Criando testes **BDD-style** com **Ginkgo**

Carol Caires

Software Developer - Estratégia Concursos

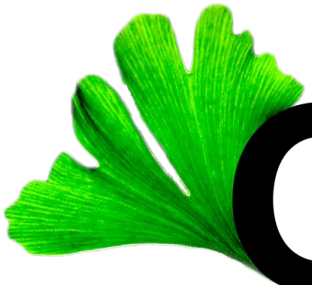
@carolinacaires

(no LinkedIn)

Igor L. Halfeld

Software Engineer - Estratégia Concursos

@igorhalfeld



Ginkgo

A Golang BDD Testing Framework

github.com/**onsi/ginkgo**

Mas... o que é BDD?

Behavior Driven Development

Dado... *um contexto*
Quando... *a ação é executada*
Então... *possuo um resultado*

Instalação

```
go get github.com/onsi/ginkgo/ginkgo
go get github.com/onsi/gomega/...
```

Vamos ao nosso primeiro teste

Imaginar uma função fibonnaci 💃

```go
func F(n int) (int, error) {
    if n < 1 {
        return 0, errors.New("Numero deve ser maior do que zero")
    }
    current, prev := 0, 1
    for i := 0; i < n; i++ {
        current, prev = current+prev, current
    }
    return current, nil
}
```

```go
package utils_test

import (
  "testing"

  . "github.com/onsi/ginkgo"
  . "github.com/onsi/gomega"
)

func TestFibonacci(t *testing.T) {
  RegisterFailHandler(Fail)
  RunSpecs(t, "Fibonacci Suite")
}
```

Avisa o GoMega que o teste falhou.

Indica o Ginkgo a suíte que vai ser rodada.


Describe
BeforeEach
It
When
AfterEach
XContext
XIt
XDescribe
Pit
Context
SPACE RANGER
LIGHTYEAR

Primeiro, identificamos a função que será testada

```
var _ = Describe("Fibonacci", func() {
```

Depois descrevemos o comportamento isolado da função, declaramos variáveis usadas nos testes...

```
Describe("Retornar o número de Fibonacci", func() {
    var (
        n   int
        f   int
        err error
    )
```

Antes de cada test case, iremos definir o valor da variável que será passada para a função

```
BeforeEach(func() {
    n = 6
})
```

E logo em seguida vamos chamar a função testada com o valor que definimos

```
JustBeforeEach(func() {
    f, err = fibonacci.F(n)
})
```

Definimos um contexto...

```
Context("quando a sequencia é 6", func() {
```

E especificamos qual é o comportamento esperado dentro desse contexto

```
Specify("Deve retornar o sexto número da sequencia de Fibonacci", func() {
```

E, por fim, testamos se o valor retornado é o que estávamos esperando

```
Expect(f).Should(Equal(8))
```

```
Context("quando a sequencia é 10", func() {
    BeforeEach(func() {
        n = 10
    })
    Specify("Deve retornar o décimo número da sequencia de Fibonacci", func() {
        Expect(f).Should(Equal(55))
    })
    It("Deve não retornar erro", func() {
        Expect(err).ToNot(HaveOccurred())
    })
})
Context("quando a sequencia é 0", func() {
    BeforeEach(func() {
        n = 0
    })
    Specify("Deve retornar zero", func() {
        Expect(f).Should(BeZero())
    })
    It("Deve retornar erro", func() {
        Expect(err).To(HaveOccurred())
    })
})
```

Seguindo esse modelo, podemos adicionar quantos contextos forem necessários

```
var _ = Describe("Fibonacci", func() {

    Describe("Retornar o número de Fibonacci", func() {
        var (
            n   int
            f   int
            err error
        )

        BeforeEach(func() {
            n = 6
        })

        JustBeforeEach(func() {
            f, err = fibonacci.F(n)
        })

        Context("quando a sequencia é 6", func() {
            Specify("Deve retornar o sexto número da sequencia de Fibonacci", func() {
                Expect(f).Should(Equal(8))
            })
            It("Deve não retornar erro", func() {
                Expect(err).ToNot(HaveOccurred())
            })
        })
        Context("quando a sequencia é 10", func() {…
        })
        Context("quando a sequencia é 0", func() {…
        })
    })
})
```

E vamos rodar os testes...

```
 □   ginkgo -v ./...
Running Suite: Fibonacci Suite
==============================
Random Seed: 1566942513
Will run 6 of 6 specs

Fibonacci Retornar o número de Fibonacci quando a sequencia é 6
  Deve retornar o sexto número da sequencia de Fibonacci
  /Users/carolina/GoProjects/ginkgo-talk/fibonacci/fibonacci_test.go:34
•
------------------------------
Fibonacci Retornar o número de Fibonacci quando a sequencia é 6
  Deve não retornar erro
  /Users/carolina/GoProjects/ginkgo-talk/fibonacci/fibonacci_test.go:37
•
```

```
------------------------------
Fibonacci Retornar o número de Fibonacci quando a sequencia é 0
  Deve retornar zero
  /Users/carolina/GoProjects/ginkgo-talk/fibonacci/fibonacci_test.go:56
•
------------------------------
Fibonacci Retornar o número de Fibonacci quando a sequencia é 0
  Deve retornar erro
  /Users/carolina/GoProjects/ginkgo-talk/fibonacci/fibonacci_test.go:59
•
Ran 6 of 6 Specs in 0.001 seconds
SUCCESS! -- 6 Passed | 0 Failed | 0 Pending | 0 Skipped
PASS

Ginkgo ran 1 suite in 1.496929051s
Test Suite Passed
```

E se o retorno da função for diferente do esperado?

```
Fibonacci Retornar o número de Fibonacci quando a sequencia é 10
  Deve retornar o décimo número da sequencia de Fibonacci
  /Users/carolina/GoProjects/ginkgo-talk/fibonacci/fibonacci_test.go:45

• Failure [0.000 seconds]
Fibonacci
/Users/carolina/GoProjects/ginkgo-talk/fibonacci/fibonacci_test.go:16
  Retornar o número de Fibonacci
  /Users/carolina/GoProjects/ginkgo-talk/fibonacci/fibonacci_test.go:18
    quando a sequencia é 10
    /Users/carolina/GoProjects/ginkgo-talk/fibonacci/fibonacci_test.go:41
      Deve retornar o décimo número da sequencia de Fibonacci [It]
      /Users/carolina/GoProjects/ginkgo-talk/fibonacci/fibonacci_test.go:45

      Expected
          <int>: 55
      to equal
          <int>: 50

      /Users/carolina/GoProjects/ginkgo-talk/fibonacci/fibonacci_test.go:46
```

```
Summarizing 1 Failure:

[Fail] Fibonacci Retornar o número de Fibonacci quando a sequencia é 10 [It]

/Users/carolina/GoProjects/ginkgo-talk/fibonacci/fibonacci_test.go:46

Ran 6 of 6 Specs in 0.001 seconds
FAIL! -- 5 Passed | 1 Failed | 0 Pending | 0 Skipped
--- FAIL: TestFibonacci (0.00s)
FAIL

Ginkgo ran 1 suite in 1.481726295s
Test Suite Failed
```

Podemos "pular" alguns test cases temporariamente...

```
XContext("quando a sequencia é 6", func() {
    Specify("Deve retornar o sexto número da sequencia de Fibonacci", func() {
        Expect(f).Should(Equal(8))
    })
    It("Deve não retornar erro", func() {
        Expect(err).ToNot(HaveOccurred())
    })
})
```

```
Ran 4 of 6 Specs in 0.001 seconds
SUCCESS! -- 4 Passed | 0 Failed | 2 Pending | 0 Skipped
PASS

Ginkgo ran 1 suite in 1.255559024s
Test Suite Passed
```

Assim como focar apenas em testes específicos

```
FContext("quando a sequencia é 6", func() {
    Specify("Deve retornar o sexto número da sequencia de Fibonacci", func() {
        Expect(f).Should(Equal(8))
    })
    It("Deve não retornar erro", func() {
        Expect(err).ToNot(HaveOccurred())
    })
})
```

```
Ran 2 of 6 Specs in 0.000 seconds
SUCCESS! -- 2 Passed | 0 Failed | 0 Pending | 4 Skipped
PASS | FOCUSED

Ginkgo ran 1 suite in 1.578942245s
Test Suite Passed
Detected Programmatic Focus - setting exit status to 197
```

ginkgo -v ./...

-r roda os testes recursivamente dentro de um diretório

-succinct retorna informações bem reduzidas da suíte de testes

-cover retorna porcentagem da cobertura de testes

-failFast para a suite de testes em andamento quando houver falha

-timeout falha o teste se demorar mais do que *n* segundos para completar


Obrigado!
@carolinacaires - @igorhalfeld