Mobile Application Development

(Flutter Framework)

Last Updated: 18 July 2024

Table of Contents

Dart Programming Language 5

Simple Hello World App 5

Passing arguments through console 5

Variables 5

Variable creation and initialization. 5

Changing value of variable. 5

We use Object data type, we want to change data type of variable. 5

Explicit data types 6

Nullable type 6

Late variables 6

Final and Const 6

Constant 7

Variable examples 7

Conditional expressions: 7

condition ? expr1 : expr2 7

expr1 ?? expr2 8

Comments: 8

Builtin-Types: 8

Strings: 8

String concatenation example: 8

Multiline string: 8

Records 9

Lists 11

List of Records: 11

Sets 11

Maps 12
Objects in Dart resembling javascript objects 14
List of map objects 14
Spread operators 14
Control-flow operators 15
Patterns 16
Variable assignment 16
Switch statements and expressions 17
For and for-in loops 18
Functions: 18
Named parameters 19
Optional positional parameters 20
The main() function 20
Functions as first-class objects 21
Anonymous functions 21
Arrow notation: 22
typdef functions 23
Error handling. 23
Classes 24
Simple class example 24
No argument constructor 24
One argument generative constructor 24
Two argument generative constructor 25
Another example of two argument constructor 25
Calling a constructor from another constructor within same class. 26
Named constructors 26
Named arguments in a constructor. 27
Immutable objects 27
Optional arguments to a constructor 28
Array of objects 28
Printing elements in an array 29
Looping through array of objects: 30
Use `continue` to skip to the next loop iteration: 30
Inheritance example. 30
Using parent class constructor in child class 31
Calling named argument constructor from Child class of Parent class 31

Flutter widgets **32**
runApp() function 33
MaterialApp Widget: 36
Scaffold Widget: 38
MaterialApp and Material Widget 40
StatelessWidget 43
Row widget. 47
StatefulWidget 70
Example: Display message in Text field on button click 70
Example: A counter application 72
Example: Two counters 74
Example: Two counters and their sum. 76
Example: Introducing separate widgets for counter increment and counter display: 79
Example to take user input using onChanged method and display as Text field 81
Assigning value of one TextField to other on button click 83
Real-time update of one TextField through other 84
Example of two TextFields using TextEditingController class and display with button 85
An input form example 87
Shopping Cart with single item: 90
A shopping cart with multiple Items 93
Creating a layout 96
Creating a ListView 102
Creating a horizontal ListView 103
Creating a Grid List 105
Create lists with different types of items 107
List with spaced items 110
Work with long lists: 112
Theme based styling: 114
Add interactivity to our app 122
Material Components 122
Drag and Drop 122
Input & Forms 122
Create and style a text field: 122
Retrieve the value of a text field 123
Handle changes to a text field 126
Manage focus in text fields 127

Build a form with validation 129
Navigation and Routing 131
Working with Tabs: 131
Navigate to new screen and back: 132
Navigation with CupertinoPageRoute 134
Send a string to a new screen 135
Send a string to new screen using RouteSettings 137
Send data shown in a list to a new screen 138
Send data shown in a list to a new screen using RouteSettings 141
Return data from a screen 142
Add a drawer to a screen 145
Create a login page in flutter 148
Connecting Flutter with Firestore 151
Write and read document from Firestore database 151
Custom objects 157
Update a document 161
Server Timestamp 161
Update elements in an array 164
Increment a numeric value 164
Delete fields 165
Get multiple documents from a collection 168
Connecting Flutter with local SQLite Database 173
Flutter shared preferences 177
Connecting with PHP and MySQL ..... Error! Bookmark not defined.
Connecting with PHP and making updates in MySQL using Flutter 182
Communicate with WebSockets 200
Handling Camera in Flutter 201
GPS tracking in Flutter 204

Dart Programming Language

Simple Hello World App

Create a Dart console application using Visual Studio Code. The following code will be generated. The calculate() function is defined in lib/dartproject.dart.

```
import 'package:dartproject/dartproject.dart' as dartproject;

void main(List<String> arguments) {
  print('Hello world: ${dartproject.calculate()}!');

}
```

Passing arguments through console

```
import 'package:dartproject/dartproject.dart' as dartproject;

void main(List<String> arguments) {
  print('Arguments: $arguments');
}
```

To pass the parameters to the above application using console, run the following command in console:

A:\AAA\flutterprojects\dartproject\bin> dart dartproject.dart one two three

Variables

Variable creation and initialization.

```
var name = 'Bob';
print(name);
```

Changing value of variable.

```
var name = 'Bob';
name = 10; // not allowed to change data type
print(name);
```

The following code will work.

We use Object data type, we want to change data type of variable.

```
Object name = 'Bob';
```

```
name = 10;
print(name);
```

Explicit data types

```
String name = 'Bob';
print(name);
```

Nullable type

```
String? name1;  // Nullable type. Can be `null` or string.

String name2;   // Non-nullable type. Cannot be `null` but can be string.

print(name1);  // null
// print(name2); // error
```

Late variables

When you mark a variable as late but initialize it at its declaration, then the initializer runs the first time the variable is used. This lazy initialization is handy in a couple of cases:

The variable might not be needed, and initializing it is costly.

You're initializing an instance variable, and its initializer needs access to this.

```
late String description;

void main() {
  description = 'Feijoada!';
  print(description);
}
```

Final and Const

A final variable can be set only once; a const variable is a compile-time constant. (Const variables are implicitly final.)

```
void main() {
final name = 'Bob'; // Without a type annotation
final String nickname = 'Bobby';

print(name);
print(nickname);

// name = 'Ali'; // This is error as value of final cannot be changed
```

```
                // after it is initialized.
}
```

Constant

```
void main() {
const bar = 1000000; // Unit of pressure (dynes/cm2)
const double atm = 1.01325 * bar; // Standard atmosphere
}
```

Variable examples

```
void main() {
var name = 'Voyager I';
var year = 1977;
var antennaDiameter = 3.7;
var flybyObjects = ['Jupiter', 'Saturn', 'Uranus', 'Neptune'];
var image = {
  'tags': ['saturn'],
  'url': '//path/to/saturn.jpg'
};

print(name);
print(year);
print(antennaDiameter);
print(flybyObjects);
print(image);

}
```

Conditional expressions:

condition ? expr1 : expr2

If *condition* is true, evaluates *expr1* (and returns its value); otherwise, evaluates and returns the value of *expr2*.

```
void main()
{
 var isPublic = 'public';
 var visibility = isPublic=='public' ? 'public' : 'private';

 print(visibility);
```

```
}
```

expr1 ?? expr2

If *expr1* is non-null, returns its value; otherwise, evaluates and returns the value of *expr2*.

```
void main()
{
var var1 = null;  // or simply write var var1; this defaults to null
var var2 = 10;

var ans = var1 ?? var2;
 print(ans);
}
```

Comments:

// single line , /* multi line */, /// for documentation.

Builtin-Types:

Numbers, Strings, Booleans.

Strings:

String concatenation example:

```
void main() {
var s1 = 'String '
    'concatenation'
    " works even over line breaks.";

  print(s1);
}
```

Multiline string:

```
void main() {

var s1 = '''
You can create
multi-line strings like this one.
''';
```

```
  print(s1);
}
```

Records

Records are an anonymous, immutable, aggregate type. Like other collection types, they let you bundle multiple objects into a single object. Unlike other collection types, records are fixed-sized, heterogeneous, and typed.

Records are real values; you can store them in variables, nest them, pass them to and from functions, and store them in data structures such as lists, maps, and sets.

Example:

```
void main() {

// Record type annotation in a variable declaration:
({int a, bool b}) record;

// Initialize it with a record expression:
record = (a: 123, b: true);

print(record);
print(record.a);
print(record.b);

}
```

Records expressions are comma-delimited lists of named or positional fields, enclosed in parentheses:

```
dart
```

Example:

Here ‘first’, “hello”, ‘last’ are positional fields, and others are named.

```
void main() {

var record = ('first', a: 2, "hello", b: true, 'last');

print(record.$1); // Prints 'first'
print(record.a); // Prints 2
print(record.b); // Prints true
print(record.$2); // Prints 'last'
```

```
print(record.$3);

}
```

In a record type annotation, named fields go inside a curly brace-delimited section of type-and-name pairs, after all positional fields. In a record expression, the names go before each field value with a colon after:

```
// Record type annotation in a variable declaration:
({int a, bool b}) record;

// Initialize it with a record expression:
record = (a: 123, b: true);
```

The names of named fields in a record type are part of the record's type definition, or its *shape*. Two records with named fields with different names have different types:

```
({int a, int b}) recordAB = (a: 1, b: 2);
({int x, int y}) recordXY = (x: 3, y: 4);

// Compile error! These records don't have the same type.
// recordAB = recordXY;
```

In a record type annotation, you can also name the *positional* fields, but these names are purely for documentation and don't affect the record's type:

```
(int a, int b) recordAB = (1, 2);
(int x, int y) recordXY = (3, 4);

recordAB = recordXY; // OK.
```

Example

```
(int x, int y, int z) point = (1, 2, 3);
(int r, int g, int b) color = (1, 2, 3);

print(point == color); // Prints 'true'.
```

Example

```
({int x, int y, int z}) point = (x: 1, y: 2, z: 3);
({int r, int g, int b}) color = (r: 1, g: 2, b: 3);
```

```
print(point == color); // Prints 'false'. Lint: Equals on unrelated types.
```

Lists

Perhaps the most common collection in nearly every programming language is the *array*, or ordered group of objects. In Dart, arrays are List objects, so most people just call them *lists*.

Dart list literals are denoted by a comma separated list of expressions or values, enclosed in square brackets ([]). Here's a simple Dart list:

```
void main() {

var list = [1, 2, 3];

print(list[0]);
}
```

List of Records:

```
void main() {

  var list = [(id: 1, name:'Ali'), (id: 2, name:'javed')];

  for( var item in list) {
    print(item.id);
  }
}
```

Sets

#

A set in Dart is an unordered collection of unique items. Dart support for sets is provided by set literals and the Set type.

Here is a simple Dart set, created using a set literal:

```
void main() {

var halogens = {'fluorine', 'chlorine', 'bromine', 'iodine', 'astatine'};

print(halogens.elementAt(0));
print(halogens.elementAt(1));

for (var element in halogens)
```

```
{
  print(element);
}
}
```

To create an empty set, use {} preceded by a type argument, or assign {} to a variable of type Set:

```
void main() {

var names = <String>{};
// Set<String> names = {}; // This works, too.
// var names = {}; // Creates a map, not a set.

}
```

Add items to an existing set using the add() or addAll() methods:

```
void main() {

var halogens = {'fluorine', 'chlorine', 'bromine', 'iodine', 'astatine'};
var elements = <String>{};
elements.add('fluorine');
elements.addAll(halogens);

}
```

Maps

In general, a map is an object that associates keys and values. Both keys and values can be any type of object. Each *key* occurs only once, but you can use the same *value* multiple times. Dart support for maps is provided by map literals and the Map type.

```
var gifts = {
  // Key:    Value
  'first': 'partridge',
  'second': 'turtledoves',
  'fifth': 'golden rings'
};

var nobleGases = {
  2: 'helium',
  10: 'neon',
  18: 'argon',
};
```

Add a new key-value pair to an existing map using the subscript assignment operator ([]=):

Dart

```
var gifts = {'first': 'partridge'};
gifts['fourth'] = 'calling birds'; // Add a key-value pair
```

Retrieve a value from a map using the subscript operator ([]):

Dart

```
void main() {

var gifts = {'first': 'partridge'};
assert(gifts['first'] == 'partridge');

}
```

To print all values of map

```
void main() {

var nobleGases = {
  2: 'helium',
  10: 'neon',
  18: 'argon',
};

for(var key in nobleGases.keys) {
print(nobleGases[key]);
}

}
```

You can create the same objects using a Map constructor:

Dart

```
void main() {

var gifts = Map<String, String>();
gifts['first'] = 'partridge';
gifts['second'] = 'turtledoves';
gifts['fifth'] = 'golden rings';

var nobleGases = Map<int, String>();
nobleGases[2] = 'helium';
nobleGases[10] = 'neon';
nobleGases[18] = 'argon';
```

```
}
```

Add a new key-value pair to an existing map using the subscript assignment operator ([]=):

Dart

```
var gifts = {'first': 'partridge'};
gifts['fourth'] = 'calling birds'; // Add a key-value pair
```

Objects in Dart resembling javascript objects

We can define javascript type objects using Map:

```
Map<String, String> myObject1 = {
  'name': 'Devin',
  'hairColor': 'brown',
};

print(myObject1);
```

List of map objects

```
var list = [
   {'name': 'Kamran', 'color' : 'red'},
   {'name': 'Shahid', 'color' : 'blue'},
   {'name': 'Zahid', 'color' : 'green'},

   ];
```

Spread operators

Dart supports the **spread operator** (...) and the **null-aware spread operator** (...?) in list, map, and set literals. Spread operators provide a concise way to insert multiple values into a collection.

For example, you can use the spread operator (...) to insert all the values of a list into another list:

```
void main() {

var list = [1, 2, 3];
var list2 = [0, ...list];
assert(list2.length == 4);
}
```

If the expression to the right of the spread operator might be null, you can avoid exceptions by using a null-aware spread operator (...?):

```
void main() {

var list2 = [0, ...?list];
assert(list2.length == 1);

}
```

We can modify the map objects using spread operator:

```
Map<String, String> myObject1 = {
  'name': 'Devin',
  'hairColor': 'brown',
};

var myObject2 = {...myObject1, 'name':'Kamran'};
print(myObject2);
```

Control-flow operators

Dart offers **collection if** and **collection for** use in list, map, and set literals. You can use these operators to build collections using conditionals (if) and repetition (for).

Here's an example of using **collection if** to create a list with three or four items in it:

```
void main() {

bool promoActive = false;
var nav = ['Home', 'Furniture', 'Plants', if (promoActive) 'Outlet'];
print(nav);
}
```

```
void main() {

var login = "Manager";

var nav = ['Home', 'Furniture', 'Plants', if (login case 'Manager')
'Inventory'];

print(nav);
}
```

Here's an example of using **collection for** to manipulate the items of a list before adding them to another list:

```
void main() {

var listOfInts = [1, 2, 3];
var listOfStrings = ['#0', for (var i in listOfInts) '#$i'];
//assert(listOfStrings[1] == '#1');
print(listOfStrings);
}
```

Patterns

Destructuring

#

When an object and pattern match, the pattern can then access the object's data and extract it in parts. In other words, the pattern *destructures* the object:

```
void main() {

var numList = [1, 2, 3];
// List pattern [a, b, c] destructures the three elements from numList...
var [a, b, c] = numList;
// ...and assigns them to new variables.
print(a + b + c);

}
```

Variable assignment

A *variable assignment pattern* falls on the left side of an assignment. First, it destructures the matched object. Then it assigns the values to *existing* variables, instead of binding new ones.

Use a variable assignment pattern to swap the values of two variables without declaring a third temporary one:

```
void main() {

var (a, b) = ('left', 'right');
(b, a) = (a, b); // Swap.
print('$a $b'); // Prints "right left".
```

```
}
```

Switch statements and expressions

Every case clause contains a pattern. This applies to switch statements and expressions, as well as if-case statements. You can use any kind of pattern in a case.

Case patterns are refutable. They allow control flow to either:

- Match and destructure the object being switched on.
- Continue execution if the object doesn't match.

The values that a pattern destructures in a case become local variables. Their scope is only within the body of that case.

```
void main() {

int obj = 3;
const first = 2;
const last = 4;

switch (obj) {
  // Matches if 1 == obj.
  case 1:
    print('one');

  // Matches if the value of obj is between the
  // constant values of 'first' and 'last'.
  case >= first && <= last:
    print('in range');

  // Matches if obj is a record with two fields,
  // then assigns the fields to 'a' and 'b'.
  case (var a, var b):
    print('a = $a, b = $b');

  default:
}


}
```

Guard clauses evaluate an arbitrary conditon as part of a case, without exiting the switch if the condition is false (like using an if statement in the case body would cause).

```
void main() {
```

```
var pair=(20,10);
switch (pair) {
  case (int a, int b):
    if (a > b) print('First element greater');
  // If false, prints nothing and exits the switch.
  case (int a, int b) when a > b:
    // If false, prints nothing but proceeds to next case.
    print('First element greater');
  case (int a, int b):
    print('First element not greater');
}
}
```

For and for-in loops

You can use patterns in for and for-in loops to iterate-over and destructure values in a collection.

This example uses object destructuring in a for-in loop to destructure the MapEntry objects that a <Map>.entries call returns:

```
void main() {
Map<String, int> hist = {
  'a': 23,
  'b': 100,
};

for (var MapEntry(key: key, value: count) in hist.entries) {
  print('$key occurred $count times');
}

}
```

Can also be written as:

```
for (var MapEntry(:key, value: count) in hist.entries) {
  print('$key occurred $count times');
}
```

Functions:

```
bool isNoble(int atomicNumber) {
  return true;
```

```
}
```

Although Effective Dart recommends type annotations for public APIs, the function still works if you omit the types:

```
isNoble(int atomicNumber) {
  return true;
}
```

For functions that contain just one expression, you can use a shorthand syntax:

```
bool isNoble(int atomicNumber) => _nobleGases[atomicNumber] != null;
```

Named parameters

Named parameters are optional unless they're explicitly marked as required.

When defining a function, use {*param1*, *param2*, …} to specify named parameters. If you don't provide a default value or mark a named parameter as required, their types must be nullable as their default value will be null:

```
void main() {
  print("Hello world");

enableFlags(); // will assign default null to bold and hidden
enableFlags(bold:true, hidden: true);
}

void enableFlags({bool? bold, bool? hidden}) {

  print("$bold $hidden");

}
```

To define a default value for a named parameter besides null, use = to specify a default value. The specified value must be a compile-time constant. For example:

```
void main() {
  print("Hello world");
  enableFlags(hidden: false);
}

void enableFlags({bool bold=true, bool hidden=false}) {

  print("$bold $hidden");

}
```

Optional positional parameters

Wrapping a set of function parameters in [] marks them as optional positional parameters. If you don't provide a default value, their types must be nullable as their default value will be null:

```
void main() {
  print("Hello world");

say("Ali", "hello"); // calling with two arguments
say("Ali", "hello", "laptop"); // calling with optional argument included
}

void say(String from, String msg, [String? device]) {
  var result = '$from says $msg';
  if (device != null) {
    result = '$result with a $device';
  }
  print(result);
}
```

To define a default value for an optional positional parameter besides null, use = to specify a default value. The specified value must be a compile-time constant. For example:

```
void main() {
 say("Ali", "hello"); // calling with two arguments
}

void say(String from, String msg, [String device = 'carrier pigeon']) {
  print('$from $msg $device');
}
```

The main() function

Every app must have a top-level `main()` function, which serves as the entrypoint to the app. The `main()` function returns `void` and has an optional `List<String>` parameter for arguments.

Here's a simple `main()` function:

The file dartapp.dart is in the bin folder. Run the app like this: dart run dartapp.dart 1 test

```
// Run the app like this: dart run args.dart 1 test
void main(List<String> arguments) {
  print(arguments);
```

```
}
```

Functions as first-class objects

You can pass a function as a parameter to another function. For example:

```
void main() {

var list = [1, 2, 3];

// Pass printElement as a parameter.
list.forEach(printElement);

}

void printElement(int element) {
  print(element);
}
```

You can also assign a function to a variable, such as:

```
void main() {
var loudify = (msg) => '!!! ${msg.toUpperCase()} !!!';

print(loudify("hello"));
}
```

Anonymous functions

Most functions are named, such as `main()` or `printElement()`. You can also create a nameless function called an *anonymous function*, or sometimes a *lambda* or *closure*. You might assign an anonymous function to a variable so that, for example, you can add or remove it from a collection.

The following example defines an anonymous function with an untyped parameter, `item`, and passes it to the `map` function. The function, invoked for each item in the list, converts each string to uppercase. Then in the anonymous function passed to `forEach`, each converted string is printed out alongside its length.

```
void main() {

const list = ['apples', 'bananas', 'oranges'];
```

```
 var result = list.map (

  (item)
    {
     return item.toUpperCase();
    }
);

print(result);

result.forEach((item) {
  print('$item: ${item.length}');
});

/*
// COMBINED FORM

list.map((item) {
  return item.toUpperCase();
}).forEach((item) {
  print('$item: ${item.length}');
});
*/


}
```

Arrow notation:

Example:

```
void main() {

const list = ['apples', 'bananas', 'oranges'];

 print( list.map((item) => item.toUpperCase()) );


}
```

Example:

```
void main() {

const list = ['apples', 'bananas', 'oranges'];
```

```
 list.map((item) => item.toUpperCase())
    .forEach((item) => print('$item: ${item.length}'));


}
```

typdef functions

```
// 1. Define a typedef for a function that takes two numbers and returns their
sum
typedef SumFunction =  int Function(int a, int b);

void main() {


  // 2. Implement a function that matches the SumFunction signature
  int add(int a, int b) => a + b;

  // 3. Use the SumFunction type to declare a variable and assign the add
function
  SumFunction sum = add;

  // 4. Call the function using the variable with the SumFunction type
  int result = sum(5, 3); // result will be 8

   print("Sum of 5 and 3 is: $result");
}
```

Error handling.

```
void main() {

    misbehave();

}

void misbehave() {
  try {
    dynamic foo = true;
    print(foo++);
  } catch (e) {
    print(e);
  }
```

```
}
```

Classes

Simple class example

```
class Point {
  double? x; // Declare instance variable x, initially null.
  double? y; // Declare y, initially null.
}

void main() {
  var point = Point();
  point.x = 4; // Use the setter method for x.
  print(point.x);

}
```

No argument constructor

```
class Car {

 Car() {
print("no argument constructor called");
 }

/*
//OR
Car();
*/

}

void main() {
Car c = Car();

}
```

One argument generative constructor

```
class Car {
 String model;
 Car(this.model);
}
```

```
void main() {
Car c = Car("Toyota");
print(c.model);
}
```

Two argument generative constructor

```
import 'dart:math';

class Point {
  final double x;
  final double y;

  // Sets the x and y instance variables
  // before the constructor body runs.
  Point(this.x, this.y);

  double distanceTo(Point other) {
    var dx = x - other.x;
    var dy = y - other.y;
    return sqrt(dx * dx + dy * dy);
  }
}

void main() {
Point p1 = Point(10, 20);
 print("x: ${p1.x}, y: ${p1.y}");
Point p2 = Point(5, 10);
print(p2.distanceTo(p1));
}
```

Another example of two argument constructor

If we want to reprocess the variables before assignment to class variables, we can define the constructor as follows.

```
class Car {

String? name;
int? age;

Car(name, age){
  this.name = '$name' ' Ali';
  this.age = age + 30;
}
```

```
}

void main() {
Car c = Car("Shahid", 45);

print('${c.name} ${c.age}');

}
```

Calling a constructor from another constructor within same class.

```
class Point {
  int? x;
  int? y;

  Point(int a, int b) {
    x = a;
    y = b;
  }


Point.fromOrigin() : this(10, 20); // Calls the main constructor
}

void main() {
  var point1 = Point(3, 4);
  var point2 = Point.fromOrigin();
  print('${point1.x} ${point1.y}');
  print('${point2.x} ${point2.y}');
}
```

Named constructors

```
class User {

User.none();
User.withId(this.id);
User.withName(this.name);
User.withidname(this.id, this.name);

// we can also place class variables after constructors
int? id;
String? name;

}
```

```
void main() {

var user1 = User.none();
var user2 = User.withId(10);
var user3 = User.withName("Javed");
var user4 = User.withidname(10, "Javed");

print('${user1.id} ${user1.name}');
print('${user2.id} ${user2.name}');
print('${user3.id} ${user3.name}');
print('${user4.id} ${user4.name}');


}
```

Named arguments in a constructor.

```
class Person {
  String? name;
  int age;

  Person({required this.age, String? name}) { // Name is optional now
    this.name = name ?? "Unknown"; // Default value for name if not provided
  }
}

void main(){
Person person1 = Person(age: 25); // Specify age first, name is unknown
print('${person1.age} ${person1.name}');

Person person2 = Person(name: "Bob", age: 42); // Specify both in any order
print('${person2.age} ${person2.name}');

}
```

Immutable objects

By adding const, the created object is immutable, its content can’t be changed.

```
class Teacher {
  const Teacher(this.name);
  final String name;
}
```

```
void main() {
  var teacher = const Teacher("Osman");

  print(teacher.name);

  //teacher.name = "Zahid"; //error

}
```

Optional arguments to a constructor

```
class Person {
  final String name;
  final int? age; // Can be null

  // Constructor with optional age parameter
  const Person(this.name, {this.age});
}

void main() {
// Create a Person with name and age
var person1 = Person('Alice', age: 30);

// Create a Person with only name (age will be null)
var person2 = Person('Bob');

print('${person1.name} ${person1.age}');

 }
```

Array of objects

```
class Student {

  String name;
  int age;

  // Constructor
  Student(this.name, this.age);

}

void main() {

// Create an array (List) of MyObject instances
```

```
  List<Student> myObjects = [];

  // Add objects to the array
  myObjects.add(Student('Alice', 25));
  myObjects.add(Student('Bob', 30));
  myObjects.add(Student('Charlie', 22));

  // Access and print elements in the array
  for (Student obj in myObjects) {
    print('Name: ${obj.name}, Age: ${obj.age}');
  }
}
```

Printing elements in an array

```
class Student {

  String name;
  int age;

  // Constructor
  Student(this.name, this.age);

}

void main(List<String> arguments) {

// Create an array (List) of MyObject instances
  List<Student> myObjects = [];

  // Add objects to the array
  myObjects.add(Student('Alice', 25));
  myObjects.add(Student('Bob', 30));
  myObjects.add(Student('Charlie', 22));

  // Access and print elements in the array
  myObjects.where((student) =>
student.name.contains('Alice')).forEach((student)=>print('Name:
${student.name}, Age: ${student.age}'));

}
```

Looping through array of objects:

```
class Candidate {
  final String name;
  final int yearsExperience;

  Candidate(this.name, this.yearsExperience);

  void interview() {
    print("$name is being interviewed...");
    // Simulate the interview process
    print("$name: Interview went well!");
  }
}

void main() {
  // List of candidates
  List<Candidate> candidates = [
    Candidate("Alice", 6),
    Candidate("Bob", 3),
    Candidate("Charlie", 8),
    Candidate("David", 4),
  ];

  // Filter candidates with 5 or more years of experience and conduct
interviews
  candidates
      .where((c) => c.yearsExperience >= 5)
      .forEach((c) => c.interview());
}
```

Use `continue` to skip to the next loop iteration:

```
for (int i = 0; i < candidates.length; i++) {
  var candidate = candidates[i];
  if (candidate.yearsExperience < 5) {
    continue;
  }
  candidate.interview();
}
```

Inheritance example.

```
class ParentClass {
  void myfunction() {
     print("This is parent class function");
  }

}
```

```
class ChildClass extends ParentClass {
  @override
  void myfunction() {
    super.myfunction();
    print("This is child class function");
  }

}
void main() {

ChildClass c = ChildClass();
c.myfunction();
}
```

Using parent class constructor in child class

```
class Person {

String? name;
int? age;

Person(this.name, this.age);

}

class Student extends Person { // Explicitly extend the Person class
  String? regno;

  Student(this.regno, {String? name = "Ali", int? age = 34}) : super(name,
age);
}

void main() {
  Student student = Student("ABC123");
  print(student.name); // Output: Ali
  print(student.age);  // Output: 34
  print(student.regno); // Output: ABC123
}
```

Calling named argument constructor from Child class of Parent class

```
class Person {
  String name;
  int age;
```

```
  Person({required this.name, required this.age}); // Named arguments
constructor
}

class Student extends Person {
  String id;

  Student({required this.id, required String name, required int age})
      : super(name: name, age: age); // Calling parent constructor with named
arguments
}

void main(){
Student s = Student(id: "334", name:"Jamal", age:34);
print('${s.id} ${s.name} ${s.age}');
}
```

Flutter widgets

In flutter, widgets act like tags in html containing information/data. Each widget is represented by class. Widgets call other widgets via class composition principles.

What's the point?

- Widgets are classes used to build UIs.
- Widgets are used for both layout and UI elements.
- Compose simple widgets to build complex widgets.

The core of Flutter's layout mechanism is widgets. In Flutter, almost everything is a widget—even layout models are widgets. The images, icons, and text that you see in a Flutter app are all widgets. But things you don't see are also widgets, such as the rows, columns, and grids that arrange, constrain, and align the visible widgets.

runApp() function

The runApp() function takes the given Widget and makes it the root of the widget tree.

Example:

Here the Text wizard is acting as root widget.

NOTE: When MaterialApp widget is used, we don't need to specify the text direction.

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

void main() {
  runApp(
    const Text(
      "hello world",
      textDirection: TextDirection.ltr,
      style: TextStyle(
        fontSize: 60,
        color: Color.fromARGB(255, 31, 136, 248),
      ),
    ),
  );
}
```

OUTPUT:

Why use const here?

The const keyword used with Text serves two main purposes:

1. **Widget Tree Optimization:** When you use const with a widget, it tells Flutter that the widget and its sub-tree (children) are unlikely to change throughout the lifetime of your app. This allows Flutter to perform some optimizations, such as caching the widget's configuration and avoiding unnecessary rebuilds. This is particularly beneficial for widgets that are expensive to create or render.
2. **Immutability:** Using const with a widget enforces immutability. This means that once the widget is created, its state cannot be changed. This aligns well with the concept of widgets in Flutter, which are supposed to represent a declarative UI state.

Key Points:

- The const keyword is most effective for widgets that are stateless and don't require dynamic updates.
- For widgets that need to update based on user interaction or data changes, using const might not be suitable. In such cases, you'd likely use the StatefulWidget approach.

In summary:

Using const with MaterialApp in this example helps Flutter optimize the widget tree and enforces immutability, which aligns with the principles of building UIs in Flutter.

Example:

To center align the Text, we use Center widget:

In this example, the widget tree consists of two widgets, the Center widget and its child, the Text widget. The framework forces the root widget to cover the screen, which means the text “Hello, world” ends up centered on screen.

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

void main() {
  runApp(
    const Center(
      child: Text(
        'Hello, world!',
        textDirection: TextDirection.ltr,
      ),
    ),
  );
}
```

Output:

MaterialApp Widget:

Many Material Design widgets need to be inside of a MaterialApp to display properly, in order to inherit theme data. Therefore, run the application with a MaterialApp.

In Flutter, **MaterialApp** represents a widget that implements the basic material design visual layout structure. It's typically used as the root widget of a Flutter application. Here's what it provides:

1. **Material Design Components**: MaterialApp provides the basic material design components such as Scaffold, AppBar, Drawer, BottomNavigationBar, SnackBar, and more. These components follow the Material Design guidelines for visual appearance and behavior.
2. **Text Style Warning**: If your app lacks a Material ancestor for text widgets, MaterialApp automatically applies an **ugly red/yellow text style**. This serves as a warning to developers that they haven't defined a default text style. Typically, the app's **Scaffold** defines the text style for the entire UI.
3. **Routing**: MaterialApp provides a navigator that manages a stack of Route objects and a route table for mapping route names to builder functions. This allows you to navigate between different screens or "routes" in your app using the **Navigator** widget.

4. **Theme**: MaterialApp allows you to define a theme for your entire application using the **theme** property. This includes defining colors, typography, shapes, and other visual properties that are consistent throughout your app.
5. **Localization**: MaterialApp supports internationalization and localization through the **localizationsDelegates** and **supportedLocales** properties. This allows you to provide translations for your app's text and adapt its behavior based on the user's locale.
6. **Accessibility**: MaterialApp includes accessibility features such as support for screen readers and semantic labels, making your app more accessible to users with disabilities.

Overall, MaterialApp serves as the foundation for building Flutter apps with material design principles, providing a consistent and intuitive user experience across different devices and platforms.

Example:

In this example, MaterialApp widget is made the root widget, and its 'home' property is invoked, in which Text wizard is passed.

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

void main() {
  runApp(const MaterialApp(
    home: Text("Hello world"),

  ));
}
```

Output:

Scaffold Widget:

While MaterialApp provides the overall structure and theme for a Flutter app, the Scaffold widget serves as a layout structure for individual screens or "pages" within the app. Here's what the Scaffold widget provides:

1. **App Bar**: Scaffold allows you to easily add an app bar at the top of the screen using the **appBar** property. The app bar typically contains a title, leading and/or trailing actions, and may also include other widgets like tabs or a search bar.
2. **Body Content**: Scaffold's **body** property is where you place the main content of the screen. This can be any widget or combination of widgets, such as text, images, lists, grids, or custom widgets.
3. **Navigation Drawer**: If your app uses a side navigation drawer, Scaffold provides the **drawer** property to easily add one to your screen. The drawer typically contains navigation links or settings options.
4. **Bottom Navigation Bar**: For apps with multiple screens or sections, Scaffold allows you to include a bottom navigation bar using the **bottomNavigationBar** property. This allows users to switch between different sections of the app.

5. **Floating Action Button**: Scaffold provides the **floatingActionButton** property to add a floating action button (FAB) to the screen. FABs are typically used for primary or frequently-used actions, such as adding a new item or starting a new task.
6. **Snackbar**: Scaffold includes methods for displaying snackbars, which are lightweight messages that appear at the bottom of the screen. This can be useful for showing brief feedback or notifications to the user.

Overall, while MaterialApp provides the overall structure and theme for the entire app, Scaffold provides a consistent layout structure for individual screens, making it easier to build and organize the UI of your Flutter app.

Example

See how the Point No. 2 of MaterialApp is addressed here when we used Scaffold widget.

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

void main() {
  runApp(const MaterialApp(
    home: Scaffold(
        body: Center(
            child: Text("hello world")
        )
    )
  ));
}
```

MaterialApp and Material Widget

MaterialApp:

- **Function:** A widget that serves as the foundation for building apps that follow Google's Material Design guidelines.
- **Purpose:** Provides a starting point for your app by configuring essential elements like:
    - **Theme:** Defines the overall look and feel of your app (colors, fonts, etc.)
    - **Routing:** Manages navigation between different screens in your app.
    - **Localization:** Enables support for different languages.
    - **Home:** Sets the initial screen displayed when the app launches.
- **Placement:** Placed at the root of your app's widget tree. There should typically be only one MaterialApp widget in your app.

Material:

- **Function:** A basic widget used for creating UI elements that adhere to Material Design principles.
- **Purpose:** Defines properties related to the visual appearance and behavior of individual UI components, such as:
    - **Elevation:** Creates a shadow effect for a 3D-like feel.
    - **Shape:** Defines the shape of the UI element (rounded corners, etc.).
    - **Clipping:** Controls how the widget's content is displayed within its bounds.
- **Placement:** Used throughout your app's widget tree to build various UI components like buttons, cards, app bars, etc.

Example:

Code snippet

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

void main() {
  runApp(MaterialApp(
    home: Scaffold( // Another Material widget for app structure
      appBar: AppBar(
        title: const Text('My Flutter App'), // Text widget (not directly
from Material)
        backgroundColor: Colors.blue, // Defined in ThemeData (part of
MaterialApp)
      ),
      body: Center(
        child: Material( // Used for specific UI element
          elevation: 4.0, // Material property
          shape: RoundedRectangleBorder(borderRadius:
BorderRadius.circular(10.0)),
          child: const Text('This is a Material widget'),
        ),
      ),
    ),
  ));
}
```

OUTPUT:

In this example:

- MaterialApp configures the app's theme (including the blue color) and sets the Scaffold as the home screen.
- Scaffold is another Material widget that provides a basic layout structure for most screens in a Material Design app.
- AppBar and Text are not directly from Material, but they can be styled within the MaterialApp's theme.
- The Center widget positions the content in the center.
- Finally, the Material widget is used to create a specific button-like element with elevation and rounded corners.

StatelessWidget

When writing an app, you'll commonly author new widgets that are subclasses of either StatelessWidget or StatefulWidget, depending on whether your widget manages any state. A widget's main job is to implement a build() function, which describes the widget in terms of other, lower-level widgets.

Stateless widgets receive arguments from their parent widget, which they store in final member variables. When a widget is asked to build(), it uses these stored values to derive new arguments for the widgets it creates.

In Flutter, a StatelessWidget is a fundamental building block for user interfaces (UIs). It represents a part of your app's UI that remains static throughout its lifetime. Here's a breakdown of what it means:

Stateless:

- The key characteristic of a StatelessWidget is that it doesn't hold any internal state that can change. This means its appearance is entirely determined by the information passed to it when it's created, and it won't update dynamically on its own.

Widget:

- Everything in Flutter is a widget. Widgets are like building blocks that you assemble to create your app's UI. A StatelessWidget is a specific type of widget that specializes in presenting a fixed UI element.

Building UIs:

- StatelessWidgets are ideal for UI elements whose appearance is based on a set of properties or data provided to them when they are built. They describe how a part of the UI should look at a given point in time.

Benefits:

- Simpler to create and understand compared to StatefulWidgets (which manage state).
- More lightweight and efficient for static UI elements.
- Easier to reason about and test due to their predictable behavior.

Common Use Cases:

- Displaying text, icons, images, or simple layouts.
- Building UI elements that only depend on the data passed to them (e.g., displaying a product name from a list).
- Creating reusable UI components that present a fixed appearance.

In this example, GreetingText is a StatelessWidget that displays the text "Hello World!" with a specific font size and color. Its appearance remains constant, making it a suitable candidate for a StatelessWidget.

When to Use StatelessWidgets:

- If a UI element doesn't need to change its appearance based on user interaction or external data updates, use a StatelessWidget.
- It's a great choice for building simple and reusable UI components.
- For more complex UIs with dynamic behavior, consider using StatefulWidgets.

NOTE:

- Keys are unique identifiers for widgets in Flutter's widget tree.
- They help Flutter track changes to the UI and optimize performance.

Example.

In this example, GreetingText() is a custom defined widget.

- **Naming Convention:** In Flutter, custom widgets typically start with a capital letter to differentiate them from built-in widgets provided by the Flutter framework (which are usually in lowercase).
- The **context** object serves as a handle that pinpoints a widget's exact position within the hierarchical structure of widgets that make up your app's UI (User Interface).
- Imagine your app's UI as an inverted tree, with MaterialApp at the root and all other widgets branching out from it. Context tells a widget exactly where it sits on that tree.

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

void main() {
  runApp(const MaterialApp(
    home: GreetingText(),
  ));
}


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

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

```
      appBar: AppBar(
        title: const Text('Flutter Greeting'), // Add a title to the app bar
      ),
      body: const Center(  // Center the text within the body
        child: Text(
          'Hello World!',
          style: TextStyle(
            fontSize: 32.0,
            color: Colors.blue,
          ),
        ),
      ),
    );
  }
}
```

Here's a detailed explanation of the code, focusing on the constructor const GreetingText ({super.key});**:**

1. Constructor Declaration:

- const GreetingText ({super.key}): This is the declaration of a constructor for the GreetingText class. It has some distinct features:
    - const **Prefix:** The const keyword indicates that this constructor creates an immutable instance of the class. This means that once a GreetingText object is created, its properties cannot be changed.
    - **Curly Braces:** The curly braces {} enclose a single optional parameter, super.key.

2. Forwarding Key to Superclass:

- super.key**:** This part of the constructor forwards a special parameter named key to the superclass constructor. In this case, the superclass is StatelessWidget (since GreetingText extends StatelessWidget).
- **Key Importance:** Keys are unique identifiers assigned to widgets in Flutter. They play a crucial role in helping the framework efficiently manage the widget tree and determine which widgets need to be rebuilt when the UI state changes.
- **Forwarding Benefits:** By forwarding the key to the superclass, GreetingText inherits the key management capabilities of StatelessWidget, ensuring proper handling of its identity and updates within the widget tree.

3. No Explicit Argument:

- You might be wondering why there's no explicit argument being passed to super.key. This is because key is an optional parameter that can be specified when creating a GreetingText widget. If a key is not provided, it will default to null.

4. Optional Parameter Syntax:

- The curly braces {} around super.key signify that it's an optional named parameter. However, if you don't provide a key, it's perfectly valid to create it without any arguments:

In summary:

- The constructor const GreetingText ({super.key}); allows for the creation of immutable GreetingText widgets with optional keys.
- It forwards the key parameter to the superclass for proper widget management.
- Understanding this syntax and the role of keys is essential for effective Flutter development.

OUTPUT:

Row widget.

A widget that displays its children in a horizontal array.

In Flutter, the Row widget is a fundamental layout widget used to arrange its child widgets horizontally in a single row. It's a core building block for creating linear UI elements like:

- Navigation bars with buttons or icons displayed side-by-side.
- Forms with labels and input fields aligned horizontally.
- Layouts with multiple images or text elements displayed in a row.

Key Characteristics:

- **Horizontal Arrangement:** Child widgets are placed from left to right within the available space.

- **Fixed Size:** The Row widget itself doesn't have an intrinsic size. Its size is determined by the combined size of its child widgets and the available space from the parent widget.
- **Multiple Children:** A Row can hold multiple child widgets, allowing you to create complex horizontal layouts.

Customizing Layout:

While child widgets are placed horizontally by default, you can control their alignment and distribution within the Row using two properties:

1. **MainAxisAlignment:** This property determines how the child widgets are positioned along the main axis (horizontal in this case). Options include:
    - MainAxisAlignment.start (default): Aligns child widgets to the left edge of the Row.
    - MainAxisAlignment.center: Centers child widgets horizontally within the Row.
    - MainAxisAlignment.end: Aligns child widgets to the right edge of the Row.
    - MainAxisAlignment.spaceBetween: Distributes child widgets evenly with spaces in between, except for the first and last ones, which are positioned at the edges.
    - MainAxisAlignment.spaceAround: Distributes child widgets evenly with spaces around them.
    - MainAxisAlignment.spaceEvenly: Distributes child widgets with equal space between them, including the first and last ones.
2. **CrossAxisAlignment:** This property controls how the child widgets are positioned along the cross-axis (vertically in this case). Options include:
    - CrossAxisAlignment.start (default): Aligns the top edges of child widgets.
    - CrossAxisAlignment.center: Centers child widgets vertically within the Row.
    - CrossAxisAlignment.end: Aligns the bottom edges of child widgets.
    - CrossAxisAlignment.stretch (stretches child widgets to fill the vertical space of the Row).

Example:

Simple example of Row widget.

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

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

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

  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      home: Scaffold(
        appBar: AppBar(
          title: const Text('Row Example'),
        ),
        body: const Center(
          child: Row(
            mainAxisAlignment: MainAxisAlignment.spaceEvenly,
            children: [
              Text('Item 1', style: TextStyle(color: Colors.black)),
              Text('Item 2', style: TextStyle(color: Colors.black)),
              Text('Item 3', style: TextStyle(color: Colors.black)),
            ],
          ),
        ),
      ),
    );
  }
}
```

Output:

Making Child Widgets Flexible:

If a Row has more child widgets than can fit comfortably in the available space, they might overflow. To handle this, you can wrap child widgets with the Expanded widget. This allows child widgets to flex and share the available horizontal space proportionally.

If the non-flexible contents of the row (those that are not wrapped in Expanded or Flexible widgets) are together wider than the row itself, then the row is said to have overflowed. When a row overflows, the row does not have any remaining space to share between its Expanded and Flexible children. The row reports this by drawing a yellow and black striped warning box on the edge that is overflowing. If there is room on the outside of the row, the amount of overflow is printed in red lettering.

Example:

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

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

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

  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      home: Scaffold(
        appBar: AppBar(
          title: const Text('Row Example'),
        ),
        body: const Center(
          child: Row(
            children: <Widget>[
              FlutterLogo(),
              Text(
                  "Flutter's hot reload helps you quickly and easily
experiment, build UIs, add features, and fix bug faster. Experience sub-second
reload times, without losing state, on emulators, simulators, and hardware for
iOS and Android."),
              Icon(Icons.sentiment_very_satisfied),
            ],
          ),
        ),
      ),
    );
  }
}
```

Output:

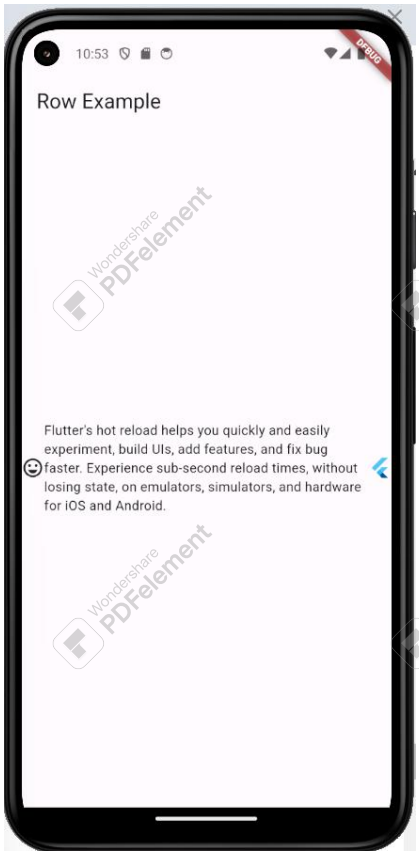
Example:

Controlling text direction in a row:

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

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

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

  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      home: Scaffold(
        appBar: AppBar(
          title: const Text('Row Example'),
        ),
        body: const Center(
          child: Row(
            textDirection: TextDirection.rtl,
```

```
            children: <Widget>[
              FlutterLogo(),
              Expanded(
                child: Text(
                    "Flutter's hot reload helps you quickly and easily
experiment, build UIs, add features, and fix bug faster. Experience sub-second
reload times, without losing state, on emulators, simulators, and hardware for
iOS and Android."),
              ),
              Icon(Icons.sentiment_very_satisfied),
            ],
          ),
        ),
      ),
    );
  }
}
```

Output:

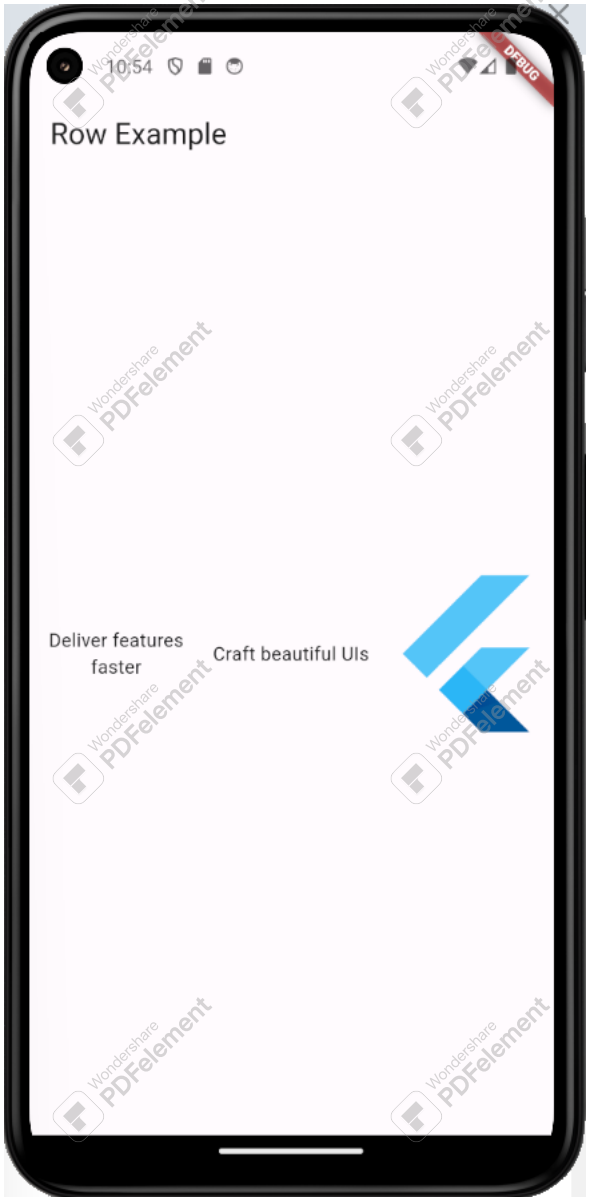


Another example of Expanded widget in Row

Example:

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

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

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

  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      home: Scaffold(
        appBar: AppBar(
          title: const Text('Row Example'),
        ),
        body: const Center(
          child: Row(
            children: <Widget>[
              Expanded(
                child: Text('Deliver features faster',
                    textAlign: TextAlign.center),
              ),
              Expanded(
                child: Text('Craft beautiful UIs', textAlign:
TextAlign.center),
              ),
              Expanded(
                child: FittedBox(
                  child: FlutterLogo(),
                ),
              ),
            ],
          ),
        ),
      ),
    );
  }
}
```

Output:



Example of mainAxisAlignment using Text widget

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

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

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

  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      home: Scaffold(
```

```
        appBar: AppBar(
          title: const Text('Row Example'),
        ),
        body: const Center(
          child: Row(
            mainAxisAlignment: MainAxisAlignment.spaceAround,
            children: <Widget>[
                Text("ITEM 1"),
                Text("ITEM 2"),
                Text("ITEM 3"),
            ],
          ),
        ),
      ),
    );
  }
}
```

Output:

Example of mainAxisAlignment and crossAxisAlignment using SizedBox and Text widget:

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

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

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

  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      home: Scaffold(
        appBar: AppBar(
          title: const Text('Row Example'),
        ),
        body: const Center(
          child: Row(
            mainAxisAlignment: MainAxisAlignment.spaceAround,
            crossAxisAlignment: CrossAxisAlignment.center,
            children: <Widget>[
              SizedBox(
                height: 20.0, // Approach 2: Set height of SizedBox
                child: Text('ITEM 1'),
              ),
              SizedBox(
                height: 140.0, // Approach 2: Set height of SizedBox
                child: Text('ITEM 2'),
              ),
              SizedBox(
                height: 20.0, // Approach 2: Set height of SizedBox
                child: Text('ITEM 3'),
              )
            ],
          ),
        ),
      ),
    );
  }
}
```

Output:

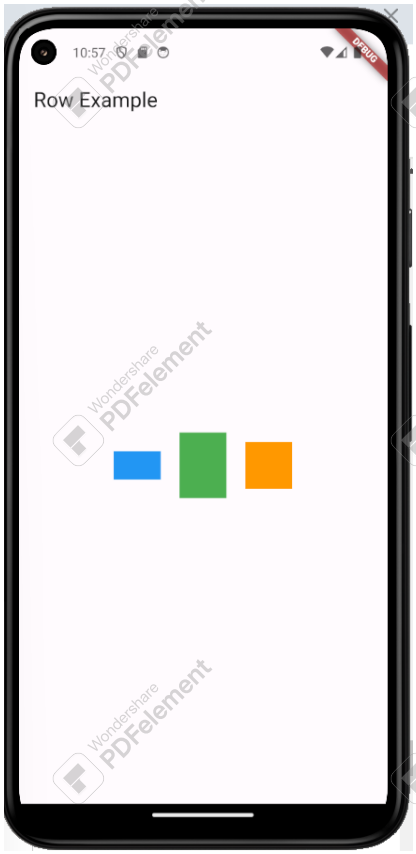
Example of mainAxisAlignment and crossAxisAlignment using Container Widget in Row

NOTE: If the heights of your container widgets are the same as the available space in the Row, you will see no difference in the crossAxisAlignment behavior.

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

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

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

  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      home: Scaffold(
        appBar: AppBar(
```

```
        title: const Text('Row Example'),
      ),
      body: Center(
        child: Row(
          mainAxisAlignment: MainAxisAlignment.center,
          crossAxisAlignment: CrossAxisAlignment.center,
          children: [
            Container(
              width: 50,
              height: 30,
              color: Colors.blue,
            ),
            const SizedBox(width: 20),
            Container(
              width: 50,
              height: 70,
              color: Colors.green,
            ),
            const SizedBox(width: 20),
            Container(
              width: 50,
              height: 50,
              color: Colors.orange,
            ),
          ],
        ),
      ),
    );
  }
}
```

Output:



Column widget example

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

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

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

  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      home: Scaffold(
        appBar: AppBar(
          title: const Text('Row Example'),
        ),
        body: Center(
          child: Column(
            mainAxisAlignment: MainAxisAlignment.start,
            crossAxisAlignment: CrossAxisAlignment.start,
            children: [
              Container(
```

```
            width: 150,
            height: 30,
            color: Colors.blue,
          ),
          const SizedBox(width: 20),
          Container(
            width: 80,
            height: 70,
            color: Colors.green,
          ),
          const SizedBox(width: 20),
          Container(
            width: 40,
            height: 50,
            color: Colors.orange,
          ),
        ],
      ),
    ),
  );
 }
}
```

Example of combining different widgets

- The context object serves as a handle that pinpoints a widget's exact position within the hierarchical structure of widgets that make up your app's UI (User Interface).
- Imagine your app's UI as an inverted tree, with MaterialApp at the root and all other widgets branching out from it. Context tells a widget exactly where it sits on that tree.

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

void main() {
  runApp(
    const MaterialApp(
      title: 'My app', // used by the OS task switcher
      home: SafeArea(
        child: MyScaffold(),
      ),
    ),
  );
}

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

  @override
  Widget build(BuildContext context) {
    // Material is a conceptual piece
    // of paper on which the UI appears.
    return Material(
      // Column is a vertical, linear layout.
      child: Column(
        children: [
          MyAppBar(
            title: Text(
              'Example title',
              style: Theme.of(context) //
                  .primaryTextTheme
                  .titleLarge,
            ),
          ),
          const Expanded(
            child: Center(
              child: Text('Hello, world!'),
            ),
          ),
        ],
```

```
      ),
    );
  }
}


class MyAppBar extends StatelessWidget {
  const MyAppBar({required this.title, super.key});

  // Fields in a Widget subclass are always marked "final".

  final Widget title;

  @override
  Widget build(BuildContext context) {
    return Container(
      height: 56, // in logical pixels
      padding: const EdgeInsets.symmetric(horizontal: 8),
      decoration: BoxDecoration(color: Colors.blue[500]),

      // Row is a horizontal, linear layout.
      child: Row(
        children: [
          const IconButton(
            icon: Icon(Icons.menu),
            tooltip: 'Navigation menu',
            onPressed: null, // null disables the button
          ),
          // Expanded expands its child
          // to fill the available space.
          Expanded(
            child: title,
          ),
          const IconButton(
            icon: Icon(Icons.search),
            tooltip: 'Search',
            onPressed: null,
          ),
        ],
      ),
    );
  }
}
```

Output:

Example: Using the MaterialApp and Scaffold Widget with a button

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

void main() {
  runApp(
    const MaterialApp(
      title: 'Flutter Tutorial',
      home: TutorialHome(),
    ),
  );
}

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

  @override
  Widget build(BuildContext context) {
    // Scaffold is a layout for
    // the major Material Components.
```

```
    return Scaffold(
      appBar: AppBar(
        leading: const IconButton(
          icon: Icon(Icons.menu),
          tooltip: 'Navigation menu',
          onPressed: null,
        ),
        title: const Text('Example title'),
        actions: const [
          IconButton(
            icon: Icon(Icons.search),
            tooltip: 'Search',
            onPressed: null,
          ),
        ],
      ),
      // body is the majority of the screen.
      body: const Center(
        child: Text('Hello, world!'),
      ),
      floatingActionButton: const FloatingActionButton(
        tooltip: 'Add', // used by assistive technologies
        onPressed: null,
        child: Icon(Icons.add),
      ),
    );
  }
}
```

Explanation:

Notice that widgets are passed as arguments to other widgets. The `Scaffold` widget takes a number of different widgets as named arguments, each of which are placed in the `Scaffold` layout in the appropriate place. Similarly, the `AppBar` widget lets you pass in widgets for the `leading` widget, and the `actions` of the `title` widget. This pattern recurs throughout the framework and is something you might consider when designing your own widgets.

Example: Adding a simple button and its function:

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

void main() {
  runApp( MaterialApp(
    home: Scaffold(
      appBar: AppBar(
        title: const Text("This is title"),
        centerTitle: true,
        backgroundColor: Colors.blue,
      ),
      body: Center(
        child: Text("This is body") ),
        floatingActionButton: FloatingActionButton(
          onPressed:() => showHelloWorld(),
          child: Text('Click'),

        ),
    )
  ));
```

```
}

void showHelloWorld() {

  print("Hello, World!");
  // You can replace the print statement with any code to display "Hello,
World!" on the screen.
}
```

Example: Add a simple button and print message using print function

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

void main() {
  runApp(
    const MaterialApp(
      home: Scaffold(
        body: Center(
          child: MyButton(),
        ),
      ),
    ),
  );
}

class MyButton extends StatelessWidget {

const MyButton( {super.key});


  @override
  Widget build(BuildContext context) {

    return GestureDetector(
      onTap: () {
        print('MyButton was tapped!');
      },
      child: Container(
        height: 50,
        padding: const EdgeInsets.all(8),
        margin: const EdgeInsets.symmetric(horizontal: 8),
        decoration: BoxDecoration(
          borderRadius: BorderRadius.circular(5),
          color: Colors.lightGreen[500],
        ),
        child: const Center(
          child: Text('Engage'),
```

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

Output:

The output will be shown in blue text of debug console.

Example: Add a simple button and print message using ScaffoldMessenger

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

void main() {
  runApp(
    const MaterialApp(
      home: Scaffold(
        body: Center(
          child: MyButton(),
        ),
      ),
    ),
  );
}

class MyButton extends StatelessWidget {

const MyButton( {super.key});


  @override
  Widget build(BuildContext context) {

    return GestureDetector(
      onTap: () {
        ScaffoldMessenger.of(context).showSnackBar(
          const SnackBar(
            content: Text('MyButton was tapped!'),
            backgroundColor: Colors.blue,
          ),
        );
      },
      child: Container(
```

```
        height: 50,
        padding: const EdgeInsets.all(8),
        margin: const EdgeInsets.symmetric(horizontal: 8),
        decoration: BoxDecoration(
          borderRadius: BorderRadius.circular(5),
          color: Colors.lightGreen[500],
        ),
        child: const Center(
          child: Text('Engage'),
        ),
      ),
    );
  }
}
```

The GestureDetector widget doesn't have a visual representation but instead detects gestures made by the user. When the user taps the Container, the GestureDetector calls its onTap() callback, in this case printing a message to the console. You can use GestureDetector to detect a variety of input gestures, including taps, drags, and scales.

Output:

The blue band at below of screen is output

StatefulWidget

StatefulWidgets are special widgets that know how to generate State objects, which are then used to hold state.

Example: Display message in Text field on button click

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

void main() {
  runApp(
    const MaterialApp(
      home: Scaffold(
        body: Center(
          child: Display(),
        ),
      ),
    ),
  );
}
```

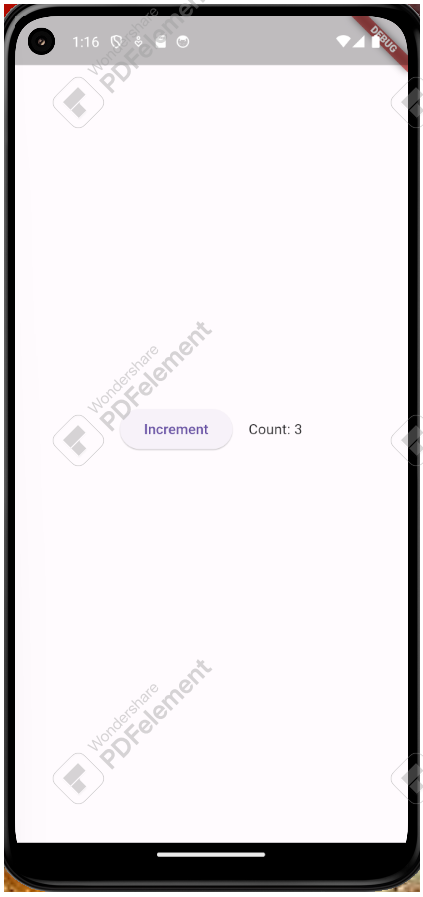
```
class Display extends StatefulWidget {
  const Display({super.key});

  @override
  State<Display> createState() => _DisplayState();
}

class _DisplayState extends State<Display> {
  String message = ""; // Store the message to display

  void _displayMessage() {
    setState(() {
      message = "Hello World"; // Update the message on button click
    });
  }

  @override
  Widget build(BuildContext context) {
    return Row( // Modified to display the message
      mainAxisAlignment: MainAxisAlignment.center,
      children: <Widget>[
        ElevatedButton(
          onPressed: _displayMessage, // Call the new function on button press
          child: const Text('Display Message'), // Updated button text
        ),
        const SizedBox(width: 16),
        Text(message), // Display the message instead of the Display
      ],
    );
  }
}
```

Example: A counter application

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

class Counter extends StatefulWidget {
  // This class is the configuration for the state.
  // It holds the values (in this case nothing) provided
  // by the parent and used by the build  method of the
  // State. Fields in a Widget subclass are always marked
  // "final".

  const Counter({super.key});

  @override
  State<Counter> createState() => _CounterState();
}

class _CounterState extends State<Counter> {
  int _counter = 0;

  void _increment() {
    setState(() {
      // This call to setState tells the Flutter framework
      // that something has changed in this State, which
      // causes it to rerun the build method below so that
      // the display can reflect the updated values. If you
```

```
      // change _counter without calling setState(), then
      // the build method won't be called again, and so
      // nothing would appear to happen.
      _counter++;
    });
  }

  @override
  Widget build(BuildContext context) {
    // This method is rerun every time setState is called,
    // for instance, as done by the _increment method above.
    // The Flutter framework has been optimized to make
    // rerunning build methods fast, so that you can just
    // rebuild anything that needs updating rather than
    // having to individually changes instances of widgets.
    return Row(
      mainAxisAlignment: MainAxisAlignment.center,
      children: <Widget>[
        ElevatedButton(
          onPressed: _increment,
          child: const Text('Increment'),
        ),
        const SizedBox(width: 16),
        Text('Count: $_counter'),
      ],
    );
  }
}

void main() {
  runApp(
    const MaterialApp(
      home: Scaffold(
        body: Center(
          child: Counter(),
        ),
      ),
    ),
  );
}
```

Output:

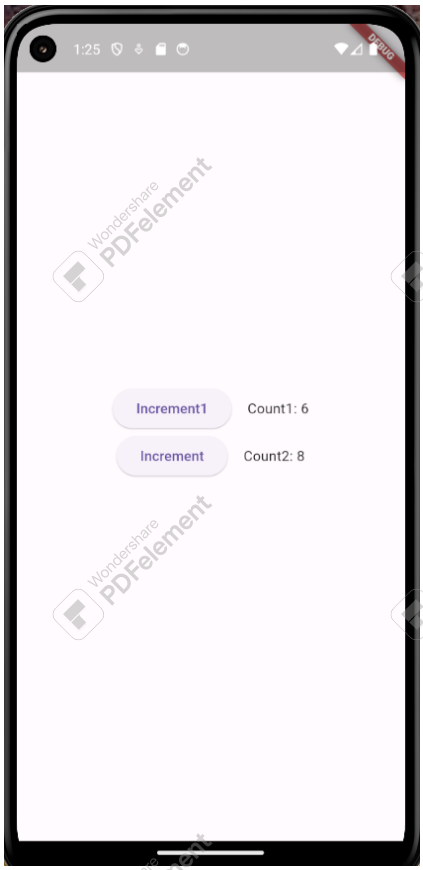


Example: Two counters

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

class Counter extends StatefulWidget {

  const Counter({super.key});

  @override
  State<Counter> createState() => _CounterState();
}

class _CounterState extends State<Counter> {
  int _counter1 = 0;
  int _counter2 = 0;

  void _increment1() {
    setState(() {
      _counter1++;
    });
  }

  void _increment2() {
```

```
    setState(() {
      _counter2++;
    });
  }

  @override
  Widget build(BuildContext context) {
    return Column(
      mainAxisAlignment: MainAxisAlignment.center,
      children: <Widget>[
        Row(
          mainAxisAlignment: MainAxisAlignment.center,
          children: <Widget>[
            ElevatedButton(
              onPressed: _increment1,
              child: const Text('Increment1'),
            ),
            const SizedBox(width: 16),
            Text('Count1: $_counter1'),
          ],
        ),
        Row(
          mainAxisAlignment: MainAxisAlignment.center,
          children: <Widget>[
            ElevatedButton(
              onPressed: _increment2,
              child: const Text('Increment'),
            ),
            const SizedBox(width: 16),
            Text('Count2: $_counter2'),
          ],
        ),
      ],
    );
  }
}

void main() {
  runApp(
    const MaterialApp(
      home: Scaffold(
        body: Center(
          child: Counter(),
        ),
      ),
    ),
  );
}
```

Output:



Example: Two counters and their sum.

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

void main() {
  runApp(
    const MaterialApp(
      home: Scaffold(
        body: Center(
          child: Counter(),
        ),
      ),
    ),
  );
}

class Counter extends StatefulWidget {

  const Counter({super.key});
```

```
  @override
  State<Counter> createState() => _CounterState();
}

class _CounterState extends State<Counter> {
  int _counter1 = 0;
  int _counter2 = 0;
  int _sum = 0;

  void _increment1() {
    setState(() {
      _counter1++;
    });
  }

  void _increment2() {
    setState(() {
      _counter2++;
    });
  }

  void _sumcounter() {
    setState(() {
      _sum = _counter1 + _counter2;
    });

  }

  @override
  Widget build(BuildContext context) {
  return Column( // Use Column for vertical layout
    mainAxisAlignment: MainAxisAlignment.center, // Center the content
vertically
    children: <Widget>[
      Row( // First row with a button and Text
        mainAxisAlignment: MainAxisAlignment.center, // Center content in the
row
        children: <Widget>[
          ElevatedButton(
            onPressed: _increment1,
            child: const Text('Increment1'),
          ),
          const SizedBox(width: 16), // Spacing between elements
          Text('Count: $_counter1'),
        ],
      ),
      const SizedBox(height: 16), // Spacing between rows
      Row( // Second row with another button and Text
```

```
        mainAxisAlignment: MainAxisAlignment.center, // Center content in the
row
        children: <Widget>[
          ElevatedButton(
            onPressed: _increment2,
            child: const Text('Increment2'),
          ),
          const SizedBox(width: 16), // Spacing between elements
          Text('Count: $_counter2'),
        ],
      ),
      const SizedBox(height: 16), // Spacing between rows
      Row( // Second row with another button and Text
        mainAxisAlignment: MainAxisAlignment.center, // Center content in the
row
        children: <Widget>[
          ElevatedButton(
            onPressed: _sumcounter,
            child: const Text('Sum'),
          ),
          const SizedBox(width: 16), // Spacing between elements
          Text('Ans: $_sum'),
        ],
      ),
    ],
  );
}
}
```

Output:

Example: Introducing separate widgets for counter increment and counter display:

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

void main() {
  runApp(
    const MaterialApp(
      home: Scaffold(
        body: Center(
          child: Counter(),
        ),
      ),
    ),
  );
}

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

  @override
```

```
  State<Counter> createState() => _CounterState();
}

class _CounterState extends State<Counter> {
  int _counter = 0;

  void _increment() {
    setState(() {
      ++_counter;
    });
  }

  @override
  Widget build(BuildContext context) {
    return Row(
      mainAxisAlignment: MainAxisAlignment.center,
      children: <Widget>[
        CounterIncrementor(onPressed: _increment),
        const SizedBox(width: 16),
        CounterDisplay(count: _counter),
      ],
    );
  }
}

class CounterIncrementor extends StatelessWidget {
  const CounterIncrementor({required this.onPressed, super.key});

  final VoidCallback onPressed;

  @override
  Widget build(BuildContext context) {
    return ElevatedButton(
      onPressed: onPressed,
      child: const Text('Increment'),
    );
  }
}

class CounterDisplay extends StatelessWidget {
  const CounterDisplay({required this.count, super.key});

  final int count;

  @override
  Widget build(BuildContext context) {
    return Text('Count: $count');
  }
```

```
}
```

Output:

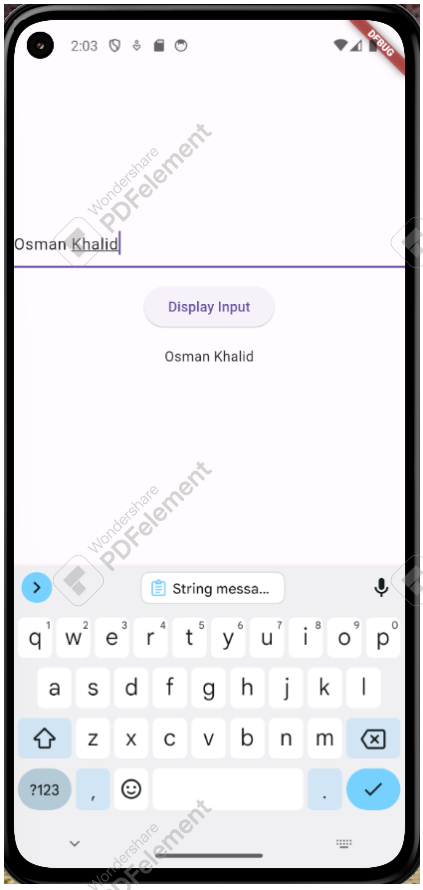
The output is same to the single counter app.

Example to take user input using onChanged method and display as Text field

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

void main() {
  runApp(
    const MaterialApp(
      home: Scaffold(
        body: Center(
          child: InputDisplay(),
        ),
      ),
    ),
  );
}

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

  @override
  State<InputDisplay> createState() => _InputDisplayState();
}

class _InputDisplayState extends State<InputDisplay> {
  String userInput = ""; // Store user input
  String message = "";

  void _displayInput() {
    setState(() {
      message = userInput; // Update message with user input
    });
  }

  @override
  Widget build(BuildContext context) {
    return Column( // Use Column for vertical layout
      mainAxisAlignment: MainAxisAlignment.center,
      children: <Widget>[
        TextField( // Text input field
```

```
        onChanged: (value) => setState(() => userInput = value), // Update
userInput on change
          decoration: const InputDecoration(
            hintText: 'Enter your message', // Hint text for the user
          ),
        ),
        const SizedBox(height: 16),
        ElevatedButton(
          onPressed: _displayInput, // Call the function to display input
          child: const Text('Display Input'), // Updated button text
        ),
        const SizedBox(height: 16),
        Text(message), // Display the message
      ],
    );
  }
}
```

Output:



Assigning value of one TextField to other on button click

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

void main() {
  runApp(const MaterialApp(
    home: Scaffold(
      body: Center(
        child: InputDisplay(),
      )
    )
  ));
}

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

  @override
  State<InputDisplay> createState() => _InputDisplayState();
}

class _InputDisplayState extends State<InputDisplay> {

String message = "";
String message1="";
String userInput="";



@override
Widget build(BuildContext context) {
  return Column(
    mainAxisAlignment: MainAxisAlignment.center,
    children: <Widget>[
      TextField(
        onChanged: (value) => setState(() => userInput = value
        ),
        decoration: const InputDecoration(
          hintText: 'Enter your message'
        ),
      ),

      ElevatedButton(onPressed: () => setState(

        () {
          message = userInput;
          message1 = userInput;
          }
```

```
        ), child: Text("Press button")),

      Text(message),
      Text(message1),
      TextField(
        controller: TextEditingController(text: message),
        decoration: const InputDecoration(
          hintText: 'Enter your message'
        ),
      ),
    ],
  );
}

}
```

Real-time update of one TextField through other

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

void main() {
  runApp(const MaterialApp(
    home: Scaffold(
      body: Center(
        child: InputDisplay(),
      )
    )
  ));
}

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

  @override
  State<InputDisplay> createState() => _InputDisplayState();
}

class _InputDisplayState extends State<InputDisplay> {

String message = "";
String message1="";
String userInput="";
```

```
@override
Widget build(BuildContext context) {
  return Column(
    mainAxisAlignment: MainAxisAlignment.center,
    children: <Widget>[
      TextField(
        onChanged: (value) => setState(() => userInput = value
          ),
        decoration: const InputDecoration(
          hintText: 'Enter your message'
        ),
      ),

      TextField(
        controller: TextEditingController(text: userInput.toUpperCase()),

        decoration: const InputDecoration(
          hintText: 'Enter your message'
        ),
      ),
    ],
  );
}

}
```

Example of two TextFields using TextEditingController class and display with button

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

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

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

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

```dart
class _MyAppState extends State<MyApp> {
  // Controllers to store text from each box
  final TextEditingController _textController1 = TextEditingController();
  final TextEditingController _textController2 = TextEditingController();
  String _displayText = ""; // Variable to store combined text

  void _onPressed() {
    // Combine text from controllers and update display text
    setState(() {
      _displayText = "${_textController1.text} - ${_textController2.text}";
    });
  }

  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      home: Scaffold(
        appBar: AppBar(
          title: const Text('Text Input Example'),
        ),
        body: Center(
          child: Column(
            mainAxisAlignment: MainAxisAlignment.center,
            children: [
              TextField(
                controller: _textController1,
                decoration: const InputDecoration(
                  hintText: 'Enter Text 1',
                ),
              ),
              const SizedBox(height: 10),
              TextField(
                controller: _textController2,
                decoration: const InputDecoration(
                  hintText: 'Enter Text 2',
                ),
              ),
              const SizedBox(height: 20),
              ElevatedButton(
                onPressed: _onPressed,
                child: const Text('Display Text'),
              ),
              const SizedBox(height: 10),
              Text(_displayText),
            ],
          ),
        ),
```

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

An input form example

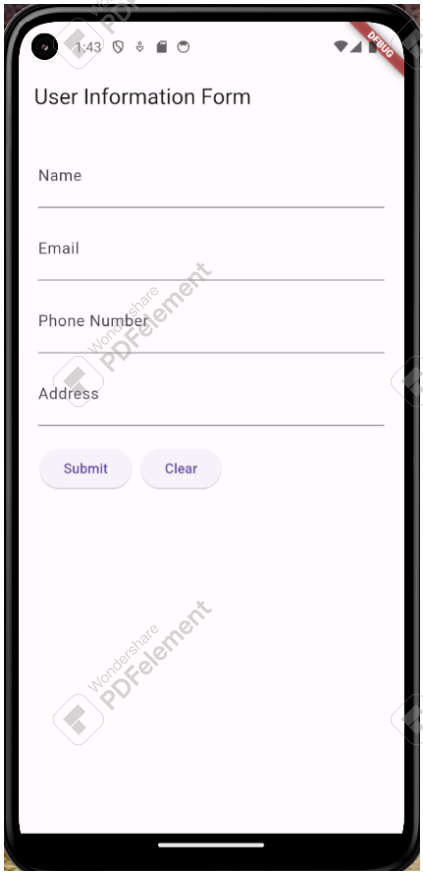
```
import 'package:flutter/material.dart';

void main() {
```

```
  runApp(const MaterialApp(
    home: ResponsiveForm(),
  ));
}

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

  @override
  State<ResponsiveForm> createState() => _ResponsiveFormState();
}

class _ResponsiveFormState extends State<ResponsiveForm> {
  // Add state variables for form fields
  final _nameController = TextEditingController();
  final _emailController = TextEditingController();
  final _phoneController = TextEditingController();
  final _addressController = TextEditingController();

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: const Text('User Information Form'),
      ),
      body: SingleChildScrollView(
        padding: const EdgeInsets.all(20.0),
        child: LayoutBuilder(
          builder: (context, constraints) {
            return Row(
              children: [
                Expanded(
                  child: _buildForm(context),
                ),
              ],
            );
          },
        ),
      ),
    );
  }

  Widget _buildForm(BuildContext context) {
    return Column(
      crossAxisAlignment: CrossAxisAlignment.start,
      children: [
        TextField(
          controller: _nameController,
```

```
        decoration: const InputDecoration(
          labelText: 'Name',
        ),
      ),
      const SizedBox(height: 10.0),
      TextField(
        controller: _emailController,
        decoration: const InputDecoration(
          labelText: 'Email',
        ),
        keyboardType: TextInputType.emailAddress,
      ),
      const SizedBox(height: 10.0),
      TextField(
        controller: _phoneController,
        decoration: const InputDecoration(
          labelText: 'Phone Number',
        ),
        keyboardType: TextInputType.phone,
      ),
      const SizedBox(height: 10.0),
      TextField(
        controller: _addressController,
        decoration: const InputDecoration(
          labelText: 'Address',
        ),
        maxLines: null, // This allows multiline input
      ),
      const SizedBox(height: 20.0),
      Row(
        children: [
          const SizedBox(width: 2.0),
          ElevatedButton(
            onPressed: () => showValues(context),
            child: const Text('Submit'),
          ),
          const SizedBox(width: 10.0),
          ElevatedButton(
            onPressed: () => {},
            child: const Text('Clear'),
          ),
        ],
      ),
    ],
  );
}

void showValues(BuildContext context) {
```

```
    // Print the current values of the form fields
    debugPrint('Name: ${_nameController.text}');
    debugPrint('Email: ${_emailController.text}');
    debugPrint('Phone: ${_phoneController.text}');
    debugPrint('Address: ${_addressController.text}');

    ScaffoldMessenger.of(context).showSnackBar(
          const SnackBar(
            content: Text('Form is submitted!'),
            backgroundColor: Colors.blue,
          ),
        );

  }
}
```

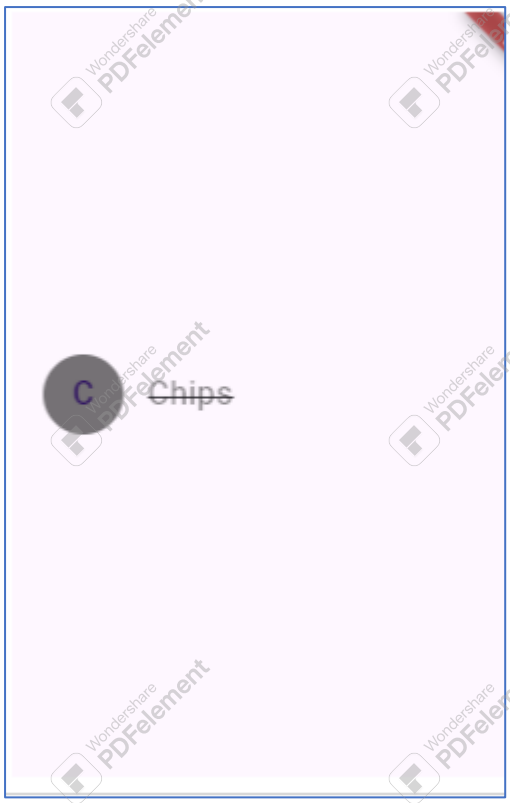
Output:



Shopping Cart with single item:

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


void main() {
  runApp(
    MaterialApp(
      home: Scaffold(
        body: Center(
          child: ShoppingListItem(
            product: const Product(name: 'Chips'),
            inCart: true,
            onCartChanged: (product, inCart) {},
          ),
        ),
      ),
    ),
  );
}

typedef CartChangedCallback = Function(Product product, bool inCart);


class ShoppingListItem extends StatelessWidget {

  final Product product;
  final bool inCart;
  final CartChangedCallback onCartChanged;


  ShoppingListItem({
    required this.product,
    required this.inCart,
    required this.onCartChanged,
  }) : super(key: ObjectKey(product));


  // The below two methods _getColor and _getTextStyle are related to styling
//

  Color _getColor(BuildContext context) {
    // The theme depends on the BuildContext because different
    // parts of the tree can have different themes.
    // The BuildContext indicates where the build is
    // taking place and therefore which theme to use.

    return inCart //
        ? Colors.black54
        : Theme.of(context).primaryColor;
  }
```

```
  TextStyle? _getTextStyle(BuildContext context) {
    if (!inCart) return null;

    return const TextStyle(
      color: Colors.black54,
      decoration: TextDecoration.lineThrough,
    );
  }

  @override
  Widget build(BuildContext context) {
    return ListTile(
      onTap: () {
        onCartChanged(product, inCart);
      },
      leading: CircleAvatar(
        backgroundColor: _getColor(context),
        child: Text(product.name[0]),
      ),
      title: Text(product.name, style: _getTextStyle(context)),
    );
  }
}

class Product {
  const Product({required this.name});

  final String name;
}
```



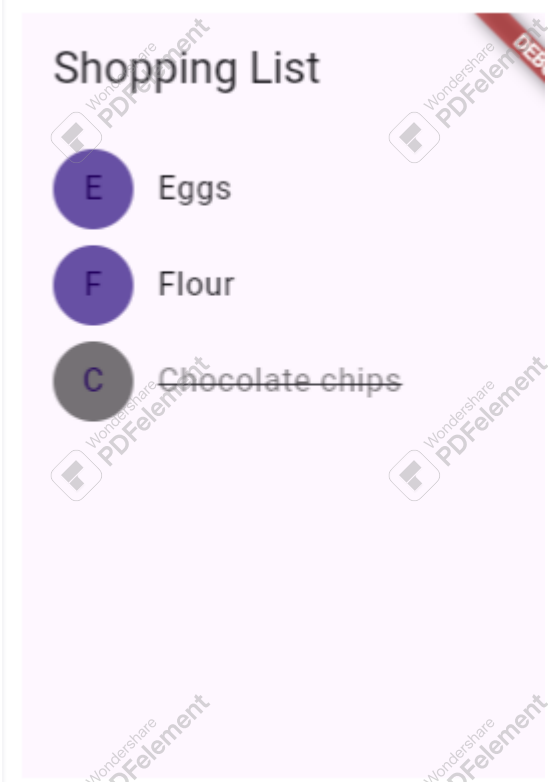
A shopping cart with multiple Items

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

void main() {
  runApp(const MaterialApp(
    title: 'Shopping App',
    home: ShoppingList(
      products: [
        Product(name: 'Eggs'),
        Product(name: 'Flour'),
        Product(name: 'Chocolate chips'),
      ],
    ),
  ));
}

class Product {
  const Product({required this.name});

  final String name;
}

typedef CartChangedCallback = Function(Product product, bool inCart);
```

```dart
class ShoppingListItem extends StatelessWidget {
  ShoppingListItem({
    required this.product,
    required this.inCart,
    required this.onCartChanged,
  }) : super(key: ObjectKey(product));

  final Product product;
  final bool inCart;
  final CartChangedCallback onCartChanged;

  Color _getColor(BuildContext context) {
    // The theme depends on the BuildContext because different
    // parts of the tree can have different themes.
    // The BuildContext indicates where the build is
    // taking place and therefore which theme to use.

    return inCart //
        ? Colors.black54
        : Theme.of(context).primaryColor;
  }

  TextStyle? _getTextStyle(BuildContext context) {
    if (!inCart) return null;

    return const TextStyle(
      color: Colors.black54,
      decoration: TextDecoration.lineThrough,
    );
  }

  @override
  Widget build(BuildContext context) {
    return ListTile(
      onTap: () {
        onCartChanged(product, inCart);
      },
      leading: CircleAvatar(
        backgroundColor: _getColor(context),
        child: Text(product.name[0]),
      ),
      title: Text(
        product.name,
        style: _getTextStyle(context),
      ),
    );
  }
}
```

```dart
class ShoppingList extends StatefulWidget {
  const ShoppingList({required this.products, super.key});

  final List<Product> products;

  // The framework calls createState the first time
  // a widget appears at a given location in the tree.
  // If the parent rebuilds and uses the same type of
  // widget (with the same key), the framework re-uses
  // the State object instead of creating a new State object.

  @override
  State<ShoppingList> createState() => _ShoppingListState();
}

class _ShoppingListState extends State<ShoppingList> {
  final _shoppingCart = <Product>{};

  void _handleCartChanged(Product product, bool inCart) {
    setState(() {
      // When a user changes what's in the cart, you need
      // to change _shoppingCart inside a setState call to
      // trigger a rebuild.
      // The framework then calls build, below,
      // which updates the visual appearance of the app.

      if (!inCart) {
        _shoppingCart.add(product);
      } else {
        _shoppingCart.remove(product);
      }
    });
  }

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: const Text('Shopping List'),
      ),
      body: ListView(
        padding: const EdgeInsets.symmetric(vertical: 8),
        children: widget.products.map((product) {
          return ShoppingListItem(
            product: product,
            inCart: _shoppingCart.contains(product),
            onCartChanged: _handleCartChanged,
```

```
        );
      }).toList(),
    ),
  );
 }
}
```



Creating a layout

To include images, add an `assets` tag to the `pubspec.yaml` file at the root directory of your app. When you add `assets`, it serves as the set of pointers to the images available to your code.

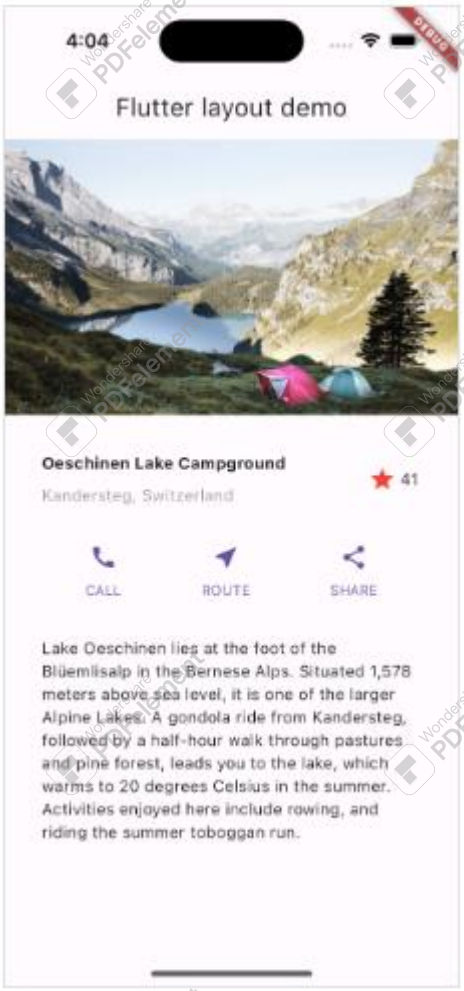
```
@@ -19,3 +19,5 @@
19 19  flutter:
20 20
   uses-material-design: true
21+ assets:
22+ - images/lake.jpg
```

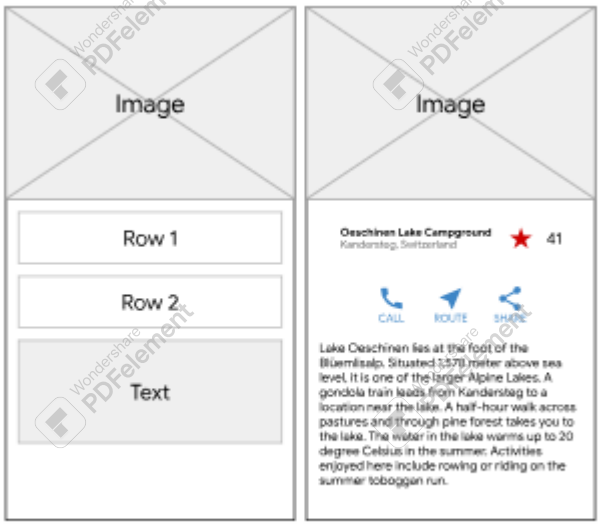
```dart
import 'package:flutter/material.dart';

void main() => runApp(const MyApp());

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

  @override
  Widget build(BuildContext context) {
    const String appTitle = 'Flutter layout demo';
    return MaterialApp(
      title: appTitle,
      home: Scaffold(
        appBar: AppBar(
          title: const Text(appTitle),
        ),
        body: const Center(
          child: const SingleChildScrollView(
              child: Column(
            children: [
              ImageSection(
                image: 'images/lake.jpg',
              ),
              TitleSection(
                name: 'Oeschinen Lake Campground',
                location: 'Kandersteg, Switzerland',
              ),
              ButtonSection(),
              TextSection(
                description:
                    'Lake Oeschinen lies at the foot of the Blüemlisalp in the
'
                    'Bernese Alps. Situated 1,578 meters above sea level, it '
                    'is one of the larger Alpine Lakes. A gondola ride from '
                    'Kandersteg, followed by a half-hour walk through pastures
'
                    'and pine forest, leads you to the lake, which warms to 20
'
                    'degrees Celsius in the summer. Activities enjoyed here '
                    'include rowing, and riding the summer toboggan run.',
              ),
            ],
          )),
        ),
      ),
    );
  }
}
```

```dart
class TitleSection extends StatelessWidget {
  const TitleSection({
    super.key,
    required this.name,
    required this.location,
  });

  final String name;
  final String location;

  @override
  Widget build(BuildContext context) {
    return Padding(
      padding: const EdgeInsets.all(32),
      child: Row(
        children: [
          Expanded(
            /*1*/
            child: Column(
              crossAxisAlignment: CrossAxisAlignment.start,
              children: [
                /*2*/
                Padding(
                  padding: const EdgeInsets.only(bottom: 8),
                  child: Text(
                    name,
                    style: const TextStyle(
                      fontWeight: FontWeight.bold,
                    ),
                  ),
                ),
                Text(
                  location,
                  style: TextStyle(
                    color: Colors.grey[500],
                  ),
                ),
              ],
            ),
          ),
          /*3*/
          Icon(
            Icons.star,
            color: Colors.red[500],
          ),
          const Text('41'),
        ],
```

```
      ),
    );
  }
}

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

  @override
  Widget build(BuildContext context) {
    final Color color = Theme.of(context).primaryColor;
    return SizedBox(
      child: Row(
        mainAxisAlignment: MainAxisAlignment.spaceEvenly,
        children: [
          ButtonWithText(
            color: color,
            icon: Icons.call,
            label: 'CALL',
          ),
          ButtonWithText(
            color: color,
            icon: Icons.near_me,
            label: 'ROUTE',
          ),
          ButtonWithText(
            color: color,
            icon: Icons.share,
            label: 'SHARE',
          ),
        ],
      ),
    );
  }
}

class ButtonWithText extends StatelessWidget {
  const ButtonWithText({
    super.key,
    required this.color,
    required this.icon,
    required this.label,
  });

  final Color color;
  final IconData icon;
  final String label;
```

```
  @override
  Widget build(BuildContext context) {
    return Column(
// ...
        );
  }
}

class TextSection extends StatelessWidget {
  const TextSection({
    super.key,
    required this.description,
  });

  final String description;

  @override
  Widget build(BuildContext context) {
    return Padding(
      padding: const EdgeInsets.all(32),
      child: Text(
        description,
        softWrap: true,
      ),
    );
  }
}

class ImageSection extends StatelessWidget {
  const ImageSection({super.key, required this.image});

  final String image;

  @override
  Widget build(BuildContext context) {
    return Image.asset(
      image,
      width: 600,
      height: 240,
      fit: BoxFit.cover,
    );
  }
}
```


4:04
DEBUG
Flutter layout demo
Oeschinen Lake Campground
Kandersteg, Switzerland
41
CALL
ROUTE
SHARE
Lake Oeschinen lies at the foot of the Blüemlisalp in the Bernese Alps. Situated 1,578 meters above sea level, it is one of the larger Alpine Lakes. A gondola ride from Kandersteg, followed by a half-hour walk through pastures and pine forest, leads you to the lake, which warms to 20 degrees Celsius in the summer. Activities enjoyed here include rowing, and riding the summer toboggan run.



The text block as sketch and prototype UI

Creating a ListView

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

void main() => runApp(const MyApp());

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

  @override
  Widget build(BuildContext context) {
    const title = 'Basic List';

    return MaterialApp(
      title: title,
      home: Scaffold(
        appBar: AppBar(
          title: const Text(title),
        ),
        body: ListView(
          children: const <Widget>[
            ListTile(
              leading: Icon(Icons.map),
              title: Text('Map'),
            ),
            ListTile(
              leading: Icon(Icons.photo_album),
              title: Text('Album'),
            ),
            ListTile(
```

```
            leading: Icon(Icons.phone),
            title: Text('Phone'),
          ),
        ],
      ),
    ),
  );
 }
}
```

Output:

Creating a horizontal ListView

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

void main() => runApp(const MyApp());

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

  @override
```

```
  Widget build(BuildContext context) {
    const title = 'Horizontal List';

    return MaterialApp(
      title: title,
      home: Scaffold(
        appBar: AppBar(
          title: const Text(title),
        ),
        body: Container(
          margin: const EdgeInsets.symmetric(vertical: 20),
          height: 200,
          child: ListView(
            // This next line does the trick.
            scrollDirection: Axis.horizontal,
            children: <Widget>[
              Container(
                width: 160,
                color: Colors.red,
              ),
              Container(
                width: 160,
                color: Colors.blue,
              ),
              Container(
                width: 160,
                color: Colors.green,
              ),
              Container(
                width: 160,
                color: Colors.yellow,
              ),
              Container(
                width: 160,
                color: Colors.orange,
              ),
            ],
          ),
        ),
      ),
    );
  }
}
```

Creating a Grid List

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

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

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

  @override
  Widget build(BuildContext context) {
    const title = 'Grid List';

    return MaterialApp(
      title: title,
      home: Scaffold(
        appBar: AppBar(
          title: const Text(title),
```

```
        ),
        body: GridView.count(
          // Create a grid with 2 columns. If you change the scrollDirection
to
          // horizontal, this produces 2 rows.
          crossAxisCount: 2,
          // Generate 100 widgets that display their index in the List.
          children: List.generate(100, (index) {
            return Center(
              child: Text(
                'Item $index',
                style: Theme.of(context).textTheme.headlineSmall,
              ),
            );
          }),
        ),
      ),
    );
  }
}
```

Output:

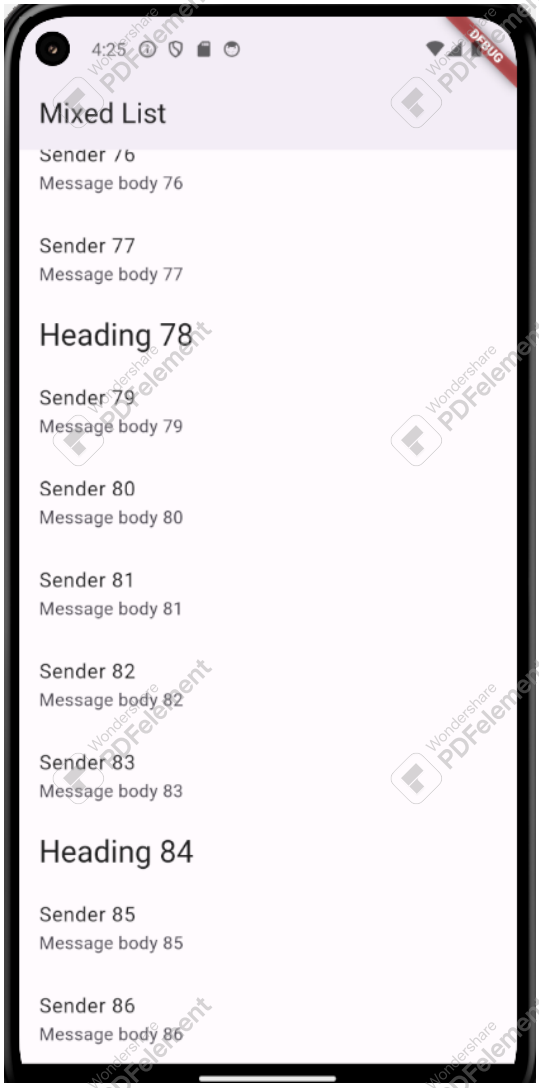
Create lists with different types of items

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

void main() {
  runApp(
    MyApp(
      items: List<ListItem>.generate(
        1000,
        (i) => i % 6 == 0
            ? HeadingItem('Heading $i')
            : MessageItem('Sender $i', 'Message body $i'),
      ),
```

```
    ),
  );
}

class MyApp extends StatelessWidget {
  final List<ListItem> items;

  const MyApp({super.key, required this.items});

  @override
  Widget build(BuildContext context) {
    const title = 'Mixed List';

    return MaterialApp(
      title: title,
      home: Scaffold(
        appBar: AppBar(
          title: const Text(title),
        ),
        body: ListView.builder(
          // Let the ListView know how many items it needs to build.
          itemCount: items.length,
          // Provide a builder function. This is where the magic happens.
          // Convert each item into a widget based on the type of item it is.
          itemBuilder: (context, index) {
            final item = items[index];

            return ListTile(
              title: item.buildTitle(context),
              subtitle: item.buildSubtitle(context),
            );
          },
        ),
      ),
    );
  }
}

/// The base class for the different types of items the list can contain.
abstract class ListItem {
  /// The title line to show in a list item.
  Widget buildTitle(BuildContext context);

  /// The subtitle line, if any, to show in a list item.
  Widget buildSubtitle(BuildContext context);
}

/// A ListItem that contains data to display a heading.
```

```dart
class HeadingItem implements ListItem {
  final String heading;

  HeadingItem(this.heading);

  @override
  Widget buildTitle(BuildContext context) {
    return Text(
      heading,
      style: Theme.of(context).textTheme.headlineSmall,
    );
  }

  @override
  Widget buildSubtitle(BuildContext context) => const SizedBox.shrink();
}

/// A ListItem that contains data to display a message.
class MessageItem implements ListItem {
  final String sender;
  final String body;

  MessageItem(this.sender, this.body);

  @override
  Widget buildTitle(BuildContext context) => Text(sender);

  @override
  Widget buildSubtitle(BuildContext context) => Text(body);
}
```

Output:



List with spaced items

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

void main() => runApp(const SpacedItemsList());

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

  @override
  Widget build(BuildContext context) {
    const items = 4;

    return MaterialApp(
      title: 'Flutter Demo',
```

```
      debugShowCheckedModeBanner: false,
      theme: ThemeData(
        colorScheme: ColorScheme.fromSeed(seedColor: Colors.deepPurple),
        cardTheme: CardTheme(color: Colors.blue.shade50),
        useMaterial3: true,
      ),
      home: Scaffold(
        body: LayoutBuilder(builder: (context, constraints) {
          return SingleChildScrollView(
            child: ConstrainedBox(
              constraints: BoxConstraints(minHeight: constraints.maxHeight),
              child: Column(
                mainAxisAlignment: MainAxisAlignment.spaceBetween,
                crossAxisAlignment: CrossAxisAlignment.stretch,
                children: List.generate(
                    items, (index) => ItemWidget(text: 'Item $index')),
              ),
            ),
          );
        }),
      ),
    );
  }
}

class ItemWidget extends StatelessWidget {
  const ItemWidget({
    super.key,
    required this.text,
  });

  final String text;

  @override
  Widget build(BuildContext context) {
    return Card(
      child: SizedBox(
        height: 100,
        child: Center(child: Text(text)),
      ),
    );
  }
}
```

Output:

Work with long lists:

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

void main() {
  runApp(
    MyApp(
      items: List<String>.generate(10000, (i) => 'Item $i'),
    ),
  );
}

class MyApp extends StatelessWidget {
  final List<String> items;

  const MyApp({super.key, required this.items});

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

```
    const title = 'Long List';

    return MaterialApp(
      title: title,
      home: Scaffold(
        appBar: AppBar(
          title: const Text(title),
        ),
        body: ListView.builder(
          itemCount: items.length,
          prototypeItem: ListTile(
            title: Text(items.first),
          ),
          itemBuilder: (context, index) {
            return ListTile(
              title: Text(items[index]),
            );
          },
        ),
      ),
    );
  }
}
```

Output:

Theme based styling:

To share colors and font styles throughout an app, use themes.

You can define app-wide themes. You can extend a theme to change a theme style for one component. Each theme defines the colors, type style, and other parameters applicable for the type of Material component.

Flutter applies styling in the following order:

1. Styles applied to the specific widget.
2. Themes that override the immediate parent theme.
3. Main theme for the entire app.

After you define a `Theme`, use it within your own widgets. Flutter's Material widgets use your theme to set the background colors and font styles for app bars, buttons, checkboxes, and more.

This code includes the theme settings and applies them to the **`Container`** widget in the **`MyHomePage`**. The **`GoogleFonts`** package is used to customize text themes. Make sure you have the **`google_fonts`** package added in your **`pubspec.yaml`**:

dependencies:

flutter:

sdk: flutter

google_fonts: ^4.0.0

Example:

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

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

ThemeData myTheme = ThemeData(
  useMaterial3: true,
  textTheme: TextTheme(
    bodyLarge: TextStyle(
      fontSize: 16.0,
    ),
    displayLarge: TextStyle(
      fontSize: 72,
      fontWeight: FontWeight.bold,
    ),
    titleLarge: GoogleFonts.oswald(
      fontSize: 30,
      fontStyle: FontStyle.italic,
    ),
    bodyMedium: GoogleFonts.merriweather(),
    displaySmall: GoogleFonts.pacifico(),
  ),
  colorScheme: ColorScheme.fromSeed(
```

```
    //primary: Colors.teal,
    //secondary: Colors.amber,
    seedColor: const Color(0xFF33CC99),
    //brightness: Brightness.dark,
  ),
);

// create another ThemeData

ThemeData myTheme1 = ThemeData(
  useMaterial3: true,
  textTheme: TextTheme(
    bodyLarge: TextStyle(
      fontSize: 80.0,
    ),
    displayLarge: TextStyle(
      fontSize: 40,
      fontStyle: FontStyle.italic,
      fontWeight: FontWeight.bold,
    ),
    titleLarge: GoogleFonts.oswald(
      fontSize: 30,
      //fontStyle: FontStyle.italic,
    ),
    bodyMedium: GoogleFonts.merriweather(),
    displaySmall: GoogleFonts.pacifico(),
  ),
  colorScheme:
      ColorScheme.fromSeed(seedColor: Color.fromARGB(255, 249, 15, 15)),
);

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

  @override
  Widget build(BuildContext context) {
    const appName = 'Custom Themes';

    return MaterialApp(
      title: appName,
      theme: myTheme,
      home: const MyHomePage(
        title: appName,
      ),
    );
  }
}
```

```
class MyHomePage extends StatelessWidget {
  final String title;

  const MyHomePage({super.key, required this.title});

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: Text(title,
            style: Theme.of(context).textTheme.titleLarge!.copyWith(
                  color: Theme.of(context).colorScheme.onPrimary,
                )),
        backgroundColor: Theme.of(context).colorScheme.primary,
      ),
      body: Column(mainAxisAlignment: MainAxisAlignment.center, children: [
        Center(
          child: Container(
            padding: const EdgeInsets.symmetric(
              horizontal: 12,
              vertical: 12,
            ),
            color: Theme.of(context).colorScheme.secondary,
            child: Text(
              'Text with a background color',
              // TRY THIS: change the Theme.of(context).textTheme
              //           to "displayLarge" or "displaySmall".
              style: Theme.of(context).textTheme.bodyMedium!.copyWith(
                    color: Theme.of(context).colorScheme.onPrimary,
                  ),
            ),
          ),
        ),
        Text("Message 1",
            style: TextStyle(
                fontSize:
Theme.of(context).textTheme.displayLarge!.fontSize)),
      Text("Message 2", style: Theme.of(context).textTheme.displayLarge),
        Text("Message 3", style: Theme.of(context).textTheme.titleLarge),
        Text(
          "Message 4",
          style: Theme.of(context).textTheme.displaySmall!.copyWith(
                color: Theme.of(context).colorScheme.primary,
              ),
        ),
        Theme(data: myTheme1, child: MyCustomWidget())
      ]),
      floatingActionButton: Theme(
```

```
        data: Theme.of(context).copyWith(
          // TRY THIS: Change the seedColor to "Colors.red" or
          //           "Colors.blue".
          colorScheme: ColorScheme.fromSeed(
            seedColor: Colors.pink,
            brightness: Brightness.dark,
          ),
        ),
        child: FloatingActionButton(
          onPressed: () {},
          child: const Icon(Icons.add),
        ),
      ),
    );
  }
}

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

  @override
  Widget build(BuildContext context) {
    return Text(
      "Message 5",
      style: Theme.of(context).textTheme.displayLarge!.copyWith(
            color: Theme.of(context).colorScheme.primary,
          ),
    );
  }
}
```

How we can apply the theme:

3. Applying the theme:

There are two main ways to apply your theme to your Flutter app:

- **MaterialApp theme:** You can set the `theme` property on your `MaterialApp` widget to apply the theme to your entire app.

Dart

```
MaterialApp(
  theme: myTheme,
  home: MyHomePage(),
);
```

Use code with caution.

- **Theme widget:** You can wrap a specific widget subtree with a `Theme` widget to apply the theme only to that part of your app.

Dart

```
Theme(
  data: myTheme,
  child: MyCustomWidget(),
);
```

Use code with caution.

Output:

Let's break down these lines:

Code Breakdown:

```dart
color: Theme.of(context).colorScheme.primary,
child: Text(
  'Text with a background color',
  style: Theme.of(context).textTheme.bodyMedium!.copyWith(
        color: Theme.of(context).colorScheme.onPrimary,
      ),
),
```

```

1. **`color: Theme.of(context).colorScheme.primary,`**:

   - **`Theme.of(context)`**: This fetches the current `ThemeData` from the closest `Theme` ancestor in the widget tree. The `ThemeData` contains all the styling information defined in the app's theme.

   - **`colorScheme.primary`**: Within the `ThemeData`, `colorScheme` is an instance of `ColorScheme` which holds a set of 13 colors that can be used in an app. `primary` is one of these colors, typically used for key UI elements such as the app bar or buttons.

   - **Result**: This line sets the background color of the `Container` to the primary color defined in the theme.

2. **`child: Text('Text with a background color',`**:

   - This line creates a `Text` widget as a child of the `Container`, displaying the string 'Text with a background color'.

3. **`style: Theme.of(context).textTheme.bodyMedium!.copyWith(color: Theme.of(context).colorScheme.onPrimary)`**:

   - **`Theme.of(context).textTheme.bodyMedium!`**: This fetches the `bodyMedium` text style from the current theme's `TextTheme`. The `bodyMedium` style is typically used for medium-sized body text.

   - **`.copyWith(color: Theme.of(context).colorScheme.onPrimary)`**: The `copyWith` method is used to create a copy of the `bodyMedium` text style with some modifications. Here, it modifies the text color.

     - **`color: Theme.of(context).colorScheme.onPrimary`**: This sets the text color to `onPrimary`, a color from the `ColorScheme` that is meant to be used on top of the `primary` color (to ensure good contrast and readability).

   - **Result**: The text inside the `Text` widget will use the `bodyMedium` text style, but with its color set to `onPrimary`.

### Summary:

- The `Container`'s background color is set to the primary color from the theme.

- The `Text` widget inside the `Container` displays text using the `bodyMedium` style from the theme, with its color set to `onPrimary` to ensure it contrasts well against the primary background color.
```

Add interactivity to our app

Please visit URL to understand working of these controls : https://docs.flutter.dev/ui/interactivity

Material Components

- Checkbox
- DropdownButton
- TextButton
- FloatingActionButton
- IconButton
- Radio
- ElevatedButton
- Slider
- Switch
- TextField

Drag and Drop

https://docs.flutter.dev/cookbook/effects/drag-a-widget

Input & Forms

Create and style a text field:

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

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

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

  @override
  Widget build(BuildContext context) {
    return MaterialApp(
        home: Scaffold(
      body: SafeArea(
          child: TextField(
        decoration: InputDecoration(
          border: OutlineInputBorder(),
          hintText: 'Enter a search term',
        ),
      )),
```

```
    ));
  }
}
```

Output:

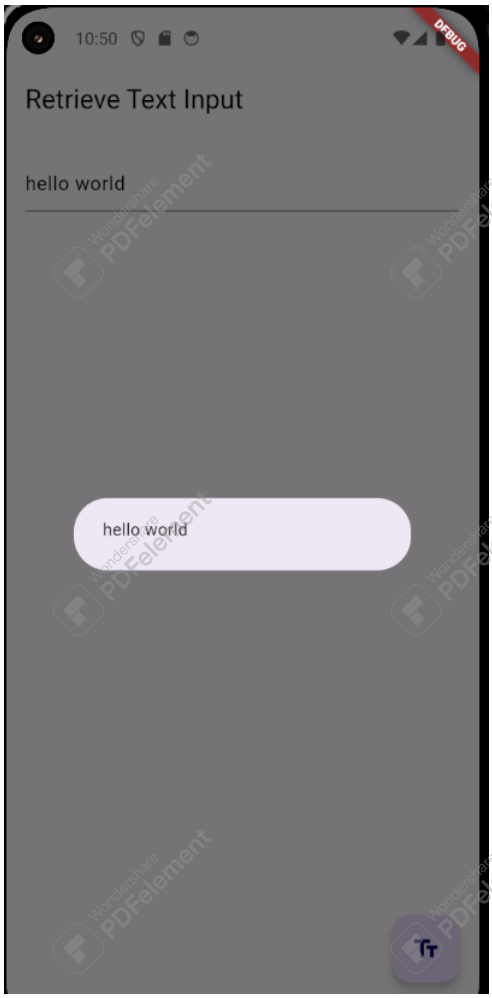
Retrieve the value of a text field

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

void main() => runApp(const MyApp());

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

  @override
  Widget build(BuildContext context) {
    return const MaterialApp(
      title: 'Retrieve Text Input',
      home: MyCustomForm(),
    );
```

```dart
  }
}

// Define a custom Form widget.
class MyCustomForm extends StatefulWidget {
  const MyCustomForm({super.key});

  @override
  State<MyCustomForm> createState() => _MyCustomFormState();
}

// Define a corresponding State class.
// This class holds the data related to the Form.
class _MyCustomFormState extends State<MyCustomForm> {
  // Create a text controller and use it to retrieve the current value
  // of the TextField.
  final myController = TextEditingController();

  @override
  void dispose() {
    // Clean up the controller when the widget is disposed.
    myController.dispose();
    super.dispose();
  }

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: const Text('Retrieve Text Input'),
      ),
      body: Padding(
        padding: const EdgeInsets.all(16),
        child: TextField(
          controller: myController,
        ),
      ),
      floatingActionButton: FloatingActionButton(
        // When the user presses the button, show an alert dialog containing
        // the text that the user has entered into the text field.
        onPressed: () {
          showDialog(
            context: context,
            builder: (context) {
              return AlertDialog(
                // Retrieve the text the that user has entered by using the
                // TextEditingController.
                content: Text(myController.text),
```

```
            );
          },
        );
      },
      tooltip: 'Show me the value!',
      child: const Icon(Icons.text_fields),
    ),
  );
 }
}
```

Output:



Handle changes to a text field

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

void main() => runApp(const MyApp());

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

  @override
  Widget build(BuildContext context) {
    return const MaterialApp(
      title: 'Retrieve Text Input',
      home: MyCustomForm(),
    );
  }
}

// Define a custom Form widget.
class MyCustomForm extends StatefulWidget {
  const MyCustomForm({super.key});

  @override
  State<MyCustomForm> createState() => _MyCustomFormState();
}

// Define a corresponding State class.
// This class holds data related to the Form.
class _MyCustomFormState extends State<MyCustomForm> {
  // Create a text controller and use it to retrieve the current value
  // of the TextField.
  final myController = TextEditingController();

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

    // Start listening to changes.
    myController.addListener(_printLatestValue);
  }

  @override
  void dispose() {
    // Clean up the controller when the widget is removed from the widget
tree.
    // This also removes the _printLatestValue listener.
    myController.dispose();
```

```
    super.dispose();
  }

  void _printLatestValue() {
    final text = myController.text;
    print('Second text field: $text (${text.characters.length})');
  }

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: const Text('Retrieve Text Input'),
      ),
      body: Padding(
        padding: const EdgeInsets.all(16),
        child: Column(
          children: [
            TextField(
              onChanged: (text) {
                debugPrint('FIRST: First text field: $text
(${text.characters.length})');
              },
            ),
            TextField(
              controller: myController,
            ),
          ],
        ),
      ),
    );
  }
}
```

Manage focus in text fields

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

void main() => runApp(const MyApp());

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

  @override
  Widget build(BuildContext context) {
    return const MaterialApp(
      title: 'Text Field Focus',
      home: MyCustomForm(),
```

```
    );
  }
}

// Define a custom Form widget.
class MyCustomForm extends StatefulWidget {
  const MyCustomForm({super.key});

  @override
  State<MyCustomForm> createState() => _MyCustomFormState();
}

// Define a corresponding State class.
// This class holds data related to the form.
class _MyCustomFormState extends State<MyCustomForm> {
  // Define the focus node. To manage the lifecycle, create the FocusNode in
  // the initState method, and clean it up in the dispose method.
  late FocusNode myFocusNode;

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

    myFocusNode = FocusNode();
  }

  @override
  void dispose() {
    // Clean up the focus node when the Form is disposed.
    myFocusNode.dispose();

    super.dispose();
  }

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: const Text('Text Field Focus'),
      ),
      body: Padding(
        padding: const EdgeInsets.all(16),
        child: Column(
          children: [
            // The first text field is focused on as soon as the app starts.
            const TextField(
              autofocus: true,
            ),
```

```
            // The second text field is focused on when a user taps the
            // FloatingActionButton.
            TextField(
              focusNode: myFocusNode,
            ),
          ],
        ),
      ),
      floatingActionButton: FloatingActionButton(
        // When the button is pressed,
        // give focus to the text field using myFocusNode.
        onPressed: () => myFocusNode.requestFocus(),
        tooltip: 'Focus Second Text Field',
        child: const Icon(Icons.edit),
      ), // This trailing comma makes auto-formatting nicer for build methods.
    );
  }
}
```

Build a form with validation

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

void main() => runApp(const MyApp());

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

  @override
  Widget build(BuildContext context) {
    const appTitle = 'Form Validation Demo';

    return MaterialApp(
      title: appTitle,
      home: Scaffold(
        appBar: AppBar(
          title: const Text(appTitle),
        ),
        body: const MyCustomForm(),
      ),
    );
  }
}

// Create a Form widget.
class MyCustomForm extends StatefulWidget {
  const MyCustomForm({super.key});
```

```
  @override
  MyCustomFormState createState() {
    return MyCustomFormState();
  }
}

// Create a corresponding State class.
// This class holds data related to the form.
class MyCustomFormState extends State<MyCustomForm> {
  // Create a global key that uniquely identifies the Form widget
  // and allows validation of the form.
  //
  // Note: This is a GlobalKey<FormState>,
  // not a GlobalKey<MyCustomFormState>.
  final _formKey = GlobalKey<FormState>();

  @override
  Widget build(BuildContext context) {
    // Build a Form widget using the _formKey created above.
    return Form(
      key: _formKey,
      child: Column(
        crossAxisAlignment: CrossAxisAlignment.start,
        children: [
          TextFormField(
            // The validator receives the text that the user has entered.
            validator: (value) {
              if (value == null || value.isEmpty) {
                return 'Please enter some text';
              }
              return null;
            },
          ),
          Padding(
            padding: const EdgeInsets.symmetric(vertical: 16),
            child: ElevatedButton(
              onPressed: () {
                // Validate returns true if the form is valid, or false
otherwise.
                if (_formKey.currentState!.validate()) {
                  // If the form is valid, display a snackbar. In the real
world,
                  // you'd often call a server or save the information in a
database.
                  ScaffoldMessenger.of(context).showSnackBar(
                    const SnackBar(content: Text('Processing Data')),
                  );
                }
```

```
            },
            child: const Text('Submit'),
          ),
        ),
      ],
    ),
  );
 }
}
```

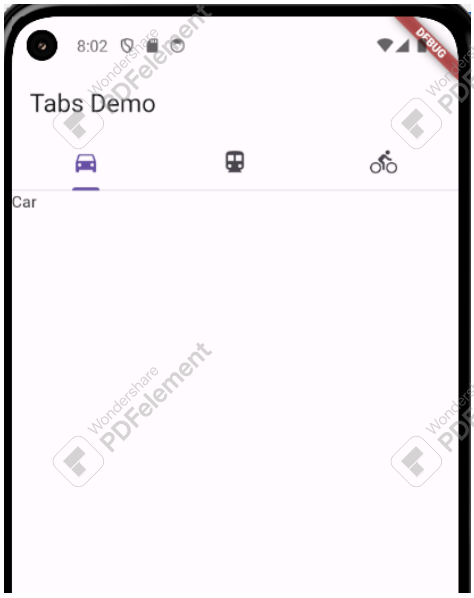
Navigation and Routing

Working with Tabs:

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

void main() {
  runApp(const TabBarDemo());
}

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

  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      home: DefaultTabController(
        length: 3,
        child: Scaffold(
          appBar: AppBar(
            bottom: const TabBar(
              tabs: [
                Tab(icon: Icon(Icons.directions_car)),
                Tab(icon: Icon(Icons.directions_transit)),
                Tab(icon: Icon(Icons.directions_bike)),
              ],
            ),
            title: const Text('Tabs Demo'),
          ),
          body: const TabBarView(
            children: [
              Text("Car"),
              Text("Bus"),
              Text("Cycle")
            ],
          ),
```

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

Output:

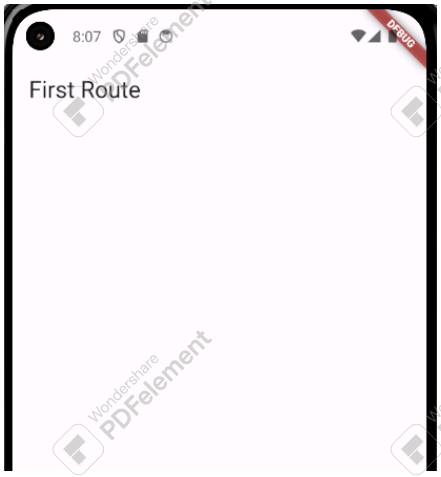


Navigate to new screen and back:

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

void main() {
  runApp(const MaterialApp(
    title: 'Navigation Basics',
    home: FirstRoute(),
  ));
}

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

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: const Text('First Route'),
      ),
      body: Center(
        child: ElevatedButton(
          child: const Text('Open route'),
```

```
        onPressed: () {
          Navigator.push(
            context,
            MaterialPageRoute(builder: (context) => const SecondRoute()),
          );
        },
      ),
    ),
  );
 }
}

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

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: const Text('Second Route'),
      ),
      body: Center(
        child: ElevatedButton(
          onPressed: () {
            Navigator.pop(context);
          },
          child: const Text('Go back!'),
        ),
      ),
    );
  }
}
```

Output:



Navigation with CupertinoPageRoute

In Flutter you are not limited to Material design language, instead, you also have access to Cupertino (iOS-style) widgets

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

void main() {
  runApp(const CupertinoApp(
    title: 'Navigation Basics',
    home: FirstRoute(),
  ));
}

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

  @override
  Widget build(BuildContext context) {
    return CupertinoPageScaffold(
      navigationBar: const CupertinoNavigationBar(
        middle: Text('First Route'),
      ),
      child: Center(
        child: CupertinoButton(
          child: const Text('Open route'),
          onPressed: () {
            Navigator.push(
              context,
              CupertinoPageRoute(builder: (context) => const SecondRoute()),
            );
          },
        ),
      ),
    );
  }
}

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

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

```
      navigationBar: const CupertinoNavigationBar(
        middle: Text('Second Route'),
      ),
      child: Center(
        child: CupertinoButton(
          onPressed: () {
            Navigator.pop(context);
          },
          child: const Text('Go back!'),
        ),
      ),
    );
  }
}
```

Output:

Send a string to a new screen

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

```
void main() {
  runApp(
    const MaterialApp(title: 'Passing Data', home: DataSender()),
  );
}

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

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: const Text('Send Data'),
      ),
      body: Center(
        child: ElevatedButton(
          onPressed: () {
            String data =
                "This is some data!"; // Replace with your desired string
            Navigator.push(
              context,
              MaterialPageRoute(
                builder: (context) => DataReceiver(data: data),
              ),
            );
          },
          child: const Text('Send'),
        ),
      ),
    );
  }
}

class DataReceiver extends StatelessWidget {
  final String data;

  const DataReceiver({super.key, required this.data});

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: const Text('Received Data'),
      ),
      body: Center(
        child: Text(
          'You sent: $data',
```

```
        style: const TextStyle(fontSize: 20.0),
      ),
    ),
  );
 }
}
```

Send a string to new screen using RouteSettings

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

void main() {
  runApp(
     const MaterialApp(title: 'Passing Data', home: DataSender()),
  );
}

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

  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      home: Scaffold(
        appBar: AppBar(
          title: const Text('Send Data'),
        ),
        body: Center(
          child: Column(
            mainAxisAlignment: MainAxisAlignment.center,
            children: [
              ElevatedButton(
                onPressed: () {
                  String data = "This is some data!"; // Replace with your
desired string
                  Navigator.push(
                    context,
                    MaterialPageRoute(
                      builder: (context) => const DataReceiver(),
                      settings: RouteSettings(arguments: data),
                    ),
                  );
                },
                child: const Text('Send'),
              ),
            ],
```

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



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

  @override
  Widget build(BuildContext context) {
    final args = ModalRoute.of(context)!.settings.arguments as String;

    return Scaffold(
      appBar: AppBar(
        title: const Text('Received Data'),
      ),
      body: Center(
        child: Text(
          'You sent: $args',
          style: const TextStyle(fontSize: 20.0),
        ),
      ),
    );
  }
}
```

Send data shown in a list to a new screen

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

void main() {
  runApp(
    MaterialApp(
      title: 'Passing Data',
      home: TodosScreen(
        todos: List.generate(
          20,
          (i) => Todo(
            'Todo $i',
```

```
          'A description of what needs to be done for Todo $i',
        ),
      ),
    ),
  ),
  );
}

class Todo {
  final String title;
  final String description;

  const Todo(this.title, this.description);
}

class TodosScreen extends StatelessWidget {
  const TodosScreen({super.key, required this.todos});

  final List<Todo> todos;

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: const Text('Todos'),
      ),
      body: ListView.builder(
        itemCount: todos.length,
        itemBuilder: (context, index) {
          return ListTile(
            title: Text(todos[index].title),
            // When a user taps the ListTile, navigate to the DetailScreen.
            // Notice that you're not only creating a DetailScreen, you're
            // also passing the current todo through to it.
            onTap: () {
              Navigator.push(
                context,
                MaterialPageRoute(
                  builder: (context) => DetailScreen(todo: todos[index]),
                ),
              );
            },
          );
        },
      ),
    );
  }
}
```

```dart
class DetailScreen extends StatelessWidget {
  // In the constructor, require a Todo.
  const DetailScreen({super.key, required this.todo});

  // Declare a field that holds the Todo.
  final Todo todo;

  @override
  Widget build(BuildContext context) {
    // Use the Todo to create the UI.
    return Scaffold(
      appBar: AppBar(
        title: Text(todo.title),
      ),
      body: Padding(
        padding: const EdgeInsets.all(16),
        child: Text(todo.description),
      ),
    );
  }
}
```

Send data shown in a list to a new screen using RouteSettings

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

class Todo {
  final String title;
  final String description;

  const Todo(this.title, this.description);
}

void main() {
  runApp(
    MaterialApp(
      title: 'Passing Data',
      home: TodosScreen(
        todos: List.generate(
          20,
          (i) => Todo(
            'Todo $i',
            'A description of what needs to be done for Todo $i',
          ),
        ),
      ),
    ),
  );
}

class TodosScreen extends StatelessWidget {
  const TodosScreen({super.key, required this.todos});

  final List<Todo> todos;

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: const Text('Todos'),
      ),
      body: ListView.builder(
        itemCount: todos.length,
        itemBuilder: (context, index) {
          return ListTile(
            title: Text(todos[index].title),
            // When a user taps the ListTile, navigate to the DetailScreen.
            // Notice that you're not only creating a DetailScreen, you're
            // also passing the current todo through to it.
```

```
            onTap: () {
              Navigator.push(
                context,
                MaterialPageRoute(
                  builder: (context) => const DetailScreen(),
                  // Pass the arguments as part of the RouteSettings. The
                  // DetailScreen reads the arguments from these settings.
                  settings: RouteSettings(
                    arguments: todos[index],
                  ),
                ),
              );
            },
          );
        },
      ),
    );
  }
}

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

  @override
  Widget build(BuildContext context) {
    final todo = ModalRoute.of(context)!.settings.arguments as Todo;

    // Use the Todo to create the UI.
    return Scaffold(
      appBar: AppBar(
        title: Text(todo.title),
      ),
      body: Padding(
        padding: const EdgeInsets.all(16),
        child: Text(todo.description),
      ),
    );
  }
}
```

Return data from a screen

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

```dart
void main() {
  runApp(
    const MaterialApp(
      title: 'Returning Data',
      home: HomeScreen(),
    ),
  );
}

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

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: const Text('Returning Data Demo'),
      ),
      body: const Center(
        child: SelectionButton(),
      ),
    );
  }
}

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

  @override
  State<SelectionButton> createState() => _SelectionButtonState();
}

class _SelectionButtonState extends State<SelectionButton> {
  @override
  Widget build(BuildContext context) {
    return ElevatedButton(
      onPressed: () {
        _navigateAndDisplaySelection(context);
      },
      child: const Text('Pick an option, any option!'),
    );
  }

  // A method that launches the SelectionScreen and awaits the result from
  // Navigator.pop.
  Future<void> _navigateAndDisplaySelection(BuildContext context) async {
    // Navigator.push returns a Future that completes after calling
```

```
    // Navigator.pop on the Selection Screen.
    final result = await Navigator.push(
      context,
      MaterialPageRoute(builder: (context) => const SelectionScreen()),
    );

    // When a BuildContext is used from a StatefulWidget, the mounted property
    // must be checked after an asynchronous gap.
    if (!context.mounted) return;

    // After the Selection Screen returns a result, hide any previous
snackbars
    // and show the new result.
    ScaffoldMessenger.of(context)
      ..removeCurrentSnackBar()
      ..showSnackBar(SnackBar(content: Text('$result')));
  }
}

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

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: const Text('Pick an option'),
      ),
      body: Center(
        child: Column(
          mainAxisAlignment: MainAxisAlignment.center,
          children: <Widget>[
            Padding(
              padding: const EdgeInsets.all(8),
              child: ElevatedButton(
                onPressed: () {
                  // Close the screen and return "Yep!" as the result.
                  Navigator.pop(context, 'Yep!');
                },
                child: const Text('Yep!'),
              ),
            ),
            Padding(
              padding: const EdgeInsets.all(8),
              child: ElevatedButton(
                onPressed: () {
                  // Close the screen and return "Nope." as the result.
                  Navigator.pop(context, 'Nope.');
```

```
                },
                child: const Text('Nope.'),
              ),
            )
          ],
        ),
      ),
    );
  }
}
```

Add a drawer to a screen

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

void main() => runApp(const MyApp());

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

  static const appTitle = 'Drawer Demo';

  @override
  Widget build(BuildContext context) {
    return const MaterialApp(
      title: appTitle,
      home: MyHomePage(title: appTitle),
    );
  }
}

class MyHomePage extends StatefulWidget {
  const MyHomePage({super.key, required this.title});

  final String title;

  @override
  State<MyHomePage> createState() => _MyHomePageState();
}

class _MyHomePageState extends State<MyHomePage> {
  int _selectedIndex = 0;
  static const TextStyle optionStyle =
      TextStyle(fontSize: 30, fontWeight: FontWeight.bold);
  static const List<Widget> _widgetOptions = <Widget>[
```

```
  Text(
    'Index 0: Home',
    style: optionStyle,
  ),
  Text(
    'Index 1: Business',
    style: optionStyle,
  ),
  Text(
    'Index 2: School',
    style: optionStyle,
  ),
];

void _onItemTapped(int index) {
  setState(() {
    _selectedIndex = index;
  });
}

@override
Widget build(BuildContext context) {
  return Scaffold(
    appBar: AppBar(
      title: Text(widget.title),
      leading: Builder(
        builder: (context) {
          return IconButton(
            icon: const Icon(Icons.menu),
            onPressed: () {
              Scaffold.of(context).openDrawer();
            },
          );
        },
      ),
    ),
    body: Center(
      child: _widgetOptions[_selectedIndex],
    ),
    drawer: Drawer(
      // Add a ListView to the drawer. This ensures the user can scroll
      // through the options in the drawer if there isn't enough vertical
      // space to fit everything.
      child: ListView(
        // Important: Remove any padding from the ListView.
        padding: EdgeInsets.zero,
        children: [
          const DrawerHeader(
```

```
            decoration: BoxDecoration(
              color: Colors.blue,
            ),
            child: Text('Drawer Header'),
          ),
          ListTile(
            title: const Text('Home'),
            selected: _selectedIndex == 0,
            onTap: () {
              // Update the state of the app
              _onItemTapped(0);
              // Then close the drawer
              Navigator.pop(context);
            },
          ),
          ListTile(
            title: const Text('Business'),
            selected: _selectedIndex == 1,
            onTap: () {
              // Update the state of the app
              _onItemTapped(1);
              // Then close the drawer
              Navigator.pop(context);
            },
          ),
          ListTile(
            title: const Text('School'),
            selected: _selectedIndex == 2,
            onTap: () {
              // Update the state of the app
              _onItemTapped(2);
              // Then close the drawer
              Navigator.pop(context);
            },
          ),
        ],
      ),
    );
  }
}
```

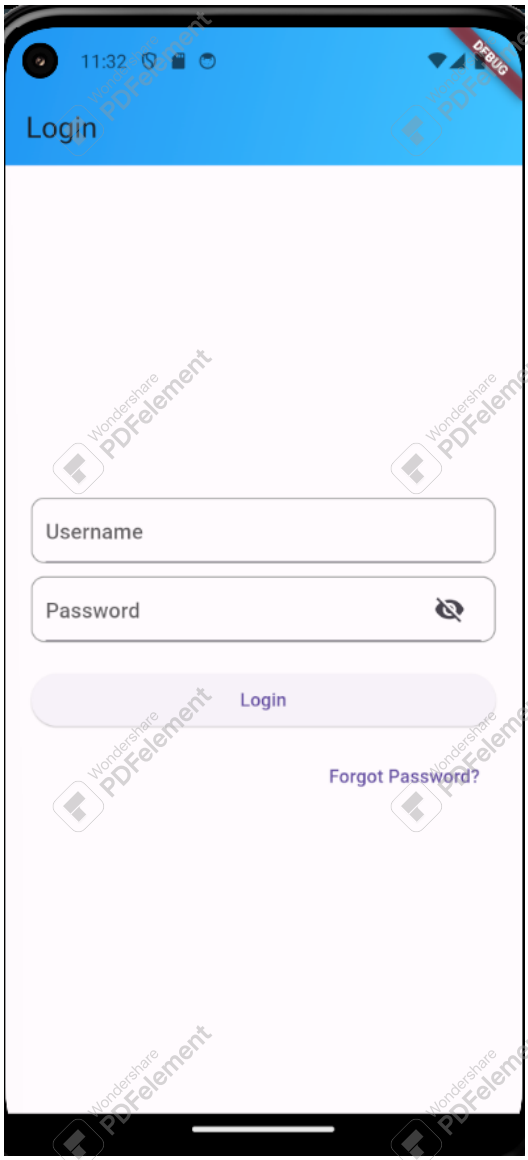
Create a login page in flutter

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

void main() => runApp( MaterialApp(home: LoginPage()));

class LoginPage extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: Text('Login'),
        flexibleSpace: Container(
          decoration: BoxDecoration(
            gradient: LinearGradient(
              colors: [Colors.blue, Colors.lightBlueAccent],
              begin: Alignment.topLeft,
              end: Alignment.bottomRight,
            ),
          ),
        ),
      ),
      body: Center(
        child: Container(
          padding: EdgeInsets.all(20.0),
          child: Column(
            mainAxisAlignment: MainAxisAlignment.center,
            children: <Widget>[
              // App Logo (Optional)
              // Image.asset('path/to/your/logo.png'),
              SizedBox(height: 20.0),
              // Username Field
              Container(
                decoration: BoxDecoration(
                  border: Border.all(color: Colors.grey),
                  borderRadius: BorderRadius.circular(10.0),
                ),
                child: Padding(
                  padding: EdgeInsets.symmetric(horizontal: 10.0),
                  child: TextField(
                    decoration: InputDecoration(
                      hintText: 'Username',
                    ),
                  ),
                ),
              ),
              SizedBox(height: 10.0),
```

```
            // Password Field
            Container(
              decoration: BoxDecoration(
                border: Border.all(color: Colors.grey),
                borderRadius: BorderRadius.circular(10.0),
              ),
              child: Padding(
                padding: EdgeInsets.symmetric(horizontal: 10.0),
                child: TextField(
                  decoration: InputDecoration(
                    hintText: 'Password',
                    suffixIcon: Icon(Icons.visibility_off),
                  ),
                  obscureText: true,
                ),
              ),
            ),
            SizedBox(height: 20.0),
            // Login Button
            Container(
              width: double.infinity, // Match parent width
              child: ElevatedButton(
                onPressed: () {
                  // Handle Login Button press
                },
                child: Text('Login'),
              ),
            ),
            SizedBox(height: 10.0),
            // Forgot Password Text (Optional)
            Row(
              mainAxisAlignment: MainAxisAlignment.end,
              children: <Widget>[
                TextButton(
                  onPressed: () {
                    // Handle Forgot Password press
                  },
                  child: Text('Forgot Password?'),
                ),
              ],
            ),
          ],
        ),
      ),
    ),
  );
}
```

```
}
```

Output:



Connecting Flutter with Firestore

Write and read document from Firestore database

First of all create a firestore database by visiting: flutter.google.com

Add some collections and documents in the database.

See if the permissions are enabled in Firestore Database -> Rules:

```
rules_version = '2';
service cloud.firestore {
  match /databases/{database}/documents {
    match /{document=**} {
      allow read, write: if
          request.time < timestamp.date(2024, 12, 30);
    }
  }
}
```

Then under the settings, add an app, and add flutter app, and follow the steps mentioned about how to configure firestore with flutter.

Make the following changes in android/src/build.gradle

Under:

android -> defaultConfig -> change minSdkVersion to 23 and targetSdkVersion to 23

For example:

```
   defaultConfig {
        // TODO: Specify your own unique Application ID
(https://developer.android.com/studio/build/application-id.html).
        applicationId "com.example.flutterapp"
        // You can update the following values to match your application
needs.
        // For more information, see:
https://docs.flutter.dev/deployment/android#reviewing-the-gradle-build-
configuration.
        //minSdkVersion flutter.minSdkVersion
        //targetSdkVersion flutter.targetSdkVersion
        minSdkVersion 23
        targetSdkVersion 23
        versionCode flutterVersionCode.toInteger()
        versionName flutterVersionName
    }
```

At the end of the file, paste this code (this code is required to remove, if any, Kotlin errors on compilation):

```
configurations.implementation {
exclude group: 'org.jetbrains.kotlin', module: 'kotlin-stdlib-jdk8'
}
```

Run these commands on project root directory:

```
flutter pub add cloud_firestore
flutter run
```

lib/firebase_options.dart

(This file will not work on your system, you need to create your own file for your firestore database, by going to settings, and adding a flutter app)

```
// File generated by FlutterFire CLI.

// ignore_for_file: type=lint

import 'package:firebase_core/firebase_core.dart' show FirebaseOptions;

import 'package:flutter/foundation.dart'
    show defaultTargetPlatform, kIsWeb, TargetPlatform;
```

```dart
/// Default [FirebaseOptions] for use with your Firebase apps.
///
/// Example:
/// ```dart
/// import 'firebase_options.dart';
/// // ...
/// await Firebase.initializeApp(
///   options: DefaultFirebaseOptions.currentPlatform,
/// );
/// ```
class DefaultFirebaseOptions {
  static FirebaseOptions get currentPlatform {
    if (kIsWeb) {
      return web;
    }

    switch (defaultTargetPlatform) {
      case TargetPlatform.android:
        return android;

      case TargetPlatform.iOS:
        return ios;

      case TargetPlatform.macOS:
        return macos;

      case TargetPlatform.windows:
        return windows;

      case TargetPlatform.linux:
        throw UnsupportedError(
          'DefaultFirebaseOptions have not been configured for linux - '
          'you can reconfigure this by running the FlutterFire CLI again.',
        );

      default:
        throw UnsupportedError(
          'DefaultFirebaseOptions are not supported for this platform.',
```

```
      );
    }
  }

  static const FirebaseOptions web = FirebaseOptions(
    apiKey: 'AIzaSyDKSJfAZLNWA328ArcCdQXwtRMiIaRm7dA',
    appId: '1:352306058995:web:6771e662eb28b9cc478986',
    messagingSenderId: '352306058995',
    projectId: 'dbproj-f3054',
    authDomain: 'dbproj-f3054.firebaseapp.com',
    storageBucket: 'dbproj-f3054.appspot.com',
  );

  static const FirebaseOptions android = FirebaseOptions(
    apiKey: 'AIzaSyC9oTf9WSDqHaeOeevpwLHxzsfyWPKLnys',
    appId: '1:352306058995:android:3abcfd249a1882c8478986',
    messagingSenderId: '352306058995',
    projectId: 'dbproj-f3054',
    storageBucket: 'dbproj-f3054.appspot.com',
  );

  static const FirebaseOptions ios = FirebaseOptions(
    apiKey: 'AIzaSyDo4Ul5FkL5f9_BBYSVJEJY_glr2SM4Fec',
    appId: '1:352306058995:ios:ebd6be1b5ca145e1478986',
    messagingSenderId: '352306058995',
    projectId: 'dbproj-f3054',
    storageBucket: 'dbproj-f3054.appspot.com',
    iosBundleId: 'com.example.flutterapp',
  );

  static const FirebaseOptions macos = FirebaseOptions(
    apiKey: 'AIzaSyDo4Ul5FkL5f9_BBYSVJEJY_glr2SM4Fec',
    appId: '1:352306058995:ios:ebd6be1b5ca145e1478986',
    messagingSenderId: '352306058995',
    projectId: 'dbproj-f3054',
    storageBucket: 'dbproj-f3054.appspot.com',
    iosBundleId: 'com.example.flutterapp',
  );

  static const FirebaseOptions windows = FirebaseOptions(
    apiKey: 'AIzaSyDKSJfAZLNWA328ArcCdQXwtRMiIaRm7dA',
    appId: '1:352306058995:web:5b68674ca0398f7b478986',
    messagingSenderId: '352306058995',
    projectId: 'dbproj-f3054',
    authDomain: 'dbproj-f3054.firebaseapp.com',
    storageBucket: 'dbproj-f3054.appspot.com',
  );
}
```

In **pubspec.yaml** file, add the following dependencies for firebase

```
dependencies:
  flutter:
    sdk: flutter


  # The following adds the Cupertino Icons font to your application.
  # Use with the CupertinoIcons class for iOS style icons.
  cupertino_icons: ^1.0.6
  firebase_core: ^3.1.0
  cloud_firestore: ^5.0.1
```

Follow the documentation provided on this URL:

https://firebase.google.com/docs/firestore/manage-data/add-data

main.dart

```
import 'package:cloud_firestore/cloud_firestore.dart';
import 'package:flutter/material.dart';
import 'package:firebase_core/firebase_core.dart';
import 'firebase_options.dart';

void main() async {
  WidgetsFlutterBinding.ensureInitialized();
  await Firebase.initializeApp(
    options: DefaultFirebaseOptions.currentPlatform,
  );
  runApp(MyApp());
}

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

  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      home: Scaffold(
        appBar: AppBar(title: Text('Firestore Example')),
        body: Center(
```

```
        child: CitiesWidget(),
      ),
    ),
  );
 }
}

class CitiesWidget extends StatelessWidget {
 final FirebaseFirestore db = FirebaseFirestore.instance;

 Future<void> saveCity(String name, String country) async {
  try {
   await db.collection('cities').add({
    'name': name,
    'country': country,
   }).then((DocumentReference doc) =>
   debugPrint('City saved successfully with ID: ${doc.id}'));
  } catch (e) {
   debugPrint('Failed to save city: $e');
  }
 }

 Future<void> retrieveCity(String documentId) async {
  try {
   DocumentSnapshot documentSnapshot = await
db.collection('cities').doc(documentId).get();
   if (documentSnapshot.exists) {
    Map<String, dynamic> data = documentSnapshot.data() as Map<String,
dynamic>;
    debugPrint('City: ${data['name']}, Country: ${data['country']}');
   } else {
    debugPrint('Document does not exist.');
   }
  } catch (e) {
   debugPrint('Failed to retrieve city: $e');
  }
 }

 @override
 Widget build(BuildContext context) {
  return Column(
   mainAxisAlignment: MainAxisAlignment.center,
   children: [
    ElevatedButton(
     onPressed: () {
      saveCity('Peshawar', 'Pakistan');
     },
     child: Text('Save City'),
```

```
        ),
        ElevatedButton(
          onPressed: () {
            retrieveCity('BJ');
          },
          child: Text('Retrieve City'),
        ),
      ],
    );
  }
}
```

Custom objects

Using Map or Dictionary objects to represent your documents is often not very convenient, so Cloud Firestore supports writing documents with custom classes. Cloud Firestore converts the objects to supported data types.

```
import 'package:cloud_firestore/cloud_firestore.dart';
import 'package:flutter/material.dart';
import 'package:firebase_core/firebase_core.dart';
import 'firebase_options.dart';

class City {
  final String? name;
  final String? state;
  final String? country;
  final bool? capital;
  final int? population;
  final List<String>? regions;

  City({
    this.name,
    this.state,
    this.country,
    this.capital,
    this.population,
    this.regions,
  });

  factory City.fromFirestore(
    DocumentSnapshot<Map<String, dynamic>> snapshot,
    SnapshotOptions? options,
  ) {
    final data = snapshot.data();
```

```
    return City(
      name: data?['name'],
      state: data?['state'],
      country: data?['country'],
      capital: data?['capital'],
      population: data?['population'],
      regions:
          data?['regions'] is Iterable ? List.from(data?['regions']) : null,
    );
  }

  Map<String, dynamic> toFirestore() {
    return {
      if (name != null) "name": name,
      if (state != null) "state": state,
      if (country != null) "country": country,
      if (capital != null) "capital": capital,
      if (population != null) "population": population,
      if (regions != null) "regions": regions,
    };
  }
}

void main() async {
  WidgetsFlutterBinding.ensureInitialized();
  await Firebase.initializeApp(
    options: DefaultFirebaseOptions.currentPlatform,
  );
  runApp(MyApp());
}

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

  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      home: Scaffold(
        appBar: AppBar(title: Text('Firestore Example')),
        body: Center(
          child: CitiesWidget(),
        ),
      ),
    );
  }
}

class CitiesWidget extends StatelessWidget {
```

```
CitiesWidget({super.key});

final FirebaseFirestore db = FirebaseFirestore.instance;

Future<void> saveCity() async {
  final city = City(
    name: "Los Angeles",
    state: "CA",
    country: "USA",
    capital: false,
    population: 5000000,
    regions: ["west_coast", "socal"],
  );

  try {
    final docRef = db
        .collection("cities")
        .withConverter(
          fromFirestore: City.fromFirestore,
          toFirestore: (City city, options) => city.toFirestore(),
        )
        .doc("LA");
    await docRef.set(city);

    debugPrint('City saved successfully with ID: $city');
  } catch (e) {
    debugPrint('Failed to save city: $e');
  }
}

Future<void> retrieveCity(String documentId) async {
  try {
    final docRef = await db
        .collection("cities")
        .withConverter(
          fromFirestore: City.fromFirestore,
          toFirestore: (City city, options) => city.toFirestore(),
        )
        .doc("LA")
        .get();

    debugPrint(docRef.data()!.name);
    debugPrint(docRef.data()!.state);
    debugPrint(docRef.data()!.country);
    debugPrint("${docRef.data()!.capital}");
    debugPrint("${docRef.data()!.population}");
    debugPrint("${docRef.data()!.regions}");
  } catch (e) {
```

```
      debugPrint('Failed to retrieve city: $e');
    }
  }

  @override
  Widget build(BuildContext context) {
    return Column(
      mainAxisAlignment: MainAxisAlignment.center,
      children: [
        ElevatedButton(
          onPressed: () {
            saveCity();
          },
          child: Text('Save City'),
        ),
        ElevatedButton(
          onPressed: () {
            retrieveCity('LA');
          },
          child: Text('Retrieve City'),
        ),
      ],
    );
  }
}
```

Saving a document with the custom ID (Why “set” method is used)

Here PK_LHR is the custom ID provided to the document.

```
  Future<void> saveCity() async {


    try {
      db.collection("cities").doc("PK_LHR").set({"name": "Chicago"});

      debugPrint('City saved successfully');
    } catch (e) {
      debugPrint('Failed to save city: $e');
    }
  }
```

Saving a method with auto-generated ID (Why “add” method is used)

```
Future<void> saveCity() async {
    try {
      // Add a new document with a generated id.
      final data = {"name": "Tokyo", "country": "Japan"};

      db.collection("cities").add(data).then((documentSnapshot) =>
          debugPrint("Added Data with ID: ${documentSnapshot.id}"));
    } catch (e) {
      debugPrint('Failed to save city: $e');
    }
  }
```

In some cases, it can be useful to create a document reference with an auto-generated ID, then use the reference later. For this use case, you can call `doc()`:

Note:

Behind the scenes, `.add(...)` and `.doc().set(...)` are completely equivalent, so you can use whichever is more convenient.

Update a document

To update some fields of a document without overwriting the entire document, use the following language-specific update() methods:

```
  Future<void> saveCity() async {
    try {
      final washingtonRef = db.collection("cities").doc("DC");
      washingtonRef.update({"capital": true}).then(
          (value) => debugPrint("DocumentSnapshot successfully updated!"),
          onError: (e) => debugPrint("Error updating document $e"));
    } catch (e) {
      debugPrint('Failed to save city: $e');
    }
  }
```

Server Timestamp

You can set a field in your document to a server timestamp which tracks when the server receives the update.

```
Future<void> saveCity() async {
    try {
      final docRef = db.collection("cities").doc("DC");
      final updates = <String, dynamic>{
        "timestamp": FieldValue.serverTimestamp(),
      };
```

```
      docRef.update(updates).then(
          (value) => print("DocumentSnapshot successfully updated!"),
          onError: (e) => print("Error updating document $e"));
    } catch (e) {
      debugPrint('Failed to save city: $e');
    }
  }
```

Update fields in a nested object

```
import 'package:cloud_firestore/cloud_firestore.dart';
import 'package:flutter/material.dart';
import 'package:firebase_core/firebase_core.dart';
import 'firebase_options.dart';

void main() async {
  WidgetsFlutterBinding.ensureInitialized();
  await Firebase.initializeApp(
    options: DefaultFirebaseOptions.currentPlatform,
  );
  runApp(MyApp());
}

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

  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      home: Scaffold(
        appBar: AppBar(title: Text('Firestore Example')),
        body: Center(
          child: CitiesWidget(),
        ),
      ),
    );
  }
}

class CitiesWidget extends StatelessWidget {
  CitiesWidget({super.key});

  final FirebaseFirestore db = FirebaseFirestore.instance;

  Future<void> saveCity() async {
    try {
      // Create our initial doc
```

```
      Map<String, dynamic> record = {
  "name": "Frank",
  "favorites": {
    "food": "Pizza",
    "color": "Blue",
    "subject": "Recess"
  },
  "age": 12
};

await db.collection("users").doc("frank").set(record).then((value) =>
debugPrint("Frank created"));

       } catch (e) {
      debugPrint('Failed to save document: $e');
    }
  }

  Future<void> updateDocument(String documentId) async {
    try {
db
    .collection("users")
    .doc( documentId )
    .update({"age": 13, "favorites.color": "Red"});

    } catch (e) {
      debugPrint('Failed to retrieve city: $e');
    }
  }

  @override
  Widget build(BuildContext context) {
    return Column(
      mainAxisAlignment: MainAxisAlignment.center,
      children: [
        ElevatedButton(
          onPressed: () {
            saveCity();
          },
          child: Text('Save document'),
        ),
        ElevatedButton(
          onPressed: () {
            updateDocument('frank');
          },
          child: Text('Update document'),
        ),
      ],
```

```
    );
  }
}
```

Update elements in an array

```
  Future<void> saveCity() async {
    try {
      final washingtonRef = db.collection("cities").doc("DC");

// Atomically add a new region to the "regions" array field.
      washingtonRef.update({
        "regions": FieldValue.arrayUnion(["greater_virginia"]),
      });

// Atomically remove a region from the "regions" array field.
      washingtonRef.update({
        "regions": FieldValue.arrayRemove(["east_coast"]),
      });
    } catch (e) {
      debugPrint('Failed to save document: $e');
    }
  }
```

Increment a numeric value

```
Future<void> saveCity() async {
    try {
      var washingtonRef = db.collection('cities').doc('DC');

// Atomically increment the population of the city by 50.
      washingtonRef.update(
        {"population": FieldValue.increment(50)},
      );
    } catch (e) {
      debugPrint('Failed to save document: $e');
    }
  }
```

Delete data from Cloud Firestore

```
db.collection("cities").doc("DC").delete().then(
      (doc) => print("Document deleted"),
      onError: (e) => print("Error updating document $e"),
    );
```

Delete fields

To delete specific fields from a document, use the following language-specific FieldValue.delete() methods when you update a document:

```
   final docRef = db.collection("cities").doc("BJ");

// Remove the 'capital' field from the document
final updates = <String, dynamic>{
  "capital": FieldValue.delete(),
};

docRef.update(updates);
```

Retrieve a Custom Object from Database

```
import 'package:cloud_firestore/cloud_firestore.dart';
import 'package:flutter/material.dart';
import 'package:firebase_core/firebase_core.dart';
import 'firebase_options.dart';

void main() async {
  WidgetsFlutterBinding.ensureInitialized();
  await Firebase.initializeApp(
    options: DefaultFirebaseOptions.currentPlatform,
  );
  runApp(MyApp());
}

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

  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      home: Scaffold(
        appBar: AppBar(title: Text('Firestore Example')),
        body: Center(
          child: CitiesWidget(),
        ),
      ),
    );
  }
}
```

```dart
class City {
  final String? name;
  final String? state;
  final String? country;
  final bool? capital;
  final int? population;
  final List<String>? regions;

  City({
    this.name,
    this.state,
    this.country,
    this.capital,
    this.population,
    this.regions,
  });

  factory City.fromFirestore(
    DocumentSnapshot<Map<String, dynamic>> snapshot,
    SnapshotOptions? options,
  ) {
    final data = snapshot.data();
    return City(
      name: data?['name'],
      state: data?['state'],
      country: data?['country'],
      capital: data?['capital'],
      population: data?['population'],
      regions:
          data?['regions'] is Iterable ? List.from(data?['regions']) : null,
    );
  }

  Map<String, dynamic> toFirestore() {
    return {
      if (name != null) "name": name,
      if (state != null) "state": state,
      if (country != null) "country": country,
      if (capital != null) "capital": capital,
      if (population != null) "population": population,
      if (regions != null) "regions": regions,
    };
  }
}

class CitiesWidget extends StatelessWidget {
  CitiesWidget({super.key});
```

```dart
final FirebaseFirestore db = FirebaseFirestore.instance;

Future<void> saveDocument() async {
  try {} catch (e) {
    debugPrint('Failed to save document: $e');
  }
}

Future<void> getDocument(String documentId) async {
  try {
    final ref = db.collection("cities").doc("LA").withConverter(
          fromFirestore: City.fromFirestore,
          toFirestore: (City city, _) => city.toFirestore(),
        );
    final docSnap = await ref.get();
    final city = docSnap.data(); // Convert to City object
    if (city != null) {
      debugPrint(city.name);
      debugPrint(city.state);
      debugPrint(city.country);
      debugPrint("${city.capital}");
      debugPrint("${city.population}");
      debugPrint("${city.regions?[0]}");



    } else {
      debugPrint("No such document.");
    }
  } catch (e) {
    debugPrint('Failed to retrieve city: $e');
  }
}

@override
Widget build(BuildContext context) {
  return Column(
    mainAxisAlignment: MainAxisAlignment.center,
    children: [
      ElevatedButton(
        onPressed: () {
          saveDocument();
        },
        child: Text('Save document'),
      ),
      ElevatedButton(
        onPressed: () {
          getDocument('frank');
```

```
        },
        child: Text('Get document'),
      ),
    ],
  );
 }
}
```

Get multiple documents from a collection

You can also retrieve multiple documents with one request by querying documents in a collection. For example, you can use where() to query for all of the documents that meet a certain condition, then use get() to retrieve the results:

```
 Future<void> getMultipleDocuments() async {
  try {
    db.collection("cities").where("capital", isEqualTo: true).get().then(
      (querySnapshot) {
        debugPrint("Successfully completed");
        for (var docSnapshot in querySnapshot.docs) {
          debugPrint('${docSnapshot.id} => ${docSnapshot.data()}');
        }
      },
      onError: (e) => debugPrint("Error completing: $e"),
    );
  } catch (e) {
    debugPrint('Failed to retrieve city: $e');
  }
 }
```

Get all documents in a collection

```
 Future<void> getMultipleDocuments() async {
  try {
   db.collection("cities").get().then(
 (querySnapshot) {
  debugPrint("Successfully completed");
  for (var docSnapshot in querySnapshot.docs) {
    debugPrint('${docSnapshot.id} => ${docSnapshot.data()}');
  }
 },
 onError: (e) => debugPrint("Error completing: $e"),
);
  } catch (e) {
    debugPrint('Failed to retrieve city: $e');
  }
```

```
  }
```

Get all documents in a subcollection

```
db.collection("cities").doc("SF").collection("landmarks").get().then(
  (querySnapshot) {
    debugPrint("Successfully completed");
    for (var docSnapshot in querySnapshot.docs) {
      debugPrint('${docSnapshot.id} => ${docSnapshot.data()}');
    }
  },
  onError: (e) => print("Error completing: $e"),
);
```

Compound Query

```
  Future<void> getMultipleDocuments() async {
    try {

      db.collection("cities")
      .where("state", isEqualTo: "CA")
      .where("population", isLessThan: 1000000).get().then(
        (querySnapshot) {
          debugPrint("Successfully completed");
          for (var docSnapshot in querySnapshot.docs) {
            debugPrint('${docSnapshot.id} => ${docSnapshot.data()}');
          }
        },
        onError: (e) => debugPrint("Error completing: $e"),
      );
    } catch (e) {
      debugPrint('Failed to retrieve city: $e');
    }
  }
```

Export schema from Firestore using Node JS

Exporting a Firestore schema involves backing up the structure of your collections and documents. Here's a step-by-step guide to help you achieve this using Node.js:

1. **Setup Your Environment:**

First, make sure you have Node.js installed on your system. If you haven't installed it yet, you can download and install it from Node.js official website.

2. **Initialize a Node.js Project:**

Create a new directory for your project and initialize it with npm:

Install Firestore SDK:

Install the Firebase Admin SDK, which will allow you to interact with Firestore:

```
npm install firebase-admin
```

Set Up Firebase Admin SDK:

To use the Firebase Admin SDK, you'll need to generate a service account key from your Firebase project. Follow these steps:

- Go to the Firebase Console.
- Select your project.
- Click on the gear icon next to "Project Overview" and select "Project settings".
- Navigate to the "Service accounts" tab.
- Click on "Generate new private key" and download the JSON file.

Move this JSON file to your project directory and rename it to serviceAccountKey.json (or keep its original name).

Write the Script:

Create a file named exportSchema.js and add the following code to it:

```
const admin = require('firebase-admin');
const fs = require('fs');

const serviceAccount = require('./serviceAccountKey.json');

admin.initializeApp({
  credential: admin.credential.cert(serviceAccount)
});

const db = admin.firestore();

async function exportCollections() {
  const collections = await db.listCollections();
  const schema = {};

  for (const collection of collections) {
    const collectionName = collection.id;
```

```
    schema[collectionName] = {};

    const documents = await collection.listDocuments();
    for (const doc of documents) {
      const docSnapshot = await doc.get();
      schema[collectionName][doc.id] = docSnapshot.data();
    }
  }

  fs.writeFileSync('firestore-schema.json', JSON.stringify(schema, null, 2));
  console.log('Schema exported successfully.');
}

exportCollections().catch(console.error);
```

Run the Script:

Execute the script using Node.js

```
node exportSchema.js
```

Handling Subcollections (Optional):

If your Firestore database includes subcollections, you need to recursively fetch them. Modify the script to handle subcollections:

```
const admin = require('firebase-admin');
const fs = require('fs');

const serviceAccount = require('./serviceAccountKey.json');

admin.initializeApp({
  credential: admin.credential.cert(serviceAccount)
});

const db = admin.firestore();

async function getSubcollections(docRef) {
  const subcollections = await docRef.listCollections();
  const subcollectionData = {};

  for (const subcollection of subcollections) {
    const subcollectionName = subcollection.id;
    subcollectionData[subcollectionName] = {};

    const subdocs = await subcollection.listDocuments();
```

```
    for (const subdoc of subdocs) {
      const subdocSnapshot = await subdoc.get();
      subcollectionData[subcollectionName][subdoc.id] = subdocSnapshot.data();

      // Recursively get subcollections of subdocuments
      const subdocSubcollections = await getSubcollections(subdoc);
      if (Object.keys(subdocSubcollections).length > 0) {
        subcollectionData[subcollectionName][subdoc.id].subcollections =
subdocSubcollections;
      }
    }
  }

  return subcollectionData;
}

async function exportCollections() {
  const collections = await db.listCollections();
  const schema = {};

  for (const collection of collections) {
    const collectionName = collection.id;
    schema[collectionName] = {};

    const documents = await collection.listDocuments();
    for (const doc of documents) {
      const docSnapshot = await doc.get();
      schema[collectionName][doc.id] = docSnapshot.data();

      const subcollections = await getSubcollections(doc);
      if (Object.keys(subcollections).length > 0) {
        schema[collectionName][doc.id].subcollections = subcollections;
      }
    }
  }

  fs.writeFileSync('firestore-schema.json', JSON.stringify(schema, null, 2));
  console.log('Schema exported successfully.');
}

exportCollections().catch(console.error);
```

This enhanced script handles both collections and subcollections recursively, ensuring a more comprehensive export of your Firestore schema.

By following these steps, you can successfully export your Firestore schema using Node.js.

Connecting Flutter with local SQLite Database

Configure SQLite Database using methods mentioned on these URLs

https://docs.flutter.dev/cookbook/persistence/sqlite

https://pub.dev/packages/sqflite

Here is a complete example:

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

Future<void> main() async {
  WidgetsFlutterBinding.ensureInitialized();

  // Open the database and store the reference.
  final database = await openDatabase(
    // Set the path to the database. Note: Using the `join` function from the
    // `path` package is best practice to ensure the path is correctly
    // constructed for each platform.
    join(await getDatabasesPath(), 'mydb.db'),
    onCreate: (db, version) {
      // Run the CREATE TABLE statement on the database.
      return db.execute(
        'create table persons(id integer primary key, name text, age
integer)',
      );
    },
    version: 1,
  );

  runApp(MyApp(db: database));
}

class Person {
  final int id;
  final String name;
  final int age;

  const Person({
    required this.id,
    required this.name,
    required this.age,
  });

  // Convert a person into a Map. The keys must correspond to the names of the
  // columns in the database.
```

```
  Map<String, Object?> toMap() {
    return {
      'id': id,
      'name': name,
      'age': age,
    };
  }

  // Implement toString to make it easier to see information about
  // each person when using the print statement.
  @override
  String toString() {
    return 'Person{id: $id, name: $name, age: $age}';
  }
}

class MyApp extends StatelessWidget {
  final Database db;
  const MyApp({super.key, required this.db});

// Define a function that inserts persons into the database
  Future<void> insertPerson(Person person) async {
    try {
      await db.insert('persons', person.toMap(),
          conflictAlgorithm: ConflictAlgorithm.fail);
      debugPrint("Record inserted");
    } on DatabaseException catch (e) {
      debugPrint("Error: $e");
    }
  }

// A method that returns all the persons from the persons table.
  Future<List<Person>> getAllPersons() async {
    // Get a reference to the database.

    // Query the table for all the persons.
    final List<Map<String, Object?>> personMaps = await db.query('persons');

    // Convert the list of each person's fields into a list of `Person`
objects.

    // the below code is (a) destructuring the person object, (b) creating a
list of person objects and returning
    return [
      for (final {
            'id': id as int,
            'name': name as String,
            'age': age as int,
```

```
        } in personMaps)
      Person(id: id, name: name, age: age),
    ];
  }

// A method that retrieves all the persons from the persons table.
  Future<void> showPersons() async {
    // Get a reference to the database.

    // Query the table for all the persons.
    final List<Map<String, Object?>> personMaps = await db.query('persons');

    // to print all persons
    //   debugPrint(personMaps.toString());

    // Convert the list of each person's fields into a list of `person`
objects.

    // to print individual persons
    for (final {
          'id': id as int,
          'name': name as String,
          'age': age as int,
        } in personMaps) {
      debugPrint("$id $name $age");
    }
  }

// A method that retrieves all the persons from the persons table.
  Future<void> showSelectivePersons() async {
    // Get a reference to the database.

    var columnId = "name";
    int personAge = 20;
    String personName = "Faisal";

    // Query the table for all the persons meeting the criteria.
    final List<Map<String, Object?>> personMaps = await db.query("persons",
        columns: ["id", "name"],
        where: '$columnId = ?',
        whereArgs: [personName]);

    // Convert the list of each person's fields into a list of `person`
objects.
    for (final {'id': id as int, 'name': name as String} in personMaps) {
      debugPrint("$id $name");
    }
  }
```

```
Future<void> updatePerson(Person person) async {
  try {
    await db.update('persons', person.toMap(),
        where: 'id = ?', whereArgs: [person.id]);
    debugPrint("Record updated");
  } on DatabaseException catch (e) {
    debugPrint("Error: $e");
  }
}

Future<void> deletePerson({required int id}) async {
  try {
    await db.delete('persons', where: 'id = ?', whereArgs: [id]);
    debugPrint("Record deleted");
  } on DatabaseException catch (e) {
    debugPrint("Error: $e");
  }
}

// This widget is the root of your application.
@override
Widget build(BuildContext context) {
  return MaterialApp(
      title: 'Flutter Demo',
      theme: ThemeData(
        colorScheme: ColorScheme.fromSeed(seedColor: Colors.deepPurple),
        useMaterial3: true,
      ),
      home: Scaffold(
          appBar: AppBar(
              title: Center(
            child: Text("Flutter SQLite Database"),
          )),
          body: SafeArea(
              child: Center(
                  child: Column(
            children: [
              ElevatedButton(
                  onPressed: () =>
                      insertPerson(Person(id: 30, name: "Faisal", age: 25)),
                  child: Text("Insert Person")),
              ElevatedButton(
                  onPressed: () async {
                    // getAllPersons() is now a list of objects
                    debugPrint("${await getAllPersons()}");
                  },
                  child: Text("Get persons as a list")),
```

```
                ElevatedButton(
                    onPressed: () => showPersons(),
                    child: Text("Show all persons")),
                ElevatedButton(
                    onPressed: () => showSelectivePersons(),
                    child: Text("Show Selective persons")),
                ElevatedButton(
                    onPressed: () => updatePerson(
                        Person(id: 30, name: "Faisal Khan", age: 55)),
                    child: Text("Update person")),
                ElevatedButton(
                    onPressed: () => deletePerson(id: 20),
                    child: Text("Delete person")),
            ],
          )))));
  }
}
```

Output:

Flutter shared preferences

If you have a relatively small collection of key-values to save, you can use the shared_preferences plugin.

Supported types

Although the key-value storage provided by shared_preferences is easy and convenient to use, it has limitations:

- Only primitive types can be used: int, double, bool, String, and List<String>.
- It's not designed to store large amounts of data.
- There is no guarantee that data will be persisted across app restarts.

Configure using the following URL:

https://docs.flutter.dev/cookbook/persistence/key-value

Here is the complete example:

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

void main() => runApp(const MyApp());

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

  @override
  Widget build(BuildContext context) {
    return const MaterialApp(
      title: 'Shared preferences demo',
      home: MyHomePage(title: 'Shared preferences demo'),
    );
  }
}

class MyHomePage extends StatefulWidget {
  const MyHomePage({super.key, required this.title});

  final String title;

  @override
  State<MyHomePage> createState() => _MyHomePageState();
}

class _MyHomePageState extends State<MyHomePage> {
  int _counter = 0;

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

  /// Load the initial counter value from persistent storage on start,
  /// or fallback to 0 if it doesn't exist.
```

```dart
Future<void> _loadCounter() async {
  final prefs = await SharedPreferences.getInstance();
  setState(() {
    _counter = prefs.getInt('counter') ?? 0;
  });
}

/// After a click, increment the counter state and
/// asynchronously save it to persistent storage.
Future<void> _incrementCounter() async {
  final prefs = await SharedPreferences.getInstance();
  setState(() {
    _counter = (prefs.getInt('counter') ?? 0) + 1;
    prefs.setInt('counter', _counter);
  });
}

Future<void> _removeCounter() async {
  final prefs = await SharedPreferences.getInstance();
  // Remove the counter key-value pair from persistent storage.
  await prefs.remove('counter');
}

@override
Widget build(BuildContext context) {
  return Scaffold(
    appBar: AppBar(
      title: Text(widget.title),
    ),
    body: Center(
      child: Column(
        mainAxisAlignment: MainAxisAlignment.center,
        children: [
          const Text(
            'You have pushed the button this many times: ',
          ),
          Text(
            '$_counter',
            style: Theme.of(context).textTheme.headlineMedium,
          ),
          ElevatedButton(
              onPressed: _incrementCounter, child: Text("Increment")),
          ElevatedButton(
              onPressed: _removeCounter, child: Text("Remove counter")),
        ],
      ),
    ),
  );
```

```
  }
}
```

Flutter http Plugin

Follow this URL to make necessary configurations:
https://docs.flutter.dev/cookbook/networking/fetch-data

In the following file

flutterapp\android\app\src\main\AndroidManifest.xml

add the following permissions above the <application tag.

<uses-permission android:name="android.permission.INTERNET" />

Here is a simple example of retrieving data from the Internet

```
import 'dart:async';
import 'dart:convert';

import 'package:flutter/material.dart';
import 'package:http/http.dart' as http;

Future<Album> fetchAlbum() async {
  final response = await http
      .get(Uri.parse('https://jsonplaceholder.typicode.com/albums/1'));

  if (response.statusCode == 200) {
    // If the server did return a 200 OK response,
    // then parse the JSON.
    return Album.fromJson(jsonDecode(response.body) as Map<String, dynamic>);
  } else {
    // If the server did not return a 200 OK response,
    // then throw an exception.
    throw Exception('Failed to load album');
  }
}

class Album {
```

```
  final int userId;
  final int id;
  final String title;

  const Album({
    required this.userId,
    required this.id,
    required this.title,
  });

  factory Album.fromJson(Map<String, dynamic> json) {
    return switch (json) {
      {
        'userId': int userId,
        'id': int id,
        'title': String title,
      } =>
        Album(
          userId: userId,
          id: id,
          title: title,
        ),
      _ => throw const FormatException('Failed to load album.'),
    };
  }
}

void main() => runApp(const MyApp());

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

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

class _MyAppState extends State<MyApp> {
  late Future<Album> futureAlbum;

  @override
  void initState() {
    super.initState();
    futureAlbum = fetchAlbum();
  }

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

```
      title: 'Fetch Data Example',
      theme: ThemeData(
        colorScheme: ColorScheme.fromSeed(seedColor: Colors.deepPurple),
      ),
      home: Scaffold(
        appBar: AppBar(
          title: const Text('Fetch Data Example'),
        ),
        body: Center(
          child: FutureBuilder<Album>(
            future: futureAlbum,
            builder: (context, snapshot) {
              if (snapshot.hasData) {
                return Text(snapshot.data!.title);
              } else if (snapshot.hasError) {
                return Text('${snapshot.error}');
              }

              // By default, show a loading spinner.
              return const CircularProgressIndicator();
            },
          ),
        ),
      ),
    );
  }
}
```

Insall Xampp server

Create the following database in phpMyAdmin

Database name: **mydb**

Connecting with PHP and making updates in MySQL using Flutter

Create the following database in mysql (you can use phpMyAdmin after installing xampp server):

```
SET SQL_MODE = "NO_AUTO_VALUE_ON_ZERO";
START TRANSACTION;
SET time_zone = "+00:00";

--
-- Database: `mydb`
--
```

```
-- --------------------------------------------------------

--
-- Table structure for table `users`
--

CREATE TABLE `users` (
  `id` int(11) NOT NULL,
  `uname` varchar(50) NOT NULL,
  `age` int(11) NOT NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_general_ci;

--
-- Dumping data for table `users`
--

INSERT INTO `users` (`id`, `uname`, `age`) VALUES
(1, 'Majid Khan', 44);

--
-- Indexes for dumped tables
--

--
-- Indexes for table `users`
--
ALTER TABLE `users`
  ADD PRIMARY KEY (`id`);

--
-- AUTO_INCREMENT for dumped tables
--

--
-- AUTO_INCREMENT for table `users`
--
ALTER TABLE `users`
  MODIFY `id` int(11) NOT NULL AUTO_INCREMENT, AUTO_INCREMENT=2;
COMMIT;
```

Create an app in php with the name: “phpapp”.

Add the following files in the app:

opendb.php

```php
<?php
$servername = "localhost";
```

```php
$username = "root";
$password = "";  // set this field "" (empty quotes) if you have not set any password in mysql
$dbname = "mydb";
$e = "";

try {
    $conn = new PDO("mysql:host=$servername;dbname=$dbname", $username, $password);
    // set the PDO error mode to exception
    $conn->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);

 // echo "Connection successful";
}
catch(PDOException $e)
    {
    echo $e->getMessage();
    }

?>
```

menu.php

```php
<table>
    <tr>
        <td>
            <a href="index.php">Home</a>
        </td>

        <td>
            <a href="insertrecord.php">Insert</a>
        </td>


        </tr>
    </table>
```

index.php

```php
<!doctype html>
<html>
<head>
<meta charset="utf-8">

<title>Untitled Document</title>
```

```
</head>

<body>
    <?php
    include("menu.php");
    include("opendb.php");

    $query = "select * from users";
    $stmt = $conn -> prepare($query);
    $stmt -> execute()
    ?>
    <br>
    <table border="1">

          <?php
          while($row = $stmt->fetch())
            {
              ?>
          <tr>
          <td><?php echo $row[0]; ?></td>
          <td><?php echo $row[1]; ?></td>
          <td><?php echo $row[2]; ?></td>
        </tr>
<?php

            }
          ?>


    </table>

    <?php
        $conn = NULL;
    ?>

</body>
</html>
```

insertrecord.php

```
<!doctype html>
<html>
<head>
<meta charset="utf-8">
<title>Insert Record</title>
```

```
</head>

<body>
    <?php
    include("menu.php");
    include("opendb.php");

    if(isset($_POST['btnsave']))
    {
        $name = $_POST['txtname'];
        $age = $_POST['txtage'];

    try {
            $query = "insert into users(uname, age) values(:uname, :age)";
            $stmt = $conn -> prepare($query);
            $stmt -> bindParam(':uname', $name);
            $stmt -> bindParam(':age', $age);
            $stmt -> execute();
    }
    catch(PDOException $e)
    {
            echo $e -> getMessage();
    }

    }
    ?>

    <form id="frm" name="frm" method="post" action="">
    <table border="1">

        <tr>
          <td>Name</td>
          <td>
          <input type="text" name="txtname" id="txtname"></td>
        </tr>
        <tr>
          <td>Age</td>
          <td>
          <input id="txtage" name="txtage" type="text" >
        </td>
        </tr>
        <tr>
          <td><input name="btnsave" type="submit" id="btnsave" title="Save"
value="Save"></td>
          <td> </td>
        </tr>

    </table>
```

```
    </form>

    <?php
        $conn = NULL;
    ?>

</body>
</html>
```

getsingleobject.php

```
<?php
    $Message = "This is a response";

    //$Response[] = array("Message" => $Message, "Second" =>"This is second");

    $data = array(
        "userid" => 100,
        "uname" => "Zahid Khan",
        "age" => 45
    );


    echo json_encode($data);

    //echo json_encode($Response);
    ?>
```

getsinglerecord.php

```
<?php
    include("opendb.php");
    $result = null;
    $userid = 1;

    try {
        $query = "select * from users where userid = :userid";
        $stmt = $conn -> prepare($query);
        $stmt -> bindParam(':userid', $userid);

        $stmt -> execute();
```

```
        $result = json_encode($stmt->fetch(PDO::FETCH_ASSOC));
    }
    catch(PDOException $e)
    {
        $result = $e -> getMessage();
    }

    echo $result;
    ?>
```

getsinglerecordforinput.php

```
<?php
   $EncodedData = file_get_contents('php://input');
   $DecodedData = json_decode($EncodedData, true);
   $userid = $DecodedData['userInput'];

    include("opendb.php");

    $result = null;


    try {
        $query = "select * from users where userid = :userid";
        $stmt = $conn -> prepare($query);
        $stmt -> bindParam(':userid', $userid);

        $stmt -> execute();

        $result = json_encode($stmt->fetch(PDO::FETCH_ASSOC));
    }
    catch(PDOException $e)
    {
        $result = $e -> getMessage();
    }

    $conn = NULL;
    echo $result;


    ?>
```

getalldata.php

```
<?php
    include("opendb.php");
```

```
    $query = "select * from users";
    $stmt = $conn -> prepare($query);
    $stmt -> execute();

    $resultJSON = json_encode($stmt->fetchAll(PDO::FETCH_ASSOC));

    $conn = null;

    echo $resultJSON;
?>
```

setdata.php

```
<?php

include("opendb.php");


        $EncodedData = file_get_contents('php://input');
        $DecodedData = json_decode($EncodedData, true);
        $name = $DecodedData['name'];
        $age = $DecodedData['age'];
        $Response = null;


        $Message = "Record inserted";


        try {
                $query = "insert into users(uname, age) values(:uname, :age)";
                $stmt = $conn -> prepare($query);
                $stmt -> bindParam(':uname', $name);
                $stmt -> bindParam(':age', $age);
                $stmt -> execute();
                $Response = array("Code" => "Success", "Message" => "Record
inserted");
        }
        catch(PDOException $e)
        {
        $Message = $e -> getMessage();
        $Response = array("Code" => "Failed", "Message" => $Message);
        }

    $conn = NULL;

    echo json_encode($Response);
```

```
    ?>
```

updatedata.php

```
<?php
include("opendb.php");

        $EncodedData = file_get_contents('php://input');
        $DecodedData = json_decode($EncodedData, true);

        $id = $DecodedData['id'];
        $name = $DecodedData['name'];
        $age = $DecodedData['age'];

        $Message = "Record updated";
        $Response = null;

    try {
            $query = "update users set uname = :uname, age = :age where userid
= :userid";

            $stmt = $conn -> prepare($query);


            $stmt -> bindParam(':uname', $name);
            $stmt -> bindParam(':age', $age);
            $stmt -> bindParam(':userid', $id);
            $stmt -> execute();

            $Response = array("Code" => "Success", "Message" => $Message);

    }
    catch(PDOException $e)
    {
        $Message = $e -> getMessage();
        $Response = array("Code" => "Success", "Message" => $Message);

    }

    $conn = NULL;
    echo json_encode($Response);

    ?>
```

deletedata.php

```php
<?php
include("opendb.php");

        $EncodedData = file_get_contents('php://input');
        $DecodedData = json_decode($EncodedData, true);

        $id = $DecodedData['id'];

        $Response = null;

        try {
            $query = "delete from users where userid = :userid";
            $stmt = $conn -> prepare($query);
            $stmt -> bindParam(':userid', $id);
            $stmt -> execute();
            $count = $stmt -> rowCount();

            if($count > 0)
            $Response = array("Code" => "Success", "Message" => "Record
deleted");
            else
            $Response = array("Code" => "NotFound", "Message" => "Record not
found");
        }
        catch(PDOException $e)
        {
        $Message = $e -> getMessage();
        $Response = array("Code" => "Failed", "Message" => $Message);
        }

    $conn = NULL;

    echo json_encode($Response);

    ?>
```

upload.php

```php
<?php
 if(!empty($_FILES['file_attachment']['name']))
 {

    $target_dir = "uploads/";
    if (!file_exists($target_dir))
    {
      mkdir($target_dir, 0777);
```

```
}
$target_file =
  $target_dir . basename($_FILES["file_attachment"]["name"]);
$imageFileType =
  strtolower(pathinfo($target_file,PATHINFO_EXTENSION));
// Check if file already exists
if (file_exists($target_file)) {
  echo json_encode(
     array(
       "status" => 0,
       "data" => array()
       ,"msg" => "Sorry, file already exists."
     )
  );
  die();
}
// Check file size
if ($_FILES["file_attachment"]["size"] > 50000000) {
  echo json_encode(
     array(
       "status" => 0,
       "data" => array(),
       "msg" => "Sorry, your file is too large."
     )
   );
  die();
}
if (
  move_uploaded_file(
    $_FILES["file_attachment"]["tmp_name"], $target_file
  )
) {
  echo json_encode(
    array(
      "status" => 1,
      "data" => array(),
      "msg" => "The file ".
               basename( $_FILES["file_attachment"]["name"]) .
               " has been uploaded."));
} else {
  echo json_encode(
    array(
      "status" => 0,
      "data" => array(),
      "msg" => "Sorry, there was an error uploading your file."
    )
  );
}
```

```
}


?>
```

main.dart file

```
import 'package:flutter/material.dart';
import 'dart:convert';
import 'package:file_picker/file_picker.dart';
import 'package:http/http.dart' as http;

String serverAddress = "http://192.168.0.109/phpapp/";

class Data {
  final int userid;
  final String uname;
  final int age;

  const Data({
    required this.userid,
    required this.uname,
    required this.age,
  });

  factory Data.fromJson(Map<String, dynamic> json) {
    return switch (json) {
      {
        'userid': int userid,
        'uname': String uname,
        'age': int age,
      } =>
        Data(
          userid: userid,
          uname: uname,
          age: age,
        ),
      _ => throw const FormatException('Failed to load Data.'),
    };
  }
}
```

```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 _uname = "Press the button to fetch user's name in Text() widget";
  FilePickerResult? _file;
  bool _isUploading = false;

  void fetchAlbum() async {
    final response = await http
        .get(Uri.parse('https://jsonplaceholder.typicode.com/albums/1'));

    if (response.statusCode == 200) {
      // If the server did return a 200 OK response,
      // then parse the JSON.
      var data = jsonDecode(response.body);
      print(data);
    } else {
      // If the server did not return a 200 OK response,
      // then throw an exception.
      throw Exception('Failed to load album');
    }
  }

  void fetchData() async {
    final response =
        await http.get(Uri.parse("$serverAddress/getsingleobject.php"));

    if (response.statusCode == 200) {
      // If the server did return a 200 OK response,
      // then parse the JSON.
      var data =
          Data.fromJson(jsonDecode(response.body) as Map<String, dynamic>);
      setState(() {
        _uname = data.uname;
      });
    } else {
      // If the server did not return a 200 OK response,
      // then throw an exception.
```

```
      throw Exception('Failed to load Data');
    }
  }

  void fetchAllData() async {
    final response = await
http.get(Uri.parse("$serverAddress/getalldata.php"));

    if (response.statusCode == 200) {
      // If the server did return a 200 OK response,
      // then parse the JSON.
      var list = jsonDecode(response.body);

      for (var item in list) {
        debugPrint("${item['userid']}  ${item['uname']}  ${item['age']}");
      }
    } else {
      // If the server did not return a 200 OK response,
      // then throw an exception.
      throw Exception('Failed to load Data');
    }
  }

  void fetchSingleData() async {
    final response =
        await http.get(Uri.parse("$serverAddress/getsinglerecord.php"));

    if (response.statusCode == 200) {
      // If the server did return a 200 OK response,
      // then parse the JSON.
      var data = Data.fromJson(jsonDecode(response.body));

      debugPrint("${data.userid} ${data.uname} ${data.age}");
    } else {
      // If the server did not return a 200 OK response,
      // then throw an exception.
      throw Exception('Failed to load Data');
    }
  }

  void fetchSingleDataAgainstValue(String userInput) async {
    final response = await http.post(
      Uri.parse("$serverAddress/getsinglerecordforinput.php"),
      headers: <String, String>{
        'Content-Type': 'application/json; charset=UTF-8',
      },
      body: jsonEncode(<String, String>{
        'userInput': userInput,
```

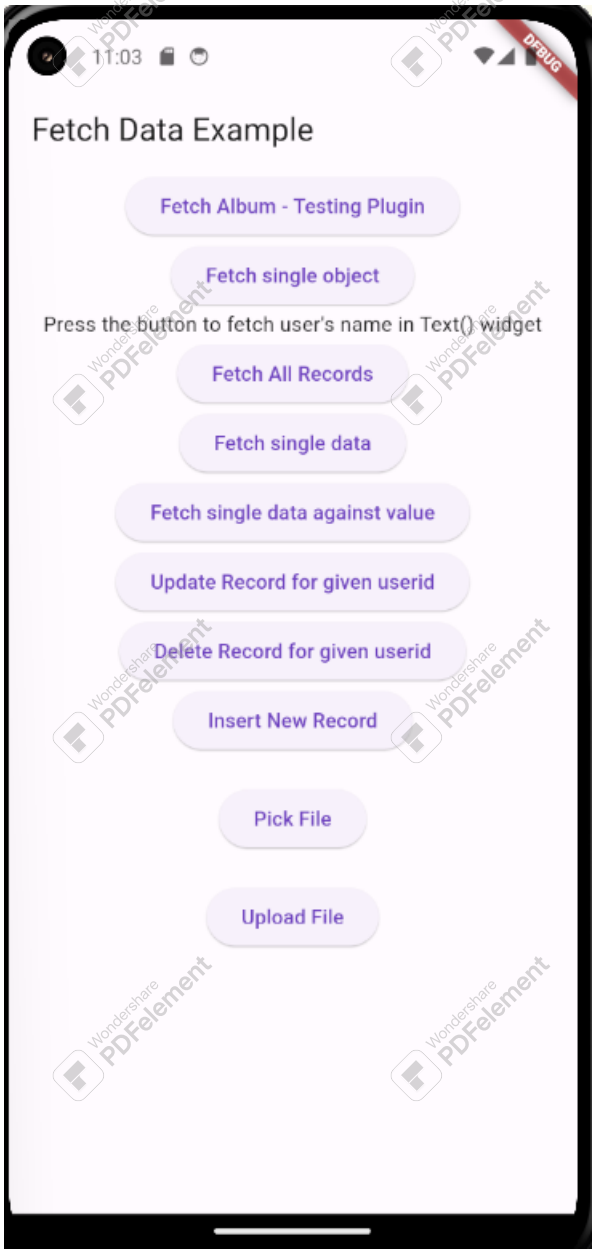
```
        // add more key value pairs here.
        // ...
      }),
    );

    if (response.statusCode == 200) {
      var result = jsonDecode(response.body);

      debugPrint(" ${result['userid']}  ${result['uname']}  ${result['age']}
");
    } else {
      // If the server did not return a 201 CREATED response,
      // then throw an exception.
      throw Exception('Failed to create album.');
    }
  }

  void updateRecord(String userId, String userName, String userAge) async {
    final response = await http.post(
      Uri.parse("$serverAddress/updatedata.php"),
      headers: <String, String>{
        'Content-Type': 'application/json; charset=UTF-8',
      },
      body: jsonEncode(
          <String, String>{'id': userId, 'name': userName, 'age': userAge}),
    );

    if (response.statusCode == 200) {
      var result = jsonDecode(response.body);

      print("${result["Code"]}  ${result["Message"]}");
    } else {
      // If the server did not return a  response,
      // then throw an exception.
      throw Exception('Failed .');
    }
  }

  void deleteRecord(String userId) async {
    final response = await http.post(
      Uri.parse("$serverAddress/deletedata.php"),
      headers: <String, String>{
        'Content-Type': 'application/json; charset=UTF-8',
      },
      body: jsonEncode(<String, String>{
        'id': userId,
      }),
    );
```

```
  if (response.statusCode == 200) {
    var result = jsonDecode(response.body);

    print("${result["Code"]}:  ${result["Message"]}");
  } else {
    // If the server did not return a  response,
    // then throw an exception.
    throw Exception('Failed .');
  }
}

void insertRecord(String userName, String userAge) async {
  final response = await http.post(
    Uri.parse("$serverAddress/setdata.php"),
    headers: <String, String>{
      'Content-Type': 'application/json; charset=UTF-8',
    },
    body: jsonEncode(<String, String>{'name': userName, 'age': userAge}),
  );

  if (response.statusCode == 200) {
    var result = jsonDecode(response.body);

    print("${result["Code"]}:  ${result["Message"]}");
  } else {
    // If the server did not return a  response,
    // then throw an exception.
    throw Exception('Failed .');
  }
}

Future<void> _pickFile() async {
  FilePickerResult? result = await FilePicker.platform.pickFiles();

  if (result != null) {
    setState(() {
      _file = result;
    });
  }
}

Future<void> _uploadFile() async {
  if (_file != null) {
    setState(() {
      _isUploading = true;
    });
```

```
    var uri = Uri.parse("$serverAddress/upload.php");
    var request = http.MultipartRequest('POST', uri);
    request.files.add(await http.MultipartFile.fromPath(
      'file_attachment',
      _file!.files.single.path!,
      filename: _file!.files.single.name,
    ));

    var response = await request.send();

    if (response.statusCode == 200) {
      // Handle success
      print(response);
    } else {
      // Handle error
      print('File upload failed');
    }

    setState(() {
      _isUploading = false;
    });
  }
}

@override
Widget build(BuildContext context) {
  return MaterialApp(
    title: 'Fetch Data Example',
    theme: ThemeData(
      colorScheme: ColorScheme.fromSeed(seedColor: Colors.deepPurple),
    ),
    home: Scaffold(
      appBar: AppBar(
        title: const Text('Fetch Data Example'),
      ),
      body: Center(
          child: Column(children: [
        ElevatedButton(
            onPressed: fetchAlbum,
            child: Text("Fetch Album - Testing Plugin")),
        ElevatedButton(
            onPressed: fetchData, child: Text("Fetch single object")),
        Text(_uname),
        ElevatedButton(
            onPressed: fetchAllData, child: Text("Fetch All Records")),
        ElevatedButton(
            onPressed: fetchSingleData, child: Text("Fetch single data")),
        ElevatedButton(
```

```
            onPressed: () => fetchSingleDataAgainstValue("1"),
            child: Text("Fetch single data against value")),
        ElevatedButton(
            onPressed: () => updateRecord("1", "Jawad Khan", "44"),
            child: Text("Update Record for given userid")),
        ElevatedButton(
            onPressed: () => deleteRecord("2"),
            child: Text("Delete Record for given userid")),
        ElevatedButton(
            onPressed: () => insertRecord("Majid Khan", "44"),
            child: Text("Insert New Record")),
        if (_file != null) Text('File selected:
${_file!.files.single.name}'),
        SizedBox(height: 20),
        ElevatedButton(
          onPressed: _pickFile,
          child: Text('Pick File'),
        ),
        SizedBox(height: 20),
        ElevatedButton(
          onPressed: _isUploading ? null : _uploadFile,
          child: _isUploading
              ? CircularProgressIndicator()
              : Text('Upload File'),
        ),
      ])),
    ),
  );
 }
}
```

Output Window:



Communicate with WebSockets

In addition to normal HTTP requests, you can connect to servers using WebSockets. WebSockets allow for two-way communication with a server without polling.

In this example, connect to a test WebSocket server sponsored by Lob.com. The server sends back the same message you send to it. This recipe uses the following steps:

1. Connect to a WebSocket server.
2. Listen for messages from the server.
3. Send data to the server.
4. Close the WebSocket connection.

Follow this URL for complete example.

https://docs.flutter.dev/cookbook/networking/web-sockets

Handling Camera in Flutter

Perform necessary configurations using this URL:

https://docs.flutter.dev/cookbook/plugins/picture-using-camera

You may also have to upgrade your api version in build.gradle file on this path:

flutterapp\android\app\build.gradle

Make following changes in android -> defaultConfig -> :

android {

…

defaultConfig {

…..

minSdkVersion 23

targetSdkVersion 23

Example:

```
import 'dart:async';
import 'dart:io';

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

Future<void> main() async {
  // Ensure that plugin services are initialized so that `availableCameras()`
  // can be called before `runApp()`
  WidgetsFlutterBinding.ensureInitialized();

  // Obtain a list of the available cameras on the device.
  final cameras = await availableCameras();

  // Get a specific camera from the list of available cameras.
```

```dart
  final firstCamera = cameras.first;

  runApp(
    MaterialApp(
      theme: ThemeData.dark(),
      home: TakePictureScreen(
        // Pass the appropriate camera to the TakePictureScreen widget.
        camera: firstCamera,
      ),
    ),
  );
}

// A screen that allows users to take a picture using a given camera.
class TakePictureScreen extends StatefulWidget {
  const TakePictureScreen({
    super.key,
    required this.camera,
  });

  final CameraDescription camera;

  @override
  TakePictureScreenState createState() => TakePictureScreenState();
}

class TakePictureScreenState extends State<TakePictureScreen> {
  late CameraController _controller;
  late Future<void> _initializeControllerFuture;

  @override
  void initState() {
    super.initState();
    // To display the current output from the Camera,
    // create a CameraController.
    _controller = CameraController(
      // Get a specific camera from the list of available cameras.
      widget.camera,
      // Define the resolution to use.
      ResolutionPreset.medium,
    );

    // Next, initialize the controller. This returns a Future.
    _initializeControllerFuture = _controller.initialize();
  }

  @override
  void dispose() {
```

```dart
    // Dispose of the controller when the widget is disposed.
    _controller.dispose();
    super.dispose();
  }

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(title: const Text('Take a picture')),
      // You must wait until the controller is initialized before displaying the
      // camera preview. Use a FutureBuilder to display a loading spinner until the
      // controller has finished initializing.
      body: FutureBuilder<void>(
        future: _initializeControllerFuture,
        builder: (context, snapshot) {
          if (snapshot.connectionState == ConnectionState.done) {
            // If the Future is complete, display the preview.
            return CameraPreview(_controller);
          } else {
            // Otherwise, display a loading indicator.
            return const Center(child: CircularProgressIndicator());
          }
        },
      ),
      floatingActionButton: FloatingActionButton(
        // Provide an onPressed callback.
        onPressed: () async {
          // Take the Picture in a try / catch block. If anything goes wrong,
          // catch the error.
          try {
            // Ensure that the camera is initialized.
            await _initializeControllerFuture;

            // Attempt to take a picture and get the file `image`
            // where it was saved.
            final image = await _controller.takePicture();

            if (!context.mounted) return;

            // If the picture was taken, display it on a new screen.
            await Navigator.of(context).push(
              MaterialPageRoute(
                builder: (context) => DisplayPictureScreen(
                  // Pass the automatically generated path to
                  // the DisplayPictureScreen widget.
                  imagePath: image.path,
```

```
                ),
              ),
            );
          } catch (e) {
            // If an error occurs, log the error to the console.
            print(e);
          }
        },
        child: const Icon(Icons.camera_alt),
      ),
    );
  }
}

// A widget that displays the picture taken by the user.
class DisplayPictureScreen extends StatelessWidget {
  final String imagePath;

  const DisplayPictureScreen({super.key, required this.imagePath});

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(title: const Text('Display the Picture')),
      // The image is stored as a file on the device. Use the `Image.file`
      // constructor with the given path to display the image.
      body: Image.file(File(imagePath)),
    );
  }
}
```

GPS tracking in Flutter

You can use the geolocator package for GPS tracking:

In the following file

flutterapp\android\app\src\main\AndroidManifest.xml

add the following dependencies above the <application tag.

```
<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION"/>
<uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION"/>
```

If you get a Kotlin version error, see the latest version online and make changes here (see the third line in below code in the following file):

\flutterapp\android\settings.gradle

```
plugins {

    id "dev.flutter.flutter-plugin-loader" version "1.0.0"
    id "com.android.application" version "7.3.0" apply false
    id "org.jetbrains.kotlin.android" version "2.0.0" apply false
}
```

Example of GPS Tracking:

```
import 'dart:async';

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

void main() => runApp(MyApp());

class MyApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      home: Scaffold(
        appBar: AppBar(
          title: Text('GPS Tracking'),
        ),
        body: LocationTracking(),
      ),
    );
  }
}

class LocationTracking extends StatefulWidget {
  @override
  _LocationTrackingState createState() => _LocationTrackingState();
}

class _LocationTrackingState extends State<LocationTracking> {
  Position? _currentPosition;
  StreamSubscription<Position>? _positionStreamSubscription;

  @override
```

```dart
void initState() {
  super.initState();
  _checkPermissions();
}

Future<void> _checkPermissions() async {
  LocationPermission permission = await Geolocator.checkPermission();
  if (permission == LocationPermission.denied) {
    permission = await Geolocator.requestPermission();
    if (permission == LocationPermission.denied) {
      // Permissions are denied, handle appropriately
      return;
    }
  }

  if (permission == LocationPermission.deniedForever) {
    // Permissions are denied forever, handle appropriately
    return;
  }

  // When permissions are granted, start tracking location
  _startTrackingLocation();
}

void _startTrackingLocation() {
  final positionStream = Geolocator.getPositionStream(
    locationSettings: LocationSettings(
      accuracy: LocationAccuracy.high,
      distanceFilter: 10,
    ),
  );

  _positionStreamSubscription = positionStream.listen((Position position) {
    setState(() {
      _currentPosition = position;
    });
  });
}

@override
void dispose() {
  _positionStreamSubscription?.cancel();
  super.dispose();
}

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

```
      child: _currentPosition == null
          ? Text('Getting location...')
          : Text(
              'Location: ${_currentPosition!.latitude},
${_currentPosition!.longitude}'),
    );
  }
}
```