

Testowanie z Pest – cookbook

Setup, factories, feature, przeglądarka, fake'i, czas, arch, CI

1 · Setup i factories

```
<!-- phpunit.xml – testy dostają WŁASNE usługi -->
<env name="APP_ENV" value="testing"/>
<env name="BCRYPT_ROUNDS" value="4"/>
<env name="CACHE_STORE" value="array"/>
<env name="DB_CONNECTION" value="sqlite"/>
<env name="DB_DATABASE" value=":memory:"/>
<env name="MAIL_MAILER" value="array"/>
<env name="QUEUE_CONNECTION" value="sync"/>
<env name="SESSION_DRIVER" value="array"/>
```

```
// tests/Pest.php
uses(Tests\TestCase::class, LazilyRefreshDatabase::class)->in('Feature');
```

```
// factory modeluje HISTORIĘ, nie tylko wiersz – stany + relacje jako intencja
final class OrderFactory extends Factory {
    public function definition(): array {
        return ['customer_id' => Customer::factory(), 'status' => 'draft',
            'total_cents' => 0, 'placed_at' => null, 'paid_at' => null];
    }
    public function placed(): static {
        return $this->state(fn () => ['status' => 'placed', 'placed_at' => now()]);
    }
    public function paid(): static {
        return $this->placed()->state(fn () => ['status' => 'paid', 'paid_at' => now()]);
    }
    public function withLine(Product $p, int $qty = 1): static {
        return $this->afterCreating(fn (Order $o) => $o->lines()->create([
            'product_id' => $p->id, 'quantity' => $qty, 'unit_price_cents' => $p->price_cents]));
    }
}

// wywołanie ujawnia historię
$order = Order::factory()->paid()->withLine($product, quantity: 2)->create();
```

2 · Feature test = kontrakt HTTP

```
// walidacja na granicy HTTP; akcja wykonuje przypadek użycia
final class StoreAccountOrderRequest extends FormRequest {
    public function authorize(): bool {
        return $this->user()?->can('createOrder', $this->route('account')) ?? false;
    }
    public function rules(): array {
```

```

        return ['product_sku' => ['required', 'string', 'exists:products,sku'],
                'quantity' => ['required', 'integer', 'min:1', 'max:500']];
    }
}

```

```

it('rejects an unorderable product with 422', function (): void {
    $account = Account::factory()->create();
    $user = User::factory()->create();
    $account->users()->attach($user, ['role' => 'buyer']);
    $product = Product::factory()->create(['is_orderable' => false]);

    $this->actingAs($user)
        ->postJson(route('accounts.orders.store', $account), [
            'product_sku' => $product->sku, 'quantity' => 3,
        ])
        ->assertStatus(422)
        ->assertJsonValidationErrors('product_sku');
});

```

Autoryzacja i walidacja to **osobne zachowania** — testuj każde. Kontroler jest warstwą tłumaczącą Request → Action → Resource.

3 · Test przeglądarkowy = ścieżka, nie lista ekranów

```

beforeEach(function (): void {
    Http::preventStrayRequests(); // żaden test nie wychodzi do sieci
    $this->customer = User::factory()->create();
    $this->billingCategory = SupportCategory::factory()->create(['name' => 'Billing', 'is_active' => true]);
});

it('lets a customer submit a support request', function (): void {
    $this->actingAs($this->customer);

    visit(route('support-requests.create'))
        ->assertSee('New support request')
        ->assertNoJavaScriptErrors()
        ->fill('Subject', 'Invoice contains the wrong company name')
        ->select('Category', $this->billingCategory->name)
        ->fill('Message', 'Please correct the name before the deadline.')
        ->click('Send request')
        ->waitForText('Your request has been sent') // czekaj na WYNIK widoczny dla klienta
        ->assertNoJavaScriptErrors();

    expect(SupportRequest::query()->where([
        'user_id' => $this->customer->id, 'subject' => 'Invoice contains the wrong company name',
    ])->exists())->toBeTrue();
});

```

Wybieraj kontrolki jak użytkownik (label, tekst), twórz cały stan jawnie, pokryj **jedną** awarię, z której można się podnieść (np. brak wymaganego pola → formularz zostaje).

4 · Fake'i, czas, granice zewnętrzne

```
Event::fake([OrderPaid::class]); // fakuje WĄSKO – tylko to zdarzenie
Event::assertDispatched(OrderPaid::class, fn (OrderPaid $e) => $e->order->is($order));

Mail::fake(); Queue::fake();
Mail::assertQueued(OrderReceipt::class, fn ($m) => $m->hasTo($order->customer->email));
Queue::assertPushed(GenerateInvoicePdf::class, fn ($j) => $j->order->is($order));

Storage::fake('invoices');
Storage::disk('invoices')->assertExists("{ $order->id }/invoice.pdf");

Http::preventStrayRequests();
Http::fake(['https://accounting.example.test/api/invoices' => Http::response(['id' => 'inv_123'], 201)]);
Http::assertSent(fn (Request $r) => $r->url() === '.../api/invoices' && $r['reference'] === $order->external_reference);

// czas – wokół granicy biznesowej
$this->travelTo(Carbon::parse('2026-09-06 10:00:00'));
$link = DownloadLink::factory()->create(['expires_at' => now()->addMinutes(15)]);
$this->travel(16)->minutes();
$this->get(route('downloads.show', $link))->assertGone();
$this->travelBack();
```

KONTRPRZYKŁAD Zostaw jeden test *integracyjny*, który przechodzi realną ścieżkę (payment → listener → dispatch joba) bez fake'ów tam, gdzie liczy się połączenie.

5 · Testy architektury + CI

```
arch('HTTP controllers are thin')
    ->expect('App\\Http\\Controllers')
    ->toOnlyUse(['App\\Http\\Requests', 'App\\Http\\Resources', 'App\\Actions', 'Illuminate']);

arch('domain code does not depend on infrastructure')
    ->expect('App\\Domain')
    ->not->toUse('App\\Infrastructure')
    ->not->toUse('Illuminate\\Support\\Facades');
```

```
# CI: najpierw tanie deterministyczne kontrole, potem zachowanie
quality: vendor/bin/pint --test + phpstan analyse (matrix PHP 8.4/8.5)
tests:   php artisan test --compact --parallel --shard=${{ matrix.shard }}/4
browser-contract: php artisan test --compact --group=browser-contract
```

Każdy shard = rozłączny fragment; branch protection wymaga **każdego**. Pokrycie to *pytanie*, dlaczego gałąź nie została wykonana — nie cel liczbowy. Raport HTML generuj przy badaniu zmiany:

```
XDEBUG_MODE=coverage php artisan test --compact --coverage-html=build/coverage
```

ZANIM WYMUSISZ Pozwól zespołowi zobaczyć kilka błędów i uzgodnić drogę naprawy. Reguła kontrolera potrzebuje oczywistej akcji przenoszącej pracę; niestabilny test równoległy — odtwarzalnej poprawki fixture'u. Trzymaj regułę, komendę i komunikat błędu blisko siebie.