At the beginning of my learning of web development, I started with [PHP](<https://secure.php.net/>). Till date it’s the most used language for serving the web. To start with the basics & OOP concepts of [PHP](<https://secure.php.net/>), it didn’t take much time for me as I had a solid background of standard Java. After having basic idea (syntax & OOP implementation) of PHP I started learning [Codeigniter](<https://www.codeigniter.com/>). It’s really simple, fast and easy to use, mostly like raw/necked PHP with some handy libraries. Then I got introduced with [Laravel](<https://laravel.com/>). At some point of my career I also worked with [Symfony](<https://symfony.com/>) framework. Besides, I recently I started learning [Django](<https://www.djangoproject.com/>). Through my journey with all these frameworks I have learnt every tools has its own advantage and limitation as they are made to serve different purposes, to solve different problems and they are built with different things (e.g. [Laravel](<https://laravel.com/>). made with [PHP](<https://secure.php.net/>) and [Django](<https://www.djangoproject.com/>). made with [Python](<https://www.python.org/>)).

Today, I am writing about the good things I found in Laravel throughout my journey with all the web frameworks I mentioned above. So, this article is like a history of my thinking while I was learning Laravel.

## Composer, a modern way to resolve PHP dependencies

I never used [composer](<https://getcomposer.org/>) before started learning [Laravel](<https://laravel.com/>). While working with Codeigniter we used to download the zip form the official site of Codeigniter as an installation process which was very straightforward to me that time. So, when I made my mind of learning Laravel, I went to the official website of Laravel and was searching for the download link somewhere in the homepage but unfortunately there was none! I burst out with surprise. What! How to can I use this framework? I forced to read the documentation or tutorial before downloading anything and I found that I need to use a piece of application called composer. I had no idea about it. I installed composer in my windows OS though as a minimalistic I didn’t like the idea of installing another application to install a framework. Fortunately, this was my first step of using composer, I never looked back. Now, I know this is an awesome tool of managing PHP dependencies. [Javascript](<https://www.javascript.com/>) has [npm](<https://www.npmjs.com/>), [yarn](<https://yarnpkg.com/en/>) etc. to do this same thing, following them Python is introducing [Pipenv](<https://pipenv.readthedocs.io/en/latest/>) to do same thing. Today, if I start a new PHP project even without a framework, the first command I am going to run is `composer init`. I cannot imagine a PHP project today without composer managing its dependencies. This is the modern way of resolving dependencies. Yes, I know composer has nothing with Laravel, it’s a tool for PHP. Many other frameworks of PHP (e.g. Symfony) use composer to install themselves. But to me, Laravel is the first framework that introduced (may be forcefully) me with this awesome tool called composer that I never knew of.

## Command Line Interface (CLI) & Artisan

Yeah, I had interest on CLI. Though I was using windows that time, I used to search for way to delete a file, to run a song with VLC player, to format my flash drive, to shut down my computer, to see the contents of the directory using command prompt. These were all about fantasy, nothing serious. Then once again Laravel forced me to use CLI. Laravel has its own CLI to do some cool staffs using its command line interface called [Artisan](<https://laravel.com/docs/5.7/artisan>).

To make a controller

```

php artisan make:controller <Controller Name>

```

To make a model

```

php artisan make:model <Model Name>

```

To make a migration!

```

php artisan make:migration <Migration Name>

```

At the beginning it was scary to me, because I was more used to write such files manually. Yes, in Laravel I can do these things manually too. It’s not mandatory to use CLI but it’s easier using CLI. I decided to make CLI my friend, instead of considering it a hacky and complex way of doing things. All modern frameworks like Symfony, Django etc. have their own command line interface and yet Laravel was the first framework to a beginner to introduce the scary CLI as a friend who can help to do the job in an easier way! Once, I saw a question in Stackoverflow, asking “why do I need to use Artisan commands work with Laravel?”. I answered “I believe you are new to Laravel and might have come from Codeigniter. Otherwise, you would have asked why Codeigniter doesn’t have a tool like Artisan?”. I answered this way because this same thing happened to me. Initially I was asking myself why do I need to use artisan, now I ask why every framework doesn’t have a CLI like Artisan.

## Eloquent, clean & simple

When we start to learn a web framework, the basic things firstly are the CRUD operations. I did the same thing. In Laravel, to make a model with migration:

```

php artisan make:model Post

```

To find the post with its primary key(ID)

```php

$post = Post::find($id);

```

As simple as that! Isn’t it clean? I was impressed. This is called [Eloquent](<https://laravel.com/docs/5.7/eloquent>), the [ORM](<https://en.wikipedia.org/wiki/Object-relational_mapping>) for Laravel. Well, as a beginner I didn’t know what ORM was, but It seemed the coolest feature of Laravel to me. While working with Symfony I found [Doctrine](<https://symfony.com/doc/current/doctrine.html>).

To do the same this as above using Doctrine we need to write as below:

```php

$post = $this->getDoctrine()

->getRepository(Post::class)

->find($id);

```

I know Doctrine has its own advantage of writing in such way but still Eloquent is the simplest and cleanest to me.

Besides, Eloquent has its own cooler way to define relationship among tables along with all the features of a modern ORM. Yeah ORM has its own limitation, it is beautiful and this beauty costs speed. Still if we need speed then we have the option to run raw query. But if you ask me to choose an ORM I will always go for Eloquent. This is the simplest to me till now.

## IoC Service Container

Simply speaking in OOP, [Inversion of control(IoC)](<https://en.wikipedia.org/wiki/Inversion_of_control>) is an awesome and classic way to remove dependency from a specific class to it’s higher order(interface or abstract class). Using IoC we can alter the entire functionality of a class just by inject a different instance of a different class which is a child of the same abstract class. Laravel has nice way to manage such things. Using [Laravel’s service container](<https://laravel.com/docs/5.7/container>) we can define a implementation for an abstract class. At any point we can just alter the implementation to change the functionality. Let’s consider we have three class a bellow:

```php

abstract class Animal

{

abstract public function speak();

}

class Dog extends Animal

{

public function speak()

{

return “Speaking like a dog”;

}

}

class Cat extends Animal

{

public function speak()

{

return “Speaking like a cat”;

}

}

```

For example, we can bind a class like this in the service provider of Laravel:

```php

$this->app->bind(Animal::class, Dog::class);

```

Here we have bound a `Dog` implementation for `Animal`. Now, in a controller:

```php

class AnimalController extends Controller

{

protected $animal;

public function \_\_construct(Animal $animal)

{

$this->animal = $animal;

}

public function speak()

{

return $this-animal->speak();

}

}

```

Here, the `speak()` method of our `AnimalController` class will return “Speaking like a dog”

Now, if we want to change the behaviour of Animal to Cat in our `AnimalController` what we do need to do? Interesting nothing in the `AminalController`. That’s why we call it Inversion of Control. We just change:

```php

$this->app->bind(Animal::class, Dog::class);

```

To this line:

```php

$this->app->bind(Animal::class, Cat::class);

```

That’s it! Now, the `speak()` method of our `AnimalController` class will return “Speaking like a cat”. Isn’t it nice and clean? Besides, it follow [PSR-11](<https://www.php-fig.org/psr/psr-11/>).

## Blade, templating engine

Blade is a very powerful templating engine that will empower you with its elegant way of extending and separating the html elements. To me, once again it’s just clean and nice. Besides, it’s directives are awesome! Check out [this](<https://laravel.com/docs/5.7/blade>) page to see. Cleaner than any templating engine I have used till now, period.

## Modern PHP & JS libraries included out of the box

This is noticeable that Laravel is nicely keeping itself up to date with all the modern tools and technologies under its belt. It’s following the [PSRs](<https://www.php-fig.org/psr/>). It includes libraries like [Carbon](<https://carbon.nesbot.com/>), [Guzzle](<http://docs.guzzlephp.org>), [Monolog](<https://seldaek.github.io/monolog/>), [PHPUnit](<https://phpunit.de/index.html>), [Faker](<https://github.com/fzaninotto/Faker>) out of the box. Its ships with [Bootstrap](<https://getbootstrap.com/>), [JQuery](<https://jquery.com/>), [VueJS](<https://vuejs.org/>) with all required setup(npm, [webpack](https://webpack.js.org/)) out of the box!

## Finally

I have learnt about composer (and other dependency management tool), Command Line Interface, IoC through learning Laravel. Now, even if I work with Django, Symfony or any other framework I still miss Eloquent and Blade etc though these frameworks have their own advantages. People from other language used to criticize PHP as a language, but PHP has changed itself a lot though it still has some ugliness in its core. But Laravel is a framework that is actively trying to make a better and cleaner way of writing code using PHP and it’s developing itself every day.