

Pierde el miedo a hacer pruebas (TDD con Laravel)

2020-04-20 :: Alfredo Mendoza


#php #backend #Laravel #TDD #Testing #web

Tabla de Contenidos

- ¿Qué es TDD?
- TDD y el método científico
- El ciclo
- ¿Cómo implementar TDD?
- ...y en Laravel?
 - Fuentes

Cuando alguien (como yo) escuchó por primera vez “**Pruebas de Software**”, uno empieza a pensar que es un tema para programadores avanzados ó ‘*Senior*’ y dices: “Ya cuando tenga más experiencia lo aprenderé”.

Sin embargo, hacer pruebas (sobre todo en Laravel) es **sencillo**, y es más...

> Si sabes hacer TDD , es un gran **PLUS** para tu CV y experiencia, además de que te sientes y te sientan como todo un profesional 😊.

Ahora veamos qué es TDD de una manera sencilla, para que le pierdas el miedo ha hacer pruebas, aunque te consideres novato ó un ‘*Junior*’:

¿Qué es TDD?

TDD significa *Test Driven Development* ó *Desarrollo guiado por pruebas* en donde primero escribes pruebas y luego escribir el software (y si quieres después refactorizas). Los principios son los siguientes:

- **Pruebas** antes de hacer “*código*” (“código” que deseas que tu aplicación realice).
- Una prueba a la vez; un prueba pasa y sigues implementando la siguiente.
- Hacer pruebas simples.

TDD y el método científico

En este [post](#) el autor hace una analogia entre el *método científico* y *TDD* y es muy interesante porque en lugar de aprender del mundo (*método científico*) creamos uno nuevo (*TDD*).


Método Científico	TDD
Pregunta	Requerimiento
Predicción	Resultado Esperado
Experimento	Afirmación de la Prueba
Resultados	Implementación de Código

El ciclo

El ciclo básicamente es el siguiente:

- RED 🟡🙄 (funcionalidad faltante)
- GREEN 🟢👍 (escribiste código para arreglar el problema)
- REFACTOR* 🔄🧐 (si es necesario)

¿Cómo implementar TDD?

1. Escribe la prueba 
2. Ejecuta la prueba 🐞
3. Arregla el código ✂
4. Ejecuta de nuevo la prueba 🐞🐞
5. Refactoriza* 🔄🧐 (opcional)
6. Y Repite 🔄🧐

...y en Laravel?

En Laravel, ya viene implementado el soporte para **PHPUnit** que es la herramienta que nos ayudará a hacer pruebas.

1. Configura el archivo `phpunit.xml` (solo si necesitas algo más)
2. Crea una nueva prueba (Feature ó Unit) `php artisan make:test PostManagementTest`

Feature	Unit (agrega <code>--unit</code>)
Pruebas de funcionalidad, para piezas de código largas como hacer una <i>Petición HTTP</i> .	Pruebas unitarias, para piezas de código pequeñas y aisladas del resto.

3. Escribe la prueba 

```
/* tests/Feature/PostManagementTest.php */
class PostManagementTest extends TestCase
{
    use RefreshDatabase;

    /** @test */
    function a_post_can_be_created()
    {
        // Para que nos arroje errores apropiados
        $this->withoutExceptionHandling();

        // Envio una petición HTTP [POST] para que almacene un Post nuevo
        $response = $this->post('/posts', [
            'title' => 'Test Title',
            'content' => 'Test Content'
        ]);

        // Verificamos si todo está bien
        $response->assertOk();

        // Verificamos si en realidad existe mi post
        $this->assertCount(1, Post::all());

        // Recupero el post que mande a crear
        $post = Post::first();

        // Verificamos si los datos de mi post creado son iguales a los que mande
        $this->assertEquals('Test Title', $post->title);
        $this->assertEquals('Test Content', $post->content);
    }
}
```

4. Ejecuta la prueba 🐞 `.\vendor\bin\phpunit --filter a_post_can_be_created` (corremos el binario que hará ejecutar nuestras pruebas)
5. RED 🟡🙄
6. Arregla el código ✂ (implementa lo necesario para que funcione: *routes, models, controllers, migrations, etc*)

```
/***** EJECUTAMOS EL TEST (🟡 Te dice que no existe la ruta) *****/

/* routes/web.php */
Route::post('/posts', 'PostController@store');

/***** EJECUTAMOS EL TEST (🟡 Te dice que no existe el controlador) *****/

/* app/Http/Controllers/PostController.php */
class PostController extends Controller
{
    // implementamos la funcionalidad
    public function store()
    {
        $data = request()->validate([
            'title' => '',
            'content' => ''
        ]);

        Post::create($data);
    }
}

/***** EJECUTAMOS EL TEST (🟡 Te dice que no existe el modelo) *****/

/* app/Post.php */
class Post extends Model
{
    // admitimos mass assignmet
    protected $guarded = [];
}

/***** EJECUTAMOS EL TEST (🟡 Te dice que no existe el campo title en la tabla pos *****/

/* database/migrations/2020_02_13_152154_create_posts_table.php */
class CreatePostsTable extends Migration
{
    public function up()
    {
        // agregamos los atributos necesarios
        Schema::create('posts', function (Blueprint $table) {
            $table->bigIncrements('id');
            $table->string('title');
            $table->text('content');
            $table->timestamps();
        });
    }
}
```

7. Ejecuta de nuevo la prueba 🐞🐞 `.\vendor\bin\phpunit --filter a_post_can_be_created`
8. GREEN 🟢👍
9. Refactoriza* 🔄🧐 (solo si es necesario)
10. Y hazlo otra vez.

Así de sencillo es hacer pruebas

Tal vez necesitas un ejemplo para consolidar todo esto, pues para tu suerte hice un **tutorial práctico**, donde realizamos un **CRUD aplicando TDD** y solo necesitas saber las **bases de Laravel**.

Fuentes

- [Testing Software: What is TDD?](#) 📖 te recomiendo echarle un ojo 🧐.
- [What is Test-Driven Development? \(And How To Get It Right\)](#)

LEE OTRAS PUBLICACIONES

← ¿Por qué Tailwind CSS es tan especi...

Funciones flecha en PHP 7.4 →