

void main() {

  runApp(const MyApp());

}

```

- **Purpose:** The `main` function is the entry point of the application.

  - `runApp(MyApp())`: Tells Flutter to start the app and use the `MyApp` widget as the root of the widget tree.

  - `const MyApp()`: Creates an instance of the `MyApp` widget with `const` to indicate that it's a constant and won't change.

The `MyApp` Widget

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

- **Purpose:** `MyApp` is a widget, which is a building block in Flutter. It is a subclass of `StatelessWidget`, meaning it represents part of the UI that doesn't change.

  - `const MyApp({Key? key})`: This is the constructor. The `Key` is used to identify the widget in the widget tree, but it's optional.

Building the UI

```dart
@override
Widget build(BuildContext context) {
  return MaterialApp(
    debugShowCheckedModeBanner: false,
    home: Scaffold(
      backgroundColor: Colors.white,
      body: SafeArea(
        child: Column(
          children: [
```

```

- **Purpose:** The `build` method is called when Flutter needs to render the UI. Here's what each part does:

  - `MaterialApp`: The root widget that holds the entire app. It provides navigation, theming, and more.

    - `debugShowCheckedModeBanner: false`: Hides the debug banner that appears in the top right corner during development.

  - `Scaffold`: A layout structure that provides common visual elements such as the app bar, drawer, and floating action button.

    - `backgroundColor: Colors.white`: Sets the background color to white.

  - `SafeArea`: Ensures that the content is displayed within the safe areas of the device (like notches or status bars).

  - `Column`: A widget that arranges its children in a vertical column.

### Adding Spacing

```dart
const SizedBox(
  height: 50,
),
```

- **Purpose:** `SizedBox` is used to create a fixed amount of space in the UI.

  - `height: 50`: Adds a vertical space of 50 pixels.

The Row for Logo and Title

```dart
Row(
  mainAxisAlignment: MainAxisAlignment.center,
```
```

```
children: [
  const Image(
    height: 50,
    width: 50,
    image: AssetImage('images/logo.png'),
  ),
  const SizedBox(
    width: 10,
  ),
  const Column(
    crossAxisAlignment: CrossAxisAlignment.start,
    children: [
      Text(
        "Maintenance",
        style: TextStyle(
          fontSize: 24,
          fontFamily: 'rubik medium',
          color: Color(0xff2D3142),
        ),
      ),
      SizedBox(height: 0.5),
      Text(
        "Box",
        style: TextStyle(
          fontSize: 24,
          fontFamily: 'rubik medium',
          color: Colors.orange,
        ),
      ),
```

```
    ],
   ),
  ],
),
```

- **Purpose:** This part creates a row containing the app's logo and title.
  - `Row`: A widget that arranges its children horizontally.
    - `mainAxisAlignment: MainAxisAlignment.center`: Centers the children within the row.
  - `Image`: Displays an image from the asset directory.
    - `height` and `width`: Define the image size.
  - `SizedBox(width: 10)`: Adds horizontal space between the image and the text.
  - `Column`: Arranges the two texts ("Maintenance" and "Box") vertically.
    - `crossAxisAlignment: CrossAxisAlignment.start`: Aligns the text to the start (left side) of the column.
  - `Text`: Displays a piece of text with a specific style.
    - `TextStyle`: Defines the appearance of the text (font size, family, and color).

Centered Login Title

```dart
const SizedBox(
  height: 20,
),
const Center(
  child: Text(
    "Login",
    style: TextStyle(
      fontSize: 24,
```

```
      fontFamily: 'rubik medium',
      color: Color(0xff2D3142),
    ),
  ),
),
```

- **Purpose:** This part creates a centered "Login" title.
  - `SizedBox(height: 20)`: Adds vertical space.
  - `Center`: Centers the child widget within its parent.
  - `Text`: Displays the "Login" text with specified styling.

Subtitle Below Login

```dart
const SizedBox(
  height: 10,
),
const Center(
  child: Text(
    "Lorem ipsum dolor sit amet, \n consectetur adipiscing elit",
    textAlign: TextAlign.center,
    style: TextStyle(
      fontSize: 16,
      fontFamily: 'rubik regular',
      color: Color(0xff4C5980),
    ),
  ),
),
```

```

- **Purpose:** This creates a subtitle text below the "Login" title.
  - `SizedBox(height: 10)`: Adds vertical space.
  - `Center`: Centers the text.
  - `Text`: Displays a placeholder subtitle text in two lines, aligned to the center.

### Email Input Field

```dart
const SizedBox(height: 20),
TextFormField(
  decoration: const InputDecoration(
    fillColor: Colors.black12,
    filled: true,
    hintText: 'Email',
    prefixIcon: Icon(
      Icons.email,
      color: Colors.black26,
    ),
    enabledBorder: OutlineInputBorder(
      borderSide: BorderSide(color: Colors.black),
    ),
    focusedBorder: OutlineInputBorder(
      borderSide: BorderSide(color: Colors.black),
    ),
  ),
),
```
```

- **Purpose:** This part creates a text input field for the user to enter their email.

  - `SizedBox(height: 20)`: Adds vertical space.

  - `TextFormField`: A widget for user input that can handle form validation.

    - `decoration`: Customizes the appearance of the input field.

      - `fillColor` and `filled`: Adds a background color to the input field.

      - `hintText`: Placeholder text when the input field is empty.

      - `prefixIcon`: An icon displayed at the beginning of the input field.

      - `enabledBorder` and `focusedBorder`: Define the border appearance when the field is enabled or focused.

Login Button

```dart
const SizedBox(height: 100),
Container(
  height: 50,
  width: 300,
  decoration: BoxDecoration(
    color: Colors.orange,
    borderRadius: BorderRadius.circular(10),
  ),
  child: const Center(
    child: Text(
      "Log In",
      style: TextStyle(color: Colors.white, fontSize: 20),
    ),
  ),
),
```

```

- **Purpose:** This creates a button for logging in.
  - `SizedBox(height: 100)`: Adds vertical space before the button.
  - `Container`: A box model for styling elements.
    - `height` and `width`: Define the button size.
    - `decoration`: Adds visual decoration to the container.
      - `color`: Sets the button color to orange.
      - `borderRadius`: Rounds the corners of the button.
  - `Center`: Centers the "Log In" text within the button.
  - `Text`: Displays the "Log In" text with specified styling.

### Sign Up Text

```dart
const SizedBox(height: 20),
Row(
  mainAxisAlignment: MainAxisAlignment.center,
  children: [
    const Text(
      "Don't have an account?",
      style: TextStyle(
        fontSize: 16,
        fontFamily: 'rubik medium',
        color: Color(0xff4C5980),
      ),
    ),
    const SizedBox(width: 5),
    const Text(
```
```

```
    "Sign Up",
    style: TextStyle(
      fontSize: 16,
      fontFamily: 'rubik medium',
      color: Colors.orange,
    ),
  ),
 ],
),
```

- **Purpose:** This part creates a text asking the user if they have an account and provides an option to sign up.

- `SizedBox(height: 20)`: Adds vertical space.

- `Row`: Arranges the two texts ("Don't have an account?" and "Sign Up") horizontally.

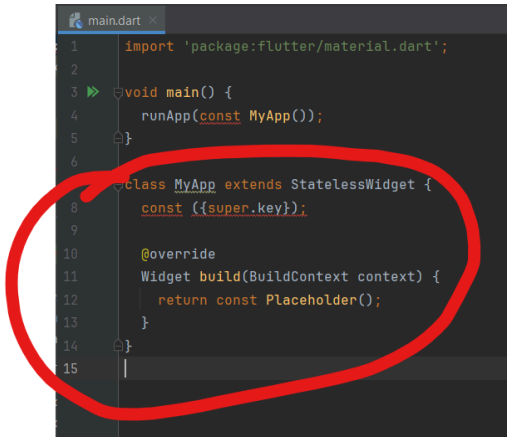
- `mainAxisAlignment: MainAxisAlignment.center`: Centers the texts within the row.

- `Text`: Displays the two pieces of text with different styles, where "Sign Up" is orange to make it stand out as a clickable option.

chapter 3

Creating stateless widgets

type “stl” and hit enter=> it will create a boiler code for stateless widget.



the class name MyApp should match with
runApp(const _____)
&
const _____ ({super.key})

always replace Placeholder() with MaterialApp()
MaterialApp has *home* property which has property *body*
]

Floating Action Button

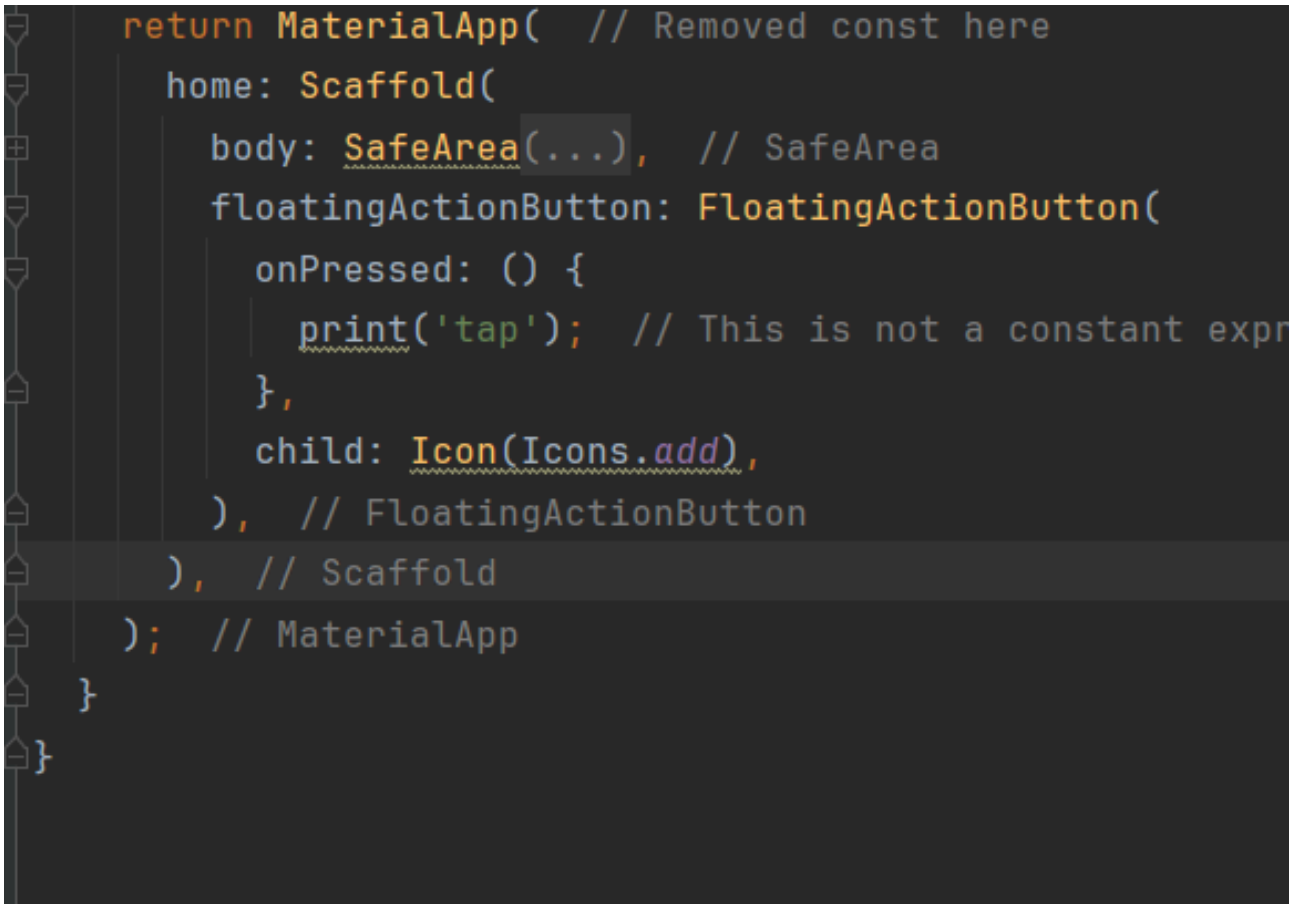
A `FloatingActionButton` (FAB) in Flutter is a button that typically "floats" above the content of the app, usually in the bottom-right corner of the screen. It is often used for a primary action in an application.

Child Widget: The `FloatingActionButton` accepts a `child` widget, usually an `Icon` or a `Text`, which is displayed at the center of the button.

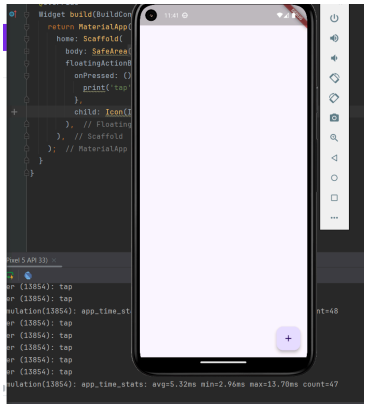
onPressed: It also accepts an `onPressed` callback function, which is triggered when the button is tapped.

- The **FAB** is usually a direct child of a `Scaffold` widget, placed within the `floatingActionButton` property.

- It's used for a main or primary action in the UI, like adding an item or starting a new task.



It is usually used as a direct child of the Scaffold widget's floatingActionButton property



Understanding variables

if use “final” with a variable, its value doesnt change
it can be used in Text widget where
it can be converted to string using .tostring()

```
      body: SafeArea(
        child: Center(child: Text(x.toString(),style: TextStyle(fontSize: 50)))
        ,
      ),  // SafeArea
      floatingActionButton: FloatingActionButton(...),  // FloatingActionButton
    ),  // Scaffold
  );  // MaterialApp
 }
}
```

Hot Reload Behavior:

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

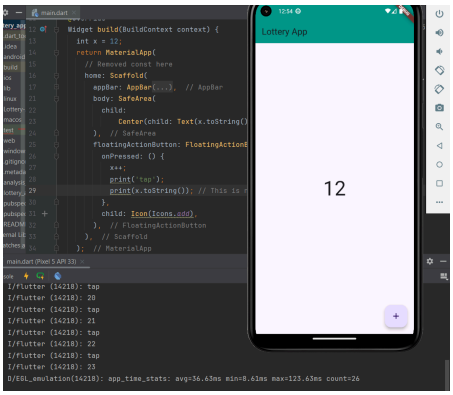
class MyApp extends StatelessWidget {
  MyApp({super.key});
 //int x = 12;

  @override
  Widget build(BuildContext context) {
    int x = 12;
    return MaterialApp(...);  // MaterialApp
  }
}
```

- if we write int x = 12 inside Widget build... then it will accurately display the value after i change it but if i write in in MyApp... meaning outside Widget build then it will display the previous value like if it was 11 earlier after changing x to 12 it will still display 11 as hot reload starts working from Widget

  Inside Widget build() Method:
  - Variables declared inside build() are reinitialized with each rebuild. Any changes you make to these variables will immediately reflect after a hot reload because the build() method is called again.
- **Outside Widget build() Method** (e.g., at the class level):
  - Variables declared outside build() are initialized only when the widget is first created. If you change their value and do a hot reload, the value won't update because the widget instance is not recreated. The old value persists until a full restart.

Stateless Widget(disadvantage)



whenever we are tapping on the add button the value being displayed is not incrementing on the screen-this is where stateful widgets come in handy.
stateless widgets donot rebuild

```
class MyApp extends StatelessWidget {
  MyApp({super.key});
int x = 12;

  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      // Removed const here
      home: Scaffold(
        appBar: AppBar(
          backgroundColor: Colors.teal,
          title: Text("Lottery App"),
        ),
        body: SafeArea(
          child:
              Center(child: Text(x.toString(), style: TextStyle(fontSize: 50))),
        ),
        floatingActionButton: FloatingActionButton(
          onPressed: () {
            x++;
            print('tap');
            print(x.toString()); // This is not a constant expression
          },
          child: Icon(Icons.add),
        ),
      ),
    );
  }
}
```

The original code used a `**StatelessWidget**` to create a simple app that displays an integer value (`**x**`) and updates it when a `**FloatingActionButton**` is pressed. However, since `**StatelessWidget**` cannot maintain or update state, the `**x**` value increment inside the `onPressed` callback did not cause the UI to update. To properly manage and reflect changes in the UI when `**x**` is incremented, the widget needed to be converted to a `**StatefulWidget**`, which allows state to be maintained and updated dynamically, ensuring that the UI reflects changes in `**x**` when the button is pressed.

STATEFUL WIDGETS

type *stf* for stateful widget boiler code

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

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

  class MyApp extends StatefulWidget {
    const MyApp({super.key});
    @override
    State<MyApp> createState() => _MyAppState();
  }

  class _MyAppState extends State<MyApp> {
    int x = 0;

    @override
    Widget build(BuildContext context) {
      return MaterialApp(
        // Removed const here
        home: Scaffold(
          appBar: AppBar(
            backgroundColor: Colors.teal,
            title: Text("Lottery App"),
          ),
          body: SafeArea(
            child:
            Center(child: Text(x.toString(), style: TextStyle(fontSize: 50))),
          ),
          floatingActionButton: FloatingActionButton(
            onPressed: () {
              x++;
              setState(() {

              });
              print('tap');
              print(x.toString()); // This is not a constant expression
            },
            child: Icon(Icons.add),
          ),
        ),
      );
    }
  }
```

This code defines a `**StatefulWidget**` called `**MyApp**`, which manages an integer variable `**x**` that is displayed in the center of the screen. The `**x**` variable is updated each time the `**FloatingActionButton**` is pressed. The UI is rebuilt to reflect this updated value using the `setState` method, which triggers a rebuild of the widget tree. The `**MaterialApp**` wraps the main structure, and the incremented value of `**x**` is displayed in a `**Text**` widget, ensuring that the app dynamically updates the displayed value when the button is tapped.

Building a lottery App

Procedure for Building the Lottery App So Far:

1. **Project Setup**:

   - The code starts with importing the `material.dart` package, which provides the necessary Flutter components.

   - The `main()` function is defined, which is the entry point of the Flutter app. It calls `runApp()` with `MyApp()` as the argument to start the application.

2. **Creating a Stateful Widget**:

   - The `MyApp` class is created as a `StatefulWidget`, allowing the app to maintain and update the state dynamically.

   - The `MyApp` class overrides the `createState()` method, which returns an instance of `_MyAppState`. This private state class will manage the state for `MyApp`.

3. **Managing State with _MyAppState**:

   - The `_MyAppState` class extends `State<MyApp>` and includes an integer variable `x` initialized to 4. This variable represents the "lottery winning number" and can change over time.

   - The `build()` method is overridden to construct the UI. The `MaterialApp` widget is used to define the app's theme and structure.

4. **Building the UI**:

   - Inside the `Scaffold` widget, an `AppBar` is provided with the title "Lottery App" and a background color of teal.

   - The main content of the app is placed in the `body` property of the `Scaffold`, wrapped in a `SafeArea` to avoid overlaps with system interfaces.

   - A `Column` is used to center the content vertically and horizontally, displaying the text "Lottery winning number is $x" in the middle of the screen, where `$x` dynamically shows the current value of `x`.

5. **Adding Interaction with FloatingActionButton**:

   - A `FloatingActionButton` is added at the bottom-right of the screen. When pressed, the `onPressed` callback increments the value of `x` and calls `setState()`.

- The `setState()` method triggers a rebuild of the widget tree, updating the UI to display the new value of `x`.

- The button is represented by an "add" icon, which visually indicates that the action will increase the number.

6. **Output**:

- The app now displays the "lottery winning number" on the screen, and each press of the floating button increases this number by 1, with the updated number immediately shown on the screen.

This structure sets up the basic functionality of a simple lottery app, with a dynamic number that changes upon user interaction.

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

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

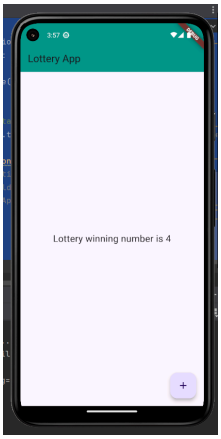
  class MyApp extends StatefulWidget {
    const MyApp({super.key});
    @override
    State<MyApp> createState() => _MyAppState();
  }

  class _MyAppState extends State<MyApp> {
    int x = 4;

    @override
    Widget build(BuildContext context) {
      return MaterialApp(
        // Removed const here
        home: Scaffold(
          appBar: AppBar(
            backgroundColor: Colors.teal,
            title: Text("Lottery App"),
          ),
          body: SafeArea(
            child:
            Column(
                mainAxisAlignment: MainAxisAlignment.center,
                crossAxisAlignment: CrossAxisAlignment.center,
                children:[ Center(child: Text('Lottery winning number is $x', style: TextStyle(fontSize: 20))),
        ]
      )
      ),
          floatingActionButton: FloatingActionButton(
            onPressed: () {
              x++;
```

```
            setState(() {

            });
            print('tap');
            print(x.toString()); // This is not a constant expression
          },
          child: Icon(Icons.add),
        ),
      ),
    );
  }
}
```



ternary operators- those operators that return true or false based on the given condition.

```
Text(x > 5 ? 'x is greater than 5' : x.toString()),
```

x>5 → condition

? if true then

: if false then

all widgets accept ternary operators

```
child: Icon( x>20?Icons.refresh: Icons.add),
```

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

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

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

  @override
  State<MyApp> createState() => _MyAppState();
}

class _MyAppState extends State<MyApp> {
  Random random = Random();
  int x = 4;
```

```
@override
Widget build(BuildContext context) {
  return MaterialApp(
    home: Scaffold(
      appBar: AppBar(
        backgroundColor: Colors.teal,
        title: Text("Lottery App"),
      ),
      body: SafeArea(
        child: Column(
          mainAxisAlignment: MainAxisAlignment.center,
          crossAxisAlignment: CrossAxisAlignment.center,
          children: [
            Center(
              child: Text(
                'Lottery winning number is $x',
                style: TextStyle(fontSize: 20),
              ),
            ),
            SizedBox(height: 20),
            Container(
              height: 250,
              width: 250,  // Ensure the container has enough width
              decoration: BoxDecoration(
                color: Colors.grey.withOpacity(0.2),
                borderRadius: BorderRadius.circular(10),
              ),
              child: Padding(
                padding: const EdgeInsets.all(15.0),
                child: Column(
                  mainAxisAlignment: MainAxisAlignment.center,
                  crossAxisAlignment: CrossAxisAlignment.center,
                  children: [
                    Icon(Icons.error, color: Colors.red, size: 35),  // Ensure the icon size is sufficient
                    SizedBox(height: 15),
                    Text(
                      'Better luck next time! Your number is $x. \nTry again.',
                      textAlign: TextAlign.center,
                    ),
                  ],
                ),
              ),
            )
                      ],

        ),

      ),
      floatingActionButton: FloatingActionButton(
    onPressed: (){
      x=random.nextInt(200);
```

```
      print(x);
      setState(() {

      });
    },
    child: Icon(Icons.refresh),
      ),
      )
    );
  }
}
```

Let's break down the code step by step to understand how **stateless** and **stateful** widgets work in Flutter, focusing on specific lines of code:

1. Stateless and Stateful Widgets Overview:

- **Stateless Widgets:** These are immutable and do not hold any state that changes over time. Once a stateless widget is built, it cannot change. Example: A widget that only displays static text.

- **Stateful Widgets:** These can change their state during the lifetime of the widget. When the state changes, the widget rebuilds to reflect those changes. Example: A widget that changes its appearance or data when a button is pressed.

Code Breakdown:

1. Import Statements:

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

- **Explanation:** These import the Flutter material design package and the Dart math library. The material package provides core components like `Scaffold`, `AppBar`, and `FloatingActionButton`, while the `dart:math` library is used to generate random numbers.

2. Main Function:

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

}

```

- **Explanation:** The `main()` function is the entry point of the application. It calls `runApp()` with `MyApp` as its argument, which inflates the widget tree and renders it on the screen.

#### **3. MyApp Class (StatefulWidget):**

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

  @override
  State<MyApp> createState() => _MyAppState();
}
```

- **Explanation:** `MyApp` extends `StatefulWidget`, meaning it can have mutable state. The `createState()` method returns an instance of `_MyAppState`, where the mutable state of this widget is defined and managed.

4. _MyAppState Class (State Management):

```dart
class _MyAppState extends State<MyApp> {
  Random random = Random();
  int x = 4;

  @override
  Widget build(BuildContext context) {
```

- **Explanation:** `_MyAppState` extends `State<MyApp>`, which is where the state management occurs.
  - The `random` object from the `dart:math` package is used to generate random numbers.
  - The integer `x` is the state variable that holds the current random number. Initially, it is set to `4`.
```

- **Stateful Nature:** The `x` value is mutable and will change when the user interacts with the UI.

5. Widget Build Method:

```dart
    return MaterialApp(
      home: Scaffold(
        appBar: AppBar(
          backgroundColor: Colors.teal,
          title: Text("Lottery App"),
        ),
        body: SafeArea(
          child: Column(
            mainAxisAlignment: MainAxisAlignment.center,
            crossAxisAlignment: CrossAxisAlignment.center,
            children: [
              Center(
                child: Text(
                  'Lottery winning number is $x',
                  style: TextStyle(fontSize: 20),
                ),
              ),
              SizedBox(height: 20),
              Container(
                height: 250,
                width: 250,
                decoration: BoxDecoration(
                  color: Colors.grey.withOpacity(0.2),
                  borderRadius: BorderRadius.circular(10),
                ),
                child: Padding(
```

```
            padding: const EdgeInsets.all(15.0),
            child: Column(
              mainAxisAlignment: MainAxisAlignment.center,
              crossAxisAlignment: CrossAxisAlignment.center,
              children: [
                Icon(Icons.error, color: Colors.red, size: 35),
                SizedBox(height: 15),
                Text(
                  'Better luck next time! Your number is $x. \nTry again.',
                  textAlign: TextAlign.center,
                ),
              ],
            ),
          ),
        )
      ],
    ),
  ),
  floatingActionButton: FloatingActionButton(
    onPressed: () {
      x = random.nextInt(200);
      print(x);
      setState(() {});
    },
    child: Icon(Icons.refresh),
  ),
 ),
 );
 }
}
```

```

- **Explanation:**

  - **App Structure:**

    - `MaterialApp` is the root widget that wraps the entire app.

    - `Scaffold` provides the basic structure of the visual interface with an `AppBar` at the top and a `FloatingActionButton` at the bottom.

  - **AppBar:** Displays the app title "Lottery App" with a teal background.

  - **Body:**

    - **Column Widget:** Vertically arranges its children.

    - **Text Widget:** Displays the current lottery number (`x`).

    - **Container:** Holds an icon and a message, providing visual feedback if the lottery number doesn't match.

  - **FloatingActionButton:**

    - **Purpose:** Allows the user to generate a new random number by pressing the button.

    - **OnPressed Callback:**

      - **State Change:** When the button is pressed, a new random number between 0 and 199 is generated and assigned to `x`.

      - **setState():** This method tells Flutter that the state has changed and the widget tree needs to be rebuilt. Without `setState()`, the UI would not reflect the new value of `x`.

#### **Stateless vs. Stateful Explanation:**

- **Stateful:** The widget is stateful because it responds to user interactions (the button press) by updating the state (`x` value). This dynamic behavior requires a rebuild of the UI, hence the use of `StatefulWidget`.

- **Stateless:** If `x` were a constant value that never changed, the entire app could be written as a `StatelessWidget`. For example, if the lottery number were hardcoded and didn't change, there would be no need to manage state, and the app would be stateless.

By understanding this code, you can see how Flutter manages state in an application and the difference between widgets that remain static (stateless) and those that change dynamically
```

(stateful).

chapter 4

top 10 widgets

Container Widget:

```
child: Column(
 mainAxisAlignment: MainAxisAlignment.center,
 crossAxisAlignment: CrossAxisAlignment.center,
 children: [
  Center(
   child: Container(
    height: 200,
    width: 200,
    color: Colors.red,
    child: Center(
     child: Text(
      'Container 1'
     ),
    )
   ),
  )
 ],

),
```

Following are the widgets discussed above:
Here’s a one-line explanation for each part of the code:

Container: Creates a rectangular area to hold and style child widgets, defining its size and appearance.

height: 200: Sets the height of the container to 200 pixels.

width: 200: Sets the width of the container to 200 pixels.

color: Colors.red: Fills the container with a solid red color.

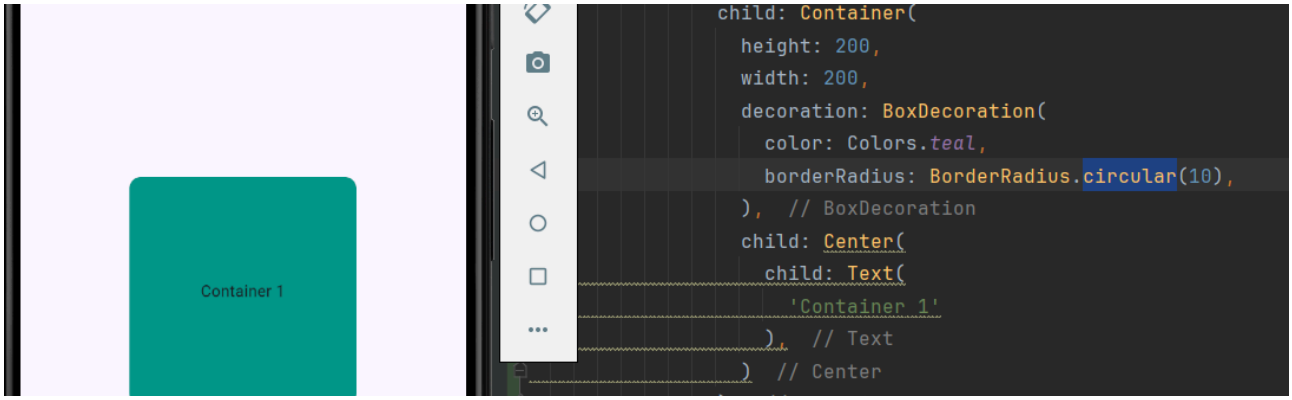
child: Center: Centers the child widget within the container.

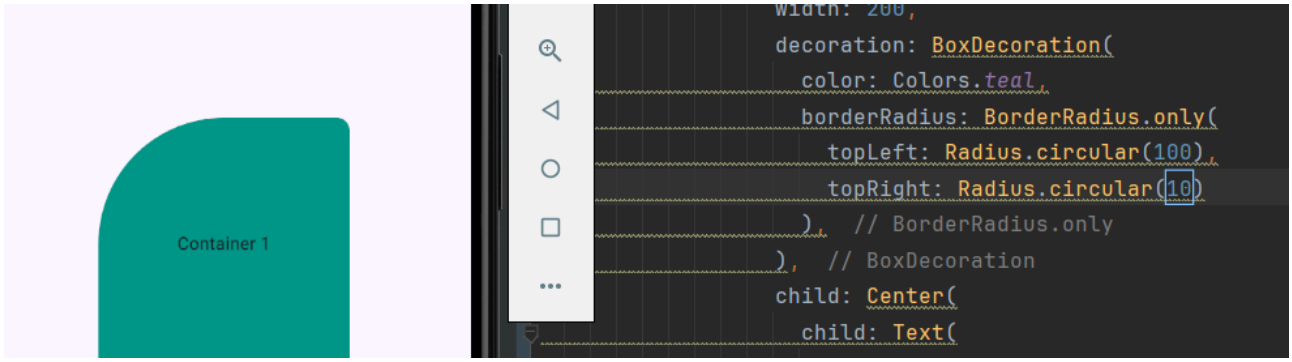
child: Text: Displays the text "Container 1" inside the container.

```
Center(
  child: Container(
    height: 200,
    width: 200,
    color: Colors.red,
    decoration: BoxDecoration(
      color: Colors.teal
    ),  // BoxDecoration
    child: Center(
      child: Text(
        'Container 1'
      ),  // Text
```

you can either give color in container widget or inside decoration property but not both else error

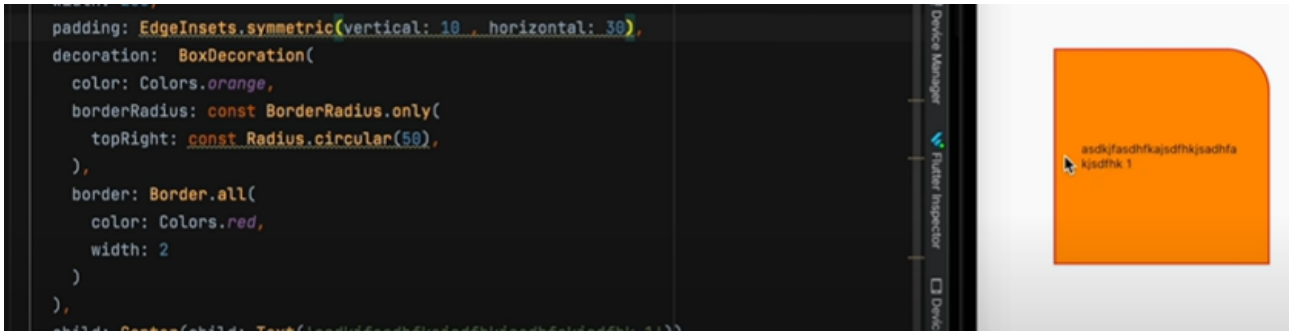
borderRadius: BorderRadius.circular(10),

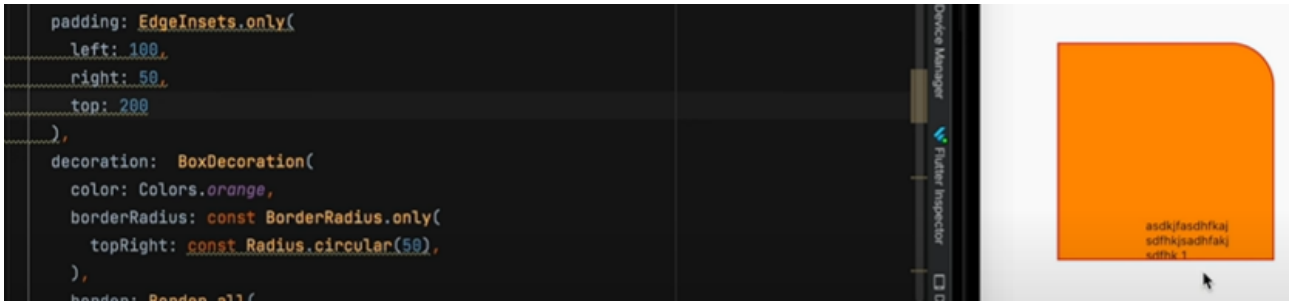




changing border color, width etc

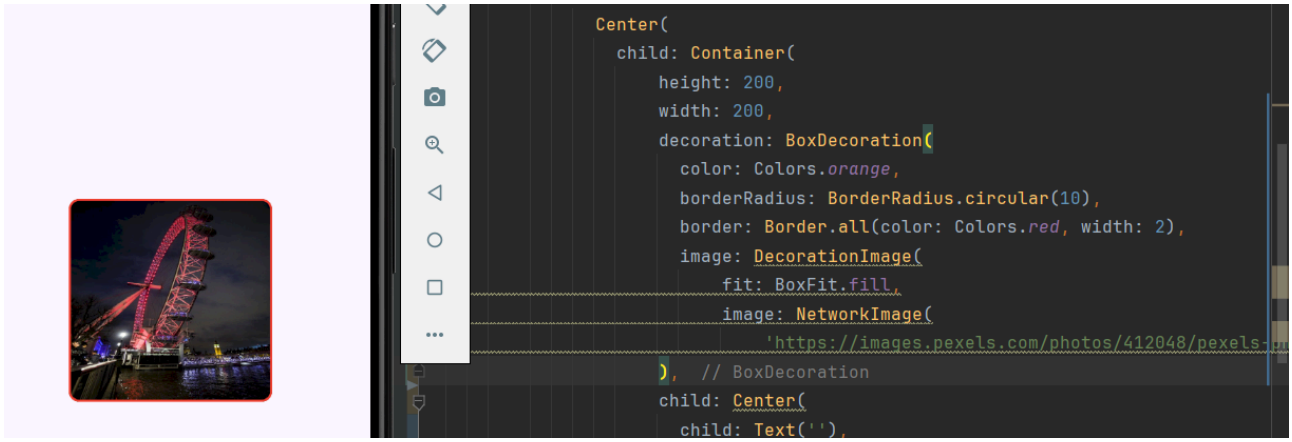
EdgeInsets in Flutter defines the amount of space around a widget's edges, effectively adding padding or margins. It is commonly used to control spacing inside a container or between widgets, allowing you to specify how much space to leave on each side (top, bottom, left, right) of the widget.



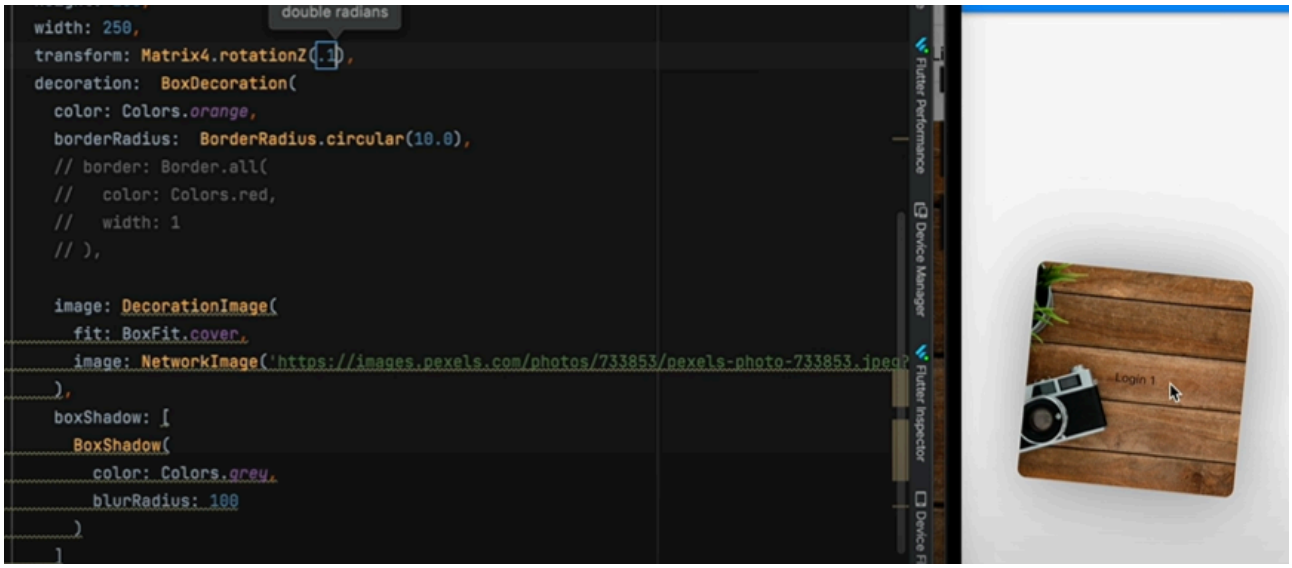


box shadow

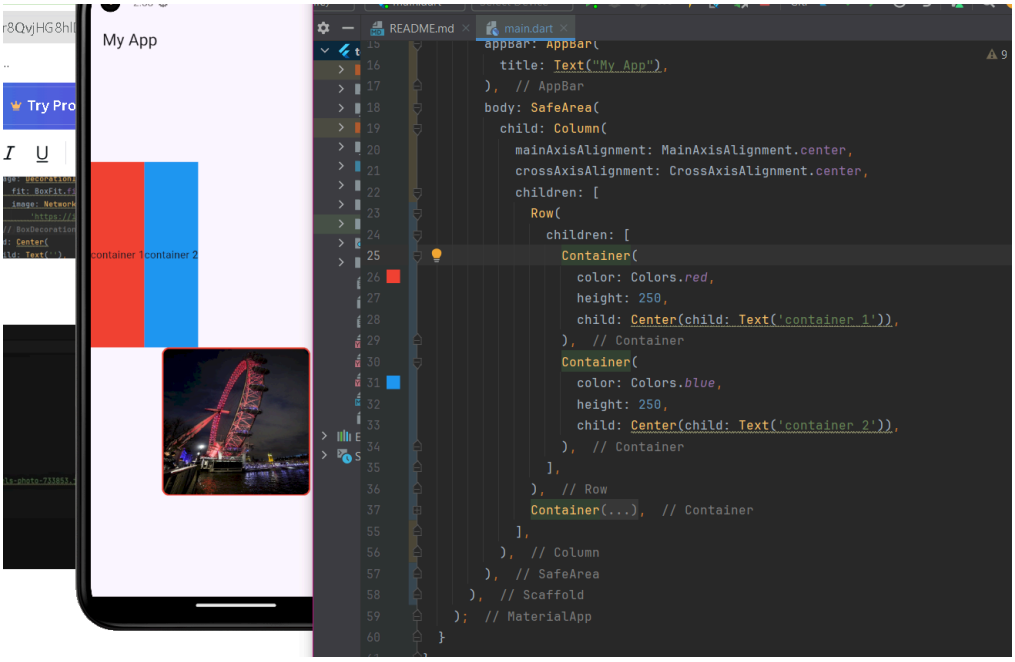
image



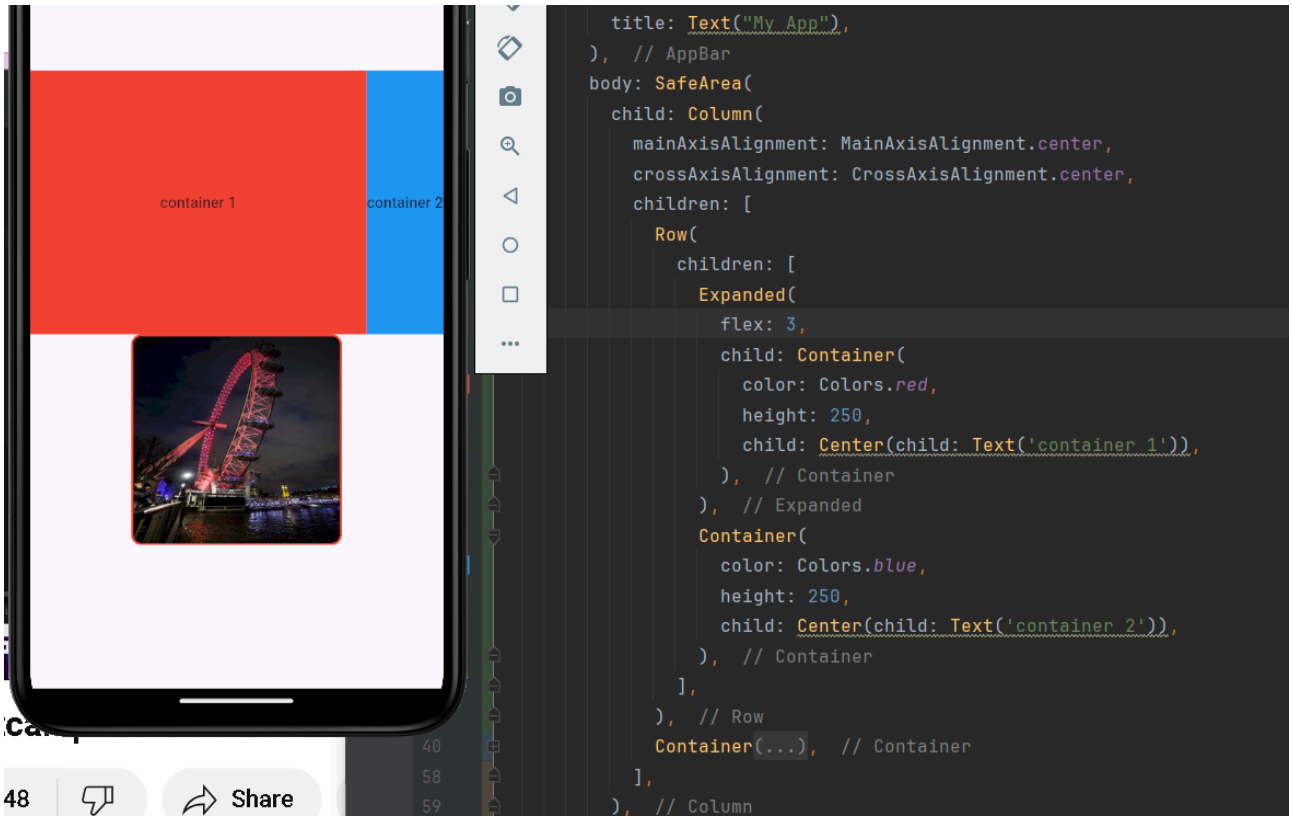
rotation

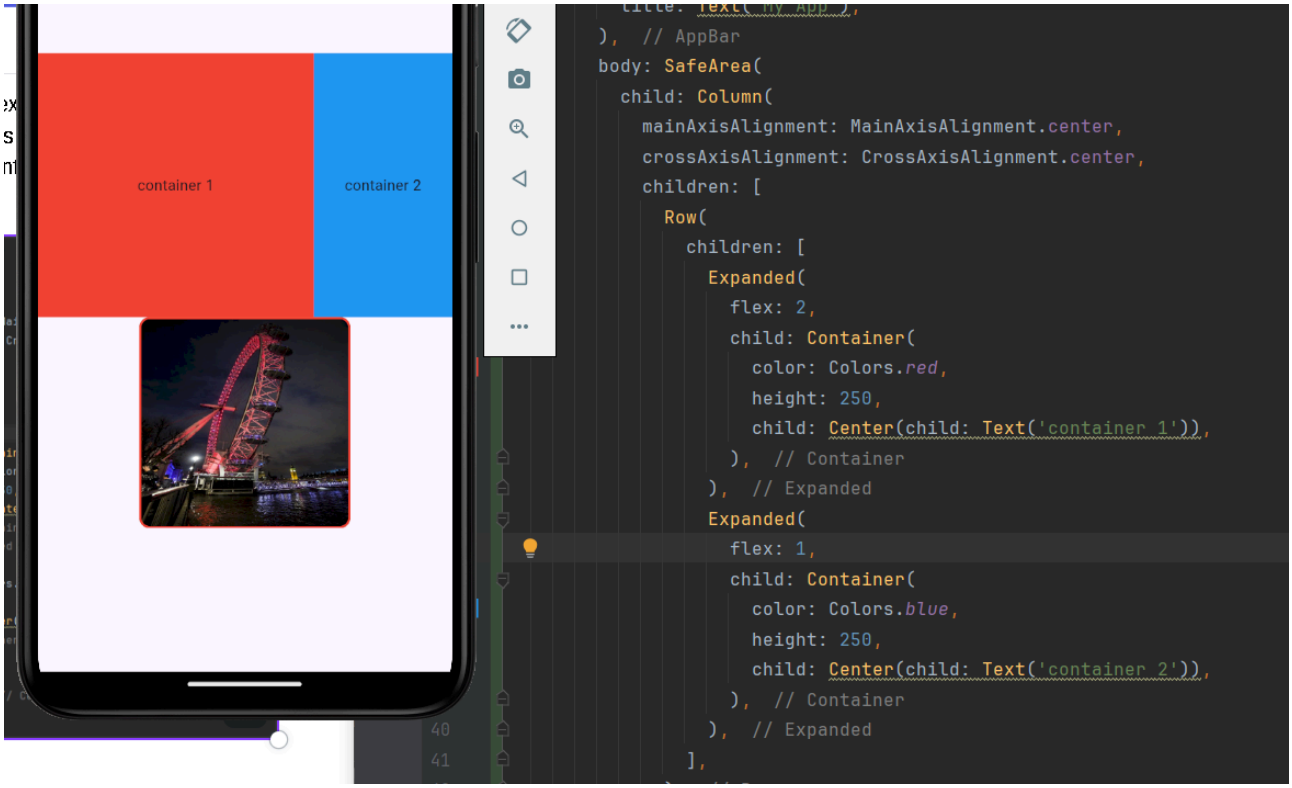


Row widget



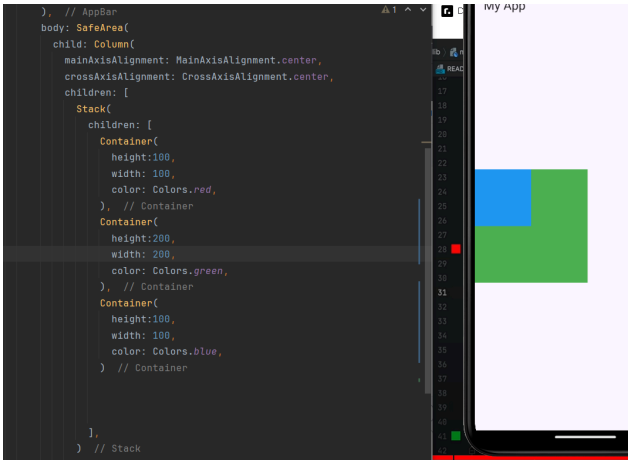
Containers bydefault take specific height, we can use expanded widget, that basically occupy full space,so we can divide it into parts khudse
75% space is occupied by container 1 remaining by container





Stack Widget

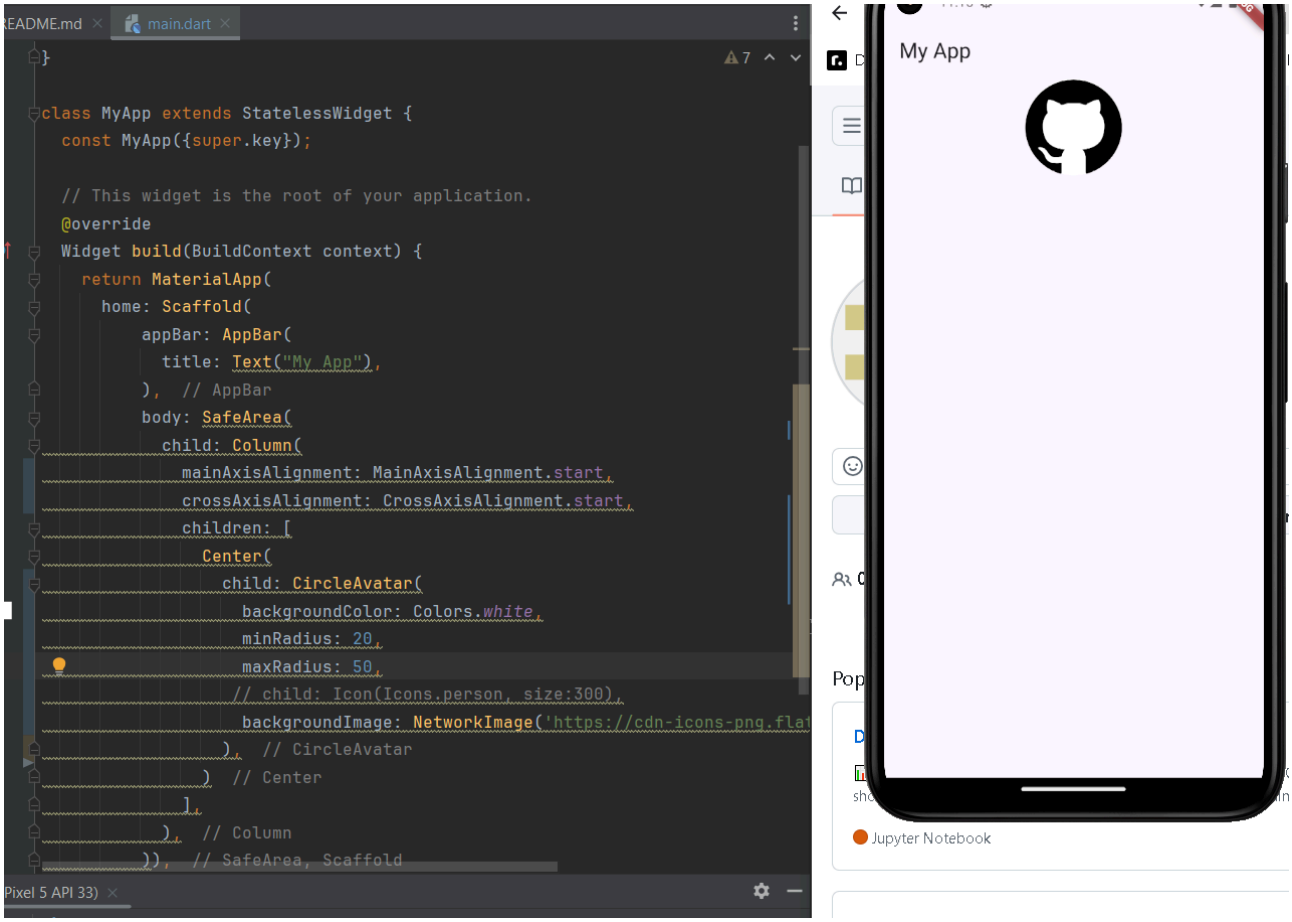
A Stack widget in Flutter is used to overlay or position multiple widgets on top of each other, allowing for layered designs.
similar to row column widget , it also has children



Stack children can be “positioned”

Circular Avatar

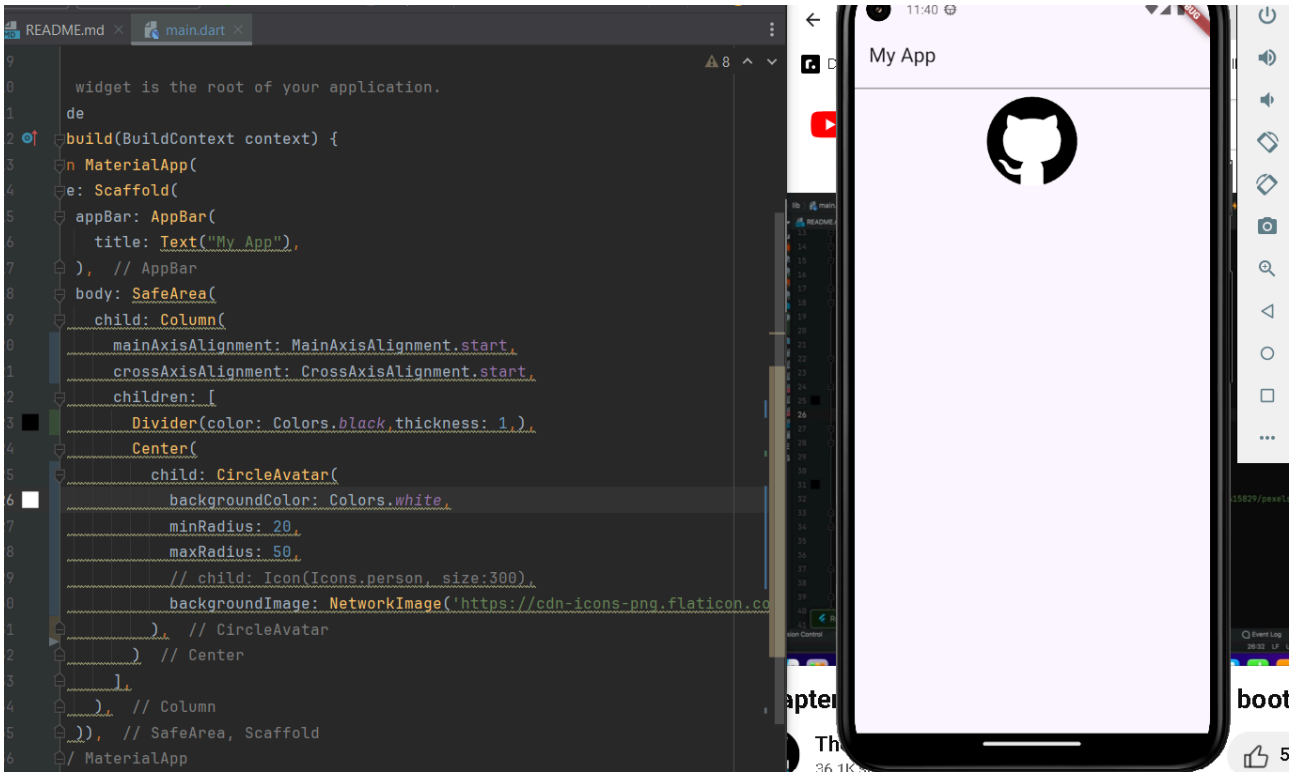
The CircleAvatar widget in Flutter is used to create a circular profile image or icon, often representing a user's profile picture.



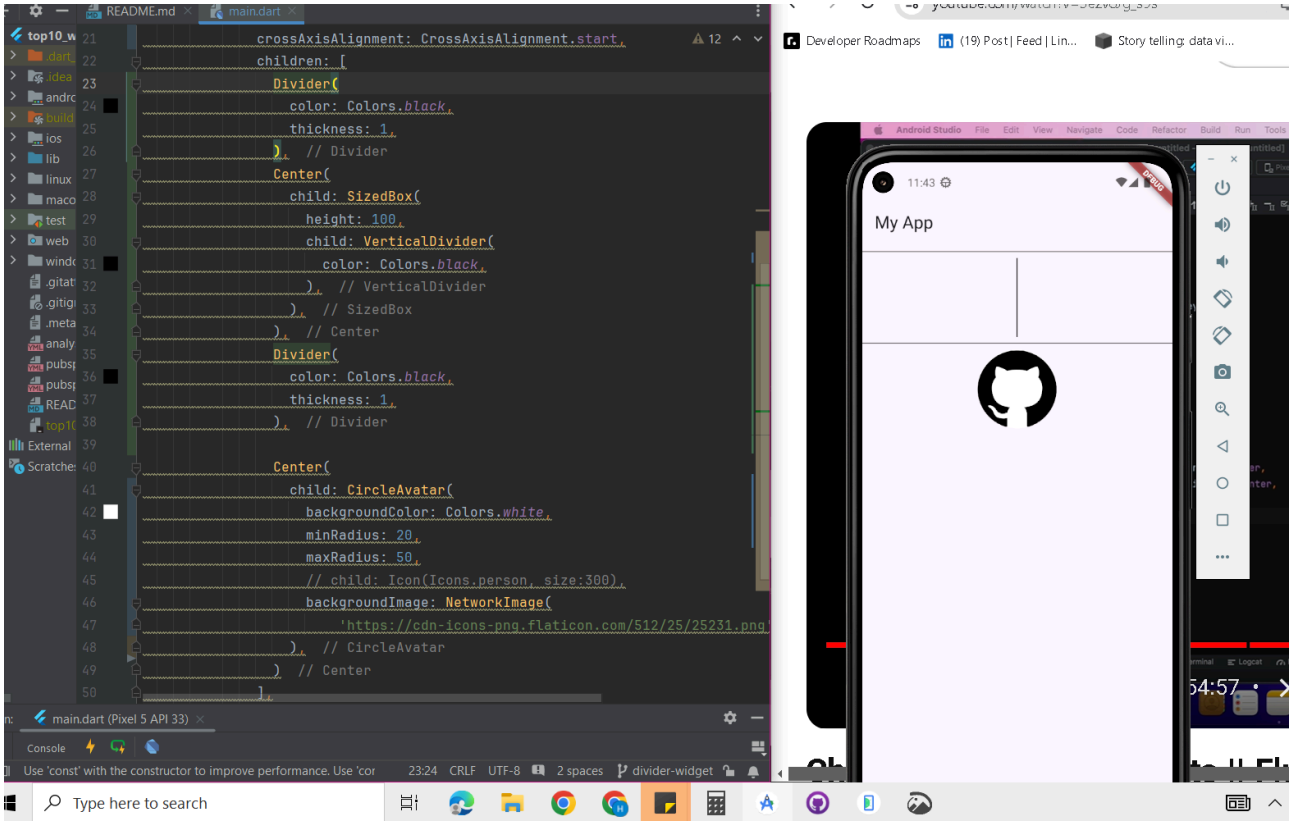
Divider Widget

The Divider widget in Flutter is used to create a thin, horizontal or vertical line that visually separates content, often used between list items or sections.

Horizontal divider



Vertical divider



Rich Text:

The RichText widget in Flutter is used to display text with multiple styles in a single block of text. It allows for complex text layouts, where you can apply different styles (like fonts, colors, and sizes) to different parts of the text by using a tree of TextSpan objects.

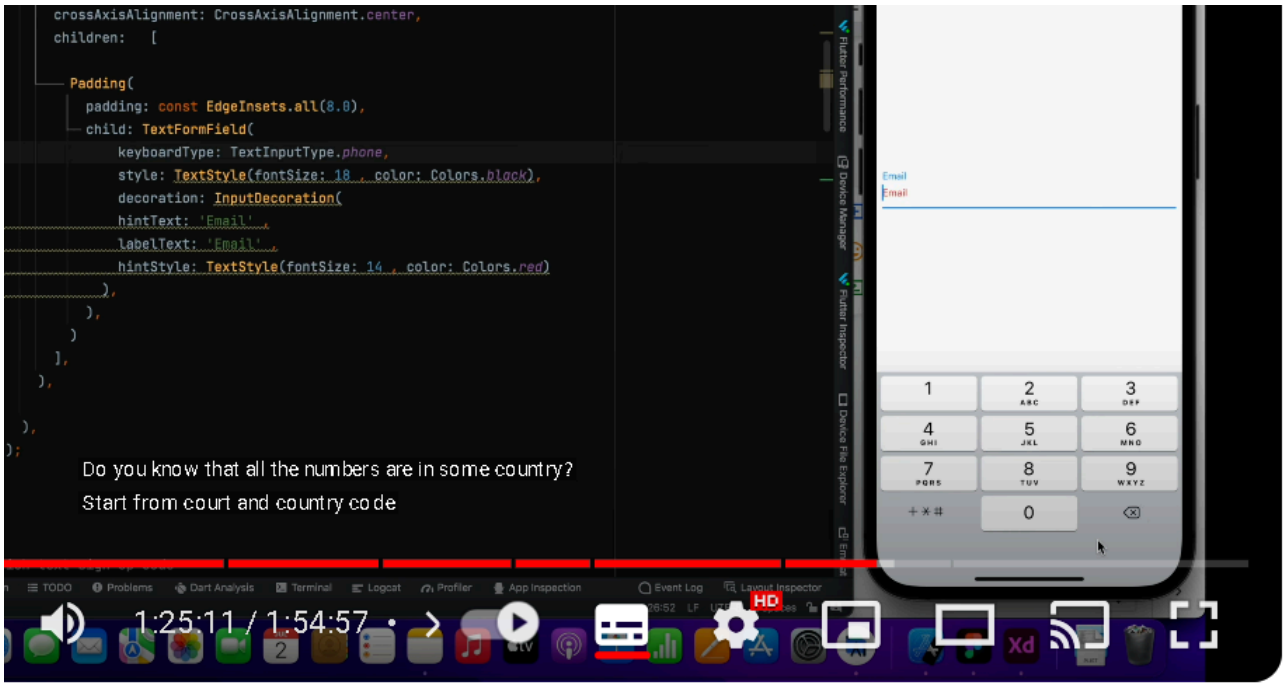
TextFormField

The TextFormField widget in Flutter is a form field that allows users to input and edit text, with additional functionality for form validation and saving. It is often used within forms and provides more features compared to the basic TextField widget.

Key Features:

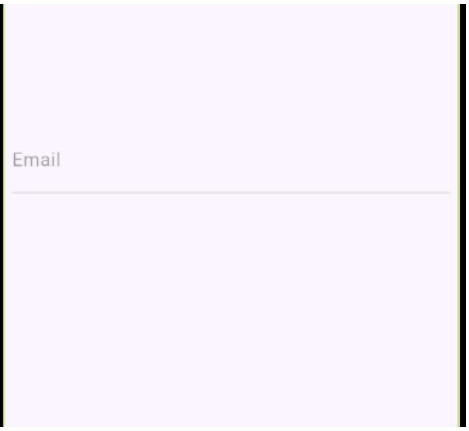
- **Validation**: You can add validation logic using the validator property, which checks the input and displays an error message if the input is invalid.
- **Input Decoration**: The decoration property allows you to style the field with labels, hints, and icons.
- **Saving Input**: The onSaved property lets you save the input to a variable when the form is submitted.
- **Keyboard Type**: You can specify the type of keyboard (e.g., number, email) using the keyboardType property.

allows to change keyboard style, label text and hint text.



use enabled:false
to make it uneditable

```
children: [
  Padding(
    padding: const EdgeInsets.all(8.0),
    child: TextFormField(
      keyboardType: TextInputType.phone,
      enabled: false,
      decoration: InputDecoration(
        hintText: 'Email',
        labelText: 'Email',
        hintStyle: TextStyle(fontSize: 13, color:Colors.blue)
      ),  // InputDecoration
    ),  // TextFormField
  )  // Padding
```

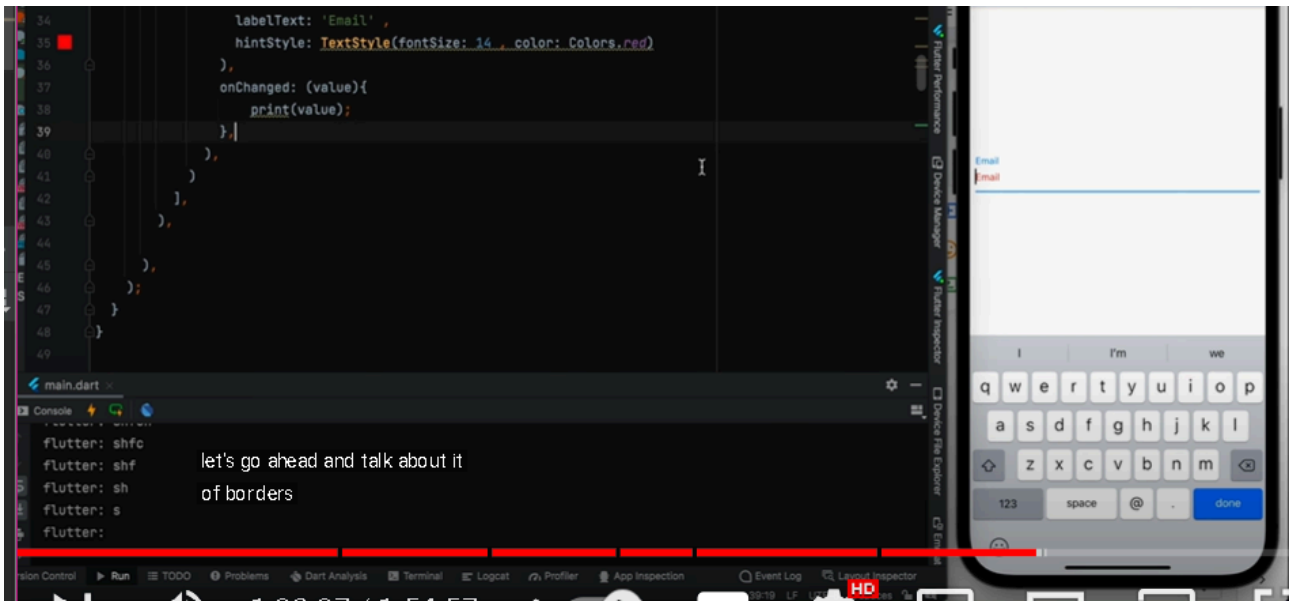


onchanged(){

}

What It Does:

- **Real-Time Feedback**: You can update the UI, validate input, or store the input value as the user types.
- **Dynamic Interactions**: Useful for implementing features like search bars, live input validation, or auto-saving data.



In Flutter, **focused** and **enabled** borders refer to the different visual styles applied to input fields, such as TextField, based on their interaction state.

1. **Focused Border:** This is the border style applied when the input field is currently active or selected by the user. It typically has a different color or thickness to indicate that the field is in focus.
2. **Enabled Border:** This is the border style applied when the input field is not currently focused but is still enabled and editable. It represents the default state of the field when it is not actively being interacted with.

List Tile

Here are explanations for each part of a `ListTile` widget in Flutter:

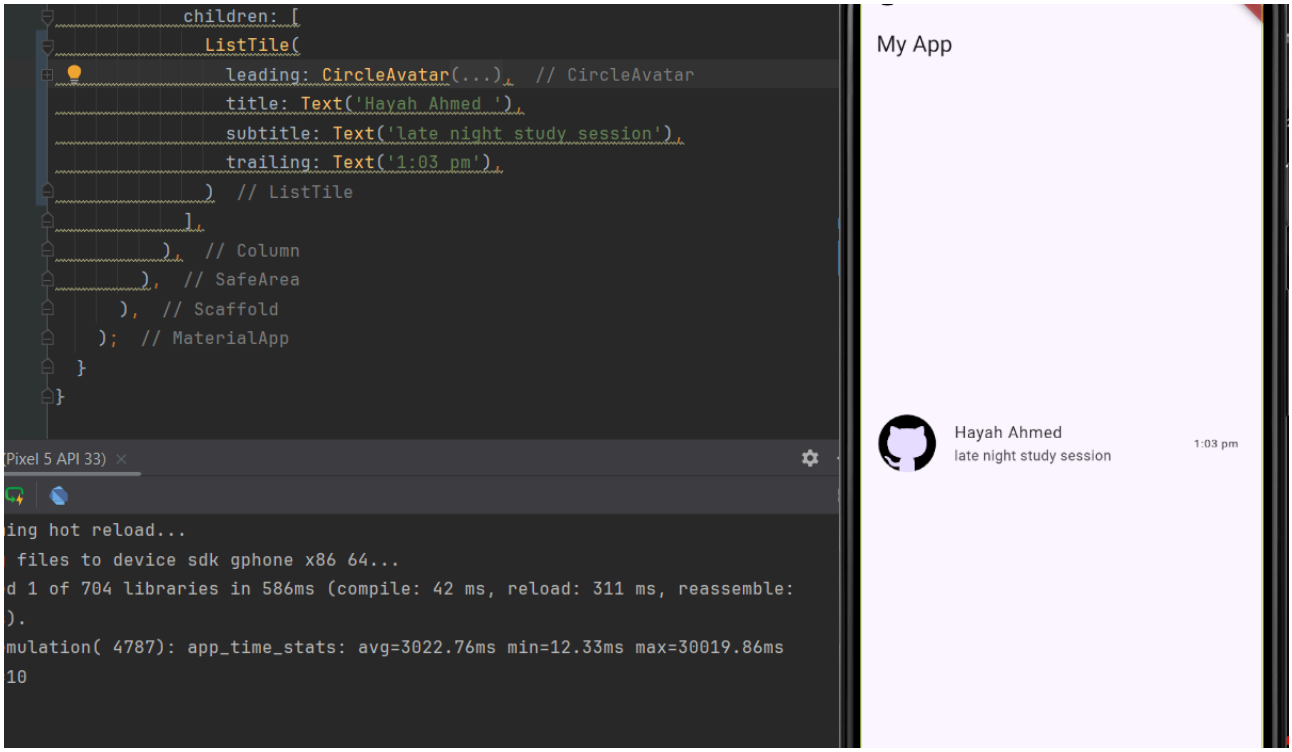
- **`ListTile`**: A widget that displays a single row of content, typically used in lists for simple items with a consistent layout.

- **`leading`**: An optional widget that appears at the beginning of the `ListTile`, often used for icons or images.

- **`title`**: The primary content of the `ListTile`, usually a `Text` widget that represents the main label of the item.

- **`subtitle`**: An optional widget displayed below the `title`, often used for additional information or a secondary label.

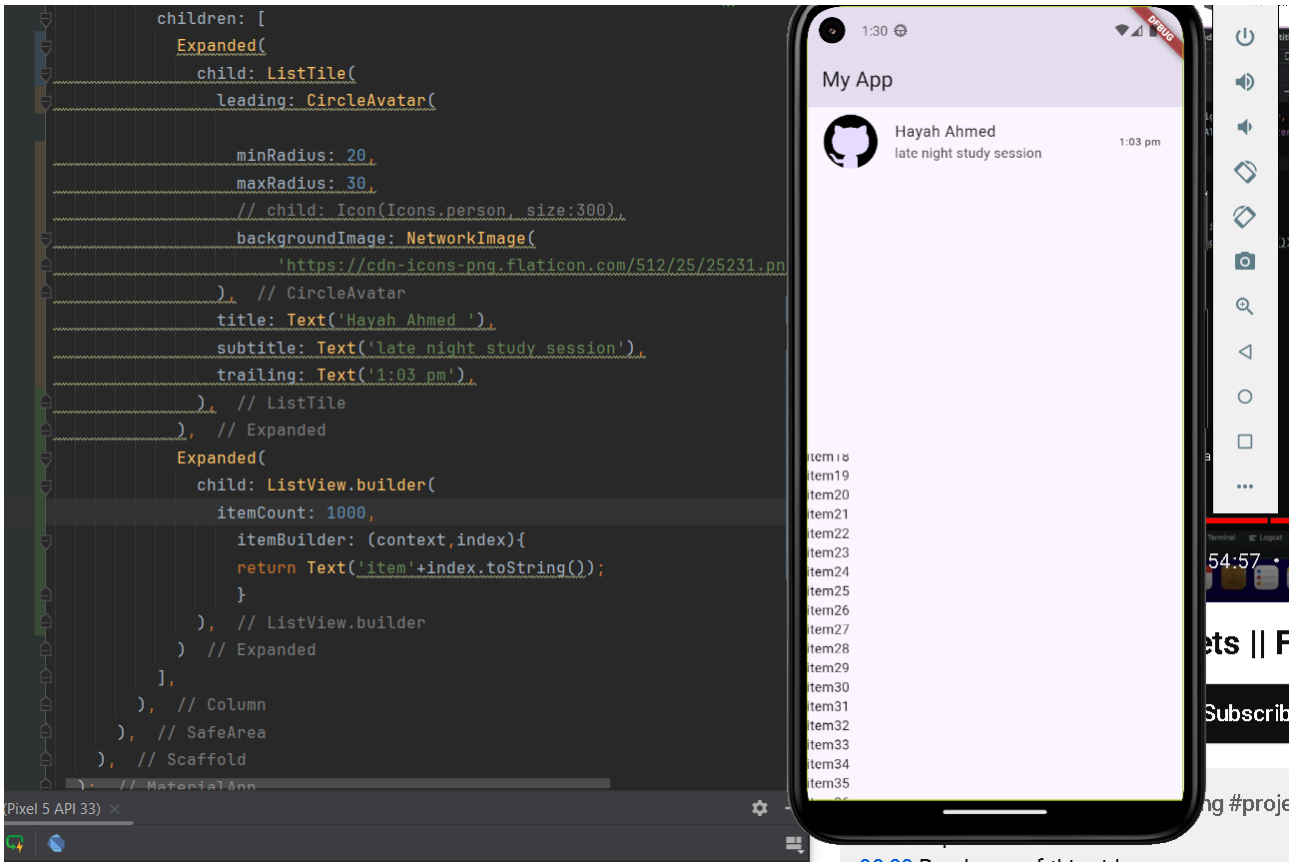
- **`trailing`**: An optional widget that appears at the end of the `ListTile`, often used for icons, buttons, or other indicators.

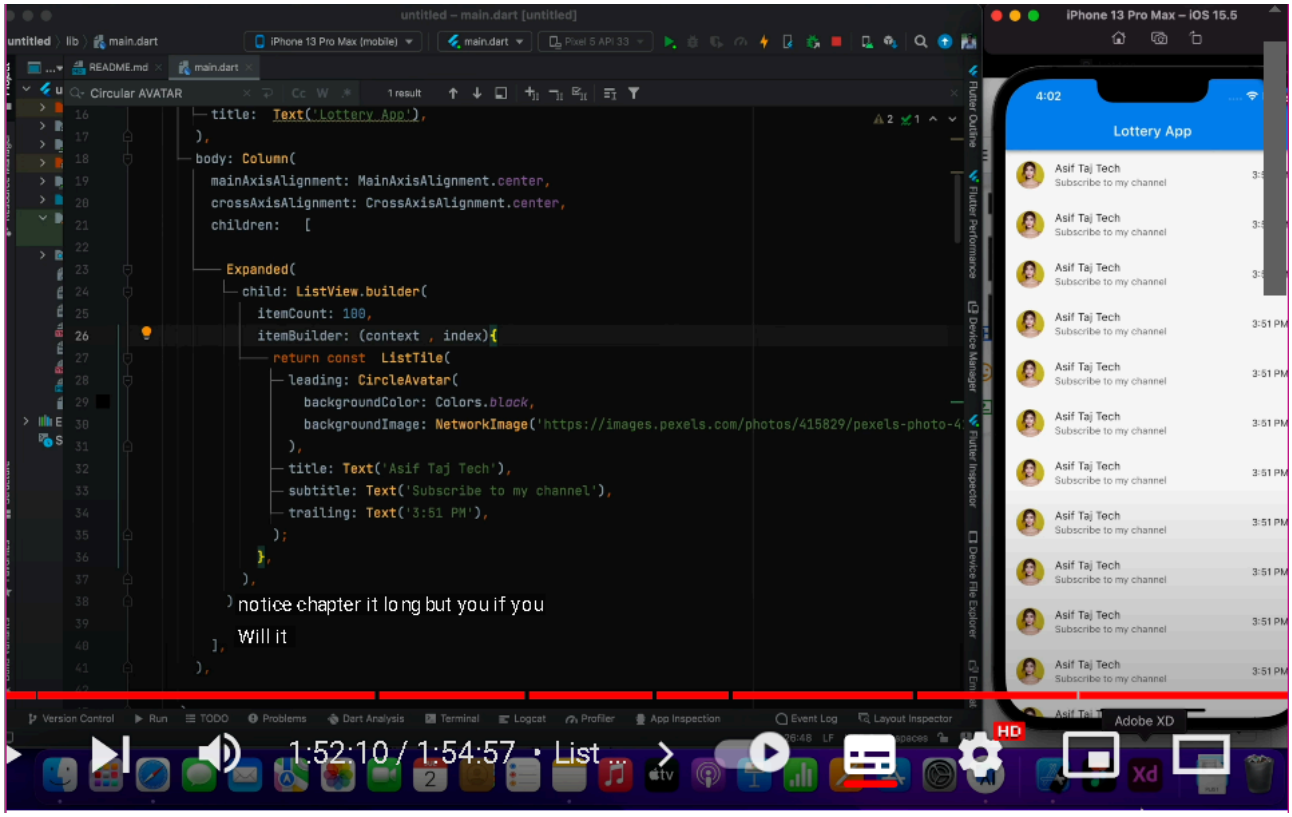


How to make a scrollable list?

List View Builder

In the given code, the `Expanded` widget is used to make the `ListTile` and `ListView.builder` widgets occupy available space in a `Column`, with `Expanded` ensuring they expand to fill the available space. The `ListView.builder` creates a scrollable list of items efficiently, by building only the items currently visible on the screen. This is achieved using the `itemBuilder` callback, which generates widgets on-demand based on the current `index`, making it suitable for large or infinite lists.


children: [
Expanded(
child: ListTile(
leading: CircleAvatar(
minRadius: 20,
maxRadius: 30,
// child: Icon(Icons.person, size:300),
backgroundImage: NetworkImage(
'https://cdn-icons-png.flaticon.com/512/25/25231.pn
), // CircleAvatar
title: Text('Hayah Ahmed '),
subtitle: Text('late night study session'),
trailing: Text('1:03 pm'),
), // ListTile
), // Expanded
Expanded(
child: ListView.builder(
itemCount: 1000,
itemBuilder: (context,index){
return Text('item'+index.toString());
}
), // ListView.builder
) // Expanded
],
), // Column
), // SafeArea
), // Scaffold
1:30
My App
Hayah Ahmed
late night study session
1:03 pm
item18
item19
item20
item21
item22
item23
item24
item25
item26
item27
item28
item29
item30
item31
item32
item33
item34
item35
(Pixel 5 API 33)


Circular AVATAR
title: Text('Lottery App'),
body: Column(
mainAxisAlignment: MainAxisAlignment.center,
crossAxisAlignment: CrossAxisAlignment.center,
children: [
Expanded(
child: ListView.builder(
itemCount: 100,
itemBuilder: (context , index){
return const ListTile(
leading: CircleAvatar(
backgroundColor: Colors.black,
backgroundImage: NetworkImage('https://images.pexels.com/photos/415829/pexels-photo-4
),
title: Text('Asif Taj Tech'),
subtitle: Text('Subscribe to my channel'),
trailing: Text('3:51 PM'),
);
},
),
notice chapter it long but you if you
Will it
iPhone 13 Pro Max – iOS 15.5
4:02
Lottery App
Asif Taj Tech
Subscribe to my channel
1:52:10 / 1:54:57 • List...
Adobe XD