Министерство образования Республики Беларусь

Учреждение образования

«Брестский государственный технический университет»

Кафедра ИИТ

Лабораторная работа №4

По дисциплине «ПИС»

за 5-й семестр

Выполнил:
студент 2 курса
группы ПО-3 (1)
Афанасьев В.В.

Проверил:
Лаврущик А.И.

Брест, 2021

Цель работы: познакомиться с механизмами модульного тестирования веб приложений, построенных на гексагональной архитектуре.

«Гексагональная архитектура»

Вариант: 2

Задание:

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).

Предметная область: продажа билетов на мероприятия

Ход работы:

Исходный код:

Было добавлено несколько «mock» классов по взаимодействию с БД.

ClientRegistrationServiceTest

```
<?php
use PHPUnit\Framework\TestCase;

class ClientRegistrationServiceTest extends TestCase
{
    private ClientRegistrationService $service;
    private MockTicketRepository $ticketRepository;
    private MockClientRepository $clientRepository;

    protected function setUp(): void {
        $this->ticketRepository = new MockTicketRepository();
        $this->clientRepository = new MockClientRepository();
        $this->service = new ClientRegistrationService($this->ticketRepository, $this-
>clientRepository);
    }

    public function test_getAllEventsClients_Should_ReturnExceptedValue()
    {
        $first = new Ticket();
        $first->setId(1);
        $first->setEventId(1);
        $first->setClientId(1);
        $first->setCost(100);

        $second = new Ticket();
        $second->setId(2);
        $second->setEventId(2);
        $second->setClientId(2);
        $second->setCost(200);

        $excepted = [$first, $second];

        $result = $this->service->getAllTickets();

        $this->assertEquals($excepted, $result);
    }

    public function test_registerClient_Should_CreateValue()
    {
        $excepted = new Ticket();
        $excepted->setId(3);
        $excepted->setEventId(3);
        $excepted->setClientId(3);
```

```
        $this->service->registerClient($excepted);

        $this->assertEquals(true, in_array($excepted, $this->ticketRepository->store));
    }

    public function test_getAllClients_Should_ReturnExceptedValue()
    {
        $first = new Client();
        $first->setId(1);
        $first->setFirstName('test_nord1');
        $first->setLastName('test_nord1');

        $second = new Client();
        $second->setId(2);
        $second->setFirstName('test_nord2');
        $second->setLastName('test_nord2');

        $excepted = [$first, $second];

        $result = $this->service->getAllClients();

        $this->assertEquals($excepted, $result);
    }

    public function test_recordClient_Should_CreateValue()
    {
        $excepted = new Client();
        $excepted->setId(4);
        $excepted->setFirstName('nord4');
        $excepted->setLastName('nord4');

        $this->service->recordClient($excepted);

        $this->assertEquals(true, in_array($excepted, $this->clientRepository->store));
    }
}
```

EventServiceTest

```
<?php
use PHPUnit\Framework\TestCase;

class EventServiceTest extends TestCase
{
    private EventService $service;
    private MockEventRepository $eventRepository;

    protected function setUp(): void {
        $this->eventRepository = new MockEventRepository();
        $this->service = new EventService($this->eventRepository);
    }

    public function test_getAllClients_Should_ReturnExceptedValue()
    {
        $first = new Event();
        $first->setId(1);
        $first->setEventName('test_nord1');
        $first->setEventTime('test_nord1');

        $second = new Event();
        $second->setId(2);
        $second->setEventName('test_nord2');
        $second->setEventTime('test_nord2');

        $excepted = [$first, $second];

        $result = $this->service->getAllEvents();

        $this->assertEquals($excepted, $result);
    }

    public function test_recordReservist_Should_CreateValue()
    {
```

```
        $excepted = new Event();
        $excepted->setId(3);
        $excepted->setEventName('text3');
        $excepted->setEventTime('text3');

        $this->service->recordEvent($excepted);

        $this->assertEquals(true, in_array($excepted, $this->eventRepository->store));
    }
}
```

Mock repositories:

MockClientRepository

```
<?php
include("Interfaces/IClientRepository.php");

class MockClientRepository implements IClientRepository
{
    public array $store;

    public function __construct()
    {
        $first = (object) [
            'Id' => 1,
            'FirstName' => 'test_nord1',
            'LastName' => 'test_nord1'];

        $second = (object) [
            'Id' => 2,
            'FirstName' => 'test_nord2',
            'LastName' => 'test_nord2'];

        $this->store = [$first, $second];
    }

    public function getAll(): array
    {
        return $this->store;
    }

    public function create(Client $entity)
    {
        array_push($this->store, $entity);
    }

    public function getByIdOrNull($id): Client
    {
        // TODO: Implement getByIdOrNull() method.
    }

    public function update(Client $entity)
    {
        // TODO: Implement update() method.
    }

    public function delete($id)
    {
        // TODO: Implement delete() method.
    }
}
```

MockEventRepository

```
<?php
include("Interfaces/IEventRepository.php");

class MockEventRepository implements IEventRepository
{
    public array $store;

    public function __construct()
    {
```

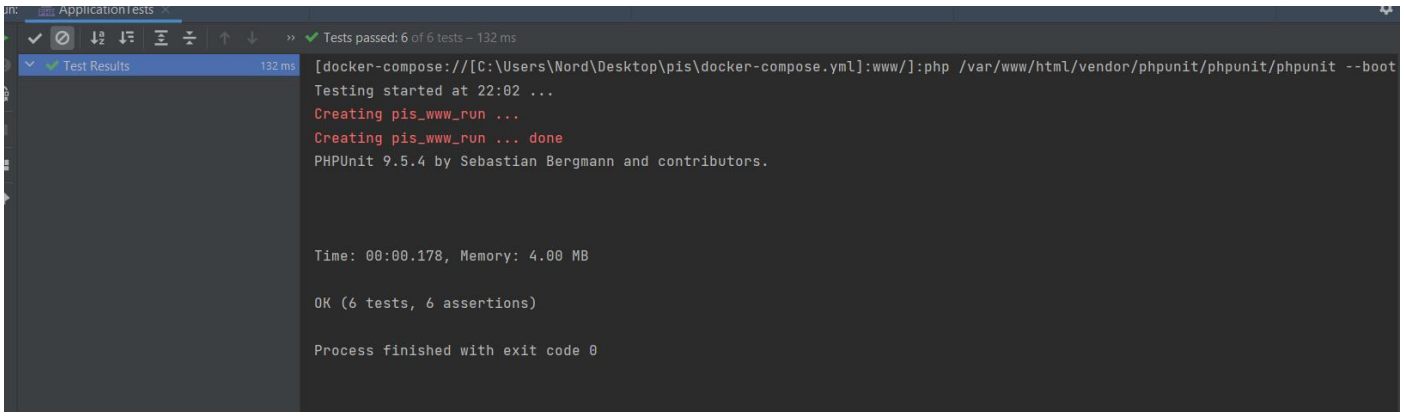
```
        $first = (object) [
            'Id' => 1,
            'EventName' => 'test_nord1',
            'EventTime' => 'test_nord1'];

        $second = (object) [
            'Id' => 2,
            'EventName' => 'test_nord2',
            'EventTime' => 'test_nord2'];

        $this->store = [$first, $second];
    }

    public function getAll(): array
    {
        return $this->store;
    }

    public function create(Event $entity)
    {
        array_push($this->store, $entity);
    }

    public function getByIdOrNull($id): Event
    {
        // TODO: Implement getByIdOrNull() method.
    }

    public function update(Event $entity)
    {
        // TODO: Implement update() method.
    }

    public function delete($id)
    {
        // TODO: Implement delete() method.
    }
}
```

MockTicketRepository

```
<?php
include("Interfaces/ITicketRepository.php");

class MockTicketRepository implements ITicketRepository
{
    public array $store;

    public function __construct()
    {
        $first = (object) [
            'Id' => 1,
            'ClientId' => 1,
            'EventId' => 1,
            'Cost' => 100];

        $second = (object) [
            'Id' => 2,
            'ClientId' => 2,
            'EventId' => 2,
            'Cost' => 200];

        $this->store = [$first, $second];
    }

    public function getAll(): array
    {
        return $this->store;
    }

    public function create(Ticket $entity)
    {
        array_push($this->store, $entity);
    }
```

```
    public function getByIdOrNull($id): Ticket
    {
        // TODO: Implement getByIdOrNull() method.
    }

    public function update(Ticket $entity)
    {
        // TODO: Implement update() method.
    }

    public function delete($id)
    {
        // TODO: Implement delete() method.
    }
}
```

Выполнение:



Выводы: в ходе выполнения лабораторной работы были ознакомлены с механизмами модульного тестирования веб приложений, построенных на гексагональной архитектуре.