GO

BOOTCAMP

Clase en vivo

//Go Web

IT BOARDING

BOOTCAMP

Objetivos de la clase:

- Repasar conceptos de Request y Response
- Conocer el package httptest (nativo de Go)
- Aplicar un test unitario en nuestra API utilizando httptest

Índice

IT BOARDING
BOOTCAMP

1 | Repaso

IT BOARDING

BOOTCAMP

Request y response

Son las peticiones que el cliente le va a hacer a nuestra API.

USUARIO INGRESA A LA WEB

REQUEST

RESPONSE

El controlador recibe la petición y la envía al servicio correspondiente.

CONTROLADOR

El controlador devuelve la petición, ya sea de forma exitosa o con error.

El servicio hace la lógica y se comunica con el servidor de datos.

SERVICIO

El servicio cumple la lógica y define si la petición se cumplió o no.

REPOSITORIO

El repositorio o server devuelve los datos.

Son las respuestas que nuestro servidor le va a dar a esas peticiones.

httptest

//Package

IT BOARDING

BOOTCAMP

// httptest

Es un paquete de testing de Go que nos permite realizar *End to End Tests*, *Integration Tests*, *entre otros*. Una de las ventajas es que puede ser usado junto con librerías nativas de GO como con librerías de terceros (Chi).

httptest.NewRequest

Sirve para generar un Request, se puede definir el método HTTP, el body y el header.

```
{}  var req *http.Request = httptest.NewRequest("GET", "http://localhost:8080/...", nil)
```

httptest.NewRecorder

Es básicamente un Response. Se pasa en el handler del Server y con este se puede obtener la respuesta HTTP después de la ejecución.

{}

```
var res *httptest.ResponseRecorder = httptest.NewRecorder()
```

Realizando nuestro Test de Integración

Elegir dos funcionalidades del proyecto que revisamos las clases anteriores

Alta de producto:

```
{}  func (p *Products) Save(w http.ResponseWriter, r *http.Request) {
```

Obtener todos los productos:

```
{}  func (p *Products) Get(w http.ResponseWriter, r *http.Request) {
```

Test Unitario al método Get

Creamos un **caso de test** para verificar que se obtengan los productos

```
{}
```

```go
// Tests for Products - Get method
func TestProductsGet(t *testing.T) {
    t.Run("success to get products", func(t *testing.T) {
        // given

        // when

        // then

    })
}
```

Test Unitario al método Get

Arrange / Given: preparamos el test del **handler**

{}

```
t.Run("success to get products", func(t *testing.T) {
    // given
    db := map[int]storage.ProductAttributes{
        1: {Name: "product 1", Type: "type 1", Count: 1, Price: 1.1},
        2: {Name: "product 2", Type: "type 2", Count: 2, Price: 2.2},
    }
    st := storage.NewProductsMap(db, 2)
    hd := handler.NewProducts(st)

    // when
})
```

Test Unitario al método Get

Act / When: preparamos el ***http.Request** y el ***http.ResponseRecorder** y ejecutamos el **handler**

{}

```
t.Run("success to get products", func(t *testing.T) {
    // when
    req := httptest.NewRequest("GET", "/products", nil)
    res := httptest.NewRecorder()
    hd.Get(res, req)

    // then
})
```

Test Unitario al método Get

Assert / Then: verificamos los elementos **code**, **body** y **headers** esperados

{}

```
t.Run("success to get products", func(t *testing.T) {
    // then
    expectedCode := http.StatusOK
    expectedBody := `{"message":"success to get products","data":{
        "1":{"name":"product 1","type":"type 1","count":1,"price":1.1},
        "2":{"name":"product 2","type":"type 2","count":2,"price":2.2}
    }}`
    expectedHeader := http.Header{"Content-Type": []string{"application/json"}}
    require.Equal(t, expectedCode, res.Code)
    require.JSONEq(t, expectedBody, res.Body.String())
    require.Equal(t, expectedHeader, res.Header())
})
```

Test Unitario al método Save

Creamos un **caso de test** para verificar que el producto se crea correctamente

```
{}
```

```go
// Tests for Products - Save method
func TestProductsSave(t *testing.T) {
    t.Run("success to save a product", func(t *testing.T) {
        // given

        // when

        // then

    })
}
```

Test Unitario al método Save

Arrange / Given: preparamos el test del **handler**

{}

```
t.Run("success to save a product", func(t *testing.T) {
    // given
    st := storage.NewProductsMap(make(map[int]storage.ProductAttributes), 0)
    hd := storage.NewProducts(st)

    // when
})
```

Test Unitario al método Save

Act / When: preparamos el ***http.Request** y el ***http.ResponseRecorder** y ejecutamos el **handler**

{}

```
t.Run("success to save a product", func(t *testing.T) {
    // when
    req := httptest.NewRequest("POST", "/products", strings.NewReader(
        `{"name": "product 1", "type": "type 1", "count": 1, "price": 1.1}`,
    ))
    res := httptest.NewRecorder()
    hd.Save(res, req)

    // then
})
```

Test Unitario al método Save

Assert / Then: verificamos los elementos **code**, **body** y **headers** esperados

{}

```
t.Run("success to save a product", func(t *testing.T) {
    // then
    expectedCode := http.StatusCreated
    expectedBody := `{"message":"success to create a product","data":{
        "id":1,"name":"product 1","type":"type 1","count":1,"price":1.1
    }}`
    expectedHeader := http.Header{"Content-Type": []string{"application/json"}}
    require.Equal(t, expectedCode, res.Code)
    require.JSONEq(t, expectedBody, res.Body.String())
    require.Equal(t, expectedHeader, res.Header())
})
```

A codear...

3 | Live Coding

IT BOARDING

BOOTCAMP

Live Coding

Conclusiones

En esta clase terminamos de afianzar los conceptos de request y response.

Además aprendimos más sobre los test unitarios, su aplicación dentro del código y cómo usarlos para probar nuestras aplicaciones

Gracias.

IT BOARDING

BOOTCAMP

Actividad