Лабораторная работа №4
«Модульное тестирование»

Цель работы

Познакомиться с механизмами модульного тестирования веб-приложений, построенных на гексагональной архитектуре.

Задание для выполнения

Установите и настройте PHPUnit.

Напишите модульные тесты для сценария транзакции из ЛР №1. Постарайтесь добиться 100% покрытия кода тестами.

При написании постарайтесь учитывать, что в дальнейшем части этого кода вам могут пригодиться при тестировании доменной модели (ЛР №5) и сервисов приложения (ЛР №6).

Теоретические сведения

Общие сведения о тестировании кода с использованием PHPUnit на русском языке ищите в официальной документации [1]. Мы же рассмотрим специфику тестирования приложений с гексагональной архитектурой.

Тестирование Варианта Использования (Use Case)

Michael Feathers ввёл определение: легаси код – это код без тестов. Мы же не хотим, чтобы наш код стал легаси, только родившись?

Возьмем для примера следующий Вариант Использования – «оценка идеи» *(продолжаем рассматривать пример из ЛР №3)*.

Для того, чтобы покрыть юнит-тестами данный объект Варианта Использования, вы решаете начать с простейшей части: что происходит, если репозиторий недоступен (отсутствует)? Как мы можем генерировать подобное поведение? Нужно ли останавливать Redis сервер на время юнит-тестирования? Нет. Нам не нужен объект, который имеет подобное поведение. Всего лишь используем фиктивный объект – заглушку *(mock-object)*, как в Листинге 1, который имитирует реальный:

Листинг 1 – Заглушка для тестирования оценки идеи

```
use PHPUnit\Framework\TestCase;

class RateIdeaUseCaseTest extends TestCase
{
    /**
     * @test
     */
```

```
    public function whenRepositoryNotAvailableAnExceptionShouldBeThrown()
    {
        $this->setExpectedException('NotAvailableRepositoryException');
        $ideaRepository = new NotAvailableRepository();
        $useCase = new RateIdeaUseCase($ideaRepository);
        $useCase->execute(
            new RateIdeaRequest(1, 5)
        );
    }
}

class NotAvailableRepository implements IdeaRepository
{
    public function find($id)
    {
        throw NotAvailableException();
    }

    public function update(Idea $idea)
    {
        throw NotAvailableException();
    }
}
```

Отлично. `NotAvailableRepository` имеет нужное нам поведение и можем использовать его с `RateIdeaUseCase`,поскольку он реализует интерфейс `IdeaRepository`.

Следующий случай для тестирования – это что происходит, если идеи нет в репозитории. Листинг 2 показывает код:

Листинг 2 – Тестирование отсутствия идеи в репозитории

```
class RateIdeaUseCaseTest extends TestCase
{
    // ...

    /**
     * @test
     */
    public function whenIdeaDoesNotExistAnExceptionShouldBeThrown()
    {
        $this->setExpectedException('IdeaDoesNotExistException');
        $ideaRepository = new EmptyIdeaRepository();
        $useCase = new RateIdeaUseCase($ideaRepository);
        $useCase->execute(
            new RateIdeaRequest(1, 5)
        );
    }
}

class EmptyIdeaRepository implements IdeaRepository
{
    public function find($id)
    {
        return null;
    }

    public function update(Idea $idea)
    {

    }
}
```

Здесь мы используем ту же самую стратегию, только с `EmptyIdeaRepository`. Он также реализует тот же самый интерфейс но имплементация всегда возвращает null независимо от того, какой идентификатор получает метод find.

Почему мы тестируем эти случаи? Вспомните слова Kent Beck: *Тестируйте всё, что потенциально может сломаться*.

Давайте продолжим с оставшимися фичами. Нам нужно проверить специальный кейс, в котором мы можем прочитать, но не можем писать в репозиторий. Решение представлено в Листинге 3:

Листинг 3 – Тестирование недоступного для записи репозитория

```
class RateIdeaUseCaseTest extends TestCase
{
    // ...

    /**
     * @test
     */
    public function whenUpdatingInReadOnlyAnIdeaAnExceptionShouldBeThrown()
    {
        $this->setExpectedException('NotAvailableRepositoryException');
        $ideaRepository = new WriteNotAvailableRepository();
        $useCase = new RateIdeaUseCase($ideaRepository);
        $response = $useCase->execute(
            new RateIdeaRequest(1, 5)
        );
    }
}

class WriteNotAvailableRepository implements IdeaRepository
{
    public function find($id)
    {
        $idea = new Idea();
        $idea->setId(1);
        $idea->setTitle('Subscribe to php[architect]');
        $idea->setDescription('Just buy it!');
        $idea->setRating(5);
        $idea->setVotes(10);
        $idea->setAuthor('hi@carlos.io');

        return $idea;
    }

    public function update(Idea $idea)
    {
        throw NotAvailableException();
    }
}
```

Ок, теперь осталась ключевая часть всей фичи. У нас есть разные пути тестирования: мы можем написать свой фиктивный объект или использовать соответствующий фреймворк, как Mockery или Prophecy. Давайте используем первое. Другим интересным упражнением было бы написать данный тест и предыдущие с использованием одного из фреймворков.

Листинг 4 – Тестирование основного метода

```
class RateIdeaUseCaseTest extends TestCase
```

```
{
    // ...

    /**
     * @test
     */
    public function whenRatingAnIdeaNewRatingShouldBeAddedAnIdeaUpdated()
    {
        $ideaRepository = new OneIdeaRepository();
        $useCase = new RateIdeaUseCase($ideaRepository);
        $response = $useCase->execute(
            new RateIdeaRequest(1, 5)
        );

        $this->assertSame(5, $response->idea->getRating());
        $this->assertTrue($ideaRepository->updateCalled);
    }
}

class OneIdeaRepository implements IdeaRepository
{
    public $updateCalled = false;

    public function find($id)
    {
        $idea = new Idea();
        $idea->setId(1);
        $idea->setTitle('Subscribe to php[architect]');
        $idea->setDescription('Just buy it!');
        $idea->setRating(5);
        $idea->setVotes(10);
        $idea->setAuthor('hi@carlos.io');

        return $idea;
    }

    public function update(Idea $idea)
    {
        $this->updateCalled = true;
    }
}
```

Поздравляю! 100% покрытие нашего Варианта Использования. Быть может, в следующий раз мы могли бы использовать TDD, и тесты буду идти впереди кода. В любом случае, тестирование этой фичи было очень простым, благодаря декомпозиции, на которой основана архитектура приложения.

Возможно, вам интересно, что это:

```
$this->updateCalled = true;
```

Нам нужен способ гарантировать, что метод update был вызван в процессе исполнения Варианта Использования. Это как раз и делается. Данный объект называется *шпион (spy)*, двоюродный брат заглушки (mock).

Когда использовать заглушки? В общем случае, используем их, когда **пересекаем границы**. В данном случае мы используем их, поскольку пересекаем границу от домена к слою хранения.

Тестирование инфраструктуры

Но что насчет инфраструктуры?

Если вы хотите достичь 100% покрытия тестами целого приложения, вам также придется тестировать и инфраструктуру. Перед тем, как этим заняться, нужно понимать, что данные тесты будут намного более связаны с вашей реализацией, чем тесты бизнес-логики. Это значит, что вероятность сломаться в процессе изменения деталей реализации намного выше.

Так что, если мы все еще хотим продолжать, нам надо сделать несколько изменений. Нужно ещё больше декомпозиции. Посмотрим на листинг:

Листинг 5 – Тестирование контроллера

```
class IdeaController extends Zend_Controller_Action
{
    public function rateAction()
    {
        $ideaId = $this->request->getParam('id');
        $rating = $this->request->getParam('rating');

        $useCase = new RateIdeaUseCase(
            new RedisIdeaRepository(
                new \Predis\Client()
            )
        );

        $response = $useCase->execute(
            new RateIdeaRequest($ideaId, $rating)
        );

        $this->redirect('/idea/'.$response->idea->getId());
    }
}

class RedisIdeaRepository implements IdeaRepository
{
    private $client;

    public function __construct($client)
    {
        $this->client = $client;
    }

    // ...
    public function find($id)
    {
        $idea = $this->client->get($this->getKey($id));
        if (!$idea) {
            return null;
        }

        return $idea;
    }
}
```

Для достижения 100% тестирования `RedisIdeaRepository` нам нужно иметь возможность передать `Predis\Client` как параметр репозиторию, без TypeHinting, благодаря чему мы смогли бы передать фиктивный объект, который позволит покрыть все необходимые кейсы.

Это заставляет нас изменить контроллер так, чтобы он строил соединение с Redis, передавал его в репозиторий и отдавал результаты в Вариант Использования [2].

Список использованных источников

1. Бергман, С. PHPUnit Manual [Электронный ресурс] – Режим доступа: https://phpunit.readthedocs.io/_/downloads/ru/latest/pdf/. – Дата доступа: 21.01.2021.

2. Buenosvinos, C. Hexagonal Architecture with PHP // php[architect]. – July, 2014. – *Перевод с англ. Кочурко, П.А.*