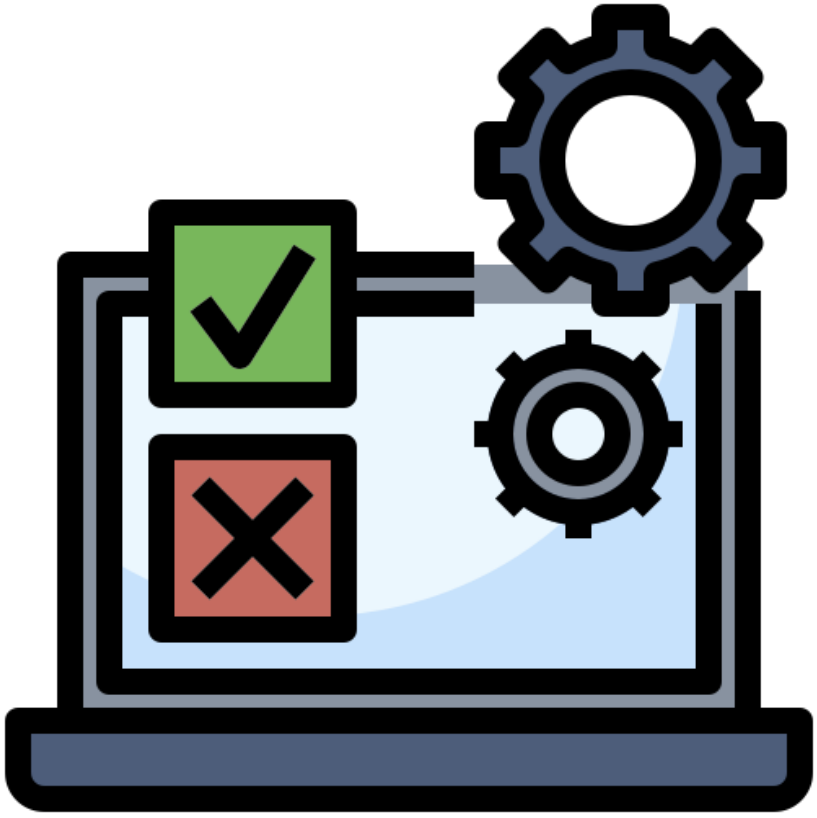
Testing

PHP

Especialidad PHP



Felipe Izquierdo Romero

Índice

1. Descripción del problema

Crear los test unitarios para una API de gestión de productos comprobando que todos devuelven una respuesta correcta tras una operación y corroborando que los datos sean correctos.

2. Desarrollo

Para este lab reutilizando el ejercicio anterior y configurando cada contenedor de manera correcta he pasado a realizar escribir las pruebas y probarlos en cada contenedor.

En Symfony realicé la migración primero y a continuación realicé cada test:

- Test Index: Comprueba que la response es correcta, devuelve un json lo paso array y es válido y que tenga todas las claves/atributos el producto.

```php
    public function testIndex()
    {
        $client = static::createClient();
        $client->request('GET','/');

        $content = $client->getResponse()->getContent();
        $products = json_decode($content, true);


        $this->assertResponseIsSuccessful();
        $this->assertIsArray($products);
        $this->assertArrayHasKey('id',$products[0]);
        $this->assertArrayHasKey('name',$products[0]);
        $this->assertArrayHasKey('price',$products[0]);
        $this->assertArrayHasKey('stock',$products[0]);
    }
```

Ilustración 1- Test Index Symfony

- Test Create: crea un producto y comprueba que la respuesta sea la esperada.

```php
public function testCreate()
{
    $client = static::createClient();
    $data = [
        'name' => 'Prod_Test',
        'description' => 'Descripcion_Test',
        'price' => "100.60",
        'stock' => 50
    ];

    $client->request('POST', '/create', [], [], ['CONTENT_TYPE' => 'application/json'], json_encode($data));

    $this->assertResponseIsSuccessful();
    $content = $client->getResponse()->getContent();
    $response = json_decode($content, true);

    $this->assertEquals('Producto añadido correctamente', $response[1]);
}
```

Ilustración 2- Test Create Symfony

- Test Show One: hace la prueba para un producto existente y uno no existente.

```php
public function testShowOne(): void
{
    $client = static::createClient();

    // Prueba para un producto existente
    $client->request('GET', '/showOne/2');
    $this->assertResponseIsSuccessful();

    // Prueba para un producto no existente
    $client->request('GET', '/showOne/999');
    $this->assertResponseStatusCodeSame(404);
}
```

Ilustración 3- Test Show One Symfony

- Test Delete: Hace la prueba borrando un producto existente y comprobando que la respuesta sea la esperada y también para uno que no existe.

```php
public function testDelete(): void
{
    $client = static::createClient();

    // Prueba para un producto existente
    $client->request('DELETE', '/delete/6');
    $this->assertResponseIsSuccessful();
    $content = $client->getResponse()->getContent();
    $response = json_decode($content, true);

    $this->assertEquals('Producto borrado correctamente', $response[1]);

    // Prueba para un producto no existente
    $client->request('DELETE', '/delete/999');
    $this->assertResponseStatusCodeSame(404);
}
```

Ilustración 4- Test Delete Symfony

- Test Modify: modifica un producto existente y verifica su respuesta sea la esperada y prueba a modificar un producto que no exista.

```php
public function testModify(): void
{
    $client = static::createClient();
    $data = [
        'name' => 'Updated Product',
        'price' => '150',
        'stock' => 75
    ];

    // Prueba para un producto existente
    $client->request('PUT', '/modify/2', [], [], ['CONTENT_TYPE' => 'application/json'], json_encode($data));
    $this->assertResponseIsSuccessful();
    $content = $client->getResponse()->getContent();
    $response = json_decode($content, true);
    $this->assertEquals('Producto modificado', $response['status']);

    // Prueba para un producto no existente
    $client->request('PUT', '/modify/999', [], [], ['CONTENT_TYPE' => 'application/json'], json_encode($data));
    $this->assertResponseStatusCodeSame(404);
}
```

Ilustración 5- Test Modify Symfony

En Laravel importante crear el test e importar:

- Illuminate\Foundation\Testing\WithoutMiddleware

De lo contrario no podrás realizar diversos test que impliquen formularios de datos.

Las pruebas serían las siguientes:

- Test Create: crea un producto y comprueba el response que devuelve el producto creado si los datos son los correctos.

```php
public function testCreate()
{
    $data=[
        'name' => 'Creacion_Test_Laravel',
        'description' => 'Descripcion_Test_Laravel',
        'price' => 0.99,
        'stock' => 33
    ];

    $response = $this->post('/',$data);

    $response->assertStatus(Response::HTTP_CREATED);
    $this->assertEquals($data['name'],$response->json('name'));
    $this->assertEquals($data['description'],$response->json('description'));
    $this->assertEquals($data['price'],$response->json('price'));
    $this->assertEquals($data['stock'],$response->json('stock'));
}
```

Ilustración 6- Test Create Laravel

- Test Update: modifica un producto y comprueba el response que devuelve el producto creado si los datos son los correctos.

```php
public function testUpdate()
{
    $id = 3;
    $dataList=[
        'name'=>'ModificacionLaravel',
        'description'=>'ModificacionLaravel',
        'price'=>0.99,
        'stock'=>8
    ];
    $data=[
        "changes" => [
        ["field" => "name", "value" => $dataList['name']],
        ["field" => "description", "value" => $dataList['description']],
        ["field" => "price", "value" => $dataList['price']],
        ["field" => "stock", "value" => $dataList['stock']]
        ]
    ];


    $response = $this->put("/modify_json/".$id,$data);


    $producto = Producto::find($id);

    $response->assertStatus(Response::HTTP_OK);
    $this->assertEquals($dataList['name'], $producto->name);
    $this->assertEquals($dataList['description'], $producto->description);
    $this->assertEquals($dataList['price'], $producto->price);
    $this->assertEquals($dataList['stock'], $producto->stock);

}
```

Ilustración 7- Test Update Laravel

- Test ShowOne: Llama al enpoint de mostrar el primer producto de la lista y comprueba el status del response.

```php
67      public function testShowOne()
68      {
69          $productos = Producto::all();
70
71          $response = $this->get('/'.$productos[0]->id);
72          $response->assertstatus(200);
73      }
```

Ilustración 8- Test Show One Laravel

- Test Delete: Borra el ultimo elemento de la lista y comprueba si la response es correcta.

```php
75      public function testDelete()
76      {
77          $productos = Producto::all();
78          $lastProductId = $productos->last()->id;
79
80          $response = $this->delete('/delete/'.$lastProductId);
81          $response->assertstatus(201);
82      }
83  }
```

Ilustración 9- Test Delete Laravel

A continuación pasamos a CodeIgniter donde hay que configurar el archivo "phpunit.xml.dist" de la siguiente manera:

```xml
50      <!-- Uncomment to provide your own database for testing-->
51      <env name="database.tests.hostname" value="3Frameworks-mysql_test"/>
52      <env name="database.tests.database" value="3Frameworks_db"/>
53      <env name="database.tests.username" value="app_user"/>
54      <env name="database.tests.password" value="982hhfn9i24ugh925hg9235thrg39w28014t08hij"/>
55      <env name="database.tests.DBDriver" value="MySQLi"/>
56      <env name="database.tests.DBPrefix" value=""/>
```

Ilustración 10- Configuracion Test CodeIgniter

Los Test de CodeIgniter son los siguientes:

- Test Index: Realiza la petición y comprueba si la respuesta es correcta y en formato JSON.

```php
12      public function testIndex()
13      {
14          $result = $this->call('get','/');
15
16          $this->assertTrue($result->isOk());
17          $this->assertJson($result->getJSON());
18
19      }
```

Ilustración 11- Test Index CodeIgniter

- Test Update: Modifica el ultimo producto de la lista y verifica que se modificaron bien los datos.

```php
public function testUpdate()
{
    $data = [
        'name' => 'Nombre tester',
        'description' => 'Descriptiton Test',
        'price' => 0.99,
        'stock' => 9
    ];

    $this->withHeaders(['Content-Type' => 'application/json']);

    $model = new \App\Models\Producto();
    $lastProductId = $model->selectMax('id')->first()['id'];

    $result = $this->withBody(json_encode($data),'application/json')->put('modify/'.$lastProductId,$data);

    $updatedProduct = $model->find($lastProductId);

    $this->assertTrue($result->isOk());
    $this->assertNotNull($updatedProduct);
    $this->assertEquals($data['name'],$updatedProduct['name']);
    $this->assertEquals($data['description'],$updatedProduct['description']);
    $this->assertEquals($data['price'],$updatedProduct['price']);
    $this->assertEquals($data['stock'],$updatedProduct['stock']);
}
```

Ilustración 12- Test Update CodeIgniter

- Test Create: Crea un producto y verifica que existe el ultimo producto de la lista que sería el recién creado.

```php
public function testCreate()
{
    $data = [
        'name' => 'Nombre tester Creacion',
        'description' => 'Descriptiton Test Creacion',
        'price' => 0.99,
        'stock' => 9
    ];

    $this->withHeaders(['Content-Type' => 'application/json']);

    $model = new \App\Models\Producto();

    $result = $this->withBody(json_encode($data),'application/json')->post('/create',$data);

    $lastProductId = $model->selectMax('id')->first()['id'];
    $createdProduct = $model->find($lastProductId);

    $this->assertTrue($result->isOk());
    $this->assertNotNull($createdProduct);
    $this->assertEquals($data['name'],$createdProduct['name']);
    $this->assertEquals($data['description'],$createdProduct['description']);
    $this->assertEquals($data['price'],$createdProduct['price']);
    $this->assertEquals($data['stock'],$createdProduct['stock']);
}
```

Ilustración 13- Test Create CodeIgniter

- Test Show One: Selecciona el ultimo producto de la lista y verifica que sean los mismos datos del producto creado anteriormente.

```php
public function testShowOne()
{
    //Data del test anterior para verificar
    $data = [
        'name' => 'Nombre tester Creacion',
        'description' => 'Descriptiton Test Creacion',
        'price' => 0.99,
        'stock' => 9
    ];

    $model = new \App\Models\Producto();
    $lastProductId = $model->selectMax('id')->first()['id'];
    $result = $this->call('get','/'.$lastProductId);

    $findedProduct = $model->find($lastProductId);

    $this->assertTrue($result->isOk());
    $this->assertJson($result->getJSON());

    //Comprueba con el test de creacion anterior para ver
    //si se encontro un producto por id correctamente
    $this->assertEquals($data['name'],$findedProduct['name']);
    $this->assertEquals($data['description'],$findedProduct['description']);
    $this->assertEquals($data['price'],$findedProduct['price']);
    $this->assertEquals($data['stock'],$findedProduct['stock']);
}
```

Ilustración 14- Test Show One

- Test Delete: Borra el ultimo producto de la lista y comprueba que es null intentando acceder a él.

```php
public function testDelete()
{
    $model = new \App\Models\Producto();
    $lastProductId = $model->selectMax('id')->first()['id'];

    $result = $this->delete('delete/'.$lastProductId);

    $deletedProduct = $model->find($lastProductId);

    $this->assertTrue($result->isOk());
    $this->assertNull($deletedProduct);

}
```

Ilustración 15- Test Delete CodeIgniter

Por último para ejecutar los test en cada framework.

- Symfony:

```
php bin/phpunit
```

- Laravel:

```
php artisan test
```

- CodeIngiter:

```
php vendor/bin/phpunit
```