Tests sur Symfony

1-Les tests unitaires

2-Les tests fonctionnels

Ababacar Fall

Introduction

Tests sur symfony

Introduction

Les tests nous permettent de s’assurer du bon fonctionnement de notre code.Il existe differents types de tests dont les Tests unitaires et les Tests Fonctionnels.

Les tests unitaires vérifient que chaque méthode et chaque fonction fonctionne correctement. Chaque test doit être aussi indépendante que possible des autres.

Les tests fonctionnels nous permettent de tester notre application de bout en bout : de la requête faite par un navigateur jusqu'à la réponse envoyée par le serveur. Ils testent toutes les couches d'une application : le routage, le modèle, les actions et les Templates.

Les outils de test

Tests sur symfony

Les outils de test

Il existe plusieurs outils de test dont **lime** de symfony ou **phpunit** est un framework open source de tests unitaires dédié au langage de programmation PHP.
Dans notre cas , nous allons utiliser **phpunit**.
Phpunit utilise des methodes qui vont nous permettre de valider nos tests.
Ces methodes sont les **Assertions** et les **expectations.**
Une asserton permet de valider qu'une valeur est bien celle attendue.
Par exemple, on a l'assertion **assertEquals(2, 1+1) .**
Le 2 est le resultat attendu et le 1+1 est l'operation dont on attend le resultat soit egal a 2. Ce test est donc valide.
Mais lorsqu'on met **assertEquals(5, 1+1) ,**ce test sera invalidé car 1+1 ne donnnent pas 5.
Il existe une multitude d'assertions selon nos besoins.
Documetation officielle:
https://phpunit.readthedocs.io/en/9.2/assertions.html

Les tests unitaires

Tests sur symfony

Les tests unitaires

Les tests unitaires vérifient que chaque méthode et chaque fonction fonctionne correctement. Chaque test doit être aussi indépendante que possible des autres.
Pour pouvoir effectuer nos tests unitaires sur Symfony avec phpunit, nous allons d'abord installer phpunit avec la commande
`php bin/phpunit`
Un dossier **tests** sera créé dans notre projet et c'est là que nous allons ecrire tout nos tests

Exemple de test unitaire

Nous allons effectuer notre premier test unitaire.Pour cela, on va créer un controller et une fonction qui va nous permettre d'additionner deux entiers et nous retourne le resultat.

```php
<?php
namespace App\Controller;
use Symfony\Bundle\FrameworkBundle\Controller\AbstractController;

class CalculatriceController extends AbstractController{
    public function additionneur($a,$b){
        return $a+$b;
    }

}
```

Exemple de test unitaire

Ensuite nous allons nous rendre dans notre dossier tests puis créer un nouveau fichier **CalculatriceTest.php**. Ce fichier doit se terminer par test par convention.On va créer alors une class **CalculatriceTest** qui va etendre de **TestCase**(classe propre a phpunit). Apres cela,on peut configurer notre fonction de test et lui donner le resultat qu'on attend avec les assertions.

```php
<?php
namespace App\Tests;

use PHPUnit\Framework\TestCase;
use App\Controller\CalculatriceController;
use Symfony\Component\HttpFoundation\Response;

Run | Show in Test Explorer
class CalculatriceControllerTest extends TestCase{
Run | Show Log | Show in Test Explorer
public  function testingAdd()
{
    $add=new CalculatriceController();

    $this->assertEquals(7, $add->additionneur(4,3));}


}
```

On crée une fonction **testingAdd()** en y instenciant notre controller **CalculatriceController** puis on fait le test avec **assertEquals** qui permet de verifier si la valeur retournée par notre méthode(`$add->additionneur(4,3)` est egal à 7.

Exemple de test unitaire

A présent on peut donc exécuter notre test et voir le résultat.
Pour cela, nous allons utiliser la commande `php bin/phpunit` Cette commande exécute tout les tests qui seront dans notre fichier tests.
Pour exécuter un test en particulier, il faudra utiliser la commande suivante
`php bin/phpunit tests/CalculatriceController.php`

```
PS C:\Users\ACER\Documents\testpoo\testSymfony> php bin/phpunit tests/CalculatriceTest.php
PHPUnit 7.5.20 by Sebastian Bergmann and contributors.

Testing App\Tests\CalculatriceControllerTest
.                                                                   1 / 1 (100%)

Time: 325 ms, Memory: 6.00 MB

OK (1 test, 1 assertion)
```

Le test a bien été effectué et nous a renvoyé un message de succés.On va maintenant retester notre fonction mais cette fois ci avec une mauvaise valeur de retour.

Exemple de test unitaire

On test avec un mauvais résultat

```php
<?php
namespace App\Tests;

use PHPUnit\Framework\TestCase;
use App\Controller\CalculatriceController;
use Symfony\Component\HttpFoundation\Response;
use Symfony\Bundle\FrameworkBundle\Test\WebTestCase;

Run | Show in Test Explorer
class CalculatriceControllerTest extends TestCase{
Run | Show Log | Show in Test Explorer
public  function testingAdd()
{
    $add=new CalculatriceController();

    $this->assertEquals(174, $add->additionneur(4,3));}


}
```

Ici on teste 4+3 et on s'attend que le résultat soit 174.Ceci est évidemment un faux résultat.

```
1) App\Tests\CalculatriceControllerTest::testingAdd
Failed asserting that 7 matches expected 174.

C:\Users\ACER\Documents\testpoo\testSymfony\tests\CalculatriceTest.php:14

FAILURES!
Tests: 1, Assertions: 1, Failures: 1.
PS C:\Users\ACER\Documents\testpoo\testSymfony>
```

Apres avoir exécuté le test, il nous retourne 1 Failure et nous indique que le résultat devrait être 7 alors que nous lui avons renseigné 174.

Les tests fonctionnelles

Tests sur symfony

Les tests fonctionnels

Les tests fonctionnels nous permettent de tester notre application de bout en bout : de la requête faite par un navigateur jusqu'à la réponse envoyée par le serveur. Ils testent toutes les couches d'une application : le routage, le modèle, les actions et les Templates.
Pour pouvoir effectuer nos tests fonctionnels sur Symfony avec phpunit, nous allons d'abord installer phpunit avec la commande
`php bin/phpunit`
Un dossier **tests** sera créé dans notre projet et c'est là que nous allons ecrire tout nos tests

Exemple de test fonctionnel

Nous allons effectuer notre premier test fonctionnel.Pour cela, on va créer une entité personne,un controller et une fonction dans qui va nous permettre d’instancier notre entité puis de l’inserer dans notre base de données.

Exemple de test fonctionnel

Creation classe personne avec attributs nom,prenom et age.

```php
<?php

namespace App\Entity;

use App\Repository\PersonneRepository;
use Doctrine\ORM\Mapping as ORM;

/**
 * @ORM\Entity(repositoryClass=PersonneRepository::class)
 */
class Personne
{
    /**
     * @ORM\Id
     * @ORM\GeneratedValue
     * @ORM\Column(type="integer")
     */
    private $id;

    /**
     * @ORM\Column(type="string", length=255, nullable=true)
     */
    private $nom;

    /**
     * @ORM\Column(type="string", length=255, nullable=true)
     */
    private $prenom;

    /**
     * @ORM\Column(type="integer", nullable=true)
     */
    private $age;
```

Exemple de test fonctionnel

Creation du fichier AddPersonneTest.php dans le dossier tests
Ce fichier va nous permettre de tester l'ajout d'une Personne en simulant la requete d'un client comme le ferait un navigateur avec la methode createClient();
Cette methode va nous permettre de faire notre requete avec la **method(POST),** la route definie dans notre controller **PersonneController(/personne/add)** et les **donnees qu'on veut envoyer($data).**
On precise enfin le type de resultat attendu apres l'execution de la requete. Dans notre cas, on s'attend que ma requete nous renvoie une reponse de type **200**.L'assertion utilisée est la suivante:

```
$this->assertResponseStatusCodeSame(Response::HTTP_OK);
```

Exemple de test fonctionnel

```php
<?php
namespace App\Controller\Tests;

use Symfony\Component\HttpFoundation\Response;
use Symfony\Bundle\FrameworkBundle\Test\WebTestCase;

class AddPersonneTest extends WebTestCase{

    public function testaddingPersonne(){
        $client=static::createClient();  //on crée un client qui simule un navigateur pour pouvoir faire notre requete
        $data['nom']="fall";             //Donnees a envoyer dans la requete
        $data["prenom"]="Babs";           //Donnees a envoyer dans la requete
        $data["age"]=54;                 //Donnees a envoyer dans la requete
        $client->request('POST','/personne/add', $data);//On envoie une requete de type post sur la route configurée
                                                        //dans notre controller avec les données
        $this->assertResponseStatusCodeSame(Response::HTTP_OK); // on fait notre test en y precisant le type de reponse attendu
    }

}
```

Exemple de test fonctionnel

Creation controller PersonneController.php
Dans ce controller, on va configurer la route ainsi que la fonction qui va nous permettre d’ajouter une personne dans notre base de données.
On recupere d’abord les données envoyées dans la requete puis verifions si ces données sont valides pour ensuite les inserer dans notre base de donnée et enfin retourner la reponse.

Exemple de test fonctionnel

```php
<?php
namespace App\Controller;
use App\Entity\Fruit;
use App\Entity\Personne;
use Symfony\Component\HttpFoundation\Request;
use Symfony\Component\HttpFoundation\Response;
use Symfony\Component\Routing\Annotation\Route;
use Symfony\Component\HttpFoundation\JsonResponse;
use Symfony\Bundle\FrameworkBundle\Controller\AbstractController;
class PersonneController extends AbstractController
{
    /**
     * @Route("/personne/add", name="add_personne")
     */
    public function index(Request $request)
    {
        $data=$request->request->all();
        if(is_string($data["prenom"])&& is_string($data["nom"]) && is_int($data['age'])){
            $personne=new Personne();
            $personne->setPrenom( $data["prenom"]);
            $personne->setNom( $data["nom"]);
            $personne->setAge( $data["age"]);
            $em=$this->getDoctrine()->getManager();
            $em->persist($personne);
            $em->flush();
            return new JsonResponse("ok",Response::HTTP_OK,[],true);
        }
        return new JsonResponse("ok",Response::HTTP_FORBIDDEN,[],true);
    }
}
```


Merci