Twitter Thread by Mohamad Fazel

Mohamad Fazel
@mdhesari

Have you ever worked with pipelines in Laravel?!

Probably you have heard it while having an interview?

It has lots of use cases and one of its usage is middleware implementation in its kernel.

I'm going to write about pipelines and its design pattern.

thread ■

1/8

First of all Pipelines in Laravel are implemented based on chain of responsibility design pattern.

In chain of responsibility which is a behavioral design pattern we pass data to receivers and this receivers implement a specific interface that has a handler method.

2/8

Handler methods may deal with the passed argument and pass it to the next handler.

read more about the design pattern here :

https://t.co/XrJKXecyTv

Let's go for the main part :

3/8

Imagine we want to implement a twitter bot that gets some tweets around the trends and filter the words that we don't wanna show finally send as a new tweet from our bot.

About the filtering part, if it was me, I would implement it with pipelines.

Why?!

4/8

Currently I know exactly what words should be filtered but what about the future?! What if I want to add more filters, I will have to modify the source code and add more switch, if else statements, etc.

I want my code to be robust and maintainable keeping SOLID principles.

5/8

So I'm gonna use pipeline like this (picture) :

First I will have my initial filters and also give the ability to myself or other developers to add more filters without even seeing the code.

See the picture and lets go deeper and check the main interface and pipelines ■

```php
public function filterTweet(string $content) {

        $pipelines = [
            \App\Filters\RemoveBadWordsFilter::class,
            \App\Filters\RemoveUrlsFilter::class,
        ];

        array_merge($pipelines, config('filters.add_pipelines'));

        $content = app(\Illuminate\Contracts\Pipeline\Pipeline::class) // resolve Pipelines contract
        ->send($content) // content will be passed to each receiver
        ->through($pipelines) // receivers
        ->via('filter') // using filter method
        ->then(function($content) {

            // final filtered content
            return $content;
        });

        return $content;
 }
```

6/8

I have defined a contract for all of my filters, every new filter and existing filters must implement this interface and implement the filter method

As you can see in the previous picture we gave our method name "filter" passed to via()

Nothing more let's go for filters

```php
namespace App\Filters\Contract;

interface Filterable {

    /**
     * Filter specific text
     *
     * @param  string $content
     * @param  \Closure $next
     * @return mixed
     */
    public function filter(string $content, \Closure $next);
}
```

7/8

This is one of the filters we have implemented as you can see everything is clear filter method gets an argument called $content which is the text we wanna filter and when we modify the content and remove bad words it calls the next receiver from pipelines filters we defined

```php
<?php

namespace App\Filters;

use App\Filters\Contract\Filterable;

class RemoveBadWordsFilter implements Filterable {

    /**
     * Filter specific text
     *
     * @param  string $content
     * @param  \Closure $next
     * @return mixed
     */
    public function filter(string $content, \Closure $next)
    {

        // filter bad words

        return $next($content);
    }
}
```

8/8

That's it all.
This is the pipeline design pattern we have in Laravel.

As you can see there are a lot of advantages using this design pattern having SOLID standards.

But sometimes it's better to keep things simple when our program is not that big and you are alone...