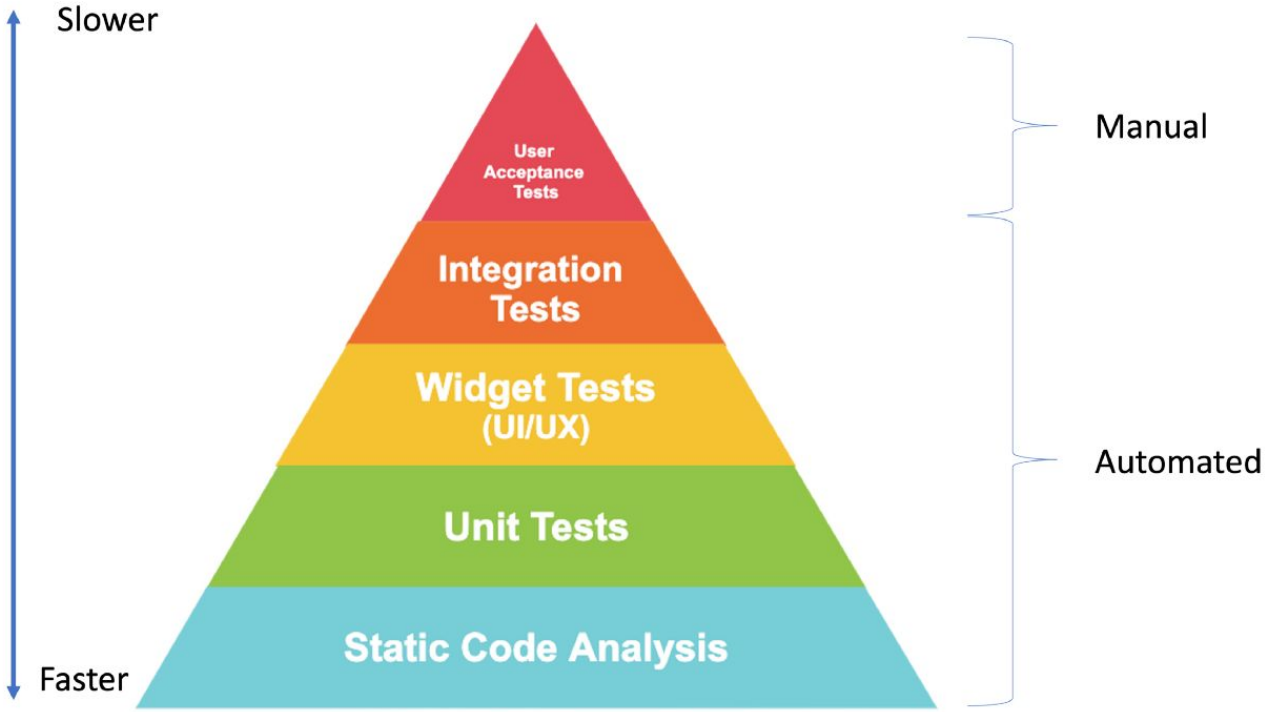
LeanCode

Programming Mobile Applications in Flutter

Testing 2 & Code Generation



Source: https://njkhanh.com/understand-the-core-concept-of-test-in-flutter-to-write-a-fewer-errors-of-flutter-app-p5f34353335

Widget Tests

Checks if single widget works as expected

Widget Tests

The **flutter_test** package provides the following tools for testing widgets:

- The *WidgetTester* allows building and interacting with widgets in a test environment
- The *testWidgets()* function automatically creates a new *WidgetTester* for each test case, and is used in place of the normal *test()* function
- The *Finder* classes allow searching for widgets in the test environment
- Widget-specific *Matcher* constants help verify whether a *Finder* locates a widget or multiple widgets in the test environment

```dart
void main() {
  testWidgets('Counter increments smoke test', (WidgetTester tester) async {
    // Build our app and trigger a frame.
    await tester.pumpWidget(const MyApp());

    // Verify that our counter starts at 0.
    expect(find.text('0'), findsOneWidget);
    expect(find.text('1'), findsNothing);

    // Tap the '+' icon and trigger a frame.
    await tester.tap(find.byIcon(Icons.add));
    await tester.pump();

    // Verify that our counter has incremented.
    expect(find.text('0'), findsNothing);
    expect(find.text('1'), findsOneWidget);
  });
}
```

```dart
class SampleWidget extends StatelessWidget {
  const SampleWidget({
    Key? key,
    required this.title,
    required this.message,
  }) : super(key: key);

  final String title;
  final String message;

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

```dart
void main() {

  testWidgets('SampleWidget has a title and message', (WidgetTester tester) async {
    await tester.pumpWidget(const SampleWidget(title: 'T', message: 'M'));

    final titleFinder = find.text('T');
    final messageFinder = find.text('M');

    expect(titleFinder, findsOneWidget);
    expect(messageFinder, findsOneWidget);
  });
}
```

LeanCode

```
class Sample2Widget extends StatelessWidget {
  const Sample2Widget({
    Key? key,
    required this.cubit,
  }) : super(key: key);

  final SampleCubit cubit;

  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'Flutter Demo',
      home: Scaffold(
        appBar: AppBar(
          title: const Text('Sample2'),
        ),  // AppBar
        body: BlocBuilder<SampleCubit, String?>(
          bloc: cubit,
          builder: (context, message) {
            return Center(
              child: message != null
                  ? Text(message)
                  : const CircularProgressIndicator(),
            );  // Center
          },
        ),  // BlocBuilder
      ),  // Scaffold
    );  // MaterialApp
  }
}

class SampleCubit extends Cubit<String?> {
  SampleCubit() : super(null);

  void changeMessage(String message) => emit(message);
}
```

```dart
class SampleCubitMock extends MockCubit<String?> implements SampleCubit {}

void main() {
  group("Sample2WidgetTest", () {
    late SampleCubit sampleCubit;

    setUp(() {
      sampleCubit = SampleCubitMock();
    });

    testWidgets('When message is empty Should show progress indicator',
        (WidgetTester tester) async {
      //when
      await tester.pumpWidget(Sample2Widget(cubit: sampleCubit));
      //then
      final progressBarFinder = find.byType(CircularProgressIndicator);
      expect(progressBarFinder, findsOneWidget);
    });

    testWidgets('When message is not empty Should show message',
        (WidgetTester tester) async {
      //given
      const message = "sampleMessage";
      whenListen(sampleCubit, Stream.value(message));
      //when
      await tester.pumpWidget(Sample2Widget(cubit: sampleCubit));
      await tester.pump();
      //then
      final progressBarFinder = find.byType(CircularProgressIndicator);
      final messageFinder = find.text(message);
      expect(progressBarFinder, findsNothing);
      expect(messageFinder, findsOneWidget);
    });
  });
}
```

Golden/Snapshot Tests

A type of output comparison testing

Golden Tests

The term **golden file** refers to a master image that is considered the true rendering of a given widget, state, application, or other visual representation you have chosen to capture.

The master golden image files that are tested against can be created or updated by running *flutter test --update-goldens* on the test.

```dart
class SampleCubitMock extends MockCubit<String?> implements SampleCubit {}

void main() {
  group("SampleWidget Golden", () {
    late SampleCubit sampleCubit;

    setUp(() {
      sampleCubit = SampleCubitMock();
    });

    testWidgets('When message is empty Should show progress indicator',
            (WidgetTester tester) async {
          //when
          await tester.pumpWidget(Sample2Widget(cubit: sampleCubit));
          //then
          expect(find.byType(Sample2Widget), matchesGoldenFile('Sample2Loading.png'));
        });

    testWidgets('When message is not empty Should show message',
            (WidgetTester tester) async {
          //given
          const message = "sampleMessage";
          whenListen(sampleCubit, Stream.value(message));
          //when
          await tester.pumpWidget(Sample2Widget(cubit: sampleCubit));
          await tester.pump();
          //then
          expect(find.byType(Sample2Widget), matchesGoldenFile('Sample2Message.png'));
        });
  });
}
```

Demo

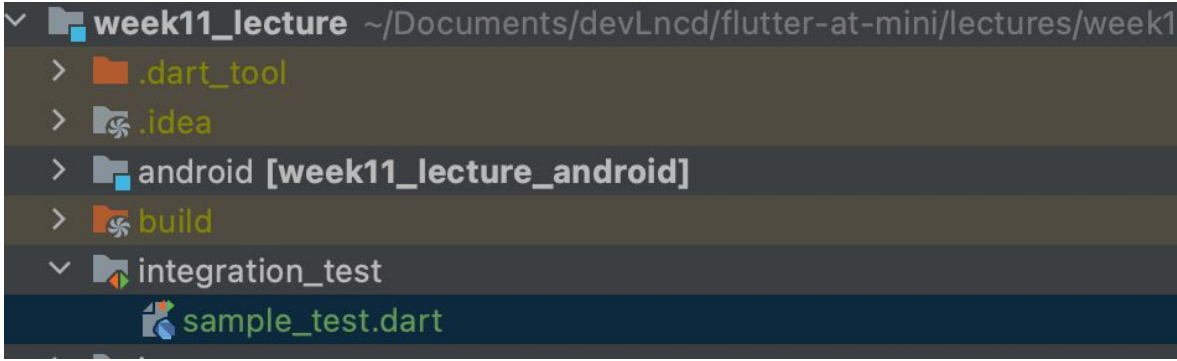
Integration Tests

Integration Tests

- 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.
- **Run on real device!!**
- Firebase Test Lab

Integration Tests

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



Integration Tests

```
@override
Widget build(BuildContext context) {
  return Scaffold(
    appBar: AppBar(
      title: Text(widget.title),
    ),  // AppBar
    body: Center(
      child: Column(
        mainAxisAlignment: MainAxisAlignment.center,
        children: <Widget>[
          const Text(
            'You have pushed the button this many times:',
          ),  // Text
          Text(
            '$_counter',
            style: Theme.of(context).textTheme.headline4,
          ),  // Text
        ],  // <Widget>[]
      ),  // Column
    ),  // Center
    floatingActionButton: FloatingActionButton(
      onPressed: _incrementCounter,
      tooltip: 'Increment',
      child: const Icon(Icons.add),
    ),  // FloatingActionButton
  );  // Scaffold
}
```

```
void main() {
  IntegrationTestWidgetsFlutterBinding.ensureInitialized();

  group('end-to-end test', () {
    testWidgets('tap on the floating action button, verify counter',
            (WidgetTester 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);
        });
  });
}
```

Integration Tests Resources

- https://docs.flutter.dev/cookbook/testing/integration/introduction
- https://docs.flutter.dev/testing/integration-tests

Demo

Code Generation

Code Generation

- Why?
- Improve code quality:
  - Reduce boilerplate
  - Make the code more readable
  - Reduce the number of possible bugs
- When?
  - Data classes - classes whose main purpose is to hold data
  - Sealed classes - used to restrict the users from inheriting the class
  - Architecture boilerplate - *MobX*
  - Common features/functions - *fromJson(), toJson(), copyWith()*

Code Generation

- **build_runner** - A *dev_dependency* that allows the generation of files from Dart code.
- ***flutter pub run build_runner build*** - Runs a single build and exits
- Annotations
- More resources about code generation:
  - Code generation in Dart: the basics
  - source_gen

Code Generation - json_serializable

```yaml
dependencies:
  flutter:
    sdk: flutter

  cupertino_icons: ^1.0.2
  flutter_bloc: ^8.0.0
  json_annotation: ^4.4.0

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

  flutter_lints: ^1.0.0
  mocktail: ^0.2.0
  bloc_test: ^9.0.1
  build_runner: ^2.1.7
  json_serializable: ^6.1.3
```

Code Generation - json_serializable

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

/// This allows the `User` class to access private members in
/// the generated file. The value for this is *.g.dart, where
/// the star denotes the source file name.
part 'user.g.dart';

/// An annotation for the code generator to know that this class needs the
/// JSON serialization logic to be generated.
@JsonSerializable()
class User {
  User(this.name, this.email);

  String name;
  String email;

  /// A necessary factory constructor for creating a new User instance
  /// from a map. Pass the map to the generated `_$UserFromJson()` constructor.
  /// The constructor is named after the source class, in this case, User.
  factory User.fromJson(Map<String, dynamic> json) => _$UserFromJson(json);

  /// `toJson` is the convention for a class to declare support for serialization
  /// to JSON. The implementation simply calls the private, generated
  /// helper method `_$UserToJson`.
  Map<String, dynamic> toJson() => _$UserToJson(this);
}
```

Code Generation - json_serializable

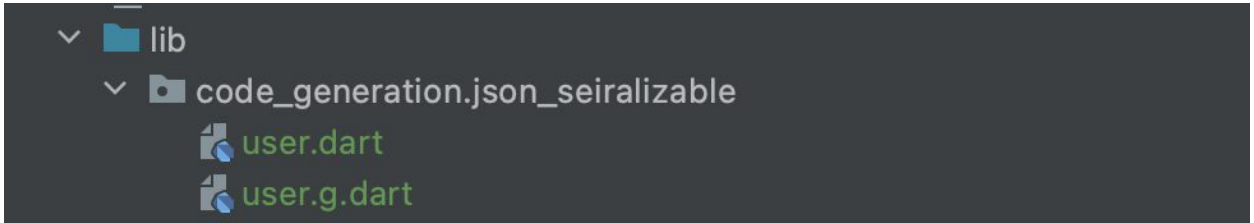
```dart
// GENERATED CODE - DO NOT MODIFY BY HAND

part of 'user.dart';

// **************************************************************************
// JsonSerializableGenerator
// **************************************************************************

User _$UserFromJson(Map<String, dynamic> json) => User(
      json['name'] as String,
      json['email'] as String,
    );

Map<String, dynamic> _$UserToJson(User instance) => <String, dynamic>{
      'name': instance.name,
      'email': instance.email,
    };
```



Freezed

Freezed

Using **Freezed**, you will get:

- A simple and concise syntax for defining models, where we don't need to define both a constructor and a property. Instead, we only need to define the constructor, removing unnecessary duplication
- A *copyWith* method, for cloning objects with different values
- **union-types/pattern matching,** for making impossible states impossible
- An automatic serialization/deserialization of your objects (including union types)
- A default **==/toString** implementation which respectively compares/shows all properties of the object

Source: https://pub.dev/packages/freezed#motivation

Freezed

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

part 'person.freezed.dart';

@freezed
class Person with _$Person {
  @Assert('age >= 0')
  factory Person({ required String name, @Default(18) int? age }) = _Person;
}
```

Demo

Freezed + json_serializable

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

part 'person.freezed.dart';
part 'person.g.dart';

@freezed
class Person with _$Person {
  @Assert('age >= 0')
  factory Person({ required String name, @Default(18) int age }) = _Person;

  factory Person.fromJson(Map<String, dynamic> json) => _$PersonFromJson(json);
}
```

Freezed - Unions/Sealed classes

```
import 'package:freezed_annotation/freezed_annotation.dart';
import 'package:week11_lecture/code_generation/freezed/person.dart';

part 'person_state.freezed.dart';

@freezed
class PersonState with _$PersonState {
  const factory PersonState.success(Person person) = _Success;

  const factory PersonState.error(String errorText) = _Error;

  const factory PersonState.loading() = _Loading;
}
```

Freezed - Unions/Sealed classes

```
final personState = PersonState.success(nextPerson);

personState.when(success: (person) {
  print(person);
}, error: (error) {
  print(error);
}, loading: () {
  print("Loading");
});

personState.maybeWhen(orElse: () {});

personState.maybeMap(orElse: () {});
```

Demo

Questions?