# Testing

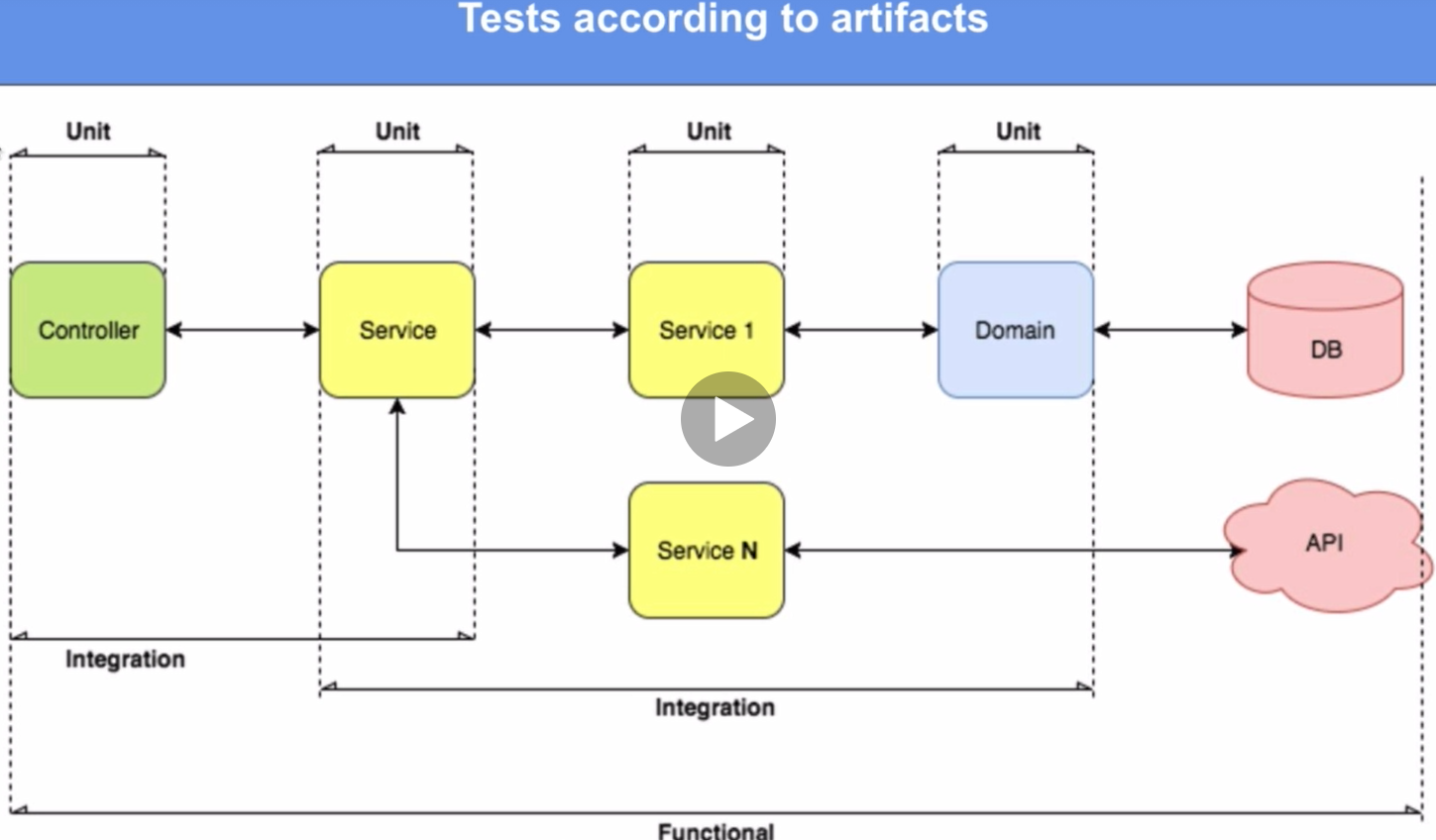
## Types of tests

* **Unit tests 🡪** *White box tests*. This means that you can have access to the source code you are creating, and you can look to the different corner cases or returns that your business logic has a given method or function.

They are key as; they not only determine if that part of the application is working or not. They also determine what part of the application is not working after a change is done in that part of the code.

* **Integration tests** 🡪 Tests that test 2 or more different functions that are calling each other. You can have integration between layers. So, integration tests are good for checking that the modules created interact correctly between them
* **Functional tests** 🡪 *Black box tests*. We call it like this because what we are doing is, running the application and performing an operation with an expected response. If the response is OK, the test will pass, if the response returned is different, it will fail…indicating that the behaviour has changed, and something is going wrong

The main difference is that in these tests you don’t why is failing, because there are many factors that could cause this to fail.

****

## Pyramid of tests

Rule of 80-15-5:

* 80% of the application tested with unit tests
* 15% of the application tested with integration tests
* 5% of the application tested with functional tests

- In order for Go to know that this is a test 2 things need to be declared:  
 - The name of the class must finalize in "\_test"  
 - The name of the test function must start with "Test"

// El profesor comenta que se debería de testear con valores para que, si cambias el string, pete. Pero yo al final  
// lo que busco es testear la funcionalidad, no el string que devuelve literalmente.  
assert.EqualValues(t, http.*StatusNotFound*, err.StatusCode)  
assert.EqualValues(t, "not\_found", err.Code)  
assert.EqualValues(t, "User 0 was not found", err.Message)

## Benchmarks

They are used in order to examine the performance of your Go code.

Benchmarks should be places inside test go files. Writing a benchmark is similar to writing a test, as they share the infrastructure from the testing package. Some of the key differences are:

* Benchmark functions start with Benchmark not Test
* Benchmark functions are run several times by the testing package. The value of b.N will increase each time until the benchmark runner is satisfied with the stability of the benchmark. This has some important ramifications
* Each benchmark must execute the code under test b.N times. The for loop in the benchmark will be present in every benchmark function.

Example:

func BenchmarkBubbleSort10Elements(b \*testing.B) {  
 elements := getElements(10)  
 for i := 0; i < b.N; i++ {  
 BubbleSort(elements)  
 }  
}

Example of output:

BenchmarkBubbleSort10Elements

BenchmarkBubbleSort10Elements-12 187490859 6.26 ns/op

PASS

This means that the number of iterations to stabilize the time has been b.N= 187490859, where the time per operation has been 6.26 ns

## How to structure Go artifacts and mock

With Go artifacts we mean the different layers of the application that are being used. So, this section defines how to structure and mock these artifacts.

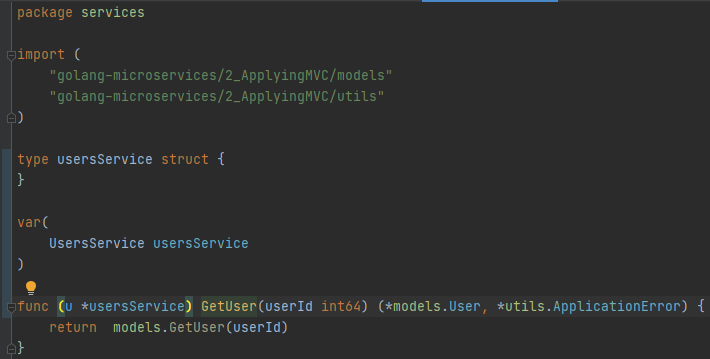
**In order to have controllers that only can use methods for a specific service, the best way to approach this is:**

1. Create an empty struct type inside the service
2. Create a public var function of the type defined before
3. Then, in the controller, only you’ll be able to call the functions for that service if you call the public variable

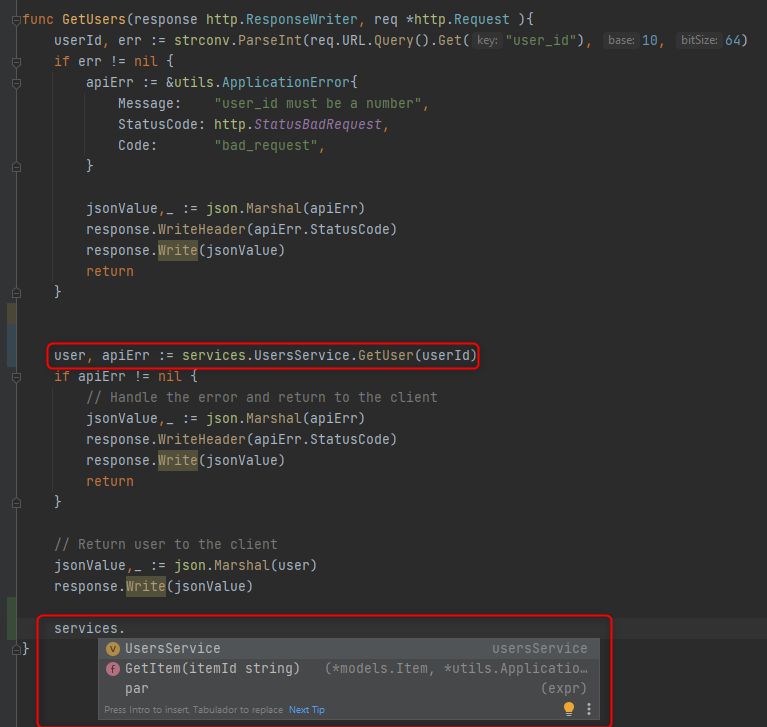
**NOTE:** If we do not define the var, the function will remain as “private”, meaning this that the function inside users\_service of type usersService will not be able outside of the package.

Example:

users\_service.go file:



users\_controller.go file:



**Notice that you will only be able to call GetUser function if you implement public UsersService variable, if not, you will not be able to call GetUser function.**

**WHY THIS IS USED?**

Because with this, you can have as many go files as you want in the same package, but the methods will not belong to the same package, as they belong to the public variable itself.