ARYATECHNO

Laravel - Facades

Topics : Laravel
Written on December 25, 2023

In Laravel, facades provide a convenient and expressive way to interact with Laravel services. Facades are a static interface to classes available in the service container. Laravel comes with many built-in facades, and you can create custom facades for your application.

1. Understanding Facades:

Facades provide a "static" interface to classes bound in the service container. They offer a concise syntax for accessing Laravel services without injecting them into your classes.

2. Built-in Facades:

Laravel provides several built-in facades that give you easy access to various features. Some common ones include:

- **Cache:**

```
use Illuminate\Support\Facades\Cache;

$value = Cache::get('key');
```

- **Config:**

```
use Illuminate\Support\Facades\Config;

$value = Config::get('app.timezone');
```

- **Session:**

```
use Illuminate\Support\Facades\Session;

$value = Session::get('key');
```

- **URL:**

```
    use Illuminate\Support\Facades\URL;

    $url = URL::to('path');
```

3. Creating Custom Facades:

You can create your own facades for custom classes. To create a custom facade, you need to create a new class and a facade class.

Create a class:

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

namespace App\Services;

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

Create a facade:

```
// app/Facades/MyServiceFacade.php

namespace App\Facades;

use Illuminate\Support\Facades\Facade;

class MyServiceFacade extends Facade
{
    protected static function getFacadeAccessor()
    {
        return 'my-service';
    }
}
```

Register the service in the service container:

In a service provider or the `AppServiceProvider`, you can bind the service to the container.

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

namespace App\Providers;

use App\Services\MyService;
use Illuminate\Support\ServiceProvider;
```

```
class AppServiceProvider extends ServiceProvider
{
    public function register()
    {
        $this->app->bind('my-service', function () {
            return new MyService();
        });
    }
}
```

4. Using Custom Facades:

Now, you can use your custom facade in your application:

```
use App\Facades\MyServiceFacade;

$result = MyServiceFacade::doSomething();
```

5. Facades vs Dependency Injection:

Facades provide a concise syntax for accessing services, but they are essentially a form of static proxy. While convenient, they can make code less testable. Dependency injection is often preferred for better testability and explicit dependencies.

6. Dynamic Properties:

Some facades, like the `Request` facade, allow you to access methods as if they were properties for a more expressive syntax:

```
use Illuminate\Support\Facades\Request;

$path = Request::path();
// or
$path = Request::path;
```

7. Facade Aliases:

You can add an alias for a facade in the `config/app.php` file:

```
'aliases' => [
    // ...
    'MyService' => App\Facades\MyServiceFacade::class,
],
```

This allows you to use a shorter alias in your code:

```
use MyService;

$result = MyService::doSomething();
```