slidenumbers: true

A four-layer proven method for testing software - Flutter/Dart.

[.footer: ##### @JamalBelilet (🐳 🐙 📸 👍)]

Table of content

We will start by highlighting few software engineering **designs** & **patterns**.

- Meaningful **unit tests**,
- To **business logic tests**,
- Then **widget and visual tests**,

Depending on the use-cases **a hybrid** of:

- **Integration**
- And **e2e tests**
- Making the most of what **integration_test** provides.

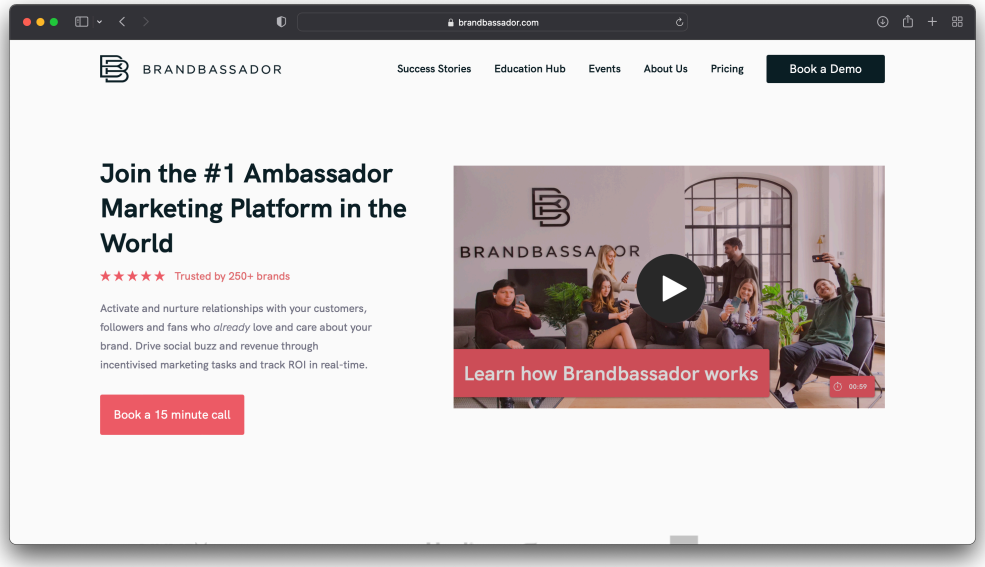
Djamaleddine Belilet (Jamal)

[.footer: ##### @JamalBelilet (🐳 🐙 📸 👍)]

Building new software while **supporting** and **solving problems** of existing products.

Brandbassador is the ultimate ambassador marketing management tool for fast-growing eCommerce brands.

Javascript, **Dart & Flutter**



right fit

Essentials.

Static Analysis Testing:

- Flutter analyse
- Dart code metrics.

Monitoring and error tracking:

- Firebase Crashlytics
- Sentry.io

Static Analysis Testing: **Flutter analyse**

- Thankfully it's part of **VSCode and Android Studio**

- But we need it to be part of **CI/CD** pipelines.

```
> flutter analyze
Analyzing brandbassador_flutter...

info • Don't import implementation files from another package...

1 issue found. (ran in 6.0s)
```

Static Analysis Testing: **Dart code metrics**

- Requires some extra work.
- Out of the box with Codemagic.

```
# Issues
No issues found!
# Anti pattern cases
No anti pattern cases found!
# Metrics
BrandsView.build -> lines-of-code -> 69
BrandsView.build -> maximum-nesting-level -> 4
BrandsView.build -> maintainability-index -> 39
```

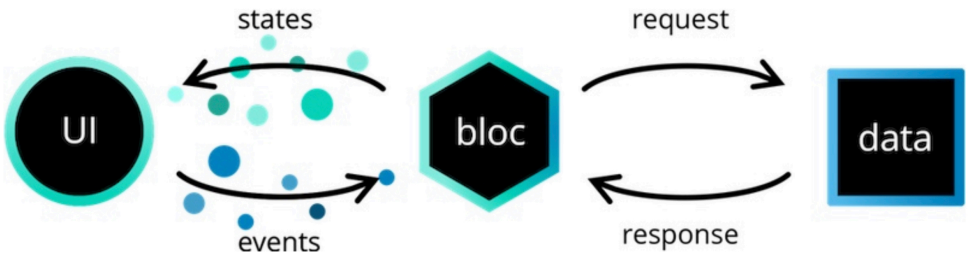
Monitoring and error tracking

- Firebase Crashlytics
- Sentry.io

Design & Patterns

- State management*
- Domain-driven design (Clarifies responsibility)

State management: **BLoC**



inline fit

Domain-driven design

Presentation:

- **Widgets**
- **States**
- And **events**.

Application: **BLoCs**

Domain: **Entities**

Infrastructure:

- **Repositories**
- **Data sources** (remote, local)

right fit

Infrastructure & Unit tests.

- Repository level fully covered.
- Any helper is to evaluate.

Application (Business logic) tests.

- Unit tests & **blocTest**.
- repository **mocked**, stubbing any related behaviour.

Presentation & widget / visual tests.

- Widget tester after evaluation of worthiness.
- Golden tests (not a fun, but depending on the visual implementation it might be valuable).

A hybrid of integration & e2e tests

How your app is built? (Dependencies)

Validation of business flows (System or component).

It will be a question of:

- integration & e2e
- integration || e2e
- a hybrid.

[.code-highlight: 1, 4-5, 8, 10, 12-17, 19-26]

```
// Integration
import 'main.mocks.dart';

@GenerateMocks([MockAuthenticationRepository])
void main({void Function(List)? stubbing}) {
  WidgetsFlutterBinding.ensureInitialized();

  final mockAuthenticationRepository =
MockAuthenticationRepository();

  stubbing?.call([mockAuthenticationRepository]);

  runApp(
    MultiRepositoryProvider(
      providers: [
        RepositoryProvider<MockAuthenticationRepository>(
          create: (context) => mockAuthenticationRepository,
        ),
        // ...
      ],
      child: const MyApp(),
    ),
  );
}

// integration_test/integration/main.dart
```

[.code-highlight: 1, 2, 12-23, 33]

```
// Integration
```

```
import '../main.dart' as main_mock;
group('Feature: user log in with email', () {
    testWidgets('Scenario 1: log in correctly with email',
        (WidgetTester tester) async {

      final email = Faker.email();
      final password = Faker.password();
      final accessToken = Faker.accessToken();

      // Given a registred user.
      main_mock.main(stubbing: (mocks) {
        for (var mock in mocks) {
          if (mock is AuthenticationRepository) {
            when(mock.signin(
              email: email,
              password: password,
            )).thenAnswer(
              (_) => Future.value(accessToken),
            );
          }
        }
      });

      // When the user log in with email.
      await tester.pumpAndSettle();
      // ...
      await tester.enterText(emailTextFormField, email);
      // ...
      expect(find.text('Feed'), findsOneWidget);
    });
});
// integration_test/integration/authentication/email.dart
```

```
// e2e
group('Feature: user log in with email', () {
    testWidgets('Scenario 1: log in correctly with email',
        (WidgetTester tester) async {
      late String email;
      late String password;

      // Given a registred user.
```

```
      email = Faker.email();
      password = Faker.password();
      await Primitives.signupWithEmailAndPassword(
        email: email,
        password: password,
      );
      main_application.main();

      // When the user log in with email.
      await tester.pumpAndSettle();
      // ...
      await tester.enterText(emailAddressTextFormField, email);
      // ...
      await tester.tap(find.byKey(const
ValueKey('authentication.login.button')));

      // Then the user is logged in.
      await tester.pumpAndSettle();
      expect(find.text('Feed'), findsOneWidget);
    });
});
```

Again

Integration tests (with mocks):

- Validation of complex application features
- When dependencies are hard to interact with.
- Or the cost of using them for tests is high.

E2e without any mocks:

- As much as possible,
- Especially critical features

Bonus: CI/CD

Both Integration and e2e can be part of CI/CD:

- integration_test package tests can be run in Firebase Test Lab.
- Codemagic supports integration_test.

Thank you!

Slides: https://github.com/**jamalbelilet/testing-flutter**

inline left

[.footer: ## @JamalBelilet (🧿 🐙 📸 👍)]