**Flutter**

# Introduction

Flutter is a free framework made by Google. It allows us to create apps for Android, iOS, web, and desktop using only one code. It is based on the Dart language and uses widgets to design beautiful and fast apps. With Flutter, developers save time, write less code, and see changes quickly using hot reload.

In short, Flutter is a modern and powerful tool for building apps easily and efficiently.

## React Native vs Flutter

* **React Native** is a framework created by **Facebook**. It uses **JavaScript** to build apps. It relies on “bridges” to connect with native code, which can make it slower sometimes. But it has a huge community, many ready libraries, and is easy for web developers who know JavaScript.
* **Flutter** is a framework created by **Google**. It uses the **Dart** language and builds apps with its own rendering engine. This makes apps **faster, smoother, and more consistent** across Android and iOS. It also has “hot reload” and many built-in widgets for beautiful designs.

👉 **In short:**

* If you know JavaScript, React Native is easier to start with.
* If you want speed, performance, and great UI, Flutter is the better choice.

## Android Studio

Android Studio is the official IDE (Integrated Development Environment) from **Google** for building Android apps.

**What it Provides**

* A code editor with smart suggestions.
* Android Emulator to test apps.
* Tools for design and layout.
* Debugging and performance analysis.
* Direct publishing to the Play Store.

👉 In short: **Android Studio gives everything needed to build, test, and publish Android apps in one place.**

## SDK (Software Development Kit)

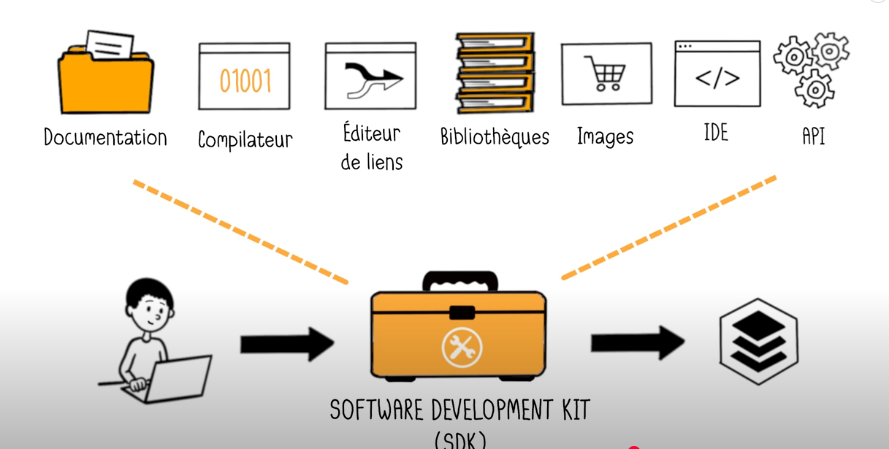
An SDK is a set of tools, libraries, and instructions that developers use to build apps for a specific platform.

* Example: **Android SDK** lets you build Android apps.
* It includes APIs, emulators, and tools to connect your code with the phone system.

**Other Examples**

* **Java SDK (JDK):** build Java programs.
* **Flutter SDK:** build cross-platform apps.
* **.NET SDK:** build Windows or web apps.

👉 In short: **SDKs give developers everything needed to create and run apps on a platform.**



An SDK is like a **toolbox** for developers. It contains everything needed to build applications:

* **Documentation** → guides on how to use it.
* **Compiler** → translates code into machine language.
* **Linker/Editor** → connects different parts of the program.
* **Libraries** → ready-made code to save time.
* **Images & resources** → for app design.
* **IDE** → the environment to write and test code.
* **APIs** → connections to communicate with the system or other apps.

👉 In short: **An SDK brings all tools together so developers can easily create, test, and run apps.**

## Dowland Flutter SDK

**Download Flutter SDK**

* Go to Flutter official site
* Download the **SDK zip** for your OS.

**Extract Flutter**

* Extract the zip to a desired location (e.g., C:\src\flutter on Windows, /home/user/flutter on Linux).

**Add Flutter to PATH**

* **Windows:**
  1. Open Start → Search *Environment Variables* → Edit System Environment Variables
  2. Under *System Variables*, edit **Path**, add the Flutter bin folder path (e.g., C:\src\flutter\bin)

### **Verify Installation**

flutter –version

Should display Flutter version info

**Install Android Studio (for Android development)**

1. Download Android Studio
2. Install **Android SDK**, **Android SDK Platform-Tools**, **Android Emulator**
3. Open Android Studio → SDK Manager → install necessary SDKs

Run Flutter Doctor

Shows missing dependencies and instructions to fix them

**Install VS Code (optional but recommended)**

* Install **Flutter** and **Dart** extensions
* Set Flutter SDK path in VS Code settings

## Big list of Common flutter Commands

***Checking & Environment***

| **Command** | **Purpose** |
| --- | --- |
| flutter doctor | Checks your environment for setup issues and missing dependencies |
| flutter --version | Shows Flutter version |
| flutter upgrade | Updates Flutter SDK to latest version |
| flutter channel | Shows current Flutter channel (stable, beta, dev, master) |
| flutter channel <name> | Switch to another Flutter channel |

***Devices & Emulators***

| **Command** | **Purpose** |
| --- | --- |
| flutter devices | Lists all connected devices (phones, emulators) |
| flutter emulators | Lists all available emulators |
| flutter emulators --launch <emulator\_id> | Launches a specific emulator |
| flutter attach | Connects to a running app on a device for hot reload/debug |

***Project Commands***

| **Command** | **Purpose** |
| --- | --- |
| flutter create <project\_name> | Creates a new Flutter project |
| flutter clean | Deletes build folder and temporary files |
| flutter pub get | Fetches dependencies from pubspec.yaml |
| flutter pub upgrade | Upgrades dependencies to latest compatible versions |
| flutter pub outdated | Shows outdated packages |

***Run & Build***

| **Command** | **Purpose** |
| --- | --- |
| flutter run | Runs the app on connected device/emulator |
| flutter run -d <device\_id> | Runs app on a specific device |
| flutter build apk | Builds an Android APK |
| flutter build appbundle | Builds an Android App Bundle for Play Store |
| flutter build ios | Builds an iOS app (requires macOS + Xcode) |
| flutter build web | Builds a web version of the app |
| flutter build windows/linux/macos | Builds desktop apps |

***Hot Reload & Hot Restart***

| **Command** | **Purpose** |
| --- | --- |
| r in terminal while running | **Hot reload** – updates code without restarting app |
| R in terminal while running | **Hot restart** – restarts app completely |
| q in terminal | Quit running app |

***Analysis & Testing***

| **Command** | **Purpose** |
| --- | --- |
| flutter analyze | Checks code for errors and warnings |
| flutter test | Runs unit and widget tests |
| flutter format . | Formats all Dart files in the project |
| flutter doctor -v | Shows verbose doctor output (detailed info) |

***Other Useful Commands***

| **Command** | **Purpose** |
| --- | --- |
| flutter channel stable | Switch to the stable release channel |
| flutter downgrade | Downgrades Flutter SDK to previous version |
| flutter config --enable-web | Enables web support |
| flutter config --enable-linux-desktop | Enables Linux desktop support |

💡 **Tip:**

Most Flutter commands support -h for help, e.g.:

flutter run -h

flutter build apk -h

It shows all options and flags.

For solving error (flutter doctor) search in yotube for courses4arab.

And now let’s get started :

# Creating Flutter project

Enter vs code :

**Step 1: Open Command Palette**

* Press Ctrl + Shift + P (Windows/Linux) or Cmd + Shift + P (Mac).

**Step 2: Start Flutter Project**

* Type:

Flutter: New Project

* Select it.

**Step 3: Choose Project Type**

* Pick **Flutter Application** (or other types if needed).

**Step 4: Name Your Project**

* Type a project name (e.g., my\_first\_app).

**Step 5: Choose Folder**

* Select the folder where the project will be created.

My device is not strong , me I will not use emulator , I will use either web or my physical phone .

Let’s run it in web :

flutter run -d chrome

also in chrome you can set any phone you want it .

delete all , and keep only :

import 'package:flutter/material.dart';

void main() {}

# widgets

In **Flutter**, **everything is a widget**. Yes, **literally everything** you see on the screen is a widget:

**The entire app its self** is a widget

runApp(MyApp());

MyApp() itself is a **widget**—it’s the **root of your widget tree**.

**The key concept: Widget tree**

* Flutter works with a **tree of widgets**.
* **Root widget** → contains **child widgets** → which contain **their own child widgets** → and so on.

void main() {

  // the next function :

  // “Take my MyApp widget and make it the main app displayed on the device.”

  runApp(MyApp());

}

**Widget naming convention**

* **All widget classes should start with a capital letter.**
* This follows **Dart’s class naming rules** (PascalCase).

MyApp is following PascalCase because it’s a widget.

💡 **Rule of thumb:**

**Classes = Capitalized → Widgets = Capitalized**  
**Variables = lowerCamelCase → Children or instances of widgets = lowerCamelCase**

## Widgets Types

**1️⃣ StatelessWidget**

* **Cannot change** after being built.
* **Static content**.
* Example: Text, Icon, Image

**2️⃣ StatefulWidget**

* **Can change** over time.
* **Dynamic content / interactive UI**.
* Example: Counter button, form, animation

**Simple rule:**

**Stateless = fixed**  
**Stateful = can change**

import 'package:flutter/material.dart';

void main() {

  // the next function :

  // “Take my MyApp widget and make it the main app displayed on the device.”

  runApp(MyApp());

}

class MyApp extends StatelessWidget {

  @override

  Widget build(BuildContext context) {

    return MaterialApp(home: Text('hello bilal  '));

  }

}



class MyApp extends StatelessWidget {

  // override is so imporantant + build method must reurn MaterialApp

  @override

  Widget build(BuildContext context) {

    return MaterialApp(home: Text('hello bilal  '));

  }

}

**1️⃣ Creating a StatelessWidget**

* To create a **stateless widget**, you **extend** the StatelessWidget class.

class MyApp extends StatelessWidget

* This means your widget **cannot change its state** after being built.

**2️⃣ Override the build function**

* Every stateless widget must **override the build method**.
* build **returns the UI** of the widget.

@override

Widget build(BuildContext context)

Here, you **return a MaterialApp** as the root of your app.

**3️⃣ MaterialApp**

* MaterialApp is **important** because it provides:
  + App structure
  + Themes
  + Navigation
  + Other Material Design features

**Note :**

Don’t focus soooo much here , we will study them well later .

**Rapel :**

**What extends does**

* extends is a **Dart keyword** used for **inheritance**.
* It means:

“My class is a type of another class, and it **inherits all its properties and methods**.”

# Text Widget

💡 **Rule of thumb:**

In Flutter, **every property inside a widget uses propertyName: value**, so Dart knows **what goes where**.

**1️⃣ Text Widget**

* **Purpose:** Display text on the screen.
* **Syntax:**

Text("Hello world")

'Hello Flutter' → the string to display.

**Optional parameters:**

* style: → apply a TextStyle
* textAlign: → align text (left, center, right)
* overflow: → handle text that is too long

**2️⃣ TextStyle Widget**

* **Purpose:** Define **how the text looks**.
* **Parameters:**
  + color → text color
  + fontSize → size of text
  + fontWeight → thickness (bold, normal, etc.)
  + fontStyle → italic or normal
  + letterSpacing → space between letters
  + decoration → underline, line-through, etc.

**3️⃣ How it works together**

* **Text widget** → “What to show”
* **TextStyle widget** → “How it looks”

**Analogy:**

* Text is like a **signboard** with words
* TextStyle is like the **paint, font, color, and style** of the letters

  home: Scaffold(

        appBar: AppBar(title: Text("Hello world")),

        body: Text(

          "Hello from body",

          style: TextStyle(

            fontSize: 30,

            color: Colors.blue,

            fontWeight: FontWeight.bold,

            fontStyle: FontStyle.italic,

            decoration: TextDecoration.underline,

          ),

        ),

      ),

For showing colors there is many methods

| **Method** | **Example** | **Notes** |
| --- | --- | --- |
| Predefined | Colors.blue | Simple, fast |
| Hex / ARGB | Color(0xFF42A5F5) | Custom color using hex |
| fromARGB | Color.fromARGB(255, 66, 165, 245) | Specify alpha + RGB |
| fromRGBO | Color.fromRGBO(66, 165, 245, 1.0) | RGB + opacity as 0.0–1.0 |

💡 **Tip:**

* For most apps, **Colors.blue and hex codes** are enough.
* Use fromRGBO if you want **precise opacity control**.

For hex , always write 0xff and then color code .

# Container widget

**1️⃣ What is a Container?**

* A **box** that can hold one child widget.
* It’s used for **styling, positioning, sizing, and decorating** widgets.

👉 Think of it like a **frame**:

* You can change its **size, color, border, margin, padding, alignment, etc.**

class MyApp extends StatelessWidget {

  @override

  Widget build(BuildContext context) {

    return MaterialApp(

      home: Scaffold(

        appBar: AppBar(title: Text("hello")),

        body: Container(

          width: 400,

          height: 400,

          color: Colors.black26,

          padding: EdgeInsets.all(10),

          margin: EdgeInsets.all(10),

          alignment: Alignment.center,

          child: const Text("This a test Text"),

        ),

      ),

    );

  }

}

**Important :**

**Rule of Thumb**

* **If only color** → use color: directly (simpler).
* **If decoration is used** → put color: **inside BoxDecoration**.

If you don’t respect this , an error will be raised .

  home: Scaffold(

        appBar: AppBar(title: Text("hello")),

        body: Container(

          width: 400,

          height: 400,

          decoration: BoxDecoration(

            // Soft background color (light blue-grey)

            color: const Color.fromARGB(255, 200, 220, 240),

            // Border with a nice color

            border: Border.all(color: Colors.blueAccent, width: 3),

            // Smooth rounded corners

            borderRadius: BorderRadius.circular(25),

            // Soft shadow for depth

            boxShadow: [

              BoxShadow(

                color: Colors.black.withValues(alpha: 0.2), // lighter shadow

                blurRadius: 15, // smoother edges

                spreadRadius: 3, // bigger shadow

                offset: const Offset(5, 5), // direction (x,y)

              ),

            ],

          ),

          padding: EdgeInsets.all(10),

          margin: EdgeInsets.all(10),

          alignment: Alignment.center,

          child: const Text("This a test Text"),

        ),

      ),

Now we will study well , let’s go

class MyApp extends StatelessWidget {

  @override

  Widget build(BuildContext context) {

    return MaterialApp(

      home: Scaffold(

        appBar: AppBar(

          title: Text("Hello World"),

          titleTextStyle: TextStyle(color: Colors.amber, fontSize: 30.5),

        ),

        body: Container(

          width: 200,

          height: 200,

          padding: EdgeInsets.all(20),

          margin: EdgeInsets.all(50),

          decoration: BoxDecoration(

            color: Colors.blueGrey,

            border: Border.all(

              color: const Color.fromARGB(255, 27, 27, 25),

              width: 5,

              style: BorderStyle.solid,

            ),

            borderRadius: BorderRadius.all(Radius.circular(20)),

            boxShadow: [

              BoxShadow(

                color: const Color.fromARGB(255, 27, 27, 25),

                offset: const Offset(7, 8),

                blurRadius: 10,

                spreadRadius: 1,

              ),

            ],

          ),

          child: Text(

            "hello",

            style: TextStyle(color: Colors.white, fontSize: 20),

            textAlign: TextAlign.center,

          ),

        ),

      ),

    );

  }

}

## Width

* **Takes:** **double?**
* Meaning: fixed horizontal size in logical pixels.
* If null → container size depends on child or constraints.

## Height

* **Takes:** double?
* Meaning: fixed vertical size in logical pixels.
* If null → container adapts to child or constraints.

## Alignment

**Takes:** AlignmentGeometry

**Common options:**

* Alignment.center → child in center.
* Alignment.topLeft, topCenter, topRight → child at top.
* Alignment.bottomLeft, bottomCenter, bottomRight → child at bottom.
* Alignment.centerLeft, centerRight → child at left or right.
* Custom: Alignment(x, y) where x = -1 (left) to 1 (right), y = -1 (top) to 1 (bottom).

## margin

**Takes:** EdgeInsets (same options as padding)

**Options:**

* EdgeInsets.all(value) → same outside space on all sides.
* EdgeInsets.symmetric(horizontal: x, vertical: y) → same space left/right, top/bottom.
* EdgeInsets.only(left: x, top: y, right: z, bottom: w) → different space each side.
* EdgeInsets.fromLTRB(left, top, right, bottom) → same as .only but with 4 numbers.

## padding

 Takes: EdgeInsets

 Options:

* EdgeInsets.all(value) → equal inner space on all sides.
* EdgeInsets.symmetric(horizontal: x, vertical: y) → inner space by axis.
* EdgeInsets.only(left: x, top: y, right: z, bottom: w) → inner space per side.
* EdgeInsets.fromLTRB(left, top, right, bottom) → same as only with 4 numbers.

## color

* Takes: Color?
* Options:
  + Predefined colors: Colors.red, Colors.blue, Colors.green, etc.
  + Shades: Colors.blue[200] (lighter), Colors.blue[900] (darker).
  + Custom ARGB: Color.fromARGB(a, r, g, b) → a=alpha, r=red, g=green, b=blue.
  + Custom HEX: Color(0xFF123456) → FF = opacity, then hex RGB.
* ⚠️ Cannot be used together with decoration.

## child

 Takes: **Widget?**

 Options: any widget (Text, Row, Column, Image, etc.).

 A container can only have **one child**.

## BoxDecoration Parameters

### color

* Takes: Color?
* Options:
  + Predefined colors → Colors.red, Colors.blue, Colors.green, …
  + Shades → Colors.blue[100], Colors.blue[900]
  + Custom ARGB → Color.fromARGB(a, r, g, b)
  + Custom HEX → Color(0xFF123456)
* Note: you cannot use both Container.color and BoxDecoration.color at the same time.

### border

* Takes: BoxBorder? (usually Border)
* Options of Border:
  + Border.all(color, width, style) → same border all around.
  + Border.symmetric(horizontal: x, vertical: y) → different for sides.
  + Border(top: BorderSide(...), bottom: BorderSide(...), …) → individual sides.
* Options of BorderSide:
  + color → border color.
  + width → thickness.
  + style → BorderStyle.solid or BorderStyle.none.

### borderRadius

* Takes: BorderRadiusGeometry?
* Options:
  + BorderRadius.circular(value) → all corners rounded equally.
  + BorderRadius.only(topLeft: Radius.circular(x), …) → round specific corners.
  + BorderRadius.all(Radius.circular(value)) → same as circular.
  + BorderRadius.vertical(top: Radius.circular(x), bottom: Radius.circular(y)) → only vertical corners.
  + BorderRadius.horizontal(left: Radius.circular(x), right: Radius.circular(y)) → only horizontal corners.

### boxShadow

* Takes: List<BoxShadow>?
* Options of BoxShadow:
  + color → shadow color.
  + offset → how far the shadow is moved (x, y).
  + blurRadius → how blurry the shadow is.
  + spreadRadius → how much shadow spreads.

You can make 2 box shadows, see this :

body: Container(

          width: 150,

          height: 150,

          decoration: BoxDecoration(

            color: Colors.blueAccent,

            shape: BoxShape.circle, // make it a circle

            boxShadow: [

              BoxShadow(

                color: Colors.blueAccent.withOpacity(0.6),

                offset: Offset(4, 4),

                blurRadius: 10,

                spreadRadius: 1,

              ),

              BoxShadow(

                color: Colors.white.withOpacity(0.8),

                offset: Offset(-4, -4),

                blurRadius: 10,

                spreadRadius: 1,

              ),

            ],

          ),

          alignment: Alignment.center,

          child: Text(

            "Flutter",

            style: TextStyle(

              color: Colors.white,

              fontSize: 20,

              fontWeight: FontWeight.bold,

            ),

          ),

        ),

      ),



Result is great and modern .

# Image

The Image widget in Flutter is used to display images , from your local pc or on internet .

To display an image from the internet .

Use **Image.network** , from local use **Image.assets**

## Image.network

Let’s test it :

Image.network(

            "https://fastly.picsum.photos/id/208/536/354.jpg?hmac=kC5LCsb06ym3nGKXTBNRgIil9aUppMjlmp6GYVK1agc",

          ),

## Image.asset

This is how it works

Container(child: Image.asset('../images/img.png')),

**How to modify pubspec.yaml to include your images**

1. **Open pubspec.yaml**
   * It’s in the **root of your Flutter project** (same level as lib and images).
2. **Find the flutter: section**
   * There’s usually a line starting with flutter: in the file.
3. **Locate the assets: line**
   * It might be **commented out** (#).
   * Uncomment it by removing #.
4. **Add your folder or image**
   * Since your folder is images in the root, write:

flutter:

assets:

- images/

* The **trailing slash** / tells Flutter to include **all images inside this folder**.

1. **Save the file**
2. **Run flutter pub get**
   * This registers the assets with Flutter so they are included in the build.

**Why it’s important**

* In **debug mode**, some IDEs may temporarily show images even if you don’t declare them in pubspec.yaml.
* **But in release builds**, the images **won’t be bundled** unless they’re listed in pubspec.yaml.
* This means your app could **crash or show blank images** on other devices or after publishing.

✅ Always declare assets in pubspec.yaml to make your app **work reliably on all devices**.

assets:

     - images/img.png

if you add a **folder** in pubspec.yaml, **all images inside that folder** are included automatically.

Example:

flutter:

assets:

- images/

* The **trailing slash /** is important.
* Any file you put inside images/ (PNG, JPG, GIF, etc.) will now be available with Image.asset('images/your\_image.png').
* You **don’t need to list each image individually**.

⚠️ But if you add new images **after running the app**, you should **run flutter pub get again** so Flutter recognizes them.

## Image widget parameters

### width

* **Type:** double?
* **What it does:** Sets the horizontal size of the image in logical pixels.
* **Options / Usage:**
  + Any number: width: 100 → image is 100 pixels wide.
  + If null → image takes natural width or is constrained by parent widget.

### height

* **Type:** double?
* **What it does:** Sets the vertical size of the image in logical pixels.
* **Options / Usage:**
  + Any number: height: 150 → image is 150 pixels tall.
  + If null → image takes natural height or is constrained by parent widget.

child: Image.asset('../images/img.png', width: 500, height: 500)

Here we can see a problem the img that not fit the height, but it’s only space



For solving this:

### fit

* **Type:** BoxFit?
* **What it does:** Determines **how the image fits into the given width/height box**.
* **Options:**
  + BoxFit.fill → stretches image to fill the box (may distort).
  + BoxFit.contain → scales image to fit inside box, keeps aspect ratio.
  + BoxFit.cover → fills the box completely, may crop.
  + BoxFit.fitWidth → fits image width, height may overflow.
  + BoxFit.fitHeight → fits image height, width may overflow.
  + BoxFit.none → no scaling, keeps original size.
  + BoxFit.scaleDown → scales down only if image is bigger than box.







**NOTE :**

**Aspect Ratio**

* **Definition:** The **ratio of the image’s width to its height**.
* **Formula:**

Aspect Ratio=WidthHeight\text{Aspect Ratio} = \frac{\text{Width}}{\text{Height}}Aspect Ratio=HeightWidth​

* Example:
  + Image width = 200, height = 100 → aspect ratio = 200 ÷ 100 = 2.0
  + This means the image is **twice as wide as it is tall**.

# Column & Row

## Column

### What does Column do ?

* It puts widgets vertically, one under the other.
* Think of it like a vertical stack .

### Most important properties :

 **children** → list of widgets inside the column.

 **mainAxisAlignment** → controls **vertical** alignment of children.

* Example: start, center, end, spaceBetween.

 **crossAxisAlignment** → controls **horizontal** alignment of children.

* Example: start (left), center, end (right).

 **mainAxisSize** → how much space the column takes vertically.

* max → full screen height.
* min → just fit its children.

Example :

  home: Scaffold(

        appBar: AppBar(title: Text("hello")),

        body: Column(

          children: [

            Text("Hello"),

            Image.asset("../images/img.png"),

            Container(width: 200, height: 2000, color: Colors.amber),

          ],

        ),

      ),

An other example with properties :

  body: Center(

          child: Column(

            mainAxisAlignment: MainAxisAlignment.center, // center vertically

            crossAxisAlignment:

                CrossAxisAlignment.center, // center horizontally

            children: [

              // Title

              Text(

                "App Development Service",

                style: TextStyle(

                  fontSize: 26,

                  fontWeight: FontWeight.bold,

                  color: Color.fromARGB(255, 33, 150, 243), // modern blue color

                ),

              ),

              SizedBox(height: 20), // spacing

              // Image

              Image.asset(

                "../images/img.png",

                width: 200,

                height: 200,

                fit: BoxFit.cover, // makes it look nicer

              ),

              SizedBox(height: 30), // spacing

              // Button-like container

              Container(

                padding: EdgeInsets.symmetric(vertical: 10, horizontal: 30),

                decoration: BoxDecoration(

                  color: Color.fromARGB(255, 33, 150, 243),

                  borderRadius: BorderRadius.circular(12), // rounded corners

                ),

                child: Text(

                  "Demand",

                  style: TextStyle(

                    color: Colors.white,

                    fontSize: 22,

                    fontWeight: FontWeight.w600,

                  ),

                  textAlign: TextAlign.center,

                ),

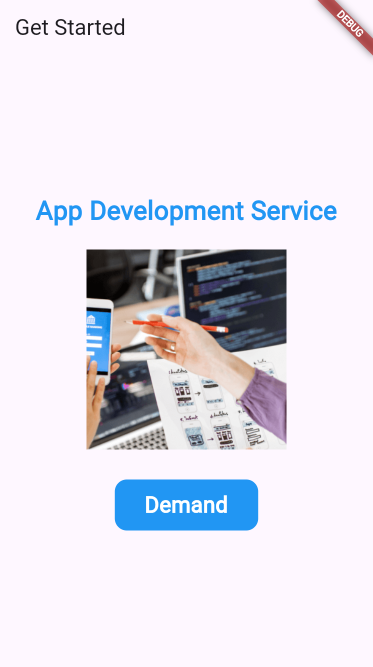
              ),

            ],

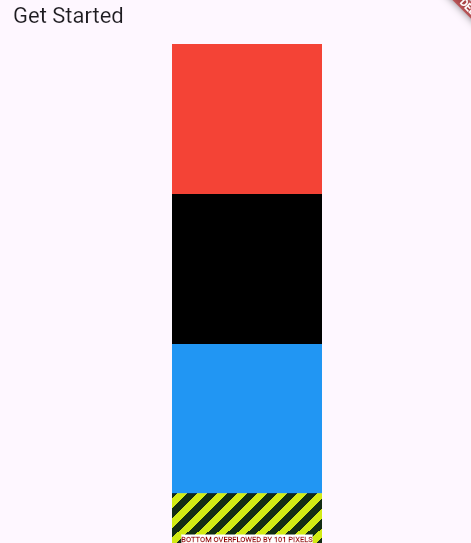
          ),

        ),

And this is how it looks like in iphone SE :



Let’s more exercice and take more imporatant note :



That yellow/black striped area you see in Flutter is called the "overflow warning".  
It happens when a widget takes more space than the screen has (too wide or too tall).

**🔹 Why it happens**

* Column (and Row) try to give their children as much space as they want.
* If a child widget is too big → it **overflows** outside the screen.
* Flutter shows the yellow/black stripes to warn you.

**🔹 Common situations**

1. **Text too long** → goes outside the screen width.
2. **Column with too many children** → goes beyond screen height.
3. **Image too large** → bigger than screen size.

**🔹 How to fix it**

Here are the most used solutions:

1. **Wrap content in SingleChildScrollView** → makes the page scrollable.
2. **Use Expanded or Flexible** inside Column/Row  
   → forces children to resize and fit.
3. **Limit size** (using SizedBox, Container with width/height).

Scaffold(

        appBar: AppBar(title: Text("Get Started")),

        body: Center(

          child: SingleChildScrollView(

            child: Column(

              children: [

                Container(width: 150, height: 150, color: Colors.red),

                Container(width: 150, height: 150, color: Colors.black),

                Container(width: 150, height: 150, color: Colors.blue),

                Container(width: 150, height: 150, color: Colors.green),

                Container(width: 150, height: 150, color: Colors.yellow),

                Container(width: 150, height: 150, color: Colors.orange),

              ],

            ),

          ),

        ),

      ),

## MainAxisAlignment

Helps us to vertical alignment for childern

### MainAxisAlignment.start (default)

**Effect:**

* All containers stick to the **top** of the available space.
* Spacing between them is just their height, no extra gap.

Center(

          child: Column(

            mainAxisAlignment: MainAxisAlignment.start, // align top

            children: [

              Container(width: 150, height: 150, color: Colors.red),

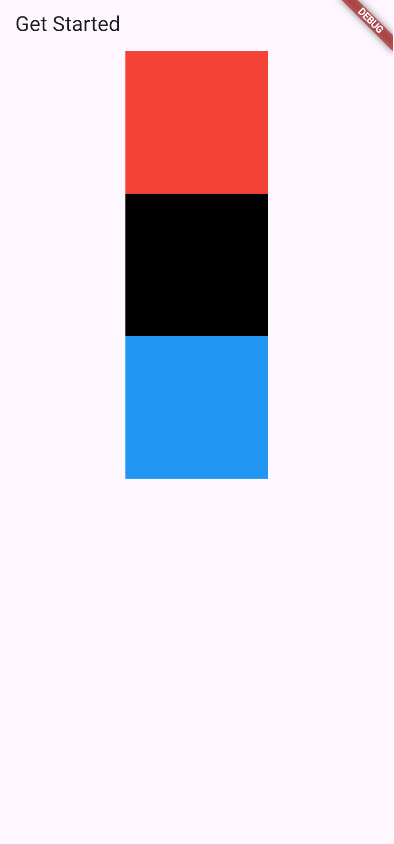
              Container(width: 150, height: 150, color: Colors.black),

              Container(width: 150, height: 150, color: Colors.blue),

            ],

          ),

        )



### MainAxisAlignment.end

Effect:

* All containers stick to the bottom of the available space.

Center(

          child: Column(

            mainAxisAlignment: MainAxisAlignment.end, // align top

            children: [

              Container(width: 150, height: 150, color: Colors.red),

              Container(width: 150, height: 150, color: Colors.black),

              Container(width: 150, height: 150, color: Colors.blue),

            ],

          ),

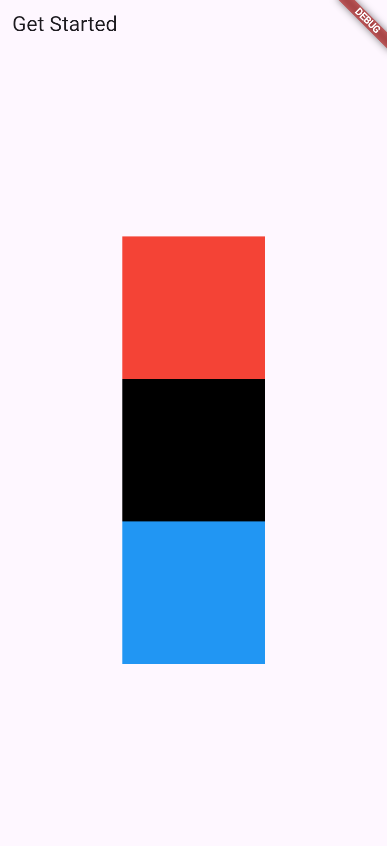
        )



### MainAxisAlignment.center

**Effect:**

* All containers are centered vertically in the screen.



### MainAxisAlignment.spaceBetween

**Effect:**

* First container goes to the **top**, last container goes to the **bottom**.
* Middle container gets **equal space** in between.

Center(

          child: Column(

            mainAxisAlignment: MainAxisAlignment.spaceBetween,

            children: [

              Container(width: 150, height: 150, color: Colors.red),

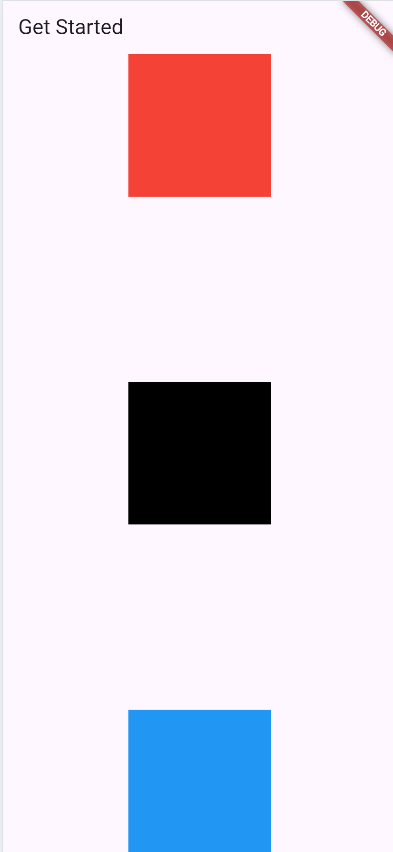
              Container(width: 150, height: 150, color: Colors.black),

              Container(width: 150, height: 150, color: Colors.blue),

            ],

          ),

        )



### MainAxisAlignment.spaceAround

**Effect:**

* Equal space **around** each container.
* Top and bottom have **half the space** compared to between containers.

Center(

          child: Column(

            mainAxisAlignment: MainAxisAlignment.spaceAround,

            children: [

              Container(width: 150, height: 150, color: Colors.red),

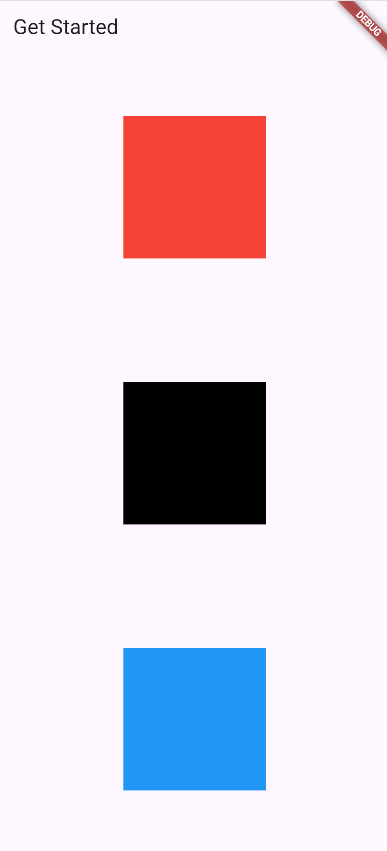
              Container(width: 150, height: 150, color: Colors.black),

              Container(width: 150, height: 150, color: Colors.blue),

            ],

          ),

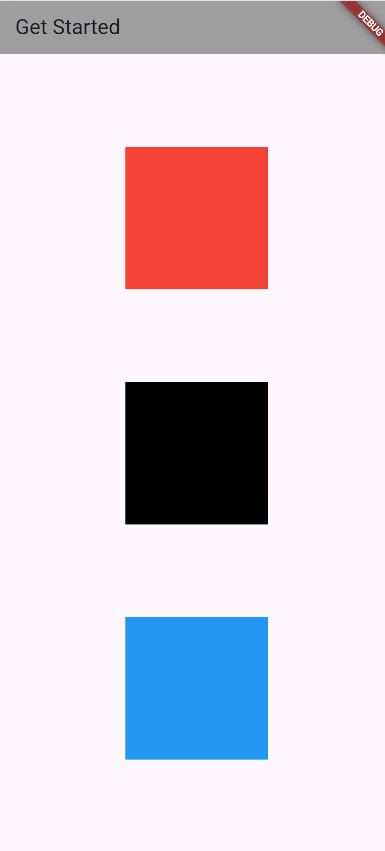
        )



### MainAxisAlignment.spaceEvenly

**Effect:**

* Equal space **before, between, and after** all containers.
* Very clean and symmetric layout.



✅ Summary of differences:

|  |  |
| --- | --- |
| Option | Effect |
| start | Containers at top |
| end | Containers at bottom |
| center | Containers centered vertically |
| spaceBetween | Equal space between, top/bottom touch edges |
| spaceAround | Equal space around each, top/bottom half-space |
| spaceEvenly | Equal space everywhere, fully balanced |

## CrossAxisAlignment

### CrossAxisAlignment.start

Effect:

* All containers are aligned to the left.
* Each container keeps its width.

Column(

            crossAxisAlignment: CrossAxisAlignment.start, // align left

            children: [

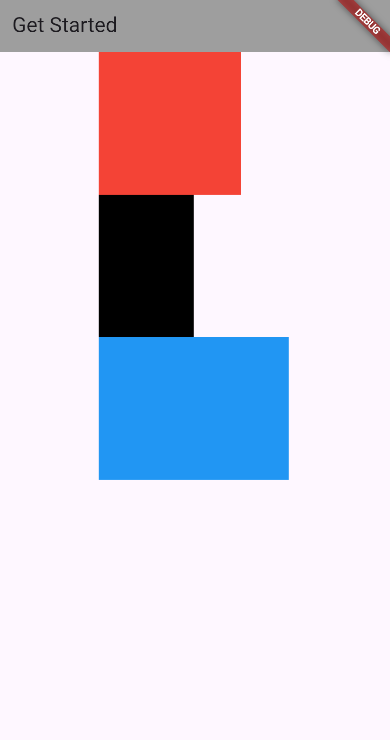
              Container(width: 150, height: 150, color: Colors.red),

              Container(width: 100, height: 150, color: Colors.black),

              Container(width: 200, height: 150, color: Colors.blue),

            ],

          )



### CrossAxisAlignment.end

Effect:

* All containers are aligned to the right.
* Different widths create a jagged right edge.

Column(

            crossAxisAlignment: CrossAxisAlignment.end, // align right

            children: [

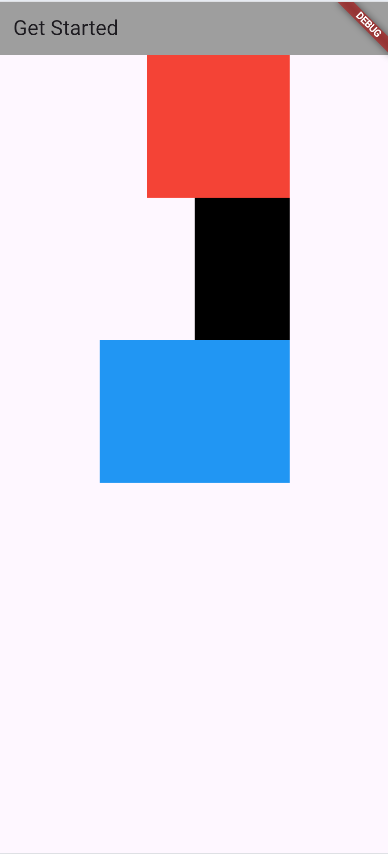
              Container(width: 150, height: 150, color: Colors.red),

              Container(width: 100, height: 150, color: Colors.black),

              Container(width: 200, height: 150, color: Colors.blue),

            ],

          )



### CrossAxisAlignment.center

Effect:

* Containers are centered horizontally.
* Looks balanced even if widths are different.

Column(

            crossAxisAlignment:

                CrossAxisAlignment.center, // align center horizontally

            children: [

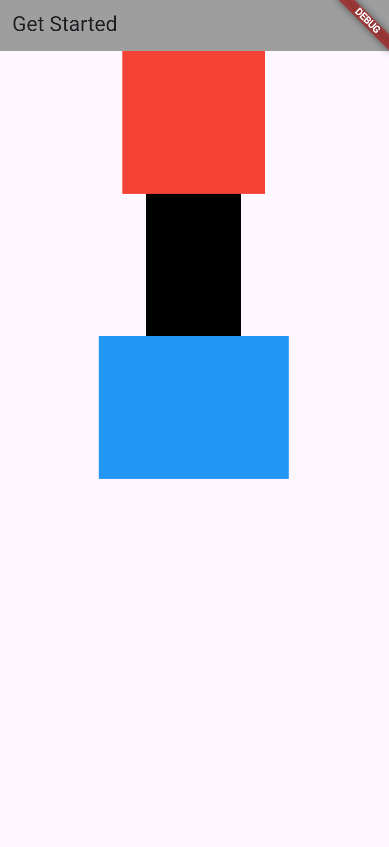
              Container(width: 150, height: 150, color: Colors.red),

              Container(width: 100, height: 150, color: Colors.black),

              Container(width: 200, height: 150, color: Colors.blue),

            ],

          )



### CrossAxisAlignment.stretch

Effect:

* Containers stretch horizontally to fill the available width.
* You don’t need to set width, they take all space.

Column(

            crossAxisAlignment:

                CrossAxisAlignment.stretch, // fill horizontal space

            children: [

              Container(height: 150, color: Colors.red),

              Container(height: 150, color: Colors.black),

              Container(height: 150, color: Colors.blue),

            ],

          )



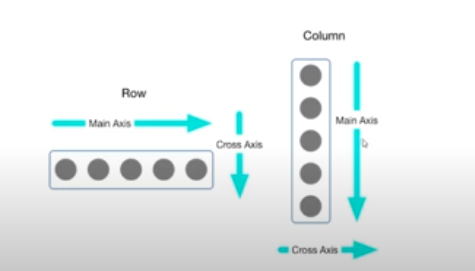
**Summary Table**

|  |  |
| --- | --- |
| Option | Horizontal alignment |
| start | Left |
| end | Right |
| center | Center |
| stretch | Fill available width |

# Row

⚡ Important:

* In a Row, the main axis is horizontal → mainAxisAlignment controls left–right positioning.
* The cross axis is vertical → crossAxisAlignment controls top–bottom positioning.



📌 What is a Row in Flutter?

* A Row is a widget that arranges its children horizontally (from left to right).
* It’s the opposite of Column (which arranges things vertically).
* Example use: placing buttons side by side, icons next to text, etc.
* 👉 Think of it like putting things next to each other in a line.

Scaffold(

        appBar: AppBar(title: Text("Profile Card")),

        body: Center(

          child: Container(

            padding: EdgeInsets.all(16),

            decoration: BoxDecoration(

              color: Colors.blue[50],

              borderRadius: BorderRadius.circular(12),

            ),

            child: Row(

              crossAxisAlignment: CrossAxisAlignment.start,

              children: [

                // Profile Image

                ClipOval(

                  child: Image.asset(

                    "../images/img.jpg",

                    width: 80,

                    height: 80,

                    fit: BoxFit.cover, // makes sure the image fills the circle

                  ),

                ),

                SizedBox(width: 16), // space between image and text

                // Profile Details (Column inside Row)

                Column(

                  crossAxisAlignment: CrossAxisAlignment.start,

                  children: [

                    Text(

                      "Bilal Elemrani",

                      style: TextStyle(

                        fontSize: 20,

                        fontWeight: FontWeight.bold,

                      ),

                    ),

                    SizedBox(height: 6),

                    Text(

                      "App Developer",

                      style: TextStyle(fontSize: 16, color: Colors.grey[700]),

                    ),

                    SizedBox(height: 6),

                    Text(

                      "bilal@example.com",

                      style: TextStyle(fontSize: 14, color: Colors.grey[600]),

                    ),

                  ],

                ),

              ],

            ),

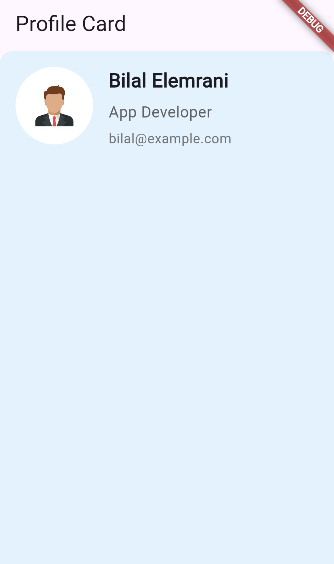
          ),

        ),

      ),

It easy just focus on the code, you will find it easy ,

Note copy & paste it in vs code to see more info when hovering.



# SingleChildScrollView

* Makes one child scrollable (often a Column or Row).
* Good for small/medium layouts, not optimized for big lists.

SingleChildScrollView(

          child: Column(

            children: [

              Container(height: 200, color: Colors.red),

              Container(height: 200, color: Colors.blue),

              Container(height: 200, color: Colors.green),

            ],

          ),

        ),

# Wrap

* Unlike Row/Column, Wrap automatically moves items to the next line if they don’t fit.
* Useful for tags, chips, buttons, etc.

Wrap(

          spacing: 10, // horizontal space

          runSpacing: 10, // vertical space

          children: [

            Container(width: 100, height: 50, color: Colors.red),

            Container(width: 100, height: 50, color: Colors.blue),

            Container(width: 100, height: 50, color: Colors.green),

            Container(width: 100, height: 50, color: Colors.orange),

            Container(width: 100, height: 50, color: Colors.purple),

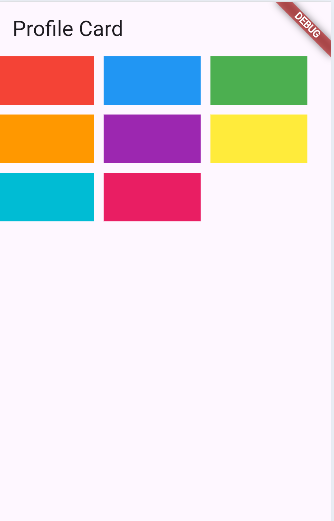
            Container(width: 100, height: 50, color: Colors.yellow),

            Container(width: 100, height: 50, color: Colors.cyan),

            Container(width: 100, height: 50, color: Colors.pink),

          ],

        ),



# ListView

## What is a ListView?

* A scrollable list of widgets arranged vertically by default.
* Can handle many items efficiently (unlike SingleChildScrollView).
* You can customize it in different ways (static items, dynamic, horizontal scrolling)

ListView(

          children: [

            Container(height: 100, color: Colors.red),

            Container(height: 100, color: Colors.blue),

            Container(height: 100, color: Colors.green),

            Container(height: 100, color: Colors.yellow),

            Container(height: 100, color: Colors.orange),

          ],

        ),

*  Each child is a widget (Container, Text, Row…).
*  Scrolls vertically automatically.

And also you can make it horizontaly :

  body: ListView(

          scrollDirection: Axis.horizontal,

          children: [

            Container(width: 100, color: Colors.red),

            Container(width: 100, color: Colors.blue),

            Container(width: 100, color: Colors.green),

            Container(width: 100, color: Colors.yellow),

            Container(width: 100, color: Colors.orange),

            Container(width: 100, color: Colors.purple),

          ],

        )

So so easy

## ListViewBuilder

void main() {

  runApp(MyApp());

}

class MyApp extends StatelessWidget {

  List students = [

    {"firstName": "Billa", "lastName": "El Emrani", "age": 19},

    {"firstName": "John", "lastName": "Doe", "age": 20},

    {"firstName": "Jane", "lastName": "Smith", "age": 22},

    {"firstName": "Alice", "lastName": "Johnson", "age": 21},

    {"firstName": "Bob", "lastName": "Brown", "age": 23},

    {"firstName": "Charlie", "lastName": "Davis", "age": 24},

    {"firstName": "Eve", "lastName": "Wilson", "age": 20},

    {"firstName": "Frank", "lastName": "Garcia", "age": 22},

    {"firstName": "Grace", "lastName": "Martinez", "age": 21},

    {"firstName": "Hank", "lastName": "Rodriguez", "age": 23},

  ];

  @override

  Widget build(BuildContext context) {

    return MaterialApp(

      home: Scaffold(

        appBar: AppBar(title: Text("Contact List")),

        body: ListView.builder(

          itemCount: students.length,

          itemBuilder: (context, i) {

            return Container(

              height: 100,

              padding: EdgeInsets.all(10),

              margin: EdgeInsets.all(10),

              decoration: BoxDecoration(

                border: Border.all(color: Colors.black),

              ),

              child: Row(

                children: [

                  // make a circle contain first letter of first name

                  CircleAvatar(child: Text(students[i]['firstName'][0])),

                  SizedBox(width: 30),

                  Container(

                    child: Column(

                      mainAxisAlignment: MainAxisAlignment.center,

                      children: [

                        Text(

                          "${students[i]['firstName']} ${students[i]['lastName']}",

                          style: TextStyle(

                            fontSize: 18,

                            fontWeight: FontWeight.bold,

                          ),

                        ),

                        Text(

                          "Age: ${students[i]['age']}",

                          style: TextStyle(fontSize: 16),

                        ),

                      ],

                    ),

                  ),

                ],

              ),

            );

          },

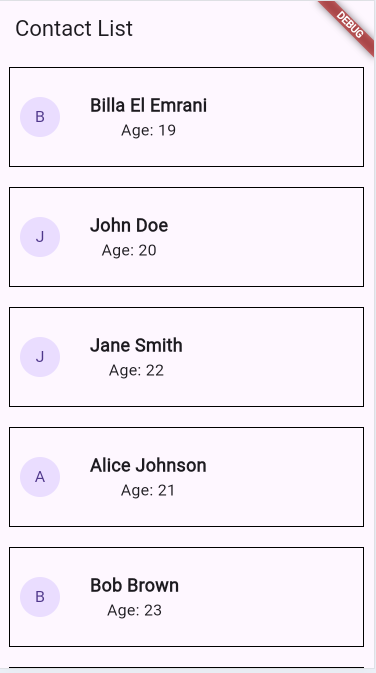
        ),

      ),

    );

  }

}



**1. What is ListView.builder?**

* ListView is a **scrollable list of widgets**.
* ListView.builder is a **smarter version** → it **creates items only when needed** (lazy loading).
* Useful for **long lists** (like 1,000 contacts).

ListView.builder(

          itemCount: X, // how many items

          itemBuilder: (context, i) {

            // builds each item

            return YourWidget; // what to show

          },

## ListView.seperated

### what is ListView.separated?

* Like ListView.builder, it **builds items lazily**.
* But it also lets you **insert a separator widget** between items automatically.
* Perfect for **lists with dividers, lines, or spacing**.

### Examples of separators

* **Line divider:** Divider(color: Colors.grey, thickness: 1)
* **Space between items:** SizedBox(height: 10)
* **Custom widget:** Container(height: 5, color: Colors.red)

### Analogy

* **ListView.builder** → like a list of items glued together.
* **ListView.separated** → like a list of items **with space or lines between each item**.

ListView.separated(

          itemBuilder: (context, index) {

            return Container(

              padding: EdgeInsets.all(10),

              child: Row(

                mainAxisAlignment: MainAxisAlignment.spaceBetween,

                children: [

                  Text(

                    "${students[index]['firstName']} ${students[index]['lastName']}",

                  ),

                  Text("${students[index]['age']} years"),

                ],

              ),

            );

          },

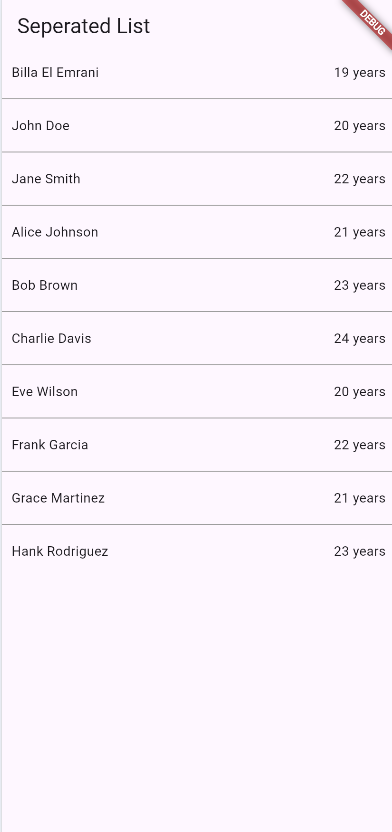
          separatorBuilder: (context, index) {

            return Divider(color: Colors.grey);

          },

          itemCount: students.length,

        ),



## Divider Widget

* **Purpose:** Draws a **horizontal line** to separate widgets in a list or column.
* **Commonly used in:** ListView.separated, Column, or Row.

### Common Properties

|  |  |
| --- | --- |
| Property | What it does |
| color | Color of the line (e.g., Colors.red) |
| thickness | Height of the line in pixels |
| indent | Space before the line (from left) |
| endIndent | Space after the line (from right) |
| height | Vertical space taken by the divider |

separatorBuilder: (context, index) {

            return Divider(

              color: Colors.blue,

              thickness: 2,

              indent: 20,

              endIndent: 20,

            );

          },



# GridView

## What is GridView

A **GridView** is like a **ListView**, but instead of one item per row, it arranges widgets in a **grid (rows & columns)**

Think of a **gallery app** where you see pictures in squares.

## Properites of GridView

Here are the most useful ones (you don’t need to remember all at once):

* **children** → the list of widgets to display (if you use GridView.count or GridView.extent).
* **crossAxisCount** → number of **columns**.
* **mainAxisSpacing** → vertical spacing between rows.
* **crossAxisSpacing** → horizontal spacing between columns.
* **padding** → space around the grid.
* **childAspectRatio** → width/height ratio of each item (default is square = 1).
* **GridView.builder** → loads items **on demand** (better for large lists).

## How to use it

You can create a grid in two main ways:

**✅** Quick way (GridView.count)

Good when you have a **small, fixed number** of items.

Scaffold(

        appBar: AppBar(title: Text('My Flutter App')),

        body: GridView.count(

          crossAxisCount: 3,

          crossAxisSpacing: 10,

          mainAxisSpacing: 10,

          padding: EdgeInsets.all(10),

          children: [

            Card(child: Center(child: Text('Item 1'))),

            Card(child: Center(child: Text('Item 2'))),

            Card(child: Center(child: Text('Item 3'))),

            Card(child: Center(child: Text('Item 4'))),

            Card(child: Center(child: Text('Item 5'))),

            Card(child: Center(child: Text('Item 6'))),

            Card(child: Center(child: Text('Item 7'))),

            Card(child: Center(child: Text('Item 8'))),

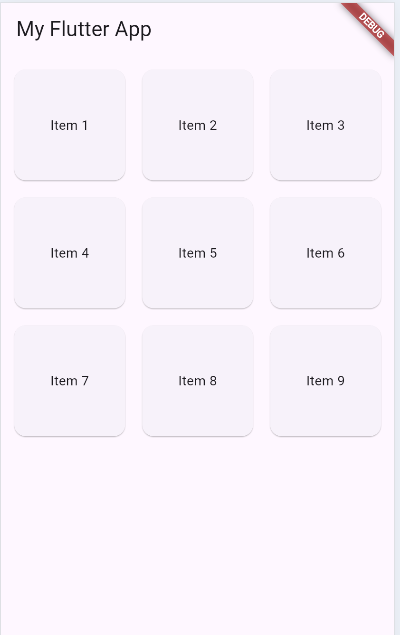
            Card(child: Center(child: Text('Item 9'))),

          ],

        ),

      ),

dd



GridView.count(

          crossAxisCount: 2, // 2 columns

          crossAxisSpacing: 10,

          mainAxisSpacing: 10,

          padding: EdgeInsets.all(10),

          childAspectRatio: 1, // square items

          children: [

            Icon(Icons.home, size: 50, color: Colors.blue),

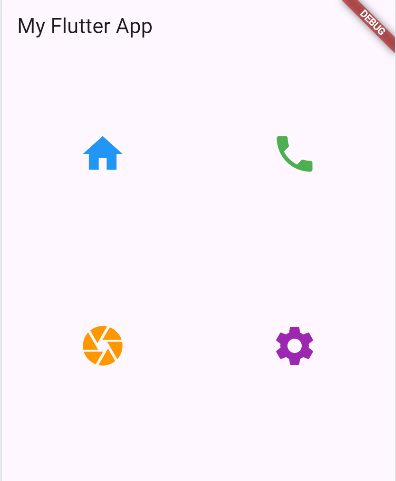
            Icon(Icons.phone, size: 50, color: Colors.green),

            Icon(Icons.camera, size: 50, color: Colors.orange),

            Icon(Icons.settings, size: 50, color: Colors.purple),

          ],

        ),



✅ Efficient way (GridView.builder)

**In this way , you have to add gridDelegate that will contain all control properties**

When you use **GridView.builder**, you can’t directly pass crossAxisCount, spacing, etc.  
Instead, you **must** give it a **gridDelegate**, which controls **how the grid looks** (columns, spacing, ratio, etc.).

#### gridDelegate Options

There are mainly **two** types you’ll use:

1. ***SliverGridDelegateWithFixedCrossAxisCount***
   * You decide the **number of columns**.
   * Flutter calculates item sizes automatically.

Scaffold(

        appBar: AppBar(title: Text('My Flutter App')),

        body: GridView.builder(

          gridDelegate: SliverGridDelegateWithFixedCrossAxisCount(

            crossAxisCount: 3,

            crossAxisSpacing: 10,

            mainAxisSpacing: 10,

          ),

          itemCount: 20, // 👈 stops after 20 items

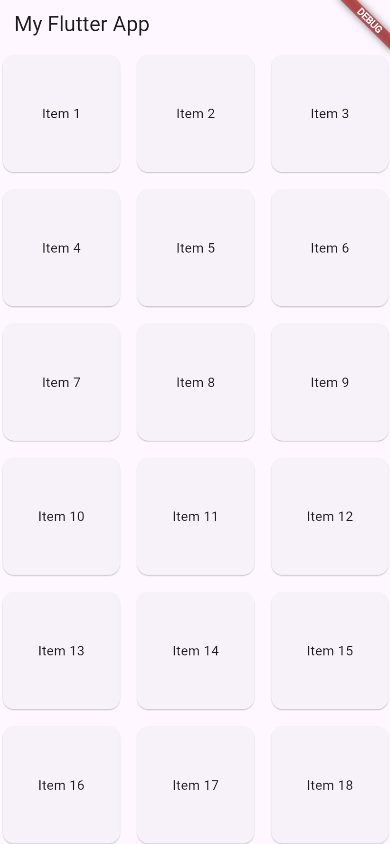
          itemBuilder: (context, index) {

            return Card(child: Center(child: Text('Item ${index + 1}')));

          },

        ),

      ),



👉 So, in **GridView.builder**:

* itemCount → how many items.
* itemBuilder → builds each item.
* gridDelegate → controls how the grid arranges them

1. ***SliverGridDelegateWithMaxCrossAxisExtent***

* You decide the **max width** of each item.
* Flutter automatically decides how many columns fit.

 Scaffold(

        appBar: AppBar(title: Text('My Flutter App')),

        body: GridView.builder(

          gridDelegate: SliverGridDelegateWithMaxCrossAxisExtent(

            maxCrossAxisExtent: 200, // max width of each item

            crossAxisSpacing: 10,

            mainAxisSpacing: 10,

            childAspectRatio: 1,

          ),

          itemCount: 20, // 👈 stops after 20 items

          itemBuilder: (context, index) {

            return Card(child: Center(child: Text('Item ${index + 1}')));

          },

        ),

      ),

You will need to maxcrossaxisExtent to precesie maximun with of each item .

### Example : Gallery

import 'package:flutter/material.dart';

void main() {

  runApp(MyApp());

}

class MyApp extends StatelessWidget {

  final List<String> images = [

    "https://picsum.photos/200?1",

    "https://picsum.photos/200?2",

    "https://picsum.photos/200?3",

    "https://picsum.photos/200?4",

    "https://picsum.photos/200?5",

    "https://picsum.photos/200?6",

  ];

  @override

  Widget build(BuildContext context) {

    return MaterialApp(

      debugShowCheckedModeBanner: false,

      home: Scaffold(

        appBar: AppBar(title: Text("Gallery App")),

        body: GridView.builder(

          padding: EdgeInsets.all(10),

          gridDelegate: SliverGridDelegateWithFixedCrossAxisCount(

            crossAxisCount: 2, // 2 columns

            crossAxisSpacing: 10,

            mainAxisSpacing: 10,

          ),

          itemCount: images.length,

          itemBuilder: (context, index) {

            return ClipRRect(

              borderRadius: BorderRadius.circular(15),

              child: Image.network(images[index], fit: BoxFit.cover),

            );

          },

        ),

      ),

    );

  }

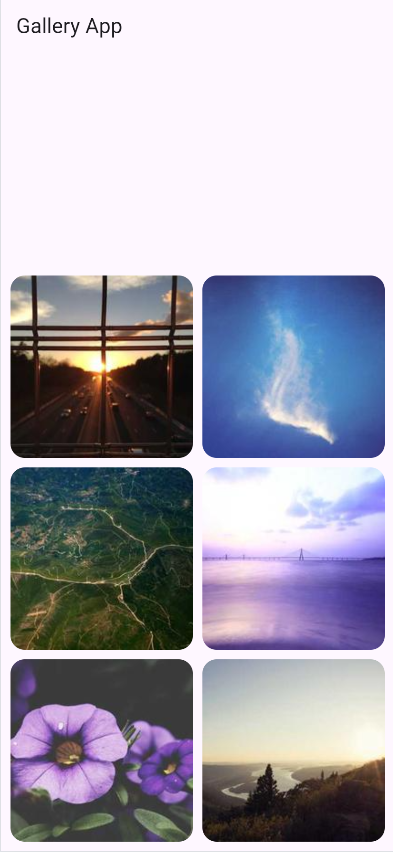
}



I want just to tell that reverse exists hhh,

 **reverse: true** → makes the scroll direction **opposite** (items appear from the bottom/right instead of top/left).

 Works in **ListView** and **GridView** (useful for chat apps or bottom-to-top lists).



# Card

A **Card** in Flutter is just a **pre-styled box** (rounded corners + shadow) that you can put any widget inside.

It comes from **Material widgets** → these are Flutter’s ready-made building blocks that follow Google’s **Material Design** (buttons, cards, app bars, etc.). They save you time because you don’t have to style everything from scratch.

Container(

          child: Card(

            child: Padding(

              padding: EdgeInsets.all(16),

              child: Text("Hello from a Card!"),

            ),

          ),

        )

👉 Think of **Material widgets** like **Lego blocks**: you snap them together instead of carving each block yourself.

The **Card** is “prestyled” because Flutter already gives it some **default visual effects** without you writing extra code:

* **Rounded corners** → by default, the card has slightly curved edges.
* **Shadow (elevation)** → it looks like it’s lifted above the background.
* **White background** (in light theme) → so content looks clean and separated.
* **Margin** → small space around the card so it doesn’t stick to other widgets.

Example:

child: Card(child: Text("I already have corners + shadow + margin"))



So when you use a Card, you don’t need to manually set border radius, box shadow, or spacing — Flutter’s **Material design** does it for you.

# ListTile

A **ListTile** is another **Material widget**. It’s a ready-made row layout that usually has:

* **leading** → something at the start (icon, image, avatar).
* **title** → main text.
* **subtitle** → smaller text below the title.
* **trailing** → something at the end (icon, button, text).

So instead of writing a Row + Column + Padding manually, you just use ListTile.

**Example inside a Card:**

Card(

          child: ListTile(

            onTap: () => {print("ListTile tap  ped")},

            leading: Icon(Icons.person),

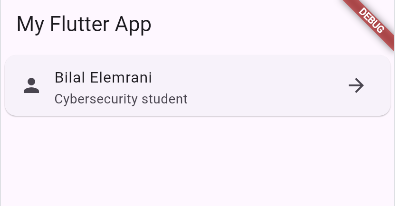
            title: Text("Bilal Elemrani"),

            subtitle: Text("Cybersecurity student"),

            trailing: Icon(Icons.arrow\_forward),

          ),

        ),



**👉 This gives you a ready profile/info row in one line of code.**

**Print become console.log if you use chorme as a device .**

* ListTile = the content (name, icon, etc.).
* Card = the frame that makes it stand out.

Let’s continue ListTile properties

* isThreeLine => set to true if you want space for 3 line of text (titile + 2 lines subtitile).

Normally, a ListTile is made for **2 lines only** (title + 1 subtitle line).  
If your subtitle text is **long** and you want it to wrap into **2 lines**, you set isThreeLine: true.

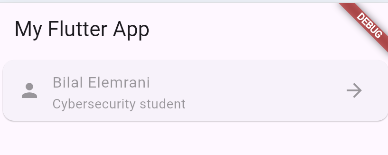
👉 Without it, Flutter cuts the text or makes it cramped.  
👉 With it, the tile gets taller to fit **3 total lines** (title + 2 subtitle lines).

* Dense => makes the tile more compact (less hight)
* ContentPadding => control inner spacing.
* enabled

Controls whether the tile is **active/clickable**.

Default: true → tile works normally and text/icons are full color.

If false → tile looks **grayed out** and **onTap does nothing**.



### Card has multiple properties

**Color** => Backgournd color for the card

**Elevation** => shadow depth (how high it looks).

**shadowColor** => color of the shadow

**margin** => space around the card.

**Shape** => customize border (rounded corners, border side, etc)

**clipBehavior** => how to clip child content (useful when you have images inside)

**child** => what hoes insed (text, image, listtile, column, etc)

Card(

          color: Colors.amber,

          elevation: 10,

          shadowColor: const Color.fromARGB(255, 5, 5, 5),

          shape: RoundedRectangleBorder(

            borderRadius: BorderRadius.circular(15),

            side: BorderSide(color: Colors.white, width: 1),

          ),

          child: ListTile(

            onTap: () => {print("ListTile tapped")},

            leading: Icon(Icons.person),

            title: Text("Bilal Elemrani"),

            subtitle: Text("Cybersecurity student"),

            trailing: Icon(Icons.arrow\_forward),

          ),

        ),

👉 So you can think of Card like a **box with ready-made style**, and these properties let you **customize how that box looks**.

# Icon And iconButton

## Icon

### What is an Icon ?

* An Icon is a **Material Widget** that shows **a pre-made symbol** (like a phone, person, menu, etc.)
* It’s **pre-styled**, so you don’t need to draw it manually.

### Main properties of Icon

* **Icons.<name>** => the icon itself (from the Material Icons library)
* **Size** => width & height in pixels
* **Color** => color of the icon
* **sem**anticalLabel => description for accessibility (screen readers).
* **textDirection** => if the icon should respect LTR/RTL text direction (rarely used).

ListTile(

            leading: Icon(

              Icons.phone,

              color: Colors.white,

              size: 30,

              semanticLabel: 'Call Icon',

            ),

            title: Text("The Owner"),

            subtitle: Text('+212 6 61 23 45 67'),

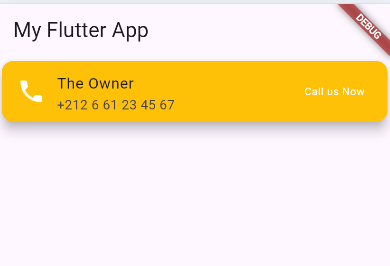
            trailing: Text(

              "Call us Now",

              style: TextStyle(color: Colors.white),

            ),

          ),



## iconButton

an IconButton is a button that contains only an icon (instead of text).

It’s like a normal ElevatedButton or TextButton, but smaller and only for icons.

IconButton(

            onPressed: () {

              print('Icon button pressed');

            },

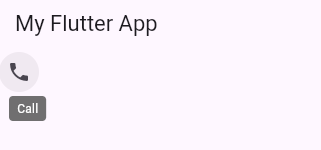
            tooltip: "Call",

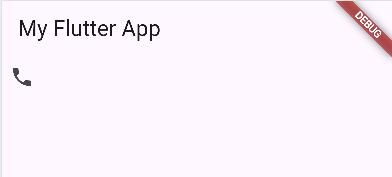
            icon: Icon(Icons.phone),

          ),

**Important Properties**

* **icon** → the icon you want to show (Icons.home, Icons.add, etc.).
* **onPressed** → what happens when the button is clicked.
* **color** → color of the icon.
* **iconSize** → size of the icon.
* **tooltip** → small text shown when you long-press or hover (on web/desktop).





Scaffold(

        appBar: AppBar(

          title: Text("My App"),

          actions: [

            IconButton(

              icon: Icon(Icons.search),

              onPressed: () {

                print("Search clicked");

              },

            ),

          ],

        ),

        body: Center(

          // <--- center the phone button

          child: IconButton(

            onPressed: () {

              print('Icon button pressed');

            },

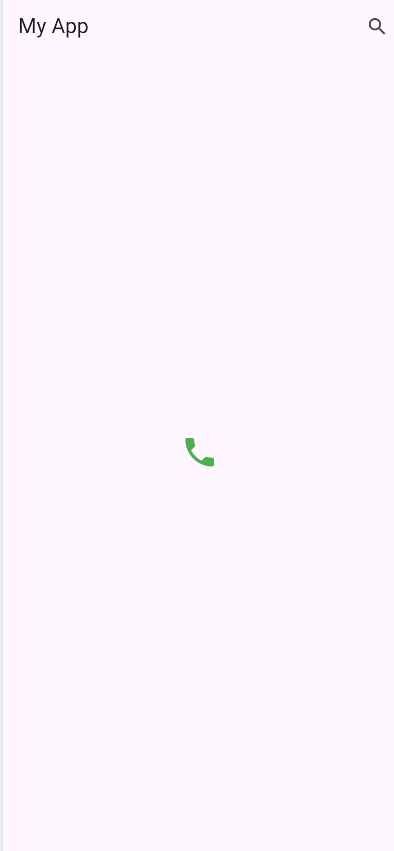
            tooltip: "Call", // shows text on hover/long-press

            icon: Icon(Icons.phone, size: 40, color: Colors.green),

          ),

        ),

      ),



But it would be better if you specified size outsie icon , meaning in the iconButton

# Expanded

## What is Expanded?

* Expanded tells a child widget inside a **Row** or **Column**:  
  👉 *“Take as much space as you can, according to the available room.”*
* It divides the free space between children.
* If you don’t wrap a child with Expanded, it keeps its **natural size**.

Scaffold(

        appBar: AppBar(title: Text("My App")),

        body: Container(

          child: Row(

            children: [

              Expanded(child: Container(color: Colors.red, height: 100)),

              Expanded(

                flex: 2,

                child: Container(color: Colors.green, height: 100),

              ),

              Expanded(child: Container(color: Colors.blue, height: 100)),

              Expanded(child: Container(color: Colors.yellow, height: 100)),

            ],

          ),

        ),

      ),



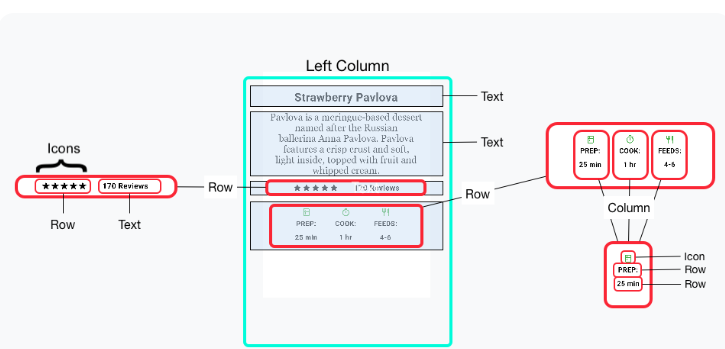
In this Row:

* Red, Blue, and Yellow each get **1 share** of space.
* Green has flex: 2, so it gets **2 shares**.
* Total = 1 + 2 + 1 + 1 = 5 shares.
* The width is divided into 5 equal parts:
  + Red = 1/5
  + Green = 2/5
  + Blue = 1/5
  + Yellow = 1/5

And sure the same with column :

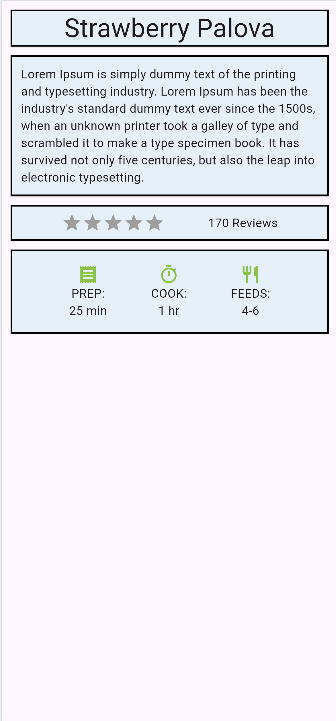


# Example 1 :



Link : <https://docs.flutter.dev/ui/layout>

Try to do it yourself , and then let’s solve it .



import 'package:flutter/material.dart';

void main() {

  runApp(MyApp());

}

class MyApp extends StatelessWidget {

  @override

  build(BuildContext context) {

    return MaterialApp(

      debugShowCheckedModeBanner: false,

      home: Scaffold(

        body: Container(

          padding: EdgeInsets.all(10),

          child: Column(

            crossAxisAlignment: CrossAxisAlignment.stretch,

            children: [

              Card(

                margin: EdgeInsets.all(0),

                color: Color(0xFFE6F0FA),

                shape: Border.all(

                  color: Colors.black,

                  width: 2,

                  style: BorderStyle.solid,

                ),

                child: Text(

                  "Strawberry Palova",

                  style: TextStyle(fontSize: 30),

                  textAlign: TextAlign.center,

                ),

              ),

              Card(

                margin: EdgeInsets.symmetric(vertical: 10),

                color: Color(0xFFE6F0FA),

                shape: Border.all(

                  color: Colors.black,

                  width: 2,

                  style: BorderStyle.solid,

                ),

                child: Padding(

                  padding: EdgeInsetsGeometry.all(12),

                  child: Text(

                    "Lorem Ipsum is simply dummy text of the printing and typesetting industry. Lorem Ipsum has been the industry's standard dummy text ever since the 1500s, when an unknown printer took a galley of type and scrambled it to make a type specimen book. It has survived not only five centuries, but also the leap into electronic typesetting.",

                  ),

                ),

              ),

              Container(

                margin: EdgeInsets.only(bottom: 10),

                padding: EdgeInsets.all(7),

                decoration: BoxDecoration(

                  color: Color(0xFFE6F0FA),

                  border: Border.all(

                    color: Colors.black,

                    width: 2,

                    style: BorderStyle.solid,

                  ),

                ),

                child: Row(

                  mainAxisAlignment: MainAxisAlignment.spaceEvenly,

                  children: [

                    Row(

                      children: [

                        Icon(Icons.star, color: Colors.grey),

                        Icon(Icons.star, color: Colors.grey),

                        Icon(Icons.star, color: Colors.grey),

                        Icon(Icons.star, color: Colors.grey),

                        Icon(Icons.star, color: Colors.grey),

                      ],

                    ),

                    Text("170 Reviews"),

                  ],

                ),

              ),

              Container(

                padding: EdgeInsets.all(15),

                decoration: BoxDecoration(

                  color: Color(0xFFE6F0FA),

                  border: Border.all(

                    color: Colors.black,

                    width: 2,

                    style: BorderStyle.solid,

                  ),

                ),

                child: Row(

                  mainAxisAlignment: MainAxisAlignment.spaceEvenly,

                  children: [

                    Column(

                      children: [

                        Icon(Icons.receipt, color: Colors.lightGreen),

                        Text("PREP:"),

                        Text("25 min"),

                      ],

                    ),

                    Column(

                      children: [

                        Icon(Icons.timer\_sharp, color: Colors.lightGreen),

                        Text("COOK:"),

                        Text("1 hr"),

                      ],

                    ),

                    Column(

                      children: [

                        Icon(Icons.restaurant, color: Colors.lightGreen),

                        Text("FEEDS:"),

                        Text("4-6"),

                      ],

                    ),

                  ],

                ),

              ),

            ],

          ),

        ),

      ),

    );

  }

}

I tried my best for now, I know that I make many mistakes but it’s good or great hhh for now .

# Button

## Button Types

Since flutter 2.0, you should use these 3 main button types :

* 1. ElevatedButton => raised with shadow .
  2. TextButton => flat, just text.
  3. outlinedButton => border outline.

(older ones like FlatButton, RaisedButton are deprected 🚫).



import 'package:flutter/material.dart';

void main() {

  runApp(MyApp());

}

class MyApp extends StatelessWidget {

  @override

  Widget build(BuildContext context) {

    return MaterialApp(

      home: Scaffold(

        body: Center(

          child: Column(

            mainAxisAlignment: MainAxisAlignment.center,

            children: [

              // ElevatedButton

              ElevatedButton(

                onPressed: () {},

                style: ElevatedButton.styleFrom(

                  backgroundColor: Colors.blue,

                  foregroundColor: Colors.white,

                  elevation: 8, // Shadow only works here

                ),

                child: Text("ElevatedButton"),

              ),

              SizedBox(height: 20),

              // OutlinedButton

              OutlinedButton(

                onPressed: () {},

                style: OutlinedButton.styleFrom(

                  foregroundColor: Colors.blue,

                  side: BorderSide(

                    color: Colors.blue,

                    width: 2,

                  ), // Border native

                ),

                child: Text("OutlinedButton"),

              ),

              SizedBox(height: 20),

              // TextButton

              TextButton(

                onPressed: () {},

                style: TextButton.styleFrom(foregroundColor: Colors.blue),

                child: Text("TextButton"),

              ),

            ],

          ),

        ),

      ),

    );

  }

}

👉 So, even if styled the same color:

* **ElevatedButton** ➝ has shadow (others cannot).
* **OutlinedButton** ➝ has border (native).
* **TextButton** ➝ stays flat, minimal.

**🔹 Real App Example (Login Screen)**

* ElevatedButton("Login") → Important action
* OutlinedButton("Sign Up") → Alternative option
* TextButton("Forgot Password?") → Extra, less important

Even if you style them all blue, **users feel the hierarchy** because of shadow/border/flatness.

## Common Properties (for all buttons)

Every modern button (ElevatedButton, textButton , OutlineButton) has :

* **onPressed:** 🡺 Function when tapped
* **onLongPress:** 🡺 Function when long-pressed
* **child** 🡺 Widget inside (Text, icon, Row, etc).
* **style** 🡺 Customize look with ButtonStyle or styleFrom.

## Styling Butttons

|  |  |  |
| --- | --- | --- |
| Property | What it does | Example |
| backgroundColor | Button’s background fill color | backgroundColor: Colors.blue |
| foregroundColor | Text & icon color inside button | foregroundColor: Colors.white |
| shadowColor | Shadow color (for ElevatedButton) | shadowColor: Colors.black |
| elevation | Shadow size (how raised it looks) | elevation: 5 |
| overlayColor | Color when pressed/hovered | overlayColor: Colors.red.withOpacity(0.2) |
| padding | Inner spacing around child | padding: EdgeInsets.all(15) |
| minimumSize | Minimum width & height | minimumSize: Size(200, 50) |
| maximumSize | Maximum width & height | maximumSize: Size(300, 60) |
| fixedSize | Exact size | fixedSize: Size(150, 50) |
| side | Border styling (color & width) | side: BorderSide(color: Colors.blue, width: 2) |
| shape | Border radius / shape | shape: RoundedRectangleBorder(borderRadius: BorderRadius.circular(12)) |
| alignment | Align child inside button | alignment: Alignment.centerLeft |
| textStyle | Customize text (font, size, weight) | textStyle: TextStyle(fontSize: 18, fontWeight: FontWeight.bold) |
| iconColor | Icon color (when using icons) | iconColor: Colors.yellow |

ElevatedButton(

            onPressed: () {},

            style: ElevatedButton.styleFrom(

              backgroundColor: Colors.blue, // button color

              foregroundColor: Colors.white, // text/icon color

              shadowColor: Colors.black, // shadow color

              elevation: 5, // shadow size

              padding: EdgeInsets.all(15), // inner space

              minimumSize: Size(200, 50), // button size

              shape: RoundedRectangleBorder(

                // border radius

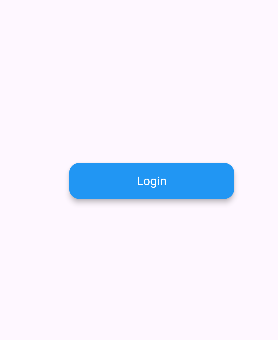
                borderRadius: BorderRadius.circular(12),

              ),

            ),

            child: Text("Login"),

          ),



## Floating Button

### What is a FloatingActionButton?

* A **round button** that “floats” above content.
* Typically used for the **main action** of a screen (example: ➕ add new item, ✉️ compose message).
* Usually placed at the **bottom-right corner** of the screen.
* It’s part of **Material Design**.

Scaffold(

        appBar: AppBar(title: Text("FAB Example")),

        body: Center(child: Text("Hello World")),

        floatingActionButton: FloatingActionButton(

          onPressed: () {

            print("FAB tapped");

          },

          child: Icon(Icons.add),

          tooltip: "Add Item",

        ),

        floatingActionButtonLocation: FloatingActionButtonLocation.endTop,

      ),



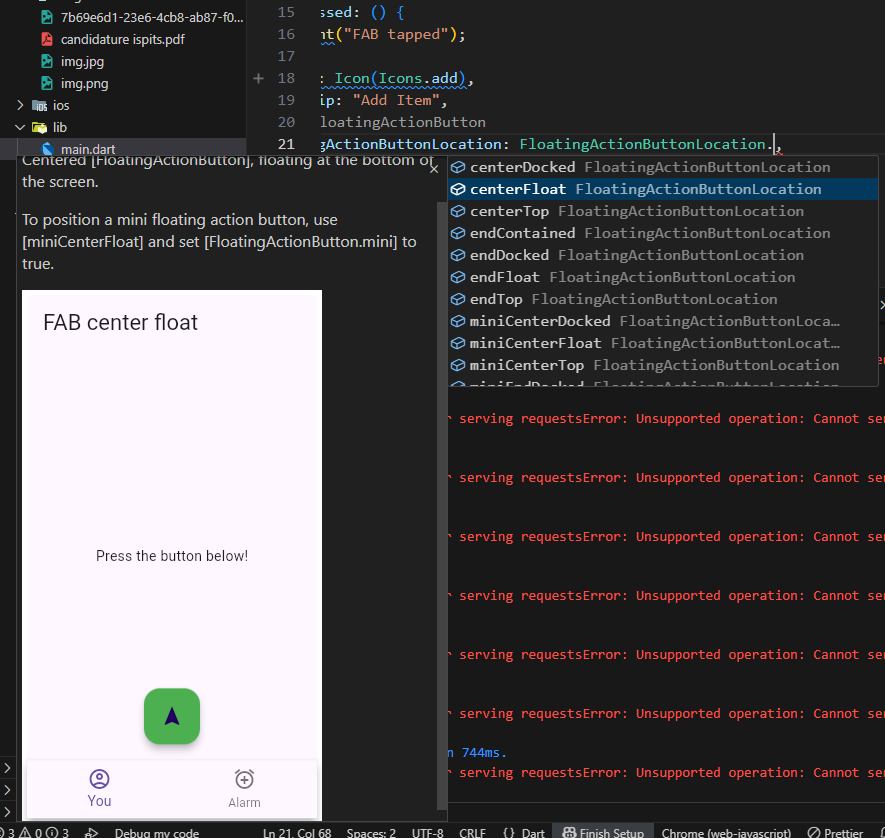
### Important Properties

|  |  |  |
| --- | --- | --- |
| Property | What it does | Example |
| onPressed | Action when tapped | onPressed: () {} |
| child | Content inside (usually Icon) | child: Icon(Icons.add) |
| tooltip | Text shown on long press (for accessibility) | tooltip: "Add item" |
| backgroundColor | FAB color | backgroundColor: Colors.blue |
| foregroundColor | Icon/text color | foregroundColor: Colors.white |
| elevation | Shadow depth | elevation: 6 |
| shape | Change shape (default is circle) | shape: RoundedRectangleBorder(borderRadius: BorderRadius.circular(10)) |
| mini | Smaller FAB (40px instead of 56px) | mini: true |
| heroTag | Used for hero animations between screens | heroTag: "fab1" |
| isExtended | If you use FloatingActionButton.extended → shows **icon + label** | FloatingActionButton.extended(...) |

✅ FAB is **different from normal buttons** because:

* It **floats** (not inline with content).
* It is **round** by default.
* It represents the **primary action** of a screen.

As you see in the picture there is many button location , you can see an image that show the place by hovering on it in vscode



# StatefulWidget & SetState

As we said earlier , that stateLessWidget is used with static page and stateFullWidget is used in interactive pages.

I know you does not understand, but let’s start making the things easier.

class MyApp extends StatelessWidget {

  @override

  Widget build(BuildContext context) {

    return MaterialApp(

      home: Scaffold(

        appBar: AppBar(title: Text("StatefulWidget & SetState")),

        body: Container(

          padding: EdgeInsets.all(10),

          child: Column(

            children: [

              IconButton(

                onPressed: () {

                  print("plus");

                },

                icon: Icon(Icons.add),

              ),

              Text("Number Now is : i"),

              IconButton(

                onPressed: () {

                  print("remove");

                },

                icon: Icon(Icons.remove),

              ),

            ],

          ),

        ),

      ),

    );

  }

}

I think now, you understand what I want to do

* I want => when plus btn is clicked => increase number by 1
* I want => when minus btn is clicked => decrease number by 1.

class MyApp extends StatelessWidget {

  int i = 0;

  @override

  Widget build(BuildContext context) {

    return MaterialApp(

      home: Scaffold(

        appBar: AppBar(title: Text("StatefulWidget & SetState")),

        body: Container(

          padding: EdgeInsets.all(10),

          child: Column(

            children: [

              IconButton(

                onPressed: () {

                  i++;

                  print("plus, $i");

                },

                icon: Icon(Icons.add),

              ),

              Text("Number Now is : $i"),

              IconButton(

                onPressed: () {

                  i--;

                  print("remove, $i");

                },

                icon: Icon(Icons.remove),

              ),

            ],

          ),

        ),

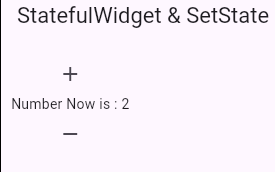
      ),

    );

  }

}

Now I see in the output that the number changes, but in the ui doesn’t.



Aaah but if I click, ctr+s (saving file), it’s rebuild my app and show me the new number.

But in reel app I can’t go code and save, and do all this staf , I need a function to rebuild my widgets on each press.

So here come the role and importance of **stateFullWidget**

class MyApp extends StatefulWidget {

  const MyApp({super.key});

  @override

  State<MyApp> createState() => \_MyAppState();

}

class \_MyAppState extends State<MyApp> {

  @override

  Widget build(BuildContext context){

    return

  }

}

And this how to create a statefullwidget.

\_Name : meaning it’s private.

And now all that we have to do is to past the material app in the new statefullwidget.

class MyApp extends StatefulWidget {

  const MyApp({super.key});

  State<MyApp> createState() => \_MyAppState();

}

class \_MyAppState extends State<MyApp> {

  int i = 0;

  @override

  Widget build(BuildContext context) {

    return MaterialApp(

      home: Scaffold(

        appBar: AppBar(title: Text("StatefulWidget & SetState")),

        body: Container(

          padding: EdgeInsets.all(10),

          child: Column(

            children: [

              IconButton(

                onPressed: () {

                  i++;

                  print("plus, $i");

                },

                icon: Icon(Icons.add),

              ),

              Text("Number Now is : $i"),

              IconButton(

                onPressed: () {

                  i--;

                  print("remove, $i");

                },

                icon: Icon(Icons.remove),

              ),

            ],

          ),

        ),

      ),

    );

  }

}

But now althought creating statefullwidget the same problem, still with us .. hhh

So ok , you did statefullwidget for rebuild but you did not told him , when you whant to do this rebuild.

## setState

**🔹 What is setState?**

* setState is a method in Flutter used **inside StatefulWidgets**.
* It **tells Flutter that something has changed** in the widget, and the UI should be rebuilt to show the new values.
* Without setState, the UI won’t update even if variables change.

🔹 Syntax

***setState(() {***

***// change some state variables here***

***});***

* The function inside setState is where you update the data (variables).
* After that, Flutter **rebuilds the widget tree** and shows the new UI.

#### Continue our example

import 'package:flutter/material.dart';

void main() {

  runApp(MyApp());

}

class MyApp extends StatefulWidget {

  const MyApp({super.key});

  State<MyApp> createState() => \_MyAppState();

}

class \_MyAppState extends State<MyApp> {

  int i = 0;

  @override

  Widget build(BuildContext context) {

    return MaterialApp(

      home: Scaffold(

        appBar: AppBar(title: Text("StatefulWidget & SetState")),

        body: Container(

          padding: EdgeInsets.all(10),

          child: Column(

            children: [

              IconButton(

                onPressed: () {

                  setState(() {

                    i++;

                  });

                  print("plus, $i");

                },

                icon: Icon(Icons.add),

              ),

              Text("Number Now is : $i"),

              IconButton(

                onPressed: () {

                  setState(() {

                    i--;

                  });

                  print("remove, $i");

                },

                icon: Icon(Icons.remove),

              ),

            ],

          ),

        ),

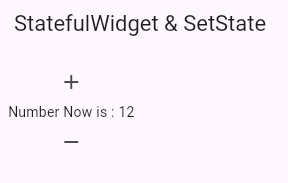
      ),

    );

  }

}

Now is working perfectly :



#### 🔹 Important Notes

* Use setState only in **StatefulWidgets** (not in Stateless).
* Don’t put heavy logic inside setState. Do calculations outside, then just update values inside.
* Every setState triggers a rebuild, so don’t overuse it.

#### Other example

import 'package:flutter/material.dart';

void main() {

  runApp(MyApp());

}

class MyApp extends StatefulWidget {

  const MyApp({super.key});

  State<MyApp> createState() => \_MyAppState();

}

class \_MyAppState extends State<MyApp> {

  int num = 0;

  @override

  Widget build(BuildContext context) {

    return MaterialApp(

      home: Scaffold(

        appBar: AppBar(title: Text("Star System For Rating")),

        body: Center(

          child: Container(

            padding: EdgeInsets.all(10),

            child: Row(

              mainAxisAlignment: MainAxisAlignment.center,

              children: [

                IconButton(

                  onPressed: () {

                    setState(() {

                      num = num == 0 ? 1 : 0;

                    });

                  },

                  icon: (num > 1 || num == 0)

                      ? Icon(Icons.star, color: Colors.amber)

                      : Icon(Icons.star\_border\_outlined),

                ),

                IconButton(

                  onPressed: () {

                    setState(() {

                      num = 2;

                    });

                  },

                  icon: num >= 2

                      ? Icon(Icons.star, color: Colors.amber)

                      : Icon(Icons.star\_border\_outlined),

                ),

                IconButton(

                  onPressed: () {

                    setState(() {

                      num = 3;

                    });

                  },

                  icon: num >= 3

                      ? Icon(Icons.star, color: Colors.amber)

                      : Icon(Icons.star\_border\_outlined),

                ),

                IconButton(

                  onPressed: () {

                    setState(() {

                      num = 4;

                    });

                  },

                  icon: num >= 4

                      ? Icon(Icons.star, color: Colors.amber)

                      : Icon(Icons.star\_border\_outlined),

                ),

                IconButton(

                  onPressed: () {

                    setState(() {

                      num = 5;

                    });

                  },

                  icon: num >= 5

                      ? Icon(Icons.star, color: Colors.amber)

                      : Icon(Icons.star\_border\_outlined),

                ),

              ],

            ),

          ),

        ),

      ),

    );

  }

}

The code is woking great but it will be better if I use loop



Simplified and cleaned code using loop

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 num = 0; // rating value

  @override

  Widget build(BuildContext context) {

    return MaterialApp(

      home: Scaffold(

        appBar: AppBar(title: Text("Star System For Rating")),

        body: Center(

          child: Container(

            padding: EdgeInsets.all(10),

            child: Row(

              mainAxisAlignment: MainAxisAlignment.center,

              children: List.generate(5, (index) {

                return IconButton(

                  onPressed: () {

                    setState(() {

                      // toggle if clicked same star again

                      num = (num == index + 1) ? 0 : index + 1;

                    });

                  },

                  icon: num >= index + 1

                      ? Icon(Icons.star, color: Colors.amber)

                      : Icon(Icons.star\_border\_outlined),

                );

              }),

            ),

          ),

        ),

      ),

    );

  }

}

Much cleaner and scalable → if you want 10 stars, just change List.generate(5, …) to List.generate(10, …).

# SwitchListTile & switch

**🔹 What is a Switch?**

* A **Switch widget** is a simple **ON/OFF toggle button**.
* It is usually used for **settings** (e.g., dark mode, notifications ON/OFF).
* Needs a **boolean value (true/false)** to track its state.

Switch(

                  value: true,

                  onChanged: (val) {

                    print(val);

                  },

                ),



But when we swip the value still false so.

class \_MyAppState extends State<MyApp> {

  bool status = false;

  @override

  Widget build(BuildContext context) {

    return MaterialApp(

      home: Scaffold(

        appBar: AppBar(title: Text("Switch")),

        body: Center(

          child: Container(

            padding: EdgeInsets.all(10),

            child: Column(

              children: [

                Switch(

                  value: status,

                  onChanged: (val) {

                    setState(() {

                      status = val;

                    });

                  },

                ),

              ],

            ),

          ),

        ),

      ),

    );

  }

}

Now is woking great , all that I do is creating a variable a name is status sho when I click a change it value and then rebuild .

#### Example

class \_MyAppState extends State<MyApp> {

  bool isSwitched = false; // initial state

  @override

  Widget build(BuildContext context) {

    return MaterialApp(

      home: Scaffold(

        appBar: AppBar(title: Text("Switch Example")),

        body: Center(

          child: Column(

            mainAxisAlignment: MainAxisAlignment.center,

            children: [

              Switch(

                value: isSwitched,

                onChanged: (bool value) {

                  setState(() {

                    isSwitched = value; // update state

                  });

                },

              ),

              SizedBox(height: 20),

              Text(

                isSwitched ? "Switch is ON 🔆" : "Switch is OFF 🌙",

                style: TextStyle(fontSize: 20),

              ),

            ],

          ),

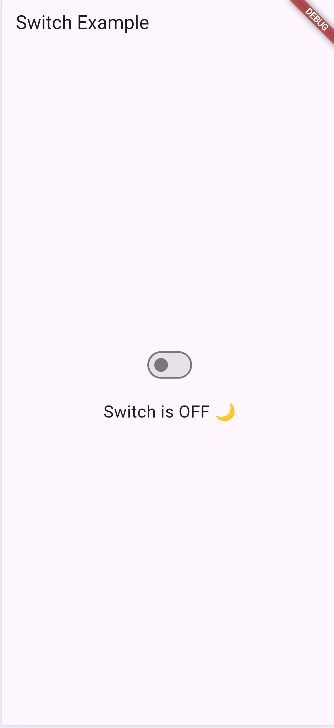
        ),

      ),

    );

  }

}

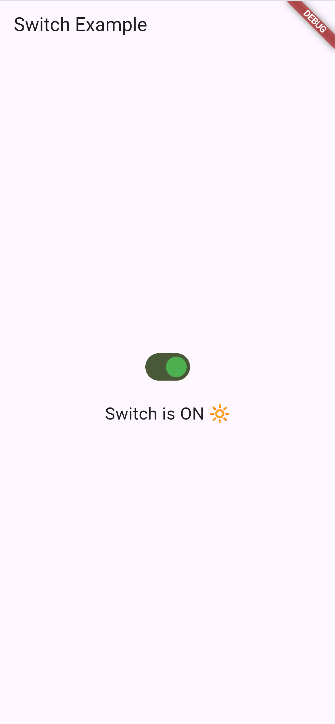


**🔹 Explanation**

* value: isSwitched → tells the switch whether it’s ON or OFF.
* onChanged: (value) { ... } → runs when the user toggles it.
* setState → updates isSwitched so the UI rebuilds.
* The text below shows **different messages depending on switch state**.

### Properties

|  |  |  |
| --- | --- | --- |
| Property | Type | Description |
| value | bool | Current state of the switch (**true = ON**, **false = OFF**). |
| onChanged | Function(bool) | Called when the user toggles the switch → update state here with setState. |
| activeColor | Color | Color of the **thumb (circle)** when switch is ON. |
| activeTrackColor | Color | Color of the **track (background bar)** when switch is ON. |
| inactiveThumbColor | Color | Color of the thumb when switch is OFF. |
| inactiveTrackColor | Color | Color of the track when switch is OFF. |
| hoverColor | Color | Color shown when mouse hovers (desktop/web). |
| focusColor | Color | Color shown when switch is focused. |
| materialTapTargetSize | MaterialTapTargetSize | Changes tap target size (default padded). |
| thumbColor | MaterialStateProperty<Color?> | Allows more advanced styling for the thumb based on states (pressed, hovered, etc.). |
| trackColor | MaterialStateProperty<Color?> | Advanced styling for the track depending on state. |



Switch(

                value: isSwitched,

                activeThumbColor: Colors.green,

                inactiveThumbColor: Colors.black,

                activeTrackColor: const Color.fromARGB(255, 73, 88, 54),

                inactiveTrackColor: Colors.white,

                onChanged: (bool value) {

                  setState(() {

                    isSwitched = value;

                  });

                },

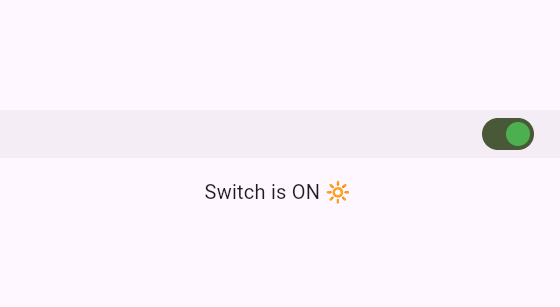
              ),

## SwitchListTile

**🔹 What is SwitchListTile?**

* It’s like a **Switch + ListTile combined**.
* You get a **label (title, subtitle, icon, etc.)** + **Switch** in one widget.
* Commonly used in **settings screens** (e.g., *Dark Mode ON/OFF*).

It like switch but it takes all the availbe space .



Where ever I click it will respond, it is easy .

### 🔹 Properties of SwitchListTile

|  |  |  |
| --- | --- | --- |
| Property | Type | Description |
| title | Widget | Main text (e.g., “Dark Mode”). |
| subtitle | Widget | Secondary text under the title. |
| secondary | Widget | An icon or widget placed before the text. |
| value | bool | Whether the switch is ON (true) or OFF (false). |
| onChanged | Function(bool) | Called when user toggles the switch. |
| activeColor | Color | Thumb color when ON. |
| activeTrackColor | Color | Track color when ON. |
| inactiveThumbColor | Color | Thumb color when OFF. |
| inactiveTrackColor | Color | Track color when OFF. |
| isThreeLine | bool | If true, allows 3 lines of text (title + subtitle). |
| dense | bool | Makes the tile more compact. |
| contentPadding | EdgeInsets | Custom padding inside the tile. |



Using title .

# RadioListTile & Radio

## Radio

### 🔹 What is a Radio?

* A **Radio button** lets the user **choose exactly ONE option from a group**.
* Example: Choosing gender → Male / Female / Other.
* Different from Checkbox (where multiple can be selected).

### 🔹 Properties of Radio

|  |  |  |
| --- | --- | --- |
| Property | Type | Description |
| value | T (generic) | The option represented by this radio. |
| groupValue | T | The currently selected option → must match one of the radios. |
| onChanged | Function(T?)? | Callback when radio is selected. |
| activeColor | Color | Color of the radio when selected. |
| fillColor | MaterialStateProperty<Color?> | Advanced color styling for states. |
| toggleable | bool | If true, tapping again unselects the radio. |

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> {

  String? country; // ? for acepting nul value

  @override

  Widget build(BuildContext context) {

    return MaterialApp(

      home: Scaffold(

        appBar: AppBar(title: Text("Switch Example")),

        body: Center(

          child: Column(

            mainAxisAlignment: MainAxisAlignment.center,

            children: [

              RadioListTile(

                title: Text("Morocco"),

                value: "Morocco",

                groupValue: country,

                onChanged: (value) {

                  setState(() {

                    country = value;

                  });

                },

              ),

              RadioListTile(

                title: Text("USA"),

                value: "USA",

                groupValue: country,

                onChanged: (value) {

                  setState(() {

                    country = value;

                  });

                },

              ),

              RadioListTile(

                title: Text("France"),

                value: "France",

                groupValue: country,

                onChanged: (value) {

                  setState(() {

                    country = value;

                  });

                },

              ),

              RadioListTile(

                title: Text("Germany"),

                value: "Germany",

                groupValue: country,

                onChanged: (value) {

                  setState(() {

                    country = value;

                  });

                },

              ),

              RadioListTile(

                title: Text("Japan"),

                value: "Japan",

                groupValue: country,

                onChanged: (value) {

                  setState(() {

                    country = value;

                  });

                },

              ),

              Text('You Selected: ${country != null ? country : ""}'),

            ],

          ),

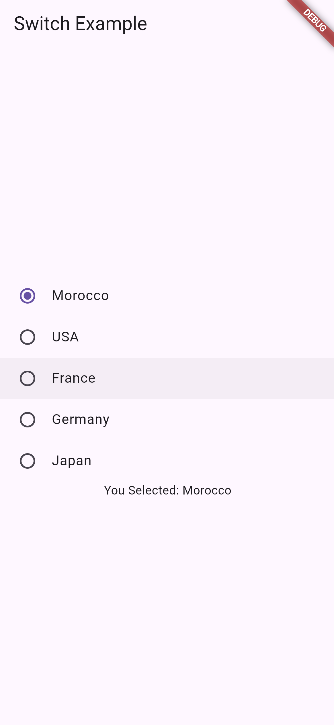
        ),

      ),

    );

  }

}



**🔹 RadioListTile in Flutter**

**Definition**

RadioListTile is a **Material Design widget** that combines a **radio button** and a **ListTile**.

* It lets the user select **one option from a group**.
* Provides **title, subtitle, and leading/trailing widgets** automatically.
* The radio button is on the **leading side** by default, with the text next to it.

It’s basically a **radio + label in one widget**.

Properties of RadioListTile

|  |  |  |
| --- | --- | --- |
| Property | Type | Description |
| value | T | The value represented by this radio. |
| groupValue | T | The currently selected value in the group. |
| onChanged | ValueChanged<T?>? | Callback when user selects the radio. Usually updates state. |
| title | Widget? | Primary text of the tile. |
| subtitle | Widget? | Secondary text under the title. |
| secondary | Widget? | Widget displayed opposite the radio button (like an icon). |
| activeColor | Color? | Color of the radio when selected. |
| toggleable | bool | If true, tapping again will deselect the radio. Default false. |
| dense | bool? | Reduces vertical space of the tile. |
| controlAffinity | ListTileControlAffinity | Position of the radio relative to the text (leading/trailing). |
| contentPadding | EdgeInsetsGeometry? | Custom padding inside the tile. |
| visualDensity | VisualDensity? | Controls the compactness of the tile. |
| tileColor | Color? | Background color of the tile (when not selected). |
| selectedTileColor | Color? | Background color when tile is selected. |
| selected | bool | Whether the tile is visually selected (affects title color etc.). |
| focusColor | Color? | Color when the tile is focused. |
| hoverColor | Color? | Color when the tile is hovered (desktop/web). |
| autofocus | bool | Whether this tile should automatically receive focus. |
| shape | ShapeBorder? | Custom shape of the tile (borders, rounded corners). |
| enableFeedback | bool | Whether taps should provide haptic or audible feedback. |
| mouseCursor | MouseCursor? | Cursor when hovering (desktop/web). |

Example :

import 'package:flutter/material.dart';

void main() {

  runApp(const MyApp());

}

class MyApp extends StatefulWidget {

  const MyApp({super.key});

  @override

  State<MyApp> createState() => \_MyAppState();

}

class \_MyAppState extends State<MyApp> {

  String? country; // selected country

  @override

  Widget build(BuildContext context) {

    return MaterialApp(

      home: Scaffold(

        appBar: AppBar(title: const Text("RadioListTile Example")),

        body: Padding(

          padding: const EdgeInsets.all(16.0),

          child: Column(

            children: [

              RadioListTile<String>(

                title: const Text("Morocco"),

                subtitle: const Text("This is Morocco"),

                value: "Morocco",

                groupValue: country,

                onChanged: (value) {

                  setState(() {

                    country = value;

                  });

                },

                secondary: const Icon(Icons.flag),

              ),

              RadioListTile<String>(

                title: const Text("USA"),

                subtitle: const Text("This is USA"),

                value: "USA",

                groupValue: country,

                onChanged: (value) {

                  setState(() {

                    country = value;

                  });

                },

                secondary: const Icon(Icons.flag),

              ),

              RadioListTile<String>(

                title: const Text("France"),

                subtitle: const Text("This is France"),

                value: "France",

                groupValue: country,

                onChanged: (value) {

                  setState(() {

                    country = value;

                  });

                },

                secondary: const Icon(Icons.flag),

              ),

              const SizedBox(height: 20),

              Text(

                'You Selected: ${country ?? ""}',

                style: const TextStyle(fontSize: 18),

              ),

            ],

          ),

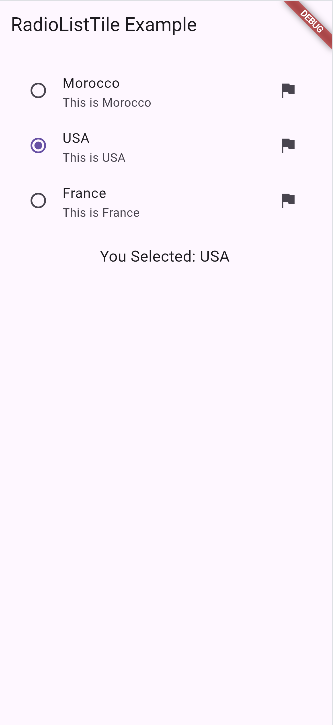
        ),

      ),

    );

  }

}



# CheckboxListTile & Checkbox

**🔹 1. Checkbox**

**Definition**

Checkbox is a **Material Design widget** that lets the user **select or deselect a single option**.

* Unlike Radio, it’s **multi-select capable** — you can check multiple checkboxes independently.

Properties of Checkbox

|  |  |  |
| --- | --- | --- |
| Property | Type | Description |
| value | bool? | Whether the checkbox is checked (true) or unchecked (false). |
| onChanged | ValueChanged<bool?>? | Callback when the user taps the checkbox; usually calls setState. |
| activeColor | Color? | Color of the checkbox when selected. |
| checkColor | Color? | Color of the check mark inside the checkbox. |
| tristate | bool | If true, checkbox can have true, false, or null (indeterminate state). Default false. |
| materialTapTargetSize | MaterialTapTargetSize? | Adjusts hit target size. |
| autofocus | bool | Automatically receives focus when screen loads. |
| focusNode | FocusNode? | Node used to manage focus. |
| visualDensity | VisualDensity? | Controls compactness of the widget. |
| side | BorderSide? | Custom border of the checkbox. |
| shape | ShapeBorder? | Shape of the checkbox (rounded, square, etc.). |
| splashRadius | double? | Splash effect radius when tapped. |

🔹 **2. CheckboxListTile**

**Definition**

CheckboxListTile is a **Checkbox + ListTile** combined in one widget.

* Lets you add **title, subtitle, secondary widget, and custom styling** easily.
* Saves you from wrapping a Checkbox in a ListTile.

Properties of CheckboxListTile

|  |  |  |
| --- | --- | --- |
| Property | Type | Description |
| value | bool? | Whether the checkbox is selected. |
| onChanged | ValueChanged<bool?>? | Callback when tapped; usually calls setState. |
| title | Widget? | Primary text. |
| subtitle | Widget? | Secondary text. |
| secondary | Widget? | Widget displayed opposite the checkbox (icon, image, etc.). |
| activeColor | Color? | Color of checkbox when selected. |
| checkColor | Color? | Color of the check mark inside the box. |
| tileColor | Color? | Background color of the tile. |
| selectedTileColor | Color? | Background color when selected. |
| dense | bool? | Reduce vertical space of the tile. |
| contentPadding | EdgeInsetsGeometry? | Custom padding inside the tile. |
| autofocus | bool | Automatically receive focus. |
| tristate | bool | If true, checkbox can be true, false, or null. |
| controlAffinity | ListTileControlAffinity | Position of the checkbox (leading/trailing). |

**✅ Key Notes**

* Checkbox → just the box, you need to wrap in Row or ListTile for text.
* CheckboxListTile → box + text + subtitle + icon + styling.
* Both use bool (or tristate) for state.
* Use setState inside onChanged to update the UI.

| **Property** | **Checkbox** | **CheckboxListTile** | **Notes** |
| --- | --- | --- | --- |
| value | bool? | bool? | Whether the box is checked (true) or unchecked (false) |
| onChanged | ValueChanged<bool?>? | ValueChanged<bool?>? | Called when user taps; usually inside setState |
| tristate | bool (default false) | bool (default false) | Allows true, false, or null (indeterminate state) |
| activeColor | Color? | Color? | Color when checked |
| checkColor | Color? | Color? | Color of the check mark |
| title | ❌ Not available | Widget? | Main text for the tile |
| subtitle | ❌ Not available | Widget? | Secondary text |
| secondary | ❌ Not available | Widget? | Widget opposite the checkbox (icon, image) |
| tileColor | ❌ Not available | Color? | Background color when not selected |
| selectedTileColor | ❌ Not available | Color? | Background color when selected |
| selected | ❌ Not available | bool | Whether the tile is visually selected |
| dense | ❌ Not available | bool? | Reduce vertical space |
| controlAffinity | ❌ Not available | ListTileControlAffinity | Position of checkbox (leading/trailing) |
| contentPadding | ❌ Not available | EdgeInsetsGeometry? | Inner padding of tile |
| autofocus | bool | bool | Auto-focus on load |
| focusNode | FocusNode? | FocusNode? | Custom focus node |
| visualDensity | VisualDensity? | VisualDensity? | Compactness of widget |
| shape | ShapeBorder? | ShapeBorder? | Shape of the checkbox or tile |
| enableFeedback | bool | bool | Haptic/audio feedback |
| mouseCursor | MouseCursor? | MouseCursor? | Cursor on desktop/web |
| materialTapTargetSize | MaterialTapTargetSize? | MaterialTapTargetSize? | Hit target size |

**Summary**

1. **Use Checkbox** when you just need a **standalone box** and will handle text/layout yourself.
2. **Use CheckboxListTile** when you want a **checkbox with title, subtitle, secondary widget, and styling**, all in one.
3. Both support tristate, activeColor, and checkColor.
4. Always use setState in onChanged to update the UI.

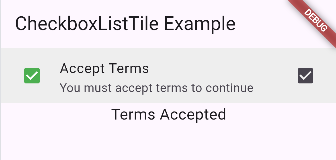
***Note***

**The ! Operator (Null Assertion)**

* In Dart, ! after a variable **tells the compiler**:

“I am sure this value is **not null**, so treat it as non-nullable.”

#### Example



import 'package:flutter/material.dart';

void main() {

  runApp(const MyApp());

}

class MyApp extends StatefulWidget {

  const MyApp({super.key});

  @override

  State<MyApp> createState() => \_MyAppState();

}

class \_MyAppState extends State<MyApp> {

  bool isChecked = false;

  @override

  Widget build(BuildContext context) {

    return MaterialApp(

      home: Scaffold(

        appBar: AppBar(title: const Text("CheckboxListTile Example")),

        body: Column(

          children: [

            CheckboxListTile(

              title: const Text("Accept Terms"),

              subtitle: const Text("You must accept terms to continue"),

              value: isChecked,

              onChanged: (bool? value) {

                setState(() {

                  isChecked = value!;

                });

              },

              secondary: const Icon(Icons.check\_box),

              tileColor: Colors.grey[200],

              selectedTileColor: Colors.green[100],

              activeColor: Colors.green,

              checkColor: Colors.white,

              controlAffinity: ListTileControlAffinity.leading,

            ),

            Text(

              isChecked ? "Terms Accepted" : "Terms Not Accepted",

              style: const TextStyle(fontSize: 18),

            ),

          ],

        ),

      ),

    );

  }

}

Other simple example :



import 'package:flutter/material.dart';

void main() {

  runApp(const MyApp());

}

class MyApp extends StatefulWidget {

  const MyApp({super.key});

  @override

  State<MyApp> createState() => \_MyAppState();

}

class \_MyAppState extends State<MyApp> {

  bool basketBall = false;

  bool footBall = false;

  bool tennis = false;

  bool volleyball = false;

  bool swimming = false;

  bool badminton = false;

  bool cricket = false;

  @override

  Widget build(BuildContext context) {

    return MaterialApp(

      home: Scaffold(

        appBar: AppBar(title: const Text("CheckboxListTile Example")),

        body: Column(

          children: [

            Text("Choose Your Hobbies", style: TextStyle(fontSize: 23)),

            CheckboxListTile(

              value: basketBall,

              title: Text("BasketBall"),

              secondary: Icon(Icons.sports\_basketball),

              onChanged: (val) => {

                setState(() {

                  basketBall = val!;

                }),

              },

            ),

            CheckboxListTile(

              value: footBall,

              title: Text("FootBall"),

              secondary: Icon(Icons.sports\_football\_sharp),

              onChanged: (val) => {

                setState(() {

                  footBall = val!;

                }),

              },

            ),

            CheckboxListTile(

              value: tennis,

              title: Text("Tennis"),

              secondary: Icon(Icons.sports\_tennis),

              onChanged: (val) => {

                setState(() {

                  tennis = val!;

                }),

              },

            ),

            CheckboxListTile(

              value: volleyball,

              title: Text("Volleyball"),

              secondary: Icon(Icons.sports\_volleyball),

              onChanged: (val) => {

                setState(() {

                  volleyball = val!;

                }),

              },

            ),

            CheckboxListTile(

              value: swimming,

              title: Text("Swimming"),

              secondary: Icon(Icons.pool),

              onChanged: (val) => {

                setState(() {

                  swimming = val!;

                }),

              },

            ),

            CheckboxListTile(

              value: badminton,

              title: Text("Badminton"),

              secondary: Icon(Icons.sports\_tennis),

              onChanged: (val) => {

                setState(() {

                  badminton = val!;

                }),

              },

            ),

            CheckboxListTile(

              value: cricket,

              title: Text("Cricket"),

              secondary: Icon(Icons.sports\_cricket),

              onChanged: (val) => {

                setState(() {

                  cricket = val!;

                }),

              },

            ),

            Text(

              "Your Favorite hobbie(s): ${[if (basketBall) "BasketBall", if (footBall) "FootBall", if (tennis) "Tennis", if (volleyball) "Volleyball", if (swimming) "Swimming", if (badminton) "Badminton", if (cricket) "Cricket"].join(" And ")}",

            ),

          ],

        ),

      ),

    );

  }

}

# Stack

**1️⃣ What is a Stack?**

* **Stack** is a **layout widget** in Flutter that lets you **place widgets on top of each other**.
* Think of it like **layers in Photoshop**: the first child is at the bottom, the next child is on top, etc.
* Useful for **overlapping widgets** like badges, images with text overlays, or floating buttons.

Let understand well , see the next code :

Scaffold(

        appBar: AppBar(title: const Text("Stack")),

        body: Container(

          child: Column(

            children: [

              Container(width: 300, height: 300, color: Colors.red),

              Container(width: 300, height: 300, color: Colors.yellow),

              Container(width: 300, height: 300, color: Colors.green),

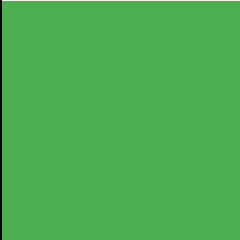
            ],

          ),

        ),

      ),

Let’s change the column by the stack



Only the top layer is shown , to see the previous one, let edit their diemenssion

Stack(

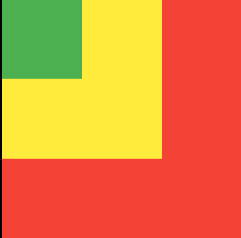
            children: [

              Container(width: 300, height: 300, color: Colors.red),

              Container(width: 200, height: 200, color: Colors.yellow),

              Container(width: 100, height: 100, color: Colors.green),

            ],

          ),

Inside the stack we can change the alignment :

Stack(

            alignment: Alignment.center,

            children: [

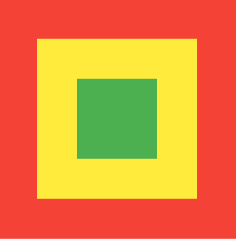
              Container(width: 300, height: 300, color: Colors.red),

              Container(width: 200, height: 200, color: Colors.yellow),

              Container(width: 100, height: 100, color: Colors.green),

            ],

          ),



2️⃣ Stack Properties

|  |  |
| --- | --- |
| Property | Description |
| alignment | Controls the alignment of children within the Stack (e.g., Alignment.center, Alignment.topRight). |
| fit | Determines how the non-positioned children are sized. Options: StackFit.loose (default), StackFit.expand. |
| clipBehavior | Determines if children can be drawn outside the Stack bounds (Clip.none, Clip.hardEdge). |
| children | List of widgets stacked on top of each other. |

### 3️⃣ Positioned Widget

**Positioned** is used **inside Stack** to place a widget at a specific position.

Properties: top, bottom, left, right, width, height.

Stack(

          children: [

            Container(width: 200, height: 200, color: Colors.blue),

            Positioned(

              top: 20,

              left: 20,

              child: Container(width: 100, height: 100, color: Colors.red),

            ),

            Positioned(

              bottom: 0,

              right: 0,

              child: Text(

                "Hello Stack",

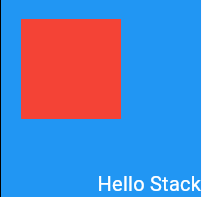
                style: TextStyle(color: Colors.white, fontSize: 20),

              ),

            ),

          ],

        ),



Here:

* Blue box is at the bottom.
* Red box is **on top** and shifted 20px from top & left.
* Text is **bottom-right corner**.

# TextField

The **TextField** widget is used to let users enter input in flutter apps ( like forms, search bars, login fields, etc).

**2️⃣ Main Properties of TextField**

I’ll group them by category so it’s easier for you:

🔹 **Input & Text Control**

|  |  |
| --- | --- |
| Property | Description |
| controller | A TextEditingController to read or control the text. |
| onChanged | Callback when text changes. |
| onSubmitted | Callback when user presses "done/enter". |
| onEditingComplete | Called when editing is finished (focus lost). |
| maxLength | Maximum characters allowed. |
| maxLines | Number of lines (default = 1). |
| minLines | Minimum number of lines (useful for multi-line input). |
| obscureText | Hides text (useful for passwords). |
| keyboardType | Type of keyboard (text, number, emailAddress, etc). |
| textInputAction | Action button in the keyboard (done, next, search). |

TextField(

                controller: TextEditingController(),

                onChanged: (value) {

                  print("Text changed: $value");

                },

                onSubmitted: (value) {

                  print("Submitted: $value");

                },

                onEditingComplete: () {

                  print("Editing complete");

                },

                maxLength: 20,

                maxLines: 1,

                minLines: 1,

                obscureText: false,

                keyboardType: TextInputType.text,

                textInputAction: TextInputAction.done,

              ),

#### 🔹 What does controller do in TextField?

A **TextEditingController** is an object that **manages and controls the text** inside a TextField.  
Think of it as the **remote control** for the input field.

**1️⃣ Without controller**

If you don’t use a controller:

TextField()

* You can type text ✅
* But you **can’t easily get the text** from it later in your code ❌

2️⃣ With controller

final myController = TextEditingController();

TextField(

controller: myController,

)

Now you can:

* **Read** what the user typed:

print(myController.text);

* **Set text** programmatically:

myController.text = "Hello Flutter!";

* **Clear the field**:

myController.clear();

I think you are asking about 2 others hhhh , I know you bro, you want to understand all .

#### keyboardType

This controls **what kind of keyboard** will show up when the user taps the field.

| **Value** | **Keyboard** |
| --- | --- |
| TextInputType.text | Normal text keyboard (default). |
| TextInputType.number | Number keypad only. |
| TextInputType.phone | Phone number keypad. |
| TextInputType.emailAddress | Keyboard with @ and . shortcuts. |
| TextInputType.url | Keyboard with / and .com. |
| TextInputType.multiline | Allows multi-line text. |
| TextInputType.datetime | Numbers + : for time input. |

#### textInputAction

This controls the **action button** on the keyboard (the button in the bottom-right corner).

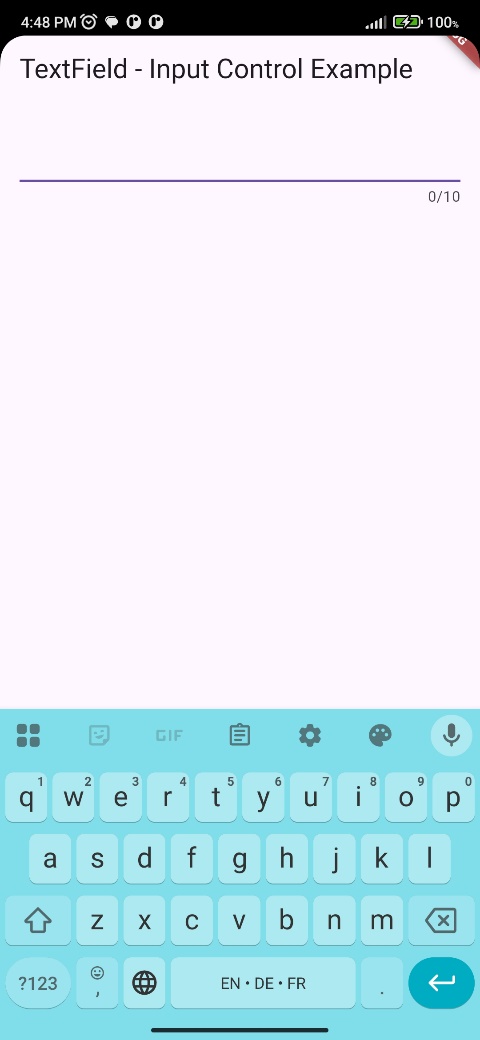
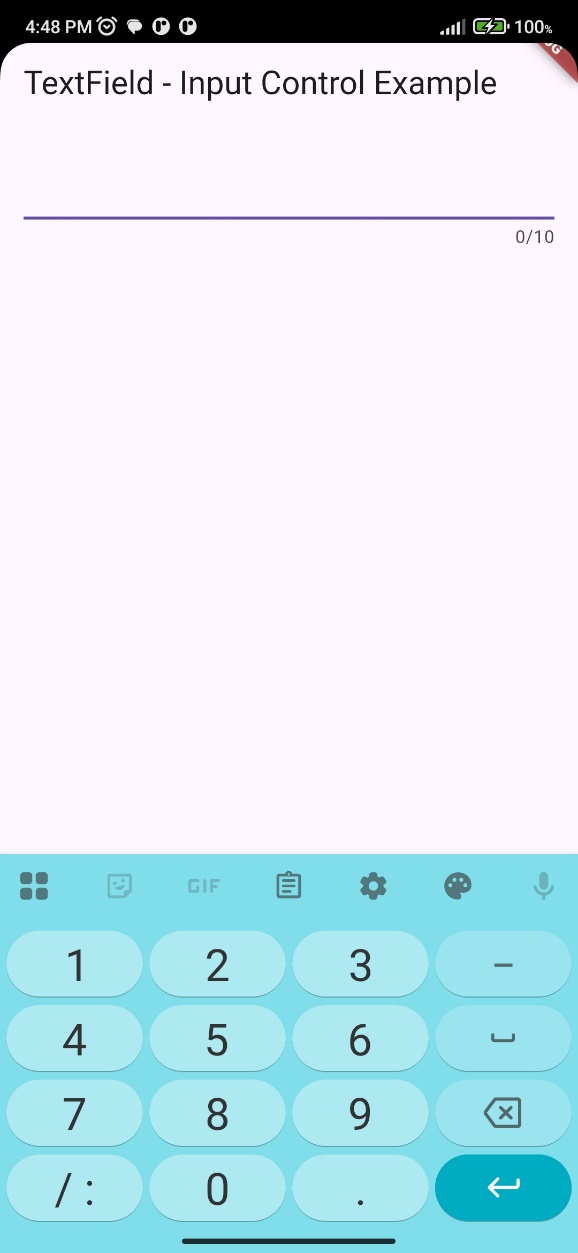
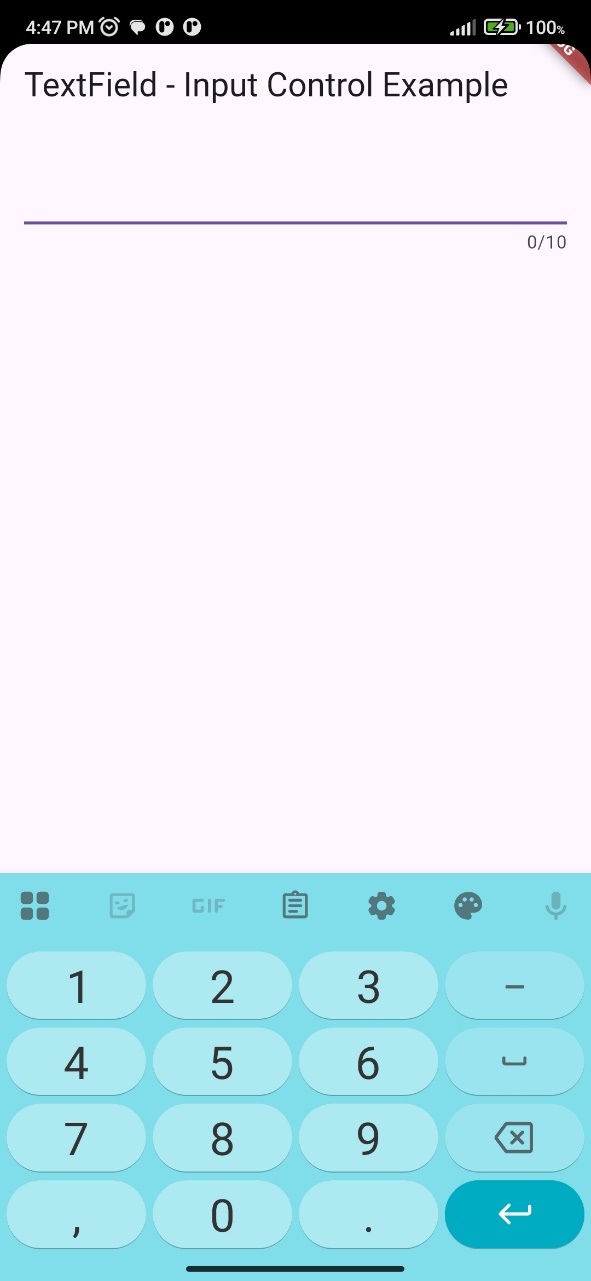
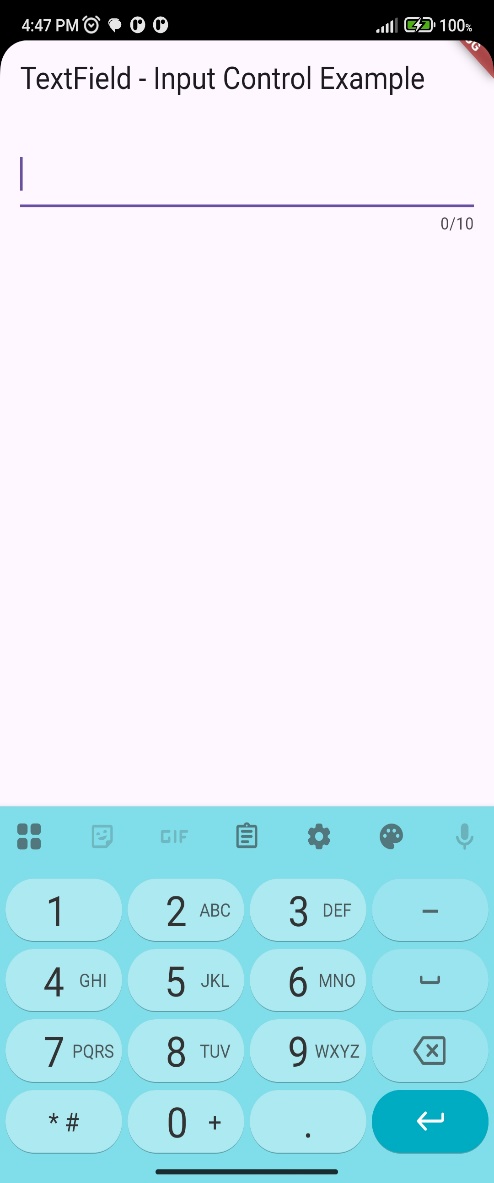
| **Value** | **Action Button shows as** |
| --- | --- |
| TextInputAction.done | ✅ Done |
| TextInputAction.next | ⏭ Next |
| TextInputAction.search | 🔍 Search |
| TextInputAction.go | ➡️ Go |
| TextInputAction.send | 📤 Send |
| TextInputAction.newline | ↵ New line |

👉 Example:

* In a **form with multiple fields**, you put textInputAction: TextInputAction.next so the user can jump to the next field.
* For a **password field**, you often use TextInputAction.done.

In chrome I didn’t see any diffrence I will use my phone .

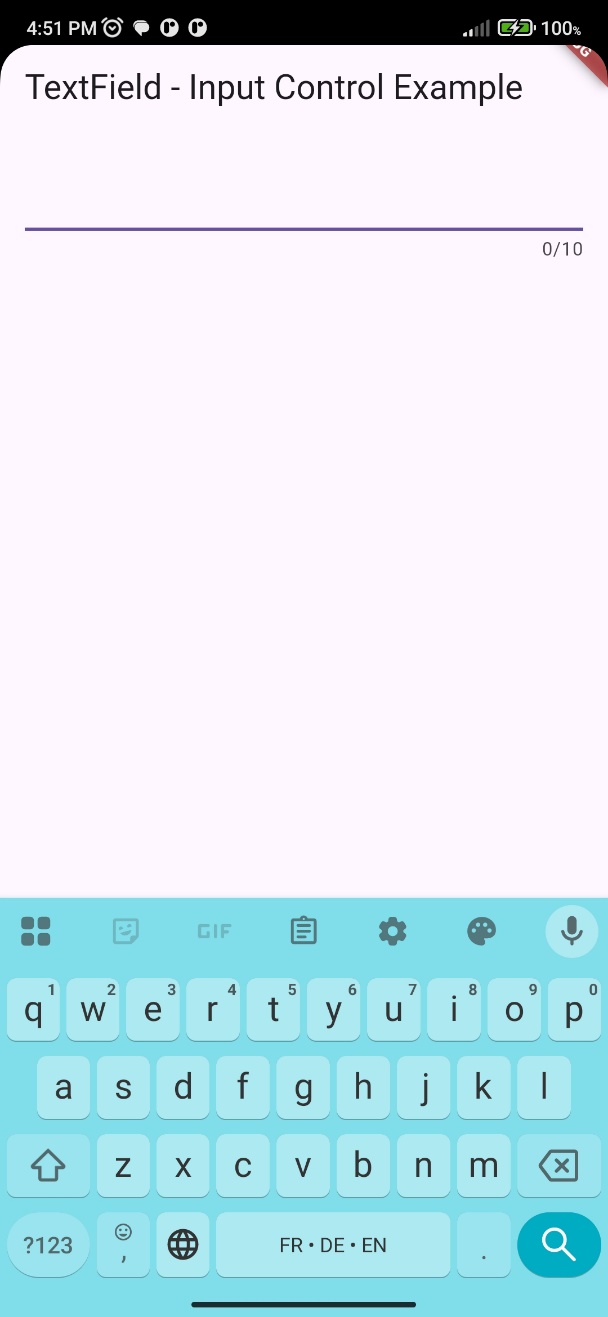
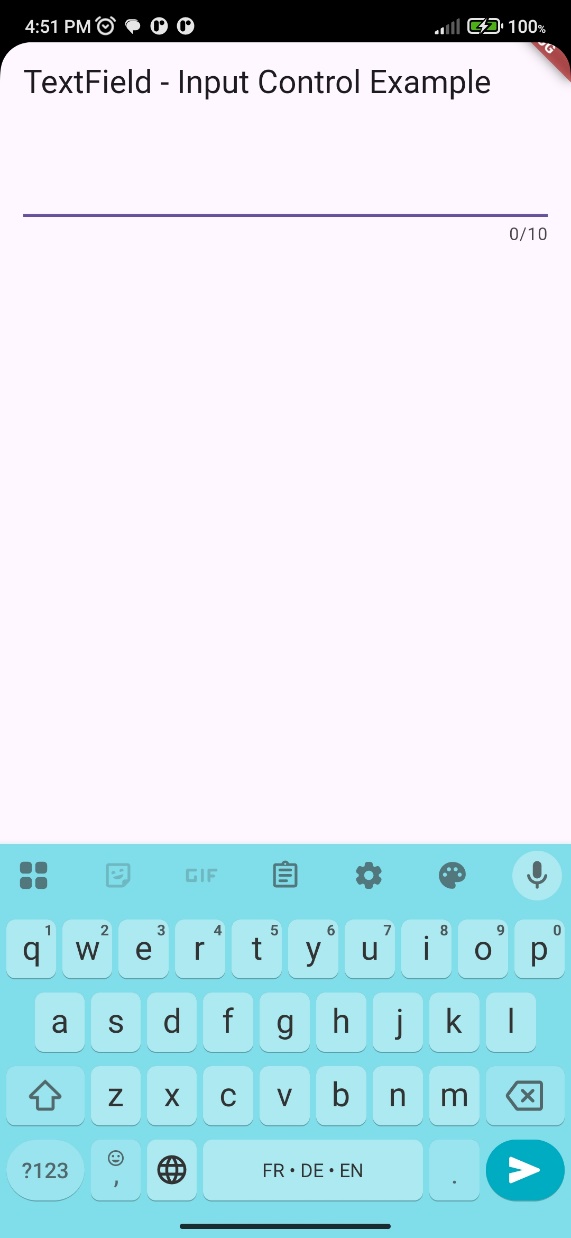
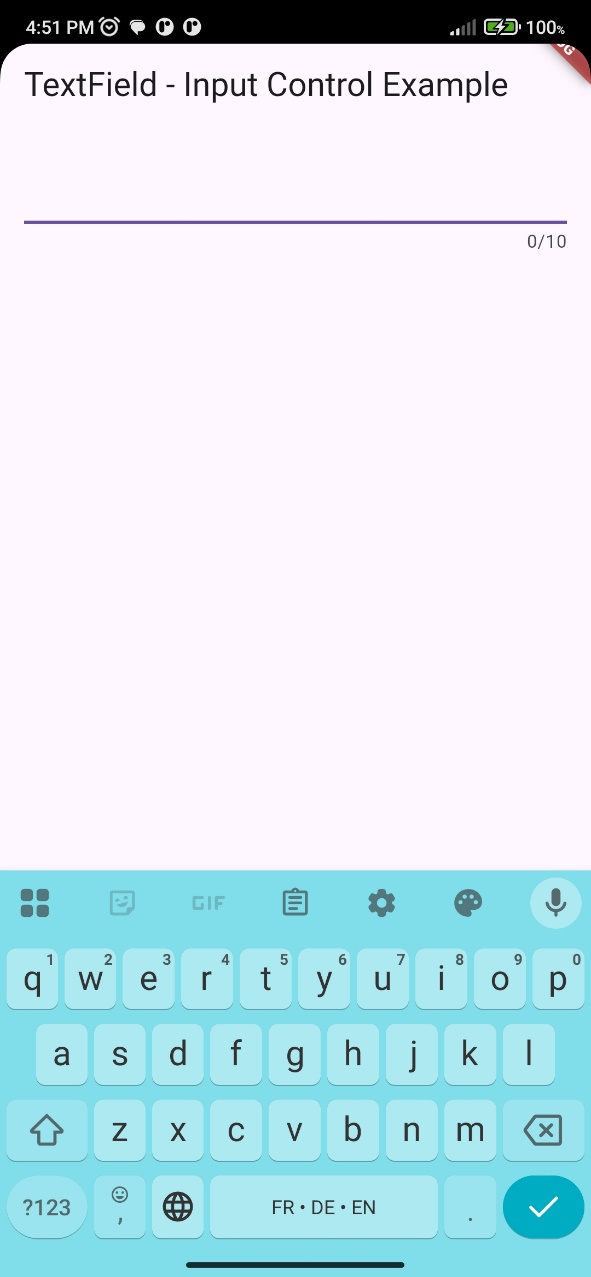
Goo to simulation .



In the url type , text + number above

In the images you can find phone , date, number and normal text .

Now you can find text input actions , like send or search extra .



Here we are, we finish simulation on our phone. Let continue .

## 🔹 Text Style & Alignment

|  |  |
| --- | --- |
| Property | Description |
| style | Style of the input text (TextStyle). |
| textAlign | Align text (left, right, center, etc). |
| textDirection | Direction (ltr or rtl). |
| cursorColor | Color of the blinking cursor. |
| cursorHeight | Height of the cursor. |
| cursorWidth | Thickness of the cursor. |

import 'package:flutter/material.dart';

void main() {

  runApp(MyApp());

}

class MyApp extends StatelessWidget {

  @override

  Widget build(BuildContext context) {

    return MaterialApp(

      debugShowCheckedModeBanner: false,

      home: Scaffold(

        appBar: AppBar(title: Text("TextField Style Example")),

        body: Padding(

          padding: const EdgeInsets.all(20.0),

          child: TextField(

            style: TextStyle(

              color: Colors.pink[300], // Input text color

              fontSize: 20, // Text size

              fontWeight: FontWeight.bold, // Bold

              letterSpacing: 4, // Space between letters

            ),

            textAlign: TextAlign.center, // Align text center

            textDirection: TextDirection.ltr, // Text direction (Left → Right)

            cursorColor: Colors.pink[50], // Cursor color ( | li kayflache fach katkon katktb)

            cursorHeight: 30, // Cursor height

            cursorWidth: 3, // Cursor thickness

            decoration: InputDecoration(

              hintText: "Enter Your Name ...", // Placeholder text

              border: OutlineInputBorder(), // Nice border

            ),

          ),

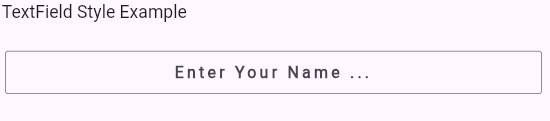
        ),

      ),

    );

  }

}



## 🔹 Decoration (via InputDecoration)

| **Property (inside InputDecoration)** | **Description** |
| --- | --- |
| labelText | Floating label above the field. |
| hintText | Placeholder inside the field. |
| prefixIcon | Icon before text. |
| suffixIcon | Icon after text (e.g., clear button, eye for password). |
| border | Border style (OutlineInputBorder, UnderlineInputBorder). |
| enabledBorder | Border style when not focused. |
| focusedBorder | Border style when focused. |
| errorText | Shows error message below field. |
| helperText | Small help text below field. |
| fillColor + filled | Background color. |
| contentPadding | Padding inside the field. |

import 'package:flutter/material.dart';

void main() {

  runApp(MyApp());

}

class MyApp extends StatelessWidget {

  @override

  Widget build(BuildContext context) {

    return MaterialApp(

      debugShowCheckedModeBanner: false,

      home: Scaffold(

        appBar: AppBar(title: Text("TextField Decoration Example")),

        body: Padding(

          padding: const EdgeInsets.all(20.0),

          child: TextField(

            decoration: InputDecoration(

              labelText: "Username", // Floating label

              hintText: "Enter your username", // Placeholder

              prefixIcon: Icon(Icons.person), // Icon before input

              suffixIcon: Icon(Icons.check), // Icon after input

              border: OutlineInputBorder(), // Default border

              enabledBorder: OutlineInputBorder(

                // Border when not focused

                borderSide: BorderSide(color: Colors.grey, width: 1.5),

              ),

              focusedBorder: OutlineInputBorder(

                // Border when focused

                borderSide: BorderSide(color: Colors.blue, width: 2.0),

              ),

              errorText: null, // Can set to "Invalid input"

              helperText: "Your username must be unique", // Help text below

              fillColor: Colors.yellow[100], // Background color

              filled: true, // Enable background color

              contentPadding: EdgeInsets.symmetric(

                // Inner padding

                horizontal: 20,

                vertical: 15,

              ),

            ),

          ),

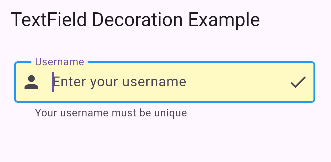
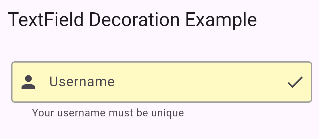
        ),

      ),

    );

  }

}



## 🔹 Behavior & Focus

| **Property** | **Description** |
| --- | --- |
| enabled | Enable/disable input. |
| readOnly | Make it read-only (no editing). |
| autofocus | Auto-focus when screen loads. |
| focusNode | Control focus manually. |
| enableSuggestions | Show keyboard suggestions. |
| autocorrect | Enable autocorrect. |

import 'package:flutter/material.dart';

void main() {

  runApp(MyApp());

}

class MyApp extends StatelessWidget {

  final FocusNode firstFocus = FocusNode();

  final FocusNode secondFocus = FocusNode();

  @override

  Widget build(BuildContext context) {

    return MaterialApp(

      home: Scaffold(

        appBar: AppBar(title: Text("TextField Behavior Example")),

        body: Padding(

          padding: const EdgeInsets.all(20.0),

          child: Column(

            children: [

              // 🔹 First field with autofocus

              TextField(

                autofocus: true, // Focus when app starts

                focusNode: firstFocus, // Custom focus node

                enableSuggestions: false,

                decoration: InputDecoration(labelText: "First Name"),

              ),

              SizedBox(height: 20),

              // 🔹 Second field (read-only + disabled)

              TextField(

                readOnly: true, // Can’t edit, but can select

                enabled: false, // Disabled completely

                decoration: InputDecoration(labelText: "Disabled Field"),

              ),

              SizedBox(height: 20),

              // 🔹 Third field with suggestions & autocorrect

              TextField(

                focusNode: secondFocus, // Controlled focus

                enableSuggestions: true, // Suggest words

                autocorrect: true, // Autocorrect mistakes

                decoration: InputDecoration(

                  labelText: "Write a message",

                  hintText: "Type something...",

                ),

              ),

              SizedBox(height: 20),

              // 🔹 Button to move focus

              ElevatedButton(

                onPressed: () {

                  FocusScope.of(context).requestFocus(firstFocus);

                },

                child: Text("Move focus to Message Field"),

              ),

            ],

          ),

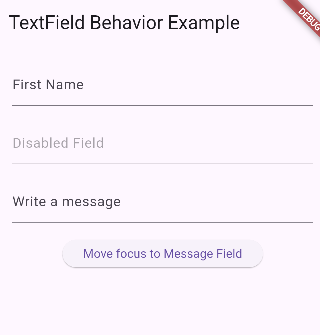
        ),

      ),

    );

  }

}



## 🔹 Security & Special Cases

| **Property** | **Description** |
| --- | --- |
| obscureText | Hide characters (passwords). |
| obscuringCharacter | Replace text with custom char (default: •). |
| enableInteractiveSelection | Enable copy/paste/select. |
| inputFormatters | Restrict input (e.g., only numbers). |



import 'package:flutter/material.dart';

import 'package:flutter/services.dart'; // Needed for inputFormatters

void main() {

  runApp(MyApp());

}

class MyApp extends StatelessWidget {

  @override

  Widget build(BuildContext context) {

    return MaterialApp(

      home: Scaffold(

        appBar: AppBar(title: Text("TextField Security Example")),

        body: Padding(

          padding: const EdgeInsets.all(20.0),

          child: Column(

            children: [

              // 🔹 Password field

              TextField(

                obscureText: true, // Hide text

                obscuringCharacter: "\*", // Replace with \*

                decoration: InputDecoration(

                  labelText: "Password",

                  prefixIcon: Icon(Icons.lock),

                ),

              ),

              SizedBox(height: 20),

              // 🔹 Disabled copy/paste

              TextField(

                enableInteractiveSelection: false, // No copy/paste

                decoration: InputDecoration(

                  labelText: "Secure Field (no copy/paste)",

                  prefixIcon: Icon(Icons.security),

                ),

              ),

              SizedBox(height: 20),

              // 🔹 Only numbers allowed

              TextField(

                keyboardType: TextInputType.number, // Show number keyboard

                inputFormatters: [

                  FilteringTextInputFormatter.digitsOnly, // Allow only digits

                ],

                decoration: InputDecoration(

                  labelText: "Enter digits only",

                  prefixIcon: Icon(Icons.numbers),

                ),

              ),

            ],

          ),

        ),

      ),

    );

  }

}

**🔍 What Happens**

* First field → password hidden (\* \* \* \*).
* Second field → user **cannot copy/paste/select text**.
* Third field → only **numbers are accepted** (letters blocked).

Let’s take some explanations for wal abo hamza course .

For border , there 2 types :

* **Underline border** is the *default* one .
* **Outline border**

We can control normal border and on focse also .

To control normal one, we have to enable border first .

TextField(

                decoration: InputDecoration(

                  border: OutlineInputBorder(),

                  enabledBorder: OutlineInputBorder(

                    borderSide: BorderSide(color: Colors.green),

                  ),

                  focusedBorder: OutlineInputBorder(

                    borderSide: BorderSide(

                      color: const Color.fromARGB(255, 234, 0, 255),

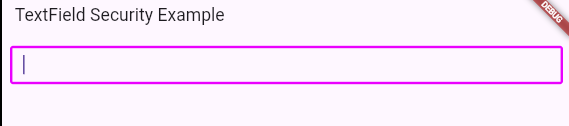
                      width: 3,

                    ),

                  ),

                ),

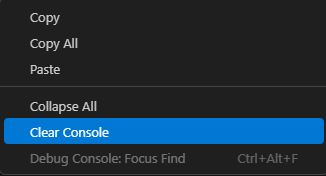
              ),

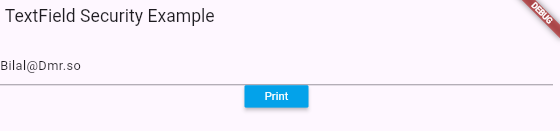


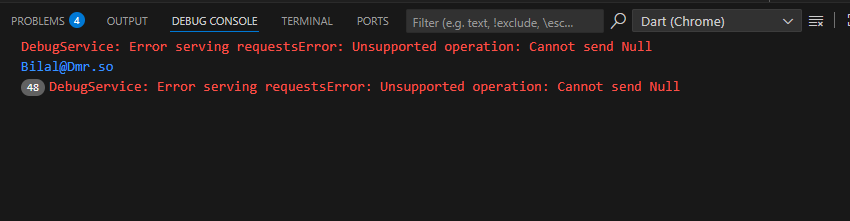
Green one is the normal one .

Pink one is the focus one .

#### Controller example

Controller is so imporant , here is an example , right click on debug console and clear it , to see the result clearly .

now click print

and here is the result in console

the code of this example here is it :

import 'package:flutter/material.dart';

void main() {

  runApp(MyApp());

}

class MyApp extends StatelessWidget {

  TextEditingController username = TextEditingController();

  @override

  Widget build(BuildContext context) {

    return MaterialApp(

      home: Scaffold(

        appBar: AppBar(title: Text("TextField Security Example")),

        body: Container(

          padding: EdgeInsets.all(10),

          child: Column(

            children: [

              TextField(controller: username),

              MaterialButton(

                onPressed: () {

                  print(username.text);

                },

                color: Colors.lightBlue,

                textColor: Colors.white,

                child: Text('Print'),

              ),

            ],

          ),

        ),

      ),

    );

  }

}

# TextFormField Validator

## 🔹 Step 1: GlobalKey<FormState>

***GlobalKey<FormState> formState = GlobalKey();***

* A **key** that uniquely identifies the Form.
* It gives us access to the **state** of the form (whether fields are valid or not).
* Without this, you can’t check **validate()** later.

## 🔹 Step 2: Form Widget

**Form(**

**key: formState,**

**child: Column(**

**children: [...]**

**),**

**)**

* A **container** for multiple input fields.
* The **key: formState** connects this form with the **GlobalKey<FormState>** we created.
* This allows validation across **all fields inside it**.

## 🔹 Step 3: TextFormField

**TextFormField(**

**validator: (value) {**

**if (value!.isEmpty) {**

**return "The Field is Empty";**

**}**

**if (value.length > 10) {**

**return "The value shouldn't be more than 10";**

**}**

**},**

**),**

**Focus well in this logic now :**

* TextFormField is like TextField but it has built-in support for **validation**.
* The validator function:
  + Runs automatically when we call **formState.currentState!.validate().**
  + If it returns a **String** → that string is shown as an **error message** under the field.
  + If it returns **null** → the input is considered valid.

👉 In your code:

* Empty field → "The Field is Empty".
* More than 10 chars → "The value shouldn't be more than 10".
* Otherwise → **valid** (no error message).

## 🔹 Step 4: MaterialButton

**MaterialButton(**

**onPressed: () {**

**if (formState.currentState!.validate()) {**

**print("Valid");**

**} else {**

**print("Not Valid");**

**}**

**},**

**color: Colors.blue,**

**textColor: Colors.white,**

**child: Text("Valid"),**

**),**

When the button is pressed:

* **formState.currentState!.validate()** goes through **all fields** inside the form.
* Runs every validator function.
* If **all fields return null** → result = true → prints "Valid".
* If **any field returns an error string** → result = false → prints "Not Valid" and shows error message below that field.

## 🔹 Flow of Your Code

1. User types something in TextFormField.
2. User presses the **button**.
3. Flutter calls the validator function.
4. If input is empty → error text appears.  
   If input > 10 → error text appears.  
   Else → form passes.
5. The print statement shows **"Valid"** or **"Not Valid"** in the console.

**🔹 Most Important Notes about Form & Validation**

**1. TextFormField must be inside a Form**

* TextFormField works with validation **only if it’s a child of Form**.
* If you use TextFormField outside a Form, validator will never run.

**2. How validator works**

* validator is a function that takes the field value (String? value).
* If it **returns a String** → that text is shown as an error message below the field.
* If it **returns null** → field is considered valid.  
  ✅ Example:

**validator: (value) {**

**if (value!.isEmpty) return "Field is empty";**

**return null; // valid**

**}**

**3. The FormState is controlled with a GlobalKey<FormState>**

* You create:
* GlobalKey<FormState> formKey = GlobalKey();
* You assign it to the Form:
* Form(key: formKey, child: ...)
* Then you can call:
  + formKey.currentState!.validate() → runs all validators.
  + formKey.currentState!.save() → saves the data (if you use onSaved).
  + formKey.currentState!.reset() → clears the form.

**4. Validation is not automatic**

* Validation happens **only when you call** formKey.currentState!.validate().
* Usually, you call it inside a button’s onPressed.

**5. Error messages appear automatically**

* When validator returns a String, Flutter shows it under the field **without extra code**.
* Example: "The Field is Empty" will be shown under the box.

**6. Multiple fields can be validated together**

* If a form has many TextFormFields, validate() checks **all of them**.
* If one field is invalid → the whole form is invalid.

**7. Good practice: Separate UI and validation logic**

* Keep validator functions small and clear.
* Example:

String? validateName(String? value) {

if (value == null || value.isEmpty) return "Name is required";

if (value.length < 3) return "Name too short";

return null;

}

Then call:

TextFormField(validator: validateName)

## Full code :

import 'package:flutter/material.dart';

void main() {

  runApp(MyApp());

}

class MyApp extends StatelessWidget {

  GlobalKey<FormState> formState = GlobalKey();

  @override

  Widget build(BuildContext context) {

    return MaterialApp(

      home: Scaffold(

        appBar: AppBar(title: Text("TextField Security Example")),

        body: Container(

          padding: EdgeInsets.all(10),

          child: Form(

            key: formState,

            child: Column(

              children: [

                TextFormField(

                  validator: (value) {

                    if (value!.isEmpty) {

                      return "The Field is Empty";

                    }

                    if (value.length > 10) {

                      return "The value shouldn't be more than 10";

                    }

                  },

                ),

                MaterialButton(

                  onPressed: () {

                    if (formState.currentState!.validate()) {

                      print("Valid");

                    } else {

                      print("Not Valid");

                    }

                  },

                  color: Colors.blue,

                  textColor: Colors.white,

                  child: Text("Valid"),

                ),

              ],

            ),

          ),

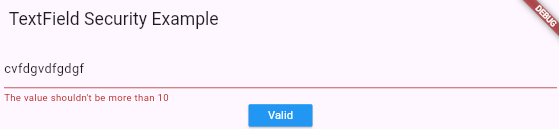
        ),

      ),

    );

  }

}





In the 2 cases , I have a printed value 🡺 “Not Valid”

# TextFormField onSaved

import 'package:flutter/material.dart';

void main() {

  runApp(MyApp());

}

class MyApp extends StatelessWidget {

  GlobalKey<FormState> formState = GlobalKey();

  String? userName;

  String? phone;

  @override

  Widget build(BuildContext context) {

    return MaterialApp(

      home: Scaffold(

        appBar: AppBar(title: Text("TextField Security Example")),

        body: Container(

          padding: EdgeInsets.all(10),

          child: Form(

            key: formState,

            child: Column(

              children: [

                TextFormField(

                  onSaved: (val) {

                    userName = val;

                  },

                  validator: (value) {},

                  decoration: InputDecoration(hint: Text("userName")),

                ),

                TextFormField(

                  onSaved: (val) {

                    phone = val;

                  },

                  validator: (value) {

                    if (value!.length > 10) {

                      return "The Number can't be more than 10";

                    }

                    if (value.isEmpty) {

                      return "The Field is Required";

                    }

                  },

                  decoration: InputDecoration(hint: Text("Phone")),

                ),

                MaterialButton(

                  onPressed: () {

                    formState.currentState!.validate();

                    print(phone);

                    print(userName);

                  },

                  color: Colors.blue,

                  textColor: Colors.white,

                  child: Text("Valid"),

                ),

              ],

            ),

          ),

        ),

      ),

    );

  }

}

Although I write values and click on button I get 2nul printed

import 'package:flutter/material.dart';

void main() {

  runApp(MyApp());

}

class MyApp extends StatelessWidget {

  GlobalKey<FormState> formState = GlobalKey();

  String? userName;

  String? phone;

  @override

  Widget build(BuildContext context) {

    return MaterialApp(

      home: Scaffold(

        appBar: AppBar(title: Text("TextField Security Example")),

        body: Container(

          padding: EdgeInsets.all(10),

          child: Form(

            key: formState,

            child: Column(

              children: [

                TextFormField(

                  onSaved: (val) {

                    userName = val;

                  },

                  validator: (value) {},

                  decoration: InputDecoration(hint: Text("userName")),

                ),

                TextFormField(

                  onSaved: (val) {

                    phone = val;

                  },

                  validator: (value) {

                    if (value!.length > 10) {

                      return "The Number can't be more than 10";

                    }

                    if (value.isEmpty) {

                      return "The Field is Required";

                    }

                  },

                  decoration: InputDecoration(hint: Text("Phone")),

                ),

                MaterialButton(

                  onPressed: () {

                    formState.currentState!.validate();

                    formState.currentState!.save();

                    print(phone);

                    print(userName);

                  },

                  color: Colors.blue,

                  textColor: Colors.white,

                  child: Text("Valid"),

                ),

              ],

            ),

          ),

        ),

      ),

    );

  }

}

But now is great

**1. What is onSaved?**

* A **callback** in TextFormField.
* It runs when you call:
* formState.currentState!.save();
* Used to **store the input value** into a variable.

**🔹 2. How it works**

Example:

String? userName;

TextFormField(

onSaved: (val) {

userName = val; // Save the value inside variable

},

)

👉 When you call save(), whatever the user typed will be assigned to userName.

**🔹 3. Difference between validator and onSaved**

* validator → **checks** if input is correct.
* onSaved → **stores** the input value into a variable.

They are often used together:

TextFormField(

validator: (value) {

if (value == null || value.isEmpty) return "Required field";

return null;

},

onSaved: (val) {

userName = val;

},

)

**🔹 4. Important Rule**

* onSaved only works **after you call formState.currentState!.save()**.
* Usually, you first validate, then save:
* if (formState.currentState!.validate()) {
* formState.currentState!.save();
* }

**✅ Summary**

* onSaved → used to **store field values** after validation.
* Runs only when you call .save().
* Works together with validator.
* Best practice: validate first, then save values.

Soooooooooo rememer you should do save inside validate please remember.

import 'package:flutter/material.dart';

void main() {

  runApp(MyApp());

}

class MyApp extends StatelessWidget {

  // GlobalKey to control the Form

  final GlobalKey<FormState> formKey = GlobalKey<FormState>();

  // Variables to hold saved values

  String? userName;

  String? phone;

  @override

  Widget build(BuildContext context) {

    return MaterialApp(

      home: Scaffold(

        appBar: AppBar(title: Text("Form Validation Best Practice")),

        body: Padding(

          padding: const EdgeInsets.all(16.0),

          child: Form(

            key: formKey,

            child: Column(

              children: [

                // Username field

                TextFormField(

                  decoration: InputDecoration(labelText: "Username"),

                  validator: (value) {

                    if (value == null || value.isEmpty) {

                      return "Username is required";

                    }

                    if (value.length < 3) {

                      return "Username must be at least 3 characters";

                    }

                    return null; // valid

                  },

                  onSaved: (val) {

                    userName = val;

                  },

                ),

                SizedBox(height: 16),

                // Phone field

                TextFormField(

                  decoration: InputDecoration(labelText: "Phone"),

                  validator: (value) {

                    if (value == null || value.isEmpty) {

                      return "Phone is required";

                    }

                    if (value.length > 10) {

                      return "Phone number must be max 10 digits";

                    }

                    return null; // valid

                  },

                  onSaved: (val) {

                    phone = val;

                  },

                ),

                SizedBox(height: 24),

                // Submit button

                MaterialButton(

                  color: Colors.blue,

                  textColor: Colors.white,

                  child: Text("Submit"),

                  onPressed: () {

                    // 1. Validate fields

                    if (formKey.currentState!.validate()) {

                      // 2. Save values if valid

                      formKey.currentState!.save();

                      // 3. Use the values

                      print("Username: $userName");

                      print("Phone: $phone");

                    }

                  },

                ),

              ],

            ),

          ),

        ),

      ),

    );

  }

}

**🔹 Note on Saving TextField Values**

Right now, we are using **onSaved** inside a Form to save input values after validation.

* This is the **recommended way for forms** when you want to **validate first, then save**.

But remember: there are **many other ways** to get or store values from a TextField in Flutter:

**1. Using a Controller**

TextEditingController usernameController = TextEditingController();

TextField(controller: usernameController);

// Later

print(usernameController.text);

* Works **without a Form**.
* You can read the value at any time.

**2. Using onChanged**

String username = "";

TextField(

onChanged: (value) {

username = value; // updates every time user types

},

);

* Saves value **immediately when user types**.
* Good for live validation or live updates.

**3. Using onSubmitted**

TextField(

onSubmitted: (value) {

print("User typed: $value");

},

);

* Called **when user presses enter/done**.

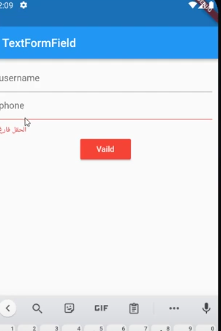
**🔹 Summary**

* onSaved → used with **Forms + validate** (best practice).
* Controller → direct access to value anytime.
* onChanged → updates live as user types.
* onSubmitted → reacts when user finishes typing.

💡 **Key idea:**  
We are **just learning the Form + onSaved method**, but Flutter gives you **many flexible ways** to work with user input.

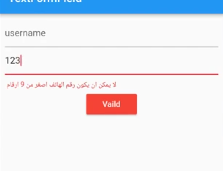
# TextFormField autovalidate

Autovalidate make a live checking, meaning : ok say with me this example :



Befoore clicking on the button, the validaton is there ,

When you start typing you will see the second condition



Let’s see the code ;

autovalidateMode: AutovalidateMode.always,

**NOTE :**

If we want to add this property to **one TextFormField** we add it inside it , but if we want to applid it to all Form **TextFormFileds**, we add it in Form .

import 'package:flutter/material.dart';

void main() {

  runApp(MyApp());

}

class MyApp extends StatelessWidget {

  // GlobalKey to control the Form

  final GlobalKey<FormState> formKey = GlobalKey<FormState>();

  // Variables to hold saved values

  String? userName;

  String? phone;

  @override

  Widget build(BuildContext context) {

    return MaterialApp(

      home: Scaffold(

        appBar: AppBar(title: Text("Form Validation Best Practice")),

        body: Padding(

          padding: const EdgeInsets.all(16.0),

          child: Form(

            key: formKey,

            child: Column(

              children: [

                // Username field

                TextFormField(

                  decoration: InputDecoration(labelText: "Username"),

                  validator: (value) {

                    if (value == null || value.isEmpty) {

                      return "Username is required";

                    }

                    if (value.length < 3) {

                      return "Username must be at least 3 characters";

                    }

                    return null; // valid

                  },

                  onSaved: (val) {

                    userName = val;

                  },

                ),

                SizedBox(height: 16),

                // Phone field

                TextFormField(

                  autovalidateMode: AutovalidateMode.,

                  decoration: InputDecoration(labelText: "Phone"),

                  validator: (value) {

                    if (value == null || value.isEmpty) {

                      return "Phone is required";

                    }

                    if (value.length > 10) {

                      return "Phone number must be max 10 digits";

                    }

                    return null; // valid

                  },

                  onSaved: (val) {

                    phone = val;

                  },

                ),

                SizedBox(height: 24),

                // Submit button

                MaterialButton(

                  color: Colors.blue,

                  textColor: Colors.white,

                  child: Text("Submit"),

                  onPressed: () {

                    // 1. Validate fields

                    if (formKey.currentState!.validate()) {

                      // 2. Save values if valid

                      formKey.currentState!.save();

                      // 3. Use the values

                      print("Username: $userName");

                      print("Phone: $phone");

                    }

                  },

                ),

              ],

            ),

          ),

        ),

      ),

    );

  }

}

# AppBar

|  |  |
| --- | --- |
| Property | What it does |
| title | Main text in the AppBar |
| centerTitle | Centers the title horizontally |
| leading | Widget at the start (usually an icon/button) |
| actions | Widgets at the end (usually icons/buttons) |
| backgroundColor | Changes background color |
| elevation | Adds shadow height |
| shadowColor | Changes shadow color |
| titleTextStyle | Customizes title font, color, weight, size |

import 'package:flutter/material.dart';

void main() {

  runApp(MyApp());

}

class MyApp extends StatelessWidget {

  @override

  Widget build(BuildContext context) {

    return MaterialApp(

      debugShowCheckedModeBanner: false, // removes the debug banner

      home: Scaffold(

        appBar: AppBar(

          title: Text("App Bar"),

          centerTitle: true,

          backgroundColor: Colors.deepOrange,

          leading: IconButton(onPressed: () {}, icon: Icon(Icons.home)),

          elevation: 16,

          shadowColor: Colors.black,

          titleTextStyle: TextStyle(

            color: Colors.white,

            fontSize: 20,

            fontWeight: FontWeight.bold,

          ),

          actions: [

            IconButton(onPressed: () {}, icon: Icon(Icons.list)),

            IconButton(onPressed: () {}, icon: Icon(Icons.search)),

            IconButton(onPressed: () {}, icon: Icon(Icons.settings)),

          ],

        ),

      ),

    );

  }

}

💡 **Key Tip:**  
AppBar is a **Material pre-styled widget**, which makes it **fast and consistent** across your app.  
You don’t have to style every detail manually — just use its properties!

# Drawer

When we just type drawer :

MaterialApp(

      home: Scaffold(

        appBar: AppBar(title: Text("App Bar")),

        drawer: Drawer(),

      ),

    );



Yeeaaah this is the drawer . when pressing the icon , the drawer open .

## endDrawer

is like drawer the only diffrence is :



So I think now you know why it name “end” .

## Drawer with no AppBar

The **Drawer** is a **side navigation panel** that slides from the left (or right) of the screen.  
Normally, we open it using the **hamburger menu (☰)** inside an **AppBar**.

But 👉 we can also **use it without AppBar** by controlling it with a **Scaffold key**.

**🔹 1. Create a Scaffold Key**

GlobalKey<ScaffoldState> scaffoldkey = GlobalKey();

* A **GlobalKey** lets us **control Scaffold state** (like openDrawer, closeDrawer).
* We assign this key to the Scaffold.

**🔹 2. Attach the Key to Scaffold**

Scaffold(

key: scaffoldkey,

drawer: Drawer(),

body: ...

)

* Now we can **manipulate the Scaffold** from anywhere inside.

**🔹 3. Open the Drawer Programmatically**

scaffoldkey.currentState!.openDrawer();

* This opens the Drawer **without needing AppBar**.
* Triggered by a button (or any widget).

**🔹 4. Example Button to Open Drawer**

MaterialButton(

color: Colors.red,

textColor: Colors.white,

elevation: 2,

onPressed: () {

scaffoldkey.currentState!.openDrawer();

},

child: Text("Open Drawer", style: TextStyle(fontSize: 25)),

),

* Pressing the button will **slide out the Drawer**.

**🔹 5. Drawer Widget**

drawer: Drawer(),

* Basic empty drawer.
* Can contain ListView, ListTile, UserAccountsDrawerHeader, etc.

import 'package:flutter/material.dart';

void main() {

  runApp(MyApp());

}

class MyApp extends StatelessWidget {

  GlobalKey<ScaffoldState> scaffoldkey = GlobalKey();

  @override

  Widget build(BuildContext context) {

    return MaterialApp(

      debugShowCheckedModeBanner: false, // removes the debug banner

      home: Scaffold(

        key: scaffoldkey,

        drawer: Drawer(),

        body: Container(

          padding: EdgeInsets.all(10),

          child: Column(

            mainAxisAlignment: MainAxisAlignment.center,

            children: [

              Center(

                child: MaterialButton(

                  color: Colors.red,

                  textColor: Colors.white,

                  elevation: 2,

                  onPressed: () {

                    scaffoldkey.currentState!.openDrawer();

                  },

                  child: Text("Open Drawer", style: TextStyle(fontSize: 25)),

                ),

              ),

            ],

          ),

        ),

      ),

    );

  }

}

**✅ Key Notes**

1. **Drawer normally needs AppBar**, but with a **ScaffoldKey**, you can open it manually.
2. Use scaffoldkey.currentState!.openDrawer() to open the Drawer.
3. scaffoldkey.currentState!.openEndDrawer() → opens Drawer on the right side (if endDrawer is defined).
4. You can place the **open button anywhere** (not just in AppBar).
5. Drawer is great for **app navigation** (profile, settings, logout, etc.).

**📌 Visual Flow**

* User taps **"Open Drawer"** button  
  ➡️ Flutter uses the **ScaffoldKey**  
  ➡️ Calls **openDrawer()**  
  ➡️ Drawer slides from the left

⚡ So this method is useful when:

* You don’t want an AppBar.
* You want **custom buttons or gestures** to open the Drawer.

#### Example of drawer

import 'package:flutter/material.dart';

void main() {

  runApp(MyApp());

}

class MyApp extends StatelessWidget {

  GlobalKey<ScaffoldState> scaffoldkey = GlobalKey();

  @override

  Widget build(BuildContext context) {

    return MaterialApp(

      debugShowCheckedModeBanner: false, // removes the debug banner

      home: Scaffold(

        key: scaffoldkey,

        drawer: Drawer(

          child: Container(

            padding: EdgeInsets.all(8),

            child: ListView(

              children: [

                Row(

                  children: [

                    Container(

                      width: 60,

                      height: 60,

                      child: ClipRRect(

                        borderRadius: BorderRadius.circular(60),

                        child: Image.asset("../images/img.jpg"),

                      ),

                    ),

                    Expanded(

                      child: ListTile(

                        title: Text("Bilal El Emrani"),

                        subtitle: Text("elemranistBilal4@gmail.com"),

                      ),

                    ),

                  ],

                ),

                ListTile(title: Text("Home"), leading: Icon(Icons.home)),

                ListTile(

                  title: Text("Account"),

                  leading: Icon(Icons.account\_box),

                ),

                ListTile(title: Text("Order"), leading: Icon(Icons.check\_box)),

                ListTile(

                  title: Text("About Us"),

                  leading: Icon(Icons.abc\_outlined),

                ),

                ListTile(

                  title: Text("Contact Us"),

                  leading: Icon(Icons.contact\_page\_outlined),

                ),

                ListTile(

                  title: Text("SignOut"),

                  leading: Icon(Icons.exit\_to\_app),

                ),

              ],

            ),

          ),

        ),

        body: Container(

          padding: EdgeInsets.all(10),

          child: Column(

            mainAxisAlignment: MainAxisAlignment.center,

            children: [

              Center(

                child: MaterialButton(

                  color: Colors.red,

                  textColor: Colors.white,

                  elevation: 2,

                  onPressed: () {

                    scaffoldkey.currentState!.openDrawer();

                  },

                  child: Text("Open Drawer", style: TextStyle(fontSize: 25)),

                ),

              ),

            ],

          ),

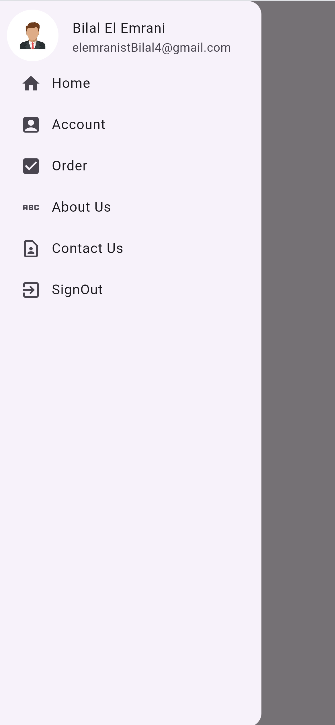
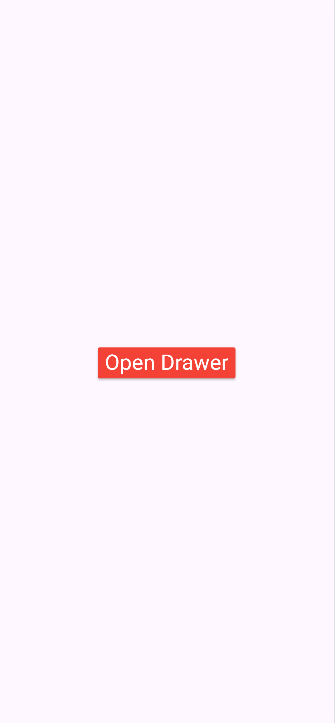
        ),

      ),

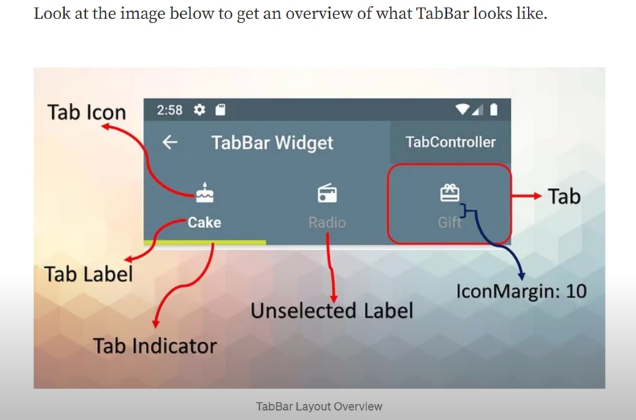
    );

  }

}

# tabbar

The tabbar is the lower side of an TabBar Widget .

import 'package:flutter/material.dart';

void main() {

  runApp(MyApp());

}

class MyApp extends StatelessWidget {

  GlobalKey<ScaffoldState> scaffoldkey = GlobalKey();

  @override

  Widget build(BuildContext context) {

    return MaterialApp(

      debugShowCheckedModeBanner: false, // removes the debug banner

      home: DefaultTabController(

        length: 3,

        child: Scaffold(

          key: scaffoldkey,

          appBar: AppBar(

            title: Text("Tabbar"),

            bottom: TabBar(

              tabs: [

                Tab(child: Text("Laptop")),

                Tab(child: Text("Mobile")),

                Tab(child: Text("Pc")),

              ],

            ),

          ),

          body: Container(

            padding: EdgeInsets.all(10),

            child: TabBarView(

              children: [

                Text("Laptop Page"),

                Text("Mobile Page"),

                Text("Pc Page"),

              ],

            ),

          ),

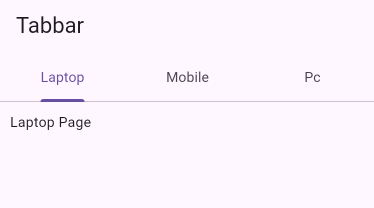
        ),

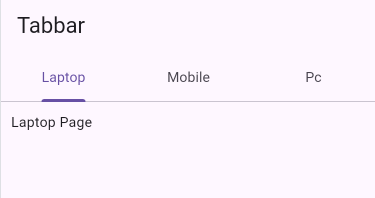
      ),

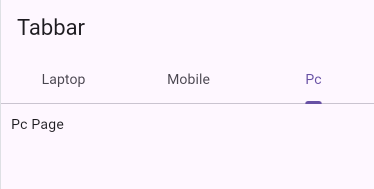
    );

  }

}







**🔹 What is TabBar in Flutter?**

* **TabBar** is a widget that creates a horizontal row of tabs (like Laptop / Mobile / PC).
* It is always used with **TabBarView** → to display the content for each tab.
* You also need a **DefaultTabController** → to manage which tab is selected.

**🔹 Your Code Explained**

1. **DefaultTabController**
2. DefaultTabController(
3. length: 3, // number of tabs
4. child: Scaffold(

👉 This is the “brain” that keeps track of which tab is selected.

1. **AppBar with TabBar**
2. appBar: AppBar(
3. title: Text("Tabbar"),
4. bottom: TabBar(
5. tabs: [
6. Tab(child: Text("Laptop")),
7. Tab(child: Text("Mobile")),
8. Tab(child: Text("Pc")),
9. ],
10. ),
11. ),

👉 Inside the AppBar, you add a **TabBar** in the bottom property.  
👉 Each Tab has text (“Laptop”, “Mobile”, “PC”).

1. **TabBarView**
2. body: Container(
3. padding: EdgeInsets.all(10),
4. child: TabBarView(
5. children: [
6. Text("Laptop Page"),
7. Text("Mobile Page"),
8. Text("Pc Page"),
9. ],
10. ),
11. ),

👉 TabBarView shows the **page content** when a tab is selected.  
👉 Order must match with the tabs.

**🔹 TabBar Structure (Simple Formula)**

DefaultTabController(

length: number\_of\_tabs,

child: Scaffold(

appBar: AppBar(

bottom: TabBar(

tabs: [ Tab(), Tab(), ... ],

),

),

body: TabBarView(

children: [ Widget(), Widget(), ... ],

),

),

);

**🔹 Notes to Add in Your Word File**

* TabBar needs **DefaultTabController** to work.
* TabBar goes inside AppBar.bottom.
* TabBarView shows the content for each tab.
* **Order of tabs and TabBarView must match**.
* You can add icons, text, or both inside a Tab.

👉 Example with icon + text:

Tab(icon: Icon(Icons.laptop), text: "Laptop"),

Lesson from ai :

**📝 Lesson: TabBar in Flutter**

**🔹 DefaultTabController**

* Acts like the **manager** of the tabs.
* It knows **which tab is selected** and controls the switching.
* Without it, the TabBar and TabBarView won’t work together.

**🔹 TabBar**

* A horizontal bar that shows the **tabs** (buttons).
* Usually placed in the **AppBar** (at the bottom).
* Each tab can have **text**, **icon**, or both.
* Example idea: “Laptop | Mobile | PC”.

**🔹 Tab**

* Represents a **single clickable item** inside the TabBar.
* Can display a **label (text)**, an **icon**, or both.
* The order of tabs must match the order of content in TabBarView.

**🔹 TabBarView**

* Displays the **content** for each tab.
* Works together with TabBar → when you click a tab, this changes automatically.
* Each child inside it is the **page/screen** for the corresponding tab.

**🔹 Scaffold + AppBar**

* **Scaffold**: provides the structure of the screen.
* **AppBar**: holds the TabBar in its bottom section.
* Together they create a familiar **top navigation bar**.

**🔹 Flow of Work**

1. **DefaultTabController** manages the tabs.
2. **TabBar** shows the clickable tabs.
3. **TabBarView** shows the content of the selected tab.
4. Both are linked by the same controller.

👉 In short:

* **TabBar** = menu of choices.
* **Tab** = each choice.
* **TabBarView** = the content shown when a choice is selected.
* **DefaultTabController** = the link that keeps everything synchronized.

## 🎨 TabBar Styling Properties

|  |  |  |
| --- | --- | --- |
| Property | What it Does | Example Effect |
| indicatorColor | Changes the color of the underline below the selected tab. | Blue line under active tab. |
| indicatorWeight | Sets the thickness of the indicator line. | Thicker or thinner underline. |
| indicatorPadding | Adds space around the indicator. | Indicator doesn’t touch text edges. |
| indicator | Fully customize the indicator with a shape/decoration. | Rounded box, gradient background. |
| labelColor | Color of the selected tab’s text/icon. | Selected tab text turns white. |
| unselectedLabelColor | Color of unselected tabs’ text/icon. | Inactive tabs are grey. |
| labelStyle | Text style of the selected tab. | Bigger font for active tab. |
| unselectedLabelStyle | Text style of unselected tabs. | Smaller font for inactive tabs. |
| isScrollable | Makes TabBar scrollable instead of fixed. | Useful when you have many tabs. |
| tabAlignment | Controls alignment of tabs inside TabBar. | Center, start, or stretched. |

import 'package:flutter/material.dart';

void main() {

  runApp(MyApp());

}

class MyApp extends StatelessWidget {

  @override

  Widget build(BuildContext context) {

    return MaterialApp(

      debugShowCheckedModeBanner: false, // removes the debug banner

      home: DefaultTabController(

        length: 3,

        initialIndex: 2,

        child: Scaffold(

          appBar: AppBar(

            title: Text("Tabbar"),

            bottom: TabBar(

              indicatorColor: Colors.red,

              indicatorWeight: 10,

              labelColor: Colors.orangeAccent,

              unselectedLabelColor: Colors.blue,

              unselectedLabelStyle: TextStyle(fontSize: 10),

              labelStyle: TextStyle(fontSize: 18),

              tabs: [

                // it's important to do text if you do icon and not child

                Tab(

                  icon: Icon(Icons.laptop),

                  text: "Laptop",

                  iconMargin: EdgeInsets.all(10),

                ),

                Tab(icon: Icon(Icons.mobile\_friendly\_outlined), text: "Mobile"),

                Tab(icon: Icon(Icons.tv), text: "PC"),

              ],

            ),

          ),

          body: Container(

            padding: EdgeInsets.all(10),

            child: TabBarView(

              children: [

                Text("Laptop Page"),

                Text("Mobile Page"),

                Text("Pc Page"),

              ],

            ),

          ),

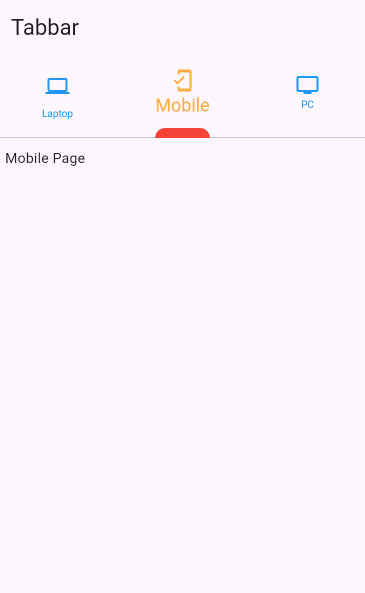
        ),

      ),

    );

  }

}



**📝 Notes for TabBar Lesson**

* **DefaultTabController**
  + Manages which tab is selected.
  + length: 3 → number of tabs.
  + initialIndex: 2 → starts with the **3rd tab (PC)** selected.
* **AppBar + TabBar**
  + AppBar holds the **TabBar** in its bottom property.
  + TabBar defines **look + tabs**.

**🔹 Styling Properties Used**

* indicatorColor: Colors.red → underline color for the active tab.
* indicatorWeight: 10 → thickness of the underline.
* labelColor: Colors.orangeAccent → color of selected tab text/icon.
* unselectedLabelColor: Colors.blue → color of inactive tabs.
* labelStyle: TextStyle(fontSize: 18) → style of the selected tab text.
* unselectedLabelStyle: TextStyle(fontSize: 10) → style of inactive tabs text.

**🔹 Tab Widget**

* Each **Tab** can have:
  + **icon only**
  + **text only**
  + **icon + text** (like Laptop, Mobile, PC).
* iconMargin → adds space around the icon (here: 10px).
* Order of tabs must match the order of content in **TabBarView**.

**🔹 TabBarView**

* Holds the **pages** for each tab.
* Children order = Laptop → Mobile → PC (must match TabBar).
* When switching tabs, the content updates automatically.

**🔹 Flow**

1. User taps a tab.
2. **DefaultTabController** updates the selected index.
3. **TabBar** highlights the active tab with styling.
4. **TabBarView** shows the correct content.

👉 In short:

* **Controller** = brain.
* **TabBar** = navigation menu.
* **Tab** = button (icon/text).
* **TabBarView** = the content for each tab.
* **Styling** lets you customize colors, size, and indicator.

# BottomNavigationBar

**🔹 BottomNavigationBar**

* It’s a **widget at the bottom of the app** that lets users **switch between different sections (pages)** of your app.
* Each item in the bar is represented by:
  + An **icon** (e.g., Icons.home)
  + A **label** (e.g., "Home")
* You can **customize**:
  + backgroundColor
  + selectedItemColor
  + unselectedItemColor
  + Font sizes & styles of labels

👉 In your code:

bottomNavigationBar: BottomNavigationBar(

currentIndex: selectedIndex, // which item is active

onTap: (value) {

setState(() {

selectedIndex = value; // change active index

});

},

items: const [

BottomNavigationBarItem(icon: Icon(Icons.home), label: "Home"),

BottomNavigationBarItem(icon: Icon(Icons.settings), label: "Settings"),

],

),

* When the user taps an item, onTap runs and updates selectedIndex.
* setState() tells Flutter to rebuild the UI with the **new active page**.

**🔹 The List of Widgets**

List<Widget> listWidget = [

Text("HOME : Page 1", style: TextStyle(fontSize: 30)),

Text("SETTINGS : Page 2", style: TextStyle(fontSize: 30)),

];

* This is just a **list** holding two widgets (texts here, but could be full pages like HomePage() or SettingsPage()).
* Why a list?
  + Instead of writing a long if/else for each page, you just use the **index** to pick the right widget.
  + Example: if selectedIndex = 0, we show the first widget (Home).  
    If selectedIndex = 1, we show the second (Settings).

👉 The display happens here:

body: Container(child: listWidget.elementAt(selectedIndex)),

* elementAt(selectedIndex) picks the widget from the list based on the active bottom nav index.

**🔹 The Logic in Simple Steps**

1. Start with selectedIndex = 0 → show the first page (Home).
2. User taps on another item in the BottomNavigationBar.
3. onTap(value) updates selectedIndex = value.
4. setState() rebuilds the widget tree.
5. listWidget.elementAt(selectedIndex) loads the correct page from the list.

✅ This makes the code:

* Shorter
* Cleaner
* Easy to extend (just add more items to the list & BottomNavigationBar).

import 'package:flutter/material.dart';

void main() {

  runApp(MyApp());

}

class MyApp extends StatefulWidget {

  @override

  State<MyApp> createState() => \_MyAppState();

}

class \_MyAppState extends State<MyApp> {

  int selectedIndex = 0;

  List<Widget> listWidget = [

    Text("HOME : Page 1", style: TextStyle(fontSize: 30)),

    Text("SETTINGS : Page 2", style: TextStyle(fontSize: 30)),

  ];

  @override

  Widget build(BuildContext context) {

    return MaterialApp(

      debugShowCheckedModeBanner: false, // removes the debug banner

      home: Scaffold(

        bottomNavigationBar: BottomNavigationBar(

          onTap: (value) {

            setState(() {

              selectedIndex = value; // rebuild with new index

            });

          },

          currentIndex: selectedIndex,

          // shared properties for label or icon

          backgroundColor: const Color.fromARGB(221, 24, 16, 26),

          selectedItemColor: const Color.fromARGB(255, 187, 151, 255),

          unselectedItemColor: Colors.white,

          selectedFontSize: 20,

          unselectedFontSize: 14,

          selectedLabelStyle: TextStyle(fontWeight: FontWeight.bold),

          items: const [

            BottomNavigationBarItem(icon: Icon(Icons.home), label: "Home"),

            BottomNavigationBarItem(

              icon: Icon(Icons.settings),

              label: "Settings",

            ),

          ],

        ),

        body: Container(child: listWidget.elementAt(selectedIndex)),

      ),

    );

  }

}



## 🔹 Table 1: BottomNavigationBar Properties

|  |  |  |
| --- | --- | --- |
| Property | Type | Description |
| items | List<BottomNavigationBarItem> | The buttons (tabs) displayed in the bar. |
| currentIndex | int | The active item index (which page is selected). |
| onTap | Function(int) | Callback when a tab is tapped. Gives the index. |
| backgroundColor | Color | The background color of the bar. |
| selectedItemColor | Color | Color of the selected item (icon + label). |
| unselectedItemColor | Color | Color of items that are not selected. |
| selectedFontSize | double | Font size for the selected item’s label. |
| unselectedFontSize | double | Font size for unselected labels. |
| selectedLabelStyle | TextStyle | Full text style for selected labels (bold, italic, etc.). |
| unselectedLabelStyle | TextStyle | Style for unselected labels. |
| type | BottomNavigationBarType | Layout style: fixed (default, all visible) or shifting (icons shift when selected). |
| elevation | double | Shadow below the bar. |
| iconSize | double | Size of all icons inside the bar. |
| showSelectedLabels | bool | Show/hide text label of the selected item. |
| showUnselectedLabels | bool | Show/hide text labels of unselected items. |

## 🔹 Table 2: BottomNavigationBarItem Properties

|  |  |  |
| --- | --- | --- |
| Property | Type | Description |
| icon | Widget | The default icon (not selected state). |
| activeIcon | Widget | Icon shown when the item is selected (optional). |
| label | String | Text shown under the icon. |
| backgroundColor | Color | Background color **when using shifting type**. |
| tooltip | String? | Message shown on long press (optional). |

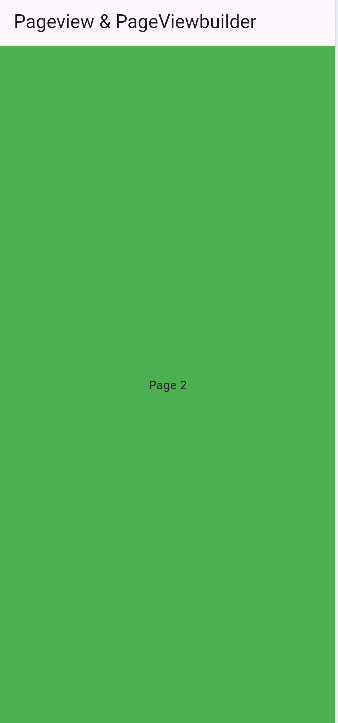
# PageView & Pagebuilder

## PageView

### What is PageView ?

* A **scrollable list of pages** (like a book or carousel).
* You can swipe **horizontally or vertically** between pages.
* Each “Page” is usually a widget (like container, Card , or even a full screen )

#### Example :



import 'package:flutter/material.dart';

void main() {

  runApp(MyApp());

}

class MyApp extends StatefulWidget {

  @override

  State<MyApp> createState() => \_MyAppState();

}

class \_MyAppState extends State<MyApp> {

  @override

  Widget build(BuildContext context) {

    return MaterialApp(

      debugShowCheckedModeBanner: false, // removes the debug banner

      home: Scaffold(

        appBar: AppBar(title: Text("Pageview & PageViewbuilder")),

        body: Container(

          height: 1000,

          child: PageView(

            children: [

              Container(

                color: Colors.red,

                child: Center(child: Text("Page 1")),

              ),

              Container(

                color: Colors.green,

                child: Center(child: Text("Page 2")),

              ),

              Container(

                color: Colors.blue,

                child: Center(child: Text("Page 3")),

              ),

            ],

          ),

        ),

      ),

    );

  }

}

👉 This creates **3 swipeable pages**.

### 🔹 Properties of PageView

|  |  |
| --- | --- |
| Property | Description |
| children | List of widgets (the pages). |
| scrollDirection | Axis.horizontal (default) or Axis.vertical. |
| controller | Controls which page to show, jump/animate to a page. |
| onPageChanged | Callback when page changes. |
| pageSnapping | If false, pages don’t snap, can stop halfway. |
| physics | Controls scroll physics (e.g., NeverScrollableScrollPhysics to disable swiping). |

#### Example of using properties

Container(

          height: 1000,

          child: PageView(

            scrollDirection: Axis.vertical,

            onPageChanged: (value) {

              print(value); // return index of the child.

            },

            // pageSnapping: true, // momkin tb9a wa9f bin 2 les page :: jarb

            // physics: NeverScrollableScrollPhysics(),

            children: [

              Container(

                color: Colors.red,

                child: Center(child: Text("Page 1")),

              ),

              Container(

                color: Colors.green,

                child: Center(child: Text("Page 2")),

              ),

              Container(

                color: Colors.blue,

                child: Center(child: Text("Page 3")),

              ),

            ],

          ),

        ),

Other great example :

import 'package:flutter/material.dart';

void main() {

  runApp(MyApp());

}

class MyApp extends StatefulWidget {

  @override

  State<MyApp> createState() => \_MyAppState();

}

class \_MyAppState extends State<MyApp> {

  int page = 0;

  PageController \_controller = PageController(initialPage: 0);

  @override

  Widget build(BuildContext context) {

    return MaterialApp(

      debugShowCheckedModeBanner: false,

      home: Scaffold(

        appBar: AppBar(title: Text("PageView & PageView.builder")),

        body: PageView(

          controller: \_controller,

          onPageChanged: (value) {

            setState(() {

              page = value;

            });

          },

          children: [

            Container(

              color: Colors.red,

              child: Center(child: Text("Page 1")),

            ),

            Container(

              color: Colors.green,

              child: Center(child: Text("Page 2")),

            ),

            Container(

              color: Colors.blue,

              child: Center(child: Text("Page 3")),

            ),

          ],

        ),

        floatingActionButton: FloatingActionButton(

          onPressed: () {

            setState(() {

              if (page < 2) {

                page++;

              } else {

                page = 0;

              }

            });

            \_controller.animateToPage(

              page,

              duration: Duration(milliseconds: 1000),

              curve: Curves.easeInOut,

            );

          },

          child: Icon(page == 2 ? Icons.arrow\_back : Icons.arrow\_forward),

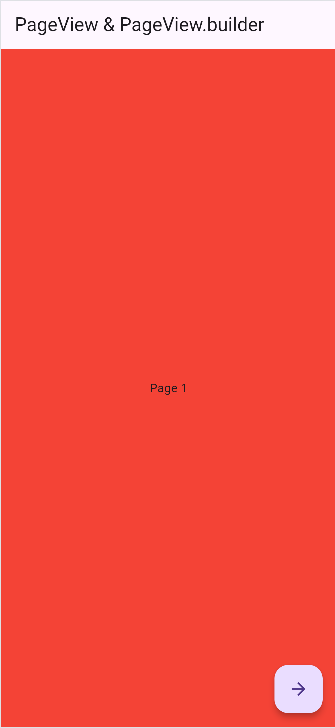
        ),

      ),

    );

  }

}



## PageView.builder

let’s move from PageView (where you manually list widgets) to **PageView.builder**, which is better when you have many pages or want to generate them dynamically.

**🔹 Key Difference**

* PageView: You give it a list of **children** directly.
* PageView.builder: You give it an **itemBuilder** function, and it builds pages only when needed (more efficient).

import 'package:flutter/material.dart';

// launcher import

import 'package:url\_launcher/url\_launcher.dart';

void main() {

  runApp(MyApp());

}

// Moroccan Cities App

class MyApp extends StatefulWidget {

  @override

  State<MyApp> createState() => \_MyAppState();

}

class \_MyAppState extends State<MyApp> {

  List<String> cities = [

    "Ait Ben Haddou",

    "Chefchaouen",

    "Essaouira",

    "Tangier",

    "Marrakech",

    "Dakhla",

  ];

  // Fixed paths: remove ../

  List<String> images = [

    "assets/cities/ben.jpeg",

    "assets/cities/chef.webp",

    "assets/cities/essa.jpeg",

    "assets/cities/tanger.webp",

    "assets/cities/mara.jpg",

    "assets/cities/dakhla.jpeg",

  ];

  List<String> descriptions = [

    "Ait Ben Haddou is a historic ksar located along the former caravan route between the Sahara and Marrakech in Morocco. It is a UNESCO World Heritage site known for its stunning earthen architecture and has been featured in numerous films and television series.",

    "Chefchaouen, also known as the Blue City, is a picturesque town in the Rif Mountains of northern Morocco. It is famous for its striking blue-painted buildings, narrow winding streets, and vibrant culture, making it a popular destination for tourists and photographers.",

    "Essaouira is a coastal city in western Morocco known for its historic medina, beautiful beaches, and vibrant arts scene. The city features a blend of Portuguese, French, and Berber architecture and is famous for its seafood, windsurfing opportunities, and annual music festival.",

    "Tangier is a port city in northern Morocco that has long been a cultural crossroads due to its strategic location at the entrance to the Mediterranean Sea. The city boasts a rich history, diverse architecture, bustling markets, and a vibrant arts scene.",

    "Marrakech, often referred to as the 'Red City' due to its distinctive red sandstone buildings, is one of Morocco's most famous cities. It is renowned for its historic medina, lively souks (markets), beautiful palaces, and gardens, as well as its vibrant nightlife.",

    "Dakhla is a city located on a narrow peninsula in Western Sahara, known for its stunning natural beauty and outdoor activities. It is a popular destination for kite surfing and windsurfing due to its consistent winds and warm climate. Dakhla also offers pristine beaches and unique desert landscapes.",

  ];

  @override

  Widget build(BuildContext context) {

    return MaterialApp(

      debugShowCheckedModeBanner: false,

      home: Scaffold(

        appBar: AppBar(title: Text("Moroccan Cities")),

        body: PageView.builder(

          itemCount: cities.length,

          itemBuilder: (context, i) {

            return SingleChildScrollView(

              child: Column(

                crossAxisAlignment: CrossAxisAlignment.center,

                children: [

                  Image.asset(images[i], height: 300, fit: BoxFit.cover),

                  SizedBox(height: 10),

                  Text(

                    cities[i],

                    style: TextStyle(fontSize: 24, fontWeight: FontWeight.bold),

                  ),

                  SizedBox(height: 10),

                  Padding(

                    padding: const EdgeInsets.symmetric(horizontal: 16.0),

                    child: Text(

                      descriptions[i],

                      textAlign: TextAlign.center,

                      style: TextStyle(fontSize: 16),

                    ),

                  ),

                ],

              ),

            );

          },

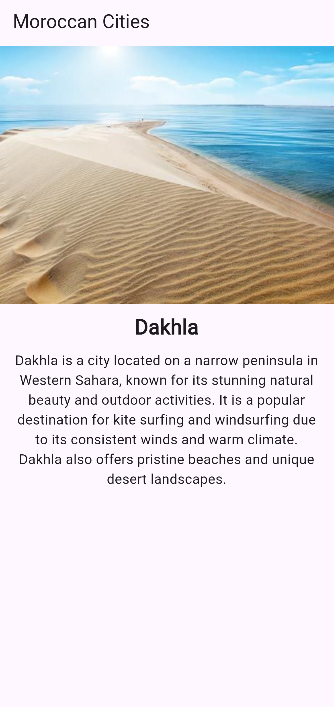
        ),

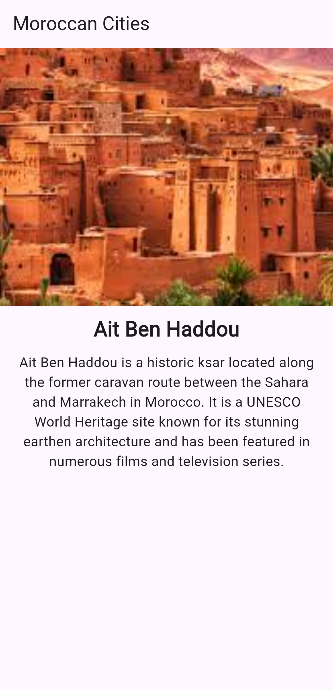
      ),

    );

  }

}





# Custom Widget Shortcut

Imagine with me, that you have to create card of employees ,

Container(

          padding: EdgeInsets.all(10),

          child: Column(

            children: [

              Card(

                child: ListTile(

                  title: Text("Bilal"),

                  subtitle: Text("elemranibilal45@gmail.com"),

                  trailing: Text("2025-10-04"),

                ),

              ),

              Card(

                child: ListTile(

                  title: Text("Ahmad"),

                  subtitle: Text("eleahmidl45@gmail.com"),

                  trailing: Text("2025-10-04"),

                ),

              ),

            ],

          ),

        ),

Okay but if you have 3 more , aah ok you will copy paste and do them , eeemm great

But if you have 100 , ooooh here you will face a problem .

import 'package:flutter/material.dart';

// launcher import

import 'package:url\_launcher/url\_launcher.dart';

void main() {

  runApp(MyApp());

}

// Moroccan Cities App

class MyApp extends StatefulWidget {

  @override

  State<MyApp> createState() => \_MyAppState();

}

class \_MyAppState extends State<MyApp> {

  List<String> cities = [

    "Ait Ben Haddou",

    "Chefchaouen",

    "Essaouira",

    "Tangier",

    "Marrakech",

    "Dakhla",

  ];

  // Fixed paths: remove ../

  List<String> images = [

    "assets/cities/ben.jpeg",

    "assets/cities/chef.webp",

    "assets/cities/essa.jpeg",

    "assets/cities/tanger.webp",

    "assets/cities/mara.jpg",

    "assets/cities/dakhla.jpeg",

  ];

  List<String> descriptions = [

    "Ait Ben Haddou is a historic ksar located along the former caravan route between the Sahara and Marrakech in Morocco. It is a UNESCO World Heritage site known for its stunning earthen architecture and has been featured in numerous films and television series.",

    "Chefchaouen, also known as the Blue City, is a picturesque town in the Rif Mountains of northern Morocco. It is famous for its striking blue-painted buildings, narrow winding streets, and vibrant culture, making it a popular destination for tourists and photographers.",

    "Essaouira is a coastal city in western Morocco known for its historic medina, beautiful beaches, and vibrant arts scene. The city features a blend of Portuguese, French, and Berber architecture and is famous for its seafood, windsurfing opportunities, and annual music festival.",

    "Tangier is a port city in northern Morocco that has long been a cultural crossroads due to its strategic location at the entrance to the Mediterranean Sea. The city boasts a rich history, diverse architecture, bustling markets, and a vibrant arts scene.",

    "Marrakech, often referred to as the 'Red City' due to its distinctive red sandstone buildings, is one of Morocco's most famous cities. It is renowned for its historic medina, lively souks (markets), beautiful palaces, and gardens, as well as its vibrant nightlife.",

    "Dakhla is a city located on a narrow peninsula in Western Sahara, known for its stunning natural beauty and outdoor activities. It is a popular destination for kite surfing and windsurfing due to its consistent winds and warm climate. Dakhla also offers pristine beaches and unique desert landscapes.",

  ];

  @override

  Widget build(BuildContext context) {

    return MaterialApp(

      debugShowCheckedModeBanner: false,

      home: Scaffold(

        appBar: AppBar(title: Text("Moroccan Cities")),

        body: Container(

          padding: EdgeInsets.all(10),

          child: Column(

            children: [

              customListTile(

                name: "Bilal",

                email: "elemranibilal45@gmail.com",

                date: "2025-10-04",

              ),

              customListTile(

                name: "Ahmad",

                email: "eleahmidl45@gmail.com",

                date: "2025-10-04",

              ),

              customListTile(

                name: "Sara",

                email: "sara@example.com",

                date: "2025-10-04",

              ),

              customListTile(

                name: "Ali",

                email: "ali@example.com",

                date: "2025-10-04",

              ),

              customListTile(

                name: "Fatima",

                email: "fatima@example.com",

                date: "2025-10-04",

              ),

              customListTile(

                name: "Omar",

                email: "omar@example.com",

                date: "2025-10-04",

              ),

            ],

          ),

        ),

      ),

    );

  }

}

class customListTile extends StatelessWidget {

  final String name;

  final String email;

  final String date;

  // super key means calling the superclass constructor

  // meaning it allows us to pass parameters to the parent class

  const customListTile({

    super.key,

    required this.name,

    required this.email,

    required this.date,

  });

  @override

  Widget build(BuildContext context) {

    return Card(

      child: ListTile(

        title: Text(name),

        subtitle: Text(email),

        trailing: Text(date),

      ),

    );

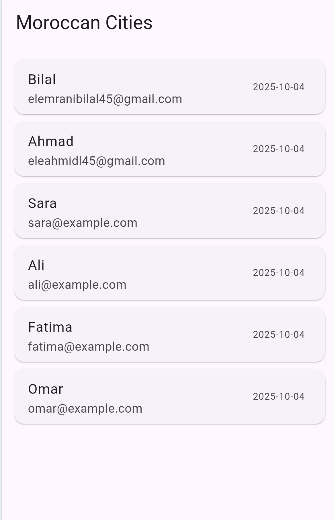
  }

}

👉 Instead of writing the same **Card → ListTile** structure over and over again (copy-paste), you make a **shortcut widget** (like your customListTile).

That way:

* You just call it with **different data** (name, email, date).
* If you ever want to change the style, you update **one place only** (inside the custom widget).
* Works perfectly for **100, 1000, or even more** employees.



And now if we want to change decoration, I don’t need to change for each card , all that I have to do is to modify the statelesswidget that I create .

Like this :

class customListTile extends StatelessWidget {

  final String name;

  final String email;

  final String date;

  // super key means calling the superclass constructor

  // meaning it allows us to pass parameters to the parent class

  const customListTile({

    super.key,

    required this.name,

    required this.email,

    required this.date,

  });

  @override

  Widget build(BuildContext context) {

    return Card(

      child: ListTile(

        leading: CircleAvatar(

          backgroundColor: Colors.blue,

          child: Text(name[0], style: TextStyle(color: Colors.white)),

        ),

        title: Text(name, style: TextStyle(fontWeight: FontWeight.bold)),

        subtitle: Text(email, style: TextStyle(color: Colors.grey[600])),

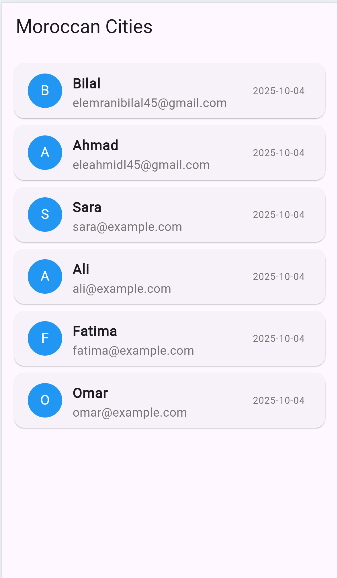
        trailing: Text(date, style: TextStyle(color: Colors.grey[600])),

      ),

    );

  }

}

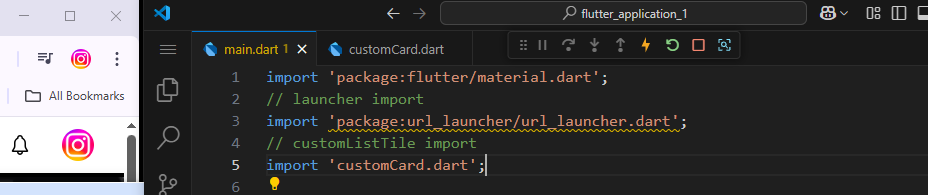


# Import

Before beginning in this chapter .

In the last example , imaging that you create the statelesswidget in anther dart file, and you need to call it in the main dart file .

Let’s do its .



As you see in this picture ,

Here is it the customCard.dart file :

import 'package:flutter/material.dart';

class customListTile extends StatelessWidget {

  final String name;

  final String email;

  final String date;

  // super key means calling the superclass constructor

  // meaning it allows us to pass parameters to the parent class

  const customListTile({

    super.key,

    required this.name,

    required this.email,

    required this.date,

  });

  @override

  Widget build(BuildContext context) {

    return Card(

      child: ListTile(

        leading: CircleAvatar(

          backgroundColor: const Color.fromARGB(255, 2, 253, 232),

          child: Text(name[0], style: TextStyle(color: Colors.white)),

        ),

        title: Text(name, style: TextStyle(fontWeight: FontWeight.bold)),

        subtitle: Text(email, style: TextStyle(color: Colors.grey[600])),

        trailing: Text(date, style: TextStyle(color: Colors.grey[600])),

      ),

    );

  }

}

Soo all that I have to do is to **import** it in the main file .

**📘 Lesson: How to Import Files in Flutter**

**🔹 1. What is an Import?**

An **import** allows you to use code from another file.  
For example, if you create a new file for a widget (like customCard.dart), you must import it into your main file before using it.

**🔹 2. Types of Imports**

**✅ A. Package Import (recommended for real projects)**

Used when you want to import files using your **project name** as the package root.

**Example:**

import 'package:flutter\_application\_1/customCard.dart';

🧠 **Important Note:**  
The word **flutter\_application\_1** must be **exactly the same name** written in your pubspec.yaml file at the top:

name: flutter\_application\_1

If your project name in pubspec.yaml is different (for example moroccan\_cities), then your import must also change:

import 'package:moroccan\_cities/customCard.dart';

If they don’t match, Flutter will show:

Target of URI doesn’t exist

**✅ B. Relative Import (simpler for small projects)**

You can also import a file **relative** to your current location (no need for project name).

**Example:**

import 'customCard.dart';

🟡 This works well for small apps or quick testing, but **package imports** are cleaner when your project grows.

**🔹 3. Folder Structure Example**

Make sure your files are placed like this:

lib/

┣ customCard.dart

┗ main.dart

# Navigator

## 🔹 1. What is Navigator?

In Flutter, the **Navigator** helps you **move between screens (pages)**.  
Each page is a **widget** — usually a StatelessWidget or StatefulWidget with a Scaffold.

Think of it like:

You have a stack of pages → when you go to a new page, Flutter **pushes** it on top of the stack.  
When you go back, Flutter **pops** it off.

## 🔹 2. Folder Structure Example

lib/

┣ main.dart

┣ homePage.dart

┗ aboutPage.dart

**🔹 3. Code Example**

**🟢 main.dart**

import 'package:flutter/material.dart';

import 'package:flutter\_application\_1/homePage.dart';

void main() {

  runApp(MyApp());

}

// Moroccan Cities App

class MyApp extends StatefulWidget {

  @override

  State<MyApp> createState() => \_MyAppState();

}

class \_MyAppState extends State<MyApp> {

  @override

  Widget build(BuildContext context) {

    return MaterialApp(debugShowCheckedModeBanner: false, home: Homepage(), );

  }

}

🟢 homePage.dart

import 'package:flutter/material.dart';

import 'package:flutter\_application\_1/aboutPage.dart';

class Homepage extends StatelessWidget {

  const Homepage({super.key});

  @override

  Widget build(BuildContext context) {

    return Scaffold(

      appBar: AppBar(title: Text("Home Page")),

      body: ListView(

        children: [

          Center(child: Text("Home Page")),

          Container(

            margin: EdgeInsets.all(20),

            child: MaterialButton(

              color: Colors.blue,

              textColor: Colors.white,

              onPressed: () {

                Navigator.of(

                  context,

                ).push(MaterialPageRoute(builder: (context) => AboutPage()));

              },

              child: Text("Go to About Page"),

            ),

          ),

        ],

      ),

    );

  }

}

🟢 aboutPage.dart

import 'package:flutter/material.dart';

import 'package:flutter\_application\_1/homePage.dart';

class AboutPage extends StatelessWidget {

  const AboutPage({super.key});

  @override

  Widget build(BuildContext context) {

    return Scaffold(

      appBar: AppBar(title: Text("About Page")),

      body: ListView(

        children: [

          Center(child: Text("About Page")),

          Container(

            margin: EdgeInsets.all(20),

            child: MaterialButton(

              color: Colors.blue,

              textColor: Colors.white,

              onPressed: () {

                Navigator.of(

                  context,

                ).push(MaterialPageRoute(builder: (context) => Homepage()));

              },

              child: Text("Go to Home Page"),

            ),

          ),

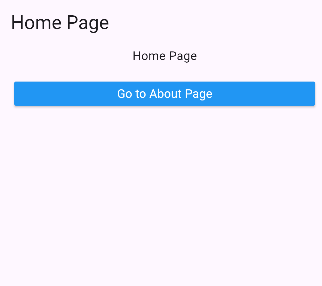
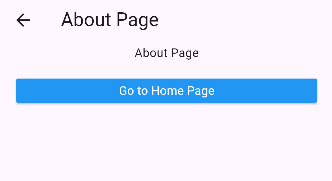
        ],

      ),

    );

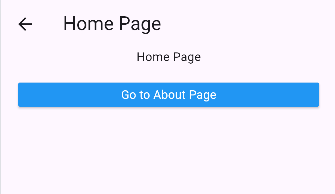
  }

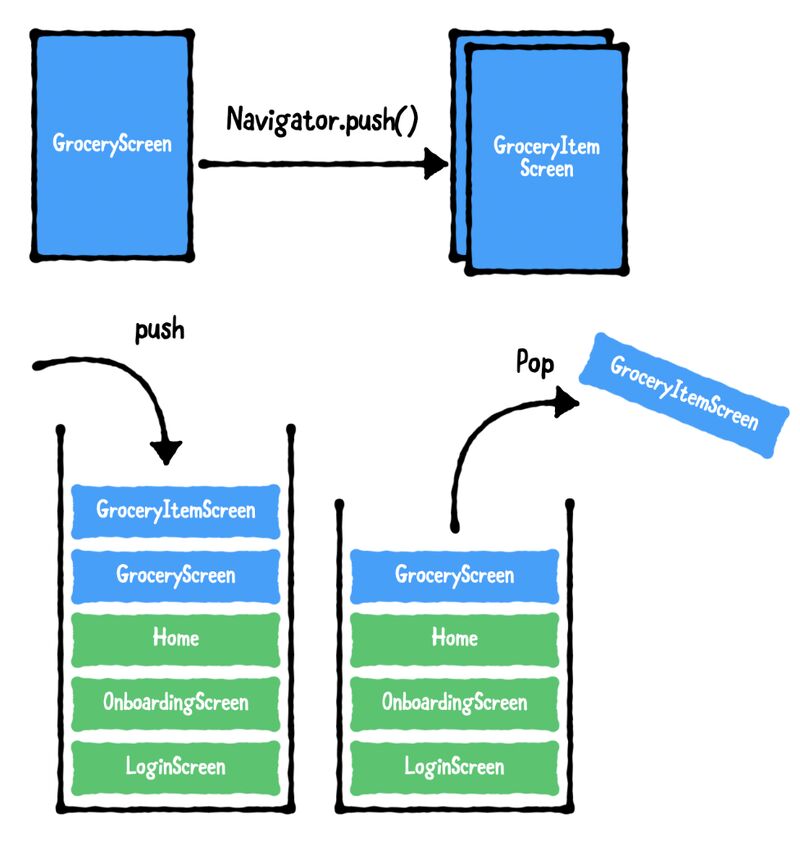
}

But if you was in about page and you clicked **“Go To Home Page”**

You will see :



I think it clear it show like home page is onther page , and not the previous page , return to the code the compare with table methods you will know why . 

## 🔹 4. Navigator Methods

|  |  |  |
| --- | --- | --- |
| Method | Description | Example |
| push() | Go **to** another page | Navigator.of(context).push(MaterialPageRoute(builder: (context) => AboutPage())); |
| pop() | Go **back** to the previous page | Navigator.of(context).pop(); |
| pushReplacement() | Replace current page with another one | Navigator.of(context).pushReplacement(MaterialPageRoute(builder: (context) => AboutPage())); |

## 🔹 5. How it Works (Analogy)

Think of **Navigator** as a **stack of cards**:

* When you open a new page → 🟩 you **push** a new card on top.
* When you go back → 🔙 you **pop** the top card.

## ✅ Summary

|  |  |
| --- | --- |
| Concept | Description |
| Navigator | Manages page transitions |
| push() | Opens a new page |
| pop() | Closes the current page and returns to the previous one |
| pushReplacement() | Replaces current page with new one |
| Pages | Are widgets (StatelessWidget or StatefulWidget) |
| Imports | You must import each page file before using it |

**Note :** the widget in the home is the first one that will be shown in our app .

 MaterialApp(debugShowCheckedModeBanner: false, home: Homepage(), );

# Push & pushReplacement

So now to well understand, let return to the last example :

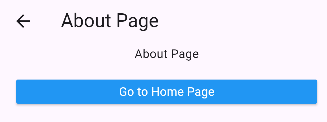
We had 2 dart files : home and about

Imagine it like a stack and you see what above meaning you see the element that is in the top .

When just your app started , you will see the root home , if you press about other card will be added to stack

Home (root)

About



Let do some other steps :

Click on “Go to Home Page “ and then “Go to About Page”.

Home (root)

About

Home

About

To Make sure that that is the reel system , click on the back icon , you need 3 back clicks to return to home root , which does not have the back icon .

Make it and then return , you don’t a code just you need the app created earlier .

But But But **PushReplecment does not add onther page to the stack , it delete the first one and add the new one so it repelace it .**

So let change the code and try our app .

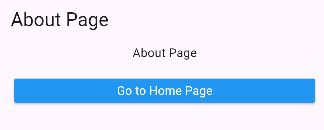
Home

When you click “Go to About Page”.

About

So the about become root hhhh .

There is no home in the stack , so there is no back icon . hh



**📘 Lesson: Navigator — push() vs pushReplacement()**

**🔹 1. Quick Recap**

Navigator controls moving between pages (screens).  
Think of it like a **stack of pages**:

🧱 Bottom → HomePage  
🧱 Top → AboutPage

When you move to a new page → you **push** it on top.  
When you go back → you **pop** it off.

**🔹 2. Navigator.push()**

➡️ **Adds (pushes)** a new page on top of the current one.

Navigator.of(context).push(

MaterialPageRoute(builder: (context) => AboutPage()),

);

🧠 **What happens**

* The new page (AboutPage) is placed **on top** of the stack.
* The old page (HomePage) stays **under it**.
* When you press the back button, you return to HomePage.

🧩 **Analogy:**  
Like putting a new card on top of a deck — the old one is still there under it.

**🔹 3. Navigator.pushReplacement()**

➡️ **Replaces** the current page with a new one.

Navigator.of(context).pushReplacement(

MaterialPageRoute(builder: (context) => AboutPage()),

);

🧠 **What happens**

* The current page (HomePage) is **removed** from the stack.
* The new page (AboutPage) **takes its place**.
* The back button won’t return you to HomePage anymore.

🧩 **Analogy:**  
Like taking out the old card and putting a new one in its place.

🧠 So:

* push() keeps the old page under the new one.
* pushReplacement() removes the old page before showing the new one.

  onPressed: () {

                Navigator.of(context).pushReplacement(

                  MaterialPageRoute(builder: (context) => AboutPage()),

                );

              },

# Navigator pop

**Pop 🡺 delete the stack , so you will see the last stack :**

**Example :**

Home (root)

About

Home

About

If you make pop now , you will see home ;

Home (root)

About

Home

If you make pop now , you will see About

It is soooooooo easy .

**📘 Lesson: Navigator — pop()**

**🔹 1. What is pop()?**

* pop() **removes (pops)** the current page from the navigation stack.
* After popping, Flutter shows the **previous page** that was under it.

🧩 **Analogy:**  
Like taking the top card off a stack → the one under it becomes visible.

**🔹 2. Syntax**

Navigator.of(context).pop();

**🔹 3. Example**

Imagine you start on **HomePage** and go to **AboutPage** with push():

**Stack before pop:**

AboutPage

HomePage

If you call ***Navigator.pop()*** inside AboutPage:

**Stack after pop:**

HomePage

✅ You return to HomePage.

✅ **Key Idea:**

* Use pop() to **go back**.
* Works only if the previous page is still in the stack (added with push()).

# Navigator.pushAndRemoveUntil()

You will learn how to :

* Go to a new page and remove all previous pages (so you can’t go back ).
* It’s useful after **login screens** , **splash screens**, or **logout** .

🧠 Explanation

|  |  |
| --- | --- |
| Function | What it does |
| Navigator.push() | Pushes a new route on top of the stack (you can go back) |
| Navigator.pushReplacement() | Replaces the current route with a new one (you can’t go back to current) |
| Navigator.pushAndRemoveUntil() | Pushes a new route and removes **some or all** previous routes |

**Syntax:**

*Navigator.pushAndRemoveUntil(*

*context,*

*MaterialPageRoute(builder: (context) => NewPage()),*

*(Route<dynamic> route) => condition,*

*);*

* If you write (route) => false → remove **all previous pages**
* If you write (route) => route.isFirst → keep only the **first page**

# Routes

Think of it like a dictionary

*routes = {*

*'route name' : (context) => WidgetToShow()*

*}*

So when you write

*Navigator.pushNamed(context, '/about');*

Flutter looks inside your **routes** map, finds '/about',  
and shows whatever widget you assigned — here it’s *AboutPage().*

So , in Flutter , inside you MaterialApp, you can define **named routes** like this :

import 'package:flutter/material.dart';

import 'package:flutter\_application\_1/homePage.dart';

import 'package:flutter\_application\_1/aboutPage.dart';

import 'package:flutter\_application\_1/contactPage.dart';

void main() {

  runApp(MyApp());

}

// Moroccan Cities App

class MyApp extends StatefulWidget {

  @override

  State<MyApp> createState() => \_MyAppState();

}

class \_MyAppState extends State<MyApp> {

  @override

  Widget build(BuildContext context) {

    return MaterialApp(

      debugShowCheckedModeBanner: false,

      initialRoute: '/home',

      routes: {

        '/home': (context) => Homepage(),

        '/about': (context) => AboutPage(),

        '/contact': (context) => ContactPage(),

      },

    );

  }

}

## Why Routes in Flutter are Great

Using **routes** (the routes: {} map in MaterialApp ) gives you **many advantages** in your app- both for organization and for cleaner navigation code.

### 🧠 1. You define all pages in one place

When you use:

*routes: {*

*'/login': (context) => LoginPage(),*

*'/home': (context) => HomePage(),*

*'/about': (context) => AboutPage(),*

*},*

✅ All your pages (screens) are declared in one central spot — inside main.dart.

➡ That means you don’t need to import every file again in other pages.  
You just write:

*Navigator.pushNamed(context, '/about');*

and Flutter already knows which widget that name points to.

### 🧭 2. Cleaner navigation everywhere

Instead of long code like:

*Navigator.push(*

*context,*

*MaterialPageRoute(builder: (context) => HomePage()),*

*);*

you simply write:

*Navigator.pushNamed(context, '/home');*

💡 This is shorter, easier to read, and better for large apps.

### 🏠 3. You can easily set the first screen with initialRoute

You can decide where your app starts:

*initialRoute: '/login',*

or change it anytime (for example, after user logs in):

*initialRoute: '/home',*

No need to change your whole main() function — just update that one line!

### 🧱 4. Works perfectly with pushNamedAndRemoveUntil

When you log in or log out, you can easily go to a route name and remove the previous ones:

*Navigator.pushNamedAndRemoveUntil(context, '/home', (route) => false);*

This is very clean compared to manually removing routes.

### 🧩 5. Easier to maintain in big projects

When your app grows (10+ screens), using routes helps you:

Keep navigation consistent

Avoid duplicate imports

Rename or reorganize screens easily

For example, if you change AboutPage to InfoPage, you just update once in routes: {} — no need to fix imports everywhere.

### 🛠️ 6. Supports arguments between pages

You can even send data between pages with routes:

*Navigator.pushNamed(*

*context,*

*'/details',*

*arguments: 'Product ID: 42',*

*);*

And then receive it in that route using:

*final args = ModalRoute.of(context)!.settings.arguments;*

So it’s not only cleaner — it’s more powerful.

### 🧩 7. Works great with Route Generators

If your app becomes huge, you can move all routes to a separate file, like **routes.dart**.  
That keeps **main.dart** super clean and professional.

### Summary :

|  |  |
| --- | --- |
| Benefit | Description |
| Central control | All screens managed from one place |
| No extra imports | Navigate by name instead of importing each page |
| Cleaner navigation | Use pushNamed instead of long route code |
| Flexible startup | Change first page easily with initialRoute |
| Powerful control | Use pushNamedAndRemoveUntil for clean navigation stack |
| Supports data | Pass arguments between pages |
| Scalable | Easy to maintain in large apps |

# showDialog()

## Alert Dialog

**🎯 Goal**

You’ll learn how to:

1. Show a simple dialog (popup)
2. Add buttons like “OK” or “Cancel”
3. Close the dialog
4. Return a value (optional)

ListView(

        children: [

          Center(

            child: MaterialButton(

              onPressed: () {

                showDialog(

                  context: context,

                  builder: (BuildContext context) {

                    return AlertDialog(

                      title: Text("Hello!"),

                      content: Text("This is a simple dialog."),

                      actions: [

                        TextButton(

                          onPressed: () {

                            Navigator.pop(context); // Close the dialog

                          },

                          child: Text("OK"),

                        ),

                      ],

                    );

                  },

                );

              },

              color: Colors.red,

              child: Text("Show Dialog"),

            ),

          ),

        ],

      ),

And you can also handle the response :

  Center(

            child: MaterialButton(

              onPressed: () async {

                final result = await showDialog(

                  context: context,

                  builder: (context) {

                    return AlertDialog(

                      title: Text("Exit"),

                      content: Text("Are you sure you want to exit?"),

                      actions: [

                        TextButton(

                          onPressed: () => Navigator.pop(context, false),

                          child: Text("No"),

                        ),

                        TextButton(

                          onPressed: () => Navigator.pop(context, true),

                          child: Text("Yes"),

                        ),

                      ],

                    );

                  },

                );

                // 👇 do something with the result

                if (result == true) {

                  print("User chose YES");

                } else {

                  print("User chose NO");

                }

              },

              color: Colors.red,

              child: Text("Show Dialog"),

            ),

          ),

And for sure you can style all :

MaterialButton(

              onPressed: () {

                showDialog(

                  context: context,

                  builder: (BuildContext context) {

                    return AlertDialog(

                      title: Text("Hello!"),

                      titleTextStyle: TextStyle(

                        color: Colors.blue,

                        fontSize: 20,

                        fontWeight: FontWeight.bold,

                      ),

                      titlePadding: EdgeInsets.all(10),

                      content: Text("This is a simple dialog."),

                      contentTextStyle: TextStyle(

                        color: Colors.black,

                        fontSize: 16,

                      ),

                      backgroundColor: Colors.white,

                      shape: RoundedRectangleBorder(

                        borderRadius: BorderRadius.circular(10),

                      ),

                      contentPadding: EdgeInsets.all(20),

                      actions: [

                        TextButton(

                          onPressed: () {

                            Navigator.pop(context); // Close the dialog

                          },

                          child: Text("OK"),

                        ),

                      ],

                    );

                  },

                );

              },

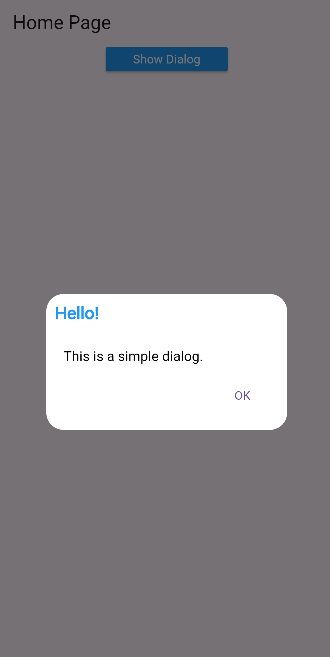
              color: Colors.blue,

              textColor: Colors.white,

              minWidth: 150,

              child: Text("Show Dialog"),

            ),



Now before passing to other that dialog types , let’s speak about some imporant properties :

|  |  |
| --- | --- |
| Property | Meaning |
| context | The current screen — needed to show the dialog. |
| builder | Tells Flutter **what dialog widget** to show (like AlertDialog). |
| barrierDismissible | If true → you can close the dialog by tapping outside it. If false → you must press a button to close. |
| barrierColor | Color of the dark background behind the dialog. |
| useSafeArea | If true, keeps dialog away from phone notches/status bar. |

## Simple Dialog :

  Center(

            child: MaterialButton(

              onPressed: () async {

                String? selectedLanguage = await showDialog(

                  context: context,

                  builder: (context) => SimpleDialog(

                    title: Text("Choose language"),

                    children: [

                      SimpleDialogOption(

                        onPressed: () => Navigator.pop(context, 'English'),

                        child: Text("English"),

                      ),

                      SimpleDialogOption(

                        onPressed: () => Navigator.pop(context, 'Arabic'),

                        child: Text("Arabic"),

                      ),

                    ],

                  ),

                );

                print("Selected Language: $selectedLanguage");

              },

              color: Colors.blue,

              textColor: Colors.white,

              minWidth: 150,

              child: Text("Choose Your Language"),

            ),

          ),

It’s so simple , see the code , you can learn all from just make an eye contact with the code HHHH .

# showBottomSheet And SnackBar

## showBottomSheet

it shows a **panel from the bottom** of the screen – usually for quick actions or info .

### ⚙️ Main Properties

|  |  |
| --- | --- |
| Property | Meaning |
| context | Current screen context. |
| builder | Function that returns the content of the bottom sheet. |
| backgroundColor | Background color (you can also use inside Container). |
| shape | Rounded corners or other shapes. |
| elevation | Shadow depth (higher = more shadow). |
| enableDrag | If true, allows dragging to close. |

Example :

Scaffold(

      key: \_scaffoldKey,

      appBar: AppBar(title: Text("Bottom Sheet Example")),

      body: Center(

        child: MaterialButton(

          color: Colors.blue,

          textColor: Colors.white,

          onPressed: () {

            \_scaffoldKey.currentState?.showBottomSheet(

              enableDrag: true,

              (context) => Container(

                height: 200,

                color: Colors.blue[100],

                child: Center(

                  child: Text(

                    "This is a Bottom Sheet",

                    style: TextStyle(fontSize: 18),

                  ),

                ),

              ),

            );

          },

          child: Text("Open Bottom Sheet"),

        ),

      ),

    );

But if you try shape and elevation will not work hh ,

**🧠 Why shape and elevation Don’t Work with showBottomSheet()**

**💬 Reason:**

When you call:

\_scaffoldKey.currentState?.showBottomSheet(

(context) => Container(...),

);

you’re only returning a **Container**, not a **BottomSheet widget** itself.

👉 So Flutter doesn’t know how to apply shape, backgroundColor, elevation, etc.  
Those properties belong to the **BottomSheet widget**, not to a normal Container

So this is why use :

**💡 Tip:**

You can also use:

showModalBottomSheet()

👉 It’s similar, but it **blocks the screen behind** (like a popup).  
It’s the **most used one** in real apps.

Example:

*showModalBottomSheet(*

*context: context,*

*builder: (context) => Container(*

*padding: EdgeInsets.all(20),*

*height: 200,*

*child: Center(child: Text("This is a Modal Bottom Sheet")),*

*),*

*);*

## showModalBottomSheet

**🧩 Qu’est-ce que c’est ?**

* C’est un **bottom sheet modal**, donc il **bloque l’écran derrière**.
* L’utilisateur peut **le fermer en glissant vers le bas** ou en **tapant à l’extérieur** (sauf si tu désactives avec isDismissible: false).
* Très utilisé pour **menus rapides, options, formulaires courts**.

Center(

        child: MaterialButton(

          color: Colors.blue,

          textColor: Colors.white,

          onPressed: () {

            // showModalBottomSheet

            showModalBottomSheet(

              context: context,

              isDismissible: true,

              backgroundColor: Colors.blueAccent,

              shape: RoundedRectangleBorder(

                borderRadius: BorderRadius.vertical(top: Radius.circular(30)),

              ),

              builder: (BuildContext context) {

                return Container(

                  height: 200,

                  child: Center(

                    child: Text(

                      "This is a Bottom Sheet",

                      style: TextStyle(color: Colors.white, fontSize: 18),

                    ),

                  ),

                );

              },

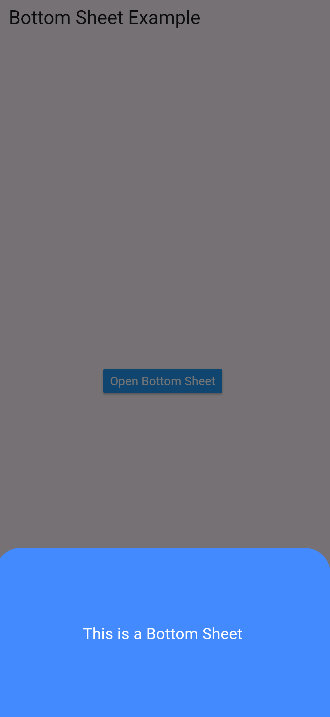
            );

          },

          child: Text("Open Bottom Sheet"),

        ),

      ),



**💡 Astuce**

* Si tu veux **avoir un fond différent à l’intérieur**, utilise un **Padding** ou un **ClipRRect**, mais **ne change pas la couleur directement dans le Container principal**.
* shape et backgroundColor de showModalBottomSheet contrôlent **le vrai fond et les coins**, pas le Container à l’intérieur.

## SnackBar

### 🧩 Qu’est-ce que c’est ?

* Un **petit message** qui apparaît en bas de l’écran.
* Idéal pour les **notifications rapides**, confirmations, alertes ou erreurs.
* **Disparaît automatiquement** après quelques secondes (ou peut avoir une action).

### ⚙️ Syntaxe simple

ScaffoldMessenger.of(context).showSnackBar(

SnackBar(

content: Text("Message envoyé!"),

duration: Duration(seconds: 2),

),

);

### 🔑 Propriétés importantes

|  |  |
| --- | --- |
| Propriété | Signification |
| content | Le texte ou widget affiché dans le snackbar |
| duration | Temps avant disparition automatique |
| backgroundColor | Couleur de fond |
| action | Bouton d’action (ex : Undo) |
| behavior | Position/floating ou fixé au bas (SnackBarBehavior.floating) |
| shape | Coins arrondis ou forme personnalisée |

### Example

Center(

        child: MaterialButton(

          color: Colors.blue,

          textColor: Colors.white,

          minWidth: 150,

          onPressed: () {

            ScaffoldMessenger.of(context).showSnackBar(

              SnackBar(

                content: Text("The Text is Copied"),

                duration: Duration(seconds: 2),

                action: SnackBarAction(

                  label: "Cancel",

                  textColor: Colors.yellow,

                  onPressed: () {

                    // Code to execute when the action is pressed

                    print("Undo action pressed : Text is not Copied");

                  },

                ),

                backgroundColor: Colors.green,

              ),

            );

          },

          child: Text("Copy Text"),

        ),

      ),

# List generate

Imagine that you have a list of users, and you want to make for each a card .

import 'package:flutter/material.dart';

void main() {

  runApp(MyApp());

}

// Moroccan Cities App

class MyApp extends StatefulWidget {

  @override

  State<MyApp> createState() => \_MyAppState();

}

class \_MyAppState extends State<MyApp> {

  List username = [

    {"name": "Bilal", "age": "19"},

    {"name": "Ahmed", "age": "20"},

    {"name": "Yassine", "age": "21"},

    {"name": "Omar", "age": "22"},

    {"name": "Salim", "age": "23"},

  ];

  @override

  Widget build(BuildContext context) {

    return MaterialApp(

      debugShowCheckedModeBanner: false,

      home: Scaffold(

        appBar: AppBar(title: Text("List Generate")),

        body: ListView(

          children: [

            Card(

              child: ListTile(

                title: Text("Name : ${username[0]["name"]}"),

                subtitle: Text(username[0]["age"]),

                leading: Icon(Icons.person),

                trailing: Icon(Icons.arrow\_forward),

              ),

            ),

          ],

        ),

      ),

    );

  }

}

But if you have 1000 user or just 50 hh , it’s realy HARD .

But with List Generate it’s easy .

ListView(

          children: List.generate(

            username.length,

            (i) => Card(

              child: ListTile(

                title: Text(username[i]['name']),

                subtitle: Text(username[i]['age']),

                leading: CircleAvatar(child: Text(username[i]['name'][0])),

              ),

            ),

          ),

        ),

### ⚙️ Parameters explained

|  |  |  |
| --- | --- | --- |
| Property | Type | Description |
| **length** | int | Number of elements the list will have. |
| **generator** | Function | A function (index) => value that returns the element at each position. |
| **growable** | bool *(optional)* | If true, the list can later add/remove elements. Default is true in Flutter. |

# initState & dispose

before starting, I just wanted to remember you about **“super”**

**🧠 Definition**

super means:

“Call something from the **parent class**.”

In Flutter, every widget you create (like StatelessWidget, StatefulWidget, or State) comes from another class.  
When you use super, you’re talking to that parent.

🔍 **Example 1 – Simple inheritance**

class Animal {

void speak() {

print("Animal speaks");

}

}

class Dog extends Animal {

void speak() {

super.speak(); // calls the parent's speak()

print("Dog barks");

}

}

🧩 Output:

Animal speaks

Dog barks

## initState

### What is initState()

* initState() is a **lifecycle method** of a StatefulWidget.
* It runs **once**, when the widget is **first created**.
* Its purpose is to **initialize things** your widget needs before it shows on the screen.

### Why we use it

* To set up variables or data.
* To start animations or timers.
* To perform tasks that need to happen only once (like fetching data from a database).

### How to use it

1. Only in a StatefulWidget’s **State class**.
2. Always call super.initState() first to let Flutter do its internal setup.
3. Add your custom setup code after super.initState().

If you have time read this :

**🧱 What happens in Flutter**

When you make a StatefulWidget, Flutter creates **two classes**:

1. The StatefulWidget itself (PageOne)
2. Its **state class** (\_PageOneState) — this is where you keep your data, logic, and lifecycle methods like initState() and build().

**⚙️ The initState() function**

initState() runs **only once**, right when the widget is first created (mounted).  
It’s perfect for:

* Printing debug info
* Loading data from a database
* Starting an animation or timer

**🚨 But here’s the important part**

Your State class **inherits** from the Flutter framework’s internal class:

class \_PageOneState extends State<PageOne>

That parent State class (from Flutter) also has its own initState() method —  
and it does important setup work for Flutter **behind the scenes** (like connecting your widget to the Flutter tree).

So when you write your own version of initState(),  
you are **overriding** the parent’s method.

That’s why you must still call:

super.initState();

➡️ It means:

“Run Flutter’s internal setup code first, then run mine.”

**🧠 Simple analogy**

Think of it like this:

|  |  |
| --- | --- |
| Role | Action |
| Flutter’s parent initState() | Builds the foundation of the house 🏠 |
| Your initState() | Puts the furniture and decorations inside 🪑 |

If you **skip super.initState(),**then the “foundation” might not be ready before you start adding your own code.

**✅ Correct pattern**

@override

void initState() {

super.initState(); // always first

print("My custom setup");

}

## Example

Main.dart

import 'package:flutter/material.dart';

import 'package:flutter\_application\_1/PageOne.dart';

import 'package:flutter\_application\_1/PageTwo.dart';

void main() {

  runApp(MyApp());

}

// Moroccan Cities App

class MyApp extends StatefulWidget {

  @override

  State<MyApp> createState() => \_MyAppState();

}

class \_MyAppState extends State<MyApp> {

  @override

  Widget build(BuildContext context) {

    return MaterialApp(

      debugShowCheckedModeBanner: false,

      routes: {

        '/pageOne': (context) => PageOne(),

        '/pageTwo': (context) => PageTwo(),

      },

      home: PageOne(),

    );

  }

}

PageOne.dart

import 'package:flutter/material.dart';

class PageOne extends StatefulWidget {

  @override

  \_PageOneState createState() => \_PageOneState();

}

class \_PageOneState extends State<PageOne> {

  @override

  void initState() {

    super.initState(); // Always call super first

    print("PageOne initialized");

  }

  @override

  Widget build(BuildContext context) {

    return Scaffold(

      appBar: AppBar(title: Text("Page One")),

      body: Center(

        child: ElevatedButton(

          onPressed: () {

            Navigator.pushNamed(context, '/pageTwo');

          },

          child: Text("Go to Page Two"),

        ),

      ),

    );

  }

}

PageTwo.dart

import 'package:flutter/material.dart';

class PageTwo extends StatefulWidget {

  @override

  \_PageTwoState createState() => \_PageTwoState();

}

class \_PageTwoState extends State<PageTwo> {

  @override

  void initState() {

    super.initState(); // Always call super first

    print("PageTwo initialized");

  }

  @override

  Widget build(BuildContext context) {

    return Scaffold(

      appBar: AppBar(title: Text("Page One")),

      body: Center(

        child: ElevatedButton(

          onPressed: () {

            Navigator.of(context).pop();

          },

          child: Text("Go to Page Two"),

        ),

      ),

    );

  }

}

**PageTwo is reinitialized each time we enter it, because when we go out of it we delete it from the stack.**

## Dispose ()

 dispose() is a **lifecycle method** of a StatefulWidget.

 It is called **once**, when the widget is **removed permanently from the widget tree**.

 Its purpose is to **clean up resources** that the widget used, such as:

* Controllers (like TextEditingController or AnimationController)
* Streams or listeners
* Timers

**Key points**

* Always call super.dispose() at the **end** of your method.
* dispose() is called **automatically** when the page is removed from the stack permanently.
* This is useful to **prevent memory leaks**.

  @override

  void initState() {

    super.initState(); // Always call super first

    print("PageTwo initialized");

  }

  @override

  void dispose() {

    super.dispose();

    print("PageTwo disposed");

  }

# PopupMenuButton

**1️⃣ PopupMenuButton**

A **PopupMenuButton** is a button that shows a **menu when clicked**.

|  |  |  |
| --- | --- | --- |
| Property | Type | What it does |
| onSelected | Function(T value) | Called when a menu item is selected. The value is returned. |
| onCanceled | Function() | Called when the user taps outside or closes the menu without selecting. |
| onOpened | Function() | Called when the menu is opened. |
| itemBuilder | Function(BuildContext) | Builds the **list of PopupMenuItem widgets**. Required. |
| icon | Widget | Replace the default 3 dots ⋮ with any icon or widget. |
| child | Widget | Alternative to icon, you can use a custom button/widget. |
| color | Color | Background color of the menu. |
| elevation | double | Shadow of the menu popup. |
| shape | ShapeBorder | Rounded corners or custom shape for the menu. |
| initialValue | T | Optional, the value initially highlighted in the menu. |

**2️⃣ PopupMenuItem**

A **PopupMenuItem** is a single option in the popup menu.

Main Properties

|  |  |  |
| --- | --- | --- |
| Property | Type | What it does |
| value | T | Value returned to onSelected when clicked. |
| child | Widget | The visible widget (usually Text or Row with Icon+Text). |
| onTap | Function() | Called when this specific item is clicked **before onSelected**. |
| enabled | bool | If false, the item is disabled (greyed out). |
| height | double | Height of this menu item. |

3️⃣ How it works together

 User taps the **PopupMenuButton**.

 Menu **appears** with all PopupMenuItems.

 User taps an item:

* First PopupMenuItem.onTap runs (if you defined it).
* Then PopupMenuButton.onSelected runs with the value of the selected item.

 If the user taps outside: onCanceled runs.

 When the menu opens: onOpened runs.

**4️⃣ Extra useful notes**

* You can put **Icons, Rows, or custom widgets** inside a PopupMenuItem.
* Can be used **anywhere**, not only AppBar.
* PopupMenuDivider() can be used to **separate items**.
* Works with **any type** for value: String, int, enum…
* You can **change the menu color, shape, shadow** with color, shape, elevation.

✅ **Analogy:** Think of it like **right-clicking on Windows**: small menu pops up with options, you pick one, it runs an action.

Example :

import 'package:flutter/material.dart';

class PageOne extends StatefulWidget {

  @override

  \_PageOneState createState() => \_PageOneState();

}

class \_PageOneState extends State<PageOne> {

  String \_selectedValue = "0";

  @override

  void initState() {

    super.initState(); // Always call super first

    print("PageOne initialized");

  }

  @override

  Widget build(BuildContext context) {

    return Scaffold(

      appBar: AppBar(

        title: Text("PopupMenu Button Example"),

        actions: [

          PopupMenuButton(

            color: Colors.amber,

            icon: Icon(Icons.color\_lens),

            onSelected: (val) {

              setState(() {

                \_selectedValue = val;

              });

              print(val);

            },

            onOpened: () {

              print("Menu opened");

            },

            onCanceled: () {

              print("Menu canceled");

            },

            itemBuilder: (context) => [

              PopupMenuItem(

                child: Text("Red"),

                value: "1",

                onTap: () {

                  print("item one is clicked");

                },

              ),

              PopupMenuItem(child: Text("Green"), value: "2"),

              PopupMenuItem(child: Text("Blue"), value: "3"),

            ],

          ),

        ],

      ),

      body: Container(

        color: \_selectedValue == "1"

            ? Colors.red

            : \_selectedValue == "2"

            ? Colors.green

            : Colors.blue,

      ),

    );

  }

}



# searchDelegate

## What is SearchDelegate ?

It’s a **built-in Flutter class** that helps you easily create a **search bar** and handle:

* The **search text** the user types
* The **results** (filtered data)
* The **suggestions** while typing
* The **actions** (like clear button) and **leading icon** (like back arrow)

Okay the search bar is in the app bar ,

So in actions make a search icon button , and in onPressed() function use the built-in function **showSearch()** accept two paramter contect and **delegate (refer to the search class that we will create)**

In the search class that we will create , we will add the list that we will search from and also the list filtered list that will contain live suggestions, see below there a simple dart logic .

In general this class must contain 4 main methods

The 4 main methods you must override 🔍

|  |  |  |
| --- | --- | --- |
| Method | Purpose | Example |
| buildActions() | The icons on the **right** of the search bar | The ❌ clear button |
| buildLeading() | The icon on the **left** (usually back arrow) | The 🔙 back button |
| buildResults() | What appears when the user **taps on a suggestion** | A result page or message |
| buildSuggestions() | Shows suggestions **while typing** | Filtered names |

import 'package:flutter/material.dart';

class PageOne extends StatefulWidget {

  @override

  \_PageOneState createState() => \_PageOneState();

}

class \_PageOneState extends State<PageOne> {

  @override

  Widget build(BuildContext context) {

    return Scaffold(

      appBar: AppBar(

        title: Text("Search Delegate"),

        actions: [

          IconButton(

            onPressed: () {

              // buil-in funcition to show search

              showSearch(context: context, delegate: customSearchDelegate());

              print("Search Clicked");

            },

            icon: Icon(Icons.search),

          ),

        ],

      ),

    );

  }

}

class customSearchDelegate extends SearchDelegate {

  List username = [

    "Bilal",

    "Ahmed",

    "Ali",

    "Hassan",

    "Hicham",

    "Hussein",

    "Mohamed",

    "Mostafa",

    "Omar",

    "Youssef",

    "Younes",

  ];

  List? filteredList;

  @override

  List<Widget>? buildActions(BuildContext context) {

    return [

      IconButton(

        onPressed: () {

          // clear the search bar

          query = "";

        },

        icon: Icon(Icons.close),

      ),

    ];

  }

  @override

  Widget? buildLeading(BuildContext context) {

    return IconButton(

      onPressed: () {

        // close search bar

        close(context, null);

      },

      icon: Icon(Icons.arrow\_back),

    );

  }

  @override

  Widget buildResults(BuildContext context) {

    return Text("Result ");

  }

  @override

  Widget buildSuggestions(BuildContext context) {

    if (query == "") {

      return ListView.builder(

        itemCount: username.length,

        itemBuilder: (context, i) {

          return InkWell(

            onTap: () {

              showResults(context);

            },

            child: Card(

              child: Padding(

                padding: EdgeInsetsGeometry.all(18),

                child: Text("${username[i]}", style: TextStyle(fontSize: 16)),

              ),

            ),

          );

        },

      );

    } else {

      // to ignore Case sensitive

      filteredList = username

          .where(

            (element) =>

                element.toString().toLowerCase().contains(query.toLowerCase()),

          )

          .toList();

      return ListView.builder(

        itemCount: filteredList!.length,

        itemBuilder: (context, i) {

          return InkWell(

            onTap: () {

              showResults(context);

            },

            child: Card(

              child: Padding(

                padding: EdgeInsetsGeometry.all(18),

                child: Text(

                  "${filteredListi]}",

                  style: TextStyle(fontSize: 16),

                ),

              ),

            ),

          );

        },

      );

    }

  }

}

![A screenshot of a phone

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

💡 Summary (easy recap)

|  |  |
| --- | --- |
| Part | What it does |
| showSearch() | Opens the search bar |
| SearchDelegate | Controls the design and logic |
| buildActions() | Add clear button |
| buildLeading() | Add back arrow |
| buildResults() | What to show when selecting |
| buildSuggestions() | Filter list live while typing |

# Scroll

## 🧠 What is a ScrollController?

A **ScrollController** allows you to:

* **Listen** to scroll movements
* **Know** how far the user has scrolled
* **Move** the scroll position programmatically (like “scroll to top”)

So first let create some items in a listview to can scroll and practice what we will learn .

A screenshot of a blue box

AI-generated content may be incorrect.

So now we need a scroll controller to add it to this listview and make it as listener :

import 'package:flutter/material.dart';

class PageOne extends StatefulWidget {

  @override

  \_PageOneState createState() => \_PageOneState();

  // scaffold key for scrolling

}

class \_PageOneState extends State<PageOne> {

  ScrollController? scrollController;

  @override

  void initState() {

    super.initState();

    scrollController = ScrollController();

    scrollController!.addListener(() {

      print("Your are scrolling");

    });

  }

  @override

  Widget build(BuildContext context) {

    return Scaffold(

      appBar: AppBar(title: Text("Scroll")),

      body: ListView(

        controller: scrollController,

        children: [

          ...List.generate(

            100,

            (index) => Container(

              height: 50,

              margin: EdgeInsets.all(5),

              color: Colors.blue,

              child: Center(

                child: Text(

                  "Item ${index + 1}",

                  style: TextStyle(color: Colors.white, fontSize: 18),

                ),

              ),

            ),

          ),

        ],

      ),

    );

  }

}

Focus well I make it in initstate ;

Now let some important ScrollController properties and methods :

## Properties and methods

### 🧠 1️⃣ .offset

➡️ **What it does:**  
Gives you the **current scroll position** (in pixels).

  scrollController!.addListener(() {

      print("Scroll Position: ${scrollController!.offset}");

    });

✅ Example:  
If you scrolled down 150 pixels → it prints 150.0.

### 🧠 2️⃣ .jumpTo(double value)

➡️ **What it does:**  
Moves the scroll **instantly** to a position (no animation).

To practice let add two , one at top for “**scrolling to bottom**” and other one at the bottom for “**scrolling to top** “

(wait for example see below )

✅ Example:  
Scrolls directly to pixel 200 without animation.

🧩 Tip: usually used when you want an *instant move* (like setting scroll position quickly).

### 🧠 3️⃣ .animateTo(double offset, {duration, curve})

➡️ **What it does:**  
Moves the scroll **smoothly** (with animation) to a position.

✅ Example:  
Scrolls smoothly to pixel 300 in 0.5 seconds.

🧩 Tip: used for nice animations (like “scroll to top” or “scroll to bottom”).

### 🧠 4️⃣ .position

➡️ **What it does:**  
Gives access to **scroll info**, like how far we can scroll.

import 'package:flutter/material.dart';

class PageOne extends StatefulWidget {

  @override

  \_PageOneState createState() => \_PageOneState();

  // scaffold key for scrolling

}

class \_PageOneState extends State<PageOne> {

  ScrollController? scrollController;

  @override

  void initState() {

    super.initState();

    scrollController = ScrollController();

    scrollController!.addListener(() {

      print("Scroll Position: ${scrollController!.offset}");

    });

  }

  @override

  Widget build(BuildContext context) {

    return Scaffold(

      appBar: AppBar(title: Text("Scroll")),

      body: ListView(

        controller: scrollController,

        children: [

          MaterialButton(

            color: Colors.green,

            onPressed: () {

              scrollController!.jumpTo(

                scrollController!.position.maxScrollExtent,

              );

            },

            child: Text(

              "Jump to Bottom",

              style: TextStyle(color: Colors.white),

            ),

          ),

          ...List.generate(

            100,

            (index) => Container(

              height: 50,

              margin: EdgeInsets.all(5),

              color: Colors.blue,

              child: Center(

                child: Text(

                  "Item ${index + 1}",

                  style: TextStyle(color: Colors.white, fontSize: 18),

                ),

              ),

            ),

          ),

          MaterialButton(

                color: Colors.green,

            onPressed: () {

              scrollController!.animateTo(

                0,

                duration: Duration(seconds: 2),

                curve: Curves.fastOutSlowIn,

              );

            },

            child: Text("Go to Top"),

          ),

        ],

      ),

    );

  }

}

### 🧠 5️⃣ .addListener()

➡️ **What it does:**  
Lets you **listen** to scroll movements.

  scrollController!.addListener(() {

      print("Scroll Position: ${scrollController!.offset}");

    });

✅ Example:  
Used to detect scrolling direction or trigger actions like “load more”.

### 🧠 6️⃣ .removeListener(listener)

➡️ **What it does:**  
Stops a previously added listener.

\_controller.removeListener(myListenerFunction);

✅ Example:  
Useful when you don’t need the listener anymore to save resources.

### 🧠 7️⃣ .hasClients

➡️ **What it does:**  
Checks if the controller is attached to a scrollable widget.

if (\_controller.hasClients) {

print("Controller connected to scroll view");

}

✅ Example:  
Used to avoid errors if you call animateTo() before the widget is ready.

### 🧠 8️⃣ .dispose()

➡️ **What it does:**  
Cleans up the controller when you’re done using it.

  @override

  void dispose() {

    scrollController!.dispose();

    super.dispose();

  }

### ⚙️ Example Summary:

|  |  |  |
| --- | --- | --- |
| Method / Property | Description | Example |
| .offset | Current scroll position | \_controller.offset |
| .jumpTo(value) | Jump instantly to position | \_controller.jumpTo(0) |
| .animateTo(value) | Smooth scroll | \_controller.animateTo(500, duration: ..., curve: ...) |
| .position | Info (pixels, maxScrollExtent, etc.) | \_controller.position.maxScrollExtent |
| .addListener() | Listen to scroll changes | \_controller.addListener(() {}) |
| .hasClients | Check if connected | if (\_controller.hasClients) |
| .dispose() | Clean controller | @override dispose() |

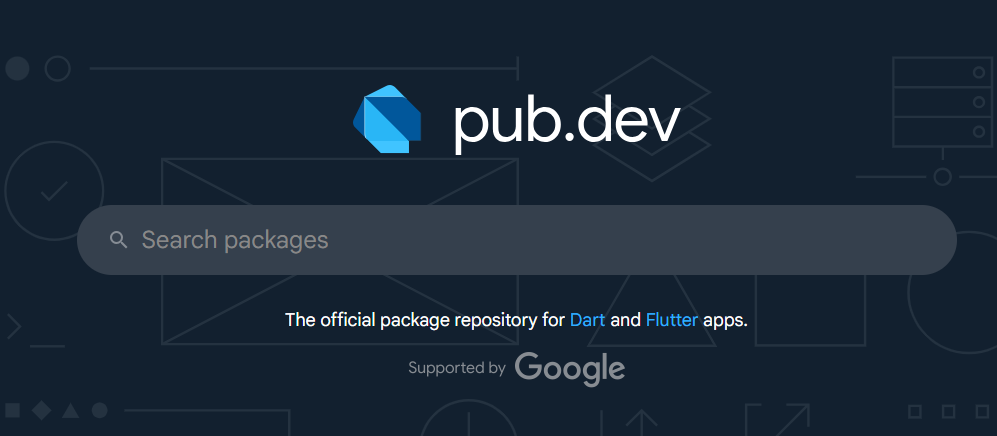
# Package

Flutter packages can be divided into **3 main parts**:

1. **Services** → for APIs, online services, maps, Firebase, etc.
2. **Hardware** → for device features like camera, Bluetooth, Wi-Fi, sensors.
3. **Built-in Widgets** → for UI components, navigation, and standard Flutter widgets.

Okay to get built-in widget ( so I am talking about part 3),

1️⃣ open ***pub.dev***



So you are in the ready widget’s market .

2️⃣ **choose** something hh

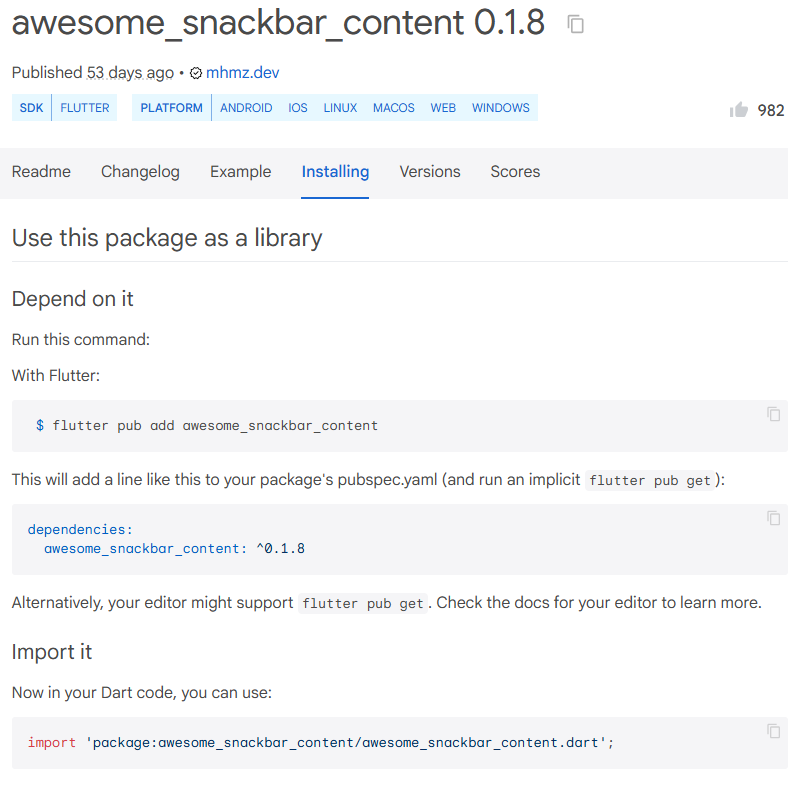
A screenshot of a chat box

AI-generated content may be incorrect.

I chooseed this awesome snackbar .

https://pub.dev/packages/awesome\_snackbar\_content

you can find all , let’s see how to get it .



So there *is 2 choices* for install it ,

***1 – automtic*** : flutter pub add awesome\_snackbar\_content

***2- manually*** : goo pubspec.yaml

dependencies:

awesome\_snackbar\_content: ^0.1.8

then **run pub get** .

**so automatic** : write it you and do pub get also .

then import it in you code normally .

import 'package:awesome\_snackbar\_content/awesome\_snackbar\_content.dart';

okay let’s me do it and see the result toghether .

and sure

**Note :**

After importing and writing the code , close your emulator and reopen it .

And I did it :

import 'package:flutter/material.dart';

import 'package:awesome\_snackbar\_content/awesome\_snackbar\_content.dart';

class PageOne extends StatefulWidget {

  @override

  \_PageOneState createState() => \_PageOneState();

}

class \_PageOneState extends State<PageOne> {

  @override

  Widget build(BuildContext context) {

    return Scaffold(

      appBar: AppBar(title: Text("Package: failure snackbar")),

      body: Center(

        child: ElevatedButton(

          onPressed: () {

            final snackBar = SnackBar(

              content: AwesomeSnackbarContent(

                title: "On Snap !",

                message: "You have failed to read this failure message",

                contentType: ContentType.failure,

              ),

              elevation: 0,

              backgroundColor: Colors.transparent,

            );

            ScaffoldMessenger.of(context).showSnackBar(snackBar);

          },

          child: Text("Show Ops Awesome SnackBar"),

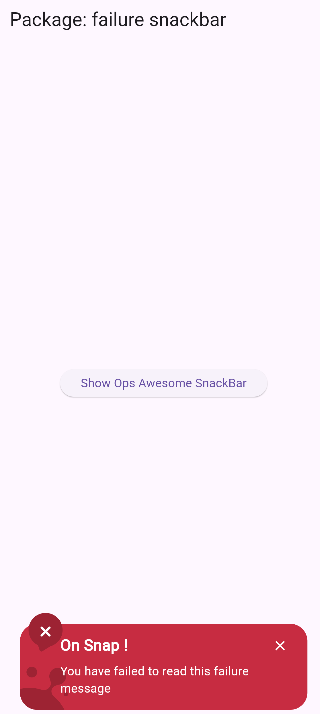
        ),

      ),

    );

  }

}



Let’s all types , for you have just to change : : this line : :

contentType: ContentType.failure,

              ),

 A white rectangular object with purple text

AI-generated content may be incorrect. A white rectangular object with purple text

AI-generated content may be incorrect.

Yeah we end this Task hhh .

# HTTP & Api

## 🧠 1. What is an API?

**API** stands for **Application Programming Interface.**

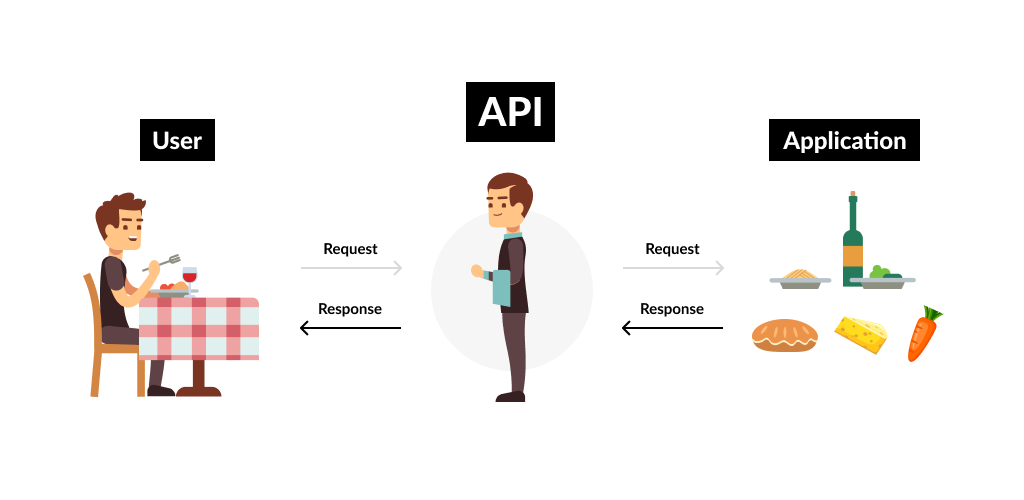
In short, it’s a **way for apps to talk to servers**.

*For example :*

If your flutter app wants to get weather info 🌤️ from a website –

You don’t open the site manually:

You **ask the site’s API** 🡺 it gives you the data (in JSON format).



In this part of our course we will practice in a fake API

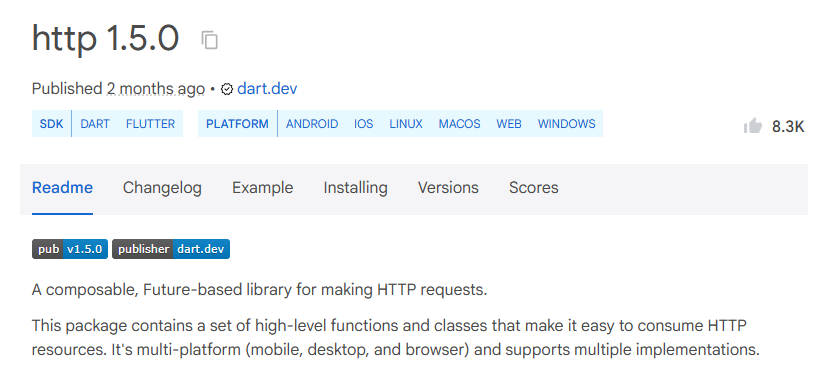
<https://jsonplaceholder.typicode.com/posts>

this is fake API just to preactice what we will learn

## HTTP Package

**HTTP** is the protocol used to send or receive data from the internet (websites, APIs, etc.).  
In Flutter, we use the **http package** to make HTTP requests.

Okay first let’s install this package, As I told you goo to pub.dev



**Note :** there other package that can make this Task , but Http package is the most known.

Install this package and let’s get started , to explore , power power yeeeeeeeeah .

### URI

Uri means **Uniform Resource Identifier**.  
It’s like an **address** that tells your app **where** to send or get data from —  
for example, an API link.

Dart needs to **understand** that this is a **web address**.  
So we convert it into a **Uri object** using:

  Uri.parse("https://jsonplaceholder.typicode.com/posts"),

Full code :

import 'package:flutter/material.dart';

import 'package:http/http.dart';

// json

import 'dart:convert';

class PageOne extends StatefulWidget {

  @override

  \_PageOneState createState() => \_PageOneState();

}

class \_PageOneState extends State<PageOne> {

  List data = [];

  bool loading = false;

  @override

  Widget build(BuildContext context) {

    return Scaffold(

      appBar: AppBar(title: Text("HTTP Example")),

      body: ListView(

        children: [

          Container(

            margin: EdgeInsets.all(10),

            child: MaterialButton(

              onPressed: () async {

                loading = true;

                setState(() {});

                var response = await get(

                  Uri.parse("https://jsonplaceholder.typicode.com/posts"),

                );

                data.addAll(jsonDecode(response.body));

                loading = false;

                setState(() {});

              },

              color: Colors.red,

              textColor: Colors.white,

              child: Text("HTTP Request"),

            ),

            // response Card using list generate

          ),

          if (loading) Center(child: CircularProgressIndicator()),

          ...List.generate(

            data.length,

            (index) => Card(

              child: ListTile(

                contentPadding: EdgeInsets.all(10),

                title: Text(

                  data[index]['title'],

                  style: TextStyle(fontWeight: FontWeight.bold),

                ),

                subtitle: Text(data[index]['body']),

              ),

            ),

          ),

        ],

      ),

    );

  }

}

This is chat gpt description :

**🧠 Step-by-step breakdown**

**1️⃣ Importing packages**

import 'package:flutter/material.dart';

import 'package:http/http.dart';

import 'dart:convert';

✅ You imported:

* **flutter/material.dart** → for UI widgets
* **http.dart** → for HTTP requests
* **dart:convert** → for decoding JSON text

Perfect combo for REST APIs in Flutter 💪

**2️⃣ Data variables**

List data = [];

bool loading = false;

✅ data → will hold your decoded JSON list  
✅ loading → used to show a loading spinner while fetching data

**3️⃣ Button click (the HTTP request)**

onPressed: () async {

loading = true;

setState(() {});

var response = await get(

Uri.parse("https://jsonplaceholder.typicode.com/posts"),

);

data.addAll(jsonDecode(response.body));

loading = false;

setState(() {});

},

Let’s break this:

* You start by setting loading = true → to show the spinner.
* await get(Uri.parse(...)) → sends the GET request.
* response.body gives JSON text.
* jsonDecode(response.body) turns JSON text into a **List of Maps**.
* data.addAll(...) → adds all posts to your data list.
* Finally, you set loading = false → spinner disappears.

✅ Clean, clear, and correct.

**4️⃣ Display section**

if (loading) Center(child: CircularProgressIndicator()),

...List.generate(

data.length,

(index) => Card(

child: ListTile(

title: Text(data[index]['title']),

subtitle: Text(data[index]['body']),

),

),

),

✅ This is **great**:

* Shows loading circle while waiting.
* Dynamically builds a list of cards from your API data.
* Displays title and body clearly.

👏 100% correct.

**⚙️ Small improvements (professional tips)**

**✅ 1. Clear data before refetching**

If you click the button twice, your list doubles (because of addAll).  
Better to clear it before reloading:

data.clear();

data.addAll(jsonDecode(response.body));

**✅ 2. Check response code for errors**

Always ensure request success before using the body:

if (response.statusCode == 200) {

data.clear();

data.addAll(jsonDecode(response.body));

} else {

print("Error: ${response.statusCode}");

}

**✅ 3. Use setState more cleanly**

You can wrap state updates inside one setState() instead of two:

onPressed: () async {

setState(() => loading = true);

var response = await get(Uri.parse("https://jsonplaceholder.typicode.com/posts"));

if (response.statusCode == 200) {

var result = jsonDecode(response.body);

setState(() {

data = result;

loading = false;

});

} else {

setState(() => loading = false);

}

},

This avoids multiple UI rebuilds unnecessarily.

**✅ 4. Optional: Show an error message**

You can add a message if the API fails:

if (response.statusCode != 200) {

ScaffoldMessenger.of(context).showSnackBar(

SnackBar(content: Text('Failed to fetch data')),

);

}

**🧩 Summary of what you learned**

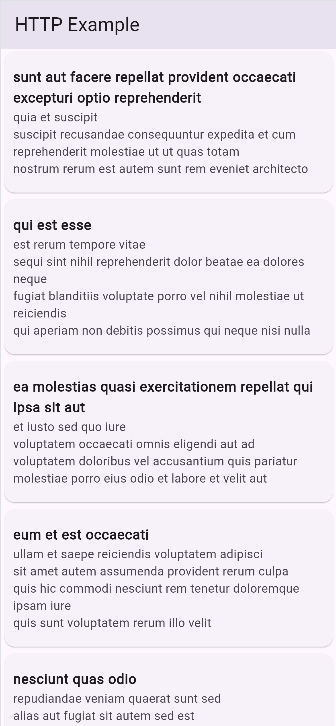
| **Concept** | **Meaning** | **Example** |
| --- | --- | --- |
| get() | Send HTTP GET request | await get(Uri.parse(...)) |
| response.body | Raw JSON text | jsonDecode(response.body) |
| jsonDecode() | Convert text → List or Map | List data = jsonDecode(...) |
| setState() | Refresh the UI | setState(() => loading = true) |
| List.generate() | Build widgets dynamically | List.generate(data.length, ...) |

A screenshot of a phone

AI-generated content may be incorrect.

# FutureBuilder

Let’s remove button from the previous example, meaning , lodaing start auto when you open the app .



import 'package:flutter/material.dart';

import 'package:http/http.dart';

// json

import 'dart:convert';

class PageOne extends StatefulWidget {

  @override

  \_PageOneState createState() => \_PageOneState();

}

class \_PageOneState extends State<PageOne> {

  List data = [];

  bool loading = true;

  getData() async {

    var response = await get(

      Uri.parse("https://jsonplaceholder.typicode.com/posts"),

    );

    data = jsonDecode(response.body);

    loading = false;

    setState(() {});

  }

  @override

  void initState() {

    super.initState();

    getData();

  }

  @override

  Widget build(BuildContext context) {

    return Scaffold(

      appBar: AppBar(title: Text("HTTP Example")),

      body: ListView(

        children: [

          if (loading) Center(child: CircularProgressIndicator()),

          ...List.generate(

            data.length,

            (index) => Card(

              child: ListTile(

                contentPadding: EdgeInsets.all(10),

                title: Text(

                  data[index]['title'],

                  style: TextStyle(fontWeight: FontWeight.bold),

                ),

                subtitle: Text(data[index]['body']),

              ),

            ),

          ),

        ],

      ),

    );

  }

}

But there is really something great , you can do all that so easyy , wait wait for it .

import 'package:flutter/material.dart';

import 'package:http/http.dart';

// json

import 'dart:convert';

class PageOne extends StatefulWidget {

  @override

  \_PageOneState createState() => \_PageOneState();

}

class \_PageOneState extends State<PageOne> {

  Future<List> getData() async {

    var response = await get(

      Uri.parse("https://jsonplaceholder.typicode.com/posts"),

    );

    List responseBody = jsonDecode(response.body);

    return responseBody;

  }

  @override

  Widget build(BuildContext context) {

    return Scaffold(

      appBar: AppBar(title: Text("HTTP Example")),

      body: FutureBuilder<List>(

        future: getData(),

        builder: (context, snapshot) {

          if (snapshot.connectionState == ConnectionState.waiting) {

            return Center(child: CircularProgressIndicator());

          }

          return ListView.builder(

            itemCount: snapshot.data!.length,

            itemBuilder: (context, index) {

              return ListTile(

                title: Text(snapshot.data![index]['title'] ?? ''),

                subtitle: Text(snapshot.data![index]['body'] ?? ''),

              );

            },

          );

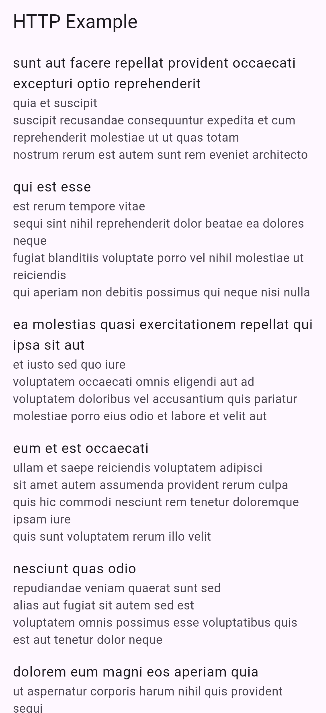
        },

      ),

    );

  }

}



I did all wait a few lines of code hh, loading and all , so let’s start .

## 🧠 1. What is a Future?

Before FutureBuilder, you must understand **Future** itself.

A **Future** in Dart represents a **value that will be available later** — like an HTTP request or database query.

***Future<String> getMessage() async {***

***await Future.delayed(Duration(seconds: 2));***

***return "Hello from the future!";***

***}***

This function doesn’t return the value immediately — it returns a **promise** (Future) that will complete after 2 seconds.

## 🧱 2. What is FutureBuilder?

FutureBuilder is a **Flutter widget** that **automatically listens** to a Future and **rebuilds the UI** depending on its state.

It’s perfect for:

* HTTP requests (fetching data from internet)
* Reading from a database
* Any async operation (file loading, waiting for a delay, etc.)

## Syntax :

***FutureBuilder<T>(***

***future: myFuture,***

***builder: (BuildContext context, AsyncSnapshot<T> snapshot) {***

***// return widget based on snapshot***

***},***

***)***

#### 📌 Parameters:

|  |  |
| --- | --- |
| Parameter | Description |
| future | The asynchronous operation to execute (Future<T>) |
| builder | Function that builds the UI based on the snapshot |
| initialData | (optional) Data to show before the Future finishes |

## 🧩 Snapshot Explained

Inside the builder, you get a snapshot — an object that tells you **the current state** of the Future.

### ✳️ Common snapshot properties:

|  |  |
| --- | --- |
| Property | Description |
| snapshot.connectionState | Shows if the Future is loading, done, etc. |
| snapshot.hasData | True when data is available |
| snapshot.hasError | True when an error occurred |
| snapshot.data | The actual result (if available) |
| snapshot.error | The actual error (if any) |

## 🔄 Connection States

|  |  |  |
| --- | --- | --- |
| State | Description | Typical UI |
| ConnectionState.none | No future yet | Empty widget |
| ConnectionState.waiting | Future running | CircularProgressIndicator() |
| ConnectionState.active | Still running (like a stream) | Optional |
| ConnectionState.done | Completed | Show result or error |

🧭 Summary

|  |  |
| --- | --- |
| Concept | Explanation |
| Future | A value that will be available later |
| FutureBuilder | Widget that rebuilds UI based on Future state |
| snapshot | Object containing data, error, or connection state |
| ConnectionState | none, waiting, active, done |
| Best use | Async UI updates (HTTP, DB, etc.) |

# Dropdown list

In this lesson, we gonna talk about this:

https://pub.dev/packages/drop\_down\_list

dowland it , and let’s begin :

okay durring this dropdown package a really faced many problems and learned a lot , so that I can understand it .

## What is a generic?

A **generic** is a way to write **code that works with any type**, but **still keeps type safety**.

Think of it like a **box that can hold any type of item**, but you tell the compiler what type it will hold.

**Example (in real life analogy):**

* A box labeled Shoes Box → can only hold shoes.
* A box labeled Book Box → can only hold books.

Generics in code do the same:

***List<String> names = ['Ali', 'Sara'];***

***List<int> numbers = [1, 2, 3];***

***List<String> names = ['Ali', 'Sara'];***

***List<int> numbers = [1, 2, 3];***

* List<String> → List of Strings only
* List<int> → List of integers only
* You **cannot accidentally put an int into List<String>**, Dart will give an error. ✅

## Generics in the dropdown

***DropDownState<String>(***

***dropDown: DropDown<String>(***

***data: myData,***

***onSelected: ...***

***),***

***)***

* <String> tells the dropdown:
  + “The items in this dropdown are **strings**.”
  + SelectedListItem<String> will carry **strings**, not any other type.

If you had DropDownState<int> → it would only accept integers.

⚡ Without <String>, Dart does **not know what type the dropdown holds**, so you get **errors** when trying to map the selected items.

## DropDown

import 'package:flutter/material.dart';

import 'package:http/http.dart';

// json

import 'dart:convert';

import 'package:drop\_down\_list/drop\_down\_list.dart';

import 'package:drop\_down\_list/model/selected\_list\_item.dart';

class PageOne extends StatefulWidget {

  @override

  \_PageOneState createState() => \_PageOneState();

}

class \_PageOneState extends State<PageOne> {

  List<SelectedListItem<String>> myData = [

    SelectedListItem<String>(data: 'Tokyo'),

    SelectedListItem<String>(data: 'New York'),

    SelectedListItem<String>(data: 'London'),

    SelectedListItem<String>(data: 'Paris'),

    SelectedListItem<String>(data: 'Berlin'),

    SelectedListItem<String>(data: 'Rabat'),

    SelectedListItem<String>(data: 'Tangier'),

    SelectedListItem<String>(data: 'Casablanca'),

    SelectedListItem<String>(data: 'Marrakesh'),

  ];

  void showDropDownList() {

    DropDownState(

      dropDown: DropDown<String>(

        data: myData,

        onSelected: (List<SelectedListItem<String>> selectedItems) {

          // Handle the selected items

        },

      ),

    ).showModal(context);

  }

  @override

  Widget build(BuildContext context) {

    return Scaffold(

      appBar: AppBar(title: Text("drop\_down\_list Example")),

      body: Center(

        child: ElevatedButton(

          onPressed: () {

            showDropDownList();

          },

          child: Text("Show Dropdown"),

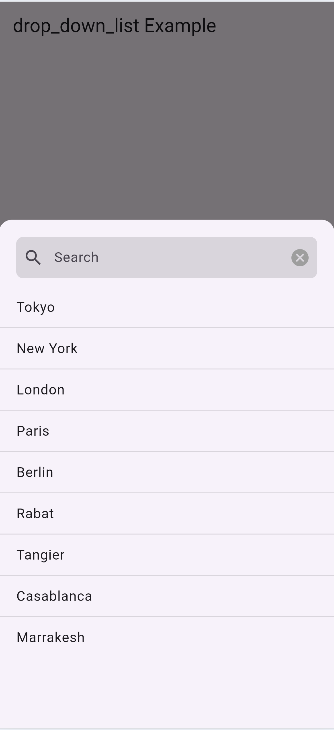
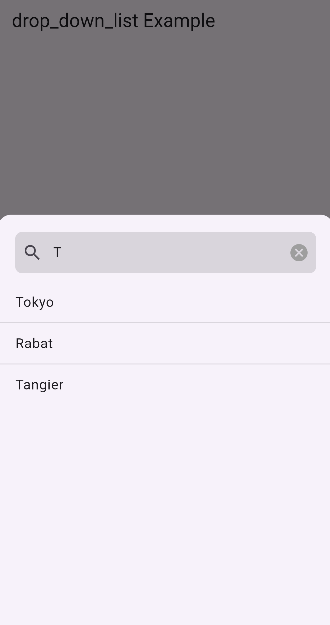
        ),

      ),

    );

  }

}



Now we will , start , we have just take the list form , so let’s see how it works code extra …

**showModal()**

* **Purpose:** Displays the dropdown **visibly as a modal bottom sheet**.
* **Usage:**
* DropDownState<String>(dropDown: myDropDown).showModal(context);
* **Important:**
  + Requires context to attach to the widget tree.
  + Can be customized with shapeBorder, backgroundColor, isDismissible.

**Summary:**

* showModal() = **actually opens the dropdown**.
* Everything else (DropDownState and DropDown) prepares the **data and UI**.

He other properties just mouse\_in in vscode and you will discover them .

# SharedPrefrences

**🧠 1. What is SharedPreferences?**

**SharedPreferences** is a way to **save small data locally** on the device — like:

* username
* theme (dark/light)
* login state
* language selection, etc.

👉 It’s like a **small key–value storage** that stays even after closing the app.

class \_PageOneState extends State<PageOne> {

  String? name;

  @override

  Widget build(BuildContext context) {

    return Scaffold(

      appBar: AppBar(title: Text("shared Prefrences")),

      body: Column(

        children: [

          MaterialButton(

            minWidth: 500,

            color: Colors.amber,

            onPressed: () {

              name = "Bilal";

            },

            child: Text("Set Name"),

          ),

          MaterialButton(

            minWidth: 500,

            color: Colors.blue,

            onPressed: () {

              print(name);

            },

            child: Text("Print Name"),

          ),

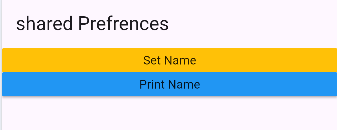
        ],

      ),

    );

  }

}



Okay As we know if we print Name before setting in we will se null but after setting it we will see bilal , but after reloading we must re\_set the Name , sharedPrefrences will helps us to avoid this .

So to Use SharedPrefrences our devices need to connect to device storage and this operation takes time, so we will need to async/await .

onPressed: () async {

              SharedPreferences prefs = await SharedPreferences.getInstance();

              prefs.setString("name", "value");

            },

And for getting it :

SharedPreferences prefif = await SharedPreferences.getInstance();

              name = prefif.getString("name");

              print(name);

see now the full code :

import 'package:flutter/material.dart';

import 'package:shared\_preferences/shared\_preferences.dart';

class PageOne extends StatefulWidget {

  @override

  \_PageOneState createState() => \_PageOneState();

}

class \_PageOneState extends State<PageOne> {

  String? name;

  @override

  Widget build(BuildContext context) {

    return Scaffold(

      appBar: AppBar(title: Text("shared Prefrences")),

      body: Column(

        children: [

          MaterialButton(

            minWidth: 500,

            color: Colors.amber,

            onPressed: () async {

              SharedPreferences prefs = await SharedPreferences.getInstance();

              prefs.setString("name", "Bilal");

            },

            child: Text("Set Name"),

          ),

          MaterialButton(

            minWidth: 500,

            color: Colors.blue,

            onPressed: () async {

              SharedPreferences prefif = await SharedPreferences.getInstance();

              name = prefif.getString("name");

              print(name);

            },

            child: Text("Print Name"),

          ),

        ],

      ),

    );

  }

}

⚙️ 2. How to get the instance

***final prefs = await SharedPreferences.getInstance();***

This gives you access to all the methods below 👇

|  |  |  |
| --- | --- | --- |
| Method | Type | Description |
| setBool(String key, bool value) | Future | Save a boolean |
| setInt(String key, int value) | Future | Save an integer |
| setDouble(String key, double value) | Future | Save a decimal number |
| setString(String key, String value) | Future | Save a text |
| setStringList(String key, List<String> value) | Future | Save a list of strings |
| getBool(String key) | bool? | Read a boolean |
| getInt(String key) | int? | Read an integer |
| getDouble(String key) | double? | Read a decimal |
| getString(String key) | String? | Read a text |
| getStringList(String key) | List? | Read a string list |
| containsKey(String key) | bool | Check if a key exists |
| remove(String key) | Future | Delete one item |
| clear() | Future | Delete all items |
| getKeys() | Set | Get all keys currently stored |
| reload() | Future | Reload values from disk (used if data changes outside your app) |

Okay let’s practice this, I have a small idea hh :

# Project and Tips and shortcut

Simple trick to memorize how to create stateless widget

|  |  |  |
| --- | --- | --- |
| Step | Meaning | Example |
| C | **Class** → starts with class MyWidget extends StatelessWidget | class MyWidget extends StatelessWidget |
| B | **Build** → always has Widget build(BuildContext context) | Widget build(BuildContext context) |
| R | **Return** → must return one widget | return Scaffold(...) |

So:  
👉 **C-B-R = Class → Build → Return**

If you remember this 3-step pattern, you can write it from scratch anytime.

## Shortcut

First to apply each abbr , you have to press **Enter / Tab**

**Like :**

Stl : create a statless widget .

|  |  |  |
| --- | --- | --- |
| Shortcut | Expands to | Description |
| stl | StatelessWidget | Creates a full Stateless widget class |
| stf | StatefulWidget | Creates a Stateful widget with its State class |
| stanim | StatefulWidget with AnimationController | Used for animations |
| matapp | MaterialApp | Creates a MaterialApp root widget |
| scaf | Scaffold | Creates a Scaffold widget quickly |
| cont | Container | Creates a Container |
| col | Column | Creates a Column widget |
| row | Row | Creates a Row widget |
| cen | Center | Creates a Center widget |
| text | Text() | Creates a Text widget |
| icon | Icon() | Creates an Icon widget |
| img | Image.asset() | Adds an image from assets |
| btn | ElevatedButton | Creates a button quickly |
| listv | ListView | Creates a scrollable list |
| exp | Expanded | Adds an Expanded widget |
| pad | Padding | Adds Padding widget |
| alig | Align | Aligns a child |
| card | Card | Creates a Card widget |
| stack | Stack | Creates a Stack (overlapping layout) |

Don’t forget you have to press Enter + Tab , not only tab !!!!!!!!!

## VS Code General Shortcuts (Windows/Linux)

|  |  |
| --- | --- |
| Shortcut | Action |
| Ctrl + Space | Show code suggestions (autocomplete) |
| Ctrl + . | Quick fix / Import missing package |
| ssssAlt + Shift + F | Format code |
| Ctrl + S | Save file |
| Ctrl + / | Comment or uncomment line |
| Ctrl + D | Select next occurrence of a word |
| Alt + ↑ / ↓ | Move line up or down |
| Shift + Alt + ↓ | Duplicate line down |
| Ctrl + P | Quick open file |
| Ctrl + Shift + P | Command palette |
| Ctrl + B | Toggle side bar |
| Ctrl + `` | Toggle terminal |
| Ctrl + K + C | Block comment |
| Ctrl + K + U | Uncomment block |

## Table

You can make a table ( rows \* Columns) using the Table widget.

It works just like an HTML table – you define :

* Rows 🡺 with TableRow
* Cells 🡺 widgets (usually Text , Icon , etc )

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> {

  @override

  Widget build(BuildContext context) {

    return MaterialApp(

      debugShowCheckedModeBanner: false,

      home: Scaffold(

        appBar: AppBar(title: const Text('User Table')),

        body: Padding(

          padding: const EdgeInsets.all(8.0),

          child: Table(

            border: TableBorder.all(),

            children: [

              TableRow(

                decoration: BoxDecoration(

                  color: const Color.fromARGB(255, 0, 162, 255),

                ),

                children: [

                  HeaderCell(text: "Username"),

                  HeaderCell(text: "Age"),

                  HeaderCell(text: "is Student"),

                  HeaderCell(text: "Favorite"),

                ],

              ),

              TableRow(

                children: [

                  const Padding(

                    padding: EdgeInsets.all(8.0),

                    child: Text("Alice"),

                  ),

                  const Padding(

                    padding: EdgeInsets.all(8.0),

                    child: Text("23"),

                  ),

                  const Padding(

                    padding: EdgeInsets.all(8.0),

                    child: Text("Yes"),

                  ),

                  Container(

                    color: Colors.pink,

                    padding: const EdgeInsets.all(8.0),

                    child: const Text("Ice Cream"),

                  ),

                ],

              ),

              TableRow(

                children: [

                  const Padding(

                    padding: EdgeInsets.all(8.0),

                    child: Text("Bob"),

                  ),

                  const Padding(

                    padding: EdgeInsets.all(8.0),

                    child: Text("30"),

                  ),

                  const Padding(

                    padding: EdgeInsets.all(8.0),

                    child: Text("No"),

                  ),

                  Container(

                    color: Colors.lightGreen,

                    padding: const EdgeInsets.all(8.0),

                    child: const Text("Pizza"),

                  ),

                ],

              ),

              TableRow(

                children: [

                  const Padding(

                    padding: EdgeInsets.all(8.0),

                    child: Text("Charlie"),

                  ),

                  const Padding(

                    padding: EdgeInsets.all(8.0),

                    child: Text("28"),

                  ),

                  const Padding(

                    padding: EdgeInsets.all(8.0),

                    child: Text("Yes"),

                  ),

                  Container(

                    color: Colors.lightBlue,

                    padding: const EdgeInsets.all(8.0),

                    child: const Text("Sushi"),

                  ),

                ],

              ),

              TableRow(

                children: [

                  const Padding(

                    padding: EdgeInsets.all(8.0),

                    child: Text("David"),

                  ),

                  const Padding(

                    padding: EdgeInsets.all(8.0),

                    child: Text("35"),

                  ),

                  const Padding(

                    padding: EdgeInsets.all(8.0),

                    child: Text("No"),

                  ),

                  Container(

                    color: Colors.orange,

                    padding: const EdgeInsets.all(8.0),

                    child: const Text("Burger"),

                  ),

                ],

              ),

            ],

          ),

        ),

      ),

    );

  }

}

class HeaderCell extends StatelessWidget {

  final String text;

  final Color color;

  const HeaderCell({super.key, required this.text, this.color = Colors.amber});

  @override

  Widget build(BuildContext context) {

    return Container(

      alignment: Alignment.center,

      color: color,

      padding: const EdgeInsets.all(8.0),

      child: Text(

        text,

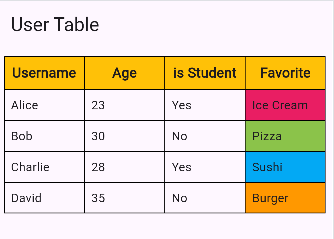
        style: const TextStyle(fontWeight: FontWeight.bold, fontSize: 16),

      ),

    );

  }

}



As you see we maked a table .

No let’s make it auto .

class \_MyAppState extends State<MyApp> {

  List<List<String>> users = [

    ["Alice", "23", "Yes", "Ice Cream"],

    ["Bob", "30", "No", "Pizza"],

    ["Charlie", "28", "Yes", "Sushi"],

    ["David", "35", "No", "Burger"],

    ["Eve", "22", "Yes", "Salad"],

  ];

  @override

  Widget build(BuildContext context) {

    return MaterialApp(

      debugShowCheckedModeBanner: false,

      home: Scaffold(

        appBar: AppBar(title: const Text('User Table')),

        body: Padding(

          padding: const EdgeInsets.all(8.0),

          child: Table(

            border: TableBorder.all(),

            children: [

              TableRow(

                decoration: BoxDecoration(

                  color: const Color.fromARGB(255, 0, 162, 255),

                ),

                children: [

                  HeaderCell(text: "Username"),

                  HeaderCell(text: "Age"),

                  HeaderCell(text: "is Student"),

                  HeaderCell(text: "Favorite"),

                ],

              ),

              ...users.map(

                (user) => TableRow(children: user.map((e) => Text(e)).toList()),

              ),

            ],

          ),

        ),

      ),

    );

  }

}

class HeaderCell extends StatelessWidget {

  final String text;

  final Color color;

  const HeaderCell({super.key, required this.text, this.color = Colors.amber});

  @override

  Widget build(BuildContext context) {

    return Container(

      alignment: Alignment.center,

      color: color,

      padding: const EdgeInsets.all(8.0),

      child: Text(

        text,

        style: const TextStyle(fontWeight: FontWeight.bold, fontSize: 16),

      ),

    );

  }

}

Remember map :

**🧠 Why this works**

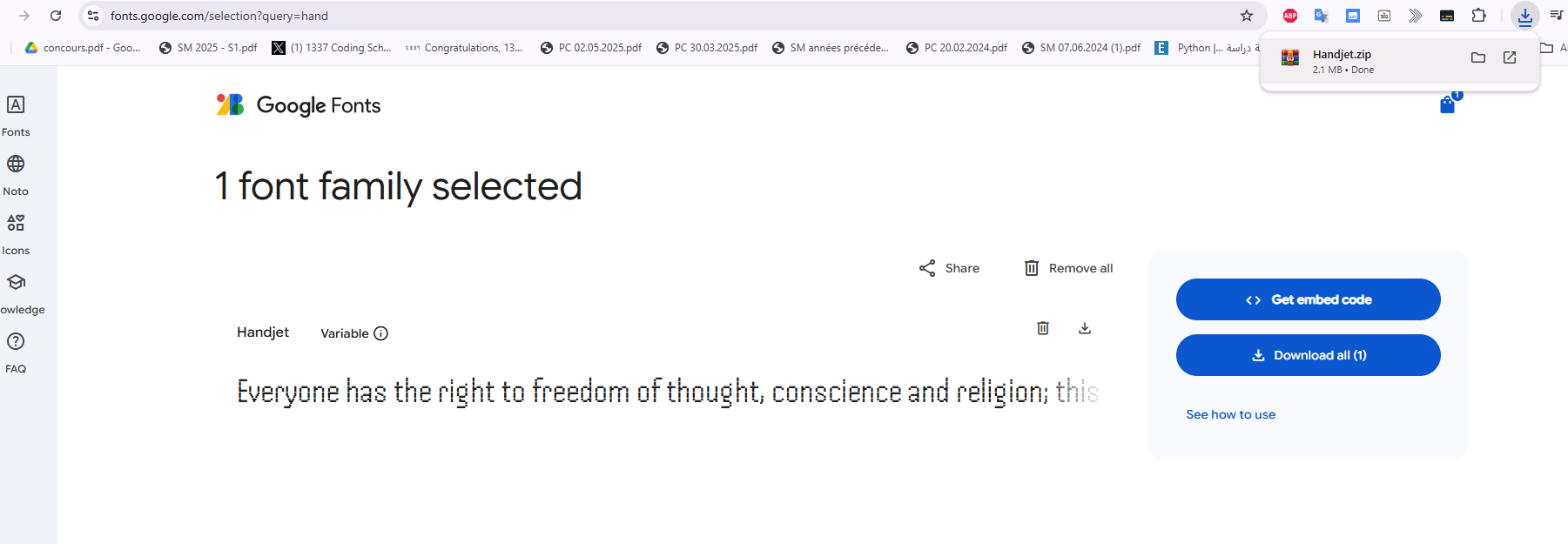
1. users.map(...) → loops through each row (e.g. ["Alice", "23", "Yes", "Ice Cream"]).
2. user.map(...) → loops through each string inside that row.
3. For each e, we return a **Cell widget**.
4. .toList() converts it to a list of widgets for the TableRow.
5. The spread operator ... expands the mapped list of rows into the children: of the Table.

# Custom Font

Now , we will learn how customize font and choose the font that we wants , let’s get started ;

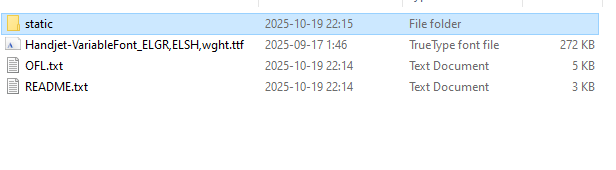
First go to <https://fonts.google.com/>

Choose you fonts



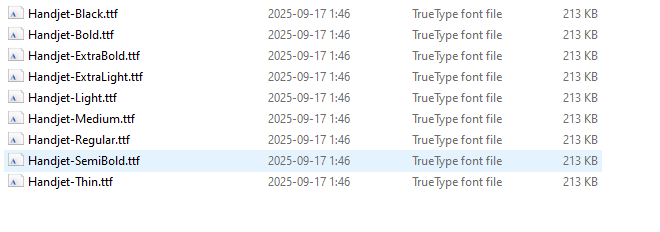
I choosed this fonts , and I dowland it from the arrow, as you see as get a zip file .

After unziping :



Just ignore .txt files , there just liscence and ……

Focus just on .ttf file, it is the font that we want , if you want all the forms of the font, go to static folder :

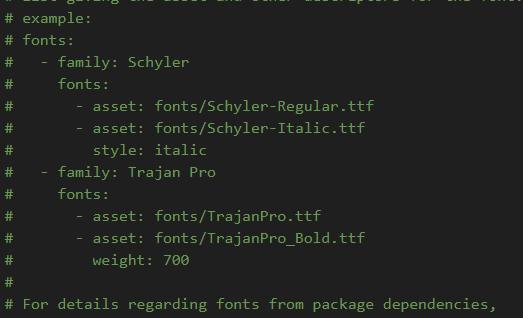


As you see it inculdes all .

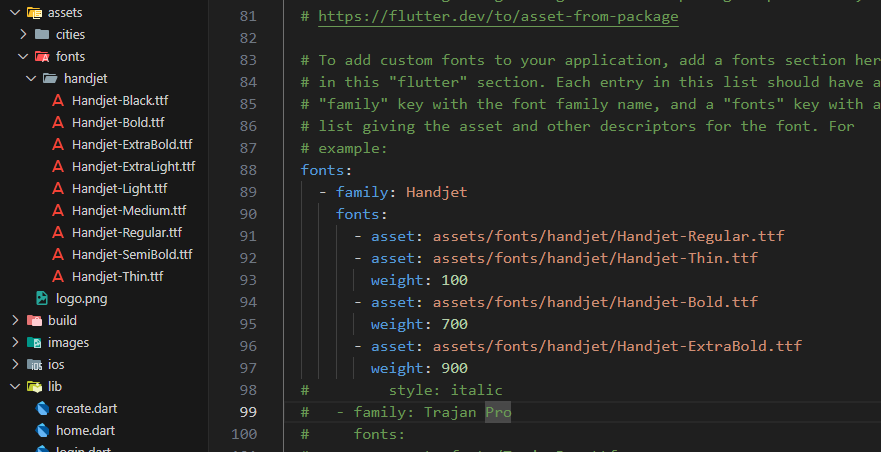
So go to vs code create font folder then Handjet folder and paste all this fonts files .

So now go to vs code .

**Pubspec.yaml**



Incoment carefully



As you see I added just 3 font type .

Now how to use it :

title: Text(

          "Create Your account",

          style: TextStyle(fontSize: 30, fontFamily: "Handjet"),

        ),

And to get thin , all that we have to do is specify the weight giving to it .

Example : Thin ==🡺 100

  style: TextStyle(

            fontSize: 30,

            fontFamily: "Handjet",

            fontWeight: FontWeight.w900,

          ),

So make WXXX (w100, w700, w900 ….) specify the number .

If we want to applly this font to all our app.

In **MaterialApp** , add this :

MaterialApp(

      debugShowCheckedModeBanner: false,

      theme: ThemeData(fontFamily: "Handjet"),

A **TTF file extension** stands for **TrueType Font**, a digital font format developed jointly by Apple and Microsoft.

An **OTF file** is an **OpenType font** file, a modern, cross-platform font format developed by Adobe and Microsoft that extends the older TrueType format

# Theme

**1️⃣ What is a Theme?**

A **Theme** in Flutter defines **the global look and feel** of your app.  
It controls things like:

* Fonts
* Colors
* Button styles
* AppBar styles
* Text styles

👉 It’s like a “style guide” that your whole app follows automatically.

**AppBarTheme**

**🧠 What it does:**

AppBarTheme controls **how all AppBars** in your app look —  
their **background color**, **title text style**, **icons color**, **elevation**, etc.

So instead of styling every AppBar separately,  
you define it **once** in your ThemeData, and it applies to all automatically.

==

**extTheme**

**🧠 What it does:**

TextTheme defines **default text styles** for all types of text in your app:

* Titles
* Body texts
* Labels
* Buttons  
  etc.

So you can define different sizes, weights, and colors for each.

theme: ThemeData(

        appBarTheme: AppBarTheme(backgroundColor: Colors.red),

        textTheme: TextTheme(

          bodySmall: TextStyle(color: Colors.blue, fontSize: 14),

          bodyMedium: TextStyle(color: Colors.black, fontSize: 16),

          bodyLarge: TextStyle(

            color: Colors.deepPurple,

            fontSize: 18,

            fontWeight: FontWeight.bold,

          ),

        ),

      ),

For textTheme there others

TextTheme({  
TextStyle? displayLarge,  
TextStyle? displayMedium,  
TextStyle? displaySmall,  
TextStyle? headlineLarge,  
TextStyle? headlineMedium,  
TextStyle? headlineSmall,  
TextStyle? titleLarge,  
TextStyle? titleMedium,  
TextStyle? titleSmall,  
TextStyle? bodyLarge,  
TextStyle? bodyMedium,  
TextStyle? bodySmall,  
TextStyle? labelLarge,  
TextStyle? labelMedium,  
TextStyle? labelSmall,  
})

For using theme go to a text and then :

title: Text(

          "Create Your account",

          style: Theme.of(context).textTheme.bodyMedium,

        ),

It so easy .

📝 **Note:**  
Unlike ThemeData or AppBarTheme, the TextTheme styles **don’t apply automatically** to every Text widget.

You must **explicitly mention** them inside the style property, like this 👇

***Text(***

***"Hello Flutter!",***

***style: Theme.of(context).textTheme.titleLarge,***

***)***

Otherwise, the text will just use Flutter’s **default text style**, not your custom

# Flutter launcher icons

To modify launcher icon , we have two method

## Manuall Method :

**⚙️ Manual Method — Change App Icon by Hand**

We’ll handle it separately for **Android** 🟢 and **iOS** 🍏.

**🟢 1️⃣ Android**

**📁 Step 1 — Prepare your icon**

* Create your app icon image (ideally **1024×1024 PNG**).
* Make sure it has **no transparent background** (Android adaptive icons use solid backgrounds).

You can use https://appicon.co/ or https://makeappicon.com/  
These tools automatically generate all Android icon sizes for you.

They will create folders like:

mipmap-mdpi/

mipmap-hdpi/

mipmap-xhdpi/

mipmap-xxhdpi/

mipmap-xxxhdpi/

Each folder will contain an icon like:

ic\_launcher.png

ic\_launcher\_round.png

**📂 Step 2 — Replace existing icons**

Go to your Flutter project directory:

*android/app/src/main/res/*

You’ll see folders like:

***mipmap-mdpi/***

***mipmap-hdpi/***

***mipmap-xhdpi/***

***mipmap-xxhdpi/***

***mipmap-xxxhdpi/***

Replace the default icons inside **each folder** with your new generated ones.

File names **must be identical**:

ic\_launcher.png

ic\_launcher\_round.png

**🧩 Step 3 — Check AndroidManifest.xml**

Open:

android/app/src/main/AndroidManifest.xml

Ensure the line below exists inside <application>:

***android:icon="@mipmap/ic\_launcher"***

✅ This tells Android which icon to use.

**🧪 Step 4 — Clean and rebuild**

Run these commands:

**flutter clean**

**flutter pub get**

**flutter run**

You should now see your **new icon** on Android.

**🍏 2️⃣ iOS**

**📁 Step 1 — Generate iOS icon set**

Use the same tools (appicon.co)  
and choose **iOS** this time.  
It will generate a folder called:

AppIcon.appiconset

**📂 Step 2 — Replace in Xcode project**

Go to your Flutter project:

ios/Runner/Assets.xcassets/AppIcon.appiconset/

Replace **everything inside** this folder with your new icons.

Make sure the file names match (they usually are like Icon-App-60x60@2x.png, etc.).

**🧩 Step 3 — Verify in Xcode (optional)**

If you open the iOS project in Xcode (ios/Runner.xcworkspace),  
select the **Runner target → General tab → App Icons and Launch Images**,  
you’ll see your new icon preview.

**🧪 Step 4 — Rebuild for iOS**

Run:

**flutter clean**

**flutter pub get**

**flutter run**

✅ Your app should now display the new icon on iOS too.

## Auto

Here we will use a packages

**Pubspec.yaml**



💡 **Tip:**  
It’s a **dev dependency**, not a runtime dependency.  
You only need it **when generating icons**, not in the released app.

This package will help us to generate the logo images in diff sized for ios and android ,

You have just to specify path like this :

# -------------------------

# Launcher icon configuration

# -------------------------

flutter\_icons:

  android: true

  ios: true

  image\_path: "assets/logo.png"

**1️⃣ flutter\_icons:**

* This is the **root key** for the launcher icon configuration.
* It tells the flutter\_launcher\_icons package:

“Here are the instructions for generating app icons.”

**2️⃣ android: true**

* Means: **Generate launcher icons for Android**.
* The package will automatically create all required sizes for Android’s mipmap- folders (mdpi, hdpi, xhdpi, xxhdpi, xxxhdpi).
* Without this line, Android icons would not be generated.

**3️⃣ ios: true**

* Means: **Generate launcher icons for iOS**.
* The package will create the proper sizes for iOS (AppIcon.appiconset) to support all device resolutions.
* Without this, iOS icons remain the default Flutter icon.

**4️⃣ image\_path: "assets/logo.png"**

* This is the **path to your source icon image**.
* Usually a **square PNG** (ideally 512×512 or 1024×1024).
* The package will **resize this image** into all necessary icon sizes for Android and iOS.

💡 Note: You must have this file in your project under assets/logo.png.

Then run the package :

**Run the icon generator**

In your terminal, run:

***flutter pub run flutter\_launcher\_icons***

then rebuild please :

**flutter clean**

**flutter pub get**

**flutter run**

# Geolocator

Flutter geolocator package in flutter is used to get **the device’s location** (latitude, longitude, etc), **handle permissions**, and even **calulate distances beween two points** ;

## Features :

* Get the last known location;
* Get the current location of the device;
* Get continuous location updates;
* Check if location services are enabled on the device;
* Calculate the distance (in meters) between two geocoordinates;
* Calculate the bearing between two geocoordinates;

## Installing

Take : <https://pub.dev/packages/geolocator/install>

After installing it folow what here in usage : <https://pub.dev/packages/geolocator>

1. Add the following to your "gradle.properties" file:

***android.useAndroidX=true***

***android.enableJetifier=true***

For me I find them there I did not change nothing :

org.gradle.jvmargs=-Xmx8G -XX:MaxMetaspaceSize=4G -XX:ReservedCodeCacheSize=512m -XX:+HeapDumpOnOutOfMemoryError

android.useAndroidX=true

android.enableJetifier=true

1. Make sure you set the compileSdkVersion in your "android/app/build.gradle" file to 35:

***android {***

***compileSdkVersion 35***

***...***

***}***

For me I find it , like this , so I will change it :

***android {***

***namespace = "com.example.flutter\_application\_1"***

***compileSdk = flutter.compileSdkVersion***

***ndkVersion = flutter.ndkVersion***

it become like this :

***android {***

***namespace = "com.example.flutter\_application\_1"***

***compileSdk = 35***

***ndkVersion = flutter.ndkVersion***

now let add permissions

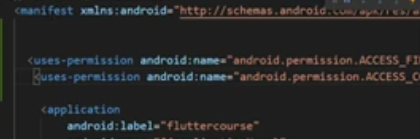
### Permissions

AndroidManifest.xml file (located under android/app/src/main)

***<uses-permission android:name="android.permission.ACCESS\_FINE\_LOCATION" />***

***<uses-permission android:name="android.permission.ACCESS\_COARSE\_LOCATION" />***

Note : Every permission should be added between manifest and application :



Like this :

<manifest xmlns:android="http://schemas.android.com/apk/res/android">

  <uses-permission android:name="android.permission.ACCESS\_FINE\_LOCATION" />

  <uses-permission android:name="android.permission.ACCESS\_COARSE\_LOCATION" />

    <application

## Usage :

So to use this , we have 2 important things :

* 1. Location service is enable In the device
  2. Location app, meaning we give the permission to use this service

After some problem , now am great :

getCurrentLocation() async {

    bool serviceEnabled;

    LocationPermission permission;

    serviceEnabled = await Geolocator.isLocationServiceEnabled();

    if (!serviceEnabled) {

      print("Please enable location services");

    } else {

      print("Location services are enabled");

    }

  }

I think it super easy ,

Example: when I turn off gps in my phone :



Next step : is permission :

getCurrentLocation() async {

    bool serviceEnabled;

    LocationPermission permission;

    serviceEnabled = await Geolocator.isLocationServiceEnabled();

    if (!serviceEnabled) {

      print("Please enable location services");

    } else {

      print("Location services are enabled");

    }

    permission = await Geolocator.checkPermission();

    if (permission == LocationPermission.denied) {

      permission = await Geolocator.requestPermission();

      if (permission == LocationPermission.denied) {

        return Future.error("Location Permission are denied");

      }

    }

    if (permission == LocationPermission.deniedForever) {

      return Future.error(

        "Location permissions are permanently denied, we cannot request permissions.",

      );

    }

    print(serviceEnabled);

    print(permission);

  }

Result :



Full code until now :

import 'package:flutter/material.dart';

import 'package:geolocator/geolocator.dart';

class HomePage extends StatefulWidget {

  const HomePage({super.key});

  @override

  HomePageState createState() => HomePageState();

}

class HomePageState extends State<HomePage> {

  getCurrentLocation() async {

    bool serviceEnabled;

    LocationPermission permission;

    serviceEnabled = await Geolocator.isLocationServiceEnabled();

    if (!serviceEnabled) {

      print("Please enable location services");

      return;

    } else {

      print("Location services are enabled");

    }

    permission = await Geolocator.checkPermission();

    if (permission == LocationPermission.denied) {

      permission = await Geolocator.requestPermission();

      if (permission == LocationPermission.denied) {

        return Future.error("Location Permission are denied");

      }

    }

    if (permission == LocationPermission.deniedForever) {

      return Future.error(

        "Location permissions are permanently denied, we cannot request permissions.",

      );

    }

    print(serviceEnabled);

    print(permission);

  }

  @override

  void initState() {

    getCurrentLocation();

    super.initState();

  }

  @override

  Widget build(BuildContext context) {

    return Scaffold(

      appBar: AppBar(title: const Text("geolocator Package")),

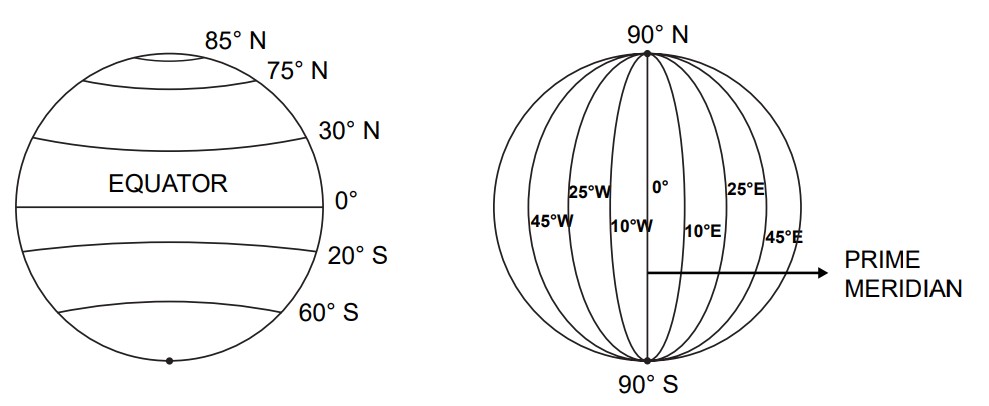
      body: Container(),

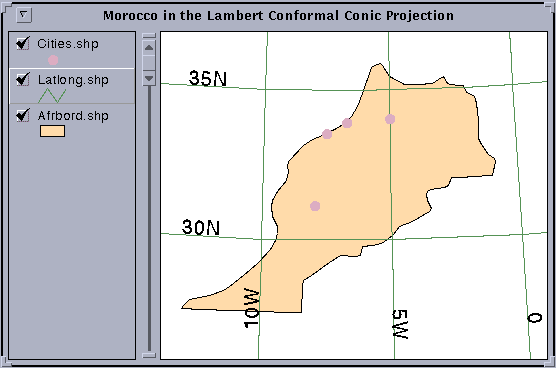
    );

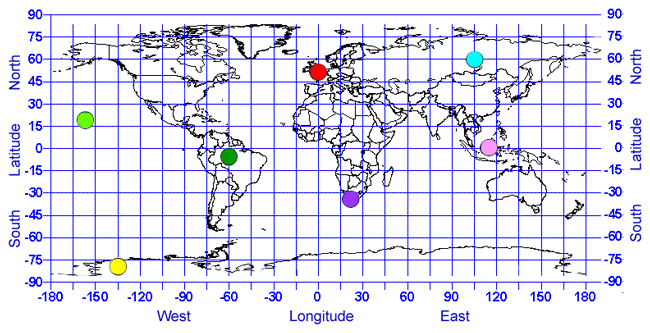
  }

}

Now we will speak about

Each location in the world can be precisely identified using **latitude** and **longitude** coordinates. Latitude indicates how far north or south a place is from the Equator, while longitude shows how far east or west it is from the Prime Meridian. You can find the latitude and longitude of any place using websites like **latlong.net**, **gps-coordinates.net**, or by using map services like **Google Maps**. Simply click on a location, and the coordinates will be displayed, allowing you to know the exact position of cities, landmarks, or any point on Earth.





Here are the coordinates for **Tangier, Morocco**:

* **Latitude:** 35.7595° N
* **Longitude:** 5.8340° W

Some others :

 **Paris, France**

* Latitude: 48.8566° N
* Longitude: 2.3522° E

 **Tokyo, Japan**

* Latitude: 35.6895° N
* Longitude: 139.6917° E

 **New York City, USA**

* Latitude: 40.7128° N
* Longitude: 74.0060° W

Now we will see how to get the location :

getCurrentLocation() async {

    bool serviceEnabled;

    LocationPermission permission;

    serviceEnabled = await Geolocator.isLocationServiceEnabled();

    if (!serviceEnabled) {

      print("Please enable location services");

      return;

    } else {

      print("Location services are enabled");

    }

    permission = await Geolocator.checkPermission();

    if (permission == LocationPermission.denied) {

      permission = await Geolocator.requestPermission();

      if (permission == LocationPermission.denied) {

        return Future.error("Location Permission are denied");

      }

    }

    if (permission == LocationPermission.deniedForever) {

      return Future.error(

        "Location permissions are permanently denied, we cannot request permissions.",

      );

    } else if (permission == LocationPermission.whileInUse ||

        permission == LocationPermission.always) {

      print("Location permission granted");

      Position position = await Geolocator.getCurrentPosition();

      print("position: $position");

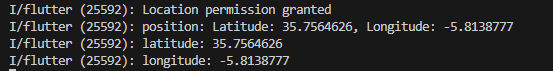
      print("latitude: ${position.latitude}");

      print("longitude: ${position.longitude}");

    }

  }

Result:



This will work if an delivery app want to know you location , but what if you want to see the delevery bike location , it changes it not fixed like yours , so in this case : we need a listener , let’s do it :

import 'package:flutter/material.dart';

import 'dart:async';

import 'package:geolocator/geolocator.dart';

class HomePage extends StatefulWidget {

  const HomePage({super.key});

  @override

  HomePageState createState() => HomePageState();

}

class HomePageState extends State<HomePage> {

  bool? serviceEnabled;

  LocationPermission? permission;

  StreamSubscription<Position>? positionStream;

  getCurrentLocation() async {

    serviceEnabled = await Geolocator.isLocationServiceEnabled();

    if (!serviceEnabled!) {

      print("Please enable location services");

      return;

    } else {

      print("Location services are enabled");

    }

    permission = await Geolocator.checkPermission();

    if (permission == LocationPermission.denied) {

      permission = await Geolocator.requestPermission();

      if (permission == LocationPermission.denied) {

        return Future.error("Location Permission are denied");

      }

    }

    if (permission == LocationPermission.deniedForever) {

      return Future.error(

        "Location permissions are permanently denied, we cannot request permissions.",

      );

    } else if (permission == LocationPermission.whileInUse ||

        permission == LocationPermission.always) {

      // listener for live location updates

      positionStream = Geolocator.getPositionStream().listen((

        Position position,

      ) {

        print(

          "Latitude: ${position.latitude} , Longitude: ${position.longitude}",

        );

      });

    }

  }

  @override

  void initState() {

    getCurrentLocation();

    super.initState();

  }

  @override

  void dispose() {

    if (positionStream != null) {

      positionStream!.cancel();

    }

    super.dispose();

  }

  @override

  Widget build(BuildContext context) {

    return Scaffold(

      appBar: AppBar(title: const Text("geolocator Package")),

      body: Container(),

    );

  }

}

Now let’s calculate distance between two points :

if (permission == LocationPermission.whileInUse ||

        permission == LocationPermission.always) {

      double distanceInMeters = Geolocator.distanceBetween(

        35.7595, // Tangier latitude

        -5.8339, // Tangier longitude

        50.8503, // Brussels latitude

        4.3517, // Brussels longitude

      );

      print("Distance: ${distanceInMeters / 1000} km");

      // Distance: 1867.5009752742283 km

    }

# Google map

Unfortunnatly , I don’t have a credit card .

# Tracking Google Map

# Geocoding

**1️⃣ Forward Geocoding**

👉 Convert a **place name or address → to latitude & longitude**

import 'package:geocoding/geocoding.dart';

***void main() async {***

***List<Location> locations = await locationFromAddress("Brussels, Belgium");***

***print("Latitude: ${locations.first.latitude}, Longitude: ${locations.first.longitude}");***

***}***

**📍 2️⃣ Reverse Geocoding**

👉 Convert **latitude & longitude → to a readable address**

import 'package:geocoding/geocoding.dart';

***void main() async {***

***List<Placemark> placemarks = await placemarkFromCoordinates(35.7595, -5.8339);***

***var place = placemarks.first;***

***print("${place.locality}, ${place.country}");***

***}***

💡 Summary table

| **Feature** | **What it does** | **Example** |
| --- | --- | --- |
| **Forward Geocoding** | Address → Coordinates | “Brussels” → (50.85, 4.35) |
| **Reverse Geocoding** | Coordinates → Address | (35.75, -5.83) → “Tangier, Morocco” |

# The End

If you want to learn any new widget just type it in chrome and add “medium” key word .

See you in firebase Course