Testing Guide - Understanding White Box, Black Box, and Unit Tests

Table of Contents

What is Testing and Why Do We Test?

The Problem Testing Solves

Imagine you build a car. Would you:

- **Option A**: Build it, hope it works, sell it to customers ❌
- **Option B**: Test the brakes, engine, airbags before selling ✅

Software is the same. Testing ensures:

1. **It works correctly** (does what it's supposed to do)
2. **It doesn't break** when you add new features
3. **It's safe** (prevents security issues)
4. **You can prove it works** (for clients, instructors, employers)

Real-World Example

```php
// Without testing:
function divide($a, $b) {
    return $a / $b;
}
// What if $b is 0? App crashes! 💥

// With testing:
function divide($a, $b) {
    if ($b === 0) {
        throw new Exception("Cannot divide by zero");
    }
    return $a / $b;
}
// Test catches this bug before production ✅
```

Unit Tests Explained

What is a "Unit"?

A **unit** is the **smallest testable piece** of code:

- A single function
- A single method
- A single class

Think of it like testing individual LEGO bricks before building the castle.

Example: Testing the Admin Model

```php
// File: tests/Unit/AdminTest.php

/**
 * WHY THIS TEST EXISTS:
 * The Admin model is a "unit" - it's a single class
 * We test it in isolation (without database, without controllers)
 */
class AdminTest extends TestCase
{
    /**
     * TEST: Can we create an admin?
     *
     * WHY WE TEST THIS:
     * If we can't create admins, the whole system fails
     *
     * HOW IT WORKS:
     * 1. Create new Admin object
     * 2. Set name, email, password
     * 3. Check if values were stored correctly
     *
     * WHAT "PASSING" MEANS:
     * - Name matches what we set
     * - Email matches what we set
     * - Password was hashed properly (not stored as plain text)
     */
    public function test_admin_can_be_created()
    {
        $admin = new Admin();
        $admin->name = 'Test Admin';
        $admin->email = 'admin@test.com';
        $admin->password = 'testpassword';

        // ASSERTION: Expected value === Actual value
        $this->assertEquals('Test Admin', $admin->name);
        $this->assertEquals('admin@test.com', $admin->email);

        // Password should be HASHED, not plain text
        $this->assertTrue(Hash::check('testpassword', $admin->password));
    }
}
```

Why is it Called "Unit" Testing?

Term	Meaning	Example
Unit	Single, isolated piece	Admin model (one class)
Integration	Multiple pieces working together	Admin model + Database
System	Entire application	Login → Dashboard → Logout flow

White Box vs Black Box Testing

The Analogy: Testing a Toaster

Black Box Testing (External View) ❤

You DON'T see inside the toaster. You only see:

- Input: Bread goes in
- Output: Toast comes out

Testing approach:

- Put bread in → Check if toast comes out
- Don't care HOW it toasts (heating element? magic?)
- Only care THAT it toasts

Example in Code:

```
// BLACK BOX: Test the LOGIN as a user would experience it
public function test_user_can_login()
{
    // ACT: Submit login form (like clicking "Login" button)
    $response = $this->post('/login', [
        'Email' => 'test@example.com',
        'password' => 'password123'
    ]);

    // ASSERT: User sees success (redirected to home)
    $response->assertRedirect('/home');

    // We don't care HOW Laravel authenticated
    // We just care that login WORKED
}
```

White Box Testing (Internal View) ♡

You CAN see inside the toaster. You test:

- The heating element gets hot
- The timer counts down correctly
- The spring mechanism pops toast up

Testing approach:

- Test each internal component
- Verify internal logic
- Check calculations and algorithms

Example in Code:

```
// WHITE BOX: Test the INTERNAL logic of password hashing
public function test_admin_password_is_hashed()
{
    $admin = new Admin();
    $admin->password = 'mysecretpassword';

    // We know INTERNALLY that password should be hashed
    // We're testing the INTERNAL behavior (hashing function)
    $this->assertNotEquals('mysecretpassword', $admin->password);
    $this->assertTrue(Hash::check('mysecretpassword', $admin->password));
}
```

Side-by-Side Comparison

Aspect	Black Box ❤	White Box ♡
What you test	Outputs/Results	Internal logic
Knowledge needed	None (just requirements)	Full code understanding
Example	"Login works"	"Password hashing works"
File location	`tests/Feature/`	`tests/Unit/`
Tools	Browser simulation	Direct class calls

In Our Project

White Box (Unit Tests)

Located in: `tests/Unit/`

Example: `AdminTest.php`

```php
// WHITE BOX: We know Admin model has $fillable property
public function test_admin_has_correct_fillable()
{
    $admin = new Admin();
    $fillable = $admin->getFillable();  // Internal method

    // We're testing INTERNAL property
    $this->assertContains('name', $fillable);
    $this->assertContains('email', $fillable);
}
```

Why white box?

- We're looking at internal properties (`$fillable`)
- We know how the code works internally
- We're testing implementation details

Black Box (Feature Tests)

Located in: `tests/Feature/`

Example: `LoginTest.php`

```php
// BLACK BOX: We're testing like a user (external behavior)
public function test_login_page_is_accessible()
{
    // ACT: Visit login page (like typing URL in browser)
    $response = $this->get('/login');

    // ASSERT: Page loads successfully
    $response->assertStatus(200);
    $response->assertSee('LOG IN');  // Text visible on page

    // We don't care HOW the page renders
    // We just care that it DOES render
}
```

Why black box?

- We're acting like an external user
- We don't look at internal code
- We only check visible results (HTTP status, page content)

Our Project's Tests

Tests in `tests/Unit/`

1. `AdminTest.php` (White Box)

Purpose: Test Admin model's internal behavior

Tests included:

```
test_admin_can_be_created()
// Why: Ensures basic object creation works
// Criteria to pass: name, email, password can be set and retrieved correctly

test_admin_has_correct_fillable_attributes()
// Why: Ensures mass assignment protection works
// Criteria to pass: fillable array contains 'name', 'email', 'password'

test_admin_hides_sensitive_attributes()
// Why: Security - password shouldn't appear in JSON responses
// Criteria to pass: hidden array contains 'password', 'remember_token'

test_admin_uses_correct_guard()
// Why: Ensures admins use 'admin' guard, not 'web' guard
// Criteria to pass: guard property equals 'admin'
```

Why these specific tests?

1. **Creation test**: Most fundamental - if you can't create objects, nothing works
2. **Fillable test**: Laravel security feature - must protect against mass assignment attacks
3. **Hidden test**: Security - prevents password leaks in API responses
4. **Guard test**: Critical for our dual-auth system (customers vs admins)

2. `UserTest.php` (White Box)

Purpose: Test User model's custom methods

Example:

```
test_user_can_register()
// Why: register() is a custom method (not standard Laravel)
// Criteria to pass:
//    - User is created in database
//    - Password is hashed
//    - All fields are set correctly
```

3. `AuthManagerTest.php` (White Box)

Purpose: Test authentication service logic

Example:

```
test_authenticate_with_valid_credentials()
// Why: Core authentication logic must work
// Criteria to pass:
//    - Returns true for valid email/password
//    - User is logged in (session created)

test_authenticate_with_invalid_credentials()
// Why: Must reject incorrect passwords
// Criteria to pass:
//    - Returns false for wrong password
//    - User is NOT logged in
```

Tests in `tests/Feature/`

1. `LoginTest.php` (Black Box)

Purpose: Test login flow as a user experiences it

Example:

```
test_successful_login_redirects_to_home()
// Why: Users expect to reach home after login
// Criteria to pass:
//    - POST to /login with valid credentials
//    - Response is redirect to /home
//    - User is authenticated
```

2. `SignupTest.php` (Black Box)

Purpose: Test registration flow

Example:

```
test_successful_registration_creates_account()
// Why: Users must be able to create accounts
// Criteria to pass:
//    - POST to /register with valid data
//    - User appears in database
//    - User is automatically logged in
//    - Redirected to home
```

How to Determine What to Test

The 3-Question Method

For every piece of code, ask:

1. What does it DO? (Functional test)

```
// UserController has a register() method
// TEST: Does registration create a user?
test_registration_creates_user()
```

2. What could go WRONG? (Edge case test)

```
// What if email already exists?
// TEST: Duplicate email should fail
test_registration_fails_with_duplicate_email()
```

3. Is it SECURE? (Security test)

```
// Passwords must be hashed
// TEST: Password is never stored in plain text
test_password_is_hashed_in_database()
```

Priority Levels for Testing

Priority	What to Test	Example
🔴 **Critical**	Core features that, if broken, make app unusable	Login, Registration, Authentication
🟡 **Important**	Secondary features users rely on	Password reset, Profile update
🟢 **Nice-to-have**	Edge cases and rare scenarios	Multiple login attempts, Special characters in name

Our Project's Test Decisions

Why we test Admin model:

- 🔴 Critical: Entire admin system depends on it
- Security risk if broken
- Custom guard logic must work

Why we test AuthManager:

- 🔴 Critical: All authentication flows through it
- If broken, nobody can log in
- Contains sensitive security logic

Why we test registration flow:

- 🔴 Critical: Can't get new users without it
- Must prevent duplicate accounts
- Must hash passwords securely

What we DON'T test (and why):

- ❌ Laravel's built-in Auth facade (already tested by Laravel team)
- ❌ Database connection (tested by framework)
- ❌ Blade rendering engine (tested by Laravel)

Test Criteria and Passing Requirements

What Makes a Test "Pass"? ✅

A test passes when **ALL assertions succeed**.

Example: Test with 3 Assertions

```
public function test_admin_creation()
{
    $admin = new Admin();
    $admin->name = 'John';
    $admin->email = 'john@example.com';

    // ASSERTION 1: Name must match
    $this->assertEquals('John', $admin->name);  // ✅ Pass

    // ASSERTION 2: Email must match
    $this->assertEquals('john@example.com', $admin->email);  // ✅ Pass

    // ASSERTION 3: Model must be Admin instance
    $this->assertInstanceOf(Admin::class, $admin);  // ✅ Pass

    // ALL 3 PASS = TEST PASSES ✅
}
```

What if ONE assertion fails?

```
public function test_admin_creation()
{
    $admin = new Admin();
    $admin->name = 'John';

    $this->assertEquals('John', $admin->name);       // ✅ Pass
    $this->assertEquals('wrong@test.com', $admin->email);  // ❌ FAIL

    // TEST FAILS ❌ (even though first assertion passed)
    // Error: "Expected 'wrong@test.com', got null"
}
```

Common Assertion Types

Assertion	What It Checks	Example
`assertEquals(expected, actual)`	Values are equal	`assertEquals(5, $result)`
`assertTrue($value)`	Value is boolean true	`assertTrue($admin->isActive())`
`assertFalse($value)`	Value is boolean false	`assertFalse($user->isBanned())`
`assertContains(item, array)`	Array contains item	`assertContains('email', $fillable)`
`assertInstanceOf(Class, object)`	Object is of correct type	`assertInstanceOf(Admin::class, $admin)`

Our Test Criteria

For Admin Model Tests

Test: `test_admin_can_be_created`

- ✅ Pass criteria:
  - `$admin->name` equals what we set
  - `$admin->email` equals what we set
  - Password is hashed (checked with `Hash::check()`)

Test: `test_admin_has_correct_fillable`

- ✅ Pass criteria:
  - Fillable array contains 'name'
  - Fillable array contains 'email'
  - Fillable array contains 'password'

For Login Feature Tests

Test: `test_successful_login`

- ✅ Pass criteria:
  - HTTP status is 302 (redirect)
  - Redirect location is '/home'
  - User is authenticated (`Auth::check()` returns true)
  - Session contains user ID

Test: `test_failed_login`

- ✅ Pass criteria:
  - HTTP status is 302 (redirect back to login)
  - Error message is displayed
  - User is NOT authenticated
  - Session does NOT contain user ID

Running Tests and Reading Results

Command:

```
php artisan test
```

Successful Output:

```
PASS  Tests\Unit\AdminTest
✓ admin can be created (5ms)
✓ admin has correct fillable (3ms)
✓ admin hides sensitive attributes (2ms)

Tests:    45 passed
Duration: 1.96s
```

What this means:

- All assertions in all tests passed ✅
- Total 45 tests ran
- Took 1.96 seconds

Failed Output:

```
FAILED  Tests\Unit\AdminTest > admin uses correct guard

Failed asserting that null matches expected 'admin'.

at tests/Unit/AdminTest.php:61
```

What this means:

- Test `test_admin_uses_correct_guard` failed ❌
- Expected value: 'admin'
- Actual value: null
- Failure at line 61

Summary Table: Testing Terminology

Term	Simple Explanation	Example
Unit Test	Test one small piece in isolation	Test Admin model alone
White Box	Test internal code/logic	Test that password gets hashed
Black Box	Test external behavior	Test that login page loads
Assertion	Check if something is true	`assertEquals(5, $result)`
Test Case	Collection of tests	AdminTest.php has 4 test methods
Test Suite	Collection of test cases	Unit suite = all files in tests/Unit/

Testing Best Practices (For Your Presentation)

What to Say When Asked "Why did you write this test?"

❌ **Bad answer:**

> "Because the assignment required tests."

✅ **Good answer:**

> "I wrote `test_admin_can_be_created` to ensure the Admin model's core functionality works. If this test fails, it means admins can't be created, which breaks the entire admin authentication system. It's a white box test because I'm testing internal properties like name and email assignment."

What to Say When Asked "How do you know your tests are good?"

✅ **Good answer:**

> "Good tests have three qualities:
>
> 1. **They catch bugs** - If I break the Admin model, this test fails
> 2. **They're clear** - Test name says exactly what it tests
> 3. **They're fast** - No database calls, just pure logic testing
>
> Our tests run in under 2 seconds for 45 tests, which means they're efficient."

Quick Reference: Our Project's Tests

File	Type	What It Tests	Why Black/White Box
`AdminTest.php`	White Box Unit	Admin model properties and methods	We test internal structure
`UserTest.php`	White Box Unit	User model custom register() method	We test internal logic
`AuthManagerTest.php`	White Box Unit	Authentication service methods	We test internal auth flow
`LoginTest.php`	Black Box Feature	Login flow from user perspective	We test external behavior
`SignupTest.php`	Black Box Feature	Registration flow	We test external form submission

Total: 45 tests passing ✅