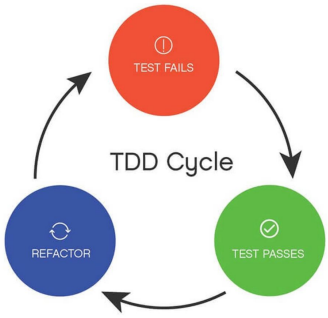
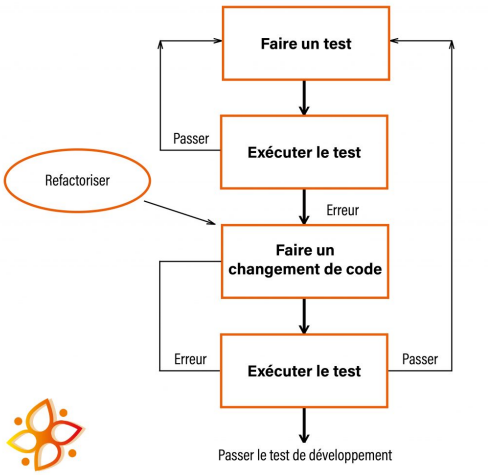
Les tests (TDD - Test Driven Development) avec Laravel

Le développement piloté par les tests (TDD) avec Laravel consiste à écrire des tests avant d'écrire le code réel de votre application. Cette approche garantit que votre code répond aux exigences et fonctionne comme prévu dès le départ. Laravel, avec ses outils de test intégrés, fait du TDD un processus pratique et efficace.



Étapes pour le TDD dans Laravel

1. **Configurez votre environnement de test**
2. **Écrivez un test en échec**
3. **Écrivez le code minimum pour réussir le test**
4. **Refactor**
5. **Répéter**



1. Configurez votre environnement de test

Assurez-vous que votre application Laravel est configurée et que les dépendances sont installées :

```
composer install
```

Installation - Composer

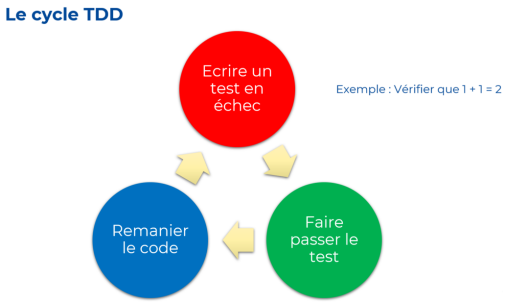
```
# composer.json
{
    "require-dev": {
        "phpunit/phpunit": "4.5.*"
    }
}

$ composer install
```

10

Assurez-vous que l'environnement de test est correctement configuré. Laravel utilise PHPUnit par défaut pour les tests. Le fichier de configuration de PHPUnit est `phpunit.xml`.

2. Écrire un test qui échoue

Commencez par écrire un test qui échouera parce que la fonctionnalité que vous testez n'existe pas encore.



Exemple : tester l'enregistrement de l'utilisateur

Créez un nouveau test à l'aide de la commande Artisan :

```
php artisan make:test UserRegistrationTest
```

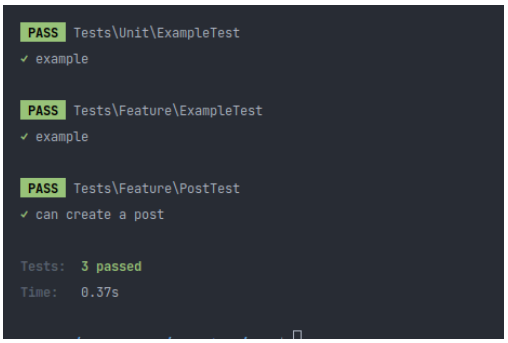
Cela crée un fichier de test dans le répertoire `tests/Feature`. Ouvrez ce fichier et écrivez un test pour l'enregistrement de l'utilisateur.

```php
<?php

namespace Tests\Feature;

use Illuminate\Foundation\Testing\RefreshDatabase;
use Tests\TestCase;

class UserRegistrationTest extends TestCase
{
    use RefreshDatabase;

    /** @test */
    public function a_user_can_register()
    {
        $response = $this->post('/register', [
            'name' => 'John Doe',
            'email' => 'john@example.com',
            'password' => 'password',
            'password_confirmation' => 'password',
        ]);
```

```
        $response->assertStatus(302);
        $this->assertDatabaseHas('users', [
            'email' => 'john@example.com',
        ]);
    }
}
```

3. Écrivez le code minimum pour réussir le test

Maintenant, écrivez le code pour réussir le test. Créez la route d'enregistrement et la logique du contrôleur.



Créer la route d'enregistrement

Ajoutez la route suivante dans `routes/web.php` :

```
use App\Http\Controllers\Auth\RegisterController;

Route::post('/register', [RegisterController::class, 'register']);
```

Créer le RegisterController

Créer le contrôleur s'il n'existe pas :

```
php artisan make:controller Auth/RegisterController
```

Ensuite, ajoutez la méthode `register` :

```
<?php

namespace App\Http\Controllers\Auth;
```

```php
use App\Http\Controllers\Controller;
use App\Models\User;
use Illuminate\Http\Request;
use Illuminate\Support\Facades\Hash;
use Illuminate\Support\Facades\Validator;

class RegisterController extends Controller
{
    public function register(Request $request)
    {
        $this->validate($request, [
            'name' => 'required|string|max:255',
            'email' => 'required|string|email|max:255|unique:users',
            'password' => 'required|string|min:8|confirmed',
        ]);

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

        return redirect('/');
    }
}
```

4. Refactorisation

Une fois le test réussi, examinez votre code et refactorisez-le si nécessaire. Assurez-vous que votre code est propre et maintenable.

5. Répétez

Continuez à écrire le prochain test en échec, en implémentant le code minimum pour réussir, puis en refactorisant.

Exécution de vos tests

Exécutez vos tests à l'aide de la commande Artisan :

```
php artisan test
```

ou

```
vendor/bin/phpunit
```

Exemple d'un cycle complet

1. **Écrire un test en échec** : ajouter une fonctionnalité ou une fonction requise.
2. **Écrire le code pour réussir le test** : implémenter juste assez de code pour que le test réussisse.
3. **Refactoriser** : nettoyer votre code, en vous assurant qu'il est efficace et maintenable.

4. **Répéter** : passer à la fonctionnalité ou à la fonction suivante.

Conclusion

En suivant ces étapes, vous pouvez appliquer efficacement TDD dans votre application Laravel, en vous assurant que votre base de code est robuste, maintenable et exempte de bugs.