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 3rd 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 ?

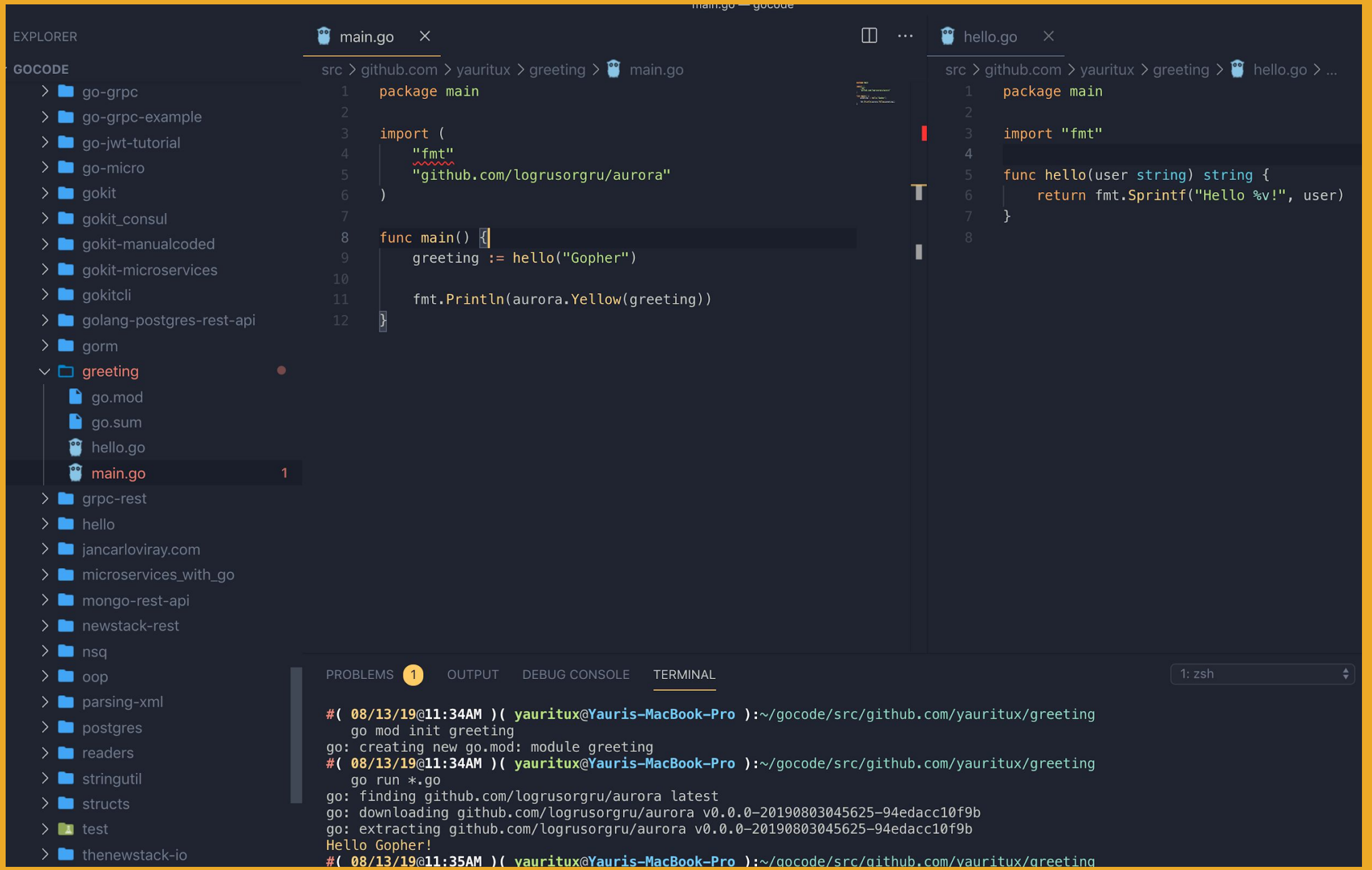
Whatever exports (exported-types) are available within the package need a unit test.

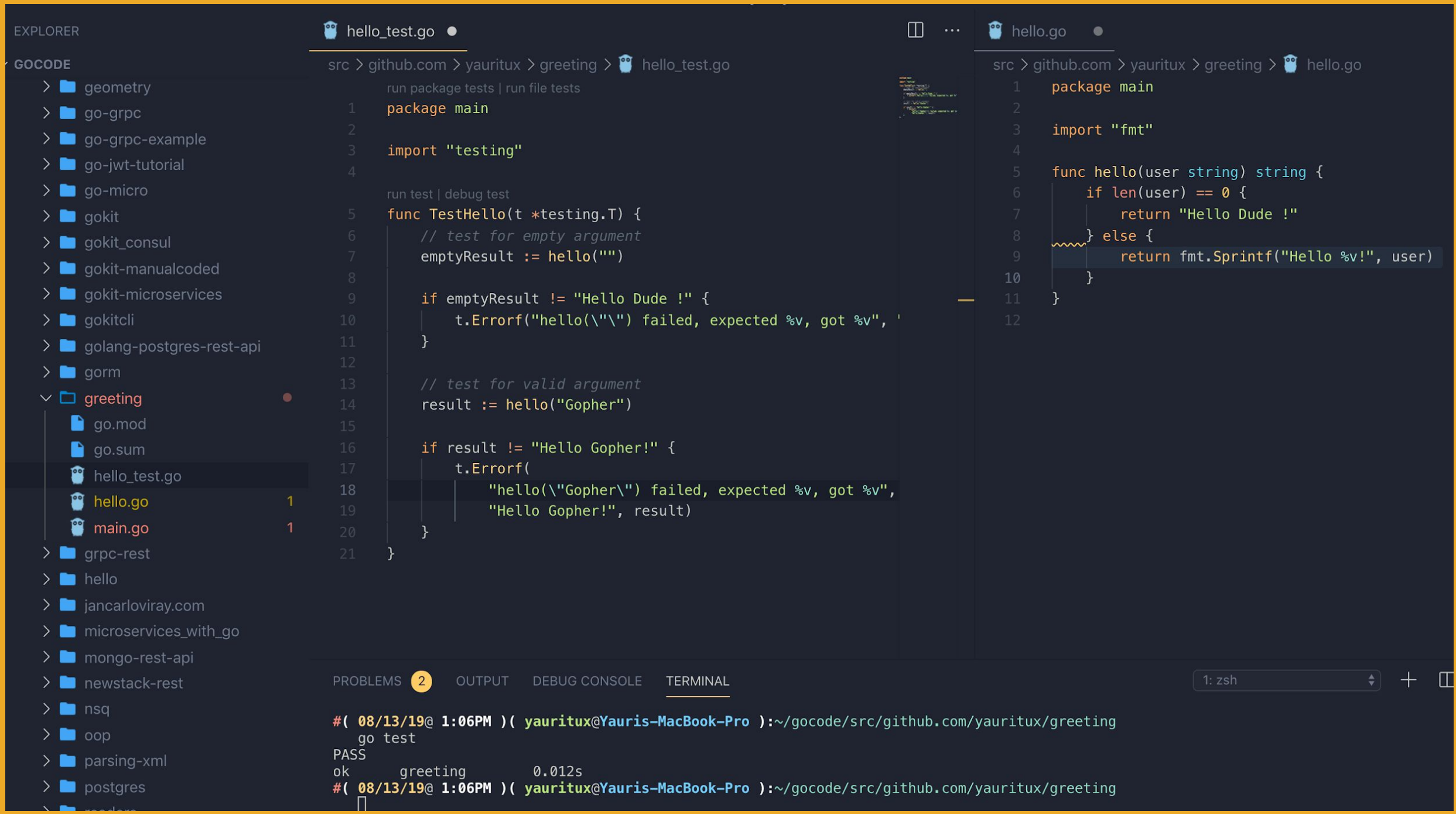
Basic Structure of Unit Test

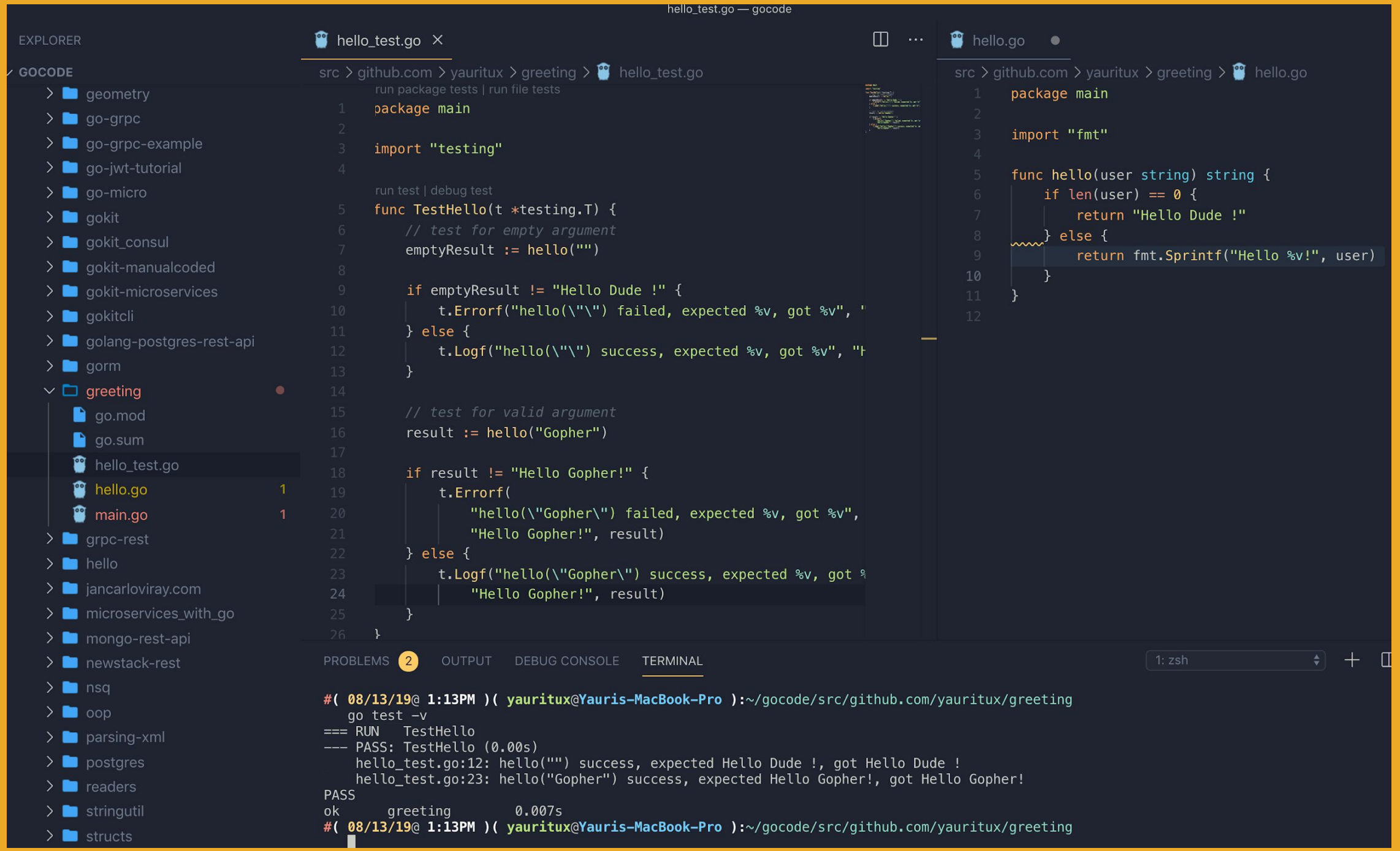
```
import “testing”

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
src > github.com > yauritux > greeting > 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
src > github.com > yauritux > greeting > hello.go > ...
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 > github.com > yauritux > greeting > hello_test.go
run package tests | run file tests
package main
import "testing"
run test | debug test
func TestHello(t *testing.T) {
// test for empty argument
emptyResult := hello("")
if emptyResult != "Hello Dude !" {
t.Errorf("hello(\"\") failed, expected %v, got %v", '
}
// test for valid argument
result := hello("Gopher")
if result != "Hello Gopher!" {
t.Errorf(
"hello(\"Gopher\") failed, expected %v, got %v",
"Hello Gopher!", result)
}
}
hello.go
src > github.com > yauritux > greeting > hello.go
package main
import "fmt"
func hello(user string) string {
if len(user) == 0 {
return "Hello Dude !"
} else {
return fmt.Sprintf("Hello %v!", user)
}
}
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

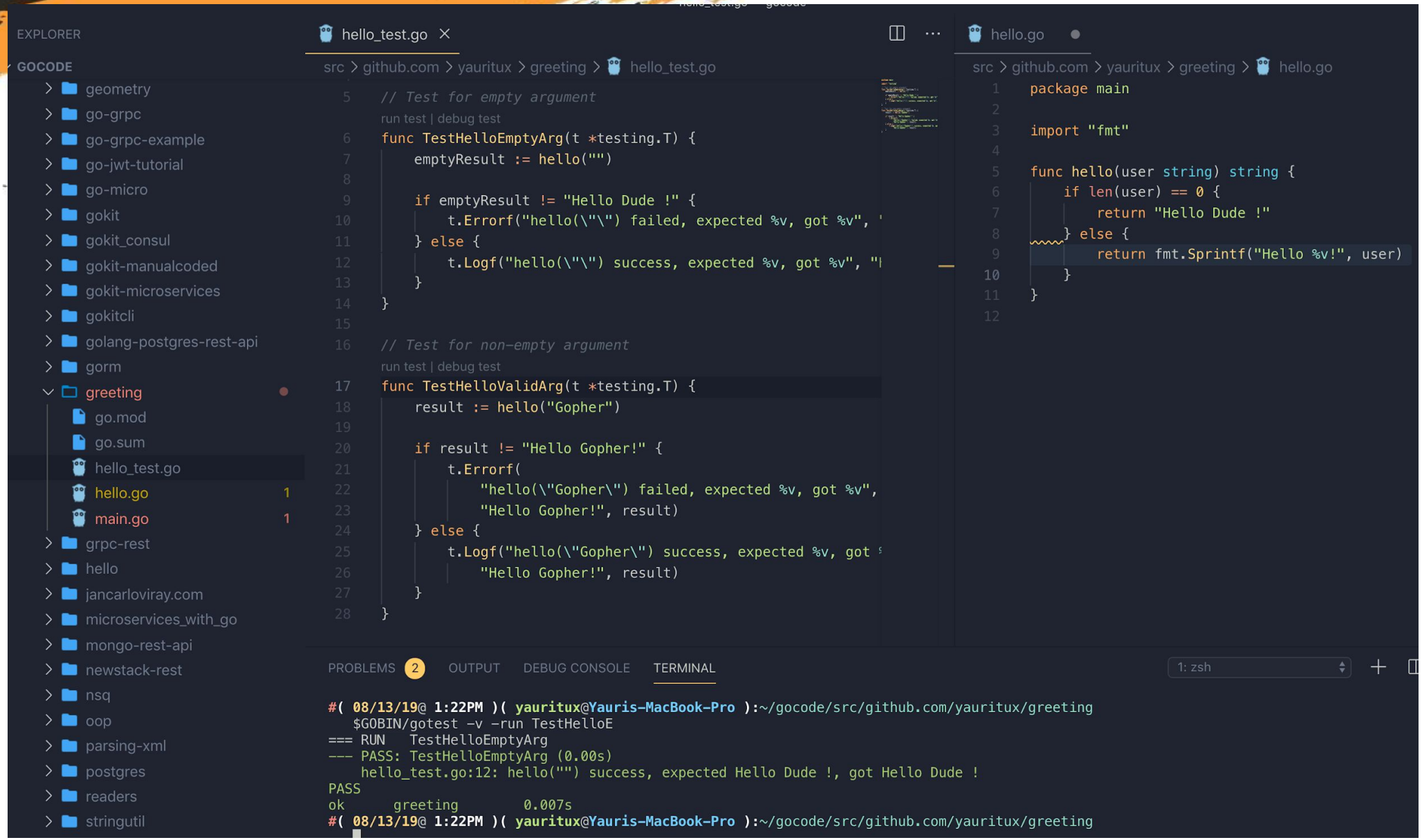

hello_test.go — gocode
EXPLORER
GOCODE
geometry
go-grpc
go-grpc-example
go-jwt-tutorial
go-micro
gokit
gokit_consul
gokit-manualcoded
gokit-microservices
gokitcli
golang-postgres-rest-api
gorm
greeting
go.mod
go.sum
hello_test.go
hello.go 1
main.go 1
grpc-rest
hello
jancarloviray.com
microservices_with_go
mongo-rest-api
newstack-rest
nsq
oop
parsing-xml
postgres
readers
stringutil
structs
hello_test.go
src > github.com > yauritux > greeting > hello_test.go
run package tests | run file tests
package main
import "testing"
run test | debug test
func TestHello(t *testing.T) {
// test for empty argument
emptyResult := hello("")
if emptyResult != "Hello Dude !" {
t.Errorf("hello(\"\") failed, expected %v, got %v", "
} else {
t.Logf("hello(\"\") success, expected %v, got %v", "H
}
// test for valid argument
result := hello("Gopher")
if result != "Hello Gopher!" {
t.Errorf(
"hello(\"Gopher\") failed, expected %v, got %v",
"Hello Gopher!", result)
} else {
t.Logf("hello(\"Gopher\") success, expected %v, got %
"Hello Gopher!", result)
}
}
hello.go
src > github.com > yauritux > greeting > hello.go
package main
import "fmt"
func hello(user string) string {
if len(user) == 0 {
return "Hello Dude !"
} else {
return fmt.Sprintf("Hello %v!", user)
}
}
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

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



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

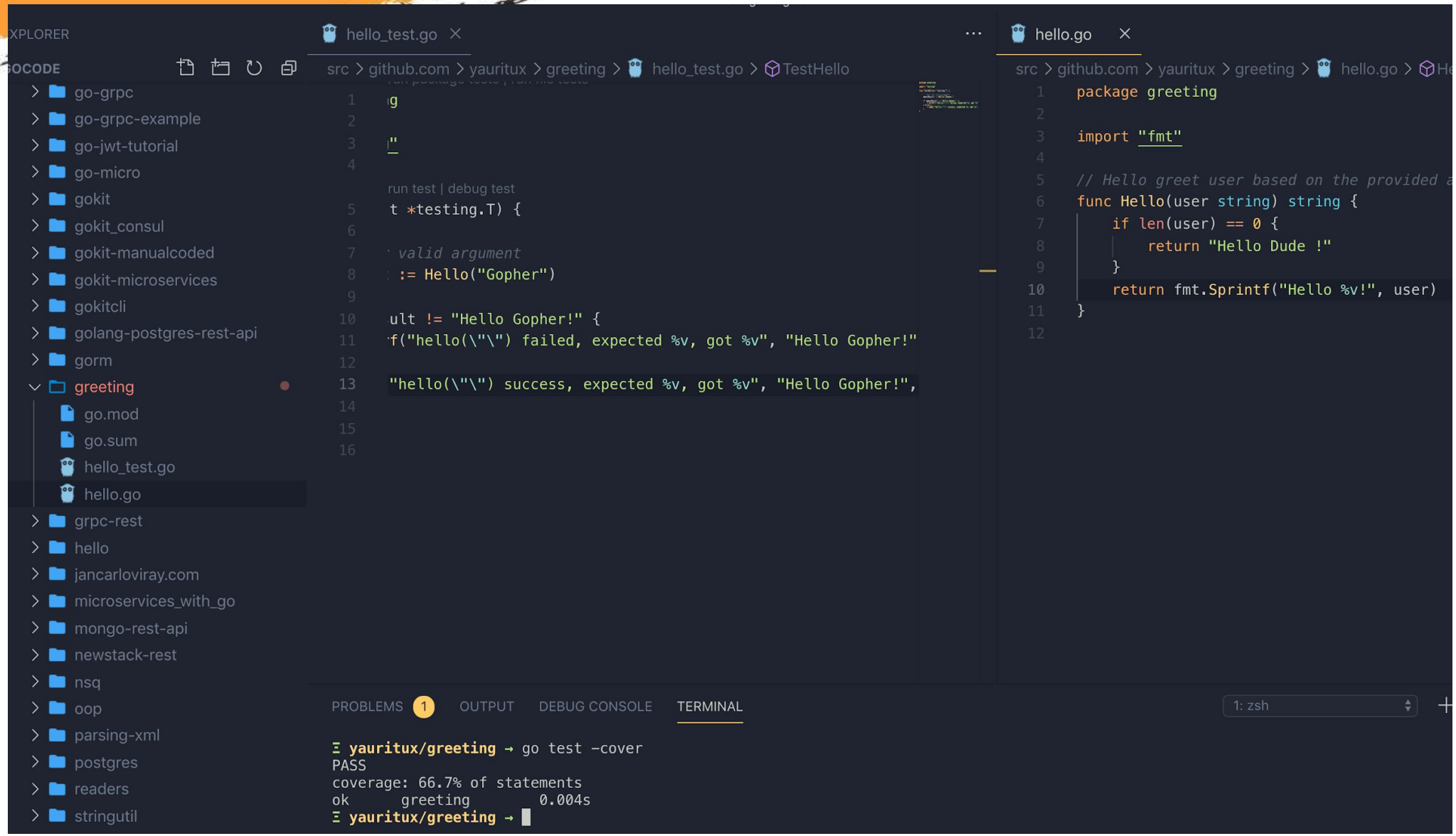
⚠️ *When you have test cases (_test.go files) in your executable(*main*) package, you can't simply execute* `go run *.go` *to run the project.* `*.go` *part also matches the test files (_test.go files) and* `go run` *command do not run them. You will get* `go run: cannot run *_test.go files (hello_test.go)`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

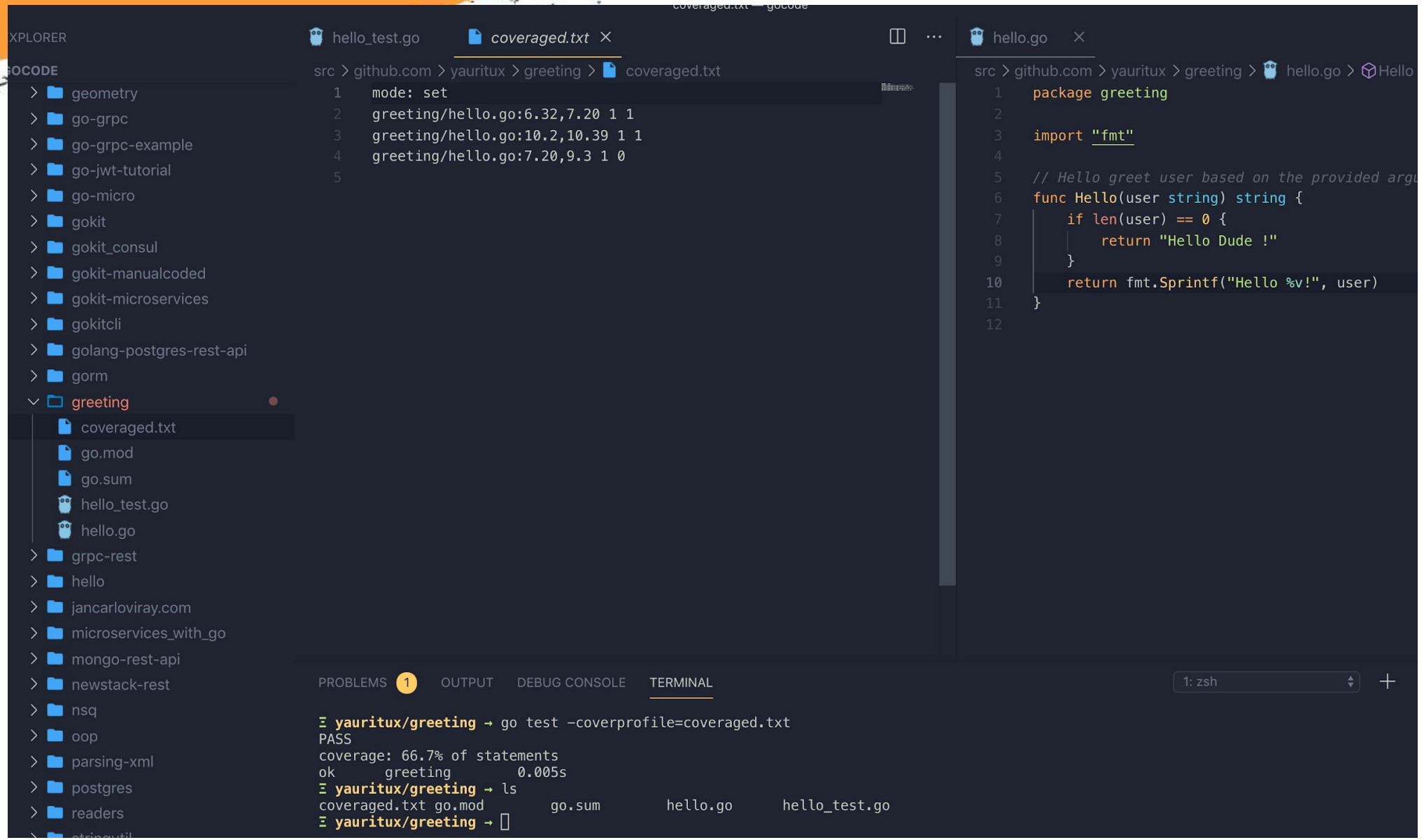
Test Coverage

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



produce test coverage report



Convert coverage file to HTML

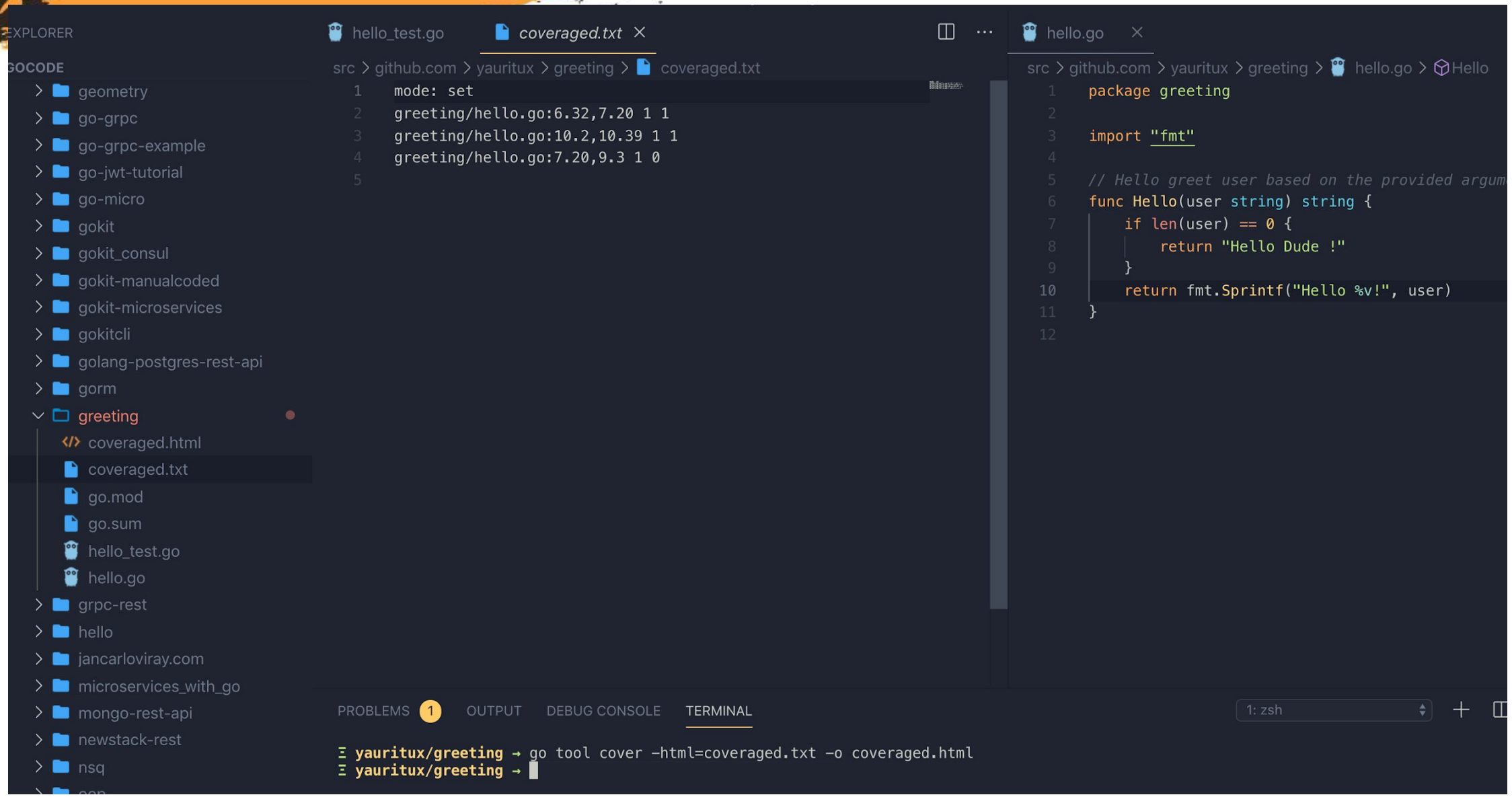


Table Driven Tests

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

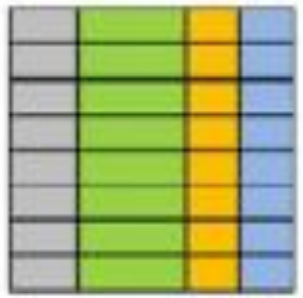
Running all Test Files

- 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.

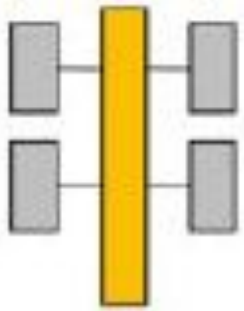
Database

SQL Database

Relational

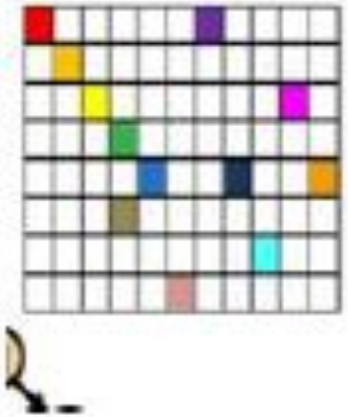


Analytical (OLAP)

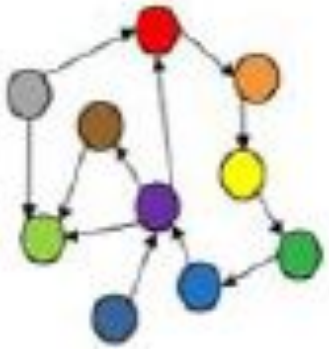


NoSQL Database

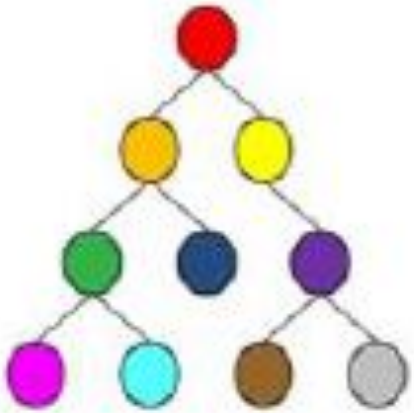
Column-Family



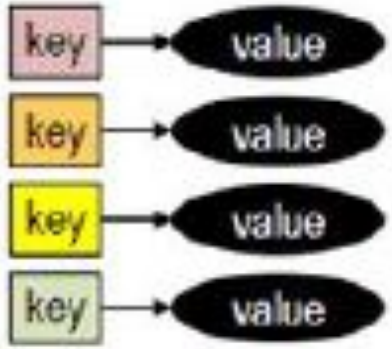
Graph



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

```go
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

```
funct 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