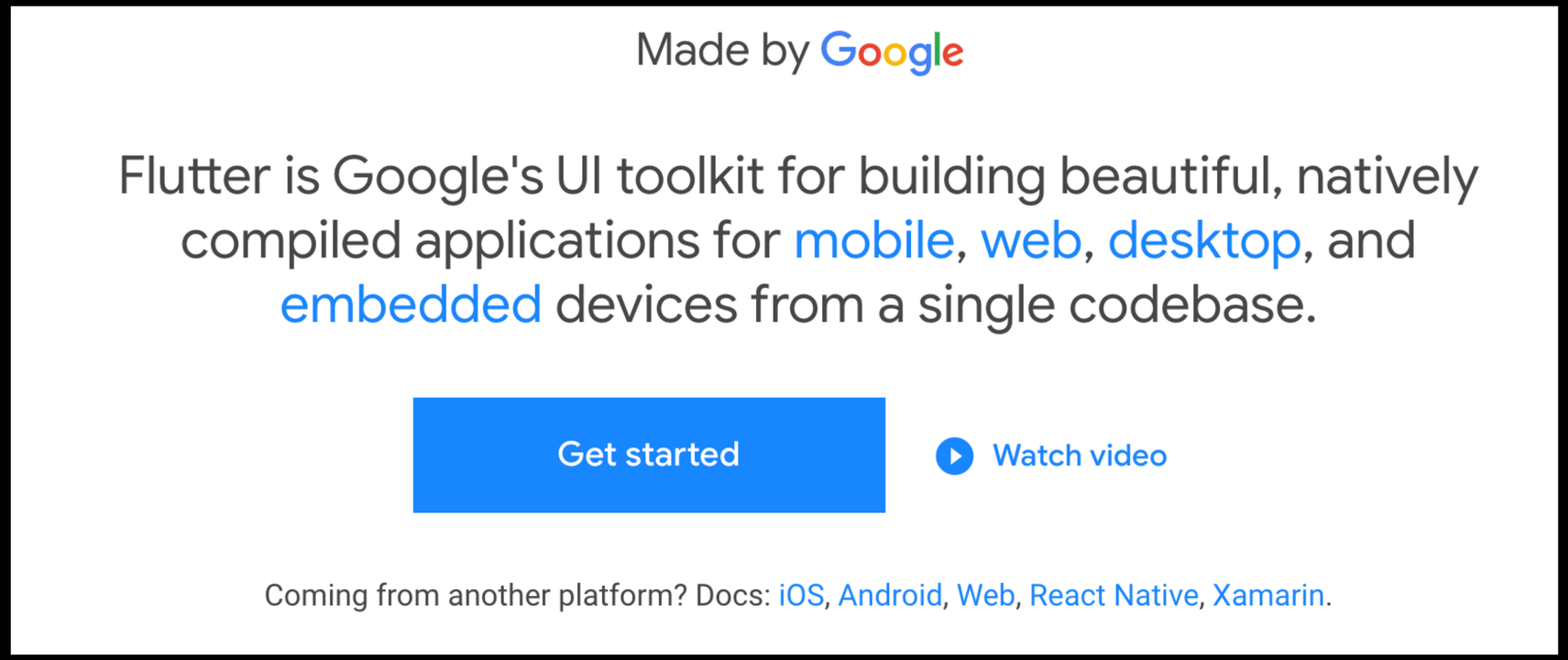
Flutter แบบ ส่งบอล

ODDS (LeSSEx)

Song, Ball ; last update at 21 May 2021

What is Flutter



Install

Flutter 2.2.0 (Dart 2.13.0) - 20 May 2021

- MacOs Download

```
export PATH="$PATH:[PATH_OF_FLUTTER_DIRECTORY]/bin"
```

```
Ex: export PATH="$PATH:/user/odds/flutter/bin"
```

Need Xcode version 12.5 (20 may 2021)

Optional Android Studio version 4.2.1 (20 may 2021)

- Windows Download

Update Path

Need Android Studio version 4.2.1 (20 may 2021)

First Project

Command

```
> flutter create <project_name (snack_case)>*
```

* Flutter 2.2.0 and above create project with sound null safety

Basic Dart

Type

```dart
String name = “Ball”;
int age = 22;
double bmi = 27.28;
bool single = true;
List<String> skills = [
  “flutter”,
  “dart”,
];
Map<String, dynamic> todo = {
  “message”: “To day is today.”,
  “createdDate”: DateTime.now(),
  “complete”: false,
};
```

Basic Dart

Null Safety

```dart
late bool enable;
```

เดี๋ยวจะมีค่าแน่ๆ แต่ยังไม่มีตอน Declare ตัวแปร

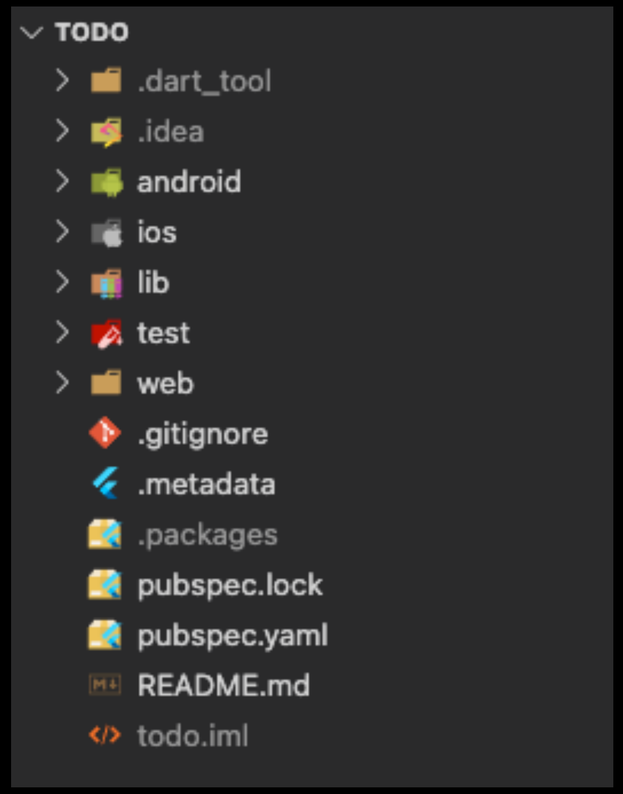
```dart
String? Note;
```

อาจจะไม่มีค่า อาจจะเป็น null ได้

https://dartpad.dev/40d7a46547164041f50773ad600d057b

https://muyonz.medium.com/dart-2-12-null-safety-d5952442bd04

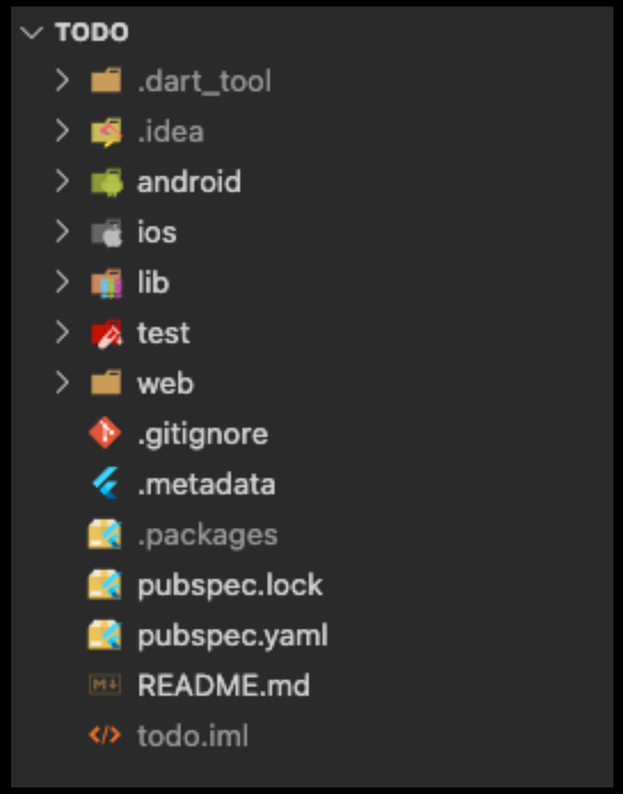
File Structure



lib

- screens
- models
- controllers
- widgets (component)
- helper
- const

File Structure



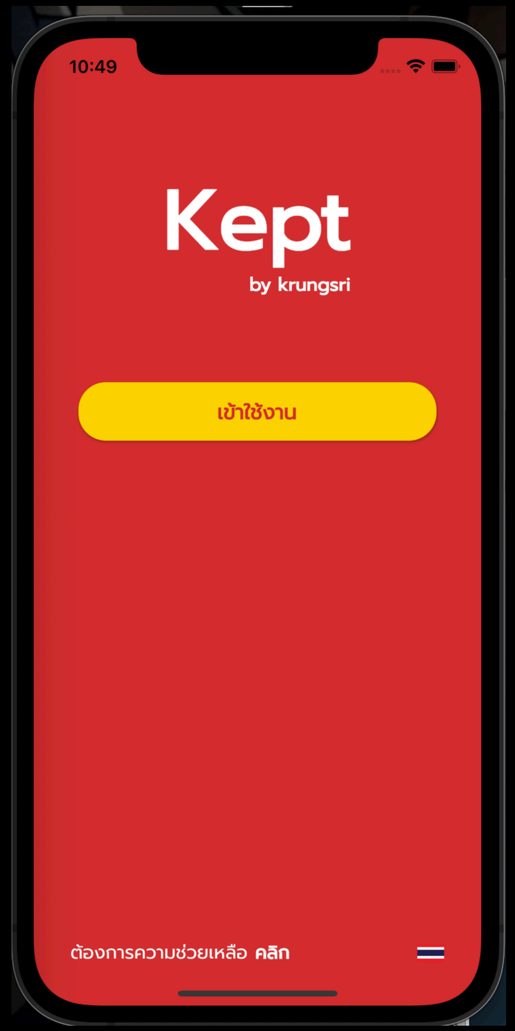
assets

- images

- icons

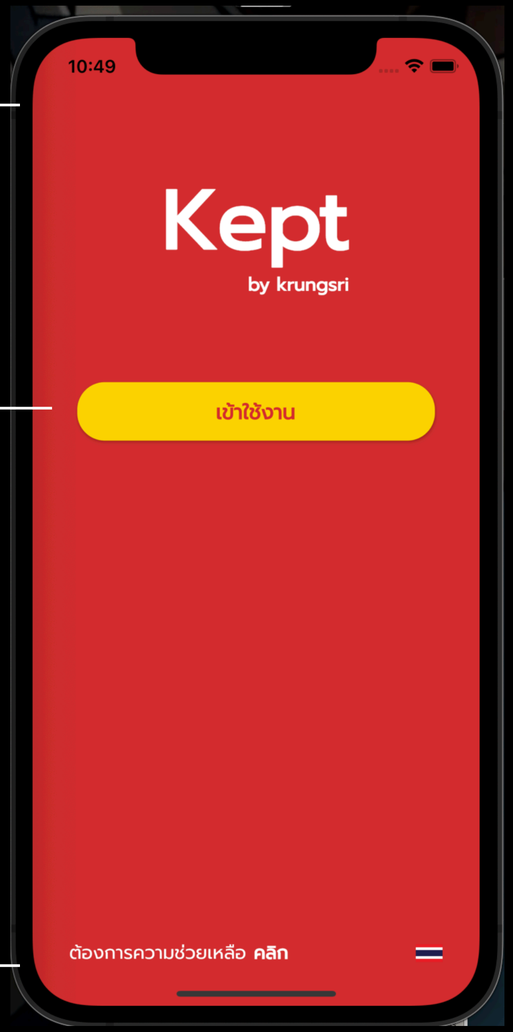
- fonts

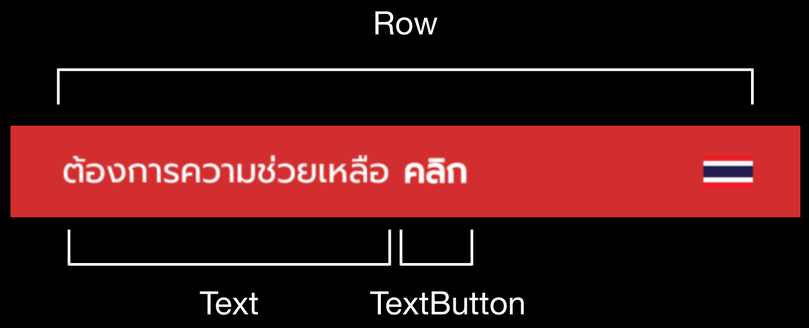
Layout



#Flutterแบบส่งบอล

Layout





#Flutterแบบส่งบอล

State Management

Install Dependency

```
dependencies:
  flutter:
    sdk: flutter
  get: ^4.1.4
```

Step 0 : Use GetMaterialApp instead of Material App

```
class MyApp extends StatelessWidget {
  const MyApp({Key? key}) : super(key: key);

  @override
  Widget build(BuildContext context) {
    return GetMaterialApp(
      home: LoginScreen(),
    ); // GetMaterialApp
  }
}
```

Step 1 : Create your business logic class

```
import “package:get/get.dart”;

class LoginController extends GetxController {}
```

Step 2.1 : Use GetBuilder to update Widgetwhenever value is changed.

```
Container (
  child: GetBuilder<LoginController>(
    init: LoginController(),
    builder: (controller){
      return Text(
        controller.text,
        style: TextStyle(
          fontSize: 20,
          color: AppColors.green[600],
          fontWeight: FontWight.bold,
        ), // TextStyle
      ); // Text
    },
  ), // GetBuilder
), // Container
```

Step 2.1 : Use GetX to update Widgetwhenever value is changed.

```
Container (
  child: GetX<LoginController>(
    init: LoginController(),
    builder: (controller){
      return Text(
        controller.text,
        style: TextStyle(
          fontSize: 20,
          color: AppColors.green[600],
          fontWeight: FontWight.bold,
        ), // TextStyle
      ); // Text
    },
  ), // GetX
), // Container
```

Declaring a reactive variable

```dart
// initial value is recommended, but not mandatory
final RxString name = RxString("");
final RxBool isLogged = RxBool(false);
final RxInt count = RxInt(0);
final RxDouble balance = RxDouble(0.0);
final RxList<String> items = RxList<String>([]);
final RxMap<String, int> myMap = RxMap<String, int>({});

// Custom classes - it can be any class, literally
final Rx user = Rx<User>();
```

Step 2.3 : Use Obx to update Text() whenever value is changed.

```
Container (
  child: Obx(
    () => Text(
      _loginController.title.toString(),
      style: TextStyle(
        fontSize: 20,
        color: AppColors.green[600],
        fontWeight: FontWight.bold,
      ), // TextStyle
    ), // Text
  ), // Obx
), // Container
```

Step 3 : Using instantiated classes

```
Container (
  child: Obx(
    () => Text(
      Get.find<LoginController>().title.value,
      style: TextStyle(
        fontSize: 20,
        color: AppColors.green[600],
        fontWeight: FontWight.bold,
      ), // TextStyle
    ), // Text
  ), // Obx
), // Container
```

Step 4.1 : Instantiate your class using Get.put

```
@override
void initState() {
  super.initState();
  Get.put(LoginController());
}
```

Step 4.2 : using Get.lazyPut

```
@override
void initState() {
  super.initState();
  Get.lazyPut<LoginController>(() => LoginController());
}
```

Step 4.3 : using Get.create

```dart
@override
void initState() {
  super.initState();
  Get.create(() => LoginController());
}
```

Step 4.4 : using Get.putAsync

```dart
@override
void initState() {
  super.initState();
  Get.putAsync<SharedPreferences>(() async {
    SharedPreferences prefs = await SharedPreferences.getInstance();
    return prefs;
  });
}
```

Route Management

Step 1.1 : Define routes, use GetMaterialApp instead of MaterialApp

```
@override
Widget build(BuildContext context) {
  return GetMaterialApp(
    initial: “/login”,
    getPages: [
      GetPage(name: “/login”, page: () => LoginScreen()),
      GetPage(name: “/register”, page: () => RegisterScreen()),
    ],
  ); // GetMaterialApp
}
```

Step 1.2 : Navigation with named routes

```
// Default Flutter navigator
Navigator.of(context).push(MaterialPageRoute(
  builder: (context) => RegisterScreen(),
));

// Get syntax
Get.put(“/register”);
```

Navigation with named routes

```
// navigate to nextScreen
Get.toNamed(“/NextScreen”);

// navigate and remove previous screen from the tree
Get.offNamed(“/NextScreen”);

// navigate and remove all previous screen from the tree
Get.offAllNamed(“/NextScreen”);

// navigate and remove all previous screen from the tree
Get.back(result: “success”);
```

Bindings

Create a class

```
import “package:get/instance_manager.dart”;
import “package:simple_login/screens/login/login_controller.dart”;

class LoginBinding extends Binding {
  @override
  void dependencies() {
    Get.put(() => LoginController());
  }
}
```

Bindings

implements Binding

```
getPages: [
  GetPage(
    name: “/login”,
    page: () => LoginScreen(),
    binding: LoginBinding(),
  ), // GetPage
],
```

Or

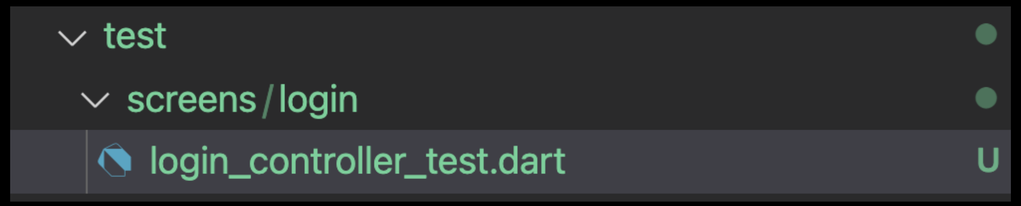
```
getPages: [
  GetPage(
    name: “/login”,
    page: () => LoginScreen(),
    binding: BindingBuilder(() {
      Get.lazyPut(() => LoginController());
    }),
  ), // GetPage
],
```

Unit Testing

Step 1 : Add the flutter_test dependency.

```
flutter_test:
  sdk: flutter
```

Step 2 : Create a test file



Step 3 : Write a test for our class

```
import “package:flutter_test/flutter_test.dart”;
import “package:simple_login/screens/login/login_controller.dart”;

main() {
  test(“valid username”,() async {
    LoginController controller = LoginController();
    const expect = null;
    var actual = controller.usernameValidator(
      “test01”
    );
    expect(actual, expect);
  })
}
```

Step 4 : Run the tests

```
> flutter test
  00:03 +1: All tests passed!
```

```
> flutter test test/screens/login/login_controller_test.dart
  00:03 +1: All tests passed!
```

Mockito 5.0.7 - Create Mock Service

Step 1 : Add a dependency in the pubspec.yaml file

```
dev_dependencies:
  mockito: 5.0.7
  build_runner: ^1.10.0
```

Step 2 : annotate a top-level library member with @GenerateMocks

```
@GenerateMocks([UserService])
main() {}
```

Step 3 : run build_runner in order to generate mocks

```
> flutter pub run builder_runner build
```

Step 4 : import generate mocks

```
import “login_controller_test.mocks.dart”;

@GenerateMocks([UserService])
main() {
  UserService mockUserService = MockUserService();
}
```

Mockito 5.0.7 - Create Mock View

```
class MockLoginView extends Mock implement LoginView {}
```

Implementing Our Unit Test

```
LoginView mockView = MockLoginView();
LoginController controller = LoginController(view: mockView);
```