LeanCode

Testing in Flutter

A method to check whether the actual software product matches expected requirements and to ensure that software product is defect free

Benefits

- Customer satisfaction
- Quality
- Cost effectiveness

Types

- Functional
  - Unit
  - Integration
  - User Acceptance
  - ...
- Non-Functional
  - Performance
  - Scalability
  - Usability
  - ...
- Maintenance
  - Regression

Boxes

White box testing

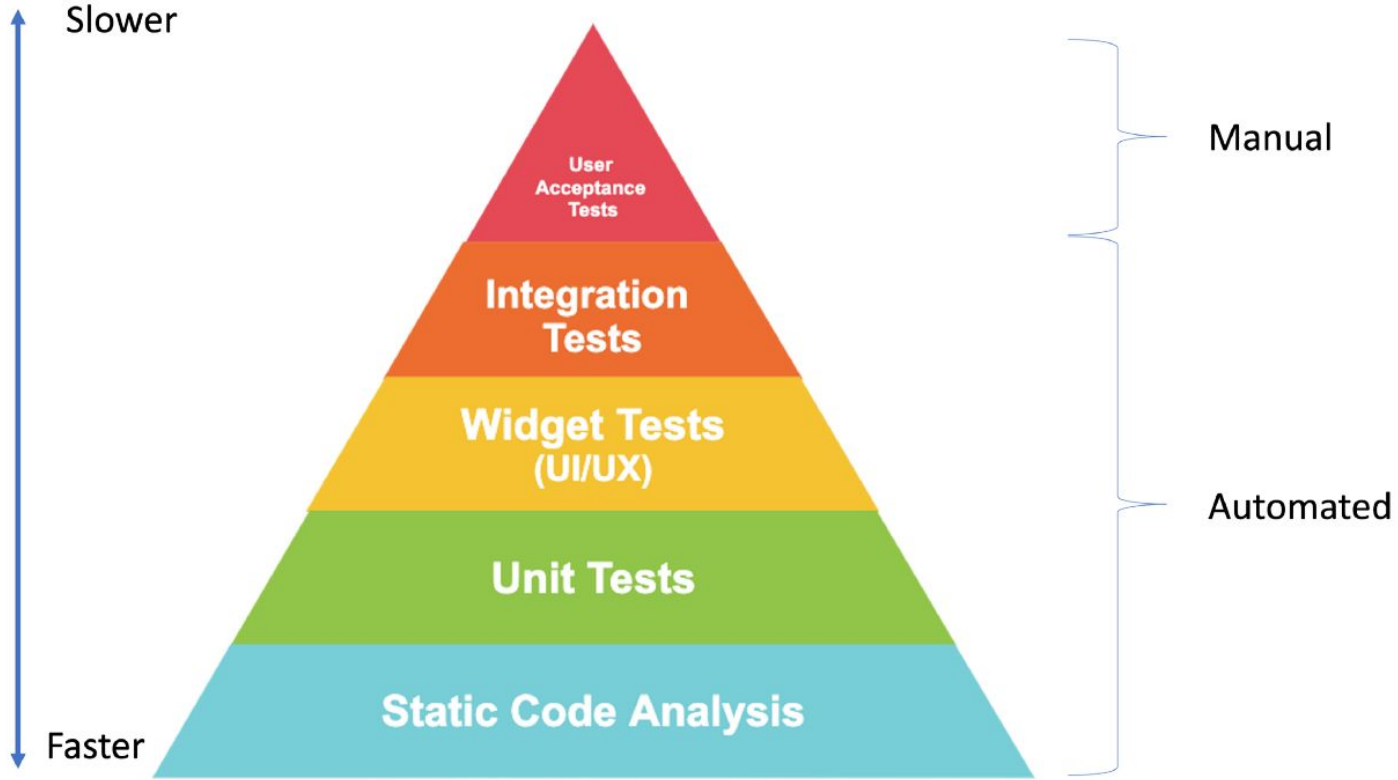
A software testing technique in which **internal structure, design and coding of software** are tested to verify flow of input-output and to improve design, usability and security. In white box testing, code is visible to testers. One of the goals of white box testing in software engineering is to verify all the decision branches, loops, statements in the code.

Black box testing

A software testing method in which the functionalities of software applications are tested **without having knowledge of internal code structure, implementation details and internal paths**. Black Box Testing mainly focuses on input and output of software applications and it is entirely based on software requirements and specifications.

Gray box testing

Software testing technique to test a software product or application with **partial knowledge of internal structure of the application**. The purpose of grey box testing is to search and identify the defects due to improper code structure or improper use of applications.



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

Static code analysis

Lint rules

- errors
- style
- pub

https://dart-lang.github.io/linter/lints/index.html

Configs:

- flutter_lints
- leancode_lint

Custom lints

- analyzer_plugin
- custom_lint package

You can write *your own* rules for *your* project.

Unit tests

Unit tests are handy for verifying the behavior of a single function, method, or class.

Unit Tests Rules

- Short, quick, and automated
- They test specific functionality of a method or class
- Deterministic
- Have a clear **pass/fail** condition

A necessary trivial example

```dart
class Counter {
  int _value = 0;

  int get value => _value;

  void increment() => _value++;
  void decrement() => _value--;
}
```

```dart
import 'package:flutter_test/flutter_test.dart';
import 'package:testing/counter.dart';

Run | Debug
void main() {
  Run | Debug
  group('Counter', () {
    Run | Debug
    test('value should start at 0', () {
      final counter = Counter();

      expect(counter.value, 0);
    });

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

      counter.increment();

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

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

      counter.decrement();

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

Testing blocs

```dart
class CounterCubit extends Cubit<int> {
  CounterCubit() : super(0);

  void increment() => emit(state + 1);
  void decrement() => emit(state - 1);
}
```

```dart
import 'package:bloc_test/bloc_test.dart';
import 'package:flutter_test/flutter_test.dart';
import 'package:testing/counter.dart';

Run | Debug
void main() {
  Run | Debug
  group('CounterCubit', () {
    late CounterCubit counterCubit;

    setUp(() {
      counterCubit = CounterCubit();
    });

    Run | Debug
    test('initial state is 0', () {
      expect(counterCubit.state, 0);
    });

    Run | Debug
    blocTest<CounterCubit, int>(
      'emits [1] when CounterEvent.increment is added',
      build: () => counterCubit,
      act: (bloc) => bloc.increment(),
      expect: () => [1],
    );

    Run | Debug
    blocTest<CounterCubit, int>(
      'emits [-1] when CounterEvent.decrement is added',
      build: () => counterCubit,
      act: (bloc) => bloc.decrement(),
      expect: () => [-1],
    );
  });
}
```

```dart
class OrdersCubit extends Cubit<OrdersState> {
  OrdersCubit(this._ordersRepository) : super(OrdersLoading());

  final OrdersRepository _ordersRepository;

  Future<void> fetch() async {
    try {
      final response = await _ordersRepository.fetchOrders();
      emit(OrdersLoaded(response.toList()));
    } catch (e) {
      emit(const OrdersError());
    }
  }
}

sealed class OrdersState {
  const OrdersState();
}

class OrdersLoading extends OrdersState {}

class OrdersLoaded extends OrdersState {
  const OrdersLoaded(this.orders);

  final List<Order> orders;
}

class OrdersError extends OrdersState {
  const OrdersError();
}
```

Mocks

Mocking libraries

https://pub.dev/packages/mockito - based on code generation
https://pub.dev/packages/mocktail - without code generation, easier to use, recommended

```dart
void main() {
  Run | Debug
  group('OrdersCubit', () {
    late _MockOrdersRepository repository;

    setUp(() {
      repository = _MockOrdersRepository();
    });

    OrdersCubit buildCubit() => OrdersCubit(repository);

    blocTest(
      'Emits success state with matches',
      build: () {
        when(() => repository.fetchOrders())
            .thenAnswer((_) async => [mockOrder]);

        return buildCubit();
      },
      act: (cubit) => cubit.fetch(),
      expect: () => [
        OrdersLoaded([mockOrder]),
      ],
    );
  });
}

class _MockOrdersRepository extends Mock implements OrdersRepository {}
```

```
blocTest(
  'Emits error state when the request fails',
  build: () {
    when(() => repository.fetchOrders()).thenThrow(Exception());

    return buildCubit();
  },
  act: (cubit) => cubit.fetch(),
  expect: () => [
    const OrdersError(),
  ],
);
```

```
test('Requests Bluetooth permission', () {
  setBluetoothPermissionNotRequested();

  bluetoothClient.connect();

  verify(permissionService.requestBluetooth).called(1);
});
```

Tests are code.

Tests are code.
Code should be clean.

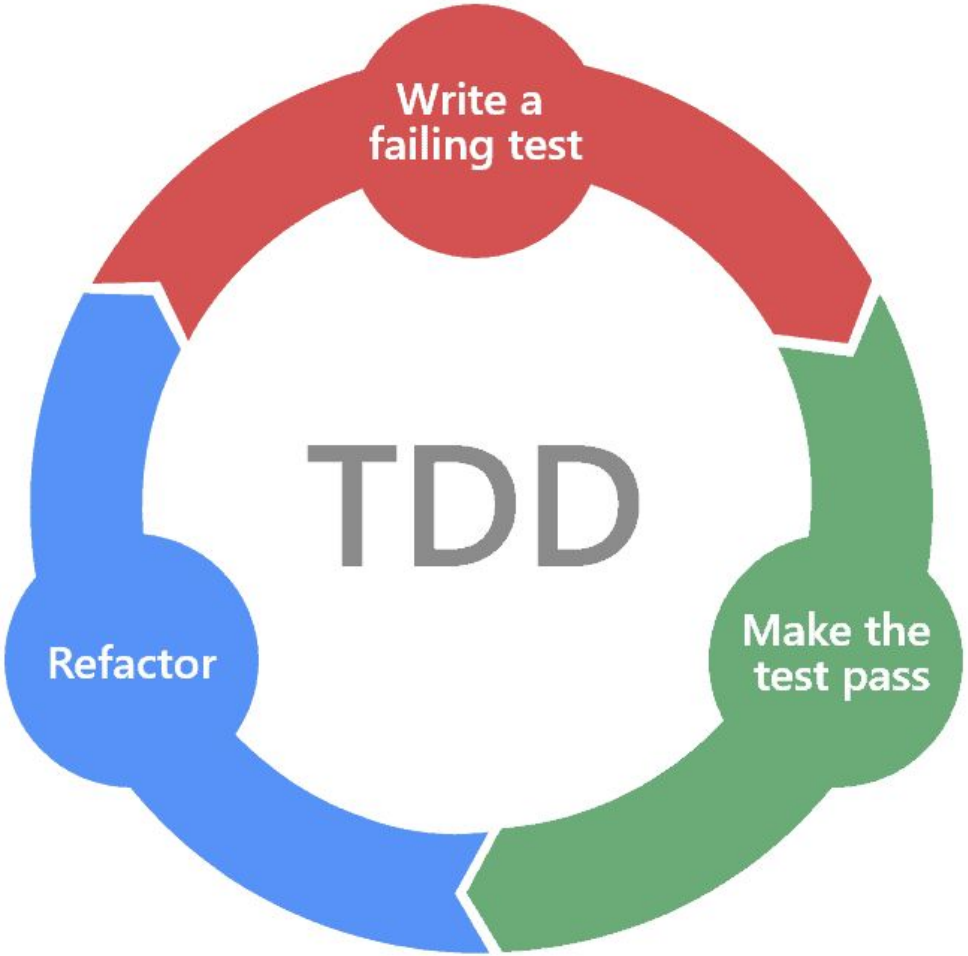
Best practices

- Try to avoid blindly copy-pasting tests and duplicating code
- Ideally aim for tests to be readable for non-technical people
  - Create functions for common setup and assertion operations, e.g. `mockOrdersSuccessResponse`, `mockLocationPermissionStatus`
  - This is not always possible for testing highly technical units, but it's a good benchmark to have
- Extract repeatable testing utils
- When fixing a bug try to write a test proving that the bug exists before fixing the implementation

TDD

Test Driven Development

Write tests first



https://marsner.com/blog/why-test-driven-development-tdd/

BDD

Behavior Driven Development

Behavior-driven development is a testing practice that follows the idea of **specification by example**. The idea is to describe how the application should behave in a very simple user/business-focused language. BDD's business-focused perspective on application behavior allows teams to create living documentation that is easy to maintain and can be consumed by all team members, including testers, developers, and product owners.

Gherkin

```
Feature: Account Holder withdraws cash

Scenario: Account has sufficient funds
    Given The account balance is $100
      And the card is valid
      And the machine contains enough money
     When the Account Holder requests $20
     Then the ATM should dispense $20
      And the account balance should be $80
      And the card should be returned
```

Testing widgets

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

WidgetTester

- `tap`
- `enterText`
- `drag`
- `fling`
- `scrollUntilVisible`

Finders

- `find.byKey(const ValueKey('continue'))`
- `find.text('Back')`
- `find.byType(IconButton)`
-
```
find.descendant(
  of: find.byType(TextField),
  matching: find.text('Key 1'),
)
```
-
```
find.byWidgetPredicate(
  (Widget widget) => widget is Tooltip && widget.message
  == 'Back',
 description: 'widget with tooltip "Back"',
)
```

```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
group('Orders page', () {
  late OrdersRepository repository;

  setUp(() {
    repository = _MockOrdersRepository();
  });
  Toggle Search Details

  void setUpOrdersResponse([List<Order>? orders]) {
    when(() => repository.fetchOrders())
        .thenAnswer((_) async => orders ?? [order1]);
  }

  Future<void> pumpPage(WidgetTester tester) {
    return tester.pumpWidget(
      Provider<OrdersRepository>(
        create: (context) {
          setUpOrdersResponse();
          return repository;
        },
        child: const OrdersPage(),
      ), // Provider
    );
  }

  Run | Debug
  testWidgets('Renders new orders on a pull-to-refresh gesture',
      (tester) async {
    await pumpPage(tester);
    setUpOrdersResponse([order1, order2]);

    await tester.drag(find.byKey(OrdersPage.listKey), const Offset(0, 300));
    await tester.pumpAndSettle();

    expect(find.byType(OrderItem), findsOneWidget);
  });
});
}
```

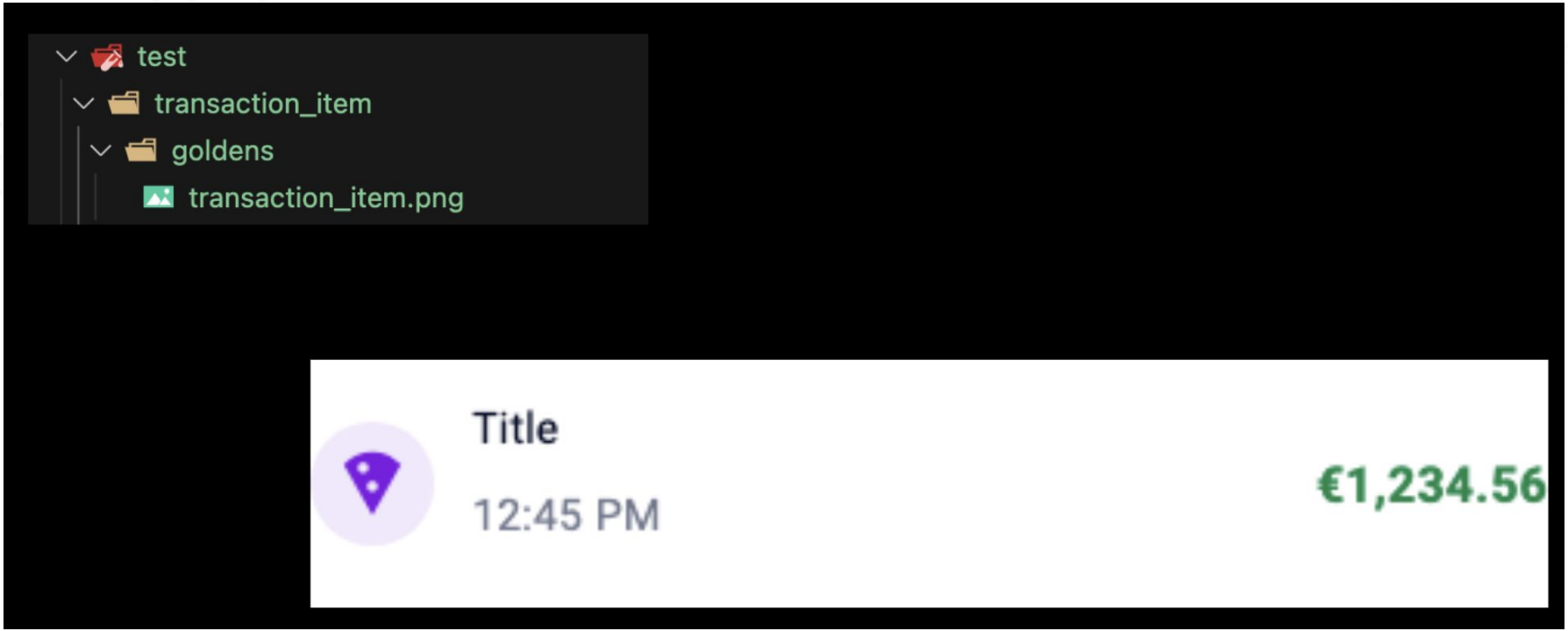
Golden tests

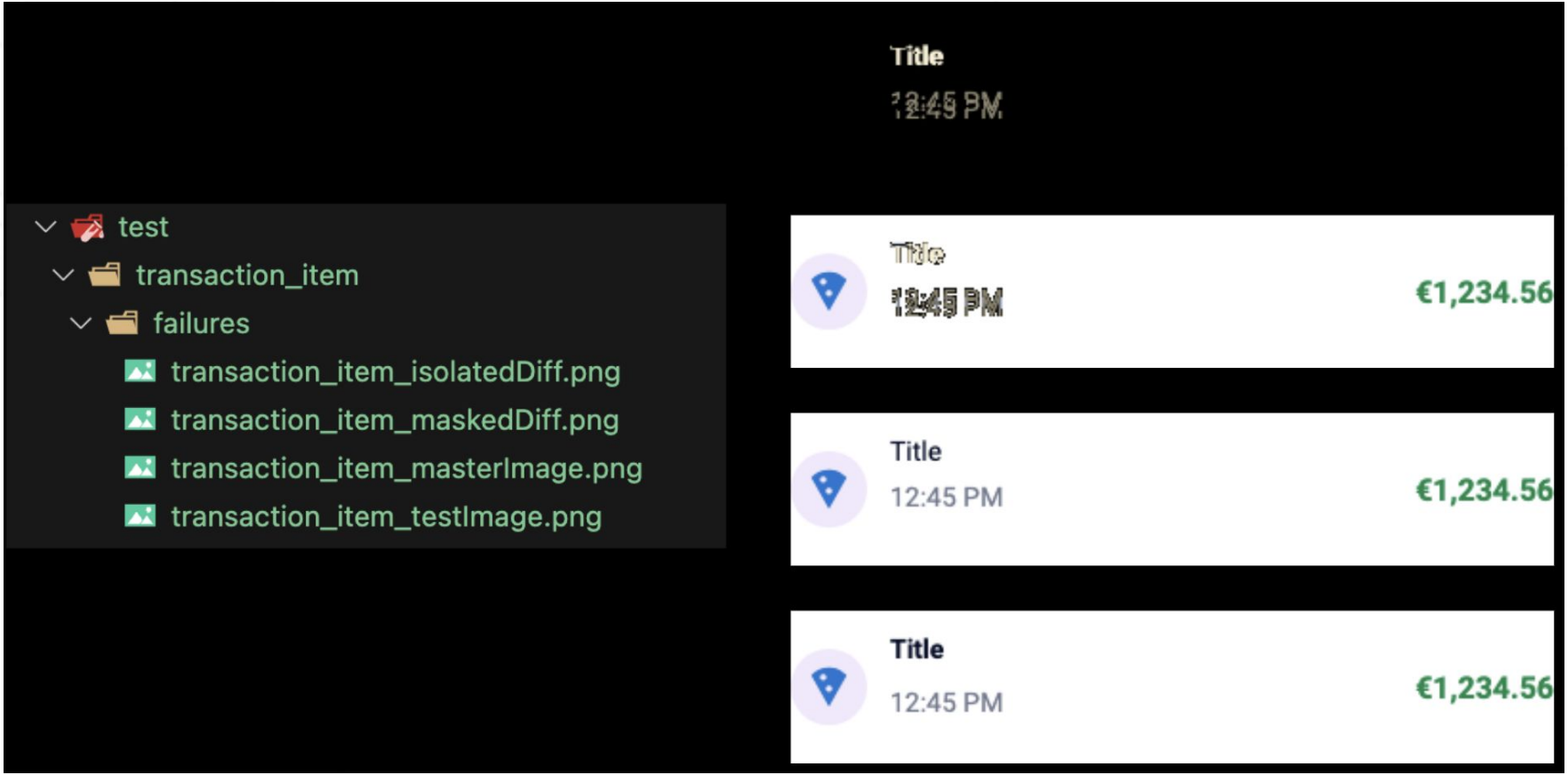
Golden tests (a.k.a. snapshot tests, screenshot tests) are widget tests that compares your widget with an image file and expects that it looks the same.

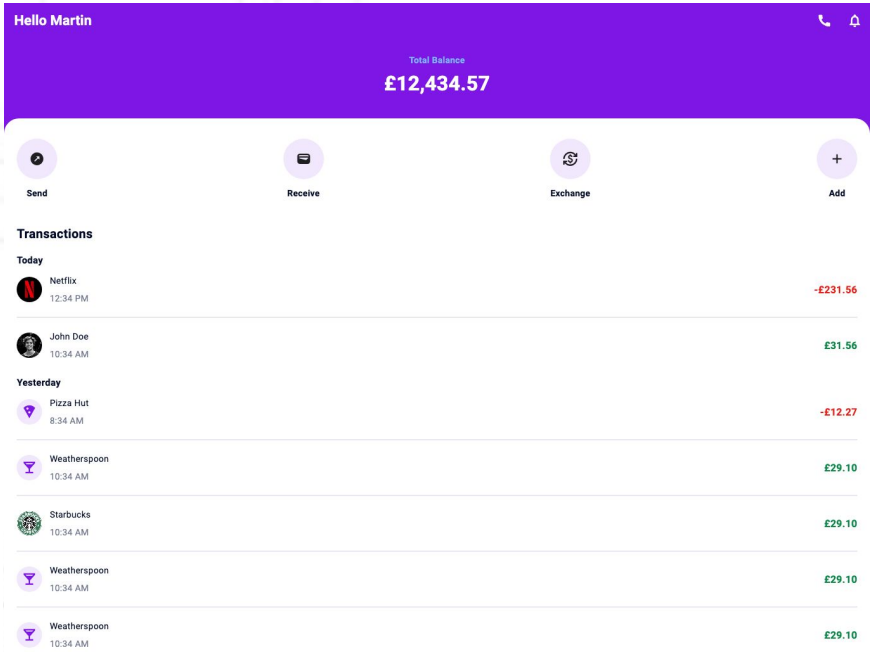
Golden tests

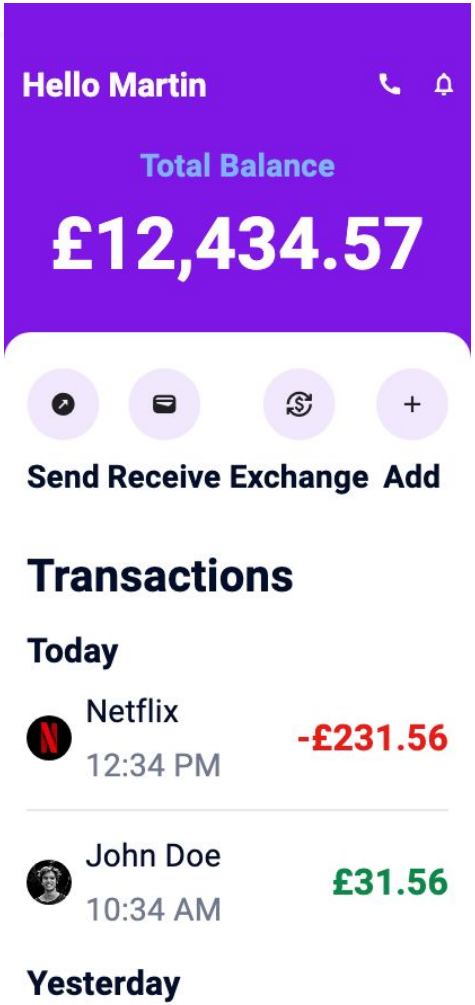
- Golden files are image files that were created from a manually verified widget.
- `flutter test` renders snapshots and compares them with generated files (typically kept in the repo)
- `flutter test --update-goldens` updates generated images (expected changes)
- Golden tests *are* widget tests (you can do drags, taps, text input, etc.)
- https://pub.dev/packages/golden_toolkit - package with useful utilities for golden testing

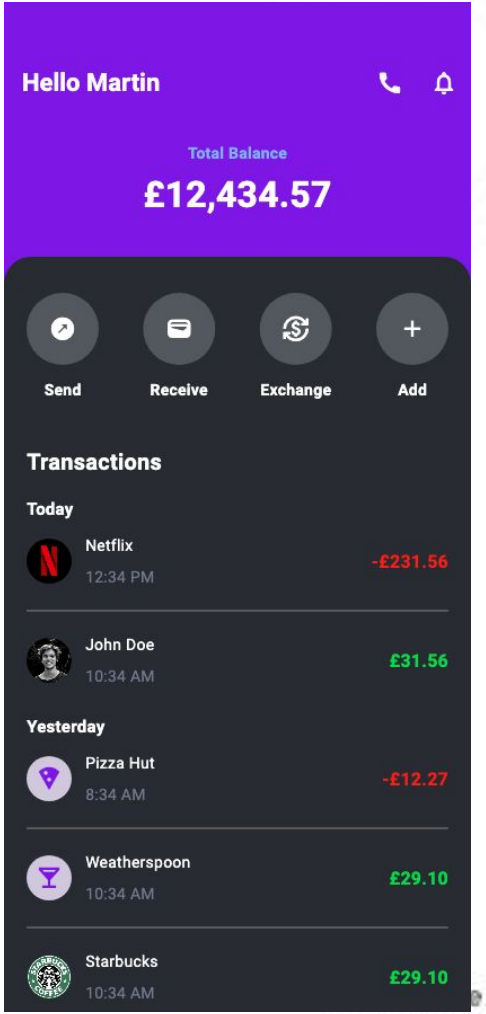
```dart
testWidgets('TransactionItem', (tester) async {
  await tester.pumpWidget(
    TransactionItem.icon(
      name: 'Title',
      amount: 1234.56,
    ),
  );

  await expectLater(
    find.byType(TransactionItem),
    matchesGoldenFile('transaction_item.png'),
  );
});
```











Problems

- Differences in renders between platforms (operating systems, CPU architecture etc.)
- Manual review needs discipline
- Storing golden files (repository, Git LFS, external bucket storage)

Integration tests

Run the whole app

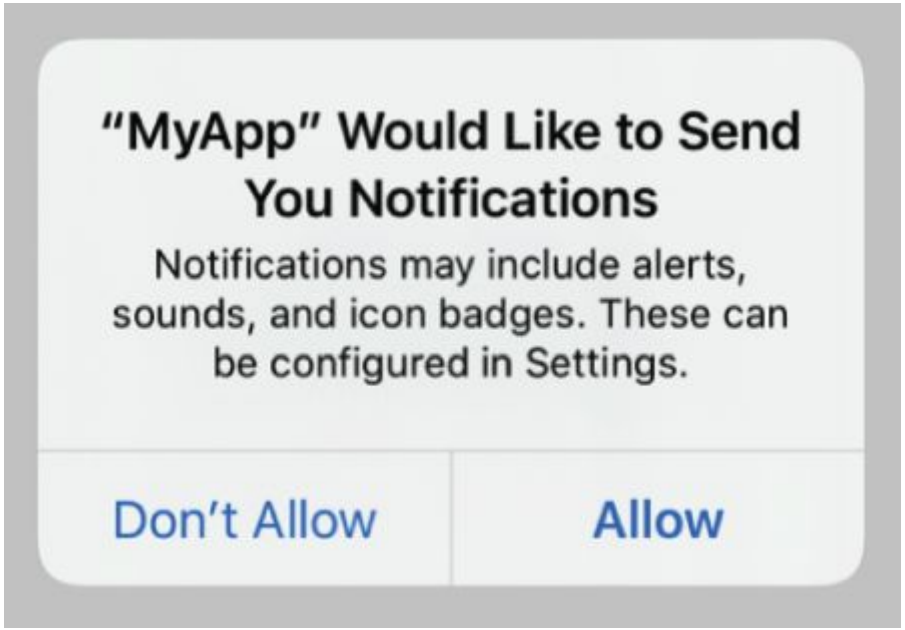
Integration tests

- 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 devices (or emulators)
- Device farms (AWS Device Farm, Firebase Test Lab)

integration_test

```dart
@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
}
```

```dart
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);
        });
  });
}
```


"MyApp" Would Like to Send You Notifications
Notifications may include alerts, sounds, and icon badges. These can be configured in Settings.
Don't Allow
Allow

Appium

- Most renowned cross-platform integration mobile testing framework
- Based on NPM (cannot write tests in Dart)
- Appium Flutter Driver extension makes it possible to access Flutter widget tree (find elements etc.)
- Can access native elements by switching context between Flutter Driver and native
- Generally hard to use and buggy with Flutter (we found many issues and had to submit PRs to Appium Flutter Driver to resolve them)

by LeanCode

We build digital products.

LeanCode

Patrol

- Open source
- Flutter-native, tests in Dart
- Native interactions
- Custom finders inspired by jQuery
- Support for native tooling and device farms

```dart
// in your UI test code

// prepare test conditions
final patrol = Patrol.forTest();
await patrol.enableCelluar();
await patrol.disableWiFi();

// handle native permission request dialog
await patrol.selectFineLocation();
await patrol.grantPermissionWhileInUse();
```

```dart
patrolTest('signs up', ($) async {
 await $.pumpWidgetAndSettle(AwesomeApp());

 await $(#emailTextField).enterText('bartek@leancode.co');
 await $(#nameTextField).enterText('Bartek');
 await $(#passwordTextField).enterText('ny4ncat');
 await $(#termsAndConditionsCheckbox).tap();
 await $(#signUpButton).tap();
 expect(await $('Welcome, Bartek!').waitUntilVisible(), findsOneWidget);
});
```


iPhone 14
iOS 16.0
3:14
Shortcuts
Contacts
Files
Watch
AutomatorServ...
Search

Questions?