Golang dot testing

Trip down the rabbit hole

Richa’rd Kova’cs
October 2020

Me, myself and I

- Kubernetes Network Engineer
- @ IBM Cloud
- 6 years in Go space
- Many years of DevOps background
- Open Source activist
- Known as mhmxs
  - Twitter
  - Linkedin

Agenda

- Built-in framework
  - The basics
  - Table driven tests
  - Code coverage
  - Race detection
- Mocks and fakes
- Monkey patching
- Helpers
- Dependency Injection

Built-in framework

Built-in framework - The basics

```go
func plus(a, b int) int {
    return a + b
}

func TestPlus(t *testing.T) {
    res := plus(1, 1)
    if 2 != res {
        t.Errorf("Result not match: %d", res)
    }
}
```

```
# go test
PASS
ok      mhmxs/golang-dot-testing   0.006s
```

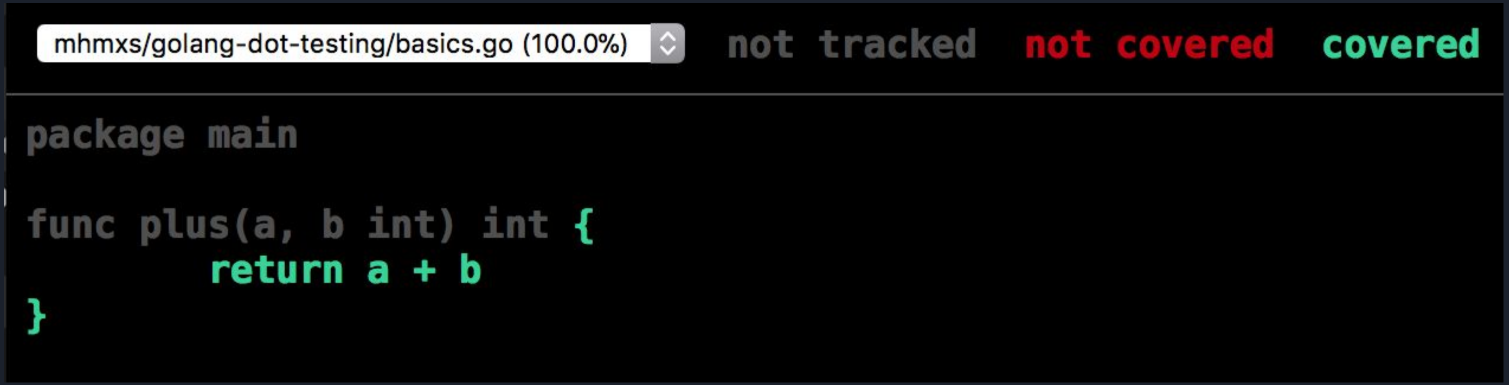
Built-in framework - Table driven tests

```
func TestPlusTable(t *testing.T) {
    tests := []struct {
        a   int
        b   int
        out int
    }{
        {1, 1, 2},
        {1, 2, 3},
    }

    for i, s := range tests {
        res := plus(s.a, s.b)

        if s.out != res {
            t.Errorf("Result not match: %d at %d" , res,
i)
        }
    }
```

```
# go test
PASS
ok      mhmxs/golang-dot-testing   0.006s
```

Built-in framework - Code coverage

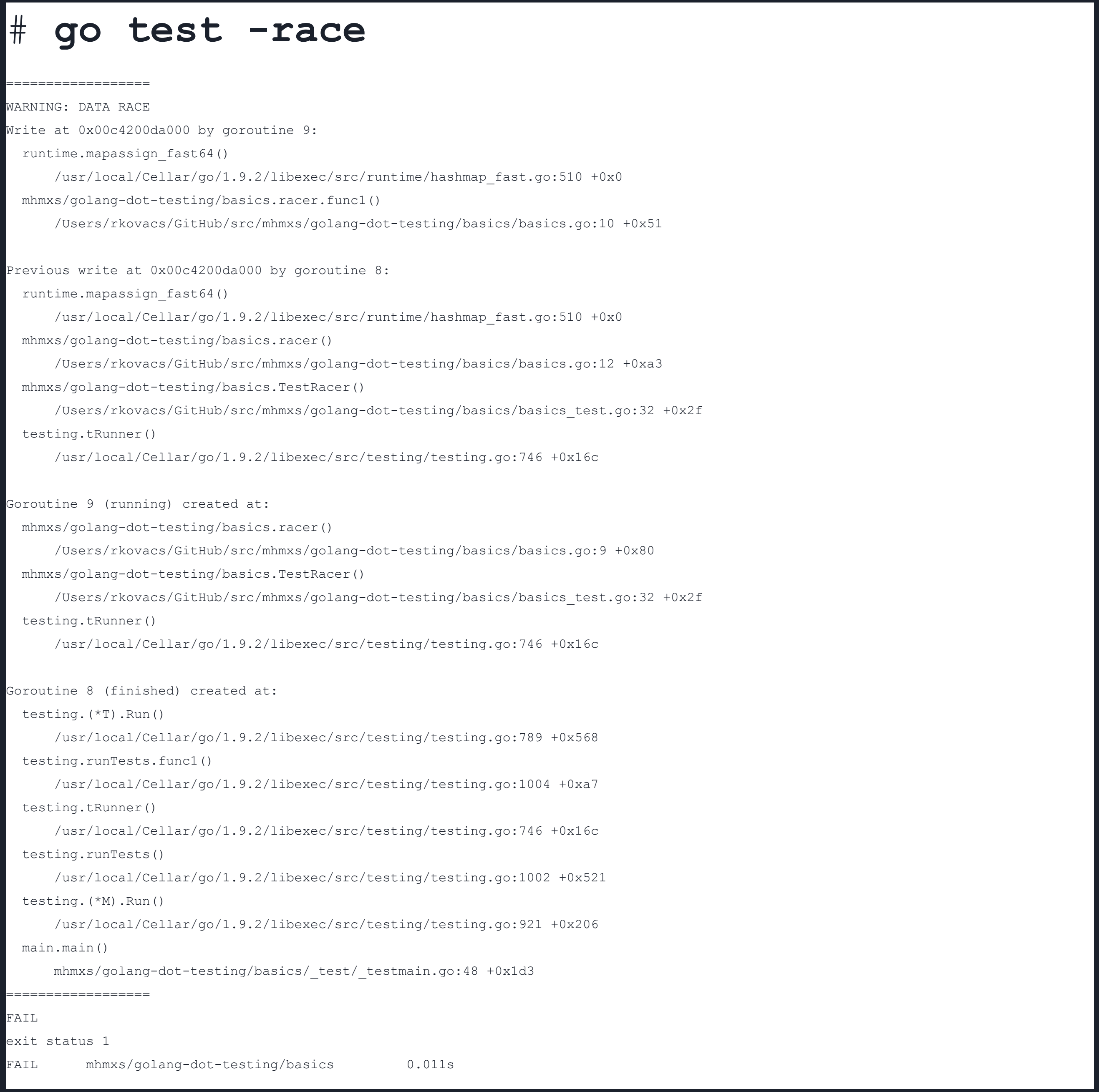
```
# go test -cover -coverprofile=cover.out
PASS
coverage: 100.0% of statements
ok      mhmxs/golang-dot-testing   0.006s


# go tool cover -html=cover.out
```



```
mhmxs/golang-dot-testing/basics.go (100.0%)   not tracked   not covered   covered

package main

func plus(a, b int) int {
        return a + b
}
```

Built-in framework - Race detection

```go
func racer() {
    m := map[int]string{}
    go func() {
        m[1] = "a"
    }()
    m[2] = "b"
}

func TestRacer(t *testing.T) {
    racer()
}
```



```
# go test -race

==================
WARNING: DATA RACE
Write at 0x00c4200da000 by goroutine 9:
  runtime.mapassign_fast64()
      /usr/local/Cellar/go/1.9.2/libexec/src/runtime/hashmap_fast.go:510 +0x0
  mhmxs/golang-dot-testing/basics.racer.func1()
      /Users/rkovacs/GitHub/src/mhmxs/golang-dot-testing/basics/basics.go:10 +0x51

Previous write at 0x00c4200da000 by goroutine 8:
  runtime.mapassign_fast64()
      /usr/local/Cellar/go/1.9.2/libexec/src/runtime/hashmap_fast.go:510 +0x0
  mhmxs/golang-dot-testing/basics.racer()
      /Users/rkovacs/GitHub/src/mhmxs/golang-dot-testing/basics/basics.go:12 +0xa3
  mhmxs/golang-dot-testing/basics.TestRacer()
      /Users/rkovacs/GitHub/src/mhmxs/golang-dot-testing/basics/basics_test.go:32 +0x2f
  testing.tRunner()
      /usr/local/Cellar/go/1.9.2/libexec/src/testing/testing.go:746 +0x16c

Goroutine 9 (running) created at:
  mhmxs/golang-dot-testing/basics.racer()
      /Users/rkovacs/GitHub/src/mhmxs/golang-dot-testing/basics/basics.go:9 +0x80
  mhmxs/golang-dot-testing/basics.TestRacer()
      /Users/rkovacs/GitHub/src/mhmxs/golang-dot-testing/basics/basics_test.go:32 +0x2f
  testing.tRunner()
      /usr/local/Cellar/go/1.9.2/libexec/src/testing/testing.go:746 +0x16c

Goroutine 8 (finished) created at:
  testing.(*T).Run()
      /usr/local/Cellar/go/1.9.2/libexec/src/testing/testing.go:789 +0x568
  testing.runTests.func1()
      /usr/local/Cellar/go/1.9.2/libexec/src/testing/testing.go:1004 +0xa7
  testing.tRunner()
      /usr/local/Cellar/go/1.9.2/libexec/src/testing/testing.go:746 +0x16c
  testing.runTests()
      /usr/local/Cellar/go/1.9.2/libexec/src/testing/testing.go:1002 +0x521
  testing.(*M).Run()
      /usr/local/Cellar/go/1.9.2/libexec/src/testing/testing.go:921 +0x206
  main.main()
      mhmxs/golang-dot-testing/basics/_test/_testmain.go:48 +0x1d3
==================
FAIL
exit status 1
FAIL	mhmxs/golang-dot-testing/basics	0.011s
```

Mock and fakes

Mocks and fakes

- Has many mocking framework
  - GoMock
  - Pegomock
  - Counterfeiter
  - Hel
  - Mockery
  - Testify
- Good integration with built-in framework
- All* based on code generation
- Some of them are supporting compiler directives a.k.a. //go:generate

* What i discovered

Monkey patching

Monkey patching

- Rewriting the running executable at runtime
- Useful because Go’s procedural design concept
- Bouk’s monkey is one of the favorites

```go
func main() {
    monkey.Patch(fmt.Println, func(a ...interface{}) (n int, err error) {
        s := make([]interface{}, len(a))
        for i, v := range a {
            s[i] = strings.Replace(fmt.Sprint(v), "hell", "*heck*", -1)
        }
        return fmt.Fprintln(os.Stdout, s...)
    })
    fmt.Println("what the hell?") // what the *heck*?
}
```

Monkey patching - Problems

- Happens at runtime, so issues will occur on first run
- Doesn’t test production code, it tests something which never runs in prod
- Easy to forget dependencies and create cross module test instead of unit
- Only single thread test execution is possible
- Out of control, last override wins

Helpers

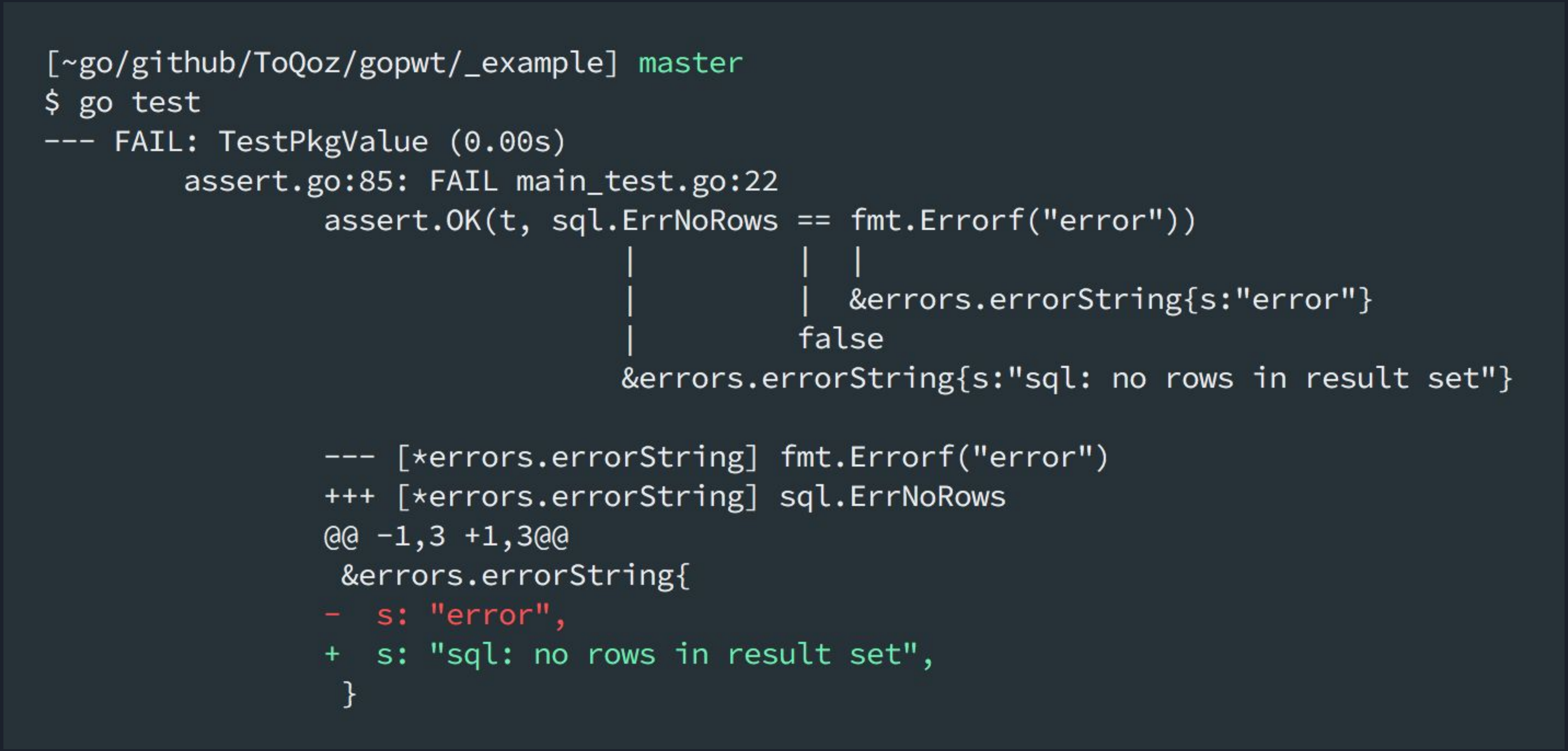
Helpers - Testify

- Easy assertions
- Mocking framework
- Test suits

```go
type MyMock struct{
    mock.Mock
}
func TestTestify(t *testing.T) {
    assert.Equal(t, 1, 1, "Not match. WTH?")

    mock := new(MyMock)
    mock.On("DoSomething", "input").Return("output")

    funcUnderTest(mock)

    mock.AssertExpectations(t)
}
```

```go
type ExampleTestSuite struct {
   suite.Suite
   VariableThatShouldStartAtFive int
}
func (suite *ExampleTestSuite) SetupTest() {
   suite.VariableThatShouldStartAtFive = 5
}
func (suite *ExampleTestSuite) TestExample() {
   assert.Equal(suite.T(), 5, suite.VariableThatShouldStartAtFive)
}
func TestExampleTestSuite(t *testing.T) {
   suite.Run(t, new(ExampleTestSuite))
}
```

Helpers - Gopwt



```
[~go/github/ToQoz/gopwt/_example] master
$ go test
--- FAIL: TestPkgValue (0.00s)
        assert.go:85: FAIL main_test.go:22
                assert.OK(t, sql.ErrNoRows == fmt.Errorf("error"))
                                 |         |  |
                                 |         |  &errors.errorString{s:"error"}
                                 |         false
                                 &errors.errorString{s:"sql: no rows in result set"}

                --- [*errors.errorString] fmt.Errorf("error")
                +++ [*errors.errorString] sql.ErrNoRows
                @@ -1,3 +1,3@@
                 &errors.errorString{
                -  s: "error",
                +  s: "sql: no rows in result set",
                 }
```

Helpres - Ginkgo ‘n Gomega

- Structure your BDD-style tests expressively
    - Describe, Context and When container blocks
    - BeforeEach and AfterEach blocks for setup and tear down
    - BeforeSuite and AfterSuite blocks to prep for and cleanup after a suite
- A comprehensive test runner that lets you
    - Mark specs as pending
    - Run your tests in random order, and then reuse random seeds to replicate the same order
    - Break up your test suite into parallel processes for straightforward test parallelization
- Watches packages for changes, then reruns tests. Run tests immediately as you develop

```go
func TestCalc(t *testing.T) {
    RegisterFailHandler(Fail)
    RunSpecs(t, "Calculator Suite")
}
var _ = Describe("Calculator", func() {
    Describe("Add numbers", func() {
        Context("1 and 2", func() {
            It("should be 3", func() {
                Expect(Add(1, 2)).To(Equal(3))
            })
        })
    })
    Describe("Subtract numbers", func() {
        Context("3 from 5", func() {
            It("should be 2", func() {
                Expect(Subtract(5, 3)).To(Equal(2))
            })
        })
    })
})
```

Dependency Injection

God helps those who help themselves

Dependency Injection

```go
func DescribeCredential(c *cli.Context) {
    client := NewOAuth2HTTPClient("localhost", "user", "password")
    params := credentials.NewGetCredentialParams().WithName( c.String(FlName.Name))
    resp, err := client.Credentials.GetCredential(params)
    if err != nil {
        panic(err.Error())
    }
    println(resp.Payload.ID)
}
```

Dependency Injection

```
func DescribeCredential (c *cli.Context) {
    client := NewOAuth2HTTPClient("localhost", "user", "password")
    println(describeCredentialImpl(c.String, client.Credentials))
}

type credentialClient interface {
    GetCredential(*credentials.GetCredentialParams) ( *credentials.GetCredentialOK,  error)
}

func describeCredentialImpl (nameFinder func(string) string, client credentialClient)  int64 {
    params := credentials.NewGetCredentialParams().WithName( nameFinder(FlName.Name))
    resp, err := client.GetCredential(params)
    if err != nil {
        panic(err.Error())
    }
    return resp.Payload.ID
}
```

Dependency Injection

```go
type mockClient struct{}

func (mc mockClient) GetCredential(params *credentials.GetCredentialParams) (*credentials.GetCredentialOK, error) {
    return &credentials.GetCredentialOK{Payload: &models.CredentialResponse{ID: 0}}, nil
}

func TestDesribeCredential(t *testing.T) {
    nameFinder := func(in string) string {
        return "name"
    }

    if 0 != describeCredentialImpl(nameFinder, mockClient{}) {
        t.Error("ID not match")
    }
}
```

Dependency Injection

- Easier to avoid tight code coupling
- Enforces to use SOLID design pattern
- Enforces to mock all dependencies
- Full control over side effects
- Supports parallel test execution

Thank You

Any question?