**In order:**

**1.A startapp project in flutter**

**i**mport 'package:flutter/material.dart';

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

class MyApp extends StatelessWidget {

@override

Widget build(BuildContext context) {

return MaterialApp(

title: 'Welcome to Flutter',

home: Scaffold(

appBar: AppBar(

title: Text('Welcome to Flutter'),

),

body: Center(

child: Text('Hello World'),

),

),

);

}

Observations

* This example creates a Material app. [Material](https://material.io/guidelines) is a visual design language that is standard on mobile and the web. Flutter offers a rich set of Material widgets. It’s a good idea to have a uses-material-design: true entry in the flutter section of your pubspec.yaml file. This will allow you to use more features of Material, such as their set of predefined [Icons](https://design.google.com/icons/).
* The main() method uses arrow (=>) notation. Use arrow notation for one-line functions or methods.
* The app extends StatelessWidget, which makes the app itself a widget. In Flutter, almost everything is a widget, including alignment, padding, and layout.
* The Scaffold widget, from the Material library, provides a default app bar, and a body property that holds the widget tree for the home screen. The widget subtree can be quite complex.
* A widget’s main job is to provide a build() method that describes how to display the widget in terms of other, lower level widgets.
* The body for this example consists of a Center widget containing a Text child widget. The Center widget aligns its widget subtree to the center of the screen

## 2. Use an external package

In this step, you’ll start using an open-source package named [english\_words](https://pub.dev/packages/english_words), which contains a few thousand of the most used English words plus some utility functions.

You can find the english\_words package, as well as many other open source packages, on [pub.dev](https://pub.dev/).

1. The pubspec.yaml file manages the assets and dependencies for a Flutter app. In pubspec.yaml, add english\_words (3.1.5 or higher) to the dependencies list:

{step1\_base → step2\_use\_package}/pubspec.yaml

|  |  |
| --- | --- |
|  | @@ -8,4 +8,5 @@ |
| 8  8 | dependencies: |
| 9  9 | flutter: |
| 10  10 | sdk: flutter |
| 11  11 | cupertino\_icons: ^0.1.2 |
| 12 | +  english\_words: ^3.1.5 |

1. While viewing the pubspec.yaml file in Android Studio’s editor view, click **Pub get**. This pulls the package into your project. You should see the following in the console:

*content\_copy*

$ flutter pub get

Running "flutter pub get" in startup\_namer...

Process finished with exit code 0

Performing Pub get also auto-generates the pubspec.lock file with a list of all packages pulled into the project and their version numbers.

1. In lib/main.dart, import the new package:

lib/main.dart

*content\_copy*

import 'package:flutter/material.dart';

import 'package:english\_words/english\_words.dart';

As you type, Android Studio gives you suggestions for libraries to import. It then renders the import string in gray, letting you know that the imported library is unused (so far).

1. Use the English words package to generate the text instead of using the string “Hello World”:

{step1\_base → step2\_use\_package}/lib/main.dart

|  |  |
| --- | --- |
|  | @@ -9,6 +10,7 @@ |
| 9  10 | **class** **MyApp** **extends** **StatelessWidget** { |
| 10  11 | **@override** |
| 11  12 | Widget build(BuildContext context) { |
| 13 | +  **final** wordPair = WordPair.random(); |
| 12  14 | **return** MaterialApp( |
| 13  15 | title: 'Welcome to Flutter', |
| 14  16 | home: Scaffold( |
|  | @@ -16,7 +18,7 @@ |
| 16  18 | title: Text('Welcome to Flutter'), |
| 17  19 | ), |
| 18  20 | body: Center( |
| 19 | -  child: Text(), |
| 21 | +  child: Text(wordPair.asPascalCase), |
| 20  22 | ), |
| 21  23 | ), |
| 22  24 | ); |

If the app is running, [hot reload](https://flutter.dev/docs/get-started/test-drive) to update the running app. Each time you click hot reload, or save the project, you should see a different word pair, chosen at random, in the running app. This is because the word pairing is generated inside the build method, which is run each time the MaterialApp requires rendering, or when toggling the Platform in Flutter Inspector.

=========================for pub spec yaml==============

name: startup\_namer

description: A startup-namer app.

version: 1.0.0+1

environment:

sdk: ">=2.10.0 <3.0.0"

# #docregion dependencies

dependencies:

flutter:

sdk: flutter

cupertino\_icons: ^0.1.2

english\_words: ^3.1.5

# #enddocregion dependencies

dev\_dependencies:

flutter\_test:

sdk: flutter

pedantic: ^1.4.0

flutter:

uses-material-design: true

=========================================

for lib/dart

// Copyright 2018 The Flutter team. All rights reserved.

// Use of this source code is governed by a BSD-style license that can be

// found in the LICENSE file.

import 'package:flutter/material.dart';

import 'package:english\_words/english\_words.dart';

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

class MyApp extends StatelessWidget {

@override

Widget build(BuildContext context) {

final wordPair = WordPair.random();

return MaterialApp(

title: 'Welcome to Flutter',

home: Scaffold(

appBar: AppBar(

title: Text('Welcome to Flutter'),

),

body: Center(

child: Text(wordPair.asPascalCase),

),

),

);

}

}

## Step 3: Add a Stateful widget

Stateless widgets are immutable, meaning that their properties can’t change—all values are final.

Stateful widgets maintain state that might change during the lifetime of the widget. Implementing a stateful widget requires at least two classes: 1) a StatefulWidget class that creates an instance of 2) a State class. The StatefulWidget class is, itself, immutable and can be thrown away and regenerated, but the State class persists over the lifetime of the widget.

In this step, you’ll add a stateful widget, RandomWords, which creates its State class, \_RandomWordsState. You’ll then use RandomWords as a child inside the existing MyApp stateless widget.

1. Create the boilerplate code for a stateful widget.  
   In lib/main.dart, position your cursor after all of the code, enter **Return** a couple times to start on a fresh line. In your IDE, start typing stful. The editor asks if you want to create a Stateful widget. Press **Return** to accept. The boilerplate code for two classes appears, and the cursor is positioned for you to enter the name of your stateful widget.
2. Enter RandomWords as the name of your widget.  
   The RandomWords widget does little else beside creating its State class.  
     
   Once you’ve entered RandomWords as the name of the stateful widget, the IDE automatically updates the accompanying State class, naming it \_RandomWordsState. By default, the name of the State class is prefixed with an underbar. Prefixing an identifier with an underscore [enforces privacy](https://dart.dev/guides/language/language-tour#libraries-and-visibility) in the Dart language and is a recommended best practice for State objects.  
     
   The IDE also automatically updates the state class to extend State<RandomWords>, indicating that you’re using a generic [State](https://api.flutter.dev/flutter/widgets/State-class.html) class specialized for use with RandomWords. Most of the app’s logic resides here—it maintains the state for the RandomWords widget. This class saves the list of generated word pairs, which grows infinitely as the user scrolls and, in part 2 of this lab, favorites word pairs as the user adds or removes them from the list by toggling the heart icon.  
     
   Both classes now look as follows:

*content\_copy*

class RandomWords extends StatefulWidget {

@override

\_RandomWordsState createState() => \_RandomWordsState();

}

class \_RandomWordsState extends State<RandomWords> {

@override

Widget build(BuildContext context) {

return Container();

}

}

1. Update the build() method in \_RandomWordsState:

lib/main.dart (\_RandomWordsState)

*content\_copy*

class \_RandomWordsState extends State<RandomWords> {

@override

Widget build(BuildContext context) {

final wordPair = WordPair.random();

return Text(wordPair.asPascalCase);

}

}

1. Remove the word generation code from MyApp by making the changes shown in the following diff:

{step2\_use\_package → step3\_stateful\_widget}/lib/main.dart

|  |  |
| --- | --- |
|  | @@ -10,7 +10,6 @@ |
| 10  10 | **class** **MyApp** **extends** **StatelessWidget** { |
| 11  11 | **@override** |
| 12  12 | Widget build(BuildContext context) { |
| 13 | -  **final** wordPair = WordPair.random(); |
| 14  13 | **return** MaterialApp( |
| 15  14 | title: 'Welcome to Flutter', |
| 16  15 | home: Scaffold( |
|  | @@ -18,8 +17,8 @@ |
| 18  17 | title: Text('Welcome to Flutter'), |
| 19  18 | ), |
| 20  19 | body: Center( |
| 21 | -  child: (), |
| 20 | +  child: RandomWords(), |
| 22  21 | ), |
| 23  22 | ), |
| 24  23 | ); |
| 25  24 | } |

1. Restart the app. The app should behave as before, displaying a word pairing each time you hot reload or save the app.

**Tip:** If you see a warning on a hot reload that you might need to restart the app, consider restarting it. The warning might be a false positive, but restarting your app ensures that your changes are reflected in the app’s UI.

### Problems?

If your app is not running correctly, look for typos. If you want to try some of Flutter’s debugging tools, check out the [DevTools](https://flutter.dev/docs/development/tools/devtools) suite of debugging and profiling tools. If needed, use the code at the following link to get back on track.

* [lib/main.dart](https://raw.githubusercontent.com/flutter/codelabs/master/startup_namer/step3_stateful_widget/lib/main.dart)

## Step 4: Create an infinite scrolling ListView

In this step, you’ll expand \_RandomWordsState to generate and display a list of word pairings. As the user scrolls the list (displayed in a ListView widget) grows infinitely. ListView’s builder factory constructor allows you to build a list view lazily, on demand.

1. Add a \_suggestions list to the \_RandomWordsState class for saving suggested word pairings. Also, add a \_biggerFont variable for making the font size larger.

lib/main.dart

*content\_copy*

class \_RandomWordsState extends State<RandomWords> {

final \_suggestions = <WordPair>[];

final \_biggerFont = TextStyle(fontSize: 18.0);

// ···

}

Next, you’ll add a \_buildSuggestions() function to the \_RandomWordsState class. This method builds the ListView that displays the suggested word pairing.

The ListView class provides a builder property, itemBuilder, that’s a factory builder and callback function specified as an anonymous function. Two parameters are passed to the function—the BuildContext, and the row iterator, i. The iterator begins at 0 and increments each time the function is called. It increments twice for every suggested word pairing: once for the ListTile, and once for the Divider. This model allows the suggested list to continue growing as the user scrolls.

1. Add a \_buildSuggestions() function to the \_RandomWordsState class:

lib/main.dart (\_buildSuggestions)

*content\_copy*

Widget \_buildSuggestions() {

return ListView.builder(

padding: EdgeInsets.all(16.0),

itemBuilder: /\*1\*/ (context, i) {

if (i.isOdd) return Divider(); /\*2\*/

final index = i ~/ 2; /\*3\*/

if (index >= \_suggestions.length) {

\_suggestions.addAll(generateWordPairs().take(10)); /\*4\*/

}

return \_buildRow(\_suggestions[index]);

});

}

* 1. The itemBuilder callback is called once per suggested word pairing, and places each suggestion into a ListTile row. For even rows, the function adds a ListTile row for the word pairing. For odd rows, the function adds a Divider widget to visually separate the entries. Note that the divider might be difficult to see on smaller devices.
  2. Add a one-pixel-high divider widget before each row in the ListView.
  3. The expression i ~/ 2 divides i by 2 and returns an integer result. For example: 1, 2, 3, 4, 5 becomes 0, 1, 1, 2, 2. This calculates the actual number of word pairings in the ListView, minus the divider widgets.
  4. If you’ve reached the end of the available word pairings, then generate 10 more and add them to the suggestions list.

The \_buildSuggestions() function calls \_buildRow() once per word pair. This function displays each new pair in a ListTile, which allows you to make the rows more attractive in the next step.

1. Add a \_buildRow() function to \_RandomWordsState:

lib/main.dart (\_buildRow)

*content\_copy*

Widget \_buildRow(WordPair pair) {

return ListTile(

title: Text(

pair.asPascalCase,

style: \_biggerFont,

),

);

}

1. In the \_RandomWordsState class, update the build() method to use \_buildSuggestions(), rather than directly calling the word generation library. ([Scaffold](https://api.flutter.dev/flutter/material/Scaffold-class.html) implements the basic Material Design visual layout.) Replace the method body with the highlighted code:

lib/main.dart (build)

*content\_copy*

@override

Widget build(BuildContext context) {

return Scaffold(

appBar: AppBar(

title: Text('Startup Name Generator'),

),

body: \_buildSuggestions(),

);

}

1. In the MyApp class, update the build() method by changing the title, and changing the home to be a RandomWords widget:

{step3\_stateful\_widget → step4\_infinite\_list}/lib/main.dart

|  |  |
| --- | --- |
|  | @@ -10,15 +10,8 @@ |
| 10  10 | **class** **MyApp** **extends** **StatelessWidget** { |
| 11  11 | **@override** |
| 12  12 | Widget build(BuildContext context) { |
| 13  13 | **return** MaterialApp( |
| 14 | -  title: ' ', |
| 15 | -  home: ( |
| 14 | +  title: 'Startup Name Generator', |
| 15 | +  home: RandomWords(), |
| 16 | -  appBar: AppBar( |
| 17 | -  title: Text('Welcome to Flutter'), |
| 18 | -  ), |
| 19 | -  body: Center( |
| 20 | -  child: RandomWords(), |
| 21 | -  ), |
| 22 | -  ), |
| 23  16 | ); |
| 24  17 | } |

https://www.youtube.com/watch?v=pTJJsmejUOQ