02_Laravel_Service_Providers.md

Create a Custom Service Provider to Register a Singleton Service and Defer Its Loading

Objective

Learn how to:

1. **Create a custom Service Provider** in Laravel.
2. **Register a singleton service** (only one instance throughout the request lifecycle).
3. **Defer loading** to improve performance.

---

1. Create a Custom Service

First, define a service class (e.g., `PaymentGateway`).

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

class PaymentGateway
{
    public function charge(float $amount): bool
    {
        // Simulate payment processing
        return true;
    }
}
```

PROF

2. Generate a Custom Service Provider

Run:

```
php artisan make:provider PaymentServiceProvider
```

This creates:

```php
// app/Providers/PaymentServiceProvider.php
namespace App\Providers;

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

```
class PaymentServiceProvider extends ServiceProvider
{
    public function register()
    {
        // Register as a singleton (single instance)
        $this->app->singleton(PaymentGateway::class, function ($app) {
            return new PaymentGateway();
        });
    }

    // Optional: Defer loading until needed
    public function provides()
    {
        return [PaymentGateway::class];
    }
}
```

- `singleton()`: Ensures only one instance is created per request.
- `provides()`: Defines which services are deferred.

---

3. Register the Service Provider

Add to `config/app.php`:

```
'providers' => [
    // ...
    App\Providers\PaymentServiceProvider::class,
],
```

PROF

4. Defer Loading (Optional)

To **delay loading** until the service is actually used:

```
// app/Providers/PaymentServiceProvider.php
class PaymentServiceProvider extends ServiceProvider
{
    protected $defer = true; // Defer loading

    public function provides()
    {
        return [PaymentGateway::class];
    }
}
```

- `$defer = true`: Only loads when explicitly requested.

---

5. Use the Singleton Service

Now, inject `PaymentGateway` anywhere (controllers, jobs, etc.):

Method 1: Dependency Injection

```php
// app/Http/Controllers/PaymentController.php
use App\Services\PaymentGateway;

public function pay(PaymentGateway $paymentGateway)
{
    $success = $paymentGateway->charge(100.00);
    return $success ? "Payment successful!" : "Payment failed!";
}
```

Method 2: Facade (Optional)

Create a Facade for easier access:

```php
// app/Facades/PaymentFacade.php
namespace App\Facades;

use Illuminate\Support\Facades\Facade;

class Payment extends Facade
{
    protected static function getFacadeAccessor()
    {
        return \App\Services\PaymentGateway::class;
    }
}
```

PROF

Then use:

```php
use App\Facades\Payment;

Payment::charge(100.00);
```

---

6. Testing

```
php artisan tinker
>>> app('App\Services\PaymentGateway')->charge(50.00); // Should return
`true`
```

Key Takeaways

✅ **Singleton Binding**: Ensures **one instance** per request.
✅ **Deferred Loading**: Optimizes performance by loading **only when needed**.
✅ **Dependency Injection**: Clean, testable way to use services.

Next Step: Try **Repository Pattern** for cleaner database interactions! 🚀
