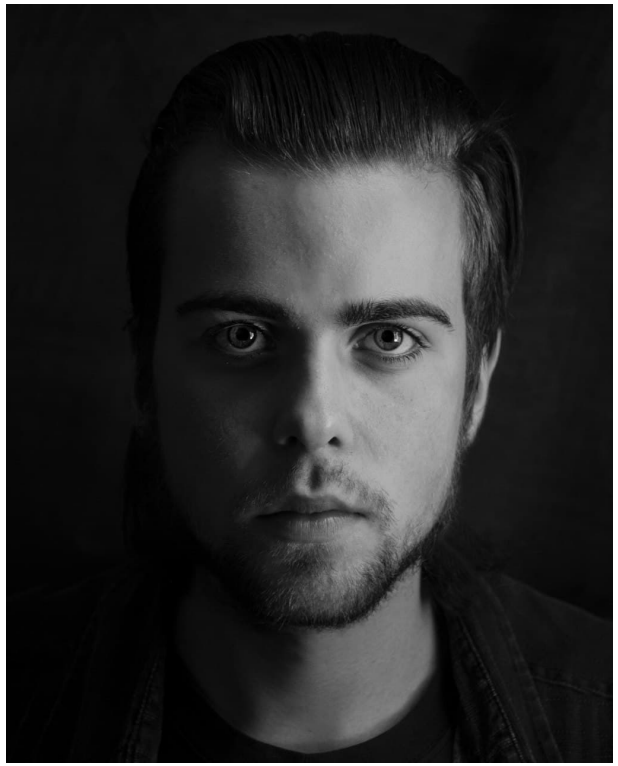
Веб разработка на PHP Symfony

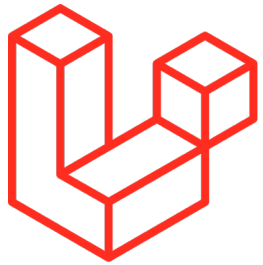
Тестирование

Introduction



Демьян Костельный
Middle PHP Developer

demian-kostelny-613b90151



Тема урока

Тестирование

План урока

1. Вступление в Unit и Function тестирование
2. Первый Unit тест с PHPUnit
3. Dataprovider
4. Тестовые двойники
5. Первый Functional тест
6. Разделяем test, prod, dev окружения
7. Фикстуры для тестов

Вступление в Unit и Function тестирование

Вступление в Unit и Function тестирование

Чтобы реализовать в Symfony приложении автоматическое тестирование - используется фреймворк PHPUnit.

PHPUnit - это PHP фреймворк, который позволяет писать юнит-тесты для PHP приложений. Юнит тестирование позволяет делать сразу же автоматическое тестирование без каких-то ручных действий. Для того чтобы начать использовать PHPUnit в своем приложении - нужно его сначала установить:

```
composer require --dev phpunit/phpunit symfony/test-pack
```

После этого для того чтобы запускать тесты можно будет использовать команду:

```
php bin/phpunit
```

Вступление в Unit и Function тестирование

Unit тест - представляет из себя механизм тестирования отдельной части кода.

Интеграционные тесты - это тесты которые работают с сервис контейнером приложения.

Application тесты - это тесты, которые написаны для тестирования целого приложения.

Перед тем как мы перейдем к написанию наших первых тестов хочу объяснить логику вызова тестов через консоль:

```
# Команда ниже вызовёт все тесты приложения
php bin/phpunit

# Ниже показанная команда вызовёт только те тесты которые в папке tests/Admin
php bin/phpunit tests/Admin

# И следующая команда вызывает только тест в файле tests/Admin/ClientTest.php
php bin/phpunit tests/Admin/ClientTest.php
```

Первый Unit тест с PHPUnit

Первый Unit тест с PHPUnit

Для того чтобы сгенерировать новый тест, можно воспользоваться специальной командой через maker-bundle:

```
php bin/console make:test
```

После ввода названия теста, maker-bundle спросит какого типа тест мы хотим создать:

```
Which test type would you like?:
 [TestCase       ] basic PHPUnit tests
 [KernelTestCase ] basic tests that have access to Symfony services
 [WebTestCase    ] to run browser-like scenarios, but that don't execute JavaScript code
 [ApiTestCase    ] to run API-oriented scenarios
 [PantherTestCase] to run e2e scenarios, using a real-browser or HTTP client and a real web server
>
```

Поскольку мы хотим создать свой первый Unit тест, то выбираем TestCase.

Первый Unit тест с PHPUnit

Сообщение об успешном создании теста.

```
The name of the test class (e.g. BlogPostTest):
> ProductTest

created: tests/ProductTest.php

 Success!

Next: Open your new test class and start customizing it.
Find the documentation at https://symfony.com/doc/current/testing.html#unit-tests
```

Первый Unit тест с PHPUnit

После ввода команды, в папке tests/ создастся класс для нашего теста:

```php
namespace App\Tests;

use PHPUnit\Framework\TestCase;

class ProductTest extends TestCase
{
    public function testSomething(): void
    {
        $this->assertTrue(true);
    }
}
```

Давайте напишем тест на проверку создания новых записей в БД, которые относятся к нашему Entity - Product. И также рассмотрим пример тестирования контроллера.

Dataprovider

Dataprovider

Провайдеры данных или dataprovider представляют из себя отдельные функции в тестах, суть которых заключается в том, чтобы передавать прописанные данные в этих функциях - в функции, которые уже и делают тесты, при этом используя данные из провайдеров данных. Пример с добавлением двух чисел:

```php
class NumbersTest extends TestCase
{
    /**
     * @dataProvider numbersProvider
     */
    public function addNumbers($a, $b, $result)
    {
        $this->assertSame($result, $a + $b);
    }

    public function numbersProvider()
    {
        return [
            [4, 4, 8],
            [0, 1, 1],
            [1, 1, 2],
            [1, 2, 3]
        ];
    }
}
```

Тестовые двойники

Тестовые двойники

Мы разбирали пример с тестированием создания новой записи в таблицу product через Entity Product. Можно заметить, что при таких тестах мы создаём записи в рабочую среду, а это в свою очередь может создавать лишние и не нужные нам данные. В таких случаях можно воспользоваться “тестовыми двойниками” о которых мы сейчас и поговорим.

Представим сначала, что мы хотим сделать тестирование ProductRepository. Именно в этой ситуации нам понадобится создать “тестового двойника” данного класса, чтобы в рабочей среде не появлялось лишних данных.

Тестовые двойники

Пример использования Mock:

```php
use PHPUnit\Framework\TestCase;
use App\Entity\Product;
use Doctrine\Persistence\ObjectManager;
use Doctrine\Persistence\ObjectRepository;

class ProductTest extends TestCase
{
    public function testSomething(): void
    {

        $product = new Product();
        $product->setName('Book for programmers');
        $product->setPrice(30);

        $productRepository = $this->createMock(ObjectRepository::class);

        $productRepository->expects($this->any())
            ->method('find')
            ->willReturn($product);

        $objectManager = $this->createMock(ObjectManager::class);

        $objectManager->expects($this->any())
            ->method('getRepository')
            ->willReturn($productRepository);

        $foundProduct = $productRepository->find(1); // Нам будет возвращать объект в переменной $product

        fwrite(STDERR, print_r($foundProduct, TRUE)); // таким образом можно делать дебагинг

        $this->assertEquals($foundProduct->getPrice(), 30); // Проверяем равняется ли цена 30

    }
}
```

Тестовые двойники

Ещё один более простой пример уже со Stub (заглушкой):

```php
class ProductTest extends TestCase
{
    public function testSomething(): void
    {

        $stub = $this->getMockBuilder(Product::class)
                     ->getMock();

        $stub->setName('Testing name in PHPUnit');

        $stub->method('getName')
            ->willReturn('Testing name in PHPUnit');

        $this->assertSame('Testing name in PHPUnit', $stub->getName());

    }
}
```

Первый Functional тест

Первый Functional тест

Что же такое functional тесты в Symfony? Это тесты, которые могут иметь в себе несколько функций которые взаимодействуют между собой и в итоге выполняют тест. Пример:

```php
/**
 * @var \Doctrine\ORM\EntityManager
 */
private $entityManager;

protected function setUp(): void
{
    $kernel = self::bootKernel();

    $this->entityManager = $kernel->getContainer()
        ->get('doctrine')
        ->getManager();
}

public function testSearchByPrice()
{
    $product = $this->entityManager
        ->getRepository(Product::class)
        ->findOneBy(['price' => 30])
    ;

    $this->assertSame(30, $product->getPrice());
}

protected function tearDown(): void
{
    parent::tearDown();

    $this->entityManager->close();
    $this->entityManager = null;
}
```

Разделяем test, prod, dev окружения

Разделяем test, prod, dev окружения

Можем вспомнить момент, как мы переключились через файл .env на prod окружение для того, чтобы можно было посмотреть на страницы ошибок. В Symfony приложении уже по умолчанию есть созданные варианты конфигурации для трёх окружений, это test, prod и dev.

Для этой конфигурации существуют отдельные три папки в папке config/packages, которые сами уже и называются как это окружение. Давайте рассмотрим какие файлы они имеют и что вообще можно настраивать.

Фикстуры для тестов

Фикстуры для тестов

Когда мы разделили своё окружение на разные типы, мы можем создать уже фикстуры, которые будут использоваться только с окружением тестирования. Сделать это можно с помощью консоли с maker-bundle:

```
php bin/console --env=test make:fixture
```

Информационный видеосервис для разработчиков программного обеспечения

TestProvider Pearson CyberBionic systematics Microsoft Partner Windows Azure EBA

Проверка знаний

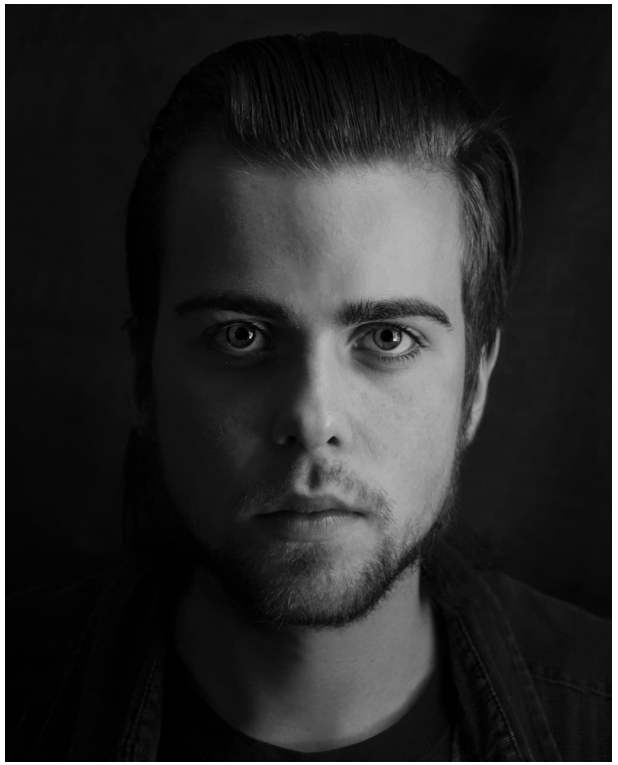
TestProvider.com

TestProvider – это online сервис проверки знаний по информационным технологиям. С его помощью Вы можете оценить Ваш уровень и выявить слабые места. Он будет полезен как в процессе изучения технологии, так и для общей оценки знаний IT специалиста.

Проверьте как Вы усвоили данный материал на TestProvider.com

Успешное прохождение финального тестирования позволит Вам получить соответствующий Сертификат.

Спасибо за внимание! До новых встреч!



Демьян Костельный
Middle PHP Developer