



# Backend Programming

M. Yauri M. Attamimi

*Upgrade your potential !*



# **Foundation Class**

**- WEEK 3 (Session 1) -**



# Basic Unit Testing

# Overview (I)

- Unit testing improves the quality of your code, it identifies every defect which may have aroused, before code is sent further for integration testing.
- Generally speaking, there are 2 approach to write your Unit test:
  - Test First (a.k.a TDD / Test Driven Development)
  - Test Last
- Check : <https://apiumhub.com/tech-blog-barcelona/top-benefits-of-unit-testing/>
- Go provides built-in functionality to test your code, no need for an expensive setup or 3<sup>rd</sup> party libraries to create unit tests.



# Overview (II)

- Just like human body which made up and depends on the functionality and reliability of human cells, software are depend on the unit components.
- Unit components can be a function, struct, method, or anything that end user might depend on.
- We have to ensure that whatever inputs to the unit components would never break the application.
- A unit test is a program that tests a unit component by all possible means and compares the result to the expected output.



# Which part need a unit test ?

Whatever exports (exported-types) are available within the package need a unit test.



# Basic Structure of Unit Test

```
import "testing"

func TestSomething(t *testing.T) {
    t.Error() // to indicate test failed
}
```

You can have as many test functions as you want inside a single test file.

The collection of test cases (functions) is called a **test suite**.





## EXPLORER

## GOCODE

- > go-grpc
- > go-grpc-example
- > go-jwt-tutorial
- > go-micro
- > gokit
- > gokit\_consul
- > gokit-manualcoded
- > gokit-microservices
- > gokitcli
- > golang-postgres-rest-api
- > gorm
- ▼ **greeting**
  - go.mod
  - go.sum
  - hello.go
  - main.go** 1
- > grpc-rest
- > hello
- > jancarloviray.com
- > microservices\_with\_go
- > mongo-rest-api
- > newstack-rest
- > nsq
- > oop
- > parsing-xml
- > postgres
- > readers
- > stringutil
- > structs
- > test
- > thenewstack-io

main.go X

src &gt; github.com &gt; yauritux &gt; greeting &gt; main.go

```
1 package main
2
3 import (
4     "fmt"
5     "github.com/logrusorgru/aurora"
6 )
7
8 func main() {
9     greeting := hello("Gopher")
10
11     fmt.Println(aurora.Yellow(greeting))
12 }
```

hello.go X

src &gt; github.com &gt; yauritux &gt; greeting &gt; hello.go &gt; ...

```
1 package main
2
3 import "fmt"
4
5 func hello(user string) string {
6     return fmt.Sprintf("Hello %v!", user)
7 }
8
```

PROBLEMS 1 OUTPUT DEBUG CONSOLE TERMINAL

1: zsh

```
##( 08/13/19@11:34AM )( yauritux@Yauris-MacBook-Pro ):~/gocode/src/github.com/yauritux/greeting
go mod init greeting
go: creating new go.mod: module greeting
##( 08/13/19@11:34AM )( yauritux@Yauris-MacBook-Pro ):~/gocode/src/github.com/yauritux/greeting
go run *.go
go: finding github.com/logrusorgru/aurora latest
go: downloading github.com/logrusorgru/aurora v0.0.0-20190803045625-94edacc10f9b
go: extracting github.com/logrusorgru/aurora v0.0.0-20190803045625-94edacc10f9b
Hello Gopher!
##( 08/13/19@11:35AM )( yauritux@Yauris-MacBook-Pro ):~/gocode/src/github.com/yauritux/greeting
```



## EXPLORER

## GOCODE

- > geometry
- > go-grpc
- > go-grpc-example
- > go-jwt-tutorial
- > go-micro
- > gokit
- > gokit\_consul
- > gokit-manualcoded
- > gokit-microservices
- > gokitcli
- > golang-postgres-rest-api
- > gorm
- ✓ > greeting
  - go.mod
  - go.sum
  - hello\_test.go
  - hello.go 1
  - main.go 1
- > grpc-rest
- > hello
- > jancarloviray.com
- > microservices\_with\_go
- > mongo-rest-api
- > newstack-rest
- > nsq
- > oop
- > parsing-xml
- > postgres
- > ...

hello\_test.go

src &gt; github.com &gt; yauritux &gt; greeting &gt; hello\_test.go

```
run package tests | run file tests
1 package main
2
3 import "testing"
4
5 run test | debug test
6 func TestHello(t *testing.T) {
7     // test for empty argument
8     emptyResult := hello("")
9
10    if emptyResult != "Hello Dude !" {
11        t.Errorf("hello(\"\") failed, expected %v, got %v", '
12    }
13
14    // test for valid argument
15    result := hello("Gopher")
16
17    if result != "Hello Gopher!" {
18        t.Errorf(
19            "hello(\"Gopher\") failed, expected %v, got %v",
20            "Hello Gopher!", result)
21    }
22 }
```

...

hello.go

src &gt; github.com &gt; yauritux &gt; greeting &gt; hello.go

```
1 package main
2
3 import "fmt"
4
5 func hello(user string) string {
6     if len(user) == 0 {
7         return "Hello Dude !"
8     } else {
9         return fmt.Sprintf("Hello %v!", user)
10    }
11 }
12
```

PROBLEMS 2

OUTPUT

DEBUG CONSOLE

TERMINAL

1: zsh

```

#( 08/13/19@ 1:06PM )( yauritux@Yauris-MacBook-Pro ):/gocode/src/github.com/yauritux/greeting
go test
PASS
ok      greeting    0.012s
#( 08/13/19@ 1:06PM )( yauritux@Yauris-MacBook-Pro ):/gocode/src/github.com/yauritux/greeting
```

## EXPLORER

hello\_test.go X

...

hello.go

## GOCODE

src &gt; github.com &gt; yauritux &gt; greeting &gt; hello\_test.go

run package tests | run file tests

1 package main

2

3 import "testing"

4

run test | debug test

5 func TestHello(t \*testing.T) {

6 // test for empty argument

7 emptyResult := hello("")

8

9 if emptyResult != "Hello Dude !" {

10 t.Errorf("hello(\"\") failed, expected %v, got %v", "

11 } else {

12 t.Logf("hello(\"") success, expected %v, got %v", "

13 }

14

15 // test for valid argument

16 result := hello("Gopher")

17

18 if result != "Hello Gopher!" {

19 t.Errorf(

20 "hello(\"Gopher\") failed, expected %v, got %v",

21 "Hello Gopher!", result)

22 } else {

23 t.Logf("hello(\"Gopher\") success, expected %v, got %v",

24 "Hello Gopher!", result)

25 }

26 }

PROBLEMS 2

OUTPUT

DEBUG CONSOLE

TERMINAL

1: zsh

```

#( 08/13/19@ 1:13PM )( yauritux@Yauris-MacBook-Pro ):/gocode/src/github.com/yauritux/greeting
go test -v
=== RUN TestHello
--- PASS: TestHello (0.00s)
    hello_test.go:12: hello("") success, expected Hello Dude !, got Hello Dude !
    hello_test.go:23: hello("Gopher") success, expected Hello Gopher!, got Hello Gopher!
PASS
ok      greeting      0.007s
#( 08/13/19@ 1:13PM )( yauritux@Yauris-MacBook-Pro ):/gocode/src/github.com/yauritux/greeting

```

src &gt; github.com &gt; yauritux &gt; greeting &gt; hello.go

1 package main

2

3 import "fmt"

4

5 func hello(user string) string {

6 if len(user) == 0 {

7 return "Hello Dude !"

8 } else {

9 return fmt.Sprintf("Hello %v!", user)

10 }

11 }

12



# Add Color in Test Report

```
$( 08/13/19@ 1:17PM )( yauritux@Yauris-MacBook-Pro ) :~/gocode/src/github.com/yauritux/greeting
go get -u github.com/rakyll/gotest
go: finding github.com/rakyll/gotest latest
go: downloading github.com/rakyll/gotest v0.0.0-20180125184505-86f0749cd8cc
go: extracting github.com/rakyll/gotest v0.0.0-20180125184505-86f0749cd8cc
go: finding github.com/fatih/color v1.7.0
go: downloading github.com/fatih/color v1.7.0
go: extracting github.com/fatih/color v1.7.0
```

```
$( 08/13/19@ 1:18PM )( yauritux@Yauris-MacBook-Pro ) :~/gocode/src/github.com/yauritux/greeting
$GOBIN/gotest -v
=== RUN    TestHello
--- PASS: TestHello (0.00s)
    hello_test.go:12: hello("") success, expected Hello Dude !, got Hello Dude !
    hello_test.go:23: hello("Gopher") success, expected Hello Gopher!, got Hello Gopher!
PASS
ok      greeting      0.006s
$( 08/13/19@ 1:18PM )( yauritux@Yauris-MacBook-Pro ) :~/gocode/src/github.com/yauritux/greeting
```

# Select Test Case to Run

The screenshot shows the Visual Studio Code interface with a Go project. The Explorer on the left lists the project structure, with the `greeting` folder selected. The main editor displays two files: `hello_test.go` and `hello.go`.

**hello\_test.go**

```
5 // Test for empty argument
  run test | debug test
6 func TestHelloEmptyArg(t *testing.T) {
7     emptyResult := hello("")
8
9     if emptyResult != "Hello Dude !" {
10         t.Errorf("hello(\"\") failed, expected %v, got %v", '
11     } else {
12         t.Logf("hello(\"\") success, expected %v, got %v", "
13     }
14 }
15
16 // Test for non-empty argument
  run test | debug test
17 func TestHelloValidArg(t *testing.T) {
18     result := hello("Gopher")
19
20     if result != "Hello Gopher!" {
21         t.Errorf(
22             "hello(\"Gopher\") failed, expected %v, got %v",
23             "Hello Gopher!", result)
24     } else {
25         t.Logf("hello(\"Gopher\") success, expected %v, got %v",
26             "Hello Gopher!", result)
27     }
28 }
```

**hello.go**

```
1 package main
2
3 import "fmt"
4
5 func hello(user string) string {
6     if len(user) == 0 {
7         return "Hello Dude !"
8     } else {
9         return fmt.Sprintf("Hello %v!", user)
10    }
11 }
```

The bottom panel shows the **TERMINAL** output:

```
1: zsh
#( 08/13/19@ 1:22PM )( yauritux@Yauris-MacBook-Pro ):/gocode/src/github.com/yauritux/greeting
$GOBIN/gotest -v -run TestHelloE
=== RUN   TestHelloEmptyArg
--- PASS: TestHelloEmptyArg (0.00s)
    hello_test.go:12: hello("") success, expected Hello Dude !, got Hello Dude !
PASS
ok      greeting      0.007s
#( 08/13/19@ 1:22PM )( yauritux@Yauris-MacBook-Pro ):/gocode/src/github.com/yauritux/greeting
```





# Running Specific Test File

```

#( 08/13/19@ 1:25PM )( yauritux@Yauris-MacBook-Pro ):~/gocode/src/github.com/yauritux/greeting
go test hello_test.go
# command-line-arguments [command-line-arguments.test]
./hello_test.go:7:17: undefined: hello
./hello_test.go:18:12: undefined: hello
FAIL    command-line-arguments [build failed]

```

```

#( 08/13/19@ 1:27PM )( yauritux@Yauris-MacBook-Pro ):~/gocode/src/github.com/yauritux/greeting
go test hello_test.go hello.go
ok      command-line-arguments 0.005s
#( 08/13/19@ 1:27PM )( yauritux@Yauris-MacBook-Pro ):~/gocode/src/github.com/yauritux/greeting

```



# Caveats

⚠ When you have test cases (`_test.go` files) in your executable(main) package, you can't simply execute `go run *.go` to run the project. `*.go` part also matches the test files (`_test.go` files) and `go run` command do not run them. You will get `go run: cannot run *_test.go files (hello_test.go)` error if you do that.

There is no running away from this. You can either put all your test files in different package or use `go build` command and then run the binary file.



# Test Coverage

Test Coverage is the percentage of your code covered by test suit. In layman's language, it is the measurement of how many lines of code in your package were executed when you ran your test suit (*compared to total lines in your code*). Go provide built-in functionality to check your code coverage.





# Check Test Coverage

The screenshot displays the Visual Studio Code interface with two editor windows and a terminal. The left window, titled 'hello\_test.go', shows a Go test function 'TestHello' that calls 'Hello' with 'Gopher' and checks for a specific output. The right window, titled 'hello.go', shows the 'Hello' function implementation. The terminal at the bottom shows the command 'go test -cover' being executed, resulting in a coverage report of 66.7% for the 'greeting' package.

**EXPLORER**

- go-grpc
- go-grpc-example
- go-jwt-tutorial
- go-micro
- gokit
- gokit\_consul
- gokit-manualcoded
- gokit-microservices
- gokitcli
- golang-postgres-rest-api
- gorm
- greeting
  - go.mod
  - go.sum
  - hello\_test.go
  - hello.go
- grpc-rest
- hello
- jancarloviray.com
- microservices\_with\_go
- mongo-rest-api
- newstack-rest
- nsq
- oop
- parsing-xml
- postgres
- readers
- stringutil

**hello\_test.go**

```
1  g
2
3  |
4
5  run test | debug test
6  t *testing.T) {
7      // valid argument
8      := Hello("Gopher")
9
10     ult != "Hello Gopher!" {
11     f("hello(\"") failed, expected %v, got %v", "Hello Gopher!")
12
13     "hello(\"") success, expected %v, got %v", "Hello Gopher!",
14
15
16
```

**hello.go**

```
1  package greeting
2
3  import "fmt"
4
5  // Hello greet user based on the provided a
6  func Hello(user string) string {
7      if len(user) == 0 {
8          return "Hello Dude !"
9      }
10     return fmt.Sprintf("Hello %v!", user)
11 }
12
```

**TERMINAL**

```
1: zsh
yauritux/greeting → go test -cover
PASS
coverage: 66.7% of statements
ok      greeting    0.004s
yauritux/greeting →
```



# produce test coverage report

The screenshot shows the Visual Studio Code interface with a Go project. The Explorer sidebar on the left shows a file tree with folders like 'geometry', 'go-grpc', and 'greeting'. The 'greeting' folder is expanded, showing files 'coveredaged.txt', 'go.mod', 'go.sum', 'hello\_test.go', and 'hello.go'. The main editor has two tabs: 'coveredaged.txt' and 'hello.go'. The 'coveredaged.txt' tab is active, showing the following content:

```
1 mode: set
2 greeting/hello.go:6.32,7.20 1 1
3 greeting/hello.go:10.2,10.39 1 1
4 greeting/hello.go:7.20,9.3 1 0
5
```

The 'hello.go' tab shows the source code:

```
1 package greeting
2
3 import "fmt"
4
5 // Hello greet user based on the provided argu
6 func Hello(user string) string {
7     if len(user) == 0 {
8         return "Hello Dude !"
9     }
10    return fmt.Sprintf("Hello %v!", user)
11 }
12
```

At the bottom, the TERMINAL panel shows the command 'go test -coverprofile=coveredaged.txt' and its output:

```
➤ yauritux/greeting → go test -coverprofile=coveredaged.txt
PASS
coverage: 66.7% of statements
ok      greeting      0.005s
```

Below the terminal output, the 'ls' command is run, showing the files 'go.sum', 'coveredaged.txt', 'go.mod', 'hello.go', and 'hello\_test.go'.

1: zsh



# Convert coverage file to HTML

The screenshot shows the Visual Studio Code interface with two editor windows and a terminal at the bottom.

**EXPLORER (Left Panel):** Displays a file tree for a project named 'greeting'. The 'greeting' folder is expanded, showing files: 'covered.html' (with a code icon), 'covered.txt', 'go.mod', 'go.sum', 'hello\_test.go', and 'hello.go'.

**Editor Window 1 (Left):** Opened to 'covered.txt'. The path is 'src > github.com > yauritux > greeting > covered.txt'. The content is:

```
1 mode: set
2 greeting/hello.go:6.32,7.20 1 1
3 greeting/hello.go:10.2,10.39 1 1
4 greeting/hello.go:7.20,9.3 1 0
5
```

**Editor Window 2 (Right):** Opened to 'hello.go'. The path is 'src > github.com > yauritux > greeting > hello.go > Hello'. The content is:

```
1 package greeting
2
3 import "fmt"
4
5 // Hello greet user based on the provided argument
6 func Hello(user string) string {
7     if len(user) == 0 {
8         return "Hello Dude !"
9     }
10    return fmt.Sprintf("Hello %v!", user)
11 }
12
```

**Terminal (Bottom):** Shows the command used to generate the HTML coverage report:

```
yauritux/greeting → go tool cover -html=covered.txt -o covered.html
yauritux/greeting →
```

**Page-Footer:** A small orange logo with the number '5' is located in the bottom right corner.

# Table Driven Tests

- <https://dave.cheney.net/2019/05/07/prefer-table-driven-tests>



# Running all Test Files

- To test all the packages in a module, you can use `go test ./...` command in which `./...` matches all the packages in the module.
- `go test ./...` command goes through each package and runs the test files. You can use all the command line flags like `-v` or `-cover` as usual.



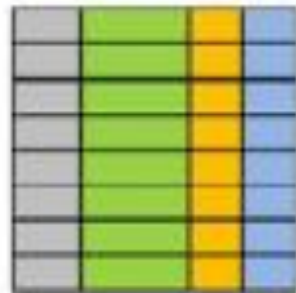


# Database

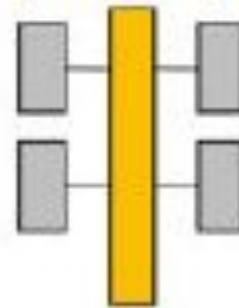


## SQL Database

### Relational

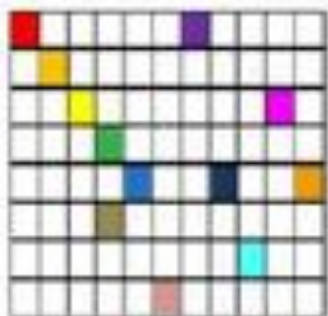


### Analytical (OLAP)

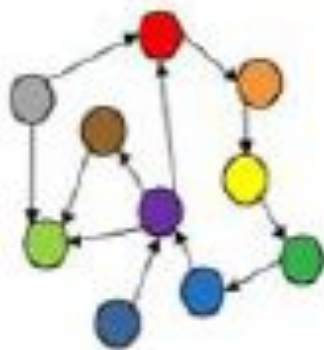


## NoSQL Database

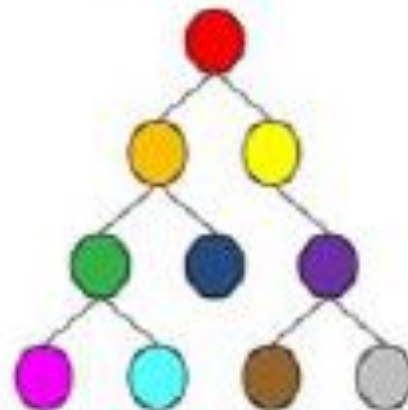
### Column-Family



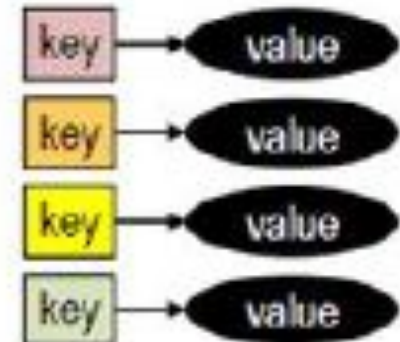
### Graph



### Document



### Key-Value







# SQL or NoSQL?

# SQL Database

1. Overview
2. Importing a Database Driver
3. Accessing the Database
4. Retrieving Result Sets
5. Modifying Data
6. Using Transactions
7. Using Prepared Statements
8. Handling Errors
9. Connection Pools





# Connect to MongoDB

```
package main

import (
    "context"
    "fmt"
    "log"
    "go.mongodb.org/mongo-driver/mongo"
    "go.mongodb.org/mongo-driver/bson"
    "go.mongodb.org/mongo-driver/mongo/options"
)

type Person struct {
    Name string
    Age  int
    City string
}

func main() {
    clientOptions := options.Client().ApplyURI("mongodb://mongodb:27017")
    client, err := mongo.Connect(context.TODO(), clientOptions)

    if err != nil {
        log.Fatal(err)
    }

    err = client.Ping(context.TODO(), nil)

    if err != nil {
        log.Fatal(err)
    }

    fmt.Println("Connected to MongoDB!")
}
```



# Writing to MongoDB

```
func main() {  
    ..  
    collection := client.Database("mydb").Collection("persons")  
  
    ruan := Person{"Ruan", 34, "Cape Town"}  
  
    insertResult, err := collection.InsertOne(context.TODO(), ruan)  
    if err != nil {  
        log.Fatal(err)  
    }  
    fmt.Println("Inserted a Single Document: ", insertResult.InsertedID)  
}
```



# Writing more than one document

```
func main() {  
    ..  
    collection := client.Database("mydb").Collection("persons")  
  
    ruan := Person{"Ruan", 34, "Cape Town"}  
    james := Person{"James", 32, "Nairobi"}  
    frankie := Person{"Frankie", 31, "Nairobi"}  
  
    trainers := []interface{}{james, frankie}  
  
    insertManyResult, err := collection.InsertMany(context.TODO(), trainers)  
    if err != nil {  
        log.Fatal(err)  
    }  
    fmt.Println("Inserted multiple documents: ", insertManyResult.InsertedIDs)  
}
```

# Updating Document

```
func main() {  
    ..  
    filter := bson.D  
    update := bson.D{  
        {"$inc", bson.D{  
            {"age", 1},  
        }},  
    }  
  
    updateResult, err := collection.UpdateOne(context.TODO(), filter, update)  
    if err != nil {  
        log.Fatal(err)  
    }  
    fmt.Printf("Matched %v documents and updated %v documents.\n",  
updateResult.MatchedCount, updateResult.ModifiedCount)  
}
```



# Reading from Document

```
func main() {  
    ..  
    filter := bson.D  
    var result Trainer  
  
    err = collection.FindOne(context.TODO(), filter).Decode(&result)  
    if err != nil {  
        log.Fatal(err)  
    }  
  
    fmt.Printf("Found a single document: %+v\n", result)  
  
    findOptions := options.Find()  
    findOptions.SetLimit(2)  
  
}
```







Check the exercises at  
<https://exercism.io>





# SUMMARY

You have learned how to cover / protect your code with Unit tests.

You have also learned how to persist your data permanently into the database.

Most of the time, you will use these things in your daily job as Go's Developer/Engineer

---





Thank You