An introduction to integration testing

Contents

Unit tests and widget tests are handy for testing individual classes, functions, or widgets. However, they generally don't test how individual pieces work together as a whole or capture the performance of an application running on a real device. These tasks are performed with *integration tests*.

Integration tests work as a pair: first, deploy an instrumented application to a real device or emulator and then "drive" the application from a separate test suite, checking to make sure everything is correct along the way.

To create this test pair, use the flutter_driver package. It provides tools to create instrumented apps and drive those apps from a test suite.

In this recipe, learn how to test a counter app. It demonstrates how to setup integration tests, how to verify specific text is displayed by the app, how to tap specific widgets, and how to run integration tests.

This recipe uses the following steps:

1. Create an app to test.
2. Add the `flutter_driver` dependency.
3. Create the test files.
4. Instrument the app.
5. Write the integration tests.
6. Run the integration test.

1. Create an app to test

First, create an app for testing. In this example, test the counter app produced by the `flutter create` command. This app allows a user to tap on a button to increase a counter.

Furthermore, provide a `ValueKey` to the `Text` and `FloatingActionButton` widgets. This allows identifying and interacting with these specific widgets inside the test suite.

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

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

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

class MyHomePage extends StatefulWidget {
  MyHomePage({Key key, this.title}) : super(key: key);

  final String title;

  @override
  _MyHomePageState 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>[
            Text(
              'You have pushed the button this many times:',
            ),
            Text(
              '$_counter',
              // Provide a Key to this specific Text widget. This allows
              // identifing the widget from inside the test suite,
              // and reading the text.
              key: Key('counter'),
              style: Theme.of(context).textTheme.display1,
            ),
```

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

2. Add the `flutter_driver` dependency

Next, use the `flutter_driver` package to write integration tests. Add the `flutter_driver` dependency to the `dev_dependencies` section of the apps's `pubspec.yaml` file.

Also add the `test` dependency in order to use actual test functions and assertions.

```
dev_dependencies:
  flutter_driver:
    sdk: flutter
  test: any
```

3. Create the test files

Unlike unit and widget tests, integration test suites do not run in the same process as the app being tested. Therefore, create two files that reside in the same directory. By convention, the directory is named `test_driver`.

1. The first file contains an "instrumented" version of the app. The instrumentation allows you to "drive" the app and record performance profiles from a test suite. This file can have any name that makes sense. For this example, create a file called `test_driver/app.dart`.
2. The second file contains the test suite, which drives the app and verifies it works as expected. The test suite also records performance profiles. The name of the test file must correspond to the name of the file that contains the instrumented app, with `_test` added at the end. Therefore, create a second file called `test_driver/app_test.dart`.

This creates the following directory structure:

```
counter_app/
  lib/
    main.dart
  test_driver/
    app.dart
    app_test.dart
```

4. Instrument the app

Now, instrument the app. This involves two steps:

1. Enable the flutter driver extensions

2. Run the app

Add this code inside the `test_driver/app.dart` file.

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

void main() {
  // This line enables the extension.
  enableFlutterDriverExtension();

  // Call the `main()` function of the app, or call `runApp` with
  // any widget you are interested in testing.
  app.main();
}
```

5. Write the tests

Now that you have an instrumented app, you can write tests for it. This involves four steps:

1. Create `SeralizableFinders` to locate specific widgets
2. Connect to the app before our tests run in the `setUpAll()` function
3. Test the important scenarios
4. Disconnect from the app in the `teardownAll()` function after the tests complete

```dart
// Imports the Flutter Driver API.
import 'package:flutter_driver/flutter_driver.dart';
import 'package:test/test.dart';

void main() {
  group('Counter App', () {
    // First, define the Finders and use them to locate widgets from the
    // test suite. Note: the Strings provided to the `byValueKey` method must
    // be the same as the Strings we used for the Keys in step 1.
    final counterTextFinder = find.byValueKey('counter');
    final buttonFinder = find.byValueKey('increment');

    FlutterDriver driver;

    // Connect to the Flutter driver before running any tests.
    setUpAll(() async {
      driver = await FlutterDriver.connect();
    });

    // Close the connection to the driver after the tests have completed.
    tearDownAll(() async {
      if (driver != null) {
        driver.close();
      }
    });

    test('starts at 0', () async {
      // Use the `driver.getText` method to verify the counter starts at 0.
      expect(await driver.getText(counterTextFinder), "0");
    });

    test('increments the counter', () async {
      // First, tap the button.
      await driver.tap(buttonFinder);

      // Then, verify the counter text is incremented by 1.
      expect(await driver.getText(counterTextFinder), "1");
    });
  });
}
```

6. Run the tests

Now that you have an instrumented app and a test suite, run the tests. First, be sure to launch an Android Emulator, iOS Simulator, or connect your computer to a real iOS / Android device.

Then, run the following command from the root of the project:

```
flutter drive --target=test_driver/app.dart
```

This command:

1. Builds the `--target` app and installs it on the emulator / device.
2. Launches the app.
3. Runs the `app_test.dart` test suite located in `test_driver/` folder.

⟨ Take a picture using the camera Performance profiling ⟩