# Laravel

## Get the Laravel framework

Install composer first

composer create-project laravel/laravel example 🡪 **Creates a folder**

composer create-project laravel/Laravel 🡪 **Current folder**

## Understanding the namespace

Folder structure

* app
  + Http 🡪 *route.php*
* *Photon.php* 🡪 class Photon {}
* *Another.php* 🡪 class Photon {} **namespace Eastern**
* *Narzia.php* 🡪 class Narzia {} **namespace Paradise**

### Accessing the classes

#### app/Http/route.php

**namespace Eastern;**

$photon = new **Photon;** 🡪 Accessing the class in Eastern namespace

$photon = new **\Photon** 🡪Accessing the class in Global namespace

$narzia = new **\Paradise\Narzia** 🡪 Accessing the class in Paradise namespace

**or**

namespace Eastern;

**use Paradise\Narzia;**

$photon = new **Photon;** 🡪 Accessing the class in Eastern namespace

$photon = new **\Photon** 🡪Accessing the class in Global namespace

$narzia = new **Narzia** 🡪 Accessing the class in Paradise namespace

### Aliasing

namespace Targaryen;

use Dothraki\Daenerys as Khaleesi;

class Daenerys{}

$daenerys = new Daenerys;

$khaleesi = new Khaleesi;

## JavaScript Object Notation [JSON]

**json\_encode**(

[

[

“id” => 1

“name“ => “Photon Khan”

],

[

“id” => 2

“name“ => “Narzia Markel”

]

]

);

**OUTPUT:** [

{

“id”: 1,

“name”: “Photon Khan”

},

{

“id”: 2,

“name”: “Narzia Markel”  
 }

];

**OUTPUT AS OBJECT**

$data = json\_decode(‘{“panda”: “awesome”}’);

$data🡪panda;

**OUTPUT AS ARRAY**

$data = json\_decode(‘{“panda”: “awesome”}’, true);

$data[‘panda’];

## Dependency Management using [Composer]

{

*"name"*: "photon/practice",

*"description"*: "Learning to learn",

*"keywords"*: ["learn", "namespace", "php"],

*"homepage"*: "http://sphotonkhan.com",

*"license"*: "MIT",

*"authors"*: [

{

*"name"*: "Shabuktagin Photon Khan",

*"email"*: "khan.photon@gmail.com",

*"homepage"*: "http://sphotonkhan.com",

*"role"*: "Web Developer"

}

],

*"require"*: {

*"xmen/wolverine"* : ">1.0.0, <1.0.2",

*"xmen/cyclope"* : "1.0.\*",

*"xmen/storm"*: "=<2.0.0",

*"xmen/gambit"*: "=>1.5.1"

}

}

### Developer Branch Name

{“xmen/gambit”: “dev-branchname”}

### Developer Master Branch

{

“require”:{

“xmen/gambit”: “dev-master”

},

“minimum-stability”: “dev”

}

### Development Dependencies

{

“require”:{

“xmen/gambit”: “dev-master”

},

“require-dev”: {

“codeception/codeception”: “1.6.0.3”

}

}

### Conflicting Packages

{

“conflict”:{

“xmen/gambit”: “dev-master”

},

}

### Replacements

{

“replace”:{

“xmen/gambit”: “dev-master”

},

}

### Provisions

{

“provide”:{

“xmen/gambit”: “dev-master”

},

}

### Suggestions

{

“suggest”:{

“xmen/gambit”: “dev-master”

},

}

## Autoloading

### To load the files (Not preferable)

{

“autoload”: {

“files”: [

“path/to/my/firstfile.php”,

“path/to/my/secondfile.php”

]

}

}

### To load the classes (Not preferable)

{

“autoload”: {

“classname”: [

“src/Models”,

“src/Controller”

]

}

}

### Namespaces matches up with the directory

{

“autoload”: {

“psr-4”: {

“Dayle**\\**Blog**\\**”: “src”

}

}

}

Dayle\Blog\Content\Post 🡪 src/Content/Post

It is important to have the double backslash

#### Example

Folder Structure

**Root**

* Eastern
  + BuildingOne
  + BuildingTwo
    - ***Photon.php***
* Paradise
  + BuildingOne
    - ***Narzia.php***
  + BuildingTwo
* **composer.json**
* read.php
* read2.php
* read3.php

##### Photon.php

1. <? php
2. namespace Eastern;
3. class Photon {
4. public
5. function \_\_construct() {
6. echo“ I am Photon”;
7. }
8. }

##### Narzia.php

1. <? php
2. namespace Paradise;
3. class Narzia {
4. public
5. function \_\_construct() {
6. echo“ I am Narzia”;
7. }
8. }

##### composer.json

{

“autoload”: {

“psr-4”: {

“Eastern\\”: “Eastern/BuildingTwo/”,

“Paradise\\”: “Eastern/BuildingOne/”,

}

}

}

##### read.php

1. <?php
2. namespace Eastern;
3. require\_once $\_SERVER['DOCUMENT\_ROOT']."/study/Namespace/vendor/autoload.php";
4. $photon = new Photon();
5. echo“ < br / > ”;
6. $narzia = new\ Paradise\ Narzia();

##### read2.php

1. <?php
2. namespace Reading;
3. require\_once $\_SERVER['DOCUMENT\_ROOT']."/study/Namespace/vendor/autoload.php";
4. $photon = new\ Eastern\ Photon();
5. echo "<br/>";
6. $narzia = new\ Paradise\ Narzia();

##### read3.php

1. <?php
2. namespace Reading;
3. require\_once $\_SERVER['DOCUMENT\_ROOT']."/study/Namespace/vendor/autoload.php";
4. use Eastern as e;
5. use Paradise as p;
6. $photon = new e\ Photon();
7. echo "<br/>";
8. $narzia = new p\ Narzia();

## Configuration

### Accessing the configuration values

We can access the configuration values in two ways

* Facade
  + Config∷get(‘app.debug’, false)
* Helper
  + config(‘app.debug’, false)

### Dissecting the app.debug

app . debug

{“app.php”} {“array key from app.php”}

The env() function is the new way of making different configuration possible in different environments but .env file is available locally in Laravel framework.

We can use $host = env(‘DB\_HOST’, ‘127.0.0.1’)

However, it is heavily discouraged since it cant cache values from environmental variables, so if we choose to use env() outside of the PHP configuration files then we might end up getting into a caching mess.

### Caching

php artisan config:cache

php artisan config:clear

## Basic Routing

Go to the **routes folder** then click on *web.php*

### Normal Routing

Route::get('/home', function(){

return "Home Page";

});

If we go to *localhost/laravel/public/home* 🡪 *Home Page* will show up

### **Anonymous function/Closure Routing**

$logic = **function(){**

**return “Contact Us”**

**};**

Route∷get(‘/contact’, $logic);

If we go to *localhost/laravel/public/contact*🡪 *Contact Us* will show up

### Route Parameters

Route∷(“/information/**{department}**”, function(**$department**){

**return “{$department}”;**

});

### Optional Parameters

*Route*::get("/partners/**{name?}**", *function*(**$name = null**){

if($name == null){

return "Sorry, we couldn't find any partners";

}else{

return "{$name}";

}

});

### Some of the routing verbs

Route∷get();

Route∷post();

Route∷put();

Route∷patch();

Route∷delete();

Route∷any();

## Views

Go to resources 🡪 then to views 🡪 then create *simple.php*

### simple.php

<!DOCTYPE html>

<html>

<head>

<meta charset="utf-8" />

<meta http-equiv="X-UA-Compatible" content="IE=edge">

<title>Page Title</title>

<meta name="viewport" content="width=device-width, initial-scale=1">

</head>

<body>

<p> Welcome to the category page</p>

</body>

</html>

**Update the *routes/web.php***

### web.php

Route∷get(“/category”, function(){

return View∷make(‘simple’);

});

### Passing array of data in the route

Route∷get(“/category/{some\_data?}”, function($some\_data = null){

if($some\_data == null){

$data[‘some\_data’] = $some\_data;

return View∷make(‘simple’, $data);

}

else{

return View∷make(‘simple’);

}

});

### Redirecting the Route

*Route*::get("/outsource", *function*(){

return *Redirect*::to('/');

});

### Custom Responses

Route∷get(‘/custom/response’, function(){

return Response∷make(‘Hello world’, 200);

});

#### Custom Header

Route∷get(‘/custom/response2’, function(){

$response = Response∷make(‘Hello World’, 200);

$response🡪headers🡪set(‘Content-Type’, ‘text/html’);

return $response;

});

We can also set the time-to-live value (TTL) for shared caches

Route∷get(‘/custome/response2’, function(){

$response = Response∷make(‘Hello World’, 200);

$response🡪setTtl(60);

return $response;

});

#### JSON Response

Route∷get(‘/get/json’, function(){

$data = [‘superman’, ‘spiderman’, ‘batman’];

return Response∷json($data);

});

#### Download Response

*Route*::get('/file/download', *function*(){

$file = $\_SERVER['DOCUMENT\_ROOT']."/laravel/resources/img/vscode\_shortcuts.pdf";

return *Response*::download($file);

});

#### Download a file with status & header

*Route*::get('/file/download', *function*(){

$file = $\_SERVER['DOCUMENT\_ROOT']."/laravel/resources/img/vscode\_shortcuts.pdf";

return *Response*::download($file, ‘vscode.pdf’, [‘iron’, ‘man’]);

});

## Blade Templating

### /routes/web.php

*Route*::get('blade/index', *function*(){

$data = ["header" => "Blade Index", "heading" => "Welcome to the blade index page"];

return *View*::make('index', $data);

});

### /resources/views/index.blade.php

<!DOCTYPE html>

<html lang = 'us'>

<head>

<meta charset="utf-8" />

<meta http-equiv="X-UA-Compatible" content="IE=edge">

<title> **{{ $header }}** </title>

<meta name="viewport" content="width=device-width, initial-scale=1">

</head>

<body>

<div class = 'container'>

<div class = 'row'>

<div class = 'col'>

<h1> **{{ $heading }}** </h1>

</div>

</div>

</div>

</body>

</html>

### Plug-In Functions

<div class = ‘row’>

<div class = ‘col’>

<span>Current Date ∷ **{{ date(‘d/m/y’) }}** </span>

</div>

</div>

### Control Statements

#### if/else

**@if($control == 'yes')**

<p> Well done!</p>

**@elseif($control == 'no')**

<p> Sorry, for your loss</p>

**@else**

<p> Nothing </p>

**@endif**

#### foreach

**@foreach ($countries as $country)**

<p>{{ $country }}</p>

**@endforeach**

#### forloop

**@for($i = 0; $i < count($countries); $i++)**

<p>{{ $countries[$i] }}</p>

**@endfor**

#### while

{{ $i = 0}}

**@while($i < count($countries))**

<p>{{ $countries[$i] }}</p>

<p>{{ $i++ }}</p>

**@endwhile**

#### unless

**@unless($control == 'yes')**

<p> Well done!</p>

**@endunless**

### Template for Inclusion

#### routes/web.php

*Route*::get('blade/template', *function*(){

return *View*::make('main');

});

#### resources/views/main.blade.php

<!DOCTYPE html>

<html>

<head>

<meta charset="utf-8" />

<meta http-equiv="X-UA-Compatible" content="IE=edge">

<title>Template Testing</title>

<meta name="viewport" content="width=device-width, initial-scale=1">

</head>

<body>

<div class = 'container'>

**@include('header')**

<div class = 'row'>

<div class = 'col'>

<span> Just starting my contents over here </span>

</div>

</div>

**@include('footer')**

</div>

</body>

</html>

#### resources/views/header.blade.php

<div class = 'row'>

<div class = 'col'>

<span>Welcome to my page! I am here to start new things </span>

</div>

</div>

#### resources/views/footer.blade.php

<div class = 'row' style = 'margin-top: 500px'>

<div class = 'col'>

<div class = 'footer'>

<span>2018 &copy; All Right Reserved | Photon Enterprise </span>

</div>

</div>

</div>

### Template Inheritance

#### routes/web.php

*Route*::get('blade/inheritance', *function*(){

return *View*::make('child');

});

#### resources/views/child.blade.php

**@extends('layouts.parent')**

**@section('head')**

**@parent**

<p> Child:: I will tell it here </p>

**@stop**

**@section('body')**

<p> Child: Let me give something to talk about. </p>

**@stop**

#### resources/views/layouts/parent.blade.php

<!DOCTYPE html>

<html>

<head>

<meta charset="utf-8" />

<meta http-equiv="X-UA-Compatible" content="IE=edge">

<title>Practice Blade Inheritance</title>

<meta name="viewport" content="width=device-width, initial-scale=1">

</head>

<body>

**@section('head')**

<p>Parent:: I go something to say</p>

**@show**

**@yield('body')**

</body>

</html>

**@yield(‘head’)** ∷ Means the fill the content later

**@section(‘body’)**

//My name is Photon

**@show**

This both together means @yield but by default show some content

**@extends(‘layouts.parent’)**

*/resources/views/layouts/parent.blade.php*

Means “.” is the folder separator in the /resources/views 🡪 layouts∷folder, parent∷file

**@section(‘body’)**

//My name is Photon Khan

**@stop**

This replaces My name is Photon to My name is Photon Khan

**@section(‘body’)**

**@parent**

//My name is Photon Khan

**@stop**

This adds up the content. Therefore, it shows My name is Photon and My name is Photon Khan

### Comment

{{-- This is comment --}}

### JavaScript Support

<script> **@{{javascriptValue}}** </script>