Laravel - Contracts

Topics : Laravel
Written on December 25, 2023

In Laravel, contracts are a set of interfaces that define the core services provided by the framework. Contracts are essentially a set of interfaces that dictate the methods a service provider must implement. They provide a consistent and standardized way to interact with various components in Laravel, promoting flexibility and interchangeability.

1. Understanding Contracts:

Contracts define a set of methods that must be implemented by any class that binds to the contract. They serve as a form of documentation and a contract between the framework and the implementation.

2. Common Laravel Contracts:

Laravel includes many built-in contracts for various services. Some common ones include:

- **Illuminate\Contracts\Cache\Factory:**

  Methods for working with the cache.

- **Illuminate\Contracts\Queue\Queue:**

  Methods for interacting with the queue system.

- **Illuminate\Contracts\Routing\ResponseFactory:**

  Methods for creating HTTP responses.

- **Illuminate\Contracts\View\Factory:**

  Methods for working with views.

3. Using Contracts:

When you type-hint a parameter with a contract, Laravel will automatically inject the appropriate implementation of that contract.

```
use Illuminate\Contracts\Cache\Factory as CacheFactory;

class SomeClass
{
    protected $cache;

    public function __construct(CacheFactory $cache)
    {
        $this->cache = $cache;
    }

    public function someMethod()
    {
        $value = $this->cache->get('key');
        // ...
    }
}
```

4. Implementing a Custom Contract:

You can create your own contracts for custom services in your application.

Create a contract:

```
// app/Contracts/MyServiceContract.php

namespace App\Contracts;

interface MyServiceContract
{
   public function doSomething();
}
```

Implement the contract:

```
// app/Services/MyService.php

namespace App\Services;

use App\Contracts\MyServiceContract;

class MyService implements MyServiceContract
{
    public function doSomething()
    {
        return 'Something done!';
    }
}
```

5. Binding a Contract to an Implementation:

In a service provider or the `AppServiceProvider`, you can bind a contract to an implementation in the service container.

```
// app/Providers/AppServiceProvider.php

namespace App\Providers;

use App\Contracts\MyServiceContract;
use App\Services\MyService;
use Illuminate\Support\ServiceProvider;

class AppServiceProvider extends ServiceProvider
{
   public function register()
   {
      $this->app->bind(MyServiceContract::class, MyService::class);
   }
}
```

6. Using the Contract:

Now, you can use the contract in your application.

```
use App\Contracts\MyServiceContract;

class SomeClass
{
    protected $myService;

    public function __construct(MyServiceContract $myService)
    {
        $this->myService = $myService;
    }

    public function someMethod()
    {
        $result = $this->myService->doSomething();
        // ...
    }
}
```

7. Advantages of Contracts:

- **Abstraction:** Contracts allow you to abstract away the implementation details of a service,

making your code more flexible and testable.

- **Interchangeability:** By using contracts, you can easily switch implementations of a service without affecting the rest of your code.

- **Documentation:** Contracts serve as documentation, specifying the methods that must be implemented for a particular service.

8. Leveraging Existing Contracts:

Many Laravel features, such as middleware, can be customized by implementing existing contracts. For example, creating a custom middleware involves implementing the `Illuminate\Contracts\Http\Kernel` contract.