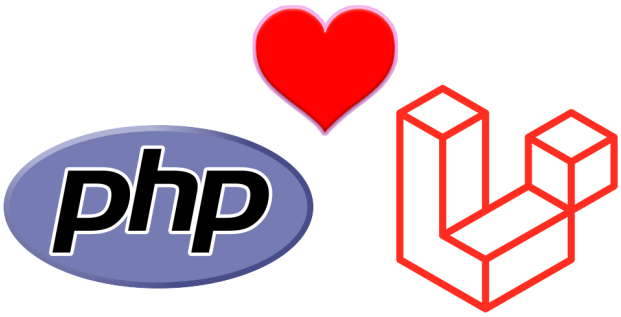
When PHP Meets Laravel



A Story of Love

The Flexible Modularity of Eloquent Cohesion

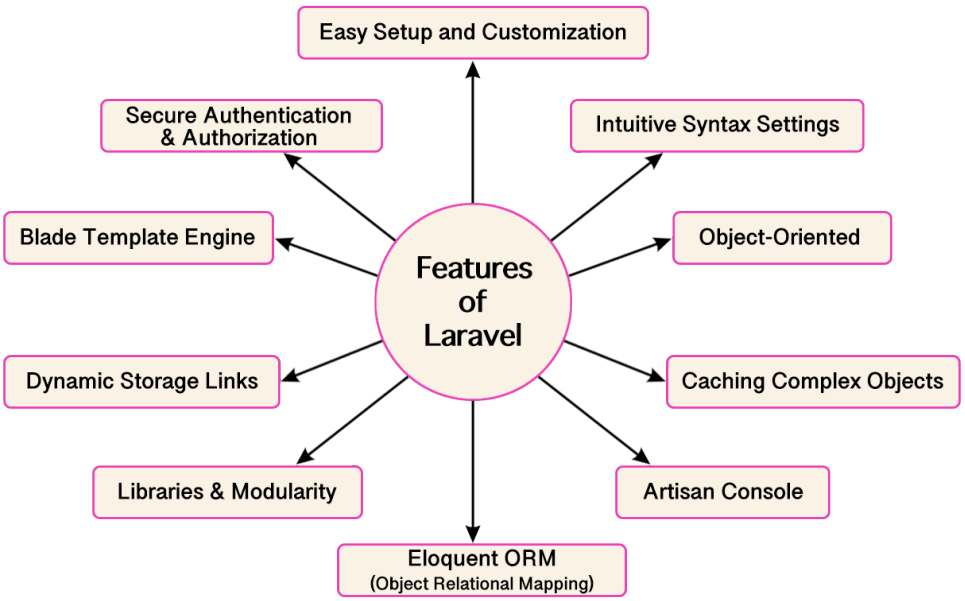
By Chad Jordan June 13th 2014

Introduction

In this guide you will learn:

1. The fundamental elements and functional advantages of using the Laravel framework
2. How to install and configure PHP, Composer, and Laravel 4.2 in a macOS environment
3. Understanding the improved Model-View-Controller architecture
4. An overall side-by-side comparison of using the Laravel framework versus core PHP

In June of 2011 Taylor Otwell created a free, and open-source web framework for PHP called Laravel. The intention of the technology was to improve the development of web applications following the model-view-controller architectural pattern based on the earlier Symfony framework released in 2005. Why does this matter? Because its architecture and **expressive, beautiful syntax** make Laravel **object-oriented**. Laravel provides an out-of-the-box configuration so that your applications can be equipped with a **secure authentication** and **authorization** system. Laravel also provides a *unified API* for various **caching systems**. The **cache configuration** is located at app/config/cache.php. In the *cache.php* file, you may specify which cache driver you would like used by default throughout your application. Laravel supports popular caching backends like **Memcached** and **Redis** out of the box. **Artisan** is the name of Laravel's own command-line interface. When it comes to publishing package assets, developing new models, migrations, and controllers, managing database migration, and doing a lot of other stuff, Artisan makes it easy. By developing new custom commands, one can even enhance the capabilities of Artisan. **ORM** stands for *Object Relational Mapping*. **Eloquent ORM** is an advanced PHP implementation that is used for active record pattern. It provides at the same time internal methods for enforcing constraints with the relational between database objects. Eloquent ORM presents database tables with object instances that are tied to single table rows. One method of using templates in Laravel is via **controller** layouts. By specifying the *layout* property on the controller, the view specified will be created for you and will be the assumed response that should be returned from actions. **Cashier** is new in Laravel 4.2 and is used in subscription-based services that require periodic payments. Moreover, it has substantial uses in handling coupons and generation invoices as well. **Blade** is a simple, yet powerful templating engine provided with Laravel. Unlike controller layouts, Blade is driven by *template inheritance* and sections. All Blade templates should use the .blade.php extension. Laravel can be termed the world's best PHP-supported framework for providing extraordinary configuration for **Authentication** and **Authorization**. It offers a



number of artisan commands at your disposal to make your applications highly authenticated and secure. These are only several features to introduce some of the advantages of using Laravel 4, but one of the other advantages of using Laravel 4 is how it is built around the architecture of events. Events make it really easy for developers to hook on to, or extend the underlying architecture of the framework in a clean and maintainable way. It also makes it really easy for us to create events for our own applications so that we create software that is easier to extend and maintain for others. Are you new to events in programming? If so, an event is just an occurrence of a particular instance at a particular moment in time. For example, if you are creating a social application, an event you might be interested in could be whenever a new user signs up. When a new user signs up using your application, you will probably have a list of things that need to occur. For example, you will want to send the user an email, subscribe them to your newsletter or add them to a queue to suggest the next steps for using your application. By defining the '*user create*' event, you can set all of these tasks in motion whenever a new user signs up for your application. So why use events? The beautiful thing about using an event-based architecture is, that you maintain your separation of concerns. When an event is triggered it does not need to know anything about what will happen as a consequence of firing the event.

Throughout this guide, I'll be providing examples, explanations, and demonstrations of how to understand the overall instance of Laravel and why it is the best framework for PHP. In this guide, I'll be using a terminal environment in macOS X 10.9.5.

Installing PHP, Composer, and Laravel

Since Laravel is a more detailed framework that manages dependencies, this requires additional tools for installation alongside Laravel. **Laravel 4.2 requires PHP 5.4 or greater**. This upgraded PHP requirement allows the developer to use new PHP features such as traits to provide more expressive interfaces for tools like Laravel Cashier. PHP 5.4 also brings significant speed and performance improvements over PHP 5.3. If you do not have version 5.4, you will need to install it. In a terminal window, run the following command:

```
sudo curl -s http://php-osx.liip.ch/install.sh | bash -s 5.4
```

enter your password, hit enter, and simply wait for the installation process to complete.

Note: *this process can take up to 5 to 10 minutes.*

Once it's done, verify in your terminal that it's there:

```
/usr/local/php5-5.4.29-20140529-223618/
```

Once you've verified that the *libphp5.so* config file is inside the directory, we have to set up the built-in apache in order to use PHP 5.4 so now we open the config with the following command:

```
sudo nano /etc/apache2/httpd.conf
```

Once opened, locate the *LoadModule* for PHP, it's usually in the end of the list of LoadModules. Comment the old PHP module out by putting a hash sign (#) in front of it and put the new module on a new line at the bottom of the *LoadModule* list, in my case it looks like this

```
LoadModule php5_module /usr/local/php5-5.4.29-20140529-223618/libphp5.so
```

Save and exit *(CTRL + o + enter) (CTRL + x)*, and then restart apache with the following command:

```
sudo apachectl restart
```

Next, we have to set the PHP library path so the new PHP binaries will default on command line execution. Do this by running the following terminal command:

```
sudo nano /etc/paths
```

This is what my file looks like, and your file should be similar to this:

```
/usr/local/php5-5.4.29-20140529-223618/bin/
/usr/bin
/bin
/usr/sbin
/sbin
/usr/local/bin
```

If not, edit accordingly based on your folder names and PHP version, save and exit just as we did above for the LoadModule config, then you should probably restart bash at this point. You can verify to make sure PHP is running normally by creating and opening a file *(this is the default location of the webserver in OS X)*. Run the following command:

```
sudo nano /Library/WebServer/Documents/php.php
```

Once in, enter the following:

```
<?php
phpinfo();
?>
```

Save and close, go to your file by entering something like *http://localhost/php.php* and you should see a display of the current version of PHP in your browser similar to this:

PHP Version 5.4.29

System	Linux gator3118.hostgator.com 4.19.150-76.ELK.el6.x86_64 #1 SMP Wed Oct 7 01:34:10 CDT 2014 x86_64
Build Date	May 29 2014 00:15:14
Configure Command	'./configure' '--build=x86_64-redhat-linux-gnu' '--host=x86_64-redhat-linux-gnu' '--target=x86_64-redhat-linux-gnu' '--program-prefix=' '--prefix=/opt/cpanel/ea-php54/root/usr' '--exec-prefix=/opt/cpanel/ea-php54/root/usr' '--bindir=/opt/cpanel/ea-php54/root/usr/bin' '--sbindir=/opt/cpanel/ea-php54/root/usr/sbin' '--sysconfdir=/opt/cpanel/ea-

Now that PHP 5.4 has officially been installed and configured, we can now focus our attention on Laravel, but before Laravel can be installed, you need to install **Composer**. Laravel utilizes Composer to manage its dependencies. First, download a copy of the *composer.phar* by entering the following into your browser address bar: *https://getcomposer.org/composer.phar*

When you enter that URL, the *composer.phar* file should automatically download to your computer. Assuming that the file was downloaded to your Downloads folder and not your Desktop, we want to execute a command to copy it to the appropriate location in your system files. You can either keep it in your local project directory or move it to *usr/local/bin (on macOS, or Linux)* to use it globally from your system. I'm moving mine to the root system folder so I can use it globally. We also need to write **chmod** permissions to the directory that we're wanting to execute from, so run the following terminal commands:

```
mv ~/Downloads/composer.phar /usr/local/bin
sudo chmod +x /usr/local/bin
```

Now verify the .phar file is in *usr/local/bin*

```
cjordan@Chads-MacBook-Pro:~$ cd /usr/local/bin
cjordan@Chads-MacBook-Pro:/usr/local/bin$ ls
brew        composer.phar
cjordan@Chads-MacBook-Pro:/usr/local/bin$
```

Now that we have the *.phar* file in place, let's verify the file by running Composer from this same directory. Run the following command: *php composer.phar*
and you should see something similar to this in your terminal window:

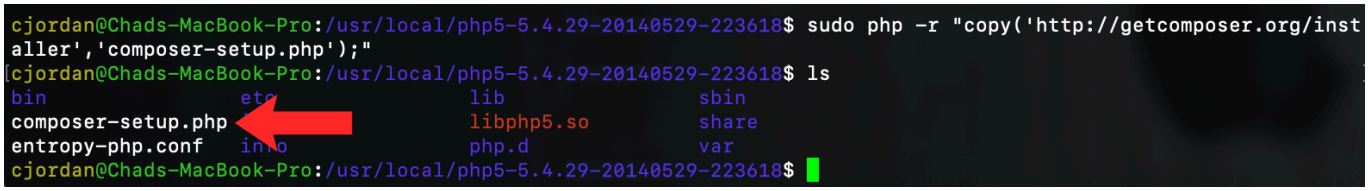
```
Cellar alias               etc                     packager                share
Frameworks                 include                 php5                    var
cjordan@Chads-MacBook-Pro:/usr/local$ cd bin
cjordan@Chads-MacBook-Pro:/usr/local/bin$ ls
brew          composer.phar
cjordan@Chads-MacBook-Pro:/usr/local/bin$ php composer.phar
   ______
  / ____/___  ____ ___  ____  ____  ________  _____
 / /   / __ \/ __ `__ \/ __ \/ __ \/ ___/ _ \/ ___/
/ /___/ /_/ / / / / / / /_/ / /_/ (__  )  __/ /
\____/\____/_/ /_/ /_/ .___/\____/____/\___/_/
                    /_/
Composer version 1.0.0-alpha8 2014-01-06 18:39:59

Usage:
  [options] command [arguments]

Options:
  --help           -h Display this help message.
  --quiet          -q Do not output any message.
  --verbose        -v|vv|vvv Increase the verbosity of messages: 1 for normal output, 2 for more verbose output
and 3 for debug
  --version        -V Display this application version.
  --ansi              Force ANSI output.
  --no-ansi           Disable ANSI output.
  --no-interaction -n Do not ask any interactive question.
  --profile           Display timing and memory usage information
  --working-dir    -d If specified, use the given directory as working directory.

Available commands:
  about            Short information about Composer
  archive          Create an archive of this composer package
  config           Set config options
  create-project   Create new project from a package into given directory.
  depends          Shows which packages depend on the given package
  diagnose         Diagnoses the system to identify common errors.
  dump-autoload    Dumps the autoloader
```

The list provides the possible commands that can be executed in order to run Composer. To get the setup file that we need to run, I'm choosing to navigate back to the php5-5.4 folder and place it in the folder with the rest of my php files.

The location shouldn't matter as long as it's within the system path, this is just my preference. Once in that directory, I run the following terminal command:

```
sudo php -r "copy('http://getcomposer.org/installer','composer-setup.php');"
```

This command gets the Composer installer and places it in the current directory.
Check to verify that the setup file is in the directory.



(If you wish to install locally)

You can install Composer to a specific directory by using the `--install-dir` option and additionally (re)name it as well using the `--filename option`. Run the following command:

```
sudo php composer-setup.php -install-dir=bin -filename=composer
```

followed by `php bin/composer` and this will run composer.

(If you wish to install globally)

```
mv composer.phar /usr/local/bin/composer
```

If you would like to install it only for your user and avoid requiring root permissions, you can use `~/.local/bin` instead. If you need to add sudo just run it again as a superuser. Some Mac OS's are not equipped with the /usr directory by default which means you may receive the error, ***"/usr/local/bin/composer: No such file or directory"*** and if this is the case, you will have to create the directory manually before proceeding:

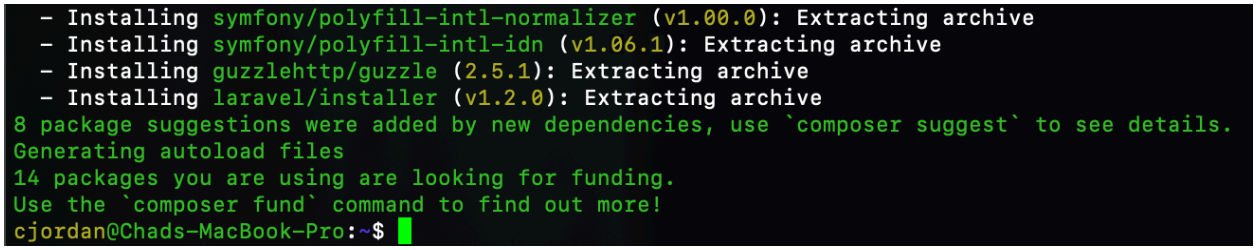
```
mkdir -p /usr/local/bin
```

Now run `composer` in order to run Composer instead of `php composer.phar`

With Composer fully installed and configured, we can now install Laravel. Composer is required to do this. Begin by running the following terminal command:

```
composer global require "laravel/installer=~1.1"
```

The terminal will start installing the list of dependencies. Once it's completed and reached the bottom, it will look something like this:



Note: Make sure to place the `~/.composer/vendor/bin` directory in your PATH so the Laravel executable is found when you run the ***Laravel*** command in your terminal. We can easily add it to the `etc/paths` just as I did earlier for the PHP binary executable paths. Run command:

```
sudo nano /etc/paths
```

You can add it to the bottom of the list as I have done here:

```
/usr/local/php5-5.4.29-20140529-223618/bin/
/usr/bin
/bin
/usr/sbin
/sbin
/usr/local/bin
~/.composer/vendor/bin
```

Save and exit *(CTRL + o + enter) (CTRL + x)*, and then restart apache

```
sudo apachectl restart
```

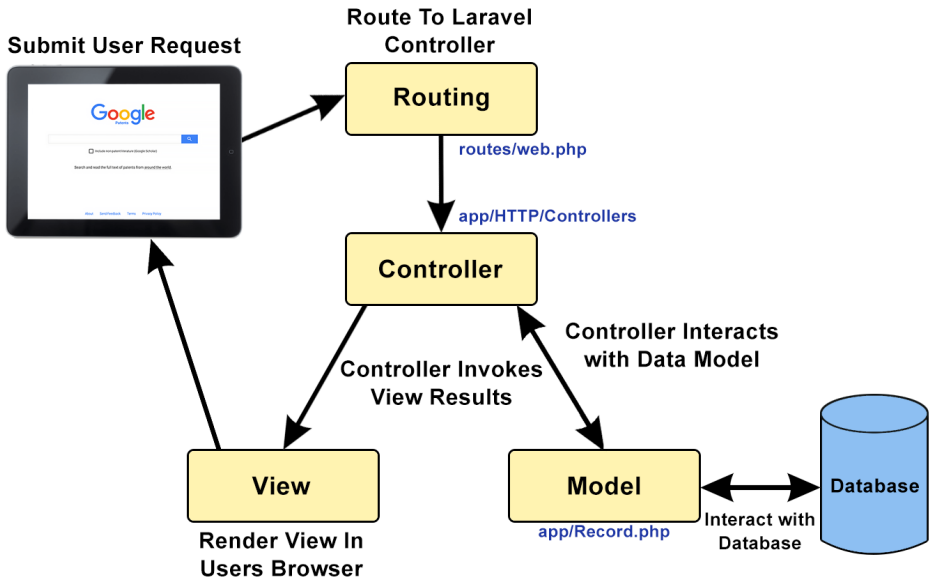
Once installed, the simple ***laravel new*** command will create a fresh Laravel installation in the directory you specify. For example, *laravel new form* would create a directory named *form* containing a fresh Laravel installation with all dependencies installed! This method of installation is much faster than installing via Composer and twice as efficient.
To install version 4.2 of Laravel, we can install it via Composer with the following command:

```
composer create project laravel/laravel {directory} 4.2 --prefer-dist
```

With that command run, this essentially completes the out-of-the-box installation and configuration. Because we used Composer for this process, our configuration key has already been generated. This is another good reason to go this route rather than manually downloading from the Laravel website, then extracting and running the file independently of Composer. The Composer route ensures more security.

Laravel's MVC Architecture

MVC *(model-view-controller)* architecture is one of the best, and Laravel supports it. MVC architecture is known to improve the performance of any web application by high leaps. It is also easy to understand because of its clear and brief documentation process. To understand how Laravel use MVC first we have to understand the MVC. Essentially PHP MVC is a design pattern that separates the data and business logic from the presentation. **Model** is basically the collection of application data and business logic. It can be used to perform many operations like data validations, process the data, and store data in the database. Data can be any kind like and static file, database, XML data, or any other kind of valid resource. The **controller** is the part that deals



with the user's request, It accepts the user's request and responds as per the user's request. For example, if the user requested through URL like /index.php?products=list than the controller will load the list of product data from the model and output to the user. The **view** is just for presenting the data to the user. This is the bridge for the interaction of users to our server. This is basically written in HTML form. At this stage we want to understand how Laravel uses the MVC architecture. In Laravel the MVC uses the following manners:

1) *Create a project in Laravel.*

```
laravel new product-store
```

2) *Create a model by using the following command.*

```
php artisan make:model Product
```

3) *Connect the database using .env file then create a migration for storing the list of products.*

```
php artisan make:migration create_products_table
```

4) *After created migration successfully just migrate the table into the database.*

```
php artisan migrate
```

5) *Create a controller for handling the request.*

```
php artisan make:controller ProductController -r
```

6) *Create a route to get the request and send it to the controller*

```
use App\Http\Controllers\UserController;
Route::get('/user/{id}', [UserController::class, 'show']);
```

7) *Create a blade template in view by which the user would request*

```
{{request()->query('blade')}}
```

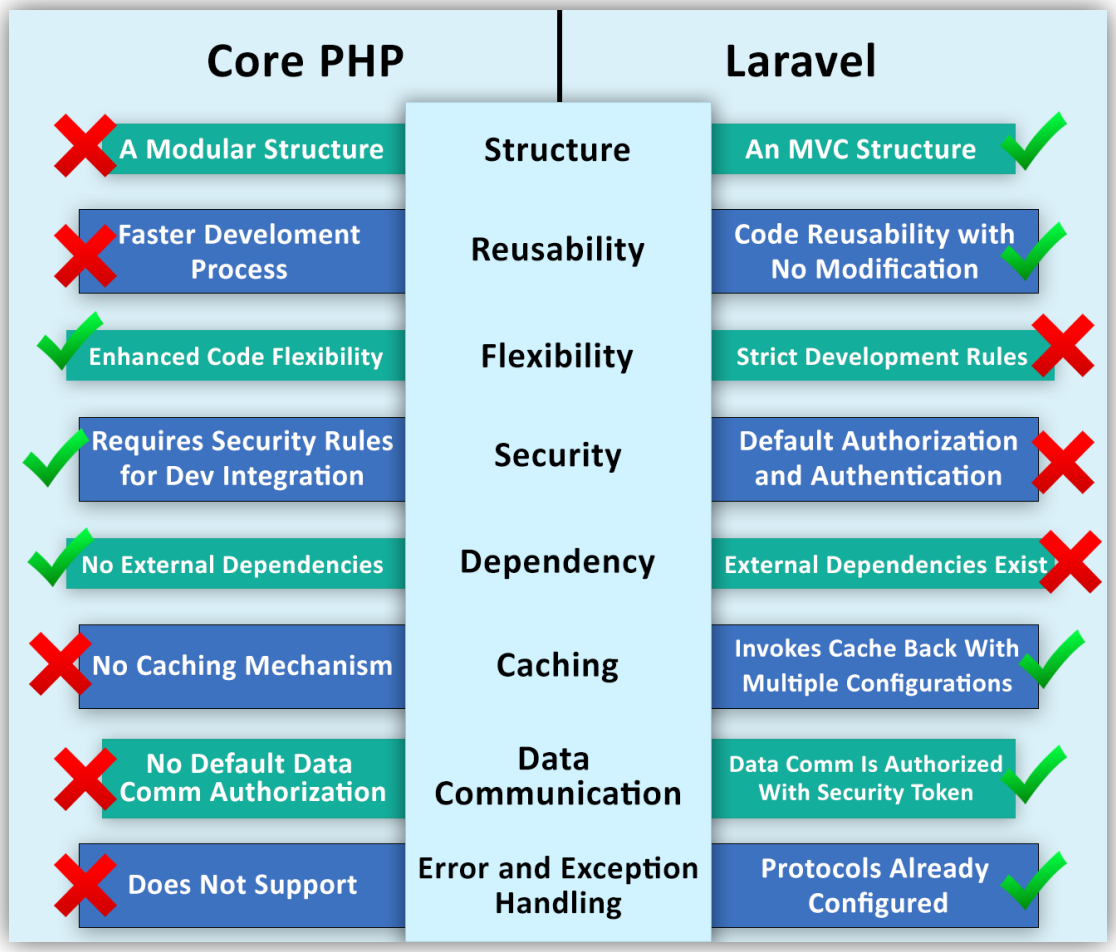
This process is essentially the behavior pattern of how the MVC is utilized through Laravel. The MVC pattern helps you break up the frontend and backend code into separate components. This way, it's much easier to manage and make changes to either side without them interfering with each other. Additional advantages of using the MVC are:

- Making model classes and code reusable without required modification
- It's a helpful design pattern when planning development
- Easier to maintain, or modify if desired
- Loosely coupled *(meaning each piece of the MVC acts independently of each other)*
- Extendable code with higher cohesiveness
- Very popular and widely used in web applications, therefore online support is higher
- Cross-platform Integration with Mail and Messaging Systems
- Fast Caching Integrations
- Supports AMI *(Asynchronous Method Invocation)*
- Ideal for Developing Large Size Web Application
- Easy for multiple developers to collaborate and work together
- Executing multiple view models

With these advantages in mind, the MVC architecture is a solid and elegant approach to building web apps. MVC frameworks that come packaged with the above-mentioned advantages, make it easy to implement in your projects. The adoption of the MVC architecture provides a lesser expense of time & money, and the ability to create multiple views makes it the best architecture for web app development. The MVC architecture tied into Laravel makes Laravel the best framework to use with PHP.

Core PHP vs Laravel

Traditional PHP or ***Core PHP***, is the foundational programming language of all PHP frameworks, including Laravel. Web application development through Core PHP can enable the creation of an exceptionally dynamic application that can push the limits of web development possibilities. On its own, PHP has remained a powerful programming language for server-side applications during the last two decades, but when we use frameworks with existing technologies, we harness the power of elegant design patterns to create more efficient applications. This is the intention of frameworks such as Laravel and it's important to distinguish the differences between the two technologies. This will help developers ascertain why they should use said technology. The following graphical chart is a side-by-side comparison to provide visual insight into supporting user decisions between core PHP and Laravel.



For these particular instances, we see that Laravel does excel further as an overall better option for meeting more efficient requirements while building web applications. For example,

1) We've learned that Laravel's MVC allows for larger amounts of content to be maintained a lot easier than traditional PHP.

2) While PHP has a faster development process, we understand that Laravel is more reusable and requires no modifications to the code. If there's one thing that PHP developers know it's that PHP is updated frequently with new changes, so this is another aspect of Laravel that can alleviate a lot of needless tasks, updating procedures, and unwanted redundancy to save on time and money.

3) On the other hand, it's actually PHP that has more flexibility since it's a programming language and Laravel is a framework. This is due to the strict rules that Laravel adheres to regardless of providing more options.

4) When it comes to security, Laravel comes with its own default authorization and authentication which is more convenient, but it's not as secure as being able to write your own security routines in PHP. PHP leaves the door of security wide open for the developer to customize the security as they see fit.

5) As we learned from my earlier installation process of Laravel, it does rely on dependencies whereas the advantage of PHP requires no external dependencies in order to use it.

6) We know that caching is a technique that stores something in memory that is being used frequently to provide better performance. Laravel takes the win on this since it invokes cache backs with multiple configurations, while the core of PHP has no caching mechanism.

7) When it comes to communicating with the MVC, the Controller is responsible for determining which View to display in response to any action including when the application loads. This differs from MVP where actions route through the *View* to the *Presenter*. In MVC, every action in the *View* correlates with a call to a *Controller* along with an action. In the web, each action invokes a call to a URL on the other side of which there is a *Controller* who responds. Once that *Controller* has completed its processing, it will return the correct *View*. In Laravel, data communication is authorized via a security token that prevents posting data from other domains, while Core PHP does not have default data communication authorizations.

8) As for PHP exception handling, this is used to change the normal flow of the code execution if a specified error *(exceptional)* condition occurs. This condition is called an exception. Error and exception handling protocols are already configured in Laravel, while Core PHP does not support default error and exception handling protocols.

Conclusion

What is to be learned from the above comparison? In the end, even if a technology makes the process more efficient, and easier to use, the final decision is still subjective from user to user. This is one of the primary reasons why I recommend learning both Core PHP & Laravel, and then deciding for yourself. In my opinion, by looking at the capabilities of Laravel, I still stand firm that there hasn't been a better PHP framework than Laravel, and that includes Symmetry and CakePHP. Developers and software companies around the globe know Laravel for its outstanding compatibility and capabilities when it comes to features, and ease of access. I have personally used PHP numerous times throughout my educational and professional career to write server-side applications with much ease. As a developer who has used PHP on more than one occasion, I'm usually on the hunt for the next framework or piece of technology to enhance the development process. One thing that does seem to stand out for other developers is the dependencies that Laravel requires at installation. Depending on which method/s developers are using for their installation approach, there seems to be a consistency regarding time invested into unwanted configuration. This is based on personal discussions and online forums that I've come across over time.

My hope is that this guide has been helpful in assisting the learning process of using Laravel 4.2 with PHP 5.4. All diagrams & charts presented in this guide were created by Chad Jordan for learning purposes only. The terminal command images were generated from screenshots that I took throughout the installation process on my MacBook Pro. For any possible inquiries such as general questions regarding this guide or other professional inquiries please feel free to email me at cjordan@wondercreationstudios.com

Resources Used:

- Otwell, Taylor - *Laravel: From Apprentice to Artisan* – 2013
- Laravel.com
- W3schools.com