



# FLUTTER TUTORIALS

---

BY: **TECMAN**

**TECMAN Lesson 2A**

# WRITING YOUR FIRST FLUTTER APP

## PART 1

---

1. This lesson will guide you to create your first flutter app.

# WRITING YOUR FIRST FLUTTER APP

## PART 1

---

### Contents

Step 1: Create the starter Flutter app

Step 2: Use an external package

Step 3: Add a Stateful widget

Step 4: Create an infinite scrolling ListView

**TECMAN LIMITED 2019**

# WRITING YOUR FIRST FLUTTER APP

## PART 1

---

### 1. What you will build in part 1

# WRITING YOUR FIRST FLUTTER APP

## PART 1

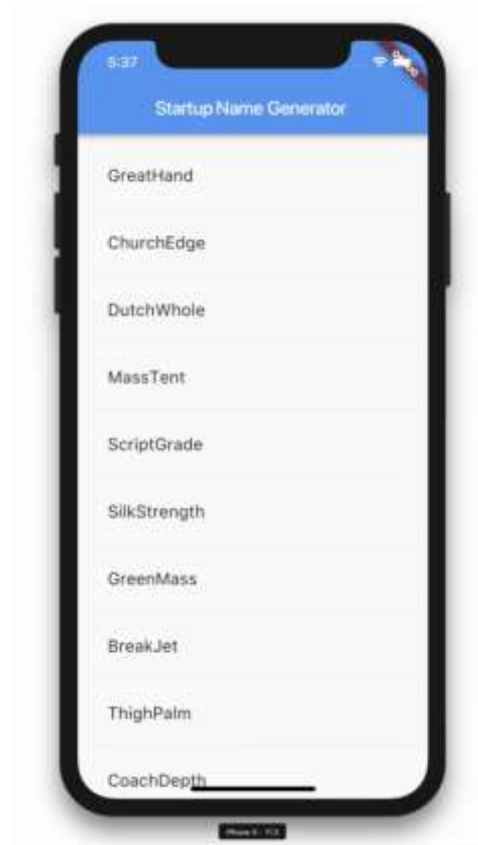
---

1. You will be taught how to implement a simple mobile app that generates proposed names for a startup company. The user can select and unselect names, saving the best ones. As the user scrolls, more names are generated.

# WRITING YOUR FIRST FLUTTER APP

## PART 1

---



TECMAN LIMITED 2019

# WRITING YOUR FIRST FLUTTER APP

## PART 1

---

### What you'll learn in part 1

- How to write a Flutter app that looks natural on both iOS and Android.
- Basic structure of a Flutter app.
- Finding and using packages to extend functionality.
- Using hot reload for a quicker development cycle.
- How to implement a stateful widget.
- How to create an infinite, lazily loaded list.

TECMAN LIMITED 2019

# Step 1: Create the starter Flutter app

---

1. Create a simple, templated Flutter app. Name the project **startup\_namer**
2. In this project you'll mostly be editing **lib/main.dart**, where the Dart code resides.



# Step 1: Create the starter Flutter app

---

1. Replace the contents of `lib/main.dart`.

Delete all of the code from **`lib/main.dart`**. Replace with the following code, which displays “Hello World” in the center of the screen.

# Step 1:

lib/main.dart

```
import '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'),
        ),
      ),
    );
  }
}
```



TECMAN LIMITED 2019

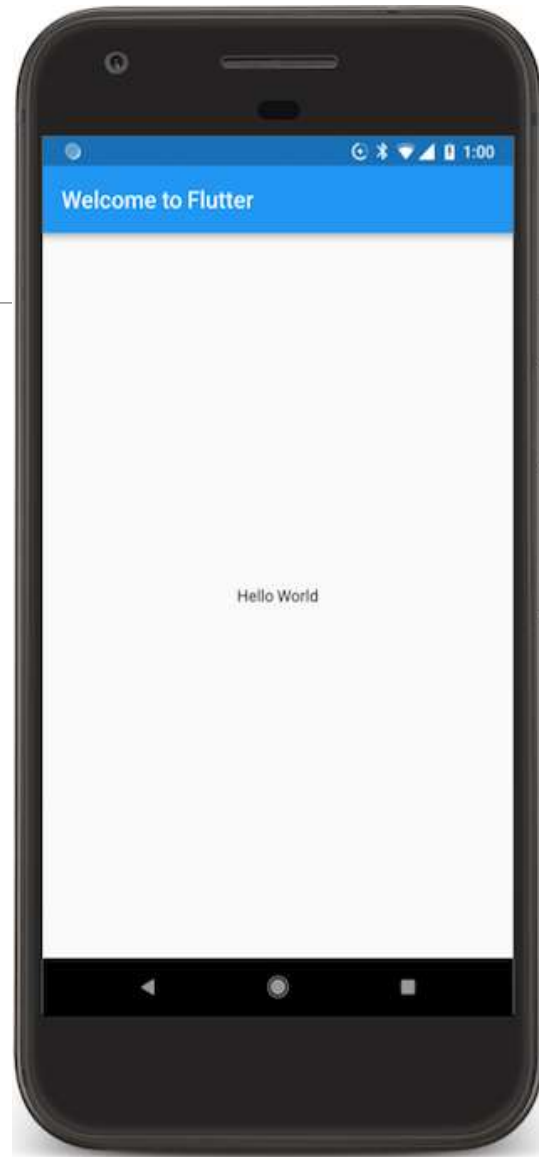
# Step 1: Create the starter Flutter app

---

1. Run the app on your device / emulator.

# Step 1:

---



ANDROID



iOS

TECMAN LIMITED 2019

# Step 1: Observations

---

- This example creates a Material app. Material is a visual design language that is standard on mobile and the web. Flutter offers a rich set of Material widgets.
- 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.

# Step 1: Observations

---

- The Scaffold widget, from the Material library, provides a default app bar, title, 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

## Step 2: Use an external package

---

1. In this step, you'll start using an open-source package named [english words](#), which contains a few thousand of the most used English words plus some utility functions.

# Step 2: Use an external package

1. The pubspec file manages the assets and dependencies for a Flutter app. In pubspec.yaml, add english\_words (3.1.0 or higher) to the dependencies list:

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

@@ -5,4 +5,5 @@

```
5 5 dependencies:
6 6   flutter:
7 7     sdk: flutter
8 8   cupertino_icons: ^0.1.2
9 +   english_words: ^3.1.0
```

TECMAN LIMITED 2019




## Step 2: Use an external package

---

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

lib/main.dart

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



2. 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).

TECMAN LIMITED 2019

## Step 2: Use an external package

---

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

## Step 2:

[icon] {step1_base → step2_use_package}/lib/main.dart		
		@@ -5,6 +6,7 @@
5	6	class MyApp extends StatelessWidget {
6	7	@override
7	8	Widget build(BuildContext context) {
	9	+    final wordPair = WordPair.random();
8	10	return MaterialApp(
9	11	title: 'Welcome to Flutter',
10	12	home: Scaffold(
		@@ -12,7 +14,7 @@
12	14	title: Text('Welcome to Flutter'),
13	15	),
14	16	body: Center(
15	-	child: Text('Hello World'),
	17	+        child: Text(wordPair.asPascalCase),
16	18	),
17	19	),
18	20	);
<		

TECMAN LIMITED 2019

## Step 2: Use an external package

---

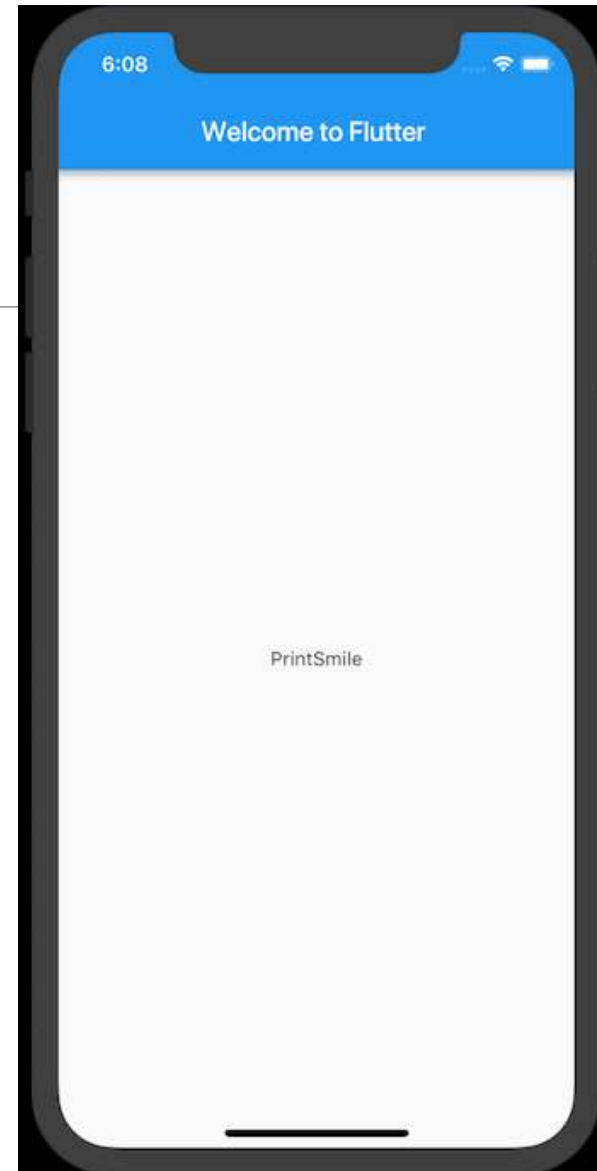
1. If the app is running, [hot reload](#) to update the running app. Each time you click hot reload, or save the project, you should see a different word pair, chosen at random, in the running app. This is because the word pairing is generated inside the build method, which is run each time the MaterialApp requires rendering, or when toggling the Platform in Flutter Inspector.

## Step 2:

---



ANDROID



iOS

TECMAN LIMITED 2019

# Step 2: Use an external package

---

## Problems?

If your app is not running correctly, look for typos.

# Step 3: Add a stateful widget

---

1. Stateless widgets are immutable, meaning that their properties can't change—all values are final.
2. 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, but the State class persists over the lifetime of the widget.

**TECMAN LIMITED 2019**

# Step 3: Add a stateful widget

---

1. 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.



# Step 3: Add a stateful widget

---

1. Create a minimal state class. Add the following to the bottom of main.dart:

lib/main.dart (RandomWordsState)

```
class RandomWordsState extends State<RandomWords> {  
  // TODO Add build() method  
}
```



# Step 3: Add a stateful widget

---

1. Notice the declaration `State<RandomWords>`. This indicates that we're using the generic `State` class specialized for use with `RandomWords`. Most of the app's logic and state resides here—it maintains the state for the `RandomWords` widget. This class saves the generated word pairs, which grows infinitely as the user scrolls, and favorite word pairs (in part 2), as the user adds or removes them from the list by toggling the heart icon.
2. `RandomWordsState` depends on the `RandomWords` class. You'll add that next.


# Step 3: Add a stateful widget

---

1. Add the stateful RandomWords widget to main.dart. The RandomWords widget does little else beside creating its State class:

lib/main.dart (RandomWords)

```
class RandomWords extends StatefulWidget {  
  @override  
  RandomWordsState createState() => RandomWordsState();  
}
```



TECMAN LIMITED 2019

## Step 3: Add a stateful widget

---

1. After adding the state class, the IDE complains that the class is missing a build method. Next, you'll add a basic build method that generates the word pairs by moving the word generation code from MyApp to RandomWordsState.

# Step 3: Add a stateful widget

---

1. Add the build() method to RandomWordsState:

lib/main.dart (RandomWordsState)

```
class RandomWordsState extends State<RandomWords> {  
  @override  
  Widget build(BuildContext context) {  
    final wordPair = WordPair.random();  
    return Text(wordPair.asPascalCase);  
  }  
}
```



TECMAN LIMITED 2019

# Step 3: Add a stateful widget

---

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

## Step 3:

```
{step2_use_package → step3_stateful_widget}/lib/main.dart

@@ -6,7 +6,6 @@
 6      6      class MyApp extends StatelessWidget {
 7      7      @override
 8      8      Widget build(BuildContext context) {
 9      -      final wordPair = WordPair.random();
10      9      return MaterialApp(
11      10         title: 'Welcome to Flutter',
12      11         home: Scaffold(
13      @@ -14,8 +13,8 @@
14      13             title: Text('Welcome to Flutter'),
15      14             ),
16      15             body: Center(
17      -             child: Text(wordPair.asPascalCase),
18      16      +             child: RandomWords(),
19      17             ),
20      18             ),
21      19             );
22      20         }
23      21     }
```

TECMAN LIMITED 2019

## Step 3: Add a stateful widget

---

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



# Step 4: Create an infinite scrolling ListView

---

1. 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 on demand.

# Step 4: Create an infinite scrolling ListView

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

```
class RandomWordsState extends State<RandomWords> {  
  final _suggestions = <WordPair>[];  
  final _biggerFont = const TextStyle(fontSize: 18.0);  
  // ...  
}
```



TECMAN LIMITED 2019

# Step 4: Create an infinite scrolling ListView

---

1. Next, you'll add a `_buildSuggestions()` function to the `RandomWordsState` class. This method builds the `ListView` that displays the suggested word pairing.
2. 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 grow infinitely as the user scrolls.

# Step 4: Create an infinite scrolling ListView

---

1. Add a `_buildSuggestions()` function to the `RandomWordsState` class:

## Step 4:

lib/main.dart (\_buildSuggestions)

```
Widget _buildSuggestions() {  
  return ListView.builder(  
    padding: const 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]);  
    }  
  ));  
}
```

TECMAN LIMITED 2019

# Step 4: Note

---

1. `/*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 may be difficult to see on smaller devices.
2. `/*2*/` Add a one-pixel-high divider widget before each row in the ListView.
3. `/*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. `/*4*/` If you've reached the end of the available word pairings, then generate 10 more and add them to the suggestions list.

# Step 4: Note

---

1. 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.

# Step 4: Create an infinite scrolling ListView

1. Add a `_buildRow()` function to `RandomWordsState`:

lib/main.dart (`_buildRow`)

```
Widget _buildRow(WordPair pair) {  
  return ListTile(  
    title: Text(  
      pair.asPascalCase,  
      style: _biggerFont,  
    ),  
  );  
}
```



TECMAN LIMITED 2019



# Step 4: Create an infinite scrolling ListView

---

1. In the RandomWordsState class, update the build() method to use `_buildSuggestions()`, rather than directly calling the word generation library. (Scaffold implements the basic Material Design visual layout.) Replace the method body with the highlighted code:

# Step 4: Create an infinite scrolling ListView

lib/main.dart (build)

```
@override
Widget build(BuildContext context) {
  return Scaffold(
    appBar: AppBar(
      title: Text('Startup Name Generator'),
    ),
    body: _buildSuggestions(),
  );
}
```



TECMAN LIMITED 2019

# Step 4: Create an infinite scrolling ListView

---

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

## Step 4:

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

```
@@ -6,15 +6,8 @@  
6      6      class MyApp extends StatelessWidget {  
7      7      @override  
8      8      Widget build(BuildContext context) {  
9      9      return MaterialApp(  
10     -      title: 'Welcome to Flutter',  
11     -      home: Scaffold(  
10     +      title: 'Startup Name Generator',  
11     +      home: RandomWords(),  
12     -      appBar: AppBar(  
13     -        title: Text('Welcome to Flutter'),  
14     -      ),  
15     -      body: Center(  
16     -        child: RandomWords(),  
17     -      ),  
18     -    ),  
19     12    );  
20     13    }
```

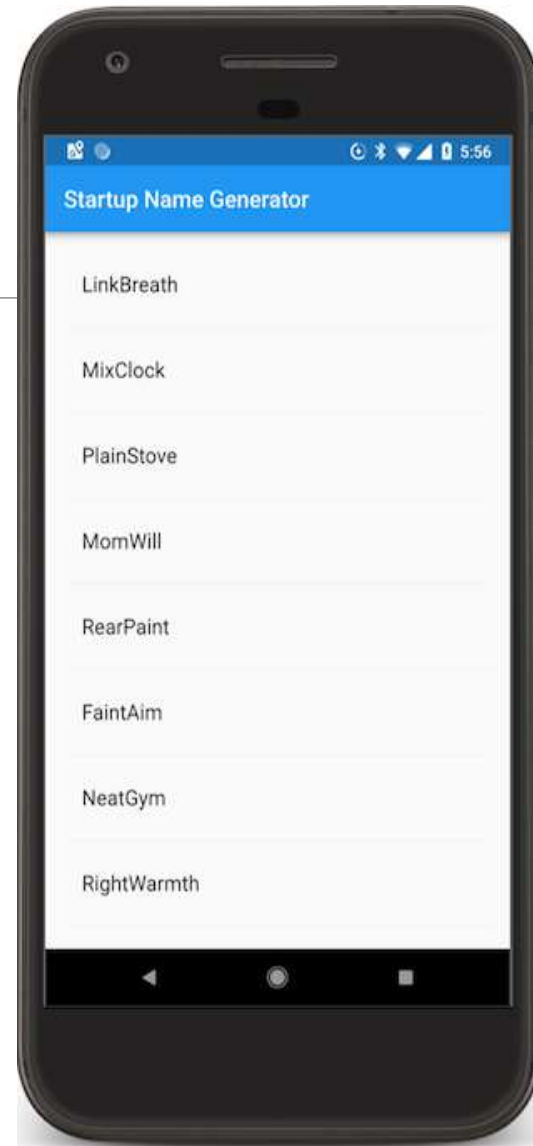
TECMAN LIMITED 2019

# Step 4: Create an infinite scrolling ListView

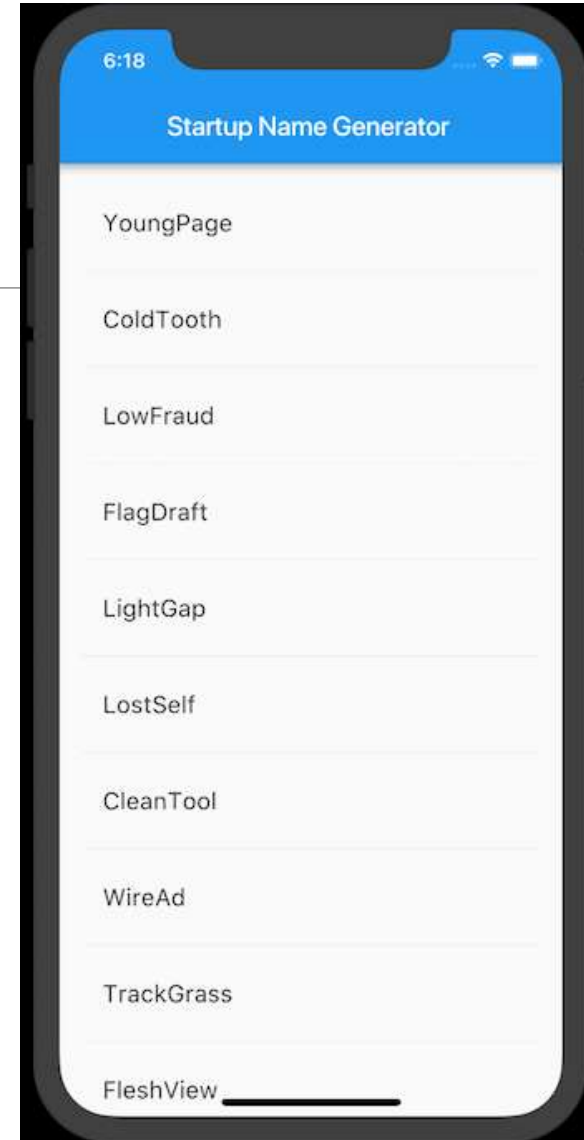
---

1. Restart the app. You should see a list of word pairings no matter how far you scroll.

## Step 4:



ANDROID



iOS

TECMAN LIMITED 2019

# Step 4: Create an infinite scrolling ListView

---

Congratulations!

1. You've written an interactive Flutter app that runs on both iOS and Android.  
In this codelab, you've:
  1. Created a Flutter app from the ground up.
  2. Written Dart code.
  3. Leveraged an external, third-party library.
  4. Used hot reload for a faster development cycle.
  5. Implemented a stateful widget.
  6. Created a lazily loaded, infinite scrolling list.

**TECMAN LIMITED 2019**

---

# THANK YOU

**TECMAN LIMITED 2019**