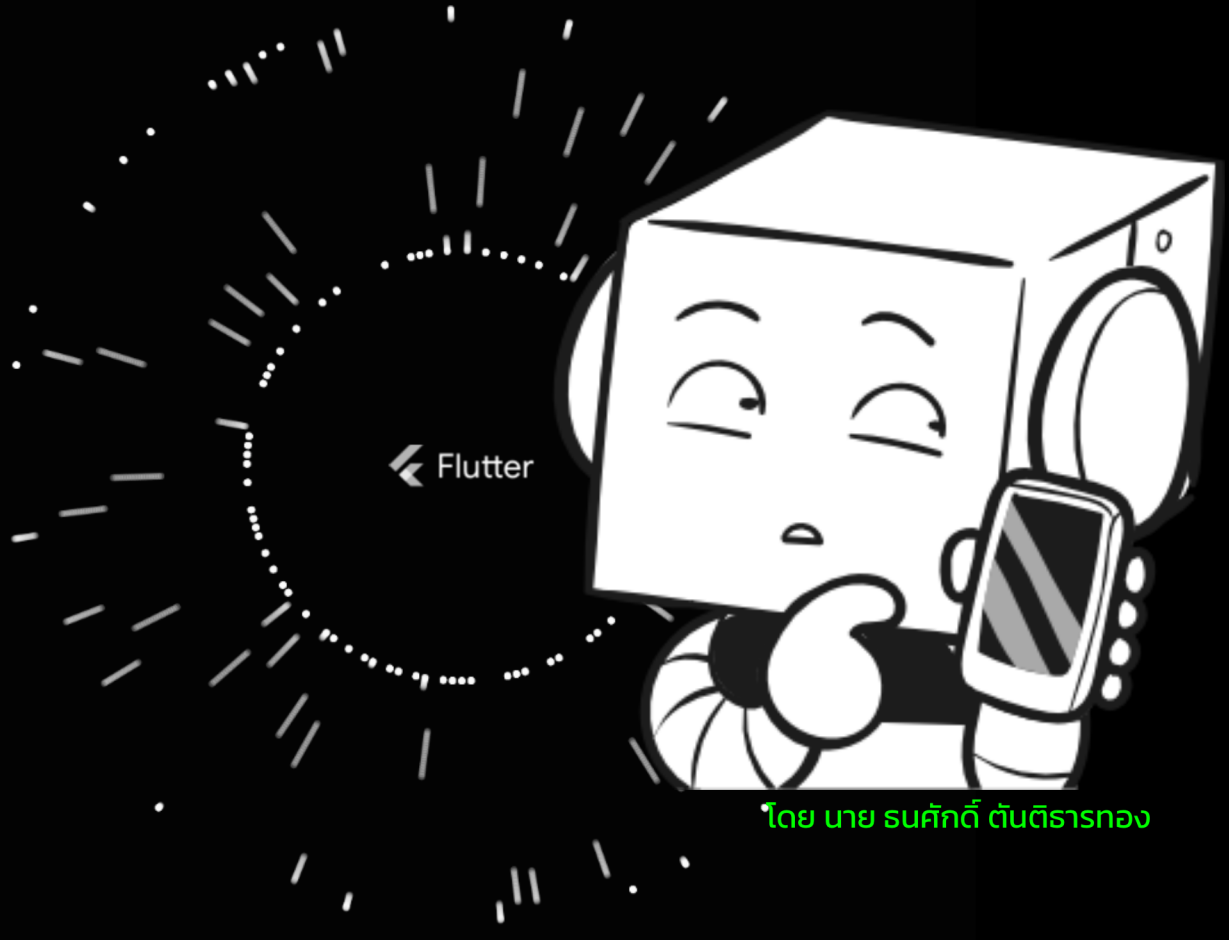
เกิดอยากจะเขียน APP

ง่ายๆ ด้วย Flutter

บันทึกการเรียนเขียน Flutter โดยผู้แต่ง
เริ่มจาก 0 เหมือนกัน



โดย นาย ธนศักดิ์ ตันติธารทอง

เกิดอยากจะเขียน App ด้วย Flutter

บันทึกการศึกษาเขียน Flutter ของผู้แต่งเอง

Tanasak Tantitarntong

This book is for sale at http://leanpub.com/soon-to-be-flutter-programmer

This version was published on 2021-09-17

Tweet This Book!

Please help Tanasak Tantitarntong by spreading the word about this book on Twitter!

The suggested hashtag for this book is #soontobeflutterprogrammer.

Find out what other people are saying about the book by clicking on this link to search for this hashtag on Twitter:

#soontobeflutterprogrammer

สารบัญ

1. เตรียมเครื่อง

วิธีการเตรียมเครื่องของผมนั้น ผมไม่ได้ทำตาม web official ของ

https://flutter.dev/docs/get-started/install

เพราะว่าผมเจอปัญหาบ่อยๆกับการต้อง manage binary หลายๆ version บนเครื่อง

ผมจึงค้นหาไปเจอ fvm หรือ flutter Version Manager

https://fvm.app/

1.1 สำหรับ Mac

เนื่องจากผมเป็น Mac User โดยตรง ผมจะละเอียดหน่อย

1. ก่อนอื่นผมขอแนะนำให้ลง Xcode จาก Mac App Store
- เมื่อลงเสร็จแล้วให้รัน

- `sudo xcode-select --switch /Applications/Xcode.app/Contents/Developer`
- `sudo xcodebuild -runFirstLaunch`
- ทดสอบ Simulator ด้วย `open -a Simulator`

2. ไปลง https://github.com/rvm/rvm เพื่อที่จะลง Ruby
3. ลง Ruby version 3.0.0 หรือล่าสุด เช่น `rvm install 3.0.0`

- ลง gem ชื่อว่า CocoaPods สำหรับใช้งานกับ iOS Development
- `gem install cocoapods`

4. เมื่อลง rvm และ ruby เสร็จ เราจะได้ brew มาใช้งานด้วย

5. รัน `brew tap AdoptOpenJDK/openjdk`
6. รัน `brew cask install adoptopenjdk8` เพื่อลง Java Development Kit
7. Install Android Studio https://developer.android.com/studio

เมื่อทำเสร็จแล้ว ค่อยมาลง fvm ต่อ ดังนี้

```
brew tap leoafarias/fvm
brew install fvm
```

1.2 สำหรับคนใช้ windows

ผมไม่ได้ใช้ Windows เขียนโปรแกรมเป็น 10 ปีแล้ว แต่เท่าที่อ่านมาคือ
ให้ใช้ powershell รัน command นี้ครับ

```
choco install fvm
```

1.3 วิธีใช้ fvm

สำหรับคนที่คุ้นเคยกับ rvm nvm pyenv นั้น วิธีการใช้ fvm แทบจะไม่แตกต่างเลยครับ

fvm releases

เป็นคำสั่งที่จะ List versions ของ flutter ที่เราสามารถลงได้ ออกมา

จะได้ output ตัวอย่างดังนี้

```
Mar 3 21    | 2.0.0
Mar 3 21    | 2.1.0-10.0.pre
Mar 3 21    | 2.0.0
Mar 4 21    | 2.0.1
Mar 4 21    | 2.0.1
Mar 12 21   | 2.0.2
Mar 13 21   | 2.1.0-12.1.pre
Mar 15 21   | 2.0.2
Mar 18 21   | 2.1.0-12.2.pre
Mar 19 21   | 2.0.3
Apr 2 21    | 2.0.4
Apr 15 21   | 2.2.0-10.1.pre
Apr 15 21   | 2.2.0-10.1.pre
Apr 16 21   | 2.0.5
Apr 27 21   | 2.3.0-0.1.pre
Apr 29 21   | 2.2.0-10.2.pre
Apr 30 21   | 2.0.6
May 10 21   | 2.2.0-10.3.pre
May 10 21   | 2.3.0-1.0.pre
-------------------------------------
May 18 21   | 2.2.0             stable
-------------------------------------
-------------------------------------
May 18 21   | 2.3.0-12.1.pre    dev
-------------------------------------
-------------------------------------
May 19 21   | 2.2.0             beta
-------------------------------------
```

fvm install

เป็นคำสั่งที่จะลง flutter version ที่เราต้องการ เช่น

```
fvm install 2.2.0
```

ก็จะเป็นการลง flutter version 2.2.0 ลงในเครื่องเราทันที

เราจะลง Version อื่นอีกก็ได้ เช่น

```
fvm install 2.0.6
```

fvm use

หากเราเปิด project ที่เป็น flutter อยู่ จะทำให้เราสามารถสั่งให้ใช้ flutter version ที่เราต้องการได้เช่น ถ้าเราเปิด project ที่ใช้ flutter 2.0.6 อยู่ เราต้องรัน

```
fvm use 2.0.6
```

เพื่อที่จะสั่งให้ flutter มาใช้ version 2.0.6 ที่เราต้องการ

fvm use -f

เป็นการบังคับใช้ version ที่เราต้องการเช่น

```
fvm use 2.2.0 -f
```

จะทำให้ flutter กลายเป็น version 2.2.0 โดยถูกบังคับใช้

fvm global

ใช้เหมือนกับ `fvm use` แต่ว่า เป็นการตั้งค่านี้เป็น global ทำให้ทุกๆที่ที่เราเรียก command จะกลายเป็น version นี้ทั้งหมด

fvm flutter

คำสั่งนี้เกิดมาเพื่อ PROXY คำสั่ง flutter มายัง fvm

เช่น

```
flutter --version
```

จะเป็นการรัน flutter version ที่เราตั้งค่าไว้เป็น global

แต่ว่าถ้าเรารัน

```
fvm flutter --version
```

จะกลายเป็น flutter version ที่เราสั่ง `fvm use` เอาไว้

1.4 Text Editor

จากเวป

https://flutter.dev/docs/get-started/editor?tab=vscode

เราจะเห็นได้ว่า มีไม่กี่ options ของ text editor ที่ flutter แนะนำ

ส่วนตัวแล้ว ทุกอันที่ List อยู่นี้ ผมไม่ชอบเลย ผมชอบ Sublime กับ Atom มากกว่า

แต่เพื่อทำให้เราเขียนโปรแกรมได้สะดวก เราก็ควรจะใช้สิ่งที่เค้าแนะนำ

ผมขอเลือก VSCode ครับ โดยสามารถ Download ได้ที่นี่

https://code.visualstudio.com/

หรือลงผ่าน brew ก็ได้ ง่ายๆ

```
brew install --cask visual-studio-code
```

เมื่อลงโปรแกรมเสร็จแล้ว ให้เปิด VSCode ขึ้นมาและเปิด Command Palette

กด View > Command Palette หรือ กด cmd+shift+p

และพิมคำว่า Extensions: Install Extensions แล้วกด Enter

ต่อมาให้พิมค้นหาคำว่า Flutter และ กด Install

เมื่อลง Flutter ใน VSCode เสร็จแล้ว ให้สั่งคำสั่ง

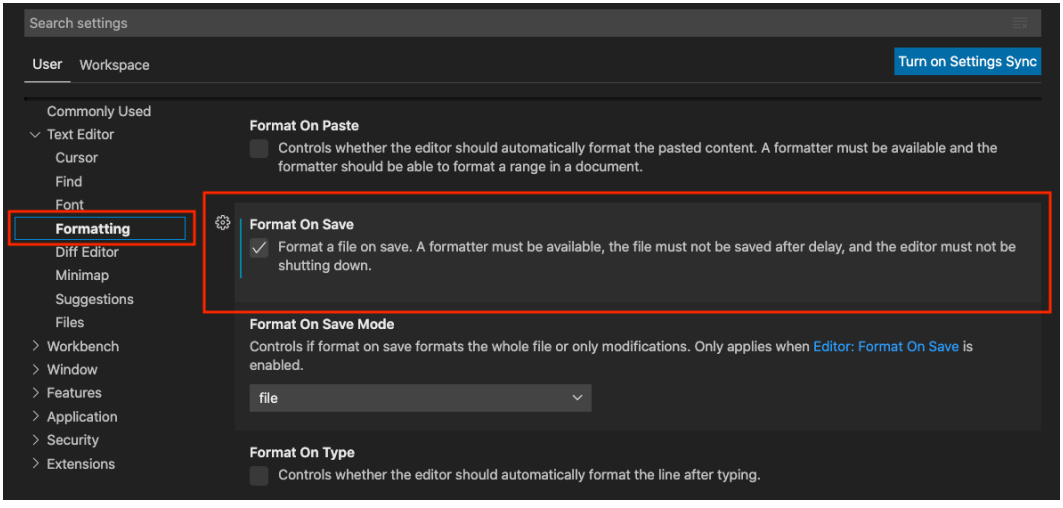
Flutter: Run Flutter Doctor.

ใน Command Palette อีกครั้ง และรอดู output ใน VSCode ว่าเครื่องเราขาดอะไรบ้าง

```
[flutter] flutter doctor -v
[✓] Flutter (Channel unknown, 2.2.0, on Mac OS X 10.15.7 19H2 darwin-x64, locale en)
    • Flutter version 2.2.0 at /Users/sakko/fvm/versions/2.2.0
    • Framework revision b22742018b (6 days ago), 2021-05-14 19:12:57 -0700
    • Engine revision a9d88a4d18
    • Dart version 2.13.0

[✓] Android toolchain - develop for Android devices (Android SDK version 29.0.2)
    • Android SDK at /Users/sakko/Library/Android/sdk
    • Platform android-29, build-tools 29.0.2
    • ANDROID_HOME = /Users/sakko/Library/Android/sdk
    • Java binary at: /Applications/Android Studio.app/Contents/jre/jdk/Contents/Home/bin\
/java
    • Java version OpenJDK Runtime Environment (build 1.8.0_202-release-1483-b49-5587405)
    • All Android licenses accepted.

[✓] Xcode - develop for iOS and macOS
    • Xcode at /Applications/Xcode.app/Contents/Developer
    • Xcode 12.4, Build version 12D4e
    • CocoaPods version 1.10.1

[✓] Chrome - develop for the web
    • Chrome at /Applications/Google Chrome.app/Contents/MacOS/Google Chrome

[✓] Android Studio (version 3.5)
    • Android Studio at /Applications/Android Studio.app/Contents
    • Flutter plugin can be installed from:
      ✗ https://plugins.jetbrains.com/plugin/9212-flutter
    • Dart plugin can be installed from:
      ✗ https://plugins.jetbrains.com/plugin/6351-dart
    • Java version OpenJDK Runtime Environment (build 1.8.0_202-release-1483-b49-5587405)

[✓] VS Code (version 1.56.2)
    • VS Code at /Applications/Visual Studio Code.app/Contents
    • Flutter extension version 3.22.0

[✓] Connected device (1 available)
    • Chrome (web) • chrome • web-javascript • Google Chrome 90.0.4430.212
```

```
39
40  • No issues found!
41  exit code 0
```

สุดท้ายสำหรับ VSCode ไปเปิด Format on Save เพื่อจะทำให้โค้ดอ่านง่ายขึ้น



format on save

โดยพื้นฐานแล้ว ควรจะเป็น • No issues found! ครับ
หากติดอะไร ให้อ่านแล้วแก้ไขครับ

2. สร้าง Project แรก

จากการลง flutter ใน chapter ที่แล้ว ตอนนี้เราจะมาเริ่มสร้าง Project แรกของเรากัน

ก่อนอื่นผมจะเริ่มจากการสร้าง `_workspace` สำหรับเก็บ code ของผม

'หากใครมี folder อยู่แล้ว ข้ามส่วนนี้ไปได้เลย'

```
cd ~/
mkdir _workspace
cd _workspace
mkdir flutter
cd flutter
```

เมื่ออยู่ใน folder ที่ต้องการสร้าง project แล้ว เราจะรันคำสั่ง

```
flutter create demo_flutter_app
```

ให้สังเกตุ Log ด้วยว่าสร้างสำเร็จไหม

หากไม่มีปัญหา เราจะได้ folder ใหม่ขึ้นมาชื่อว่า `demo_flutter_app` เข้าไปใน folder เพื่อเตรียมพร้อม

```
cd demo_flutter_app
```

2.1 Running Your Application

ก่อนที่เราจะรันโปรแกรม ให้เราเปิด Simulator ขึ้นมาก่อน

```
open -a Simulator
```

เมื่อ Simulator พร้อมแล้ว ให้รัน

```
flutter devices
```

ให้สังเกตุว่าจะมี devices แสดงมาสองอัน เช่น

```
iPhone 12 Pro Max (mobile) • xxx-xx-xx • ios            • com.apple...
Chrome (web)               • chrome     • web-javascript • Google Chrome...
```

นั่นหมายความว่าเราสามารถรัน Flutter ไปยังที่ใดก็ได้ เช่น

```
flutter run -d chrome
```

จะเป็นการรันไปยัง chrome โดยที่เราจะเห็น chrome เด้งขึ้นมาเอง

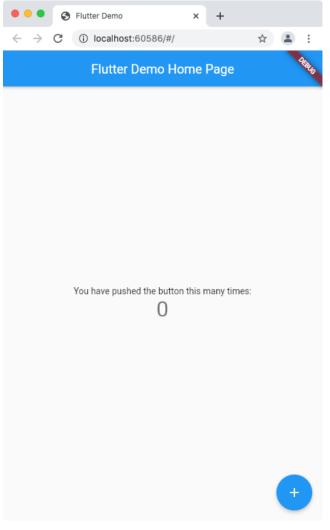
แต่ว่าถ้ารัน

```
flutter run -d xxx-xx-xx
```

ซึ่ง `xxx-xx-xx` เป็น id ของ simulator ก็จะกลายเป็นรันโปรแกรมบน iOS Simulator ที่เราเปิดอยู่

iOS

Chrome



preview app

หากเรากด `ctrl+c` ที่ terminal ก็จะเป็นการ Stop application
ให้เราเข้าไป Stop app ให้หมดก่อนที่เราจะไป section ต่อไป

2.2 ใช้ VSCode เพื่อทำการ Debug

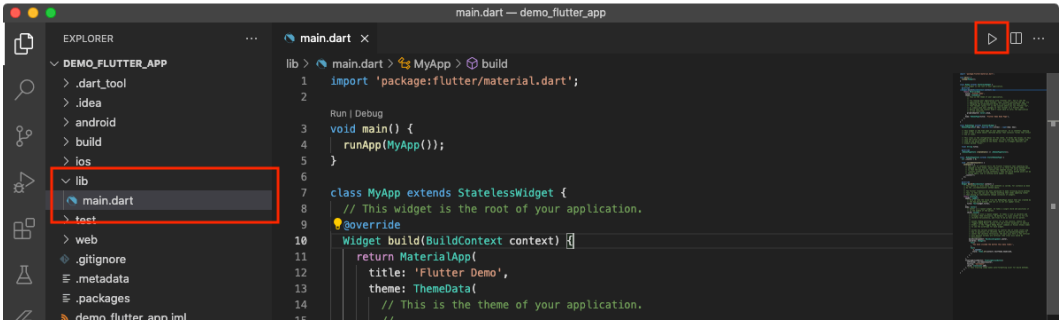
หลังจากนี้ไป ผมจะเริ่มทดสอบผ่าน Chrome และ เป็นครั้งคราว จะไปรันผ่าน Simulator

ที่ผมเลือก Chrome เพราะว่า Chrome น่าจะ Lightweight สุดครับ

ก่อนอื่นให้ เปิด Project ของเราขึ้นมาด้วย VSCode และ เปิด File ที่ชื่อว่า

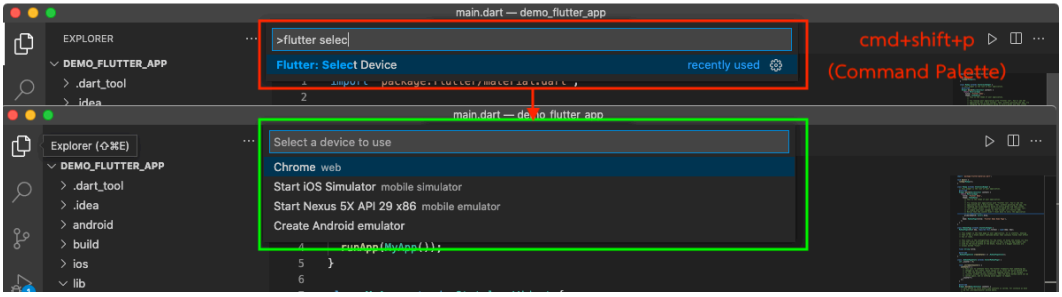
`lib/main.dart`

และกดปุ่ม Start Debugging ด้านบนขวา ตามใน Screenshot



start debugging

> เราสามารถใช้คำสั่งใน Command Palette สั่งให้รันใน Device อื่นได้โดยการพิมคำว่า
> `Flutter: Select Device` และ เลือก Device ที่ต้องการ ดังรูป
> ให้เราเลือกเป็น Chrome ไว้ก่อน



start debugging

เมื่อกด Start Debuging แล้ว จะได้ Chrome ขึ้นมาหนึ่ง Session ไว้ให้เราทดสอบ

2.3 Hot reload

ให้เลื่อนลงมาด้านล่างของ `lib/main.dart` เพื่อแก้คำว่า

```
'You have clicked the button this many times:',
```

ให้กลายเป็น

```
'You have pushed the button this many times:',
```

แล้วลองกด save (cmd+s) ดู แล้วสังเกตว่า Chrome นั้นทำการ Hot reload หลังเรากด Save หรือไม่

Hot reload จริงๆแล้วไม่ใช่เรื่องใหม่ มันคือการที่ทำให้ App ของเรา Reload ทุกครั้งที่มีการแก้ไข Code แล้ว save

สำหรับผม ปัญหาที่ผมเจอมาหลายๆครั้งกับ Hot reload คือ บางที Save บ่อยไปจนทำให้ Hot reload ไม่ทัน หากเจอบัคที่บางทีไม่เข้าใจ ลอง Restart Debugging ก็อาจจะช่วยได้ครับ

2.4 เริ่มแก้ไข Code

สำหรับผม ปัญหาของการเรียนรู้เรื่องใหม่ๆคือ มันช่างมีอะไรเยอะแยะไปหมดเหลือเกิน

เรามาเริ่มจากสิ่งที่ Flutter.dev แนะนำน่าจะดีที่สุด ดังนี้ครับ

นำ code ด้านล่างไปแปะไว้ที่ lib/main.dart

```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';

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

หน้าจอควรจะเปลี่ยนไปเป็น Hello world

- ก่อนอื่น ตัวอย่างนี้ ใช้ Material Design ของ Flutter ซึ่งมีการ Config `uses-material-design: true` ไว้ที่ `pubspec.yaml` ซึ่งจะช่วยให้เราสามารถใช้ Feature ต่างๆของ Material Design ได้
- `main() =>` เป็น Arrow Function ใช้สำหรับการรัน method ใน 1 บรรทัด
- `MyApp` เป็นการ extends StatelessWidget หรือการที่เราจะ Inherit ค่าต่างๆจาก สิ่งที่เรา extends ตาม concept ของ OOP.
- ส่วนอื่นๆผมว่ายังข้ามไปก่อนได้ เดี๋ยวค่อยอธิบายเพิ่มเติม

2.5 Adding External Package

สำหรับผมแล้ว มันเป็นได้ยากมากที่เราจะเขียนทั้งแอพโดยไม่พึ่ง Library อะไรเลย
ซึ่งในตัวอย่างก็มีการ Import `english_words` ให้ดู และเราจะมาลองทำกัน

ให้เปิด File ชื่อว่า `pubspec.yaml` เพื่อเพิ่ม `english_words: ^4.0.0-0`

โดยให้เพิ่มไปตรงที่อยู่ใต้ `dependencies` ดังนี้

```
dependencies:
  flutter:
    sdk: flutter
  english_words: ^4.0.0-0
```

เมื่อเพิ่มเสร็จแล้ว เราจะต้องให้ flutter ไปดึง package มาโดยรันคำสั่ง

```
flutter pub get
```

การรันคำสั่งนี้ จะทำให้ flutter ลง package ตามที่อยู่ใน `pubspec.yaml` และ `pubspec.lock` โดยที่จะอ้างอิง Version ของ Package ต่างๆใน `pubspec.lock` สำหรับสิ่งที่เคยลงไว้แล้วใน Project และ ทำการลง Package ใหม่ที่ถูกเพิ่มใน `pubspec.yaml`

เมื่อทุกอย่างลงเสร็จ `pubspec.lock` จะถูกแก้ไขให้มีชื่อ Package ใหม่ พร้อมกับ Version ที่ถูกลงตอนที่เรา Add Package นี้ไว้ด้วย

กลับมาที่ ‘lib/main.dart’

เราจะแก้ไข Code ดังนี้

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

เพิ่มบรรทัด Import english_words และ แก้ไข body ให้ใช้ WordPair.random() มาแสดงแทน

หากทดลองรัน Code จะเห็นว่า Body ของ App เราจะเปลี่ยน Text ไปเรื่อยๆ

2.6 Stateless vs Stateful Widget

ที่เราทำการเขียนมาทั้งหมดนั้น เป็น Stateless Widget ซึ่ง ความหมายของ Stateless Widget คือ

> Stateless Widget ทุกอย่างจะไม่สามารถเปลี่ยนค่าใดๆได้ (Immutable)

ซึ่งการเขียนโปรแกรมนั้น ก็มี Stateful ด้วยเช่นกัน

> Stateful Widget สามารถบันทึกการเปลี่ยนแปลงบางสิ่งบางอย่างได้ตลอดช่วงอายุของ Widget นั้นๆใน Application ของเรา โดยที่ Stateful Widget จำเป็นต้องใช้ StatefulWidget class และ State class

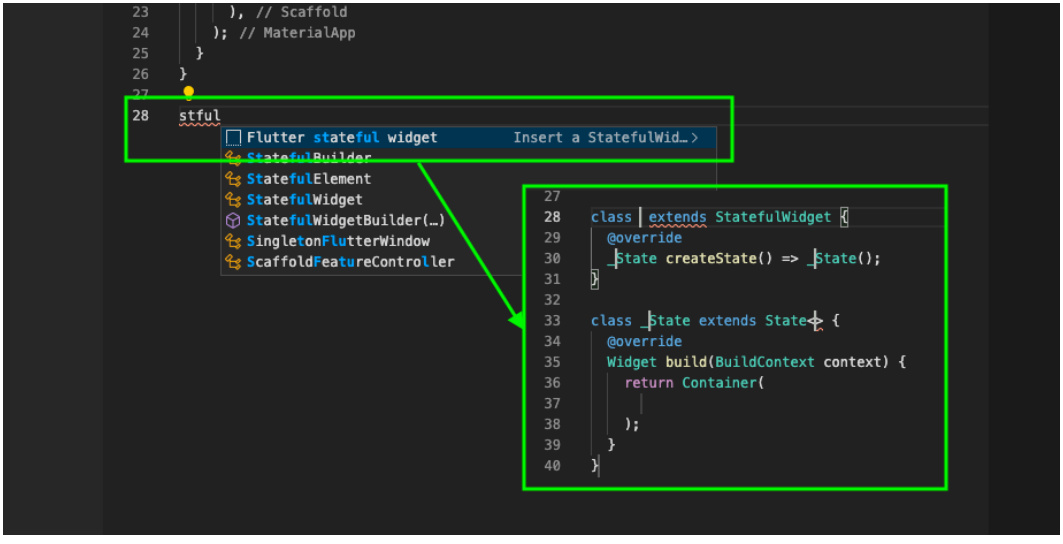
- StatefulWidget class - เป็น Widget ที่ใช้สร้าง Instance ของ Widget Class
- StatefulWidget class นั้นเป็น Immutable เช่นกัน แต่ว่าสามารถใช้ร่วมกับ State ซึ่งเป็น Mutable และ สามารถถูกนำมาใช้งานซ้ำได้ตลอดช่วงอายุของ StatefulWidget

2.7 Stateful Widget

ใน ตัวอย่าง ต่อ มา เรา จะ สร้าง StatefulWidget ชื่อ ว่า RandomWords ที่ มี State ชื่อ ว่า _RandomWordsState

และเราจะนำ RandomWords ไปใช้กับ MyApp ซึ่งเป็น Stateless Widget

ให้ไปด้านล่างสุดของ file `lib/main.dart` และพิมคำว่า stful แล้วกด Enter ซึ่ง VSCode จะทำ Auto Complete ให้ดังรูป



start debugging

ให้สังเกตว่า Cursor จะกระพริบใน VSCode หลายๆจุดพร้อมกัน ซึ่งสิ่งนี้จะทำให้เราสามารถพิมตัวหนังสือได้หลายที่พร้อมกัน ให้ลองพิมคำว่า RandomWords จะทำให้ได้ Code ดังนี้

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

class _RandomWordsState extends State<RandomWords> {
  @override
  Widget build(BuildContext context) {
    return Container(

    );
  }
}
```

ให้แก้ไข _RandomWordsState ให้ return ค่าดังนี้

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

ข้อสังเกตุ State จะขึ้นด้วย _ อย่าลืมใช้ให้ถูกต้อง

กลับไปยัง MyApp และ แก้ไขดังนี้

```dart
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'),
        ), // AppBar
        body: Center(
          // child: Text(wordPair.asPascalCase),
          child: RandomWords(),
        ), // Center
      ), // Scaffold
    ); // MaterialApp
  }
}
```

start debugging

เมื่อทดลอง restart application จะเห็นว่าผลลัพท์ได้เหมือนเดิม แต่ว่าเดี๋ยวเราจะมาดูกันว่ามันแตกต่างกันอย่างไรทีหลัง

2.8 Listview and Stateful Widget

เพื่อทำการทดสอบ Stateful Widget เราจะทำการใช้ Listview เพื่อดูว่า _RandomWordsState นั้นเป็น mutable (เปลี่ยนแปลงได้) ไม่เหมือน Stateless ซึ่งเป็น Immutable

ก่อนอื่น ผมจะเริ่มที่ class `_RandomWordsState`

เราจะแก้ไข `build` ให้ Return AppBar ออกมา กับ body ซึ่งจะเป็น Listview ดังนี้

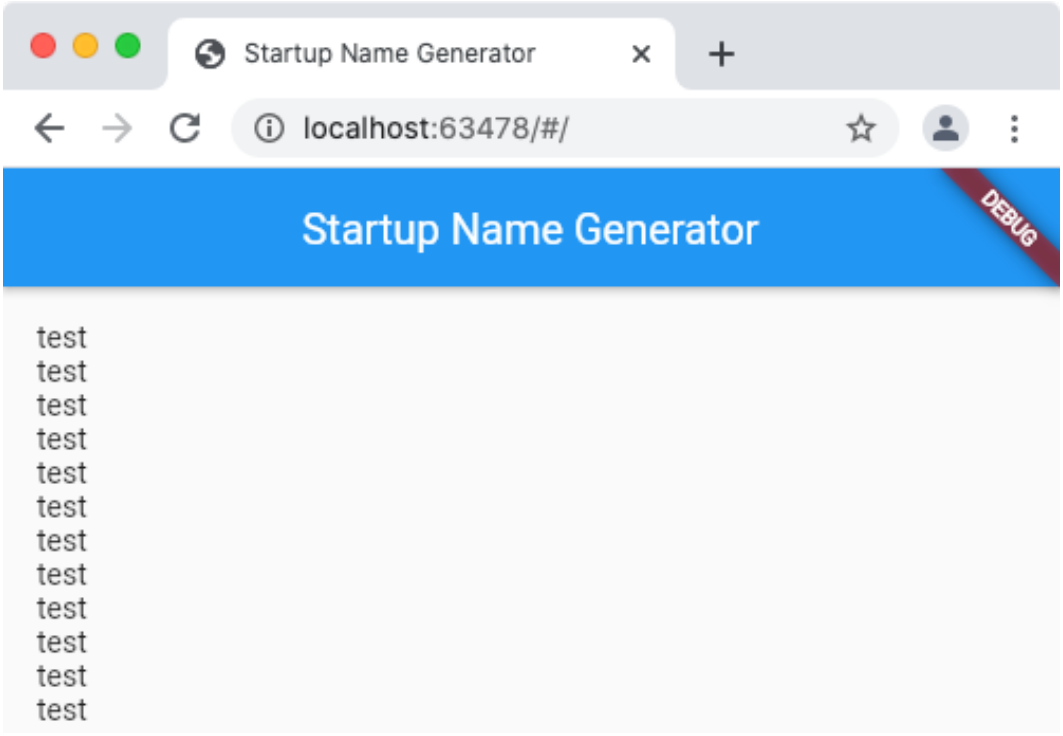
```
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: Text('Startup Name Generator'),
      ),
      body: _buildWordsListView(),
    );
  }
```

ต่อมาใน class `_RandomWordsState` เราก็จะเพิ่ม _buildWordsListView ซึ่งจะ Return Widget ที่เป็น ListView กลับไป

```
  Widget _buildWordsListView() {
    return ListView.builder(
        padding: EdgeInsets.all(16.0),
        itemBuilder: (context, i) {
          return Text("test")
        });
  }
```

ใน ListView เรา return แค่ text “test” ออกมา โดยที่อ้างอิงจาก `itemBuilder: (context, i)` ซึ่ง `i` จะเป็น index ที่ ListView พยายามจะแสดงค่า

การที่เรา return Text(“test”) ออกมาตลอดไม่ว่า `i` จะเป็นอะไรก็ตาม ผลลัพท์ที่ได้จะเป็นแบบนี้



ต่อมาเราจะเพิ่ม State ให้กับ class _RandomWordsState โดยเพิ่มไว้ที่นี่

```
class _RandomWordsState extends State<RandomWords> {
  final _words = <WordPair>[];

  ...
  ...
  ...
}
```

`_words` - คือ Array ที่เราจะเก็บ WordPair เอาไว้ และ `WordPair` เป็น Class ของ english_words lib

แก้ไข _buildWordsListView ดังนี้ เพื่อให้ ใช้ data จาก _words state

```
  Widget _buildWordsListView() {
    return ListView.builder(
        padding: EdgeInsets.all(16.0),
        itemCount: _words.length,
        itemBuilder: (context, i) {
          return Text(_words[i].asPascalCase);
        });
  }
```

สิ่งที่เกิดขึ้นใน function นี้คือ

- `itemCount` เป็นตัวบอกว่า ListView นี้มี Data ทั้งหมดกี่ Rows ซึ่งจะเป็นตัว Link ไปบอก `itemBuilder` ด้วยว่า `i` จะสิ้นสุดที่ index ที่เท่าไหร่
- นำ `_words[i]` (ตำแหน่งที่ `i`) return ออกไปใน Text
- ตอนนี้ `_words` ยังไม่มีค่าอะไร จึงเป็น Empty List

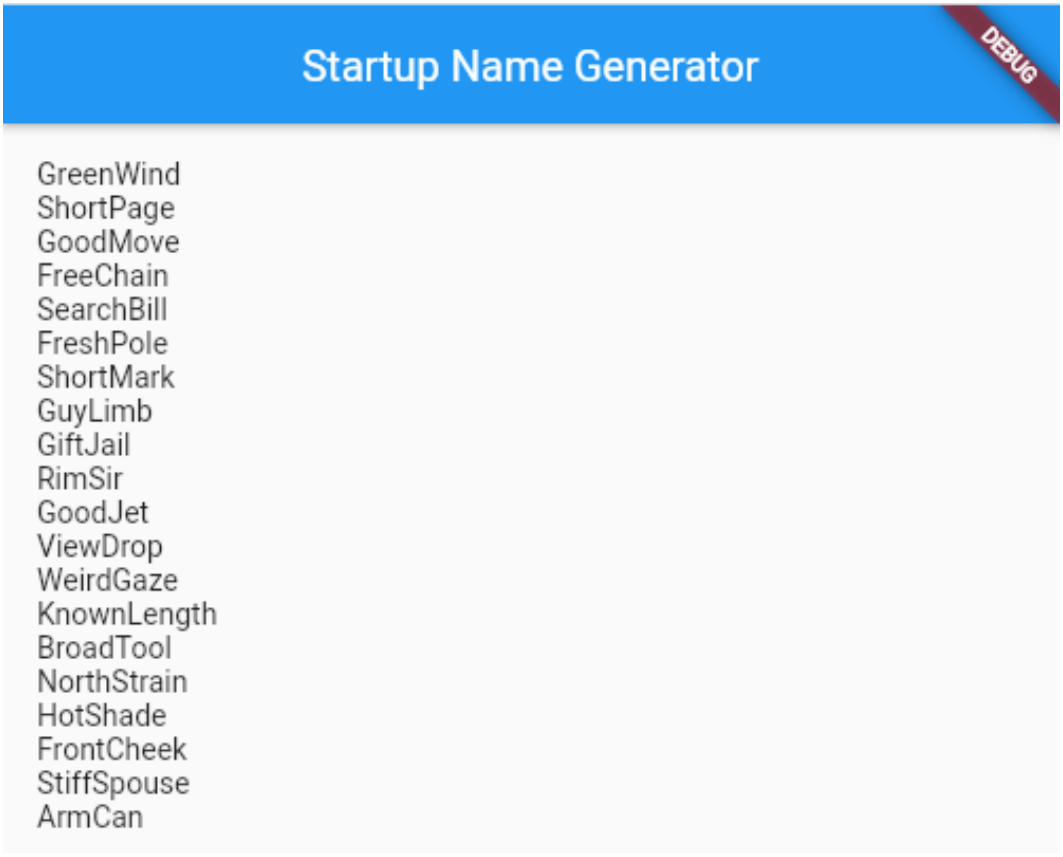
ต่อมาเราจะไปเพิ่ม `_words` กัน ดังนี้

ใน class _RandomWordsState ที่ `build` Widget ให้เพิ่ม Code เข้าไป 1 บรรทัด

```
  Widget build(BuildContext context) {
    _words.addAll(generateWordPairs().take(20));
    return Scaffold(
      ...
      ...
      ...
    );
  }
```

ซึ่งจะเพิ่ม Words มาทั้งหมด 20 words เข้าไปใน `_words` Array

จะได้ผลลัพท์ ดังนี้



2.9 Divider ใน ListView

ต่อมา เราจะทำการเพิ่ม Divider เข้าไปใน ListView จะได้มีเส้นคั่นระหว่าง Rows

ให้แก้ `ListView.builder` เป็น `ListView.separated` และเพิ่ม option `separatorBuilder` เข้าไปดังนี้

```
  Widget _buildWordsListView() {
    return ListView.separated(
        padding: EdgeInsets.all(16.0),
        itemCount: _words.length,
        itemBuilder: (context, i) {
          return Text(_words[i].asPascalCase);
        },
        separatorBuilder: (context, i) {
          return Divider();
        });
    }
```

`separatorBuilder` มีหน้าที่ return `Divider` Widget ออกมาเพื่อคั่นระหว่าง Row เพื่อความเข้าใจ ลองแก้ `Divider()` เป็น `Text("_____________")` ดูก็ได้

3. Dart

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/soon-to-be-flutter-programmer.

3.1 Overview

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/soon-to-be-flutter-programmer.

Dart Libraries

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/soon-to-be-flutter-programmer.

3.2 Platforms

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/soon-to-be-flutter-programmer.

3.3 Samples

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/soon-to-be-flutter-programmer.

Hello world

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/soon-to-be-flutter-programmer.

3.4 Variables

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/soon-to-be-flutter-programmer.

3.5 Variable Null Safety

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/soon-to-be-flutter-programmer.

if else

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/soon-to-be-flutter-programmer.

for in

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/soon-to-be-flutter-programmer.

for loop

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/soon-to-be-flutter-programmer.

while loop

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/soon-to-be-flutter-programmer.

3.6 Functions

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/soon-to-be-flutter-programmer.

3.7 Comments

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/soon-to-be-flutter-programmer.

3.8 Import

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/soon-to-be-flutter-programmer.

3.9 Printing

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/soon-to-be-flutter-programmer.

3.10 Assert

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/soon-to-be-flutter-programmer.

4. Dart Class

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/soon-to-be-flutter-programmer.

4.1 Class Members

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/soon-to-be-flutter-programmer.

4.2 Constructor

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/soon-to-be-flutter-programmer.

4.3 null

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/soon-to-be-flutter-programmer.

4.4 final

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/soon-to-be-flutter-programmer.

4.5 fromJson

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/soon-to-be-flutter-programmer.

4.6 Named Constructor

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/soon-to-be-flutter-programmer.

4.7 Subclassing Constructor

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/soon-to-be-flutter-programmer.

4.8 Class Variable

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/soon-to-be-flutter-programmer.

4.9 Constant constructor

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/soon-to-be-flutter-programmer.

4.10 Methods in class

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/soon-to-be-flutter-programmer.

4.11 Operators

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/soon-to-be-flutter-programmer.

4.12 Getter Setters

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/soon-to-be-flutter-programmer.

4.13 Abstract Class

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/soon-to-be-flutter-programmer.

4.14 Implicit Interface

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/soon-to-be-flutter-programmer.

4.15 overriding

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/soon-to-be-flutter-programmer.

4.16 Mixins

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/soon-to-be-flutter-programmer.

4.17 Summary

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/soon-to-be-flutter-programmer.

5. Dart ก่อนจะกลับไป Flutter

วกออกไปดู Class ตั้งนาน ขอย้อนกลับมาส่วนสำคัญของ Dart อีกนิดหน่อยก่อนจะกลับไปยัง Flutter กันครับ

5.1 Future

บางทีเราจะทำอะไรก็ตาม มันไม่เสร็จทันที มันต้องรอ มันก็เลยเกิดสิ่งที่ขึ้นมาทดแทน concept นี้

ใน Dart เรียกว่า Future ใน JS เรียกว่า Promise

`Future`

เปรียบเสมือน Promise ใน javascript สำหรับคนที่ไม่เคยเขียนเลยก็คือ มันเป็นสิ่งที่อาจจะยังทำไม่เสร็จ แต่ว่าเมื่อเสร็จ จะสามารถแยกออกมาทำต่อได้ดังนี้

`.then`

คือการบอกว่า เมื่อ Future ทำเสร็จ ให้มาทำสิ่งนี้ต่อ

`.catchError`

หากมีปัญหาอะไรใน Future ให้มาแก้ปัญหาในนี้

ตัวอย่างเช่น

```dart
Future<void> fetchUserOrder() {
  return Future.delayed(
      Duration(seconds: 2), () => print('Order done: Large Latte'));
}

void main() {
  print("program started");
  fetchUserOrder();
  print("program should end");
}
```

หากเรารัน โปรแกรมนี้ เราจะได้ log ดังนี้

```
program started
program should end
Order done: Large Latte
```

ให้สังเกตว่า `Order done: Large Latte` นั้น print ช้ากว่า `program should end` ตามที่เราตั้งไว้ 2 วินาที เคสนี้เกิดได้ในชีวิตจริงบ่อยมากเช่น ถ้าเราต้องไป query database แล้วต้องรอซัก 0.5s หรือว่าไปเรียก API แล้วต้องรอ 1s

5.2 Async Await

หากเราต้องการให้ `program should end` นั้น รันเป็นบรรทัดสุดท้าย เราจะทำแบบนี้

```dart
Future<void> fetchUserOrder() {
  return Future.delayed(
      Duration(seconds: 2), () => print('Order done: Large Latte'));
}

void main() async {
  print("program started");
  await fetchUserOrder();
  print("program should end");
}
```

การที่เราใช้ `await` กับ fetchUserOrder ได้ก็เพราะว่า return type ของ fetchUserOrder นั้นเป็น Future Object. และมีการใส่คำว่า async ไว้ใน function `main()` แล้วด้วย

ในการใช้ Future นั้น เราใช้ `.then` ก็ได้ แต่ว่า การใช้ `.then` นั้นก็อาจจะทำให้เราเจอปัญหา Callback hell เหมือนใน Javascript

```dart
const oneSecond = Duration(seconds: 10);

Future<void> printWithDelay(String message) {
  return Future.delayed(oneSecond).then((_) {
    print(message);
  });
}
```

หากเราไม่อยากใช้ `.then` ก็ใช้ แบบนี้ได้

```dart
const oneSecond = Duration(seconds: 10);

Future<void> printWithDelay(String message) async {
  await Future.delayed(oneSecond);
  print(message);
}
```

เป็นการเพิ่ม `async` และ `await` เข้าไปใน code เหมือน javascript เลย

5.3 Exceptions

เมื่อเราเจออะไรก็ตามที่เราสังหรใจไว้ว่า อาจจะมีความผิดพลาดเกิดขึ้นใน เราอาจจะต้องพึ่งพาการ throw error ออกมาเพื่อให้ code ส่วนๆอื่นๆทำงานได้ถูกต้อง เช่น

```
void launchShip(List<String> astronauts) {
  if (astronauts.length == 0) {
    throw StateError('No astronauts.');
  }
}
```

ถ้าเราลองให้ `launchShip([]);` โดยการส่ง Array เปล่าๆเข้าไป ก็จะเห็นได้ว่ามีการ throw error ออกมา และแอพจะ crash เลย

ซึ่งถ้าเรารู้อยู่แล้วว่าอาจจะเกิดปัญหานี้ เราก็อาจจะต้อง catch error ด้วย

```
try {
  launchShip([]);
} on StateError catch (e) {
  print(e);
}
```

ซึ่งจะทำให้ Application ของเราไม่ crash เพราะว่าเรา catch error ไว้แล้ว

สำหรับ error type อื่นๆ และการใช้ให้ถูกต้อง ดูได้ที่

https://api.dart.dev/stable/1.10.1/dart-core/Error-class.html

ตอนนี้ขอข้ามไปก่อน เริ่มเบื่อกับ Dart ละครับ

6. Widget

หลังจากออกไป Dart ซะนาน เรากลับมาที่ Widget ใน Flutter กันครับ

Widget ถ้าสังเกตดีๆ มันได้รับแรงบันดาลใจมาจาก React คล้ายๆกับเป็น Component หรือ View เดี๋ยวเรามาดูกันว่าใช้อย่างไรบ้าง

6.1 Real Hello, world

เริ่มจากการเขียน Project ใหม่ดังนี้

```
flutter create hello_world
cd hello_world
code .  # open project in VSCode
```

ใน VSCode กด Start Debugging

แก้ lib/main.dart ดังนี้

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

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

สังเกตได้ว่า เราจะเห็นแค่ text ชื่อว่า Hello, world เปล่าๆ ซึ่งเกิดจาก Center และ Text Widget

ซึ่งทั้งหมดเป็นส่วนหนึ่งของ `package:flutter/material.dart`

เราเขียน StatelessWidget และ StatefulWidget มาแล้ว โดยที่ทดสอบมาแล้วว่ามันต่างกันอย่างไร ซึ่ง Widget ที่เราจะเขียนทั้งหมด ก็จะ Subclass มาจาก สองอันนี้นี่แหละ เมื่อ Subclass ออกมาแล้ว สิ่งที่เราจะทำก็คือ override `build()` ซึ่งเราจะลองทำกันต่อๆไป

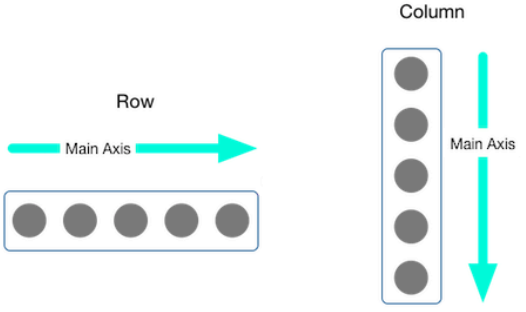
6.2 Basic Widgets

Flutter มี Widgets standard มาให้เราใช้ ขอยกตัวอย่างดังนี้

`Text` ไว้แสดงตัวหนังสือ ประมาณว่าเหมือน Label

`Row, Column`

ไว้จัด Layout ในจอ Row คือการจัดช่องแนวนอน ส่วน Column คือ แบ่งช่องแนวตั้ง จริงๆอันนี้ทำผมงงเหมือนกัน รู้สึกว่ามันสลับๆกัน แต่ก็ต้องจำไว้



`Stack`

มีไว้ให้วาง Widget ซ้อนๆกันไปเรื่อย เมื่อวาง เราก็เลือกวางได้ว่าเราจะวางเรียงจากไหน เป็นการเรียง stack แบบ web โดยคล้ายกับการใช้ `absolute` ในเวป layout

Container

มีไว้สร้างกล่องสี่เหลี่ยม สามารถตกแต่งได้ ใส่สี แรเงา ใส่เส้น มี margin เรียกว่าเป็น `div` ก็ได้มั้ง เห็นว่าเปลี่ยนเป็น 3d ได้ด้วย ถ้าตั้งเป็น `matrix`

สมมุติว่าสร้าง `MyAppBar` Widget ขึ้นมา 1 อัน

```dart
class MyAppBar extends StatelessWidget {
  final Widget title;

  // constructor
  MyAppBar({required this.title});

  @override
  Widget build(BuildContext context) {
    return Container(
      height: 56.0,
      padding: const EdgeInsets.symmetric(horizontal: 8.0),
      decoration: BoxDecoration(color: Colors.blue[500]),
      child: Row(
        children: <Widget>[
          IconButton(
            icon: Icon(Icons.menu),
            tooltip: 'Navigation menu',
            onPressed: null,
          ),
          Expanded(
            child: title,
          ),
          IconButton(
            icon: Icon(Icons.search),
            tooltip: 'Search',
            onPressed: null,
          ),
        ],
      ),
    );
  }
}
```

มาลองอ่าน Code กัน

1. `MyAppBar` เป็น Subclass ของ StatelessWidget โดยบังคับห้าม Widget title (สังเกตุว่าเป็น final) เป็น null และ Assign ได้แค่ครั้งเดียว
2. มี Constructor ที่จะรับค่าเป็น `MyAppBar({required this.title});` บังคับให้ส่งมาสร้าง MyAppBar
3. @override `build()` เพื่อสร้าง Widget Container แล้ว Return กลับไป
4. Container สามารถตั้งค่า `height`, `padding`, `decoration` ได้
5. เมื่อตั้งค่า Container แล้ว ข้างใน Container จะมี child (ซึ่งจะเก็บ Widget อื่นๆไว้)
6. ใน `child` ก่อนอื่นเลยมี Widget ชื่อว่า `Row`
7. ใน `Row` (1 บรรทัดแนวนอน) ต้องการให้มี children 3 อัน
8. มี IconButton, Expanded, IconButton
9. IconButton ถ้า onPressed เป็น null คือ เทียบเท่า Disable
10. Icon นี่น่าจะมาจาก Material Icon
11. Expanded คือ อะไรก็ตามในนั้นจะขยายกว้างตามพื้นที่ที่เหลืออยู่

ถ้าอ่าน Code เสร็จแบบยังไม่ได้นำไปใช้ก็คือ มี AppBar ด้านบน ที่มีปุ่ม hamburger, มี text, มีปุ่ม search ด้านขวานั่นเอง

ต่อมาเราจะนำ MyAppBar มาใช้งาน แต่ว่าเราจะให้ Application ของเรามีทั้ง AppBar และ Body

เราจะตั้งหน้าใหญ่หน้านี้ว่า `MyApp`

```
class MyApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return Material(
      child: Column(
        children: <Widget>[
          MyAppBar(
            title: Text(
              'Example title',
              style: Theme.of(context) //
                  .primaryTextTheme
                  .headline6,
            ),
          ),
          Expanded(
```

```
            child: Center(
              child: Text('Hello, world!'),
            ),
          ),
        ],
      ),
    );
  }
}
```

สิ่งที่ MyApp ทำก็คือ

1. return Widget `Material` ออกมา ซึ่ง Material มันเป็นแค่สิ่งที่ทำให้แอพดูเป็น Material - อ้างอิงจาก
2. ข้างใน Material จะมี child ซึ่งรับเป็น Widget ซึ่งเราจะทำการส่ง Column เข้าไป
3. Column เป็นการแบ่งจอในรูปแบบแนวตั้ง ซึ่งรับ children เข้าไป
4. เราส่ง <Widget> Array เข้าไปใน children ดังนี้ MyAppBar และ Expanded
5. MyAppBar เราแค่สร้าง Text ส่งไปยัง Initializer เพื่อตั้ง Title ที่ถูก required ไว้
6. Expanded เราส่ง Text ชื่อ Hello, world เข้าไปเฉยๆ
7. เนื่องจากว่า MyAppBar สูงแค่ 56 Pixel ทำให้ Hello, world นั้นถูกขยายจนเต็มจอ

ทดลองใช้ Code ของเราโดยการแก้ไข `main()` ดังนี้

```
void main() {
  runApp(MaterialApp(
    title: 'My app',
    home: SafeArea(
      child: MyApp(),
    ),
  ));
}
```

เป็นการใช้ runApp ซึ่งเป็นส่วนหนึ่งของ `material.dart` อ้างอิงจาก
https://api.flutter.dev/flutter/widgets/runApp.html
ซึ่งเป็นการรัน Widget ขึ้นมาเต็มจอ

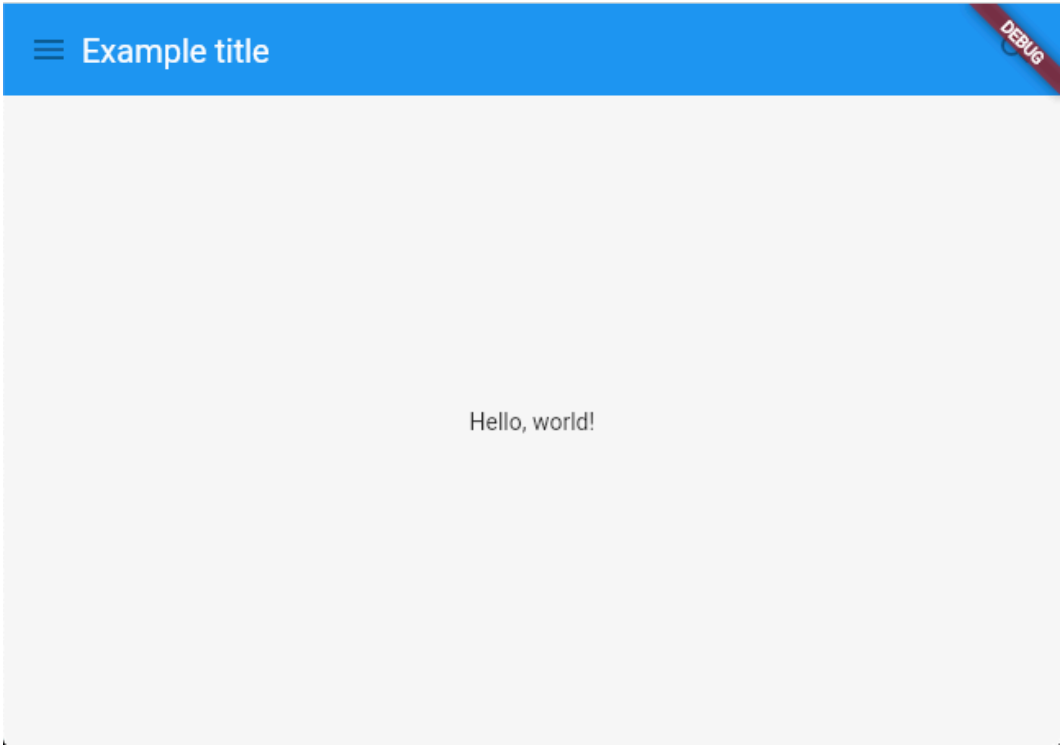
จากที่อ่าน ณ ตอนนี้คือ ถ้าสั่ง runApp จะต้องใช้ MaterialApp ซึ่งผมก็ยังไม่เข้าใจว่าทำไม แต่ว่าอ่านไปเรื่อยๆ เดี๋ยวก็คงเข้าใจเอง

Material App ต้องการให้ส่ง `title` เพราะว่าต้องใช้ใน Task Switcher และ `home` ซึ่งต้องการให้ส่ง Widget เข้าไป

ซึ่ง `home` ของเรานั้นจะใช้ `SafeArea` ซึ่งเป็นตัวบ่งบอกว่า เราต้องการให้ใช้ขนาดของจอที่ปลอดภัยจาก status bar, notch, ปุ่ม navigation ใน Android ซึ่งเราจะโยน `MyApp()` เข้าไปใช้ใน SafeArea นั้นเอง

เอาเป็นว่าอ่านมาถึงนี่ สงสัยมากว่า ทำไมมันใช้อะไรเยอะจัง (□_□|||) แต่เดี๋ยวอ่านไปเรื่อยๆก็น่าจะเข้าใจขึ้นอีกเองแหละ

ลองรัน code ดูแล้วได้แบบนี้ไหม



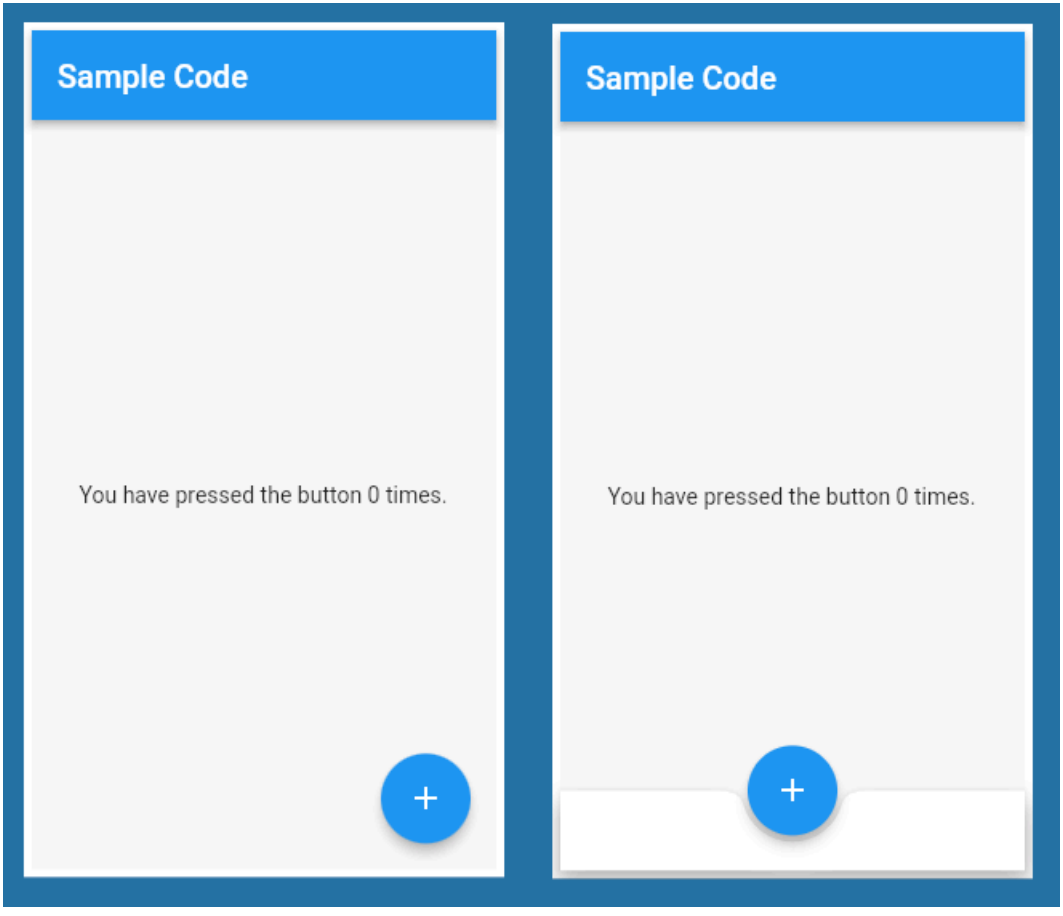
6.3 Material's Widgets

จริงๆแล้วที่เราทำใน Section ที่แล้วนั้น มันเป็นอะไรที่ปรกติมาก และ Material เค้าก็มี AppBar มาให้อยู่แล้ว ลบ Code ทิ้งให้หมดแล้วลองแปะ

```dart
class TutorialHome extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        leading: IconButton(
          icon: Icon(Icons.menu),
          tooltip: 'Navigation menu',
          onPressed: null,
        ),
        title: Text('Example title'),
        actions: <Widget>[
          IconButton(
            icon: Icon(Icons.search),
            tooltip: 'Search',
            onPressed: null,
          ),
        ],
      ),
      // body is the majority of the screen.
      body: Center(
        child: Text('Hello, world!'),
      ),
      floatingActionButton: FloatingActionButton(
        tooltip: 'Add', // used by assistive technologies
        child: Icon(Icons.add),
        onPressed: null,
      ),
    );
  }
}
```

ถ้าลองอ่านดู จะเห็นสิ่งแรกที่แปลกตาคือ `Scaffold` ซึ่งส่วนใหญ่แล้วเวลาเจอคำนี้ ผมจะกลัวมันนิดๆ เพราะว่าไม่รู้ว่ามันทำอะไรให้บ้าง มันเหมือนคำสั่งทำให้ทุกอย่างเลย ไม่ต้องเขียนเอง

อ้างอิงจาก https://api.flutter.dev/flutter/material/Scaffold-class.html
สิ่งที่ Scaffold ทำก็คือ จัด Layout ให้เหมือนกับของ Material Design ประมาณในรูป



รับ appBar, body, floatingActionButton ซึ่งเป็น style ของ Material Design

ซึ่ง appBar ก็รับ Widget ชื่อว่า AppBar ของ Material เข้าไปเลย ประกอบด้วย leading, title, actions แต่ถ้าใครลองรันแล้วมองไม่เห็นปุ่ม actions ไม่ต้องตกใจนะครับ น่าจะโดนคำว่า DEBUG บังอยู่

ใน body ก็มีแค่ `Center` Widget ที่มี Text อยู่

และสุดท้าย floatingActionButton ก็เป็นแค่ปุ่มลอยๆอยู่ด้านล่างขวา

6.4 handle event

ใน FloatingActionButton มันมี onPressed ซึ่งเป็น `null` อยู่ ด้วยความสงสัย ผมลองเพิ่ม Methods เข้าไปใน Class TutorialHome ดูเล่นๆชื่อว่า `handleAddButton` และแก้ onPressed ให้ไป call `handleAddButton` ดังนี้

```
class TutorialHome extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    ...
    ...
    ...
    ...
      floatingActionButton: FloatingActionButton(
        tooltip: 'Add',
        child: Icon(Icons.add),
        onPressed: handleAddButton,
      ),
    );
  }

  void handleAddButton() {
    print("test");
  }
}
```

ลองรันแล้วกดปุ่ม + ดู จะเห็น Log ใน Console

6.5 Gesture Detector

หรือหากเราต้องการเช็ค Gesture บน Widget ด้วยตนเอง ก็ใช้ `GestureDetector` ครอบไว้แล้วเช็ค onTap

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

class MyButton extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return GestureDetector(
      onTap: () {
        print('MyButton was tapped!');
      },
      child: Container(
        height: 50.0,
        padding: const EdgeInsets.all(8.0),
        margin: const EdgeInsets.symmetric(horizontal: 8.0),
        decoration: BoxDecoration(
          borderRadius: BorderRadius.circular(5.0),
          color: Colors.lightGreen[500],
        ),
        child: Center(
          child: Text('Engage'),
        ),
      ),
    );
  }
}

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

ความจริงแล้วมีอีกเยอะเลย นอกจาก onTap

`onDoubleTap`, `onDoubleTapCancel`, `onDoubleTapDown`, `onForcePressEnd`, ..

https://api.flutter.dev/flutter/widgets/GestureDetector-class.html

6.6 Handling Simple State

ก่อนหน้านี้ใน Stateless ทุกอย่างเป็น final ทำให้เราแก้ค่าอะไรไม่ได้เลยหลังจากแอพรัน

ตอนนี้เราจะลองทำให้แอพเรามีการเปลี่ยนแปลงตามปุ่มที่กดดูบ้าง

ลบโค้ดทิ้งให้หมดยกเว้น import แล้ว เริ่มสร้าง StatefulWidget กัน

```
class Counter extends StatefulWidget {
  @override
  _CounterState createState() => _CounterState();
}
```

สร้าง Counter ซึ่งเป็น StatefulWidget และมีแค่ method ชื่อว่า `createState()` ซึ่งจะ return _CounterState กลับไป

สังเกตุ Naming Convention ว่า State นั้นจะขึ้นต้น Class ด้วย _ เสมอ เช่น _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>[
        ElevatedButton(
          onPressed: _increment,
          child: Text('Increment'),
        ),
        SizedBox(width: 16),
        Text('Count: $_counter'),
      ],
    );
```

```
23    }
24  }
```

ใน _CounterState นั้นมี instance variable ชื่อว่า _counter เป็น int (สังเกตว่าขึ้นต้นด้วย _ เหมือนกัน) เป็นการตั้งชื่อเพื่อไม่ให้เข้าใจผิดกับส่วนอื่นๆ

ใน class นี้จะมี method ชื่อว่า `_increment()` ซึ่งภายในจะ call `setState()` ซึ่งเป็นสิ่งที่ flutter บังคับให้เรา call ทุกครั้งที่เราต้องการให้ `build()` methods นั้นรันใหม่ หรือ! เราต้องการให้ Widget เรานั้นเปลี่ยนแปลงตาม State ใหม่

ใน `build()` นั้น เราแค่ return widget `Row` ออกไปโดยที่ให้

- align center
- มีปุ่ม เมื่อกดแล้ว call `_increment()`
- มี Text ที่อัพเดทตาม `int _counter`

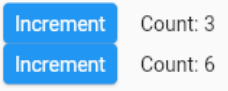
> เมื่อกดปุ่ม Increment ตัว code จะรัน `_increment()` ซึ่งจะรัน `setState()` เมื่อ setState() เสร็จ จะทำให้ `build()` ถูกบังคับให้รันใหม่และหน้าจอเปลี่ยนตามสิ่งที่ return ออกมาจาก `build()`

State ของ Counter นั้นใช้งานแยกกัน ลองดูตัวอย่างนี้

```
1   void main() {
2     runApp(MaterialApp(
3       home: SafeArea(
4         child: Scaffold(
5           body: Center(
6             child: Column(children: [
7               Expanded(child: Text("")),
8               Counter(),
9               Counter(),
10              Expanded(child: Text(""))
11            ]),
12          ),
13        ),
```

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

`Counter()` ถูกสร้างสองครั้ง ลองรัน Code ดูจะเห็นว่า _CounterState นั้นนับแยกออกจากกัน



6.7 Stateless with Stateful

จริงๆแล้ว ที่เราเขียนเนี่ย เรามีการใช้ Stateless widget ไปในโค้ดของเราด้วย เช่น

ElevatedButton (ซึ่งผมเข้าใจว่า เป็น Stateless Widget) ที่เรานำมาใช้

แต่เพื่อให้เข้าใจว่า Stateless และ Stateful ทำงานร่วมกันอย่างไร เราจะดูตัวอย่าง code นี้

Stateless widget ที่จะแสดงค่าแค่ Text Count: ออกไป ไม่มีการเปลี่ยนแปลงค่า

```dart
class CounterDisplay extends StatelessWidget {
  CounterDisplay({required this.count});

  final int count;

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

ปุ่มไว้กด เปลี่ยนค่า count แต่ว่าปุ่มไม่มีค่าอะไรเปลี่ยน มีหน้าที่แค่ส่ง event

```dart
class CounterIncrementorButton extends StatelessWidget {
  CounterIncrementorButton({required this.onPressed});

  final VoidCallback onPressed;

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

CounterDisplay และ CounterIncrementorButton เป็น Stateless Widget เพราะว่าไม่มีค่าอะไรเปลี่ยนข้างในเลย หรือ ไม่มีอะไร Mutate เลย (mutate ที่แปลว่ากลายพันธ์)

ต่อมาเราก็จะนำ Stateless Widget เนี่ยไปใช้กับ Stateful Widget

```dart
class Counter extends StatefulWidget {
  @override
  _CounterState 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>[
        CounterIncrementorButton(onPressed: _increment),
        SizedBox(width: 16),
```

```
22          CounterDisplay(count: _counter),
23        ],
24      );
25    }
26  }
```

Counter เป็น Stateful Widget ซึ่งพึ่งพา _CounterState ที่มีการเปลี่ยนค่า `int _counter` เรื่อยๆ

ใน _CounterState ซึ่ง ถูก ใช้ โดย Counter เนื่ย ก็ มี การ ใช้ CounterIncrementorButton และ CounterDisplay ที่เป็น Stateless ใน `build()`

สังเกตว่า ทั้งสองอย่างนั้น ทำงานแยกกันอย่างเห็นได้ชัดว่า เฉพาะคนที่ต้องการมีค่าเปลี่ยนเท่านั้นที่จะเป็น Stateful

```
1   void main() {
2     runApp(
3       MaterialApp(
4         home: Scaffold(
5           body: Center(
6             child: Counter(),
7           ),
8         ),
9       ),
10    );
11  }
```

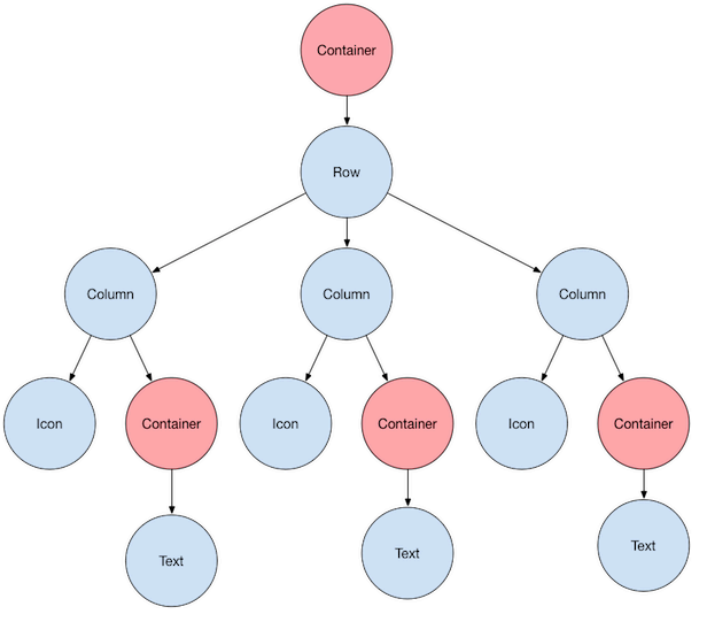
ลอง Start Debugging ดู ผลได้เหมือนเดิม แต่ว่าการใช้ Widget ต่างกัน

7. Layout

การสร้าง Layout เป็นพื้นฐานของการเขียน Flutter เนื่องจากว่าเราจะต้องสร้าง View เยอะมากๆ

ยกตัวอย่างด้านล่าง

จะวาดออกมาเป็น Widget Layout Diagram ได้แบบนี้



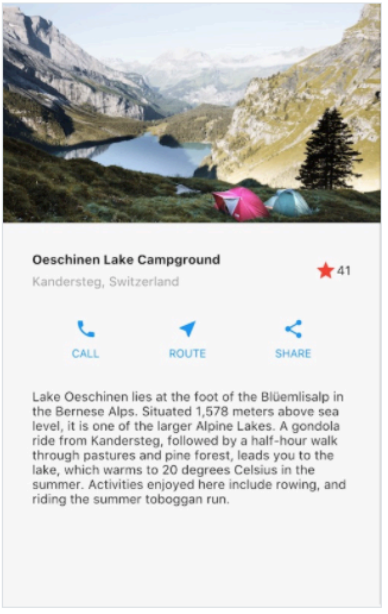
ซึ่งเป็นการทำ Container ซ้อน Row ซ้อน Column ลงไปเรื่อยๆ

ในบทนี้ ผมอ่านจาก

https://flutter.dev/docs/development/ui/layout/tutorial

แล้วลองทำความเข้าใจไปเรื่อยๆ อันไหนงงก็หาอ่านเพิ่ม

เริ่มต้นจากผลลัพท์ของบทนี้ ที่เราจะเขียนโค้ดกัน



เริ่มต้นจาก Code นี้เป็นหลัก

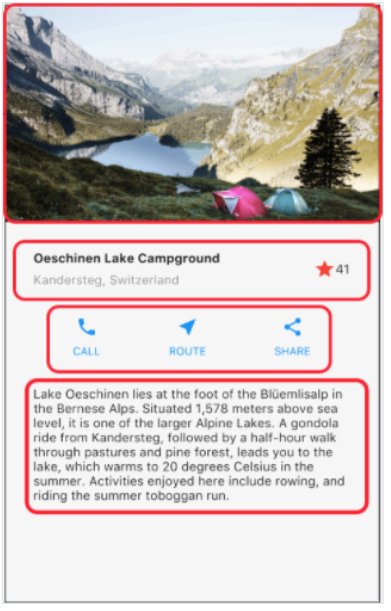
‘lib/main.dart’

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

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

class MyApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'Flutter layout demo',
      home: Scaffold(
        appBar: AppBar(
          title: Text('Flutter layout demo'),
        ),
```

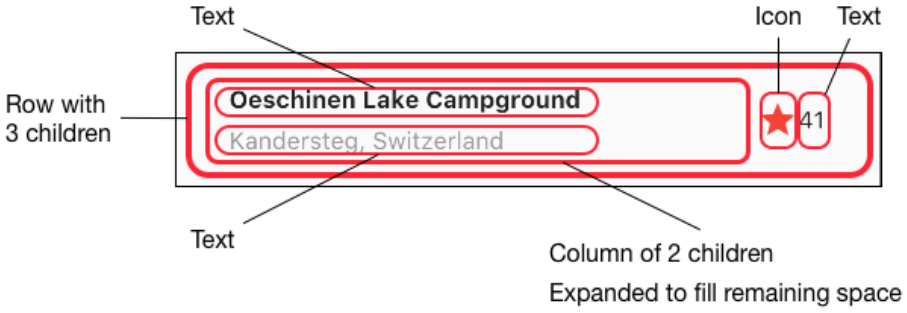
```
14          body: Center(
15            child: Text('Hello World'),
16          ),
17        ),
18      );
19    }
20  }
```

ต่อมา เริ่มมาแบ่งสัดส่วนของจอ ว่ามันมีอะไรบ้าง

เริ่มจาก Column ดังนี้



ต่อมา เราจะมาแบ่งส่วนที่ดูเหมือนจะมีอะไรซ้อนข้างในเยอะๆ เริ่มจาก



สังเกตส่วน Row และ Column ด้านในให้ดีว่าแบ่งอย่างไร

หากลองเขียนโต้ดดู จะเขียนได้ดังนี้

```
Widget titleSection = Container(
  padding: const EdgeInsets.all(32),
  child: /*1*/ Row(
    children: [
      /*2*/
      Expanded(
        child: /*3*/ Column(
          crossAxisAlignment: CrossAxisAlignment.start,
          children: [
            /*4*/
            Container(
              padding: const EdgeInsets.only(bottom: 8),
              child: Text(
                'Oeschinen Lake Campground',
                style: TextStyle(
                  fontWeight: FontWeight.bold,
                ),
              ),
            ),
            /*5*/
            Text(
              'Kandersteg, Switzerland',
              style: TextStyle(
                color: Colors.grey[500],
              ),
            ),
          ],
        ),
```

```
29        ),
30        /*6*/
31        Icon(
32          Icons.star,
33          color: Colors.red[500],
34        ),
35        /*7*/
36        Text('41'),
37      ],
38    ),
39  );
```

/*1*/ Row

เราเริ่มจากกรอบนอกสุด ซ้ายไปขวา มีทั้งหมด 3 Widget ประกอบด้วย Widget ดังนี้

`/*2*/ /*6*/ /*7*/`

/*2*/ Expanded

มีส่วนช่วยให้ Widget ของเรานั้น ขยายตามพื้นที่ที่เหลืออยู่ ยืดจนเต็มจอโดยหัก space จาก `/*6*/ /*7*/`

/*3*/ Columns

ต่อมา Widget ที่มีแต่ Text ด้านใน บนลงล่าง ประกอบด้วย

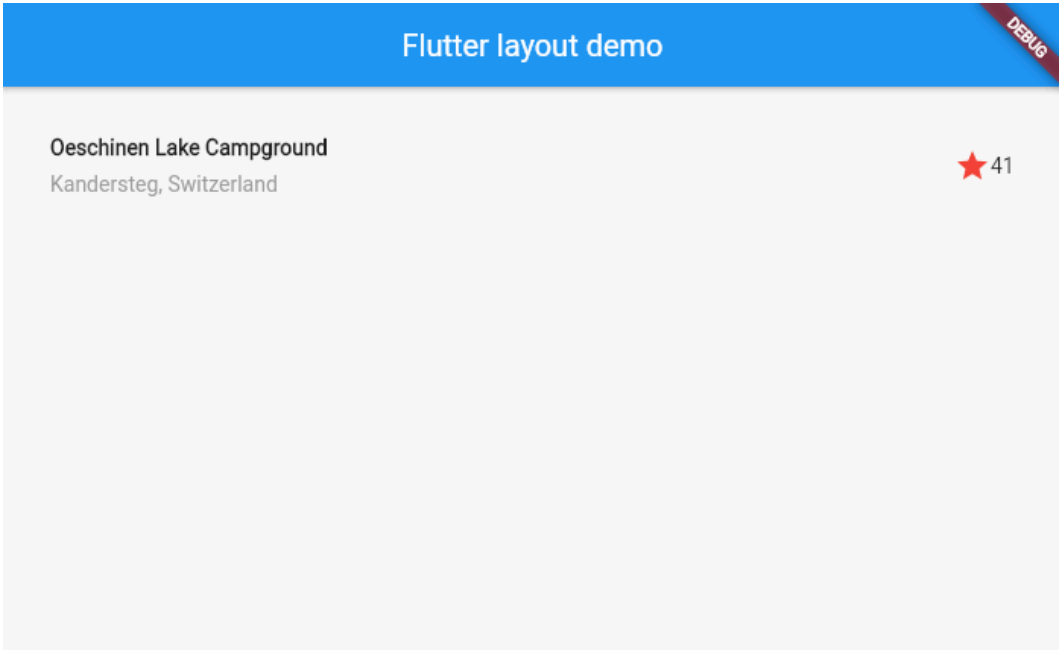
`/*4*/ /*5*/`

/*4*/ Container

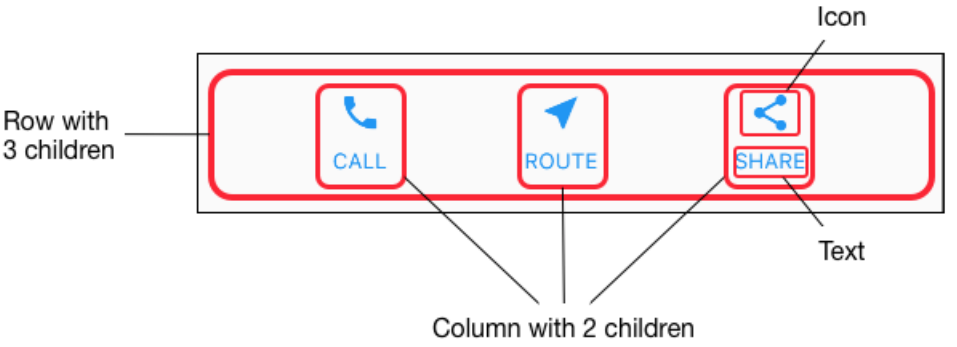
จริงๆแล้ว ไม่จำเป็นต้องมี Container ก็ได้ แต่ว่ามีไว้เพื่อให้เราสามารถ set padding ได้ก็ที่จะเข้าไป set Text ทำให้ห่างออกจาก `/*5*/` ด้วย

ลองนำโค้ดไปใช้ โดยแก้ไข `body:` ดังนี้

```dart
class MyApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'Flutter layout demo',
      home: Scaffold(
        appBar: AppBar(
          title: Text('Flutter layout demo'),
        ),
        body: Column(
          children: [titleSection],
        ),
      ),
    );
  }
}
```



ต่อมา ส่วนของปุ่ม เราสามารถแบ่งได้ดังนี้



แต่ว่าเนื่องจากว่า ปุ่มแต่ละปุ่มนั้น หน้าตาเกือบเหมือนกันเลย ต่างกันแค่ Icon และ Text เราจึงจะเริ่มจากการสร้าง Method เพื่อสร้างปุ่ม

```dart
Column _buildButtonColumn(Color color, IconData icon, String label) {
  return Column(
    mainAxisSize: MainAxisSize.min,
    mainAxisAlignment: MainAxisAlignment.center,
    children: [
      Icon(icon, color: color),
      Container(
        margin: const EdgeInsets.only(top: 8),
        child: Text(
          label,
          style: TextStyle(
            fontSize: 12,
            fontWeight: FontWeight.w400,
            color: color,
          ),
        ),
      ),
    ],
  );
}
```

การสร้างปุ่มก็แค่รับค่า สี icon label มาเท่านั้น และสร้าง Column เพื่อมาเก็บ Widget Icon ไว้บน และ Text ไว้ล่าง

หากใคร call function นี้ไป ก็จะได้ Widget Column กลับไปใช้งาน

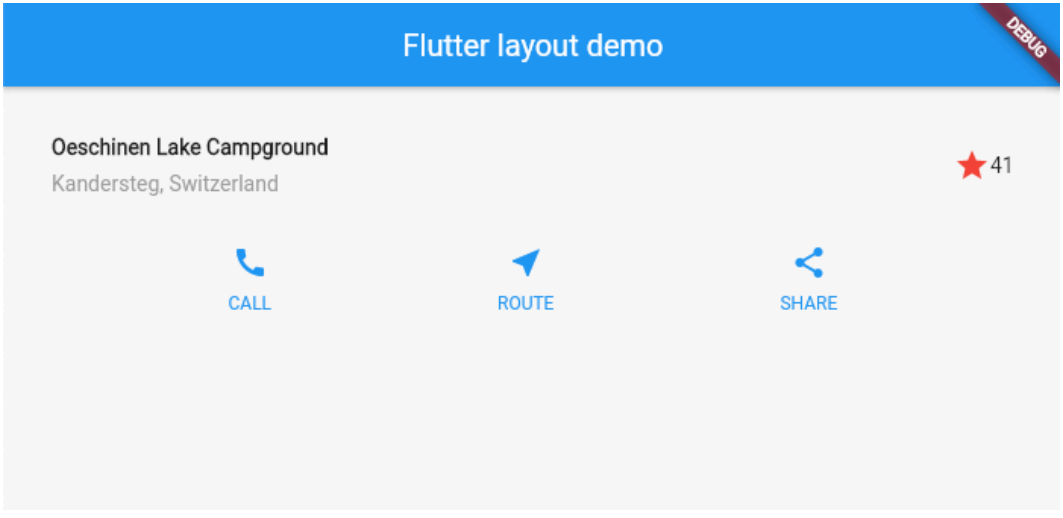
ใน `build()` เราแค่เข้าไปเพิ่มว่าเราจะสร้าง ปุ่ม 3 ปุ่มด้วยสีอะไร

```
    Color color = Theme.of(context).primaryColor;
    Widget buttonSection = Container(
      child: Row(
        mainAxisAlignment: MainAxisAlignment.spaceEvenly,
        children: [
          _buildButtonColumn(color, Icons.call, 'CALL'),
          _buildButtonColumn(color, Icons.near_me, 'ROUTE'),
          _buildButtonColumn(color, Icons.share, 'SHARE'),
        ],
      ),
    );
```

และนำ buttonSection ไปเพิ่ม `children` ใน `body` ต่อจาก `titleSection` ดังนี้

```
      body: Column(
        children: [titleSection, buttonSection],
      ),
```

แอพที่ได้จะหน้าตาแบบนี้



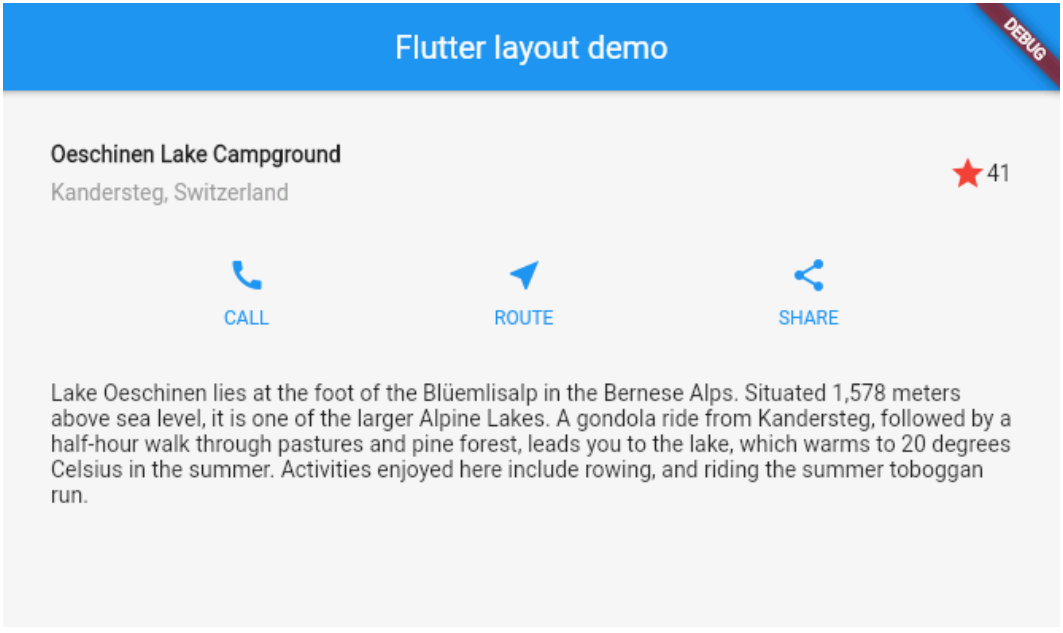
ต่อมาแค่เพิ่ม Text ที่ต้องการให้ Wrap ขึ้นบรรทัดใหม่หากเกินจอง่ายๆ ดังนี้

```
Widget textSection = Container(
  padding: const EdgeInsets.all(32),
  child: Text(
    '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.',
    softWrap: true,
  ),
);
```

และนำ `textSection` ไปแปะต่อจาก `buttonSection` ใน `build()` ดังนี้

```
      body: Column(
        children: [titleSection, buttonSection, textSection],
      ),
```

แอพที่ได้จะหน้าตาแบบนี้



สุดท้ายการเพิ่มรูปภาพ

รูปภาพนั้นถือเป็น asset ชนิดหนึ่งที่ต้องถูกตั้งค่าไว้ก่อนใน project ก่อนที่จะนำมาใช้งานได้

สามารถโหลดรูปได้ที่นี่

`https:// raw.githubusercontent.com/ flutter/ website/ master/ null_safety_examples/ layout/lakes/step5/images/lake.jpg`

หรือใช้รูปอะไรก็ได้ ให้สร้าง folder ชื่อว่า assets/images และนำ lake.jpg ไปไว้ใน folder

ต่อมาให้เปิด file ชื่อว่า `pubspec.yaml` แล้วเลื่อนไปหา `flutter:`

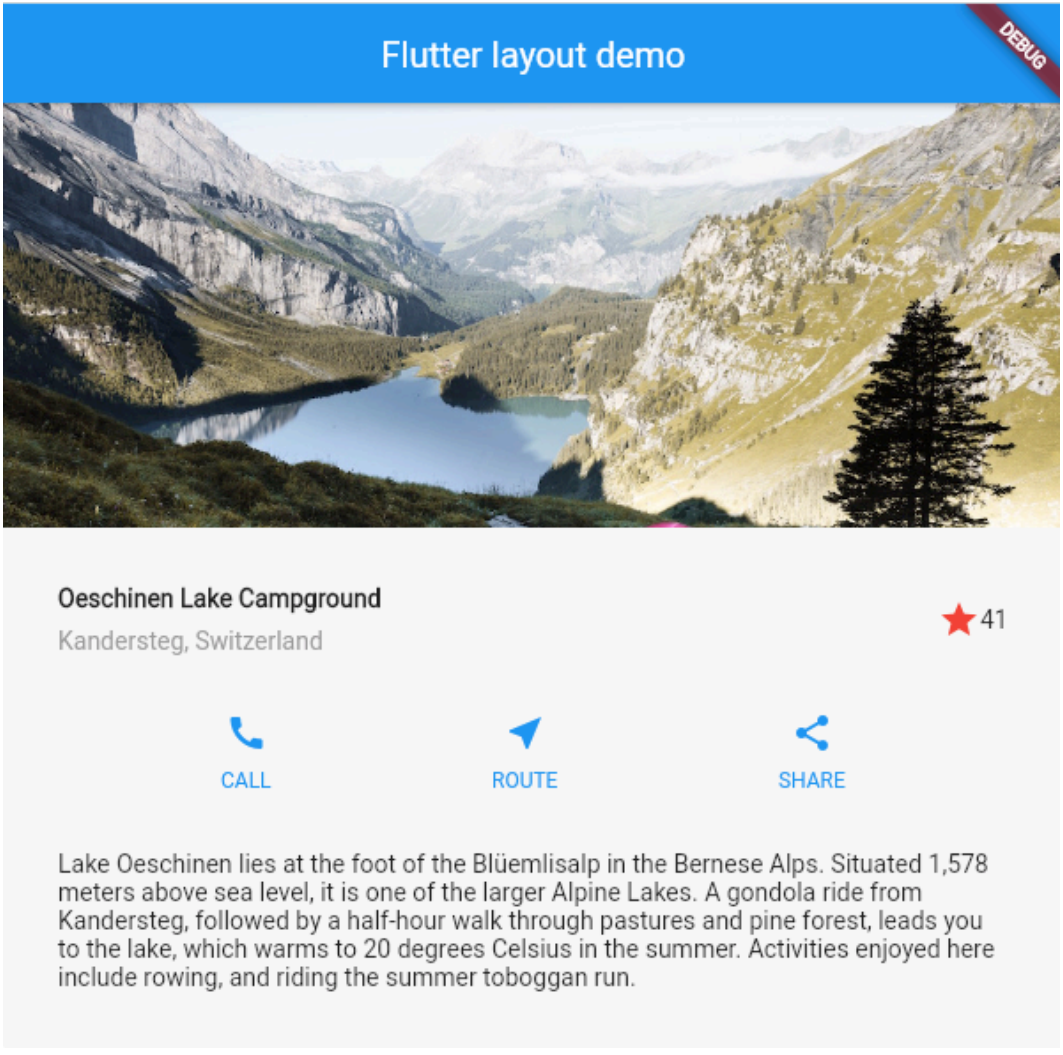
```
flutter:

  # The following line ensures that the Material Icons font is
  # included with your application, so that you can use the icons in
  # the material Icons class.
  uses-material-design: true

  # To add assets to your application, add an assets section, like this:
  assets:
    - assets/images/lake.jpg
```

ให้มองหาคำว่า `assets:` ที่อยู่ใต้ `flutter:` แล้วเพิ่มรูปที่เราต้องการใช้เข้าไป

> - podspec เป็น case sensitive จะต้องพิมทุกอย่างให้เป๊ะ ตัวเล็ก ตัวใหญ่ space
> - เมื่อเพิ่ม Assets แล้ว สิ่งที่ผมเจอก็คือ ต้อง restart ตัว debuging ที่เรารันอยู่ ไม่งั้น assets จะไม่ถูกโหลด

สุดท้าย ไปเพิ่มรูปเข้า `build()` ในส่วนของ `body:` ดังนี้

```
          body: Column(
            children: [
              Image.asset('assets/images/lake.jpg',
                  width: 600, height: 240, fit: BoxFit.cover),
              titleSection,
              buttonSection,
              textSection
            ],
          ),
```



สุดท้าย การที่เราสร้าง Layout แล้ว Content เกินจอนั้น มันจะมีแถบเหลืองๆขึ้นมาบอกว่ามันเกินจอไปกี่ pixels ซึ่งในกรณีนี้ หากเราต้องการให้แอพของเราสามารถเลื่อนขึ้นลง (scroll) ได้นั้น เราจะใช้ ListView แทน Column ครับ

ไปแก้ที่ `build()` ในส่วน `body` เหมือนเดิมโดยเปลี่ยน Column เป็น ListView

```
        body: ListView(
          children: [
            Image.asset('assets/images/lake.jpg',
                width: 600, height: 240, fit: BoxFit.cover),
            titleSection,
            buttonSection,
            textSection
          ],
        ),
```

กลับไปดูที่แอพ แถบเหลืองๆก็จะหายไปและ scroll ได้แล้ว

สำหรับบทนี้ ขอจบแต่เพียงเท่านี้ เพื่อที่จะให้เข้าใจ Layout ทั่วๆไป

8. Basic Routes with GETX

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/soon-to-be-flutter-programmer.

8.1 Installing getx

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/soon-to-be-flutter-programmer.

8.2 Create Project with getx

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/soon-to-be-flutter-programmer.

8.3 Files / Folders Structure

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/soon-to-be-flutter-programmer.

8.4 Creating new page

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/soon-to-be-flutter-programmer.

8.5 Search field

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/soon-to-be-flutter-programmer.

8.6 Show Manga Detail Page

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/soon-to-be-flutter-programmer.