TDD and Refactoring

Software as a Service - Back-End Development
Session 03

Developed by Adrian Gould

Contents

Refactoring

A very important part of the TDD process is refactoring code.

What is refactoring?

Code refactoring is the process of restructuring existing computer code—changing the factoring—without changing its external behaviour.

This improves non-functional attributes of the software, such as maintainability, extensibility, readability, and simplicity.

Refactoring is a disciplined way to clean up code that minimizes the chances of introducing bugs.

Why Refactor?

- Improved Readability: Makes the code easier to understand.
- Maintainability: Easier to modify and extend.
- Performance: Sometimes, refactoring can lead to performance improvements.
- Reduced Technical Debt: Helps in managing and reducing technical debt over time.

Benefits of Refactoring

- Separation of Concerns: The controller is now responsible only for handling the request and response, while the request class handles validation and the service class handles business logic.
- Reusability: The validation logic and user creation logic can be reused in other parts of the application.
- Testability: The service class can be easily tested in isolation.

Practical Example (Create)

Let's consider a simple example where we refactor a controller method to improve its structure and readability.

Before Refactoring

Suppose we have a `UserController` with a `store` method that handles user creation and validation directly within the method.

```
// 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);
    }
}
```

After Refactoring

We'll refactor the code by moving the validation logic into a dedicated request class and encapsulating the user creation logic into a service class.

Step 1: Create a Request Class

First, create a request class to handle validation.

```
php artisan make:request StoreUserRequest
```

Define the validation rules in the request class:

```
// app/Http/Requests/StoreUserRequest.php
namespace App\Http\Requests;

use Illuminate\Foundation\Http\FormRequest;

class StoreUserRequest extends FormRequest
{
    public function authorize()
    {
        return true;
    }

    public function rules()
    {
        return [
            'name' => 'required',
            'email' => 'required|email|unique:users',
            'password' => 'required',
        ];
    }
}
```

Step 2: Create a Service Class

Next, create a service class to handle the user creation logic.

```
php artisan make:service UserService
```

Define the user creation method in the service class:

```
// app/Services/UserService.php
namespace App\Services;

use App\Models\User;

class UserService
{
    public function createUser(array $data)
    {
        return User::create([
            'name' => $data['name'],
            'email' => $data['email'],
            'password' => bcrypt($data['password']),
        ]);
    }
}
```

Step 3: Update the Controller

Finally, update the `UserController` to use the request class and service class.

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

use App\Http\Requests\StoreUserRequest;
use App\Services\UserService;

class UserController extends Controller
{
    protected $userService;

    public function __construct(UserService $userService)
    {
        $this->userService = $userService;
    }

    public function store(StoreUserRequest $request)
```

```
    {
        $user = $this->userService->createUser($request->validated());

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