Разработка многостраничного сайта на PHP

ТЕСТИРОВАНИЕ ПРИЛОЖЕНИЙ
ЗАНЯТИЕ № 10 - ЛЕКЦИЯ/ ПРАКТИКА

Тема занятия – Тестирование приложений

Цель –

Изучить способы тестирования приложения и понять, зачем это надо применять на практике.

Актуализация

На прошлых занятиях мы уже познакомились с:

1) Архитектурой приложения и шаблонами проектирования;
2) Расширенными методами в PHP;
3) С работой с контейнерами и системой сборки Docker;

Теперь мы приступаем к заключительной части нашего курса «Разработка многостраничного сайта на PHP» –

Тестирование приложений

Введение

PHP - это мощный инструмент для создания веб-приложений. Однако, как и любое программное обеспечение, PHP-приложения могут содержать ошибки и несоответствия, которые могут привести к проблемам с производительностью и безопасностью. Поэтому тестирование приложения - это критически важный этап разработки, который позволяет обнаруживать и исправлять ошибки до того, как они станут проблемами в продакшене.

Введение

Тестирование является важной частью разработки приложений, так как позволяет выявлять ошибки и улучшать качество продукта.

Сегодня мы рассмотрим несколько инструментов тестирования в PHP –

1) PHPUnit,
2) моки и стабы,
3) Phing,
4) JMeter.

PHPUnit

PHPUnit - это популярный фреймворк для юнит-тестирования в PHP. Он предоставляет различные методы и функции для тестирования кода.

PHPUnit позволяет автоматически запускать тесты при каждом изменении кода и быстро выявлять ошибки.

Пример использования PHPUnit

Допустим, у нас есть класс Calculator, который содержит метод для вычисления суммы двух чисел:

```php
class Calculator {
    public function sum($a, $b) {
        return $a + $b;
    }
}
```

Пример использования PHPUnit

Мы можем написать модульный тест для этого метода с помощью PHPUnit. Для этого создадим новый файл, например, CalculatorTest.php, в котором определим тест-кейс:

```php
use PHPUnit\Framework\TestCase;

class CalculatorTest extends TestCase {
    public function testSum() {
        $calculator = new Calculator();
        $this->assertEquals(4, $calculator->sum(2, 2));
        $this->assertEquals(10, $calculator->sum(5, 5));
        $this->assertEquals(0, $calculator->sum(-2, 2));
    }
}
```

Пример использования PHPUnit

```php
use PHPUnit\Framework\TestCase;

class CalculatorTest extends TestCase {
    public function testSum() {
        $calculator = new Calculator();
        $this->assertEquals(4, $calculator->sum(2, 2));
        $this->assertEquals(10, $calculator->sum(5, 5));
        $this->assertEquals(0, $calculator->sum(-2, 2));
    }
}
```

В этом тест-кейс мы создаем экземпляр класса Calculator и вызываем его метод sum с разными параметрами. Затем мы проверяем, что результаты вызовов метода sum соответствуют ожидаемым значениям.

Моки и стабы

Моки и стабы - это инструменты, которые используются для создания имитации объектов и методов. Они позволяют тестировать код без необходимости использования реальных объектов и среды. Моки используются для создания объектов с заданным поведением, тогда как стабы используются для создания объектов с фиксированным состоянием. Это позволяет тестировать код, который зависит от сложных объектов или внешних сервисов.

Пример использования моков и стабов

Предположим, у нас есть класс User и класс Mailer. Класс User содержит метод sendEmail, который использует класс Mailer для отправки электронной почты.

```php
use PHPUnit\Framework\TestCase;

class UserTest extends TestCase {
    public function testSendEmail() {
        $mailer = $this->getMockBuilder(Mailer::class)
                       ->disableOriginalConstructor()
                       ->getMock();
        $mailer->expects($this->once())
               ->method('send')
               ->with('test@example.com', 'Hello, World!');
        $user = new User($mailer);
        $user->sendEmail('test@example.com', 'Hello, World!');
    }
}
```

Пример использования моков и стабов

Мы можем написать модульный тест для метода sendEmail с помощью моков и стабов. Для этого мы создадим новый файл, например, UserTest.php, и определим тест-кейс:

```php
use PHPUnit\Framework\TestCase;

class UserTest extends TestCase {
    public function testSendEmail() {
        $mailer = $this->getMockBuilder(Mailer::class)
                       ->disableOriginalConstructor()
                       ->getMock();
        $mailer->expects($this->once())
               ->method('send')
               ->with('test@example.com', 'Hello, World!');
        $user = new User($mailer);
        $user->sendEmail('test@example.com', 'Hello, World!');
    }
}
```

Пример использования моков и стабов

```php
use PHPUnit\Framework\TestCase;

class UserTest extends TestCase {
    public function testSendEmail() {
        $mailer = $this->getMockBuilder(Mailer::class)
                       ->disableOriginalConstructor()
                       ->getMock();
        $mailer->expects($this->once())
               ->method('send')
               ->with('test@example.com', 'Hello, World!');
        $user = new User($mailer);
        $user->sendEmail('test@example.com', 'Hello, World!');
    }
}
```

В этом тест-кейсе мы создаем мок объект класса **Mailer**, который не имеет конструктора и не вызывает реальный метод send. Затем мы ожидаем, что метод send будет вызван один раз с определенными параметрами. Далее мы создаем объект класса **User** с мок объектом **Mailer** и вызываем метод **sendEmail**, который должен вызвать метод send у мока объекта **Mailer**.

Phing

Phing - это инструмент для автоматизации процесса сборки и развертывания приложений. С его помощью можно управлять сборкой проекта, выполнять задачи, такие как очистка кеша, компиляция кода и запуск тестов. Phing также поддерживает непрерывную доставку, которая позволяет автоматически разворачивать новые версии приложения на серверах.

Пример использования непрерывной доставки (continuous delivery) с помощью Phing:

Непрерывная доставка - это методика разработки программного обеспечения, при которой изменения в коде автоматически тестируются, собираются и доставляются в производственную среду.

Мы можем использовать Phing для автоматизации процесса сборки, тестирования и доставки нашего приложения.
Для этого мы можем создать файл build.xml, который будет содержать инструкции для Phing о том, как собирать и доставлять наше приложение.

Пример

```xml
<?xml version="1.0" encoding="UTF-8"?>
<project name="MyApp" default="deploy">
    <target name="clean">
        <delete dir="build"/>
    </target>

    <target name="build" depends="clean">
        <!-- Сборка приложения -->
    </target>

    <target name="test" depends="build">
        <!-- Запуск модульных тестов -->
    </target>

    <target name="deploy" depends="test">
        <!-- Доставка приложения в производственную среду -->
    </target>
</project>
```

JMeter

JMeter - это инструмент для тестирования производительности и нагрузки веб-приложений. Он позволяет создавать тестовые сценарии, которые могут имитировать поведение реальных пользователей, например, нажатие на кнопки и заполнение форм.

JMeter также может генерировать отчеты о производительности приложения, которые помогут выявить узкие места и улучшить производительность.

JMeter

Представим, что у нас есть веб-приложение для онлайн-магазина, и мы хотим протестировать его производительность при большой нагрузке. Мы можем создать тестовый сценарий, который будет имитировать действия реальных пользователей.

JMeter

Допустим, у нас есть веб-приложение, которое должно обрабатывать большое количество запросов от пользователей. Мы можем использовать JMeter для проведения нагрузочного тестирования наше приложения. Для этого мы создадим новый тест-план в JMeter, определим группу потоков пользователей, которые будут отправлять запросы на наше приложение, и определим параметры тестирования, такие как количество потоков и количество запросов.

Пример использования JMeter

После того, как мы запустим тестирование, JMeter будет отправлять запросы на наше приложение и собирать статистику по производительности, такую как время ответа сервера, количество успешных и неуспешных запросов и т.д. Эту статистику мы можем анализировать, чтобы определить, как наше приложение работает при высокой нагрузке и какие изменения необходимо внести для улучшения производительности.

Пример использования JMeter

Например, мы можем запустить тестовый сценарий с 100 пользователями, которые будут выполнять действия в течение 10 минут. JMeter будет генерировать запросы к серверу с заданной частотой и записывать результаты. После завершения теста мы можем посмотреть отчеты о производительности приложения, которые покажут время отклика сервера, количество ошибок и другие параметры.

Практическая часть

В практической части мы рассмотрим применение упомянутых выше инструментов тестирования на PHP.

У нас будет 12 задач: начиная с легких и заканчивая сложными задачи: для начала начнем с самого простого, не будет указывать инструмент тестирования, а затем начнем усложнять.

Будьте внимательны! Постарайтесь не отвлекаться от написания задач на другие вопросы.

Задача № 1

Напишите функцию для проверки корректности введенного пароля.

Решение:

```php
<?php
function validatePassword($password) {
    // Проверяем, что пароль не пустой
    if (empty($password)) {
        return false;
    }

    // Проверяем, что длина пароля не менее 8 символов
    if (strlen($password) < 8) {
        return false;
    }

    // Проверяем, что пароль содержит хотя бы одну заглавную букву,
    //одну строчную букву и одну цифру
    if (!preg_match('/^(?=.*[a-z])(?=.*[A-Z])(?=.*\d).+$/', $password)) {
        return false;
    }

    // Все проверки пройдены, пароль корректный
    return true;
}
?>
```

Комментарии

- Функция **validatePassword** принимает в качестве аргумента пароль для проверки и возвращает **true**, если пароль корректный, и **false**, если нет.
- Проверка на пустоту пароля осуществляется с помощью функции **empty**.
- Проверка на длину пароля осуществляется с помощью функции **strlen**.
- Проверка на наличие хотя бы одной заглавной буквы, одной строчной буквы и одной цифры осуществляется с помощью регулярного выражения.
- Если все проверки пройдены, то функция

Задача № 2

Написать функцию для проверки корректности введенного email-адреса.

Решение

```php
function validateEmail($email) {
  // Проверяем, что email не пустой
  if (empty($email)) {
    return false;
  }

  // Проверяем, что email соответствует формату
  if (!filter_var($email, FILTER_VALIDATE_EMAIL)) {
    return false;
  }

  // Проверяем, что домен email-адреса имеет запись MX
  list($user, $domain) = explode('@', $email);
  if (!checkdnsrr($domain, 'MX')) {
    return false;
  }

  // Все проверки пройдены, email-адрес корректный
  return true;
}
```

Комментарии

- Функция **validateEmail** принимает в качестве аргумента email-адрес для проверки и возвращает **true**, если адрес корректный, и **false**, если нет.
- Проверка на пустоту адреса осуществляется с помощью функции **empty.**
- Проверка на соответствие формату email-адреса осуществляется с помощью функции **filter_var** с параметром **FILTER_VALIDATE_EMAIL.**
- Проверка на наличие записи MX для домена email-адреса осуществляется с помощью функции **checkdnsrr**.
- Если все проверки пройдены, то функция возвращает **true.**
- Если хотя бы одна проверка не пройдена, то функция возвращает

Задача № 3

Напишите код, который запрашивает у пользователя число и выводит его квадрат. При этом необходимо проверить, является ли введенное значение числом.

Решение

```php
<?php

// получаем значения от пользователя
$num1 = readline("Введите первое число: ");
$num2 = readline("Введите второе число: ");

// проверяем, что второе число не равно нулю
if ($num2 == 0) {
  // если второе число равно нулю, выводим ошибку и завершаем скрипт
  echo "Ошибка: деление на ноль невозможно";
  exit;
}

// вычисляем результат деления и выводим его
$div = $num1 / $num2;
echo "Результат деления " . $num1 . " на " . $num2 . " равен " . $div;

?>
```

Задача № 4

Напишите код, который запрашивает у пользователя имя и выводит приветственное сообщение с этим именем. При этом необходимо проверить, было ли введено имя.

Решение

```php
<?php

// получаем значение от пользователя
$name = readline("Введите ваше имя: ");

// проверяем, было ли введено имя
if (empty($name)) {
  // если имя не было введено, выводим ошибку и завершаем скрипт
  echo "Ошибка: вы не ввели имя";
  exit;
}

// выводим приветственное сообщение с именем пользователя
echo "Привет, " . $name . "! Добро пожаловать на наш сайт.";

?>
```

Задача № 5

Напишите код, который запрашивает у пользователя два числа и выводит результат их деления. При этом необходимо проверить, что второе число не равно нулю.

Решение

```php
<?php

// получаем значения от пользователя
$num1 = readline("Введите первое число: ");
$num2 = readline("Введите второе число: ");

// проверяем, что второе число не равно нулю
if ($num2 == 0) {
  // если второе число равно нулю, выводим ошибку и завершаем скрипт
  echo "Ошибка: деление на ноль невозможно";
  exit;
}

// вычисляем результат деления и выводим его
$div = $num1 / $num2;
echo "Результат деления " . $num1 . " на " . $num2 . " равен " . $div;

?>
```

Задача № 6

Проверить, что функция "strlen" возвращает длину строки в символах.

Решение

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

class StringLengthTest extends TestCase
{
    public function testStringLength()
    {
        $str = "Hello, world!";
        $length = strlen($str);
        $this->assertEquals(13, $length);
    }
}
?>
```

Комментарии

В этом тесте мы используем фреймворк **PHPUnit** для проверки того, что функция "**strlen**" правильно возвращает длину строки в символах. Мы создаем новый тестовый класс "**StringLengthTest**", который расширяет класс "**TestCase**" из **PHPUnit**, и определяем метод "**testStringLength**", который выполняет тестирование. Мы создаем строку "**Hello, world**!", вызываем функцию "**strlen**" для этой строки и сравниваем результат с ожидаемым значением 13 с помощью метода "**assertEquals**".

Задача № 7

Написать тесты для функции, которая принимает на вход два числа и возвращает их сумму.

Решение

```php
// Код функции, которую нужно протестировать
function sum($a, $b) {
  return $a + $b;
}

// Код теста
class SumTest extends PHPUnit_Framework_TestCase {
  public function testSum() {
    $this->assertEquals(5, sum(2, 3));
    $this->assertEquals(0, sum(0, 0));
    $this->assertEquals(-10, sum(-5, -5));
  }
}
```

Комментарии

В этом примере мы используем **PHPUnit** для написания тестов для функции **sum()**. Мы создаем класс **SumTest**, который наследуется от **PHPUnit_Framework_TestCase**, и определяем в нем тест **testSum()**. В этом тесте мы используем метод **assertEquals()**, который проверяет, что два значения равны. Мы вызываем функцию **sum()** с разными аргументами и проверяем, что результаты соответствуют ожидаемым.

Задача № 8

Написать тесты для функции, которая проверяет, является ли строка палиндромом.

Решение

```php
// Код функции, которую нужно протестировать
function isPalindrome($string) {
  $string = strtolower($string);
  $string = preg_replace('/[^a-z]/', '', $string);
  return $string == strrev($string);
}

// Код теста
class IsPalindromeTest extends PHPUnit_Framework_TestCase {
  public function testIsPalindrome() {
    $this->assertTrue(isPalindrome('racecar'));
    $this->assertTrue(isPalindrome('A man a plan a canal Panama'
    $this->assertFalse(isPalindrome('hello world'));
  }
}
```

Комментарии

В этом примере мы также используем **PHPUnit** для написания тестов для функции **isPalindrome**(). Мы создаем класс **IsPalindromeTest**, который наследуется от **PHPUnit_Framework_TestCase**, и определяем в нем тест **testIsPalindrome**(). В этом тесте мы используем методы **assertTrue**() и **assertFalse**(), которые проверяют, что выражение истинно или ложно соответственно. Мы вызываем функцию **isPalindrome**() с разными строками и проверяем, что результаты соответствуют ожидаемым.

Задача № 9

Написать функцию, которая получает данные от API и возвращает их в определенном формате. Необходимо протестировать функцию, используя моки для API.

Решение

```php
function get_data_from_api($url) {
  $data = file_get_contents($url);
  $data_array = json_decode($data, true);
  $formatted_data = [];

  foreach($data_array as $item) {
    $formatted_data[] = [
      'id' => $item['id'],
      'name' => $item['name'],
      'email' => $item['email'],
    ];
  }

  return $formatted_data;
}
```

Решение

- Код теста с использованием мока для API:

```
public function test_get_data_from_api() {
    $mock_api = $this->getMockBuilder('Api')
                     ->setMethods(['get_data'])
                     ->getMock();
    $mock_api->expects($this->once())
             ->method('get_data')
             ->willReturn('{"users":[{"id":1,"name":"John","email":"john@example.com"},
                                     {"id":2,"name":"Jane","email":"jane@example.com"}]}');

    $result = get_data_from_api($mock_api->get_data());
    $expected_result = [
        ['id' => 1, 'name' => 'John', 'email' => 'john@example.com'],
        ['id' => 2, 'name' => 'Jane', 'email' => 'jane@example.com']
    ];

    $this->assertEquals($expected_result, $result);
    }
```

Комментарии

Мы создаем мок-объект для класса Api и используем его для замены реального вызова API. Мы проверяем, что функция get_data_from_api правильно форматирует данные из API в ожидаемый формат.

Задача № 10

Написать функцию, которая получает данные от базы данных и возвращает их в определенном формате. Необходимо протестировать функцию, используя стаб для базы данных.

Решение

```php
function getUsers() {
  $db = new Database();
  $users = $db->query("SELECT * FROM users");

  $result = array();
  foreach ($users as $user) {
    $result[] = array(
      'id' => $user['id'],
      'name' => $user['name'],
      'email' => $user['email']
    );
  }

  return $result;
}
```

Решение

В этом примере функция **getUsers()** использует объект базы данных **$db** для выполнения запроса **SELECT * FROM users**. Результаты запроса затем преобразуются в массив, содержащий только **id, name и email** каждого пользователя.

Чтобы протестировать эту функцию с использованием **стаба** базы данных, мы можем создать фиктивный класс **DatabaseStub**, который будет имитировать объект базы данных и возвращать фиктивные данные.

Решение

```php
function testGetUsers() {
  $dbStub = new DatabaseStub();

  $expectedResult = array(
    array('id' => 1, 'name' => 'John', 'email' => 'john@example.com'),
    array('id' => 2, 'name' => 'Jane', 'email' => 'jane@example.com')
  );

  $result = getUsers($dbStub);

  // Проверяем, что результат соответствует ожидаемому
  assert($result == $expectedResult);
}

testGetUsers();
```

Комментарии

В этом примере функция **testGetUsers()** создает объект **стаба** базы данных **$dbStub**, который будет использоваться при вызове функции **getUsers()**. Затем функция вызывает **getUsers() с $dbStub** в качестве параметра и проверяет, что результат соответствует ожидаемому.

Задача № 11

Написать тесты для функции, которая отправляет HTTP-запросы. Проверить, что запрос был успешно выполнен и вернул корректные данные.

Решение

```
class HttpTest extends PHPUnit_Framework_TestCase {
  public function testHttpGet() {
    $url = 'https://example.com';
    $response = http_get($url);
    $this->assertEquals(200, $response->getStatusCode());
    $this->assertContains('Example Domain', $response->getBody());
  }
}
```

Комментарии

Данная задача проверяет работу функции **http_get()**, которая отправляет HTTP-запрос по указанному URL и возвращает ответ в виде объекта. Тест проверяет, что ответ имеет статус 200 (то есть запрос был успешно выполнен) и содержит текст "Example Domain" в теле ответа.

Для успешного прохождения теста необходимо, чтобы функция **http_get()** корректно выполняла запросы и возвращала ожидаемые данные в нужном формате. Также необходимо убедиться, что URL, указанный в тесте, действительно возвращает ожидаемый ответ.

Тестирование HTTP-запросов является важной частью тестирования приложений, особенно для тестирования взаимодействия с внешними сервисами или API. При написании тестов необходимо учитывать возможные ошибки и исключения, которые могут возникнуть в процессе отправки запросов и обработки ответов, и предусмотреть соответствующие проверки.

Задача № 12

Написать функцию, которая будет записывать текстовую строку в файл, и написать на неё тест.

Условие:

Функция должна принимать два аргумента: путь к файлу и текстовую строку. Если файл не существует, функция должна создать его. Если файл уже существует, функция должна дописать строку в конец файла. Функция должна возвращать true, если запись прошла успешно, и false, если произошла ошибка.

Решение

```php
function writeToFile($filename, $content) {
    $result = file_put_contents($filename, $content, FILE_APPEND | LOCK_EX);
    return ($result !== false);
}
```

Комментарии

Функция **file_put_contents**() используется для записи данных в файл. Флаг **FILE_APPEND** указывает на дополнение данных в файле вместо перезаписи, а флаг **LOCK_EX** предотвращает возможность параллельной записи в файл другими процессами. Если функция **file_put_contents**() возвращает false, то функция **writeToFile**() также возвращает false, иначе возвращает true.

Решение

Тестирование:

```
class FileTest extends PHPUnit_Framework_TestCase {
  public function testWriteToFile() {
    $filename = 'test.txt';
    $content = 'Hello, World!';
    $result = writeToFile($filename, $content);
    $this->assertTrue($result);
    $this->assertFileExists($filename);
    $this->assertStringEqualsFile($filename, $content);
    unlink($filename);
  }
}
```

Комментарии

В тесте создается файл **test.txt**, вызывается функция **writeToFile()** для записи в файл строки "**Hello, World!**", после чего проверяется, что файл существует и содержит ожидаемую строку. В конце файла удаляется созданный файл, чтобы не оставлять мусор на диске. Тест проверяет, что функция **writeToFile()** корректно записывает данные в файл и возвращает true в случае успеха.

Заключение

В заключение, тестирование является важной частью разработки приложений.

PHPUnit, моки и стабы, Phing, и JMeter - это инструменты, которые помогают разработчикам тестировать и улучшать качество своих приложений. При использовании этих инструментов необходимо учитывать особенности своего проекта и задачи, которые необходимо выполнить.

Рефлексия

Вот и подошел к концу на курс “Разработка многостраничного сайта на PHP”. Надеюсь, этот курс вам понравился. Для многих этот курс станет отправной точкой в программировании. Не останавливайтесь на достигнутом, достигайте всех поставленных целей! Желаю вам успехов и удачи в столь сложном, но при этом увлекательном пути.

1) Вам понравился этот курс?
2) Чтобы вы хотели добавить или изменить в нем?
3) Что вам понравилось больше всего? Что не понравилось? Почему?

Спасибо за внимание