Министерство образования Республики Беларусь

Учреждение образования

«Брестский государственный технический университет»

Кафедра ИИТ

Лабораторная работа №6

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 практической реализацией принципа инверсии зависимостей, принципа единственной ответственности и механизмом внедрения зависимостей.

Вариант: 2

Задание:

Реализуйте сервисы на уровне приложения, реализующие с доменными моделями те же действия, что и сценарий транзакции из ЛР №1. Не забудьте о принципе единственной ответственности. Хотя бы одно действие должно реализовываться доменным сервисом. Напишите модульные тесты для сервисов. Поскольку инфраструктурный слой еще не реализован – используйте заглушки вместо репозиториев (ЛР №4).

Предметная область: продажа билетов на мероприятия.

Ход работы:

Исходный код:

Interfaces:

IClientRegistrationService.php

```
<?php

interface IClientRegistrationService
{
    public function registerClient(Ticket $ticket);

    public function recordClient(Client $client);

    public function getAllTickets(): array;

    public function getAllClients(): array;
}
```

IEventService.php

```
<?php

interface IEventService
{
    public function recordEvent(Event $event);

    public function getAllEvents(): array;
}
```

Services:

ClientRegistrationService.php

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

class ClientRegistrationService implements IClientRegistrationService
{
    private ITicketRepository $ticketRepository;
    private IClientRepository $clientRepository;

    public function __construct(
        ITicketRepository $ticketRepository,
        IClientRepository $clientRepository)
    {
        $this->ticketRepository = $ticketRepository;
        $this->clientRepository = $clientRepository;
    }
```

```
    public function registerClient(Ticket $ticket)
    {
        $this->ticketRepository->create($ticket);
    }

    public function recordClient(Client $client)
    {
        $this->clientRepository->create($client);
    }

    public function getAllTickets(): array
    {
        $tickets = $this->ticketRepository->getAll();

        $result = array();

        foreach ($tickets as $value)
        {
            $ticket = new Ticket();
            $ticket->setId($value->Id);

            $ticket->setClientId($value->ClientId);
            $ticket->setEventId($value->EventId);
            $ticket->setCost($value->Cost);

            array_push($result, $ticket);
        }

        return $result;
    }

    public function getAllClients(): array
    {
        $clients = $this->clientRepository->getAll();

        $result = array();

        foreach ($clients as $value)
        {
            $client = new Client();
            $client->setId($value->Id);

            $client->setFirstName($value->FirstName);
            $client->setLastName($value->LastName);

            array_push($result, $client);
        }

        return $result;
    }
}
```

EventService.php

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

class EventService implements IEventService
{
    private IEventRepository $eventRepository;

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

    public function recordEvent(Event $event)
    {
        $this->eventRepository->create($event);
    }

    public function getAllEvents(): array
    {
        $events = $this->eventRepository->getAll();
```

```
        $result = array();

        foreach ($events as $value)
        {
            $event = new Event();
            $event->setId($value->Id);

            $event->setEventName($value->EventName);
            $event->setEventTime($value->EventTime);

            array_push($result, $event);
        }

        return $result;
    }
}
```

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

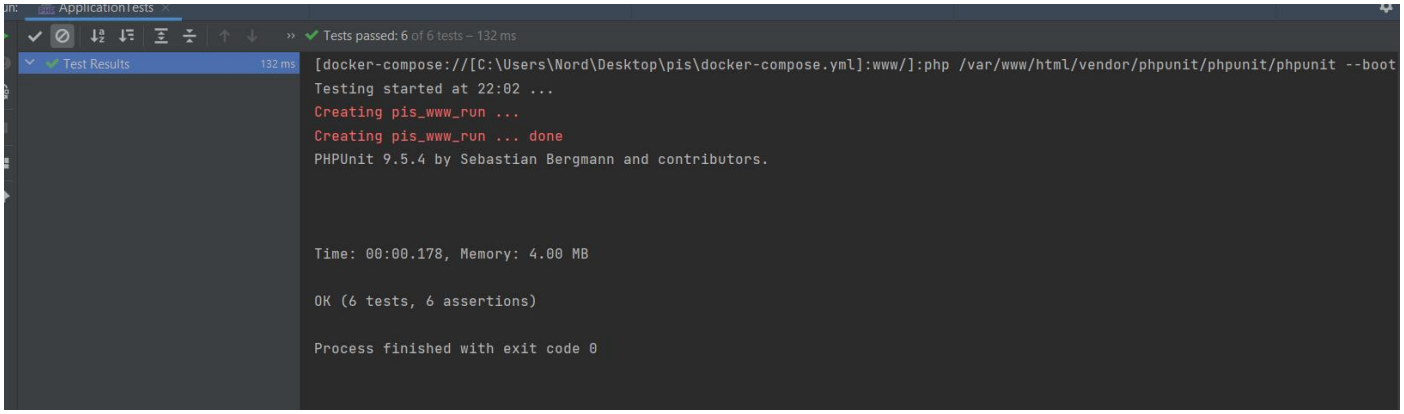
```
<?php
include("Interfaces/IEventRepository.php");

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.
    }
}
```



Выводы: в ходе выполнения лабораторной работы были ознакомлены с практической реализацией принципа инверсии зависимостей, принципа единственной ответственности и механизмом внедрения зависимостей.