Testes em Dart

Temistocles Carvalho Zwang

Testes

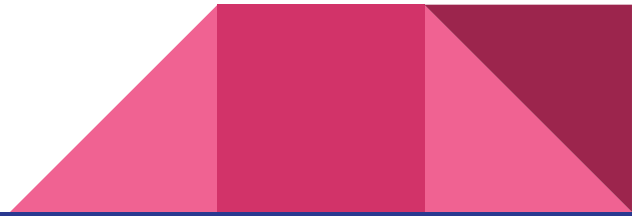
- Testes em linhas gerais
- Se códigos grandes são construídos de trechos pequenos, testando o código pequeno estamos abrangendo o grande

- Em Dart:
  - “Unit tests focus on verifying the smallest piece of testable software, such as a function, method, or class. Your test suites should have more unit tests than other kinds of tests.”
  - Component tests
  - Integration and end-to-end tests

Iniciando o ambiente

- dart create e dart test --help
- Adicionando o package test e solucionando o erro “Target of URI doesn't exist”
- nome_test.dart

“The test runner considers any file that ends with _test.dart to be a test file. If you don't pass any paths, it will run all the test files in your test/ directory, making it easy to test your entire application at once.”



package e documentação

- O Dart conta com ferramentas nativas para testes através do package test.dart
  - test()
  - expect()
  - group()
  - setUp() e tearDown()

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

void main() {
  test('String.split() splits the string on the delimiter', () {
    var string = 'foo,bar,baz';
    expect(string.split(','), equals(['foo', 'bar', 'baz']));
  });

  test('String.trim() removes surrounding whitespace', () {
    var string = '  foo ';
    expect(string.trim(), equals('foo'));
  });
}
```

Matchers

- matchers “The base class for all matchers.”
  - isA<T>() → TypeMatcher<T>
    - Returns a matcher that matches objects with type T. [...]
  - contains(Object? expected) → Matcher
    - Returns a matcher that matches if the match argument contains the expected value. [...]
  - startsWith(String prefixString) → Matcher
    - Returns a matcher that matches if the match argument is a string and starts with prefixString

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

void main() {
  test('.split() splits the string on the delimiter', () {
    expect('foo,bar,baz', allOf([
      contains('foo'),
      isNot(startsWith('bar')),
      endsWith('baz')
    ]));
  });
}
```

Escrevendo testes personalizados

- O teste por vezes vai exigir a reescrita do código
-

```
stdout.write('insira o dia:');
String? diaString = stdin.readLineSync();

stdout.write('insira o mês:');
String? mesString = stdin.readLineSync();

stdout.write('insira o ano:');
String? anoString = stdin.readLineSync();
```

```
test('testando entrada de dia', () {
  int dia = DateTime.now().day;
  expect(dia, isPositive);
  expect(dia, isNotNull);
  expect(dia, greaterThan(0));
  expect(dia, lessThan(32));
});
```

Escrevendo testes personalizados

- Testando uma String

```
 8  test('verificando string', () {
 9    String frase = r'O caractere de escape \t representa uma "tabulação"';
10    print(frase);
11    expect(frase, allOf(
12      contains('\\t'),
13      contains('"tabulação"')));
14  });
15  }
```

Escrevendo testes personalizados

- Ao criar testes precisamos testar e desconfiar do óbvio
-

```
 6   test('lista de numeros', () {
 7     final lista = [1, 2, 3];
 8     print(somarLista(lista));
 9     expect (lista,isList);
10   });
     Run | Debug
11   test('Soma', () {
12     expect(somarLista([1, 2, 3]), 6);
13   });
14 }
```

Resumindo

- definir o que será testado antes
- Selecionar o package e ver as ferramentas disponíveis na documentação
- Verificar as práticas de testes em linhas gerais
- O que não abordei