Testing

허가받지 않은 복제(복사), 전송, 수정 및 배포를 금합니다.

Testing Flutter apps

test | Dart Package

A full featured library for writing and running Dart tests across platforms.

https://pub.dev/packages/test

pub.dev

목표.

앱이 기능이 늘어나면서 점점 무거워질수록 테스트 하기 어렵다.
테스트 코드를 작성하여 자동화하여 올바르게 작동하는지 확인하자.

- A *unit test* tests a **single function, method, or class**.
- A *widget test* (in other UI frameworks referred to as *component test*) tests a **single widget**.
- An *integration test* tests a **complete app or a large part** of an app.

trade-offs

	Unit	Widget	Integration
Confidence (신뢰도)	Low	Higher	Highest

Maintenance cost (유지보수 비용)	Low	Higher	Highest
Dependencies (의존성)	Few	More	Most
Execution speed (실행 속도)	Quick	Quick	Slow

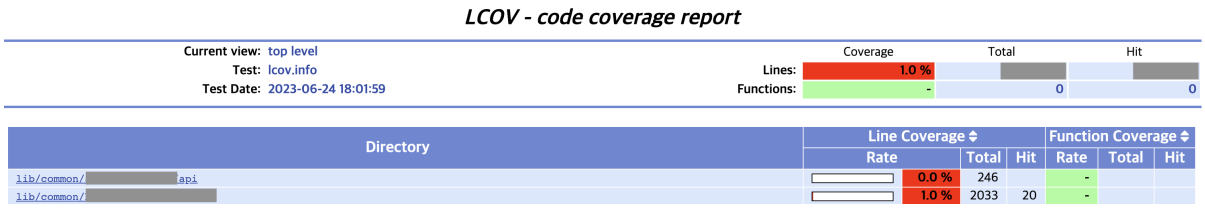
code coverage 활용하여 수치화.

```
flutter test --coverage

# 결과물
# ./coverage/lcov.info
```

https://github.com/linux-test-project/lcov 활용하여 HTML 로 확인하기

```
genhtml coverage/lcov.info -o coverage/html

# 결과물
# ./coverage/html/index.html
```

index.html:



LCOV - code coverage report

Current view: top level
Test: lcov.info
Test Date: 2023-06-24 18:01:59

	Coverage	Total	Hit
Lines:	1.0 %		
Functions:	-	0	0

Directory	Line Coverage			Function Coverage		
	Rate	Total	Hit	Rate	Total	Hit
lib/common/[redacted]api	0.0 %	246		-		
lib/common/[redacted]	1.0 %	2033	20	-		

Continuous integration services.

Continuous integration (CI) services allow you to run your tests **automatically** when **pushing new code changes**.

- 모든 브랜치 커밋 푸시 단위로 테스트를 돌리면, queue 에 job 이 너무 많이 쌓일 가능성이 있음.

👉 PR 올라온 브랜치만 workflow 실행
👉 Github Action 기준 `concurrency` , `cancel-in-progress` 활용하여 그룹핑 할 수 있음

해당 docs 에서 많은 자료 제공하였지만, 강력하게 **Github Actions** 추천

Flutter Festival Korea - CI & CD 소개 - 데비트

TDD.

Test Driven Development : 테스트 주도 개발 (Test First Development)

as-is

- 기획 및 디자인 ⇒ 코드 개발 ⇒ 테스트 코드 작성

to-be

- 기획 및 디자인 ⇒ 테스트 코드 작성 ⇒ 코드 개발

장점

- 더 안전한 객체 지향적인 코드 생산
- 코드 재사용을 위해 기능별 모듈화
- 높은 code coverage
- 빠른 디버깅
- docs 대체
- 사이드 이펙 가능성 최소화

단점

👉 **생산성 저하**

언제 적용 해보는게 가장 좋을까?

- 무엇보다도 안전성이 중요한 개발
- 서비스 앱 개발보다는 모듈 개발(채팅, 동영상 플레이어, 사진 편집기 등)에 더 적합
- ⭐ **우리 팀이 TDD 를 추구할 때.**

BDD.

Behavior Driven Development : 행위 주도 개발

- **TDD** 에서 파생
- 개발자 / 비개발자간 협업을 위함

- **사용자의 행위(시나리오) 기반으로 테스트 케이스를 작성**
- Unit Test 권장 x
- 시나리오는 최대한 이해하기 쉽게 작성

Given : 주어진 환경

When : 행위

Then : 기대 결과

시나리오

로그인된 사용자가 로그아웃 버튼을 누를 경우, 사용자 정보가 초기화된다.

- **Given :** 로그인된 사용자
- **When :** 로그아웃 버튼을 누를 경우
- **Then :** 사용자 정보가 초기화

Unit tests.

A *unit test* tests a **single function, method, or class.**

- 목표 : 다양한 conditions 에서 로직의 정확도를 검증
- disk 읽고 쓰기 ❌
- 화면 렌더링 ❌

▼ ***테스트 input 값은 어떻게 정하는게 좋을까?***

👉 **Edge Cases : 경계 값(최대/최소 값)**

> Corner Cases : Edge Case 모두 통과하지만 **외부 요인 또는 특정 조건에서 문제** 발생할 경우

An introduction to unit testing:

```
import 'package:test/test.dart';

class Counter {
  static const int maxValue = 10;
  int value = 0;

  void increment() {
    if (value >= maxValue) return;

    value++;
  }

  void decrement() => value--;
}

void main() {
  /// Single
  test('Counter value should be incremented', () {
    final counter = Counter();

    counter.increment();

    expect(counter.value, 1);
  });

  test('Counter value should be incremented', () {
    final counter = Counter();

    for (int index = 0; index < Counter.maxValue; index++) {
      counter.increment();
    }

    counter.increment();

    expect(counter.value, 10);
  });

  /// Group
```

```
  group('Counter', () {
    test('value should start at 0', () {
      expect(Counter().value, 0);
    });

    test('value should be incremented', () {
      final counter = Counter();

      counter.increment();

      expect(counter.value, 1);
    });

    test('value should be decremented', () {
      final counter = Counter();

      counter.decrement();

      expect(counter.value, -1);
    });
  });
}
```

Mock dependencies using Mockito.

종종 유닛테스트가 **서버 API 또는 DB**로 부터 데이터 fetch 해야하는데, 이때 어려움 존재

- 서버 live 할 때 호출해야함. DB slows down
- 데이터는 변할 수 있기 때문에 응답 값을 예측하기 어려움

👉 Mockito 활용하여 **목업 데이터 생성하여** 테스트를 실행 ⇒ 서버 데이터와의 의존성 제거

▼ ***클린아키텍처가 적용 되어있다면?***

👉 Data Layer > Data Source 만 교체해서 테스트 가능

matcher | Dart Package

Support for specifying test expectations via an extensible Matcher class. Also includes a number of built-in Matcher implementations for common cases.

https://pub.dev/packages/matcher

- **complex validations**
- **Exception**
- **Future**
- **Stream**

플러그인 테스트.

Plugins in Flutter tests.

Widget Tests.

A *widget test* (in other UI frameworks referred to as *component test*) tests **a single widget**.

- 목표 : widget's **UI looks and interacts** 예상대로 작동하는지 검증
- Unit Test 보다는 포괄적

`flutter_test` 패키지 사용.

- `WidgetTester` **활용**하여 테스트 환경에서 위젯들간의 상호작용
- Unit Test: `test()`, Widget Test: `testWidgets()`
- `Finder` 클래스 활용하여 위젯 search
- `Matcher` 활용하여 검증

Create a widget to test.

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

  final String title;
  final String message;

  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'Flutter Demo',
      home: Scaffold(
        appBar: AppBar(
          title: Text(title),
        ),
        body: Center(
          child: Text(message),
        ),
      ),
    );
  }
}
```

Create a testWidgets test.

- `testWidgets()` function provided by the `flutter_test`
- `WidgetTester` **tester**

```
void main() {
  // Define a test. The TestWidgets function also provides a WidgetTester
  // to work with. The WidgetTester allows you to build and interact
  // with widgets in the test environment.
  testWidgets('MyWidget has a title and message', (tester) async {
    // Test code goes here.
  });
}
```

Build the widget using the WidgetTester.

- `WidgetTester` 클래스의 `pumpWidget()` 메서드 활용하여 builds and renders.
- `StatefulWidget` or animations 유용함

```
void main() {
  testWidgets('MyWidget has a title and message', (tester) async {
    // Create the widget by telling the tester to build it.
    await tester.pumpWidget(const MyWidget(title: 'T', message: 'M'));
  });
}
```

`tester.pump(Duration duration)`

Schedules a frame and triggers a rebuild of the widget.

- 애니메이션 활용하려면 한번은 호출 해야함

`tester.pumpAndSettle()`

특정 동작, 애니메이션 등 수행이 끝날 때까지 **wait**

Search for our widget using a Finder.

```
void main() {
  testWidgets('MyWidget has a title and message', (tester) async {
    await tester.pumpWidget(const MyWidget(title: 'T', message: 'M'));

    // Create the Finders.
    final titleFinder = find.text('T');
    final messageFinder = find.text('M');
  });
}
```

Verify the widget using a Matcher.

```
void main() {
  testWidgets('MyWidget has a title and message', (tester) async {
    await tester.pumpWidget(const MyWidget(title: 'T', message: 'M'));
    final titleFinder = find.text('T');
    final messageFinder = find.text('M');

    // Use the `findsOneWidget` matcher provided by flutter_test to verify
    // that the Text widgets appear exactly once in the widget tree.
    expect(titleFinder, findsOneWidget);
    expect(messageFinder, findsOneWidget);
  });
}
```

- `findsOneWidget` : exactly **one widget** in the widget tree.
- `findsNothing` : Verifies that **no widgets** are found.
- `findsWidgets` : Verifies that **one or more widgets** are found.
- `findsNWidgets` : Verifies that **a specific number of widgets** are found.
- `matchesGoldenFile` Verifies that a **widget's rendering matches a particular bitmap image** ("golden file" testing).

Integration tests.

integration test tests **a complete app or a large part of an app.**

- 목표 : 예상대로 실제 앱이 작동 하는지 검증 (전체 또는 일부분)
- 실제 디바이스 또는 OS emulator (iOS Simulator)

An introduction to integration testing.

- integration_test 패키지 활용
- example: counter app

Create an app to test.

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

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

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

  @override
  Widget build(BuildContext context) {
    return const MaterialApp(
      title: 'Counter App',
      home: MyHomePage(title: 'Counter App Home Page'),
    );
  }
}

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

  final String title;

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

class _MyHomePageState extends State<MyHomePage> {
  int _counter = 0;

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

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: Text(widget.title),
      ),
      body: Center(
        child: Column(
          mainAxisAlignment: MainAxisAlignment.center,
          children: <Widget>[
            const Text(
              'You have pushed the button this many times:',
            ),
            Text(
              '$_counter',
              // Provide a Key to this specific Text widget. This allows
              // identifying the widget from inside the test suite,
              // and reading the text.
              key: const Key('counter'),
              style: Theme.of(context).textTheme.headline4,
            ),
          ],
```

```
        ),
      ),
      floatingActionButton: FloatingActionButton(
        // Provide a Key to this button. This allows finding this
        // specific button inside the test suite, and tapping it.
        key: const Key('increment'),
        onPressed: _incrementCounter,
        tooltip: 'Increment',
        child: const Icon(Icons.add),
      ),
    );
  }
}
```

Add the integration_test dependency

dev_dependencies

- `integration_test`
- `flutter_test`

```
dev_dependencies:
  integration_test:
    sdk: flutter
  flutter_test:
    sdk: flutter
```

⭐ 별도 테스트 폴더 생성 필요

`lib/integration_test`

Write the integration test.

```
import 'package:flutter_test/flutter_test.dart';
import 'package:integration_test/integration_test.dart';

import 'package:counter_app/main.dart' as app;

void main() {
  IntegrationTestWidgetsFlutterBinding.ensureInitialized();

  group('end-to-end test', () {
    testWidgets('tap on the floating action button, verify counter',
        (tester) async {
      app.main();
      await tester.pumpAndSettle();

      // Verify the counter starts at 0.
      expect(find.text('0'), findsOneWidget);

      // Finds the floating action button to tap on.
      final Finder fab = find.byTooltip('Increment');

      // Emulate a tap on the floating action button.
      await tester.tap(fab);

      // Trigger a frame.
      await tester.pumpAndSettle();

      // Verify the counter increments by 1.
      expect(find.text('1'), findsOneWidget);
    });
  });
}
```

Golden test.

특정 실제 위젯과 image 파일 비교

하나의 화면까지 확장 해나간다면?

👉 **snapshot(screenshot) test**

Continuous Integration

- 주기적으로 snapshot 테스트 실행

Golden test 참고.

golden_toolkit | Flutter Package

Common patterns for screenshot-based widget testing using Goldens.

https://pub.dev/packages/golden_toolkit

https://github.com/Visual-Regression-Tracker/Visual-Regression-Tracker

- 이미지 비교 도구