FreteBras

BDD

BEHAVIOUR DRIVEN DEVELOPMENT

Data: 04/03/2022

ENGENHARIA

FreteBras

AGENDA

- O que é BDD?
- Prós e Contras
- Implementação do BDD em GoLang

O QUE BDD?

Desenvolvimento Orientado ao Comportamento:

- Dado: Um determinado contexto
- Quando: A ação é executada
- Então: Possuo um resultado

BDD é uma metodologia de teste que evoluiu do TDD.
O BDD é uma maneira útil de descrever seus casos de teste da perspectiva do usuário. Ele usa um formato *Dado-Quando-Então* que é cognitivamente fácil de analisar.
A saída pode ser depurada e compreendida pelas partes interessadas em tecnologia e negócios. Além de envolver as partes interessadas através do Desenvolvimento de Fora para Dentro (Outside-in-Development).

PRÓS E CONTRAS

FreteBras

Prós:

- Uma única fonte unificada de entendimento do BDD garante que todas as partes interessadas colaborem em conjunto para garantir que estejam sempre na mesma página para evitar ambiguidade de condições e prevenir defeitos o mais cedo possível.
- Documentação em forma de código

  À medida que os requisitos de negócios ou a documentação mudam, o caso de teste também se ajusta de acordo com as necessidades. Isso fará com que o caso de teste seja uma boa documentação para o desenvolvedor e aumentará a confiança do desenvolvedor ao refatorar ou fazer algo.

Contras:

- A simplicidade da declaração dada pode levar você a uma falsa sensação de segurança, pois pode ser muito mais difícil converter essas declarações em instruções de configuração e scripts.
- É mais difícil escrever testes automatizados de antemão, antes de ver a funcionalidade que se espera testar. O processo BDD envolve escrever os testes primeiro, fazer com que esses testes falhem e, em seguida, continuar o desenvolvimento até que eles sejam aprovados. Isso não é tão fácil ou rápido quanto escrever um teste para uma funcionalidade existente que você já pode ver e trabalhar. A abordagem BDD exige um pouco mais de trabalho para evoluir esses testes à medida que o aplicativo cresce.

BDD EM GOLANG

Crie um módulo para o teste:

```
go mod init github.com/carloshdurante/bdd-golang
```

Instale as bibliotecas Ginkgo e Gomega

```
go get github.com/onsi/ginkgo/v2
```

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

Crie o *Suite Test* no diretório da implementação:

```
ginkgo bootstrap
```

Agora rode o comando abaixo para testar o Ginkgo:

```
ginkgo -v ./...
```

```
Will run 0 of 0 specs

Ran 0 of 0 Specs in 0.000 seconds
SUCCESS! -- 0 Passed | 0 Failed | 0 Pending | 0 Skipped
PASS

Ginkgo ran 1 suite in 1.819579s
Test Suite Passed
```

O arquivo do Suite Test ficará assim:

```go
package truck_test

import (
    "testing"

    . "github.com/onsi/ginkgo/v2"
    . "github.com/onsi/gomega"
)

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

BDD EM GOLANG

Seguindo a boa prática do BDD, escrevemos primeiro os cenários de teste.

Cenário: Buscar dados de um caminhão

- Dado: que exista um caminhão
- Quando: solicitado os dados do caminhão
- Então: deve retornar a marca e modelo do caminhão

Feito isso utilize esse comando para rodar os testes:

```
ginkgo -v ./...
```

```
Summarizing 2 Failures:
  [FAIL] Truck GetTruck when the truck is valid [It] should return the truck
  C:/Users/carlo/golang/bdd-golang/truck/truck_test.go:19
  [FAIL] Truck GetTruck when the truck is empty [It] should return truck is empty
  C:/Users/carlo/golang/bdd-golang/truck/truck_test.go:26

Ran 2 of 2 Specs in 0.001 seconds
FAIL! -- 0 Passed | 2 Failed | 0 Pending | 0 Skipped
--- FAIL: TestTruck (0.00s)
FAIL

Ginkgo ran 1 suite in 1.7191356s

Test Suite Failed
```

Os testes falharam, pois ainda não implementamos a solução.

O arquivo de teste ficará assim:

```
package truck_test

import (
    . "github.com/onsi/ginkgo/v2"
    . "github.com/onsi/gomega"

    "github.com/carloshdurante/bdd-golang/truck"
)

var _ = Describe("Truck", func() {

    Describe("GetTruck", func() {
        TruckValid := truck.Truck{
            Brand: "Ford",
            Model: "F150",
        }
        Context("when the truck is valid", func() {
            It("should return the truck", func() {
                Expect(TruckValid.GetTruck(TruckValid)).Should(Equal("Brand: Ford\nModel: F150"))
            })
        })

        Context("when the truck is empty", func() {
            TruckNotValid := truck.Truck{}
            It("should return truck is empty", func() {
                Expect(TruckNotValid.GetTruck(TruckNotValid)).Should(Equal("Truck is empty"))
            })
        })
    })
})
```

BDD EM GOLANG

Após os testes falharem, chegou o momento de implementar a solução.

Cenário: Buscar dados de um caminhão

- Dado: que exista um caminhão
- Quando: solicitado os dados do caminhão
- Então: deve retornar a marca e modelo do caminhão

Feito isso utilize esse comando para rodar os testes:

```
ginkgo -v ./...
```

```
Will run 2 of 2 specs
------------------------------
Truck GetTruck when the truck is valid
  should return the truck
  C:/Users/carlo/golang/bdd-golang/truck/truck_test.go:18
+
------------------------------
Truck GetTruck when the truck is empty
  should return truck is empty
  C:/Users/carlo/golang/bdd-golang/truck/truck_test.go:25
+

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

Ginkgo ran 1 suite in 1.7339104s
Test Suite Passed
```

Após a implementação da solução, todos os testes passaram.

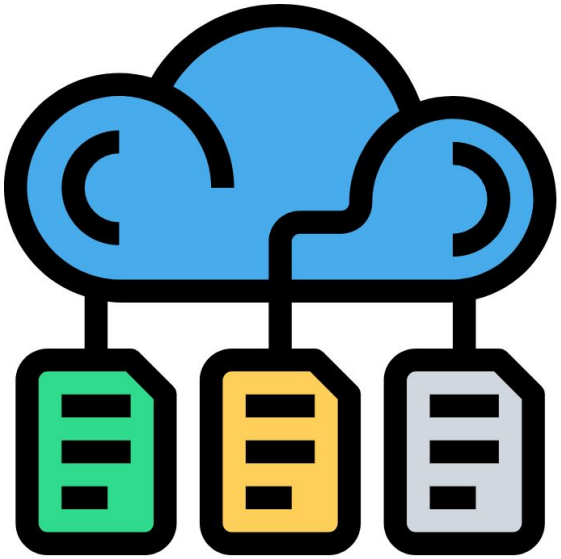
O arquivo da implementação ficará assim:

```go
package truck

import (
    "fmt"
)

type Truck struct {
    Brand string
    Model string
}

func (t *Truck) GetTruck(truck Truck) string {
    if truck == (Truck{}) {
        return "Truck is empty"
    }

    return fmt.Sprintf("Brand: %s\nModel: %s", truck.Brand, truck.Model)
}
```

FreteBras

REFERÊNCIAS

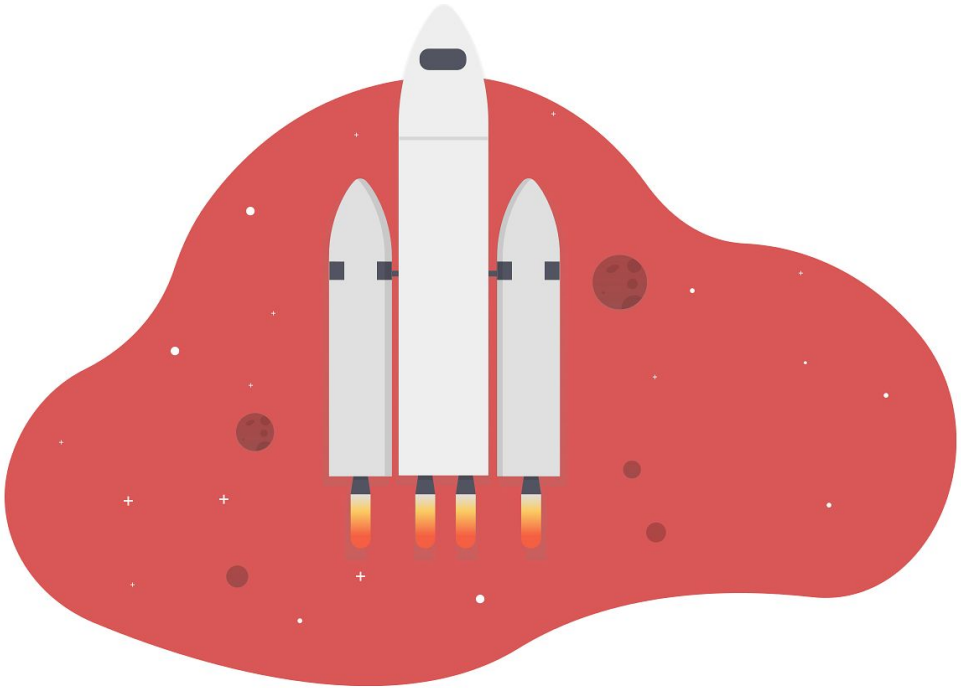
- Automated Testing
- LevelUp GitConnected
- Youtube - GoLang SP
- Github - Repositório Ginkgo
- Github – Repositório Gomega
- Github - Repositório dos exemplos



PERGUNTAS, DÚVIDAS E CONTRIBUIÇÕES

BORA TIME!

FreteBras



ENGENHARIA