Course: Automating PHP Tests with PHPUnit

Introduction: Unit Testing and Automation

Unit tests in PHP are used to verify that each function or method works correctly in isolation.

Test automation means that these tests are executed automatically, often through tools like PHPUnit. This ensures code quality, facilitates maintenance, and quickly detects regressions.

Why Use PHPUnit?

- PHPUnit is the most popular testing framework in PHP.
- It’s easy to install with Composer.
- It provides a clear syntax for writing tests (classes, assertions).
- It includes tools to automatically run tests and generate reports.
- Compatible with native PHP projects as well as frameworks (Symfony, Laravel, etc.).

---

1. Installing PHPUnit

The recommended method is via Composer.

1. Install PHPUnit as a development dependency:

   ```
   composer require --dev phpunit/phpunit
   ```

2. Verify the installation:

   ```
   ./vendor/bin/phpunit --version
   ```

3. Create a shortcut in the **composer.json** file (optional):

   ```json
   {
       "scripts": {
           "test": "./vendor/bin/phpunit tests"
       }
   }
   ```

---

2. Basic PHPUnit Test Structure

A PHPUnit test is a PHP class that extends `\PHPUnit\Framework\TestCase`. Each public method of this class that starts with `test` is considered a test.

Simple example in `tests/MathTest.php`:

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

class MathTest extends TestCase
{
    public function testAdd()
    {
        $result = 2 + 3;
        $this->assertEquals(5, $result);
    }
}
?>
```

---

3. Running the Tests

To run all tests in the `tests/` folder:

```bash
./vendor/bin/phpunit tests
# Or if you created the shortcut:
composer test
```

PHPUnit will display a summary of passed, failed, or skipped tests.

---

4. Real-World Example with a Class to Test

Imagine a `Calculator.php` class in `src/`:

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

    public function divide($a, $b)
    {
        if ($b === 0) {
            throw new InvalidArgumentException("Division by zero");
        }
        return $a / $b;
    }
}
?>
```

Test file `tests/CalculatorTest.php`:

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

require_once __DIR__ . '/../src/Calculator.php';

class CalculatorTest extends TestCase
{
    private $calculator;

    protected function setUp(): void
    {
        // Instantiate before each test
        $this->calculator = new Calculator();
    }

    public function testAdd()
    {
        $this->assertEquals(7, $this->calculator->add(3, 4));
    }

    public function testDivide()
    {
        $this->assertEquals(2, $this->calculator->divide(6, 3));
    }

    public function testDivideByZero()
    {
        $this->expectException(InvalidArgumentException::class);
        $this->calculator->divide(5, 0);
    }
}
?>
```

---

5. Example with HTML Integration

File `csrf.php` (function to test)

```php
<?php
session_start();

/**
```

```php
 * Generates a CSRF token, stores it in session, and outputs a hidden input
 */
function setCSRF(): void
{
    if (session_status() !== PHP_SESSION_ACTIVE) {
        session_start();
    }

    // Generate a random token
    $token = bin2hex(random_bytes(16));
    // Store the token in session
    $_SESSION['csrf_token'] = $token;

    // Output the hidden input with the token
    echo '<input type="hidden" name="csrf_token" value="' . $token . '">';
}
?>
```

File `CSRFTest.php` (PHPUnit test)

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

// Include the function to test
require_once 'csrf.php';

class CSRFTest extends TestCase
{
    protected function setUp(): void
    {
        // Start a session for each test
        if (session_status() !== PHP_SESSION_ACTIVE) {
            session_start();
        }

        // Clear the session before each test
        $_SESSION = [];
    }

    public function testSetCSRFOutputAndSession()
    {
        // Start output buffering
        ob_start();

        // Call the function that echoes and sets the session
        setCSRF();

        // Get the output
        $output = ob_get_clean();
        // Check that the output contains a hidden input with name csrf_token
        $this->assertStringContainsString('<input type="hidden"', $output);
        $this->assertStringContainsString('name="csrf_token"', $output);

        // Check that the value is a 32-character hexadecimal token
        preg_match('/value="([a-f0-9]{32})"/', $output, $matches);
        $this->assertNotEmpty($matches, "The CSRF token was not found or is
incorrect.");

        $tokenFromInput = $matches[1];

        // Check that the token is stored in session
        $this->assertArrayHasKey('csrf_token', $_SESSION);
        $this->assertEquals($tokenFromInput, $_SESSION['csrf_token']);
    }
}
?>
```

Explanations

- The `setCSRF()` function starts the session if not already active.
- It generates a token and stores it in `$_SESSION['csrf_token']`.
- It outputs an HTML input with the token.
- In the test, the session is also started (via `setUp()`).
- The session is cleared before each test to avoid interference.
- Output is captured to verify the HTML.
- It checks that the token in the input matches the one in the session.

With this pattern, you ensure that CSRF protection is correctly implemented on the server side and visible on the client side.

---

6. Common Assertions

Here are some useful assertions:

Method	**Description**
`assertEquals(\$expected, \$actual)`	Checks if the values are equal
`assertTrue(\$condition)`	Checks if the condition is true
`assertFalse(\$condition)`	Checks if the condition is false
`assertNull(\$variable)`	Checks if the variable is null
`assertInstanceOf(\$class, \$object)`	Checks if the object is an instance of the class
`expectException(\$exceptionClass)`	Expects an exception to be thrown

---

6. Organizing Your Tests

- Place your tests in a `tests/` folder at the root of your project.
- Structure the tests to reflect source namespaces or folders.
- Use `setUp()` and `tearDown()` to initialize and clean before/after each test.
- Name your methods with the `test` prefix followed by what you're testing.

---

7. Continuous Integration and Reports

- PHPUnit can generate XML reports (e.g., for Jenkins, GitHub Actions).
- Integrate PHPUnit into a CI pipeline to run tests automatically.
- Use `--coverage` to measure code coverage.

---

8. Useful Resources

- PHPUnit Official Documentation
- Symfony PHPUnit Guide
- Composer
- Example PHPUnit XML Configuration

> With PHPUnit, you can build a solid foundation of automated tests for your PHP projects, improve code reliability, and simplify maintenance.