

# **\*Swoole and Pest: A Modern Testing Approach**



## PUG Torino

- [PHP User Group Torino](#) is a group of web developers interested in the PHP language (and not only)
- We are part of [GrUSP](#) association
- On [meetup.com](#) we are about 621 members
- We have also a [mailing list](#) with more than 100 members
- [Toolbox Coworking](#) is sponsoring the group



TOOLBOX  
COWORKING





# Daniele Barbaro

Staff Software Engineer - Tech Lead

Avid hiker, motorcyclist, and baker.

Passionate about sharing knowledge within tech communities and pushing the boundaries of innovation in software engineering.



<https://daniele.barbaro.online>





Why are we here?!



Sadly, not for pizza time.







## What We'll Cover

- \*Swoole vs PHP Built-in Server
- Pest Testing Framework
- Building a Mock API
- Use a Mock API in a CI/CD





# SWOOLE

- Event-driven, non-blocking I/O
  - Built-in multi-threading
  - Coroutines support
  - Better memory management
  - Improved performance over traditional PHP servers
  - Enhanced configuration options
- 
- 



## Differences from Swoole

- Community-driven development
- More frequent updates and better community support
- Enhanced configuration options
- Improved error handling
- Better PHP 8.x compatibility



# Swoole vs PHP Built-in Server

```
// Swoole Server
$server = new Swoole\HTTP\Server('127.0.0.1', 9501);
$server->on('request', function ($request, $response) {
    $response->header('Content-Type', 'application/json');
    $response->end(json_encode(['message' => 'Hello
World']));
});
$server->start();
```

```
// PHP Built-in Server
$ php -S localhost:8000
```





# OpenSwoole

```
// OpenSwoole Server
<?php
use OpenSwoole\HTTP\Server;
use OpenSwoole\HTTP\Request;
use OpenSwoole\HTTP\Response;

$server = new Server('0.0.0.0', 9501);

$server->set([
    'worker_num' => 4,           // Multi-threading
    'enable_coroutine' => true,  // Async support
    'max_request' => 10000,      // Auto worker restart
    'hook_flags' => SWOOLE_HOOK_ALL // Full async hooks
]);

$server->on('request', function (Request $request, Response $response) {
    $response->header('Content-Type', 'application/json');
    $response->end(json_encode(['message' => 'Hello World']));
});

$server->start();
```



# Built-in Server Limitations

```
// PHP Built-in Server  
$ php -S localhost:8000
```

- ✗ Single-threaded
- ✗ One request at a time
- ✗ No WebSocket support
- ✗ No concurrent connections
- ✓ Easy to start
- ✓ Good for basic development



## \*Swoole Power

```
$server = new Swoole\HTTP\Server('127.0.0.1', 9501);

// Configure workers and processes
$server->set([
    'worker_num' => 4,
    'task_worker_num' => 2,
]);

$server->on('request', function ($request, $response) {
    $response->header('Content-Type', 'application/json');
    $response->end(json_encode(['message' => 'Hello
World']));
});

$server->start();
```

- ✓ Multi-threaded
- ✓ Concurrent requests
- ✓ WebSocket support
- ✓ Event-driven
- ✓ Memory management
- ✗ More complex setup



## Memory Usage Comparison:

Feature	PHP Built-in	OpenSwoole
Concurrent Users	1	10,000+
Memory Usage	~2MB/req	~2MB total
Response Time	~50ms	~2ms
Multi-threading	✗	✓
Async I/O	✗	✓
Hot Reload	✓	✗
Production Ready	✗	✓





## When to Use What?

### PHP Built-in Server

- Local development
- Simple applications
- Quick prototypes
- Debugging

### \*Swoole

- Mock API servers
- High concurrency needs
- WebSocket applications
- Production-like testing





## Why Testing Matters

*"Testing leads to failure, and failure leads to understanding"*  
- Burt Rutan





# Testing Best Practices

- Arrange, Act, Assert
- One Assertion per Test
- Clean Test Data
- Meaningful Names
- Isolated Tests



## Arrange, Act, Assert" (Pattern AAA)

```
it('calculates total price correctly', function() {  
    // Arrange  
    $cart = new Cart();  
    $cart->add(new Product('book', 10.00));  
  
    // Act  
    $total = $cart->getTotal();  
  
    // Assert  
    expect($total)->toBe(10.00);  
});
```





## One Assertion per Test

```
// NO ❌  
it('validates user data', function() {  
    $user = new User('john@example.com');  
    expect($user->email)->toBe('john@example.com');  
    expect($user->isValid())->toBeTrue();  
});  
  
// YES ✅  
it('sets user email correctly', function() {  
    $user = new User('john@example.com');  
    expect($user->email)->toBe('john@example.com');  
});
```





## Clean Test Data



```
beforeEach(function() {  
    $this->mockData = require 'tests/Mock/Data/mock-data.php';  
});  
  
afterEach(function() {  
    // Cleanup  
});
```



# Meaningful Names



// NO ❌

```
it('test1', function() {});
```

// YES ✅

```
it('returns 404 when station not found', function() {});
```





## Isolated Tests



```
it('fetches station data independently', function() {  
    $client = new Client($this->mockHttpClient);  
    // Test does not depend on other tests or global state  
});
```





# PEST

- Testing Framework
- Built on PHPUnit
- Focused on Simplicity
- Extensibility
- Powerful Tooling
- Focused on Developer Experience

# PHPUnit vs Pest: Syntax Comparison

```
// Pest
it('fetches stations', function () {
    $client = new Client();
    $result = $client->getStations();

    expect($result)
        ->toBeArray()
        ->toHaveCount(1)
        ->and($result[0])
        ->toHaveKey('translations');
});
```

```
// PHPUnit
public function testItFetchesStations(): void
{
    $client = new Client();
    $result = $client->getStations();

    $this->assertIsArray($result);
    $this->assertCount(1, $result);
    $this->assertArrayHasKey('translations',
        $result[0]);
}
```



# PHPUnit vs Pest: Syntax Comparison

```
describe('station translations', function() {
    it('fetches translations successfully', function ()
    {
        $result = $this->service->getStations();
        expect($result)->toBeArray()->toHaveCount(1);
    });

    it('validates language support', function () {
        $result = $this->service->getStations();
        expect($result[0]['translations'])
            ->toHaveKey('it')
            ->toHaveKey('en');
    });
});
```





## Lessons Learned

*"The best TDD can do, is assure that code does what the developer thinks it should do"*

*- James O Coplien*





# Mock Server Implementation

```
$server = new Swoole\HTTP\Server('127.0.0.1', 9501);

$server->on('request', function ($request, $response) {
    match ($request->server['request_uri']) {
        '/translations/stations' =>
            sendJson($response, $mockData['stations']),

        '/it/rally/stations/1' =>
            sendJson($response, $mockData['station']),

        default =>
            send404($response)
    };
});

function sendJson($response, $data) {
    $response->header('Content-Type', 'application/json');
    $response->end(json_encode($data));
}
```



# OpenAPI Integration

```
paths:
  /translations/stations:
    get:
      summary: Get stations translations
      responses:
        '200':
          content:
            application/json:
              schema:
                type: array
                items:
                  $ref: '#/components/schemas/Station'
```



# CI Pipeline Integration

```
name: Run Pest tests

on: ['push', 'pull_request']

jobs:
  pest-tests:
    [...]
    steps:
      [...]

      - name: Start Swoole server
        run: php tests/swoole-test-server.php &
        continue-on-error: true
        timeout-minutes: 1

      - name: Wait for Swoole to be ready
        run: |
          timeout 10s bash -c 'until echo > /dev/tcp/127.0.0.1/9501; do sleep 1; done'

      - name: Run Pest tests
        run: vendor/bin/pest --ci --colors=always --do-not-cache-result

      - name: Stop Swoole server
        run: |
          pkill -f tests/swoole-test-server.php

    [...]
```



# Demo Time





## Resources and Links

### \*Swoole:

- <https://openswoole.com>
- <https://www.swoole.com>

### PEST:

- <https://pestphp.com>

### Repo:

- <https://github.com/danielebarbaro/mock-api-testing>



# Grazie!



<https://daniele.barbaro.online>

<https://github.com/danielebarbaro/mock-api-testing>

