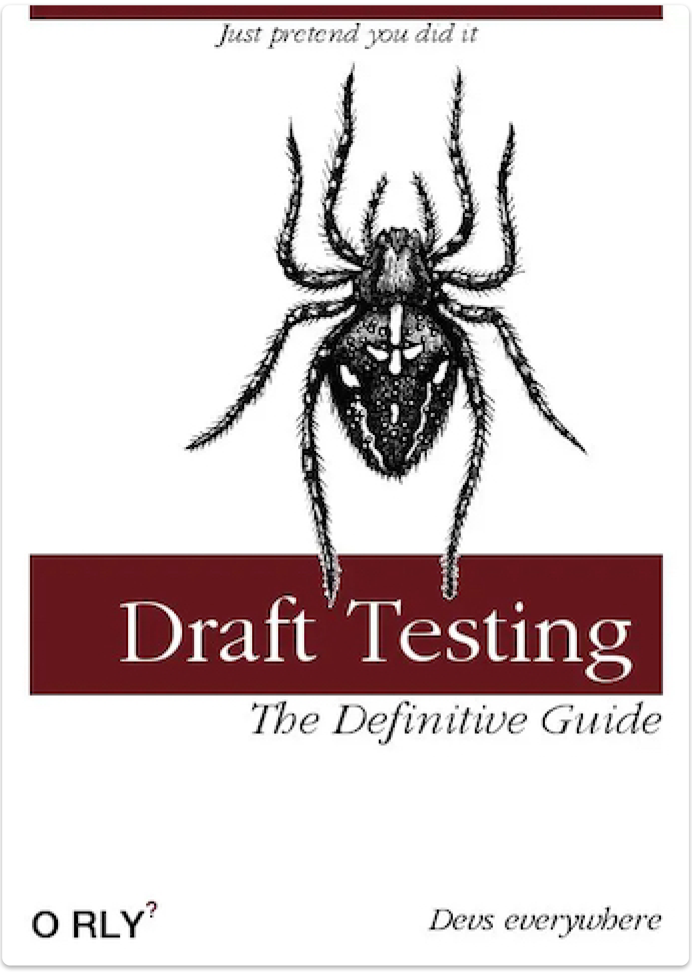
TDD Example

Software as a Service - Back-End Development
Session 03

Developed by Adrian Gould



Contents

Setting Up Pest in Laravel 11

Install Pest

Quite often Pest is installed as part of the creation of the base application. But you may ensure it is by using:

```
composer require pestphp/pest --dev
composer require pestphp/pest-plugin-laravel --dev
```

Pest is only needed in development situations, and that will include the CI/CD testing process.

Configure Pest

- Run `php artisan pest:install` to set up Pest in your Laravel project.
- This command will create a `tests` directory with example tests.

Writing Your First Test

Create a Test File

- Navigate to the `tests/Feature` directory.
- Create a new test file, e.g., `UserTest.php`.

You may also do this using:

```
php artisan make:test UserTest
```

Write a Failing Test

```php
use App\Models\User;
use Illuminate\Foundation\Testing\RefreshDatabase;

uses(RefreshDatabase::class);

it('can create a user', function () {
    $response = $this->post('/api/users', [
        'name' => 'John Doe',
        'email' => 'john@example.com',
        'password' => 'password',
    ]);

    $response->assertStatus(201);
    $this->assertDatabaseHas('users', [
        'email' => 'john@example.com',
    ]);
});
```

Implementing the Code

Create a Route

```php
// routes/api.php
Route::post('/users', [UserController::class, 'store']);
```

Create a Controller

```
php artisan make:controller UserController
```

Implement the Controller Method

```php
// app/Http/Controllers/UserController.php
namespace App\Http\Controllers;

use App\Models\User;
use Illuminate\Http\Request;

class UserController extends Controller
{
    public function store(Request $request)
    {
        $request->validate([
            'name' => 'required',
            'email' => 'required|email|unique:users',
            'password' => 'required',
        ]);

        $user = User::create([
            'name' => $request->name,
            'email' => $request->email,
            'password' => bcrypt($request->password),
        ]);

        return response()->json($user, 201);
    }
}
```

Running the Test

Run the Test

```
php artisan test
```

Check the Output

- Ensure the test passes.
- If it fails, debug the code and rerun the test.

Refactoring

Improve the Code

- Refactor the code to improve readability, performance, or maintainability.
- Ensure the tests still pass after refactoring.

Writing More Tests

Test Different Scenarios

Write tests for different scenarios, such as validation errors, duplicate emails, etc.

```
it('validates user data', function () {
    $response = $this->post('/api/users', [
        'name' => '',
        'email' => 'invalid-email',
        'password' => '',
    ]);

    $response->assertStatus(422);
    $response->assertJsonValidationErrors(['name', 'email', 'password']);
});

it('does not allow duplicate emails', function () {
    User::factory()->create(['email' => 'john@example.com']);

    $response = $this->post('/api/users', [
        'name' => 'John Doe',
        'email' => 'john@example.com',
        'password' => 'password',
    ]);

    $response->assertStatus(422);
    $response->assertJsonValidationErrors(['email']);
});
```

Continuous Integration

Set Up CI/CD

Integrate your tests with a CI/CD pipeline (e.g., GitHub Actions, GitLab CI) to automatically run tests on every commit.

Exercises

The exercises here require you to complete the B R E A D process.

Exercise 1: Add User

Complete this tutorial.

Did you encounter any issues?

Exercise 2: Browse User

Complete the TDD process for Browsing the list of users.

Exercise 3: Read User

Complete the TDD process for Reading a single user.

Exercise 4: Edit User

Complete the TDD process for editing a user's details, including, but not limited to...

- change password
- change name
- change email

Exercise 5: Delete User

Complete the TDD process to delete a user from the application.