

Easily reach 100% coverage in Flutter

Guillaume Bernos - 11 March 2022

Who Am I?

- Bad Medium articles in 2017
- Creator of the `location` package
- Tech Lead at Bam in Paris
- Writing guillaume.bernos.dev on my free time



Overview

1. Coverage? Why should I care?
2. How to make testing easier
3. Architecture your project better thanks to testing

Coverage? Why should I care?

- Coverage is the percentage of code you're testing automatically
- It helps track down bugs automatically
- A higher coverage often means a longer time to write tests
- Help you understand edge cases
- Help you remove dead code

**Testing helps you understand your code
better**

Which percentage should I go for?

- It depends on the constraints and on the aim of the project
- If the existing codebase does not have a lot of tests, just improving this number is good
- For fast-paced projects, it can still be useful

Which tool should I use?

- Coverage Gutters in VSCode
- Flutter Coverage
- Special launch.json snippet
- LCov package
- CI Tools

```
{
  "name": "Coverage",
  "type": "dart",
  "request": "launch",
  "args": ["--coverage"],
  "codeLens": {
    "for": [ "run-test", "run-test-file" ],
    "title": "Coverage"
  },
}
```

How to make testing easier?

Typical widget test

- You need to inject your theme
- To create a golden
- And to be sure it has been tested on different sizes if you're responsive



Create extension on the tester

- With the `tester.pumpApp`, you can easily remove a lot of boilerplate
- If you modify something in the root of your app, easily modify your tests
- Should be pretty close to what you put in your root

```
pumpApp

extension PumpApp on WidgetTester {
  Future<void> pumpApp(
    Widget widget, {
      Widget Function(BuildContext context, Widget? child)? builder,
    }) {
    return pumpWidget(
      AppTheme(
        data: AppThemeData.main(),
        child: Builder(
          builder: (context) => MaterialApp(
            theme: AppTheme.of(context).materialTheme,
            localizationsDelegates: const [
              AppLocalizations.delegate,
              GlobalMaterialLocalizations.delegate,
            ],
            builder: builder,
            supportedLocales: AppLocalizations.supportedLocales,
            home: widget,
          ),
        ),
      ),
    );
}
```

Support different screen sizes

- You need to change the size of your screen
- Usually we agree with the client which screen they want us to test the app on
- You can change the screen size easily

```
const iPhone11Max = ScreenSize('iPhone_11_Max', 414, 896, 3);

final responsiveVariant = ValueVariant<ScreenSize>({
  iPhone11Max,
  ...
});

extension ScreenSizeManager on WidgetTester {
  Future<void> setScreenSize(ScreenSize screenSize) async {
    return _setScreenSize(
      width: screenSize.width,
      height: screenSize.height,
      pixelDensity: screenSize.pixelDensity,
    );
  }

  Future<void> _setScreenSize({
    double width = 540,
    double height = 960,
    double pixelDensity = 1,
  }) async {
    final size = Size(width, height);
    await binding.setSurfaceSize(size);
    binding.window.physicalSizeTestValue = size;
    binding.window.devicePixelRatioTestValue = pixelDensity;
  }
}
```

Custom extension for size

- You can reuse the `testResponsiveWidgets` to create goldens according to the current size

```
void testResponsiveWidgets(
  String description,
  WidgetTesterCallback callback, {
  Future<void> Function(String sizeName, WidgetTester tester)?
  goldenCallback,
  test_package.Timeout? timeout,
  bool semanticsEnabled = true,
}) {
  final variant = breakpoints ?? responsiveVariant;
  testWidgets(
    description,
    (tester) async {
      await tester.setScreenSize(variant.currentValue!);
      await callback(tester);
      if (goldenCallback != null) {
        await goldenCallback(variant.currentValue!.name, tester);
      }
    },
    skip: skip,
    timeout: timeout,
    semanticsEnabled: semanticsEnabled,
    variant: responsiveVariant,
  );
}
```

Test your navigation

- Be sure you are redirected at the right place
- Be sure you are displaying the right page



Example on GoRouter

- The easiest way is to Mock GoRouter and inject it to test the redirection

```
import 'package:flutter/material.dart';
import 'package:go_router/go_router.dart';
import 'package:go_router/src/inherited_go_router.dart';

import 'package:mocktail/mocktail.dart';

class MockGoRouter extends Mock implements GoRouter {}

class MockGoRouterProvider extends StatelessWidget {
  const MockGoRouterProvider({
    required this.goRouter,
    required this.child,
    Key? key,
  }) : super(key: key);

  /// The mock navigator used to mock navigation calls.
  final MockGoRouter goRouter;

  /// The child [Widget] to render.
  final Widget child;

  @override
  Widget build(BuildContext context) => InheritedGoRouter(
    goRouter: goRouter,
    child: child,
  );
}
```

Example on GoRouter

- Then you can use it directly in your tests

```
await tester.pumpWidget(
  MaterialApp(
    home: MockGoRouterProvider(
      goRouter: mockGoRouter,
      child: ChangeNotifierProvider.value(
        value: loginInfo,
        child: FamilyScreen(family: Families.data[0]),
      ),
    ),
  ),
);

await tester.tap(find.byType(ListTile).first);
await tester.pumpAndSettle();

verify(() => mockGoRouter.go('/family/f1/person/p1')).called(1);
verifyNever(() => mockGoRouter.go('/family/f1/person/p2'));
```

Example on GoRouter

- To test you redirect your user to the correct place, you need to make `initialLocation` injectable

```
GoRouter routerBuilder(BuildContext context, [String? location]) => GoRouter(  
  initialLocation: location ?? '/',  
  routes: [  
    GoRoute(  
      path: '/',  
      ...  
    )  
  ]  
);
```

**Architecture your project better
thanks to testing**

Which code smell can detect testing?

- You need to repeat yourself to test your code
- You cannot mock a certain dependency
- You need to write a lot of mocks just for one test
- You cannot test certain parts of your code

Example with Flutter Bloc

- You need to create your blocTest

```
bloctest<AppBloc, AppState>(
  'emits unauthenticated when user is empty',
  setUp: () {
    when(() => authenticationRepository.user).thenAnswer(
      (_) => Stream.value(User.empty),
    );
  },
  build: () => AppBloc(
    authenticationRepository: authenticationRepository,
  ),
  expect: () => const [AppState.unauthenticated()],
);
```

Example with Flutter Bloc

- You separate your UI between Screen and View
- Screen injects your Bloc/Cubit
- View is only responsible of displaying and reacting to changes
- Multiples Widgets are totally separated from business logic

```
testWidgets('renders $AppView', (tester) async {  
  await tester.pumpWidget(  
    AppScreen(  
      authenticationRepository: authenticationRepository,  
      shouldDisplayOnboarding: false,  
    ),  
  );  
  await tester.pump();  
  expect(find.byType(AppView), findsOneWidget);  
});
```

Example with Flutter Bloc

- You separate your UI between Screen and View
- Screen injects your Bloc/Cubit
- View is only responsible of displaying and reacting to changes
- Multiples Widgets are totally separated from business logic

```
testWidgets('render $AppView states', (tester) async {  
  await tester.setScreenSize(desktop);  
  
  final appBloc = MockAppBloc();  
  
  when(() => appBloc.state).thenReturn(  
    AppInitialState(),  
  );  
  
  final expectedStates = [AppLoaded()];  
  
  whenListen(appBloc, Stream.fromIterable(expectedStates));  
  
  await tester.pumpApp(  
    BlocProvider<AccountLegalBloc>.value(  
      value: appBloc,  
      child: const AppView(),  
    ),  
  );  
});
```

Example with Flutter Bloc

- You separate your UI between Screen and View
- Screen injects your Bloc/Cubit
- View is only responsible of displaying and reacting to changes
- Multiples Widgets are totally separated from business logic

```
testWidgets('render $AppView states', (tester) async {  
  await tester.setScreenSize(desktop);  
  
  final appBloc = MockAppBloc();  
  
  when(() => appBloc.state).thenReturn(  
    AppInitialState(),  
  );  
  
  final expectedStates = [AppLoaded()];  
  
  whenListen(appBloc, Stream.fromIterable(expectedStates));  
  
  await tester.pumpApp(  
    BlocProvider<AccountLegalBloc>.value(  
      value: appBloc,  
      child: const AppView(),  
    ),  
  );  
});
```

100%

Is not required for a project

Thanks for listening!

Guillaume Bernos