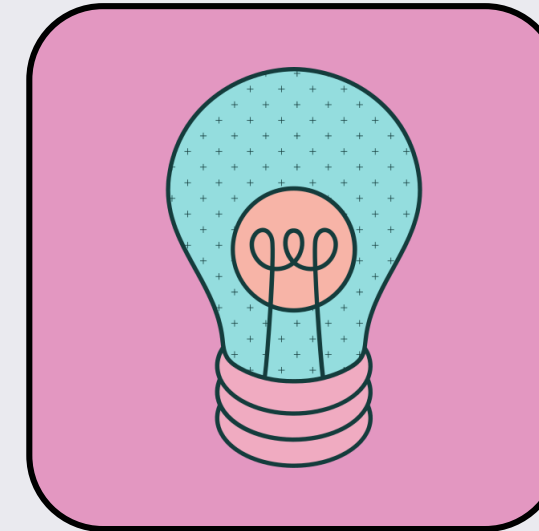


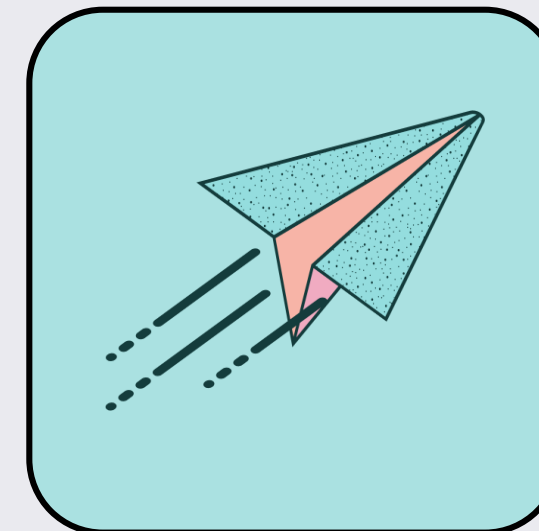
# ADVANCED FLUTTER

Instructor: Dr.Vahidi Asl  
Presenter: Nazanin Farhanj  
Fall 1403

# TABLE OF CONTENTS



**STATE MANAGEMENT**



**NAVIGATION AND  
ROUTING**



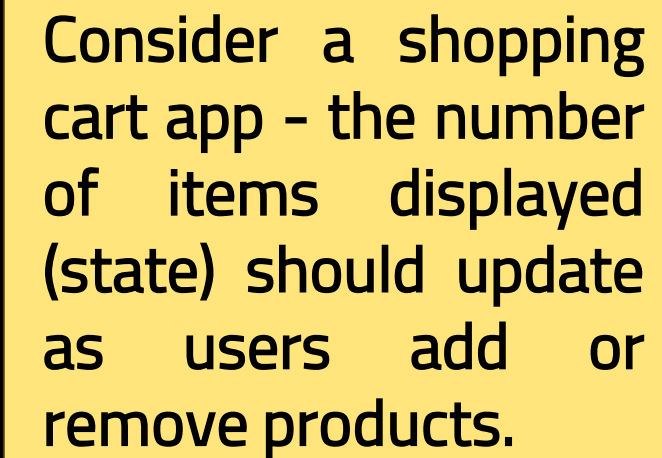
**RESPONSIVE DESIGN**

# WHAT IS STATE MANAGEMENT?

State refers to the data determining the application's visual representation. In Flutter, 'State' is a dynamic value that can be changed on the runtime based on user interaction or other events.

It is created when you build your widget and changes throughout the widget's lifetime. Developers can provide dynamic and responsive User Experiences with the help of state management in Flutter. Without proper state management, the UI might not reflect these changes,

leading to a confusing and unresponsive user experience.



Consider a shopping cart app - the number of items displayed (state) should update as users add or remove products.

# WHAT IS SETSTATE?

setState is a method provided by the State class in Flutter. It's commonly used for updating the state of a StatefulWidget. This method tells Flutter to rebuild the widget with the updated state. It's a simple and built-in way to manage the state of your widget tree.



Simple and understandable for beginners.  
Easy implementation for small apps.  
Suitable for self-contained widgets.



Requires passing data down the widget tree. Limited scalability for large apps.

# WHEN TO USE SETSTATE?

## Toggling a boolean

One of the simplest use cases for setState is toggling a boolean value. Consider a scenario where you want to change the background color of a widget when a button is pressed.

```
bool isBackgroundRed = false;

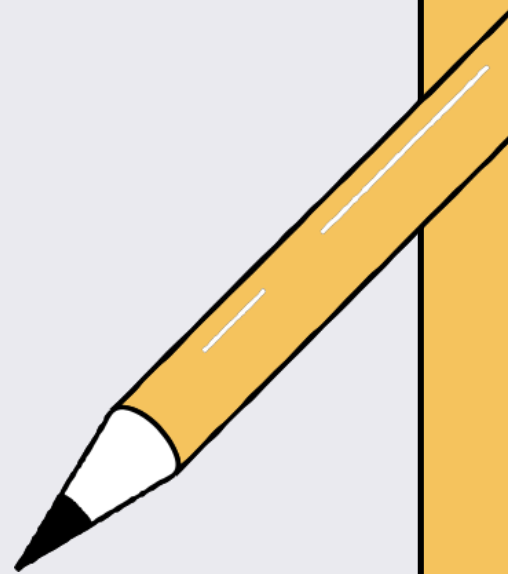
void toggleBackgroundColor() {
  setState(() {
    isBackgroundRed = !isBackgroundRed;
  });
}
```

## Counters and Incrementing Values

Another common use case is maintaining and updating counters or other numeric values.

```
int counter = 0;

void incrementCounter() {
  setState(() {
    counter++;
  });
}
```

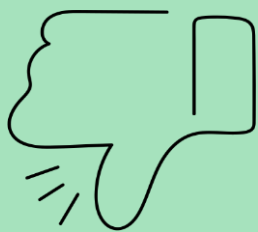


# WHAT IS PROVIDER?

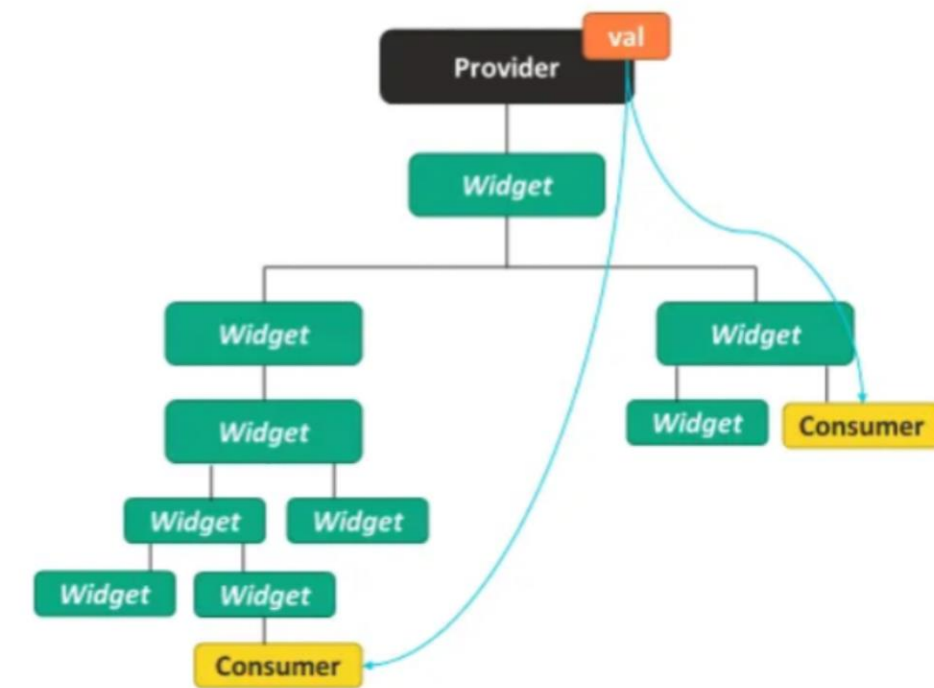
Provider is a state management solution in Flutter that builds upon the foundation of InheritedWidget. It has gained popularity for its simplicity and effectiveness in managing state within Flutter applications. With Provider, developers can easily provide data to any widget in their app, offering flexibility and enhancing productivity.



- Simplifies state management.
- Offers flexibility in data provisioning.
- Optimizes widget rebuilds efficiently.



- Learning curve for newcomers.
- Potential boilerplate code overhead.



In this graph, you can see the performance of the Provider being the parent of all widgets. It can share the information with whoever needs it, and the Consumer can update the information of its parent.

# WHAT IS BLOC?

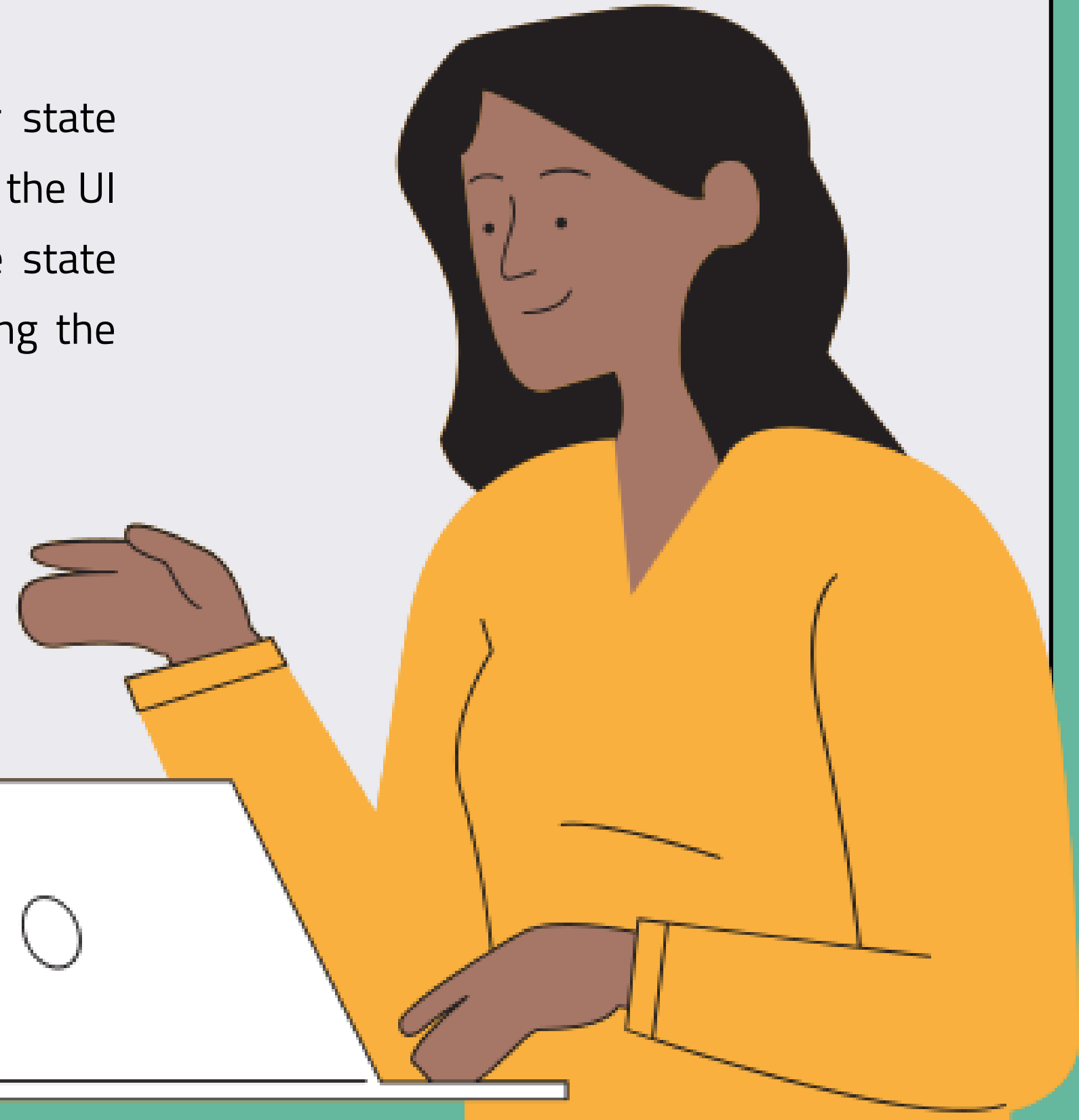
The Bloc (Business Logic Component) pattern is commonly used for state management in Flutter, especially for complex applications. It separates the UI from the business logic and uses streams to manage and propagate state changes. The `flutter_bloc` package is widely adopted for implementing the Bloc pattern.



Separates UI from business logic.  
Comprehensive testing with **bloc\_test**.  
Complete documentation and community support.



Requires more boilerplate code.  
Less ideal for simple scenarios.



# NAVIGATION AND ROUTING

Navigation and routing are some of the core concepts of all mobile application, which allows the user to move between different pages.

We know that every mobile application contains several screens for displaying different types of information. **For example**, an app can have a screen that contains various products. When the user taps on that product, immediately it will display detailed information about that product.

In Flutter, the screens and pages are known as **routes**, and these routes are just a widgets.

In any mobile app, navigating to different pages defines the workflow of the application, and the way to handle the navigation is known as **routing**. Flutter provides a basic routing class **MaterialPageRoute** and two methods **Navigator.push()** and **Navigator.pop()** that shows how to navigate between two routes. The following steps are required to start navigation in your application.

To navigate to a new screen in Flutter, you can use the `Navigator.push()` method. This method allows you to push a new route onto the stack and transition to the new screen.

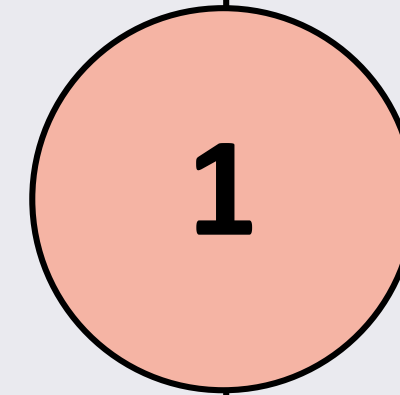
```
Navigator.push(  
  context,  
  MaterialPageRoute(builder: (context) => NewScreen()),  
);
```

To return to the previous screen, you can use the `Navigator.pop()` method. This method pops the current route off the stack, revealing the previous route.

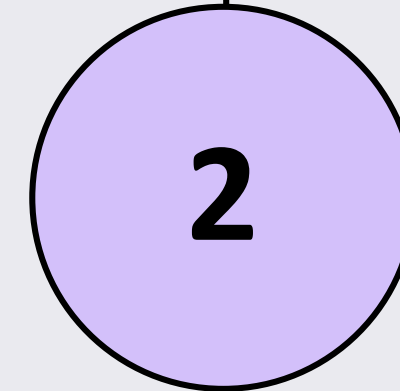
```
Navigator.pop(context);
```



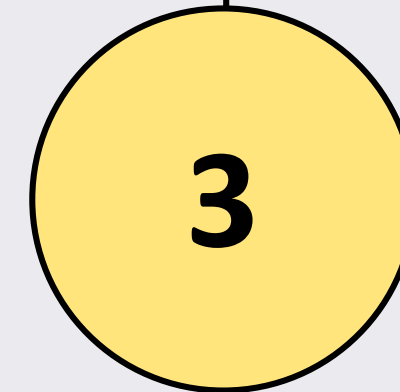
# RESPONSIVE DESIGN IN FLUTTER



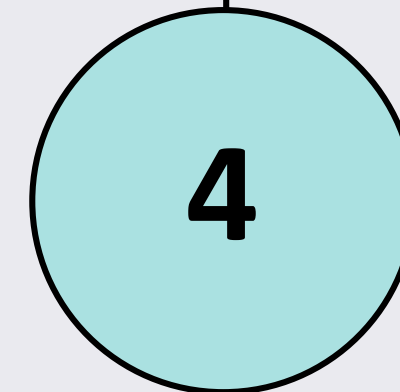
MediaQuery



LayoutBuilder



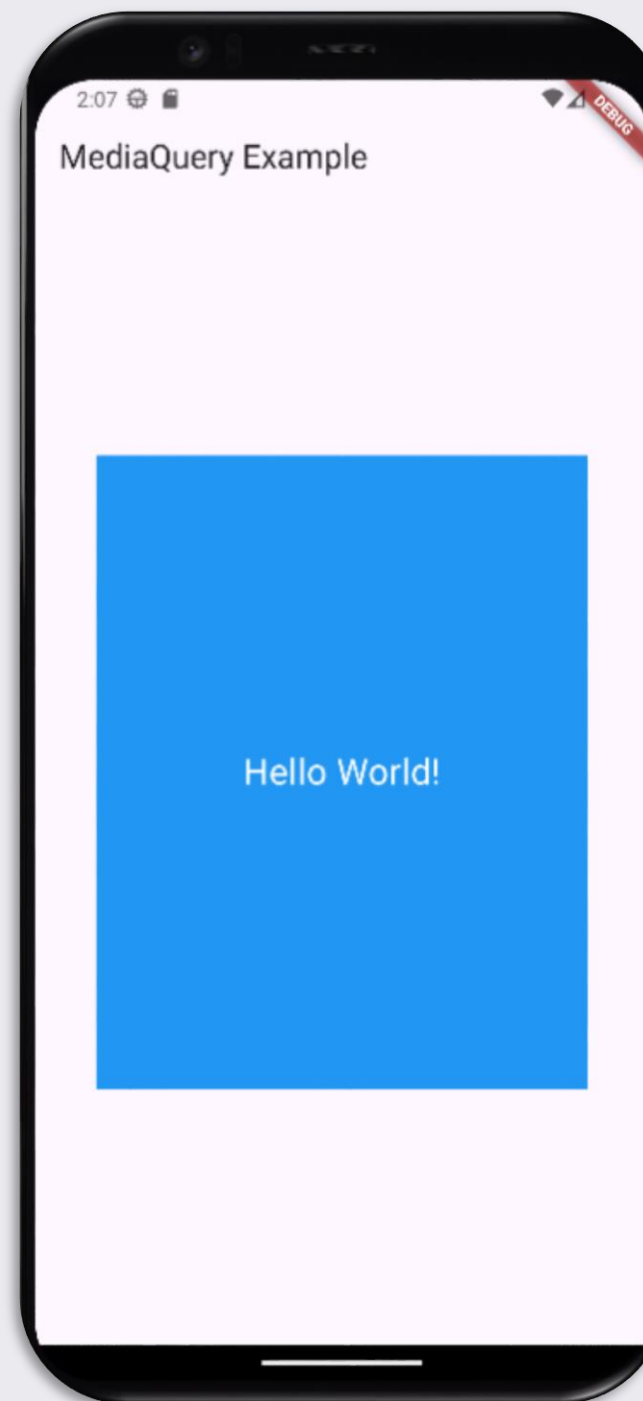
Expanded and flexible



AspectRatio

# MEDIA QUERY

```
body: Center(  
  child: Container(  
    // Use MediaQuery to retrieve the screen width and height  
    width: MediaQuery.of(context).size.width * 0.8,  
    height: MediaQuery.of(context).size.height * 0.5,  
    color: Colors.blue,  
    child: const Center(  
      child: Text(  
        'Hello World!',  
        style: TextStyle(color: Colors.white, fontSize: 24),  
      ), // Text
```

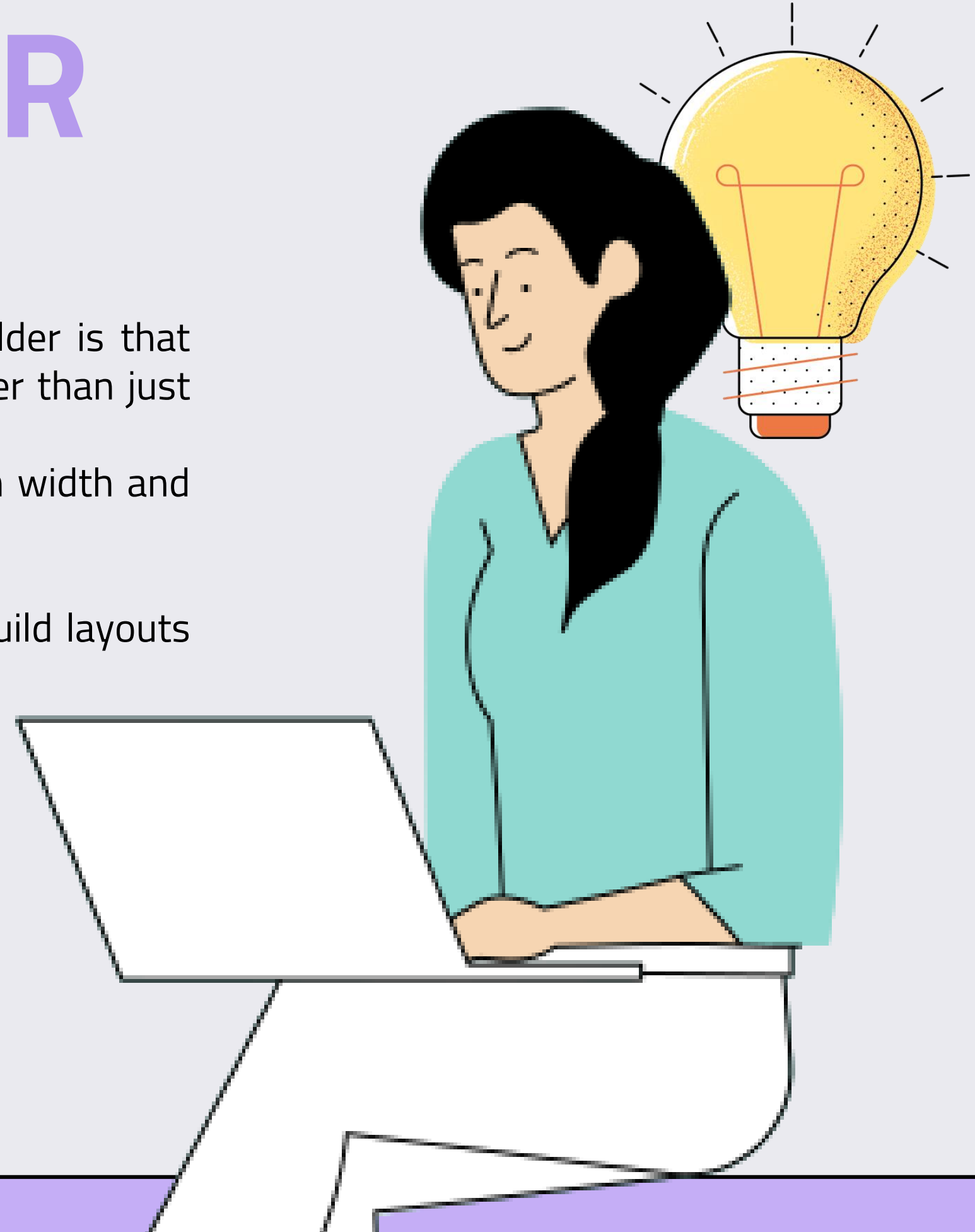


The MediaQuery widget gets information about the device, like screen size, dimensions, orientation, and pixel density. This information can be used to adjust your app's layout to fit the device screen size and aspect ratio. Media Query is a widget that is used to make the responsive User interface using size aware widgets. It is used to get the size of the current media (screen size or window)

# LAYOUT BUILDER

Layout Builder is just a simplified version of Media Query. The main difference between Media Query and Layout Builder is that Media Query is based on the full context of the screen rather than just the size of a particular widget; on the other hand, Layout Builder determines the maximum width and height of a specific widget only.

The main purpose of LayoutBuilder is to provide a way to build layouts that can adapt to different screen sizes and orientations.



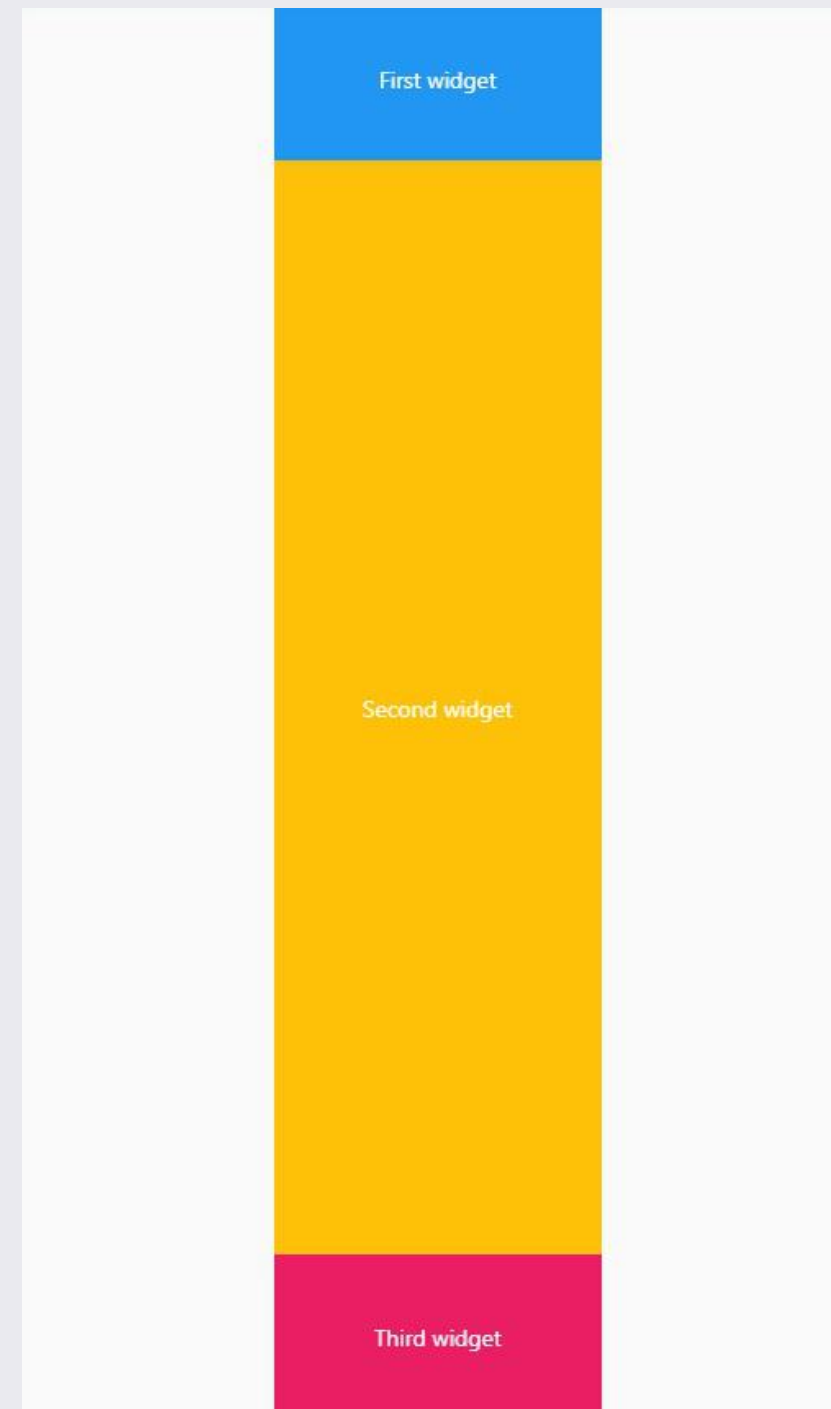
# EXPANDED AND FLEXIBLE

The widgets that are especially useful inside a Column or a Row are Expanded and Flexible.

The Expanded widget expands a child of a Row, Column, or Flex so that the child fills the available space, whereas Flexible does not necessarily have to fill the entire available space.

The Expanded Widget is a single child widget.

This means that it can have only one child assigned to it. To make this widget work properly, it must be used inside a row or column.



## Properties

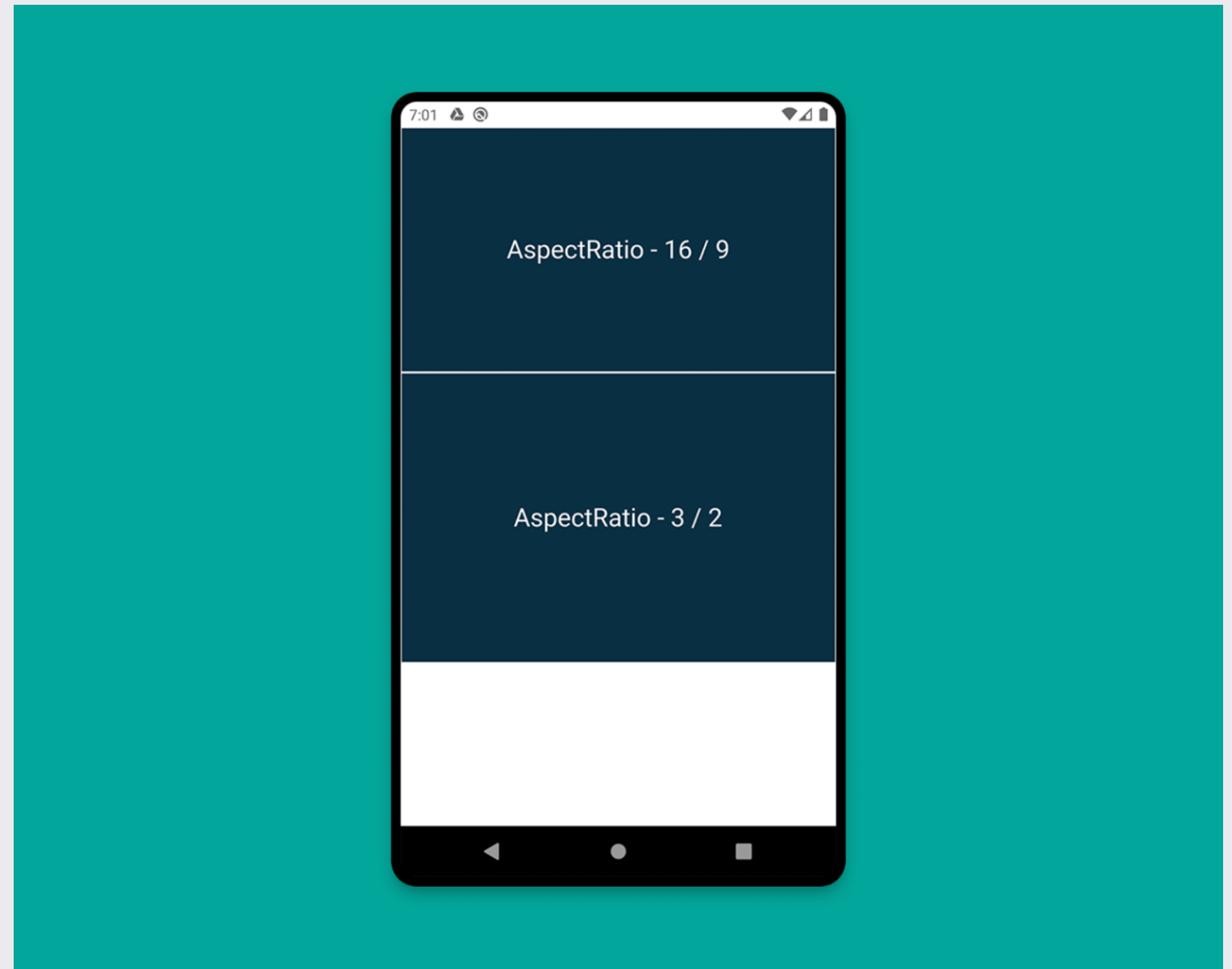
**child:** Child widget is used to place inside the expanded widget. Which we can take in rows and columns.

**Flex:** The flex property, which uses flex to distribute the available space unevenly between child widgets.



# ASPECT RATIO

In Flutter, the **AspectRatio** widget is used to enforce a specific aspect ratio for its child widget. It's particularly useful when you want to control the width-to-height ratio of a widget, ensuring that it maintains a specific aspect ratio regardless of the available space or layout constraints.



# USEFUL WEBSITES

## State management

<https://reliasoftware.com/blog/state-management-in-flutter>

<https://medium.com/@enitmehra/state-management-in-flutter-a-comprehensive-guide-7212772f026d>

<https://www.techaheadcorp.com/blog/master-state-management-in-flutter-best-practices-and-patterns/>

<https://medium.com/@midhunarmid/state-state-management-in-flutter-53338aaa0a1a>

## Navigation and Routing

<https://www.javatpoint.com/flutter-navigation-and-routing>

<https://reliasoftware.com/blog/navigation-in-flutter>

<https://remelehane.medium.com/introduction-to-navigation-and-routing-in-flutter-de72c3c09c11>

<https://medium.com/@chetan.akarte/explain-the-flutter-navigator-widget-and-its-methods-0f6f5023ff0c>

## Responsive design

<https://medium.com/@sharonatim/building-responsive-layouts-in-flutter-ea329c3637d3>

<https://blog.codemagic.io/building-responsive-applications-with-flutter/>

<https://medium.com/@kamrani062/responsiveness-in-flutter-ui-crafting-screens-for-all-devices-a264c3f435ec>

<https://www.browserstack.com/guide/make-flutter-app-responsive>

<https://blog.openreplay.com/responsive-design-with-flutter/>



**THANK YOU!**