# Prerequisites

## Laravel

Laravel is a PHP framework that uses the MVC patten as a key concept.
It will make writing our code easier. We will see in detail this later in the course.

## Composer

Composer is a free dependency manager written in PHP.
It allows its users to declare and install the libraries that the project needs.

Basically, other developers have written bookstores and make them available for free, ready for use to be integrated and used in other projects.

## Create a new project

To create a new Laravel project using composer, you will need to type this command:

**composer create-project -prefer-dist laravel/laravel blog**

# Laravel Framework

## Overview

We will go through important files and folders in Laravel.
We do a non-exhaustive tour because some things we will understand better later.

- The folder **app / http** :
    - **Controller** will contain our classes controllers
    - **Middleware** it's security
- The folder **config** :
    - app.php: we register the external classes to use, the packages / plugin to use.
    - database.php: configuration file for connection to the database.
      We can see **env () functions.** Well, these function go into thefile **.envlook** for the variables it needs (located at the root).
    - mail.php: contains the configuration for sending mail.
- Thefile: **gitignore**
- The **.env** file containing the variable configurations of your project.
- The folder **database:**
    - **Migration** : These are Laravel functions that allow you to create tables with fields and so on. directly in our BDD.
      So we can create commands that will create DBs and tables directly from Laravel.
- The folder **public** : where we put all our content available to the public
- The folder **resources** : will contain all our resources as well as our **views!**
- The folder **vendor** : This is where all the packages are installed.
- folder **routes** : See chapter below.

# Routes

What is a route?

The routes are what is in the address bar!
Basically, http://yourapp/user or http://yourapp/contact, this is the routes that will manage!

Open routes / web.php:

Open thefile *web.php* located in thefolder *routes.*

We can see that 'Route' is a class that calls a static get () method.
The parameter corresponds to the URL route. Then, the called function returns a view.
So if you go back to **resources / views** there is the 'welcome' view.

We change our route, we put anything for example:

```
Route :: get ('/ foo', function () {
    return view ('welcome');
});
```

Try to access myapp / toto now.

We will slightly modify our code and return a string:

```
Route :: get ('/ foo', function () {
    return "Hello, World";
});
```

Now imagine that I must access the page of a book.
For that, I must receive every time the id of my book in my url.
For that I can create a 'dynamic' routes. She will wait for an id but it can be anything.
Then, we can recover this ID and treat it.

```
Route :: get ('/ book / {id}', function ($ id) {
    return "A book, it's id:". $ Id;
});
```

Give a name to your routes.

It is possible to name your route.

```
Route :: get ('/ book / detail / foo', function () {
//
}) -> name ('book.detail');
```

Then you can access the route with its name and route function:

```
$ url = route ('book.detail');
```

For example:

```
Route :: get ('/ book / detail / foo', function () {
    $ url = route ('book.detail');
    return "This is url:". $ url;
}) -> name ( 'book.detail');
```

With the id in parameter:

```
Route :: get ('/ book / detail / {id}', function ($ id) {
    $ url = route ('book.detail', ['id' => $ id]) ;
    return "This is url:". $ url;
}) -> name ( 'book.detail');
```

To get an overview of your routes, what they do and their descriptions: **php artisan route: list**

# Controllers

What is a controller?

Remember, the controller will be the conductor between our model and our view.
Our controller will therefore request information from the database and transmit the information to the view.

Creating a controller

With Laravel, you can create a new controller easily from the terminal (console).
The advantage of using the console command is that Laravel will configure us the namespace.
Here is the command: **php artisan make: controller TestController**

Once the command is executed, go to the app / http / controllers folder.
You should see your new TestController.php

You can also use another option with the same command, to use the resources.
The resources will be the basic methods defined to manage the CRUD.
Try this command: **php artisan make: controller --resource Test2Controller.**

Router a controller

First, we will create a new route.
This route will use our 'TestController'.

We will write a route by telling him that when this route is requested in the browser, we will use our controller:

```
Route :: get ('/ test', 'TestController');
```

We must now specify the controller method to use:

```
Route :: get ('/ test', 'TestController @ index');
```

Now try going to 'yourapp / test'
If nothing is displayed and the page is blank, it works.

Now back on our controller, in the index method
Try to fill your function with this piece of code:

```
return 'hello, world';
```

Send a parameter to our controller

We will create a new route:

```
Route :: get ('/ book / {id}', 'TestController @ book);
```

In our controller:

```
public function book ($ id)
{
        return 'Book id:'. $ id;
}
```

Controller & Resource

Resources will help us by creating special routes for us.
Let's try an example: Route :: resource ('test', 'TestController');

No need to specify the method. Indeed, we will not send the requests for the index method but we send the requests to create / delete / update etc ...

Now try again in your console the command: php artisan route: list

Ok, to understand better here are some examples.
In the index method, change the code:

```
public function index () {
      return 'This is the index';
 }
```

Next, the show method:

```
public function show ($ id)
        {return'This is the show method';
}
```

Try to access urls 'myapp / test' & 'myapp / book / 2'

# The views

Remember, the views are in the 'resource / views' folder.
By the way, you notice that the view has the extension '.blade.php', WHY?

PHP uses a template engine that will simplify our life.
We will come back to it later but here is an example.
Instead of writing <? Php echo $ maVar; ?>
We can write {{$ maVar}}

Create a view

Create a new file 'comment.blade.php'.
Inside we can just put a little test sentence.
We will also create the function that will handle that in our controller as well as the associated route:

```
public function comment () {
        return "This is a comment";
}

Route :: get ('/ comment', 'TestController @ comment');
```

Let's modify the controller: return view ("comment");

The 'view' function will look in the 'resources / views' folder if it finds a file with the corresponding name.
You can nevertheless create your own folder (other than 'views'). In this case you need to change: return view ("folder.comment");

Send the data to the view.

For now, we know how it goes between the router and the controller:

```
Route :: get ('/ comment / {id}', 'TestController @ comment');

public function comment ($ id) {
        return view ("comment");
  }
```

Now, there are several ways to convey this data to the view.

- With the method with: return view ("comment") -> with ('id', $ id);
- Without: return view ("comment", ['id' => $ id]);

In all cases we must name our variable.
Then retrieve it this way in our view:

```
<p> This is the id: {{$ id}} </ p>
```

## Blade template engine

What is Blade?

To schematize, Blade will take the PHP code and make its use easier and more intuitive within HTML tags.
It makes PHP code easier.

Two of Blade's greatest assets are template inheritance and sections.

Create a template

We will create a template in the folder '/ views / layouts'.
This file will be called 'mytemplate.blade.php'.

```
<! DOCTYPE html>
<html>
<head>
	<title> @yield ('title') </ title>
</ head>
<body>
	<div class = "content">
		@yield ('content')
	</ div >
</ body>
</ html>
```

Here's the new content of comment.blade.php:

```
@extends ('layouts.mytemplate')

@section ('title', 'Comment page title')

@section ('content')
   < p> Inside comment view </ p>
@endsection
```

Even more with blade

On blade we will be able to use another tag to make the use of php easier.
For example logical conditions can be used easily.

Modify your controller:

```
public function comment () {
        $ people = ['John', 'Sarah', 'Michel'];
        return view ('comment', ['people' => $ people]);
  }
```

The view:

```
@section ('content')

   <p> Inside comment view </ p>

   @if (count ($ people))
        <ul>
        @foreach ($ people as $ person)

                <li> Name: {{$ person}}

        @endforeach
        </ ul>
   @endif

@endsection
```