# 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

**json\_encode**(

[

[

“id” => 1

“name“ => “Photon Khan”

],

[

“id” => 2

“name“ => “Narzia Markel”

]

]

);

**OUTPUT:** [

{

“id”: 1,

“name”: “Photon Khan”

},

{

“id”: 2,

“name”: “Narzia Markel”  
 }

];

### JSON Decode

**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(‘/custom/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>

## Request Data

### Request All

#### /routes/web.php

Route∷get(‘/dumpdata’, function(){

$data = Request∷all();

var\_dump($data);

});

<http://localhost/laravel/public/dumpdata>

#### /routes/web.php

Route∷get(‘/dumpdata’, function(){

$data = Request∷all();

var\_dump($data);

});

*Route*::get('/dumpdata', *function*(){

$data = *Request*::get('foo');

echo "<pre>";

var\_dump($data);

});

//Give only the data of $\_GET[‘foo’]

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

return View::make('form');

});

#### /resources/views/form.blade.php

<!DOCTYPE html>

<html>

<head>

<meta charset="utf-8" />

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

<title>Form Action to the rescue</title>

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

</head>

<body>

<form action = " {{ **url('/dumpdata')** }}">

**{{csrf\_field()}}**

<input type = 'hidden' name = 'foo' value = 'bar'/>

<input type = 'hidden' name = 'baz' value = 'boo'/>

<input type = 'submit' value = 'send'/>

</form>

</body>

</html>

### Post Request

#### /routes/web.php

*Route*::get('/postform', *function*(){

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

});

*Route*::post('/dumppostdata', *function*(){

$data = *Request*::all();

echo "<pre>";

var\_dump($data);

});

#### /resources/views/postform.blade.php

<!DOCTYPE html>

<html>

<head>

<meta charset="utf-8" />

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

<title>Form Action to the rescue</title>

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

</head>

<body>

<h4> Simple Post + Get Request with hidden forms </h4>

<form action = **"{{ url('/dumppostdata') }}?foo=get&baz=get"**>

{{csrf\_field()}}

<input type = 'hidden' name = 'foo' value = 'bar'/>

<input type = 'hidden' name = 'baz' value = 'boo'/>

<input type = 'submit' value = 'send'/>

</form>

</body>

</html>

It seems the GET data is handled last, and the values are replaced. Now we know that that GET data, takes a higher priority than POST data within request data array.

* Ex: In this situation if we put $\_GET[‘war’] if would give us null since it is not sent from the form. i.e Request∷get(‘war’)
* Ex: In this situation if we put both $\_GET[‘war’] & $\_GET[‘foo’]

i.e Request∷get(‘war’, ‘foo’) we will get “bar” as a string

### Request Has

#### /routes/web.php

Route∷get(‘/dumpdata’, function(){

$data = **Request∷has(‘foo’);**

var\_dump($data);

});

If we send a get request from a form it will show us **bool(true)**

### Request Only

#### /routes/web.php

Route∷get(‘/dumpdata’, function(){

$data = **Request∷only(‘foo’, ‘baz’);**

var\_dump($data);

});

It returns only the $\_GET[‘…’] variables that we want

### Request Except

Route∷get(‘/dumpdata’, function(){

$data = **Request∷except(‘baz’);**

var\_dump($data);

});

It returns all the $\_GET[‘…’] except the one mentioned over here

### Store request for longer time

#### /routes/web.php

Request∷get(‘/dumpdata’, function(){

return Redirect∷to(‘/new/request/’);

});

Route∷get(‘/new/request’, function(){

var\_dump(Request∷all());

});

When the form sends a request to the dumpdata, it redirects to new/request. Unfortunately, it gives no value in the array. Therefore, to make the redirect work we need to use **Request∷flash()**

Then, use the Request∷old() to let the Laravel know that we want the previous value

#### /routes/web.php

Request∷get(‘/dumpdata’, function(){

**Request∷flash();**

return Redirect∷to(‘/new/request/’);

});

Route∷get(‘/new/request’, function(){

var\_dump(**Request∷old()**);

});

### Flash Only

#### /routes/web.php

Request∷get(‘/dumpdata’, function(){

**Request∷flashOnly(‘foo’);**

return Redirect∷to(‘/new/request/’);

});

Route∷get(‘/new/request’, function(){

var\_dump(**Request∷old()**);

});

### Flash Except

#### /routes/web.php

Request∷get(‘/dumpdata’, function(){

**Request∷flashExcept(‘foo’);**

return Redirect∷to(‘/new/request/’);

});

Route∷get(‘/new/request’, function(){

var\_dump(**Request∷old()**);

});

### Alternative to Flash

#### /routes/web.php

*Route*::get('/dumpdata', *function*(){

return ***Redirect*::to('/new/request')->withInput();**

});

which is equivalent to

*Route*::get('/dumpdata', *function*(){

Request∷flash();

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

});

### Alternative to Flash Partial (Flash Only)

#### /routes/web.php

Route∷get(‘/dumpdata’, function(){

return **Redirect∷to(‘/new/request’)→withInput(Request∷only(‘foo’));**

});

which is equivalent to

*Route*::get('/dumpdata', *function*(){

Request∷flashOnly(‘foo’);

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

});

### Alternative to Flash Partial (Flash Except)

#### /routes/web.php

Route∷get(‘/dumpdata’, function(){

return **Redirect∷to(‘/new/request’)→withInput(Request∷except(‘foo’));**

});

which is equivalent to

*Route*::get('/dumpdata', *function*(){

**Request∷flashExcept(‘foo’);**

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

});

### File Upload

#### /resources/views/fileupload.blade.php

<!DOCTYPE html>

<html>

<head>

<meta charset="utf-8" />

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

<title>File Upload</title>

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

</head>

<body>

**<form action = "{{url('/filepost')}}" method = 'POST' enctype = 'multipart/form-data'>**

{{csrf\_field()}}

<label for = 'Upload'>File Upload</label><br/>

<input type = 'hidden' name = "MAX\_FILE\_SIZE" value = '524288'/>

<input type = 'file' name = 'upload'/><br/>

<input type = 'submit'/>

</form>

</body>

</html>

#### /routes/web.php

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

return View::make('fileupload');

});

Route::post('/filepost', function(){

**var\_dump(Request::file('upload'));**

});

#### Procedural PHP to Laravel PHP

*$\_FILES[‘upload’][‘name’]*

file('upload')->**getClientOriginalName() ∷** Original File Name

*photon.pdf*

file(‘upload’)->**getFileName() ∷** Get Temporary File Name

*php2D4D.tmp*

*$\_FILES[‘upload’][‘size’]*

file(‘upload’)->**getClientSize() ∷** File Size

*2370413*

*$\_FILES[‘upload][‘type’]*

file(‘upload’)->**getMimeType() ∷** File Type

*application/pdf, image/jpeg, image/jpg, image/png*

file(‘upload’)->**guessExtension()** ∷ Guesses the Extension of the file

*pdf, jpg*

$\_FILES[‘upload’][‘tmp\_name’]

file(‘upload’)->**getRealPath()** ∷ Get the temporary file location

/tmp/php/php2D4D.tmp

*move\_uploaded\_file($temporary\_location, $destination\_location)*

file(‘upload’)->move(**$\_SERVER['DOCUMENT\_ROOT'].'/laravel/storage/directory/'**);

### Original Filename

#### /routes/web.php

Route∷get(‘/filepost’, function(){

var\_dump(Request∷file(‘upload’)🡪**getClientOriginalName**()):

});

### Temporary Filename

#### /routes/web.php

Route∷get(‘/filepost’, function(){

var\_dump(Request∷file(‘upload’)🡪**getFileName**()):

});

### File Size

#### /routes/web.php

Route∷get(‘/filepost’, function(){

var\_dump(Request∷file(‘upload’)🡪**getClientSize**()):

});

### Get the MIME type

#### /routes/web.php

Route∷get(‘/filepost’, function(){

var\_dump(Request∷file(‘upload’)🡪**getMimeType**()):

});

### Guess the Extension of the files

#### /routes/web.php

*Route*::post('/filepost', *function*(){

echo "<pre>";

var\_dump(*Request*::file('upload'**)->guessExtension**());

});

### Get the Temporary Path

#### /routes/web.php

*Route*::post('/filepost', *function*(){

echo "<pre>";

var\_dump(*Request*::file('upload'**)->getRealPath**());

});

### Move the uploaded file

#### /routes/web.php

Route::post('/filepost', function(){

**Request::file('upload')->move($\_SERVER['DOCUMENT\_ROOT'].'/laravel/storage/directory/');**

return "File was moved";

});

### Move the uploaded file with our designated file name

#### /routes/web.php

*Route*::post('/filepost', *function*(){

$name = *Request*::file('upload')->getClientOriginalName();

***Request*::file('upload')->move($\_SERVER['DOCUMENT\_ROOT'].'/laravel/storage/directory/', $name);**

return "File was moved";

});

## Cookies

*setcookie(*name,value,expire,path,domain,secure,httponly*) 🡪 Set the cookie*

*setcookie(name, “”) 🡪 Delete the cookie*

*$\_COOKIE[name] 🡪 Get the cookie*

#### /routes/web.php

*Route*::get('/createcookie', *function*(){

$cookie = ***Cookie*::make('username', 'khan.photon', 30);**

return ***Response*::make('/readcookie')->withCookie($cookie);**

});

### Get Cookie

#### /routes/web.php

*Route*::get('/readcookie', *function*(){

$cookie = ***Cookie*::get('username');**

var\_dump($cookie);

});

### Has Cookie

#### /routes/web.php

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

$cookie = Cookie::has('username');

var\_dump($cookie);

});

### Cookie with no expiry

#### /routes/web.php

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

$cookie = Cookie::forever('username');

return Response∷make(‘/readcookie’)->withCookie($cookie)

});

### Forget the cookie

#### /routes/web.php

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

$cookie = Cookie::forget('username');

return $cookie;

});

## Advanced Routing

### Named Routes

#### /route/web.php

*Route*::get('/i/like/shortcuts', [**"as" => "shortcuts",**

*function*(){

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

}

]);

#### /resources/views/shortcuts.blade.php

<!DOCTYPE html>

<html>

<head lang = 'en'>

<meta charset="utf-8" />

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

<title>Shortcuts</title>

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

</head>

<body>

I like shortcuts

**{{ route('shortcuts') }}**

</body>

</html>

### Redirecting the route

#### /route/web.php

Route::get('/i/like/shortcuts', **["as" => "shortcuts",**

**function(){**

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

**}**

**]);**

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

return **Redirect::route('shortcuts');**

});

### Get the name of the current shortcut name

#### /route/web.php

Route::get('/i/like/shortcuts', ["as" => "shortcuts",

function(){

return **Route::currentRouteName()**;

}

]);

### Route with the controller name

#### /route/web.php

*Route*::get('/first/shortcuts', [

**"as" => "shortcuts",**

**"uses" => "FirstController@show"**

]);

*Route*::get('/redirectshortcuts', *function*(){

**return *Redirect*::route('shortcuts');**

});

#### /app/http/controllers/FirstController

namespace **App\Http\Controllers**;

use **App\Http\Controllers\Controller**;

class **FirstController** extends **Controller**{

public function **show()**{

return **view('child')**;

}

}

*For the* ***view(‘child’)*** *🡪 The* ***service method worked*** *but façade didn’t work*

*view(‘child’) ∷ Service Method; View∷make(‘child’) ∷ Façade method*

### Parameter Constraints

#### /route/web.php

Route::get('/save/{name}', function($name){

return "My name is: {$name}";

**})->where('name', '[A-Za-z]+');**

#### /route/web.php

Route::get('/save/{firstname}/{lastname}',

function($firstname, $lastname){

return "My name is : {$firstname} {$lastname}";

**})->where('firstname', '[A-Za-z]+')->where('lastname', '[A-Za-z]+');**

### Route Groups

#### /route/web.php

**Route::group([], function(){**

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

return "Critical";

});

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

return "Medical";

});

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

return "Condition";

});

**});**

### Route Prefix

#### /route/web.php

**Route::group(["prefix" => "books"], function(){**

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

return "Critical";

});

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

return "Medical";

});

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

return "Condition";

});

});

*Therefore the routing name needs to be /books/critical, /books/medical, /books/condition*

### Domain Routing

#### /route/web.php

Route::group(["domain" => "admin.localhost"], function(){

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

return "Critical";

});

});

*This group needs to be placed at the top of the file & this do not work in localhost*

#### /route/web.php

Route::group(["domain" => "{user}.admin.localhost:8000"], function(){

Route::get('profile/{page}', function($user, $page){

return "{$user} {$page}";

});

});

*This group needs to be placed at the top of the file & this do not work in localhost*

## Controllers

### Creating Controllers

#### /routes/web.php

Route::group(['prefix' => "article"], function(){

Route::get('/', 'ArticleController@index');

Route::get('/new', 'ArticleController@new');

});

#### /app/Http/Controllers/ArticleController.php

namespace App\Http\Controllers;

use App\Http\Controllers\Controller;

class ArticleController extends Controller{

public function index(){

return view('article');

}

public function new(){

return view('articlenew');

}

}

*Small Reminder: The View∷make(‘article’) won’t work over here!*

*Only the service method works, the façade method do not works*

### Creating Controller using Artisan

artisan make:controller TestingController --resource

#### /apps/Http/Controllers/TestingController.php

TestingController now have several methods

index()

create()

store(Request $request)

show($id)

edit($id),

update(Request $request, $id)

destroy($id)

Instead of writing Route∷group we can now write Route∷resource(‘testing’, ‘TestingController’)

#### /routes/web.php

Route::resource('testing', 'TestingController');

index() [GET] - */testing*∷ Read from the database, Generate links

create() [GET] - */testing/create* ∷ Show the form for adding data in database

store() [POST] - */testing*∷ Handle the form and store it in the database

show($id) [GET] - */testing/{id}*∷ Render the read in index() using the $id

edit($id) [GET] - /testing/{id}/edit∷ Show the form for editing using the $id

update($id) [PUT|PATCH] - */testing/{id}*∷ Updates the data in database

destroy($id) [DELETE] - /testing/{id} ∷ Deletes the data in database

### Dependency Injection

Therefore, using the dependency injection in controllers means that we no longer use facades like View∷make(‘hello’) or Request∷all(‘hello’). Instead, we inject those components directly. Add the line below to inject the view. Contracts are interfaces that all services follow.

#### /apps/Http/Controllers/TestingController.php

namespace App\Http\Controllers;

use Illuminate\Http\Request;

**use Illuminate\Contracts\View\Factory as View;**

class TestingController extends Controller{

protected $view;

public function \_\_construct(**View $view**){

**$this->view = $view;**

}

public function index(**View $view**) {

return **$view->make('child');**

}

}

## URL Generation

### Get the Current URL

#### /routes/web.php

Route::get('/current/url', function(){

return **URL::current();**

});

*http://localhost/laravel/public/current/url*

### Get the full URL including the request

#### /routes/web.php

Route::get('/current/url', function(){

return **URL::full();**

});

*http://localhost/laravel/public/current/url?foo=bar*

### Get the previous URL

#### /routes/web.php

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

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

});

Route::get("/showurl", function(){

return **URL::previous();**

});

### Generated URL

#### /routes/web.php

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

return **URL::to('/anotherdimension');**

});

It will return a generated URL rather than an URL generation instance

*http://localhost/laravel/public/anotherdimension*

#### /routes/web.php

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

return [**URL::to('/anotherdimension**](URL::to('/anotherdimension)**', [‘ok’, ‘no’]);**

});

*http://localhost/laravel/public/anotherdimension/ok/no*

#### /routes/web.php

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

return [**URL::to('/anotherdimension**](URL::to('/anotherdimension)**', [‘ok’, ‘no’], true);**

});

[*http://localhost/laravel/public/anotherdimension/ok/no*](http://localhost/laravel/public/anotherdimension/ok/no)

### HTTPS URLS

#### /routes/web.php

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

return **URL::secure('/secureroute', ['ok', 'secret']);**

});

### Reveal Shortcuts

#### /routes/web.php

Route::get('/superhero/shortcut', [

**'as' => 'superhero',**

function(){

return "It is the superhero shortcut";

}

]);

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

return **URL::route('superhero');**

});

### Named Routes with parameters

#### /routes/web.php

Route::get('/hail/{first}/the/{second}', [

'as' => 'superhero',

function($first, $second){

return "It is the $first the $second";

}

]);

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

return URL::route('superhero', ['damn', 'son']);

});

### Controller Route

#### /routes/web.php

Route::get('triggercontroller', function(){

return **URL::action('FirstController@show');**

});

### Absolute Asset Path

#### /routes/web.php

Route::get('triggercontroller', function(){

return **URL::asset('img/logo.png’);**

});

Get the absolute path of the CSS or the JavaScript files

### Asset Secure Path

#### /routes/web.php

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

return **URL::asset**('/img/book.png', true);

});

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

return **URL::secureAsset**('/img/book.png', true);

});

## Databases

Eloquent provides a great database setup for Laravel

Change the information of the database in */config/database.php*

After setting up the host, database name, username and password

By default, database results will be returned as instances of the PHP stdClass object; however, we may desire to retrieve records in an array format

PDO∷FETCH\_CLASS, PDO∷FETCH\_ASSOC, PDO∷FETCH\_NUM

Also set .***env file***

### Create Table

#### /routes/web.php

**Schema::create('users', function($table)**{

Schema∷create(‘table-name’, closure);

First parameter :: Name of the table

Second parameter :: It is used to build the table structure

### Column Types

$table→increments(‘id’);

$table→bigIncrements(‘id’);

$table→string(‘name’, 128);

$table→text(‘description’);

$table→integer(‘size’);

$table→bigInteger(‘huge\_size’);

$table→mediumInteger(‘medium\_size’);

$table→tinyInteger(‘127 to 128’);

$table→smallInteger(‘-32768 to 32767);

$table→float(‘size’);

$table→decimal(‘size’, ‘significant\_figure’);

$table→boolean(‘trueornot’);

$table→enum(‘what’, [‘damn’, ‘ok’, ‘nope’]);

$table→date(‘when’);

$table→dateTime(‘when’);

$table→time(‘when’);

$table→timestamp(‘when’);

$table→timestamps();

$table→binary(‘image’);

$table→softDeletes(); ∷ Mark table row as deleted without actually being deleted

### Column Modifiers

$table→string(‘username’)→unique();

$table→primary(‘username’);

$table→primary([‘id’, ‘username’, ‘email’]) ∷ Composite Unique

$table→integer(‘age’)→index ∷ FULLTEXT INDEX

$table→index([‘age’, ‘weight’]); ∷ Multiple Indices

$table→nullable(false) ∷ Cannot be null

$table→nullable() ∷ Can be null by default

$table→string(‘name’)→default(‘John Doe’) ∷ Default value to a column

$table→integer(‘age’)→unsigned() ∷ To make sure it won’t have any negative numbers

### Reordering the column order

Schema∷create(‘table-name’, function($table){

$table→string(‘name’)→**after(‘id’);**

$table→increments(‘id’);

});

### Updating Tables

Schema∷rename(‘users’, ‘customers’); ∷ Renaming the table

Schema∷table(‘table-name’, closure); ∷ Modify the table

Schema∷table(‘table-name’, function(){

$table→increments(‘id’);

$table→string(‘name’);

});

Schema∷table(‘table-name’, function($table){

**$table→dropColumn(‘name’);**

});

Schema∷table(‘table-name’, function($table){

**$table→dropColumn([‘name’, ‘email’]);**

});

Schema∷table(‘table-name’, function($table){

**$table→renameColumn([‘name’, ‘email’]);**

});

Schema∷table(‘table-name’, function($table){

**$table→dropPrimary(‘id’);**

});

Schema∷table(‘table-name’, function($table){

**$table→dropPrimary([‘id’, ‘name’]);**

});

Schema∷table(‘table-name’, function($table){

**$table→dropUnique([‘id’, ‘name’]);**

});

### Deleting Tables

Schema∷drop(‘table-name’)

Schema∷dropIfExists(‘table-name’)

### Schema Tricks

Schema∷connection(‘mysql’)→create(‘example’, function($table){

$table→increments(‘id’);

});

This can be used alternatively to modify the columns or table

Schema∷hasTable(‘table-name’) ∷ Checks whether the table exists or not

Schema∷hasColumn(‘table-column) ∷ Checks if the columns exist or not

### Change Database Engine

$table→engine = ‘InnoDB’;

## Migration

php artisan make:migration create\_users\_table

These creates the migration timings and records with a class and up() and down() methods

php artisan make:migration create\_users\_table --create=”users”

Creates a migration in */database/migrations/…*

php artisan make:migration create\_users\_table –-path=”app/migs”

Change the saving destination of the migration

php artisan migrate:install

php artisan migrate:rollback

php artisan migrate:refresh

For different database connection 🡪 php artisan migrate –database=mysql

php artisan migrate –pretend 🡪 To see what the outcome may be

## Eloquent ORM(Object Relational Mapping)

### Creating New Table

php artisan make:migration create\_games\_table --create=games

#### /database/migrations/ …migration file…

<?php

use Illuminate\Support\Facades\Schema;

use Illuminate\Database\Schema\Blueprint;

use Illuminate\Database\Migrations\Migration;

class CreateGamesTable extends Migration

{

public function up(){

Schema::create('games', function (Blueprint $table) {

$table->increments('id');

$table->string('name', 128);

$table->text('description');

$table->timestamps();

});

}

public function down(){

Schema::dropIfExists('games');

}

}

php artisan migrate

### Add data to the table

php artisan make:model Game

#### /app/Game.php

<?php

namespace App;

use Illuminate\Database\Eloquent\Model;

class Game extends Model{

public $timestamps = true;

}

#### /routes/web.php

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

$game = **new \App\Game;**

$game->name = 'Assasins Creed';

$game->description = 'Assassins Vs Templars.';

$game->**save();**

});

In this way, we know eloquent is smart. We usually name our tables in plural form and we call models by its singular form Therefore; a table name can be “users” and its model name will be “User”. We can also set the property ($timestamps) to false if we do not want our ORM to update by itself.

### Add data without following the Model and tables name convention

#### /app/Game.php

<?php

namespace App;

use Illuminate\Database\Eloquent\Model;

class Game extends Model{

**public $timestamps = true;**

**public $table = ‘some\_other\_names’;**

}

If we do not want our table name to be the plural form of the Model. Then we run this model

### Read the data using ID

#### /routes/web.php

Route::get('readdata', function(){

$game **= \App\Game::find('1');**

return $game->name;

});

We used the static method to access the object’s method

### Update data using ID

#### /routes/web.php

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

$game = \App\Game::find(2);

$game->name = "Alladin";

$game->description = "Too lazy to work";

$game->save();

});

### Delete Data using ID

#### /routes/web.php

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

$game = \App\Game::find(2);

$game->delete();

});

### Delete Multiple Data using ID

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

$game = new \App\Game;

$game->destroy([3,4,5]);

});

## Eloquent Queries

### Creating New Table

#### /database/migrations/ …migration..name…

Schema∷create(‘albums’, function($albums){

$table->increments('id');

$table->string('title', 256);

$table->string('artist', 256);

$table->string('genre', 128);

$table->integer('year');

$table->timestamps();

});

Create a model name 🡪 Albums

### Adding Multiple Data

#### /routes/web.php

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

$album = new \App\Album;

$album->title = 'Some Made Hope';

$album->artist = 'Matt Nathanson';

$album->genre = 'Acoustic Rock';

$album->year = 2007;

$album->save();

$album = new \App\Album;

$album->title = 'Please';

$album->artist = 'Matt Nathanson';

$album->genre = 'Acoustic Rock';

$album->year = 1993;

$album->save();

$album = new \App\Album;

$album->title = 'Leaving through the window';

$album->artist = 'Something Corporate';

$album->genre = 'Piano Rock';

$album->year = 2002;

$album->save();

$album = new \App\Album;

$album->title = 'North';

$album->artist = 'Something Corporate';

$album->genre = 'Piano Rock';

$album->year = 2002;

$album->save();

$album = new \App\Album;

$album->title = '...Anywhere But Here';

$album->artist = 'The Ataris';

$album->genre = 'Punk Rock';

$album->year = 1997;

$album->save();

$album = new \App\Album;

$album->title = '...Is a Real Boy';

$album->artist = 'Say Anything';

$album->genre = 'Indie Rock';

$album->year = 2006;

$album->save();

});

### Reading the Data

#### /routes/web.php

##### Reading single data (property)

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

$album = **\App\Album::find(1);**

return $album->title;

});

##### Reading single data (whole)

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

$album = \App\Album::find(1);

return $album;

});

It gives the data in json format

##### Reading all the data

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

$albums = **\App\Album::all();**

foreach($albums as $album):

echo $album->title."<br/>";

endforeach;

});

This works because the \_\_toString() method has been inherited by the model

##### Reading collection of data

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

$albums = **\App\Album::find([1,3]);**

echo "<pre>";

var\_dump($albums);

});

Reads row of id 1 & 3

##### Reading first data

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

$albums = \App\Album::first();

return $albums;

});

### Updating the data using where

#### /routes/web.php

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

**\App\Album::where('artist', '=', 'Matt Nathanson')**

**->update(['artist' => 'Photon Khan']);**

return \App\Album::all();

});

### Deleting the data using where

#### /routes/web.php

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

\App\Album::where('title', '=', 'North')

->delete();

return \App\Album::all();

});

### Reading the data using get and where

#### /routes/web.php

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

return \App\Album::where('title', '=', 'Please')

**->get();**

});

### Reading the data using specific parameters in get

#### /routes/web.php

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

return \App\Album::where('title', '=', 'Please')

-**>get(['id', 'title', 'artist']);**

});

### Fetch Method Query

#### /routes/web.php

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

**return \App\Album::pluck('artist');**

});

### Retrieve array of values

#### /routes/web.php

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

return \App\Album::where('artist', '=', 'Something Corportate')

**->toSql();**

});

### Query Constraints

#### /routes/web.php

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

**return \App\Album::where('artist', '=', ‘Photon Khan')→get();**

});

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

**return \App\Album::where('artist', '=', 'Photon Khan')->first();**

});

### LIKE Search

#### /routes/web.php

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

**return \App\Album::where('title', 'LIKE', '...%')->get();**

});

### Multiple Where

#### /routes/web.php

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

**return \App\Album::where('artist', '=', 'Photon Khan')**

**->where('genre', '=', 'Acoustic Rock')**

**->get();**

});

### OR Where

#### /routes/web.php

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

**return \App\Album::where('artist', '=', 'Photon Khan')**

**->orWhere('artist', '=', 'the Ataris')->get();**

});

### Raw Where

#### /routes/web.php

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

**return \App\Album::whereRaw('artist = ? AND title LIKE ?', ['Say Anything', '...%'])->get();**

});

### Between Where

#### /routes/web.php

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

**return \App\Album::whereBetween('year', ['2000', '2010'])→get();**

});

### Nested Where

#### /routes/web.php

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

**return \App\Album::whereNested(function($query){**

$query->where('year', '>', 2000);

$query->where('year', '<', 2035);

**})->get();**

});

### Where In

#### /routes/web.php

*Route*::get('/wherein', *function*(){

$values = ['Photon Khan', 'The Ataris'];

**return \App\*Album*::whereIn('artist', $values)->get();**

});

### Where Not In

#### /routes/web.php

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

$values = ['Photon Khan', 'The Ataris'];

**return \App\Album::whereNotIn('artist', $values)->get();**

});

### Where Null

#### /routes/web.php

*Route*::get('/wherenull', *function*(){

**return \App\*Album*::whereNull('artist')->get();**

});

### Where Not Null

#### /routes/web.php

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

**return \App\Album::whereNotNull('artist')->get();**

});

### Order By

#### /routes/web.php

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

**return \App\Album::where('artist', '=', 'Photon Khan')->orderby('year')->get();**

});

### Limit

#### /routes/web.php

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

**return \App\Album::where('artist', '=', 'Photon Khan')->orderby('year')->take(1)->get();**

});

### Skip

#### /routes/web.php

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

**return \App\Album::where('artist', '=', 'Photon Khan')->orderby('year')->take(2)->skip(1)->get();**

});

### Magic Query

#### /routes/web.php

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

**return \App\Album::whereArtist('Photon Khan')->get();**

});

### Query Scopes

#### /app/Album.php

<?php

namespace App;

use Illuminate\Database\Eloquent\Model;

class Album extends Model

{

**public function scopeTriplePeriod($query){**

**return $query->where('title', 'LIKE', '...%');**

**}**

}

#### /routes/web.php

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

**return \App\Album::triplePeriod()->get();**

});

## Eloquent Collections

### Gives all the data

#### /routes/web.php

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

return \App\Album::all();

});

Gives all the data in that table

### All data taken then it is filtered to first

#### /routes/web.php

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

$data = \App\Album::all();

return $data->first();

});

### All data taken then it is filtered to last

#### /routes/web.php

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

$data = \App\Album::all();

return $data->last();

});

### Shift

#### /routes/web.php

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

$data = \App\Album::all();

return $data->shift();

});

It is similar to first but it also removes the first data from the array

### Pop

#### /routes/web.php

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

$data = \App\Album::all();

return $data->pop();

});

It is similar to last but it also removes the last data from the array

### Each

#### /routes/web.php

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

$data = \App\Album::all();

$data->each(function($each\_item){

echo $each\_item->title."<br/>";

});

});

It is similar to foreach closure

### Map

#### /routes/web.php

Route::get('mapcollection', function(){

$data = \App\Album::all();

$data = $data->map(function($each\_data){

return "Great Albums ::".$each\_data->title;

});

echo "<pre>";

var\_dump($data);

});

It is similar to each but it returns new collection

### Filter

#### /routes/web.php

Route::get('filtercollection', function(){

$data = \App\Album::all();

$data = $data->filter(function($each\_data){

if($each\_data->artist == 'Something Corporate'):

return true;

endif;

});

echo "<pre>";

var\_dump($data);

});

Filters the collection

### Sort

#### /routes/web.php

Route::get('sortcollection', function(){

$collection = \App\Album::all();

$collection = $collection->sort(function($a, $b){

$a = $a->year;

$b = $b->year;

if($a == $b):

return 0;

else:

return ($a > $b)? 1 : -1;

endif;

});

echo "<pre>";

$collection->each(function($album){

var\_dump($album->year);

});

});

Sorts the collection

### Reverse

#### /routes/web.php

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

$collection = \App\Album::all();

echo "<pre>";

echo "Normal Loop <br/>";

$collection->each(function($each\_data){

echo $each\_data->title."<br/>";

});

echo "<br/><br/>";

echo "Reverse Loop <br/>";

$reverse = $collection->reverse();

$reverse->each(function($each\_data){

echo $each\_data->title."<br/>";

});

});

Reverses the collection

### Slice

#### /routes/web.php

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

$collection = \App\Album::all();

$sliced\_data = $collection->slice(2,4);

$sliced\_data->each(function($each\_data){

echo $each\_data->title."</br/>";

});

});

Takes out the required data from an array

### Merge

#### /routes/web.php

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

$artist = \App\Album::whereArtist('Photon Khan')->get();

$title = \App\Album::whereTitle('Leaving through the window')->get();

$merged = $artist->merge($title);

$merged->each(function($each\_data){

echo "Artist:: {$each\_data->artist} Title:: {$each\_data->title}<br/>";

});

});

Merges the two arrays.

### Is Empty

#### /routes/web.php

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

$data = \App\Album::whereArtist('Boom')->get();

if($data->isEmpty()):

echo "Life is so empty";

endif;

});

Checks whether the array is empty or not

### Object to Array Conversion

#### /routes/web.php

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

$data = \App\Album::all();

echo "<pre>";

var\_dump($data->toArray());

});

### Object to JSON

#### /routes/web.php

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

$data = \App\Album::all();

echo "<pre>";

var\_dump($data->toJson());

});

### Count

#### /routes/web.php

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

$data = \App\Album::all();

echo "<pre>";

echo $data->count();

});

Give the count of arrays

## Eloquent Relationships

Polymorphic Relations: Also known as many to many relationships

php artisan make:migration create\_tables

### Mirgration

#### /database/…migration…file…

<?php

use Illuminate\Support\Facades\Schema;

use Illuminate\Database\Schema\Blueprint;

use Illuminate\Database\Migrations\Migration;

class CreateTables extends Migration

{

/\*\*

\* Run the migrations.

\*

\* @return void

\*/

public function up()

{

Schema::create('authors', function($table){

$table->increments('id');

$table->string('name', '64');

$table->timestamps();

});

Schema::create('books', function($table){

$table->increments('id');

$table->string('name', 64);

$table->integer('author\_id')->unsigned();

$table->foreign('author\_id')->references('id')->on('authors');

$table->timestamps();

});

Schema::create('readers', function($table){

$table->increments('id');

$table->string('name', 64);

$table->timestamps();

});

Schema::create('book\_reader', function($table){

$table->integer('book\_id')->unsigned();

$table->foreign('book\_id')->reference('id')->on('books');

$table->integer('reader\_id')->unsigned();

$table->foreign('reader\_id')->references('id')->on('readers');

});

}

/\*\*

\* Reverse the migrations.

\*

\* @return void

\*/

public function down()

{

Schema::drop('authors');

Schema::drop('books');

Schema::drop('readers');

Schema::drop('book\_reader');

}

}

### Creating Models

php artisan make:model Book

php artisan make:model Author

php artisan make:model Reader

#### /app/Author.php

namespace App;

use Illuminate\Database\Eloquent\Model;

class Author extends Model

{

public function books(){

return $this->hasMany(Book::class);

}

}

#### /app/Book.php

namespace App;

use Illuminate\Database\Eloquent\Model;

class Book extends Model

{

public function author(){

return $this->belongsTo(Author::class);

}

public function readers(){

return $this->belongsToMany(Reader::class);

}

}

#### /app/Reader.php

namespace App;

use Illuminate\Database\Eloquent\Model;

class Reader extends Model

{

public function books(){

return $this->belongsToMany(Book::class);

}

}

#### /route/web.php

Route::get('relationships', function(){

$author = new \App\Author;

$author->name = 'J.K Rowling';

$author->save();

$book = new \App\Book;

$book->name = "Harry Potter and the Sorcerer's Stone";

$book->author()->associate($author);

$book->save();

return "Successfully Added the Relationship";

});

#### /route/web.php

Route::get('longrelation', function(){

$author = new \App\Author;

$author->name = "Dan Brown";

$author->save();

$book = new \App\Book;

$book->name = "Da Vinci Code";

$book->author()->associate($author);

$book->save();

$reader = new \App\Reader;

$reader->name = 'Photon Khan';

$reader->save();

$reader->books()->save($book);

return "Long Relation Addded";

});

### Adding the relationship

$book→author()→attach(2)

This would associate with the primary key directl

### Removing the relationship

$book→author()→detach(2)

## Validation

#### /route/web.php

Route::resource('/validationform', 'ValidationFormController');

php artisan make:controller ValidationFormController –resource

#### /app/Http/Controllers/ValidationFormController

namespace App\Http\Controllers;

use Illuminate\Http\Request;

use Illuminate\Contracts\View\Factory as View;

class ValidationFormController extends Controller

{

protected $view;

public function \_\_construct(View $view){

$this->view = $view;

}

public function index(View $view){

return $this->view->make('validationform');

}

}

#### /resources/views/validationform.blade.php

<!DOCTYPE html>

<html>

<head>

<meta charset="utf-8" />

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

<title>Validation Form</title>

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

</head>

<body>

<div class = 'container'>

<div class = 'row'>

<div class = 'col'>

<h1> Registration form for our club!</h1>

</div>

<form action = "{{url('/registration')}}" method = "POST">

{{csrf\_field()}}

<label for = 'username'> Username </label><br/>

<input type = 'text' name = 'username'/><br/><br/>

<label for = 'email'> Email </label><br/>

<input type = 'text' name = 'email'/><br/><br/>

<label for = 'password'> Password </label><br/>

<input type = 'password' name = 'password'/><br/><br/>

<label for = 'password\_confirmation'> Password Confirmation </label><br/>

<input type = 'password' name = 'password\_confirmation'/><br/><br/>

<input type = 'submit' value = 'Register'/>

</form>

</div>

</div>

</body>

</html>

#### /routes/web.php

Route::resource('/validationform', 'ValidationFormController');

Route::post('/registration', function(Request $request){

$formData = **$request::all();**

**$rules = [**

**'username' => 'alpha\_num'**

**];**

**$validator = Validator::make($formData, $rules);**

if**($validator->passes()):**

return 'Data was saved.';

endif;

if**($validator->fails()):**

return redirect('/validationform');

endif;

});

### Validator Rules

$rules = [

‘username’ => ‘alphanum|min:3’ ∷ We can use pipe line

‘username’ => [‘alphanum’, ‘min:3’] ∷ We can use array

];

active\_url: ∷ Validation same as checkdnsrr()

after:04/24/16 ∷ Validate date exists after parameter

before:04/25/16 ∷ Validate date before parameter

alpha ∷ Validate alphabets

alpha\_dash ∷ Validates alphabets as well as “/”, “\_”

alpha\_num ∷ Validate alphanumeric characters

between:5,7 ∷ Validate value between 5 to 7

confirm(Not Working) ∷ It can be sued to ensure that another field exists that matches the name of the current field appended with \_confirmation.

date ∷ Validates date

date\_format:d/m/y ∷ Validates the given date format

different:*another\_field* ∷ Makes sure the value given is different from another field

email ∷ Validates email

exists:users, username ∷ User exists in the database table “user” column “usernames”

exists:users,username,role,admin ∷ Checks username column and checks whether role

column has the admin value in it or not

image ∷ Validates whether image is present or not (.bmp, .gif, .jpeg, .png)

mimes:pdf, doc, docx ∷ Validates whether file is in given format or not

in:red,brown,white ∷ Validates whether the field matches the red, brown or white

integer ∷ Validates the field is integer

ip ∷ Validates ip

max:3 ∷ Maximum three characters, numbers or symbols

min:3 ∷ Minimum three characters, numbers or symbols

not\_in:blue,green,pink ∷ Validates the field do not have blue, green or pink

numeric ∷ The validated field containing numeric value

regex:[a-z] ∷ Validation using regular expression

required ∷ Validation checks whether the field is empty or not

required\_if:username, zoidberg ∷ Required if the first parameter of a field matches the second parameter

required\_with:age,height ∷ Validate value if another field exists (age, height)

required\_without:age,height ∷ Validate if another field does not exist

same:age ∷ Validate value is the same as another field

size:8 ∷ Validate length or size of value 🡪 length of the string, comparison mathematically

size of the file in bytes

unique:users,username ∷ Validate value is unique in the database (users table, username column)

unique:users,username,4,3,2,1 ∷ Same as before but this time the additional parameters will be ignored by the unique role

url ∷ Validates URL format

### Validating with Errors

#### /routes/web.php

Route::post('/registration', function(Request $request){

$formData = $request::all();

$rules = [

'username' => 'required|alpha\_num|min:3|max:32',

'email' => 'required|email',

'password' => 'required|confirm|min:3',

];

$validator = Validator::make($formData, $rules);

if($validator->passes()){

return "Validator Passed";

}

if($validator->fails()){

// $errors = $validator->messages();

return **Redirect::to('/validationform')->withErrors($validator);**

}

});

#### /resources/views/validationform.blade.php

<body>

<div class = 'container'>

<div class = 'row'>

<div class = 'col'>

<h1> Registration form for our club!</h1>

</div>

<form action = "{{url('/registration')}}" method = "POST">

{{csrf\_field()}}

<label for = 'username'> Username </label><br/>

<input type = 'text' name = 'username'/><br/>

**@if(($errors->has('username')))**

<small> {{$errors->first('username')}} </samll>

**@endif**

<br/>

<label for = 'email'> Email </label><br/>

<input type = 'text' name = 'email'/><br/>

**@if($errors->has('email'))**

<small>{{$errors->first('email')}}</small>

**@endif**

<br/>

<label for = 'password'> Password </label><br/>

<input type = 'password' name = 'password'/><br/>

**@if($errors->has('password'))**

<small>{{$errors->first('password')}}</small>

**@endif**

<br/>

<label for = 'password\_confirmation'> Password Confirmation </label><br/>

<input type = 'password' name = 'password\_confirmation'/><br/>

**@if($errors->has('password\_confirmation'))**

<small>{{$errors->first('password\_confirmation')}}</small>

**@endif**

<br/>

<input type = 'submit' value = 'Register'/>

</form>

</div>

</div>

</body>

### Gives all the errors

**@foreach($errors->all() as $message)**

**{{$message}}<br/>**

**@endforeach**

### Error Message Formatting (Not Working showing html tags)

#### /resources/views/validationform.blade.php

{{$errors->first('username', '<span class = "error">:message</span>')}}

{{$errors->first('email', '<span class = "error">:message</span>')}}

{{$errors->first('password', '<span class = "error">:message</span>')}}

{{$errors->first('password\_confirmation', '<span class = "error">:message</span>')}}

### Custom Validation Rules

#### /routes/web.php

Validator::extend('awesome', function($field, $value, $params){

return $value == 'awesome';

});

Route::post('/registration', function(Request $request){

$formData = $request::all();

$rules = [

'username' => 'required|alpha\_num|min:3|max:32|awesome',

'email' => 'required|email',

'password' => 'required|min:3',

'password\_confirmation' => 'required|min:3|same:password',

];

$validator = Validator::make($formData, $rules);

if($validator->passes()){

return "Validator Passed";

}

if($validator->fails()){

// $errors = $validator->messages();

return Redirect::to('/validationform')->withErrors($validator);

}

});

### Creating Custom Validation Class

#### /routes/web.php

Validator::extend('awesome', 'App\Validators\CustomValidation@awesome');

#### /app/Validators/CustomValidation.php

<?php

namespace App\Validators;

class CustomValidation{

public function awesome($field, $value, $params){

return $value == 'awesome';

}

}

### Custom Validation Messages

#### /routes/web.php

Validator::extend('awesome', 'App\Validators\CustomValidation@awesome');

Route::post('/registration', function(Request $request){

$formData = $request::all();

$rules = [

'username' => 'required|alpha\_num|min:3|max:32|awesome',

'email' => 'required|email',

'password' => 'required|min:3',

'password\_confirmation' => 'required|min:3|same:password',

];

**$messages = [**

**'min' => 'Yo dawg, you need more to write!',**

**'required' => 'Bro, are you kidding me?'**

**];**

**$validator = Validator::make($formData, $rules, $messages);**

if($validator->passes()){

return "Validator Passed";

}

if($validator->fails()){

// $errors = $validator->messages();

return Redirect::to('/validationform')->withErrors($validator);

}

});

### Custom Validation Messages for specific fields

$messages = [

**'username.min' => "Nice try Bro!",**

'min' => 'Yo dawg, you need more to write!',

'required' => 'Bro, are you kidding me?'

];

$validator = Validator::make($formData, $rules, $messages);

## Events

php artisan make:event ActionDone

php artisan make:listener ThingsToDoAfterEventWasFired –event=”ActionDone”

or we can create EventServiceProvider then php artisan event:generate

First, we need to register the events

### Creating Events Using Class

#### /app/Providers/EventServiceProvider.php

namespace App\Providers;

use Illuminate\Support\Facades\Event;

use Illuminate\Foundation\Support\Providers\EventServiceProvider as ServiceProvider;

class EventServiceProvider extends ServiceProvider

{

/\*\*

\* The event listener mappings for the application.

\*

\* @var array

\*/

protected $listen = [

'App\Events\Event' => [

'App\Listeners\EventListener',

],

**'App\Events\ActionDone' => [**

**'App\Listeners\ThingsToDoAfterEventWasFired'**

**],**

];

/\*\*

\* Register any events for your application.

\*

\* @return void

\*/

public function boot()

{

parent::boot();

//

}

}

However, we may also register Closure based events manually in the boot method of our ***EventServiceProvider***

#### /routes/web.php

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

**return event(new \App\Events\ActionDone(5,3));**

});

#### /app/Events/ActionDone.php

<?php

namespace App\Events;

use Illuminate\Broadcasting\Channel;

use Illuminate\Queue\SerializesModels;

use Illuminate\Broadcasting\PrivateChannel;

use Illuminate\Broadcasting\PresenceChannel;

use Illuminate\Foundation\Events\Dispatchable;

use Illuminate\Broadcasting\InteractsWithSockets;

use Illuminate\Contracts\Broadcasting\ShouldBroadcast;

class ActionDone

{

use Dispatchable, InteractsWithSockets, SerializesModels;

**public $a;**

**public $b;**

/\*\*

\* Create a new event instance.

\*

\* @return void

\*/

**public function \_\_construct($a, $b)**

**{**

**$this->a = $a;**

**$this->b = $b;**

**}**

/\*\*

\* Get the channels the event should broadcast on.

\*

\* @return \Illuminate\Broadcasting\Channel|array

\*/

public function broadcastOn()

{

return new PrivateChannel('channel-name');

}

}

#### /app/Listeners/ThingsToDoAfterEventWasFired.php

<?php

namespace App\Listeners;

use App\Events\ActionDone;

use Illuminate\Queue\InteractsWithQueue;

use Illuminate\Contracts\Queue\ShouldQueue;

class ThingsToDoAfterEventWasFired

{

/\*\*

\* Create the event listener.

\*

\* @return void

\*/

public function \_\_construct()

{

//

}

/\*\*

\* Handle the event.

\*

\* @param ActionDone $event

\* @return void

\*/

**public function handle(ActionDone $event)**

**{**

**if($event->a > $event->b):**

**return true;**

**endif;**

**echo "b is greater";**

**return false;**

**}**

}

### Simple Event Creation

#### /routes/web.php

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

$first = 4;

$second = 7;

**return Event::fire('boom', [$first, $second]);**

});

**Event::listen('boom', function($first, $second){**

**if($first > $second){**

**return "{$first} is greater than {$second}";**

**}else{**

**return "{$second} is greater than {$first}";**

**}**

**});**

### Simple Event with Multiple Listeners

#### /routes/web.php

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

$first = 4;

$second = 7;

return Event::fire('boom', [$first, $second]);

});

Event::listen('boom', function($first, $second){

if($first > $second){

return "{$first} is greater than {$second}";

}else{

return "{$second} is greater than {$first}";

}

}, 1);

Event::listen('boom', function($first, $second){

if($first > $second){

return "{$first} is the best";

}else{

return "{$second} is the best";

}

}, 3);

Event::listen('boom', function($first, $second){

if($first > $second){

return "{$first} is powerful";

}else{

return "{$second} is powerful";

}

}, 5);

### Subscriber

#### /routes/web.php

**Event::subscribe(new \App\Listeners\MyListener);**

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

Event::fire('first.event', "Firing One");

Event::fire('second.event', "Firing Two");

});

#### /App/Listeners/MyListener.php

namespace App\Listeners;

use Illuminate\Queue\InteractsWithQueue;

use Illuminate\Contracts\Queue\ShouldQueue;

class MyListener

{

/\*\*

\* Create the event listener.

\*

\* @return void

\*/

**public function \_\_construct()**

**{**

**//**

**}**

**public function firstListener($event){**

**echo $event;**

**}**

**public function secondListener($event){**

**echo $event;**

**}**

/\*\*

\* Subscribe the event.

\*

\* @param object $event

\* @return void

\*/

**public function subscribe($event)**

**{**

**$event->listen('first.event', "\App\Listeners\myListener@firstListener");**

**$event->listen('second.event', "\App\Listeners\myListener@SecondListener");**

**}**

}

Array of variables in the Event∷Fire do not work

### Global Events

Passing a closure too the **App∷after()** method, then the Closure will be executed after the framework has completed its request-response cycle.

**App∷after(function($request, $response){**

**$response→headers→set(‘Access-Control-Allow-Origin’, ‘\*’);**

**});**

Similarly, there is also **App∷before**

## Dependency Injection

## Terminology Building Block

json\_encode($arr) ∷ Converts array into json format

json\_decode($json, bool) ∷ Converts json format to object(false) or array (true)

Route::get('/home', function(){ return “Damn”;}) ∷ Simple Route

Route∷get(‘/home’, $closure) ∷ Anonymous Function in Route

Route∷get(‘/{{random}}’, function($random){return $random}); ∷ Any route

Route∷get(‘/{{random?}}’, function($random){return $random}); ∷ Any route or none

View∷make(‘simple’) ∷ Reads the html/blade file

Redirect∷to(‘/somewhere’) ∷ Redirects the page to ‘/somewhere’

$response = Response∷make(‘Damn Son’, 200) ∷ Creates a custom response

$response→headers()→set(‘Content-Type’, ‘text/html’) ∷ Sets the content type

$response→setTtl(value) ∷ Sets the shared cache value living time

Response∷json($array) ∷ Prints out the json data

Response∷download($file, $alternative\_name, [$status, $header]) ∷ Download file

{{$variable}} ∷ Allows us to access the php variable through blade template

{{date(‘m/d/y’)}} ∷ Allows us to use function to access the current date in html

@if@elseif@else@endif ∷ If/else statement in blade html

@foreach($data as $key => $value)@endforeach ∷ For each statement in blade html

@while($statement)@endwhile ∷ While loop in blade html

@for($i = 0; $i < count($var); $i++)@endfor ∷ For loop in blade html

@unless($statement)@endunless ∷ Unless in blade html

@include(‘file-name’) ∷ Allows us to include other html file in blade

@yield(‘some-var’) ∷ Allows us to include a snippet extending from parent to child

@extends(‘folder.parent\_file’) ∷ Allows child to extend the parent file

@section(‘some-var’)@show ∷ Same like yield but by default it shows something

@section(‘some-var’)@stop ∷ Substitute the element in parents with child

@section(‘some-var’)@parent@stop ∷ Adds the elements of both parents and child

{{-- --}} ∷ Proper way to comment in blade html

@{{}} ∷ Some JavaScript libraries uses this tag Ex: Angular JS, to remove conflict

{{url(‘/your-url’}} ∷ In forms, this allows to forward the message to that page

{{csrf\_field()}} ∷ A hidden token allows to have a distinct form

Request∷all() ∷ Get all the data send from the form

Request∷only(‘var’) ∷ Get selected the data

Request∷except(‘var’) ∷ Get all data except the selected ones

Request∷has(‘var’) ∷ Checks whether this variable is present or not

Request∷flash(),Redirect∷to(‘some-url’),Request∷old() ∷ Get the value from redirected form

Request∷flashOnly(‘var’) ∷ Flash only certain values

Request∷flashExcept(‘var’) ∷ Flash everything except the one chosen

Redirect∷to(‘your-url’)🡪withInput() ∷ Alternative to Flash

Redirect∷to(‘your-url’)🡪withInput(Request∷only(‘var’)) ∷ Alternative to Flash Only

Redirect∷to(‘your-url’)🡪withInput(Request∷except(‘var’)) ∷ Alternative to Flash Except

Request∷file(‘file-name’) ∷ Uploaded file details

Request∷file(‘file-name’)🡪getClientOriginalName() ∷ Get the name of the file

Request∷file(‘file-name’)🡪getFileName() ∷ Get the temporary file name

Request∷file(‘file-name’)🡪getClientSize() ∷ Get the size of the file

Request∷file(‘file-name’)🡪getMimeType() ∷ Get the mime type of the file

Request∷file(‘file-name’)🡪guessExtension() ∷ Get the file extension

Request∷file(‘file-name’)🡪getRealPath() ∷ Get the temporary file path

Request∷file(‘file-name’)🡪move(‘destination\_path’, $desired\_name) ∷ Move uploaded file

Cookie∷make(‘name’, ‘value’, ‘expiry’) ∷ setcookie(‘name’, ‘value’, ‘expiry’)

Cookie∷get(‘name’) ∷ Get the cookie value, alternative to $\_COOKIE[‘name’]

Cookie∷forever(‘name’, ‘value’) ∷ Cookie with no expiry date

Cookie∷has(‘name’) ∷ Does the named cookie exist

Cookie∷forget(‘name’) ∷ Deletes the cookie

Redirect∷to(‘url’)🡪withCookie(Cookie∷make(‘name’, ‘value’, ‘expiry’) ∷ Redirect with cookie

Route∷get(‘/make/a/shortcut’, [‘as’ => “shortcut\_name”, $closure) ∷ Shortcuts to route

Redirect∷route(‘shortcut-name’) ∷ Similar to Redirects∷to(‘url’) but used for shortcuts

{{route(‘shortcut\_name’)}} ∷ Accessing the shortcut name

Route∷CurrentRouteName() ∷ Get the name of the shortcut route name

Route∷get(‘url’, [‘as’ => “shortcut’, “uses” => “controller@method”]) ∷ Advanced Routing

Route∷get(‘url/{{random}}’, function($random){})🡪where(‘random’, ‘RegExp’)

Route∷group([], Route∷get(‘url’, function(){}, … , ….) ∷ Multiple Routes in one group

Route∷group([‘prefix’ => ‘some\_prefix’], Route∷get(‘url’, function(){}, …, )

Route∷group([‘domain’ => ‘some\_prefix’], Route∷get(‘url’, function(){}, …, )

Route∷resource(‘testing’, ‘TestingController’) ∷ Alternative to group names

<URL::current()> ∷ Get current URL

<URL::full()> ∷ It is same as current() but it includes the request data

<URL::previous()> ∷ Get the previous URL

[URL::to(*route*, *array, boolean*)](URL::to(route,%20array,%20boolean)) ∷ Generated URL

[URL::secure(*route*)](URL::secure(route)) ∷ Secure Route (HTTPS)

[URL::route(*shortcut)*](URL::route(shortcut)) :: Shortcut Route Reveal

<URL::action(controller@method)> ∷ Gives the URL of the controller

[URL::asset(route, boolean)](URL::asset(route,%20boolean)) ∷ Get the location of the CSS, JS Files

[URL::secureAsset(route)](URL::secureAsset(route,%20boolean)) ∷ Secure the asset

Schema∷create(‘table-name’, closure)

Schema∷rename(‘users’, ‘customers’); ∷ Renaming the table

Schema∷table(‘table-name’, closure); ∷ Modify the table

Schema∷drop(‘table-name’) ∷ Delete the table

Schema∷dropIfExists(‘table-name’) ∷ Deletes the table if exists

Schema∷connection(‘mysql’)→create(‘table-name’, closure)

Schema∷hasTable(‘table-name’) ∷ Checks whether the table exists or not

Schema∷hasColumn(‘table-column) ∷ Checks if the columns exist or not

## Command Lines

composer global require "laravel/installer"

Global Installer Laravel

php artisan serve

Laravel Default server (8000)

php artisan serve --port=8080

Laravel Server at 8080 (change to)

php artisan make:controller NameController –-resource

Create controller using the artisan command line

php artisan list

Gives the list of all the commands

php artisan make:migration create\_users\_table

Migration are some PHP scripts that are used to change the structure of content of our database

php artisan make:migration create\_users\_table --create=”users”

Migration are some PHP scripts that are used to change the structure of content of our database

php artisan make:migration create\_users\_table –-path=”app/migs”

Change the saving destination of the migration

php artisan migrate

Install the migrations

php artisan migrate –database=mysql

For different database connection

php artisan migrate –pretend

To see what the outcome may be

php artisan make:model *model-name*

Creating models in laravel