Министерство образования Республики Беларусь

Учреждение образования

«Брестский государственный технический университет»

Кафедра ИИТ

Лабораторная работа №4

По дисциплине: «ПИС»

Тема: “Модульное тестирование”

Выполнил:

Студент 3 курса

Группы ПО-7

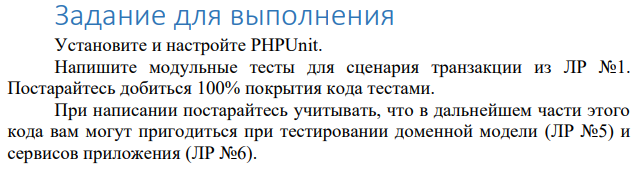
Комиссаров А.Е.

Проверил:

Михняев А.Л.

2023

**Цель работы:** познакомиться с механизмами модульного тестирования вебприложений, построенных на гексагональной архитектуре.

****

**Ход работы.**

**Код программы:**

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.  
 }  
}

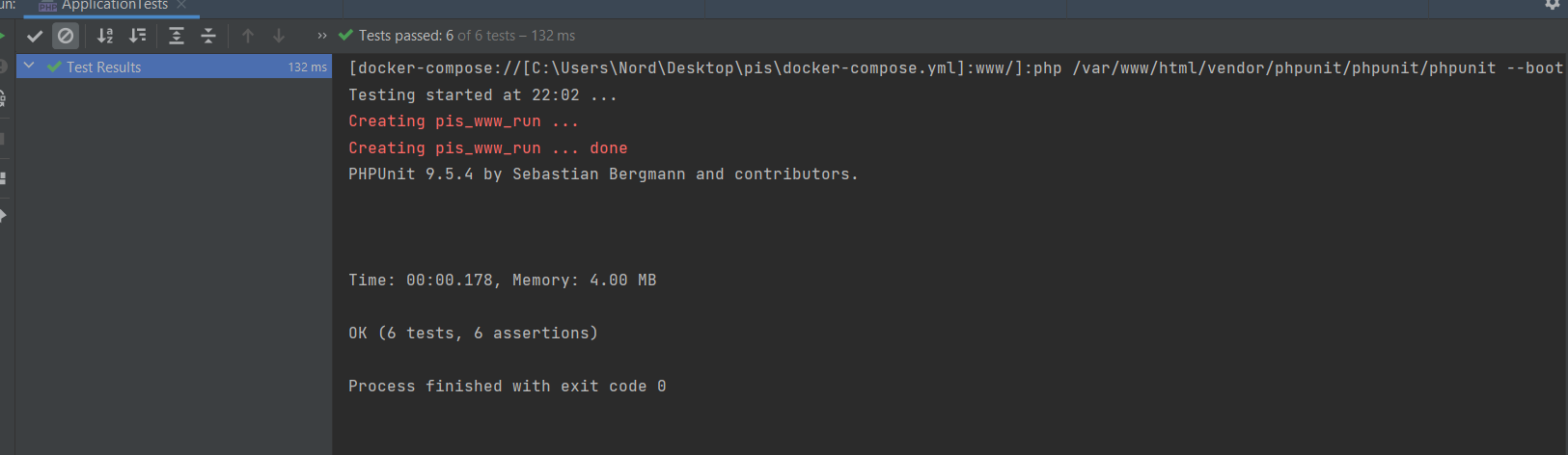
MockEventRepository

<?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.  
 }  
}

**Результат работы программы:**

****

**Вывод:** я познакомился с механизмами модульного тестирования веб-приложений, построенных на гексагональной архитектуре.