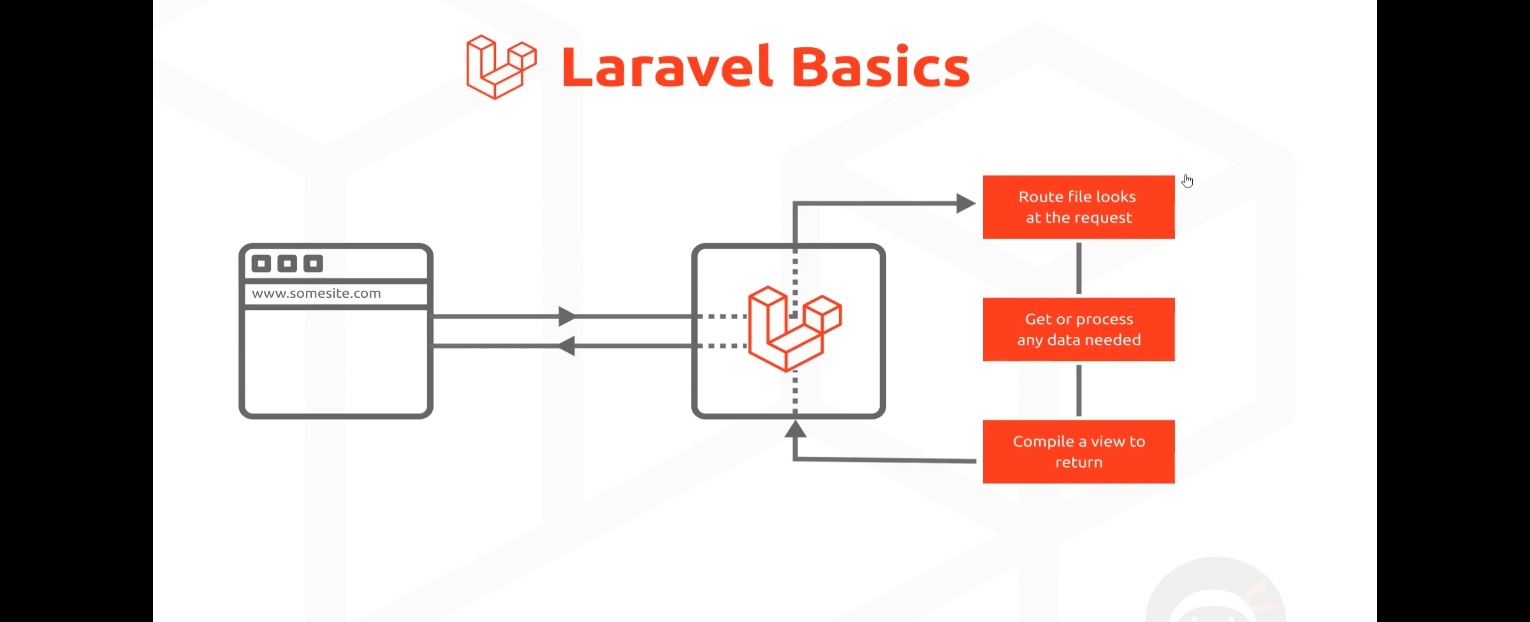
Suraj Khot

selfemloyed  jangamwadi

mY Laravel study

#1 25-feb-24

**#2 Laravel basic - #3 vid** **26-feb-24**



*//routes>web.php*

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

*return* view('welcome');

});

To run web.php file

**Step 1**-open terminal in current directory>>>>>>>>>>>>>>>>>>>>>>>>>

Asus@ASUS-VIVOBOOK-PRO-15 MINGW64 /d/PHP Workspace/Laravel

$ cd pizzahouse

Asus@ASUS-VIVOBOOK-PRO-15 MINGW64 /d/PHP Workspace/Laravel/pizzahouse

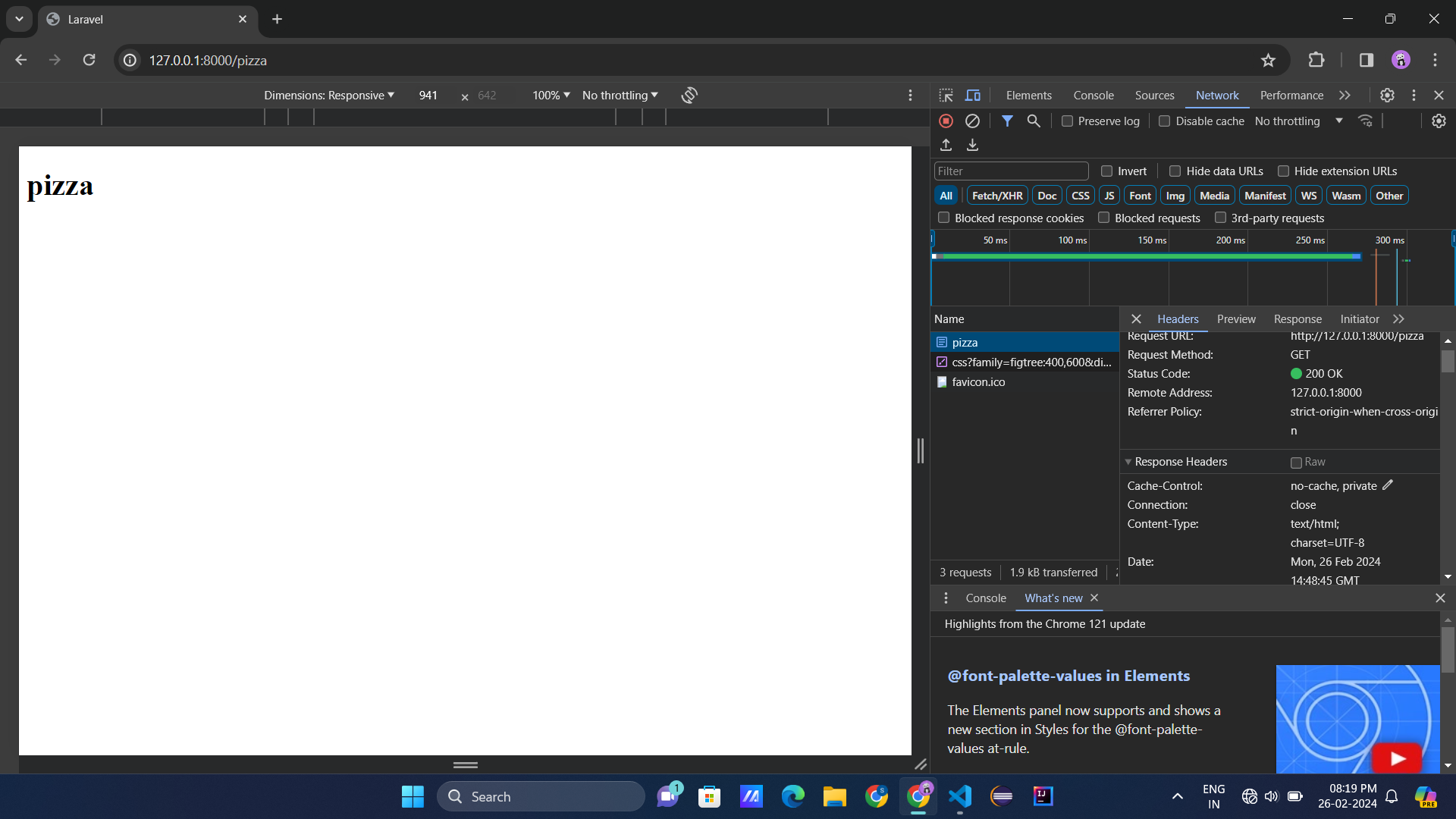
$ php artisan serve

INFO Server running on [http://127.0.0.1:8000].

Press Ctrl+C to stop the server

**Step 2-**then just visit the link and you see your project

**Note:do not forget to close a sever>>by ctrl+c**



*//pizza blade*

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

*//it will return Content-Type:txt/html*

*//return view('pizza');*

*//it will return Content-Type:application/json*

*//return ['name'=>'veg pizzas!','base'=>'classic'];*

});

Means we can route a web application to view using blade

Also it has types as txt/html or application/json.

#4 vid- passing data to views

[key => value] as an pair

{{ }} is allows to use a variable or to print

*/pizza blade*

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

    $arr=[

        'type'=>'hawaiian',

        'base'=>'chessy base',

        'price'=>'50 Rs'

    ];

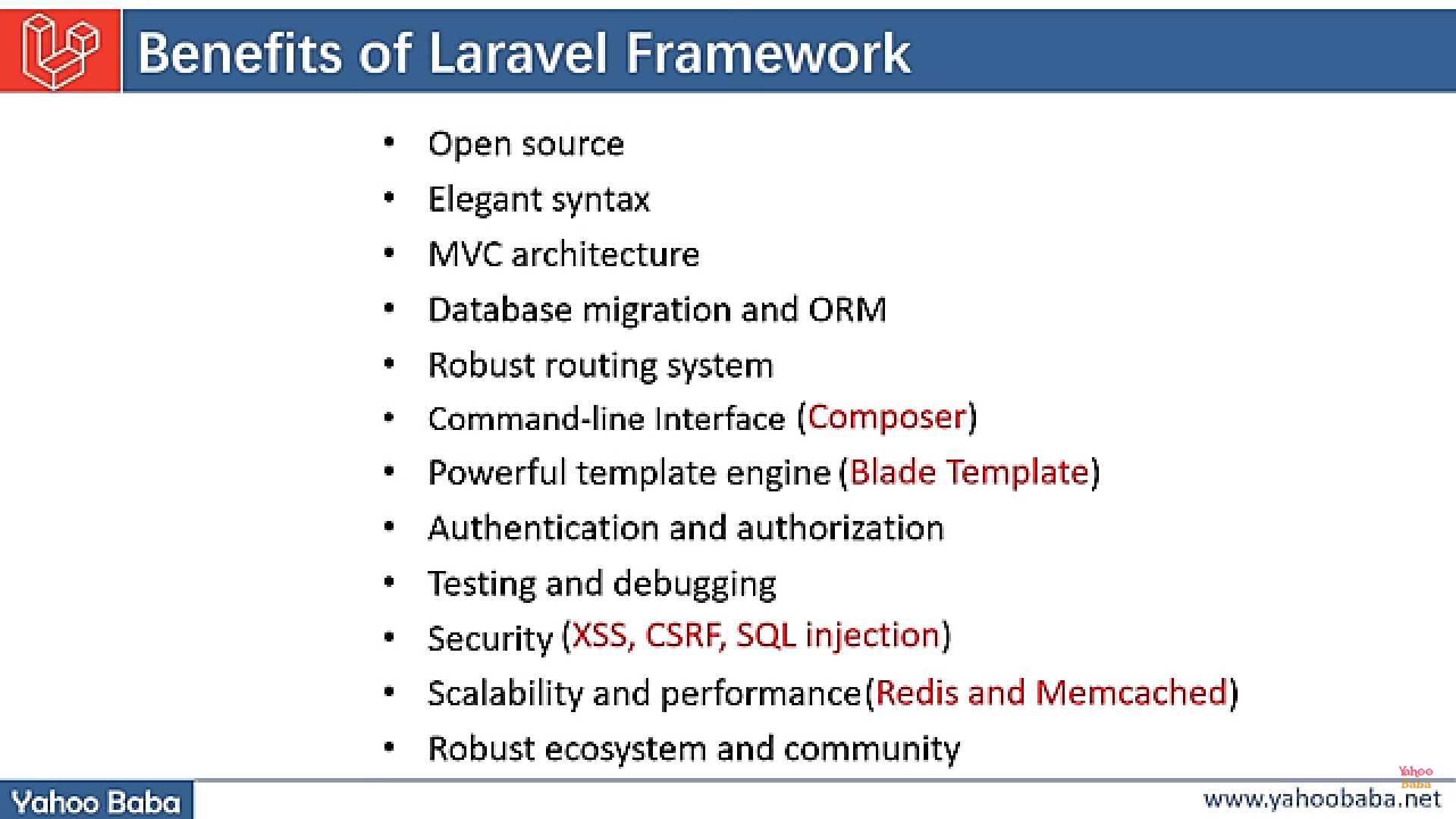
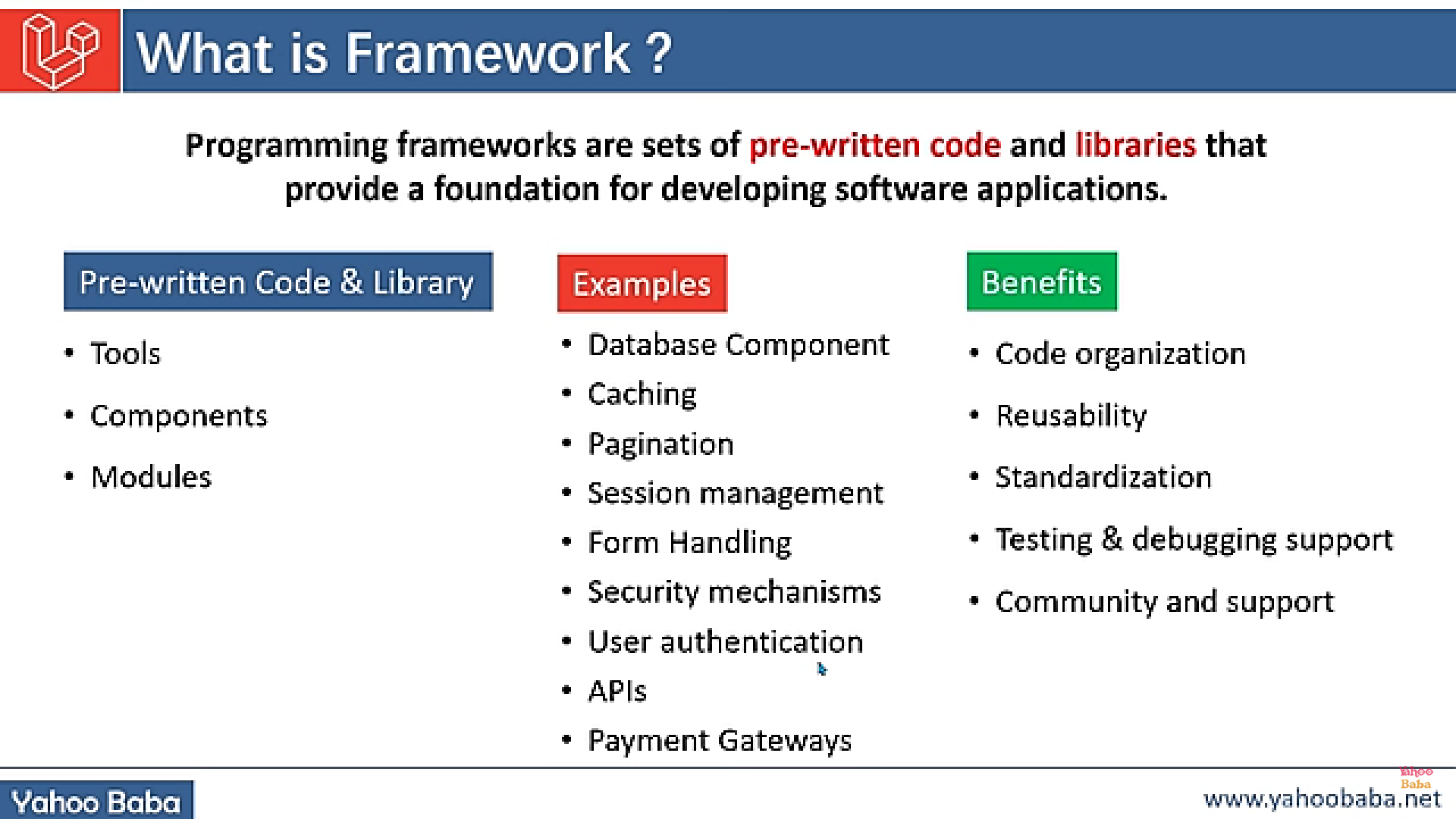
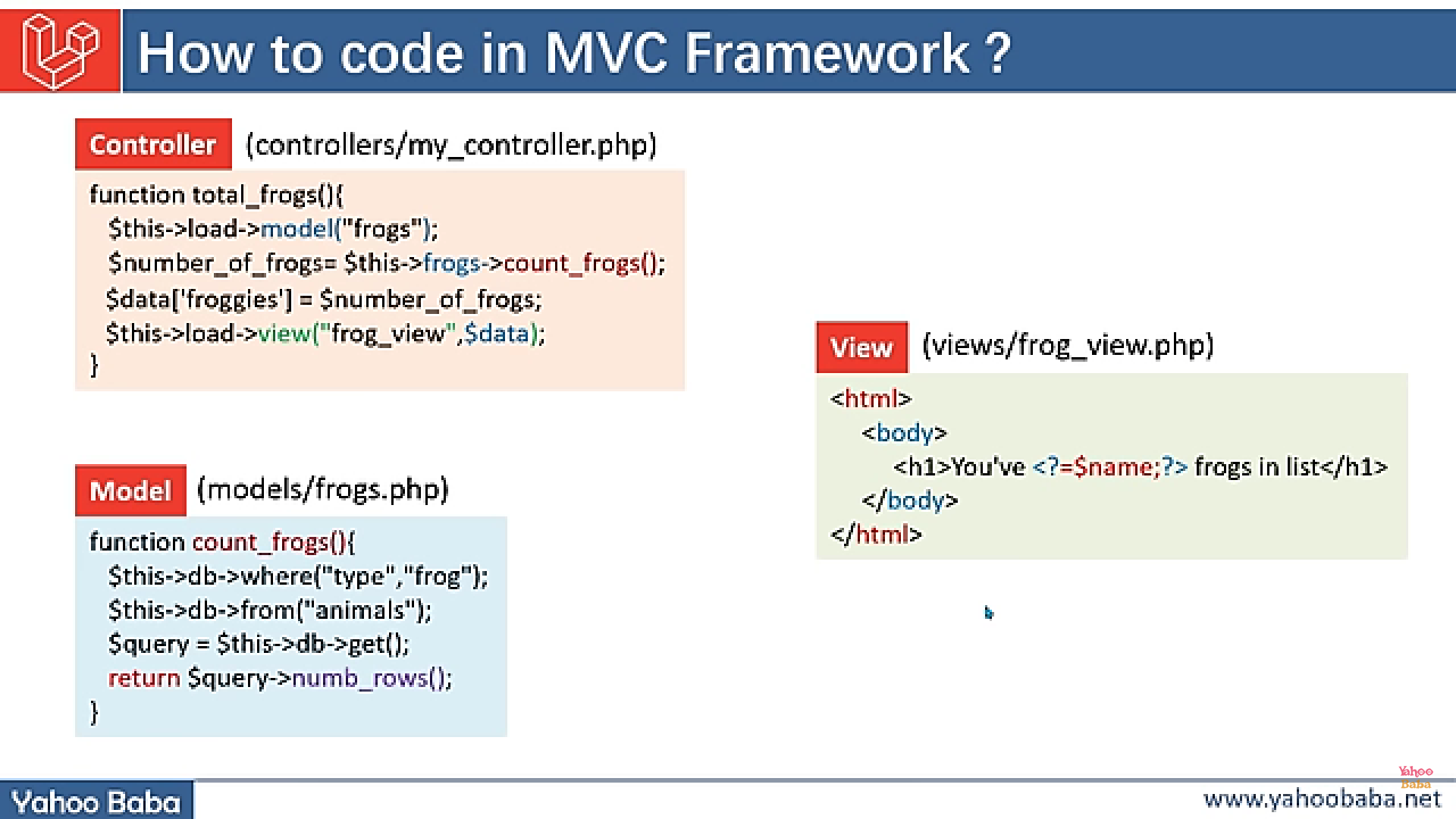
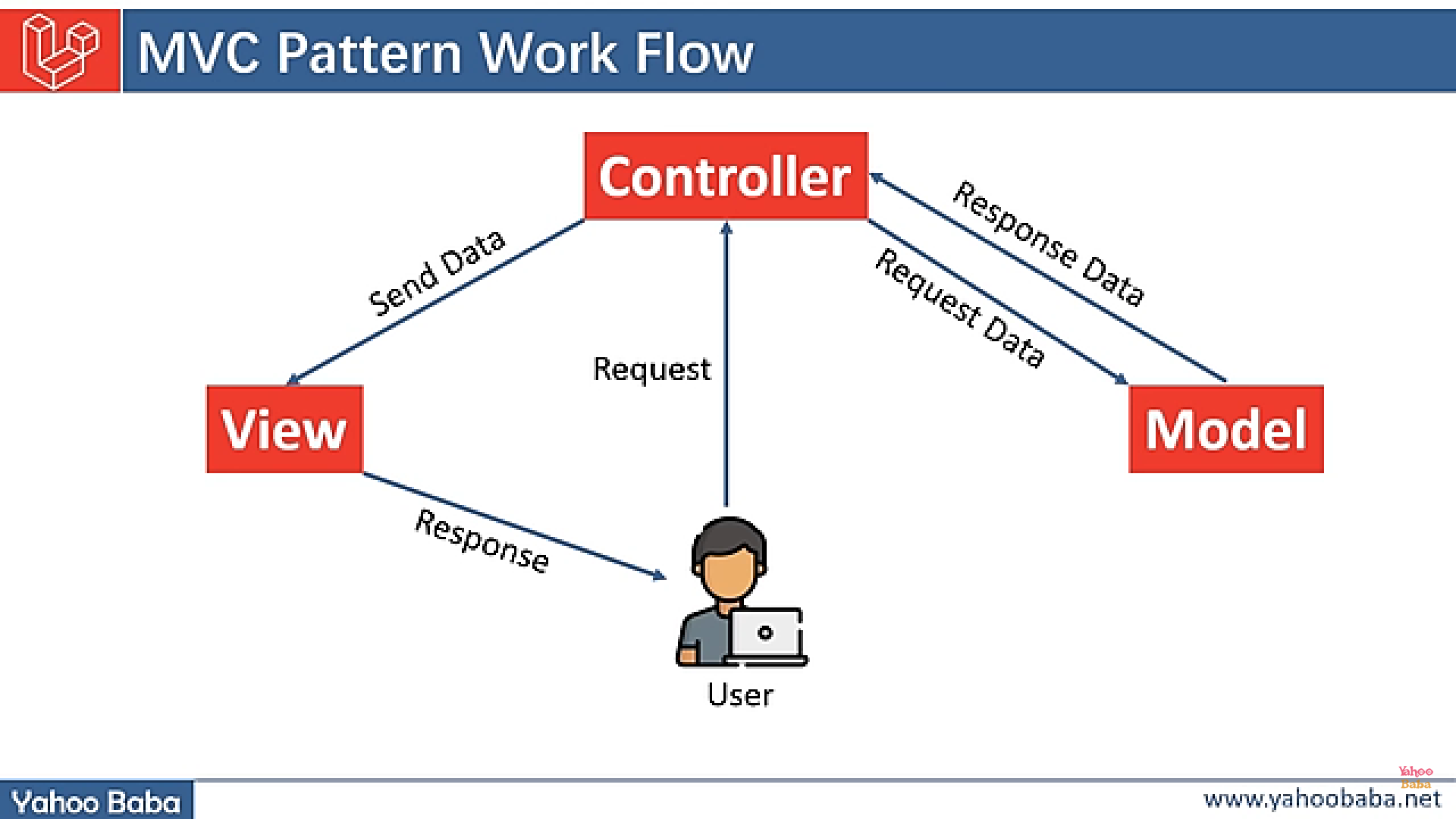
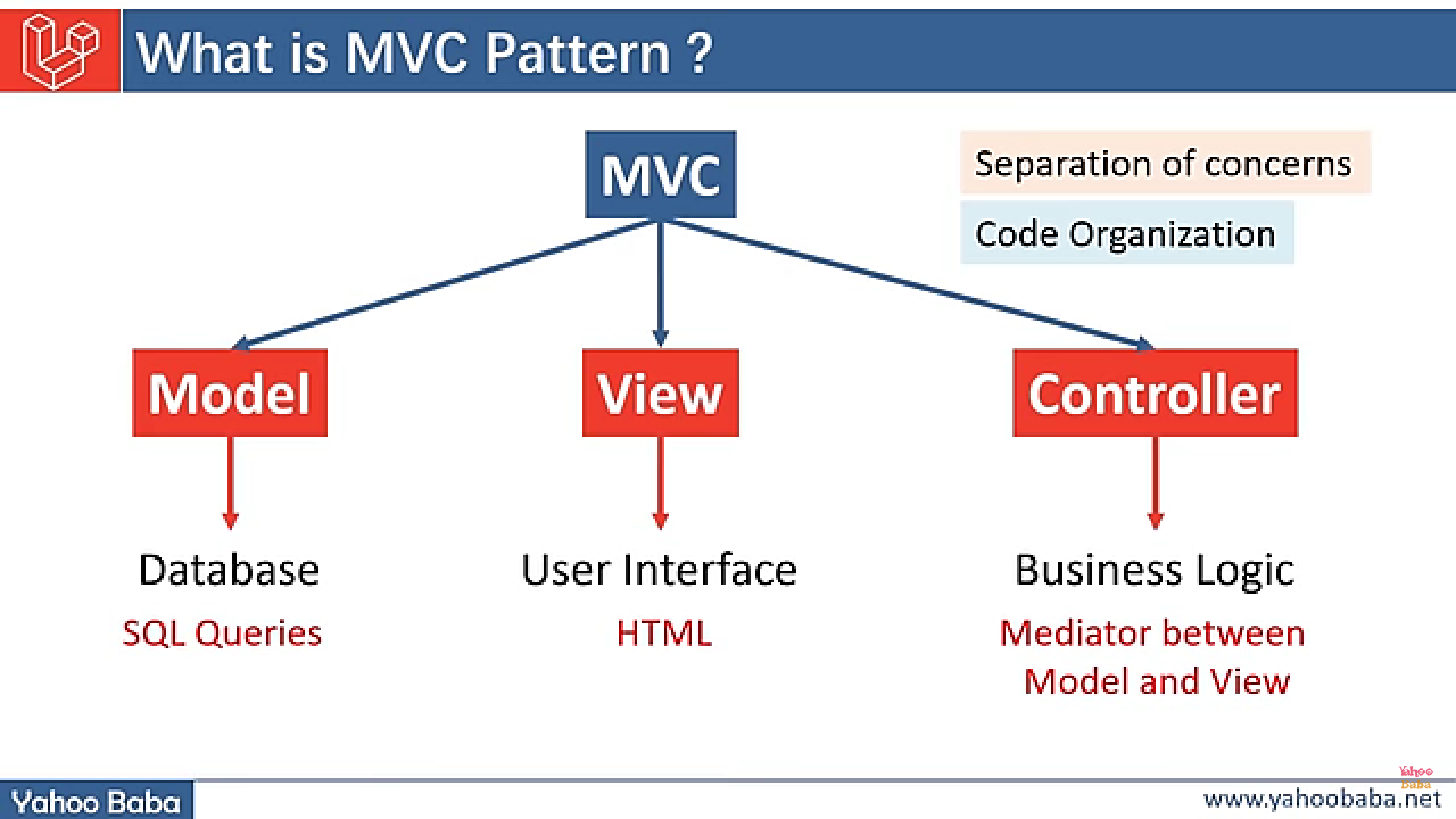
*return* view('pizza',$arr);

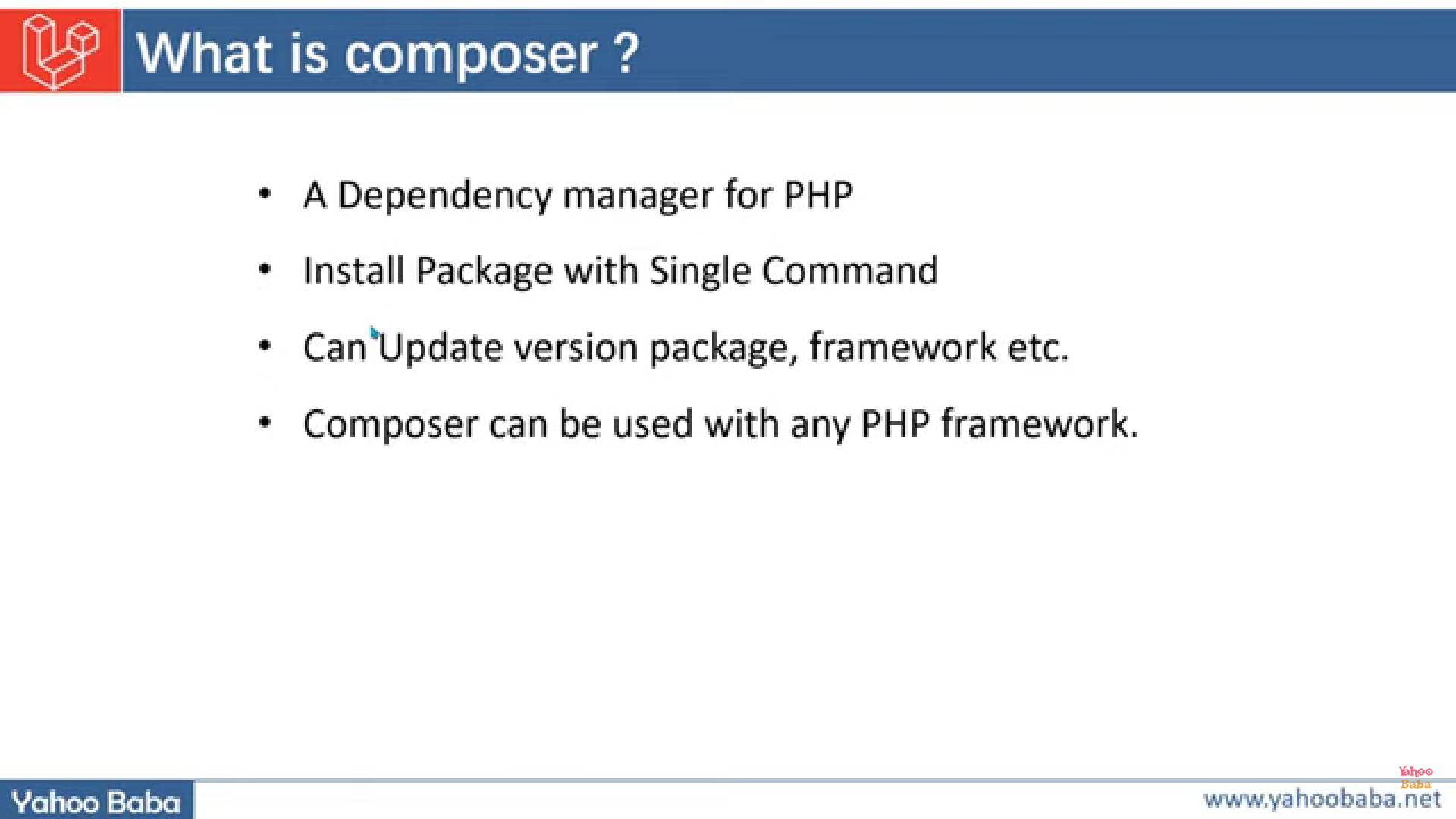
});

*//pizza view*

<p>{{$type}}  -  {{$base}}  -  {{$price}}</p>

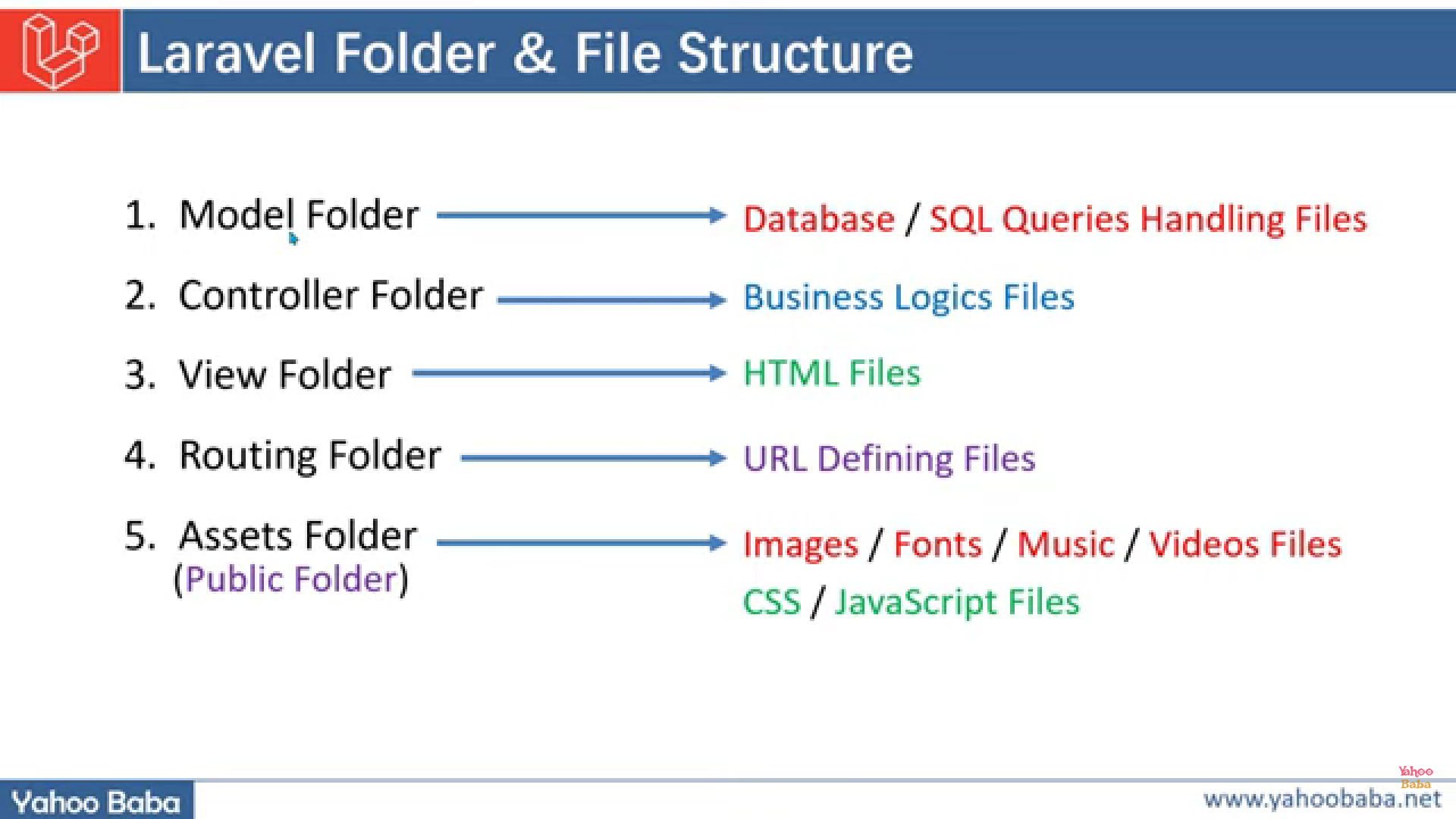
14 March MVC yahho baba yt







15 March 2024 File Format and All



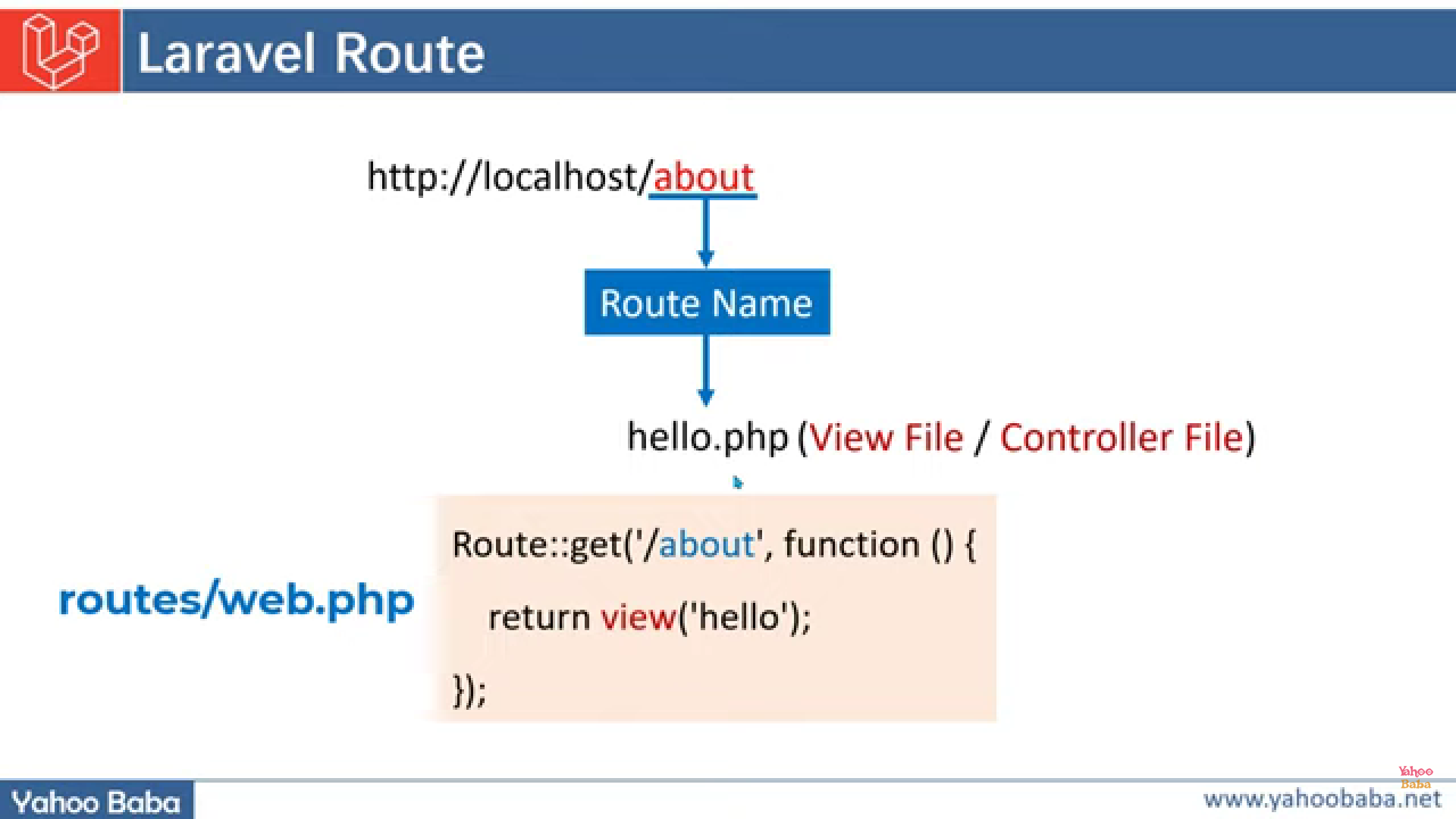
**Model =** app/Models

**Controller**=app/http/controllers/controller.php

**View** =resources/views/welcome.blade.php

**Assets** (**public**)=public/index.php

**Routing** =routes/web.php/



Laravel Route defining:

//way 1:

Route::get('route\_name',function(){

return view(view\_name);

});

//way 2:

Route::view('route\_name','view\_name');

**Example**

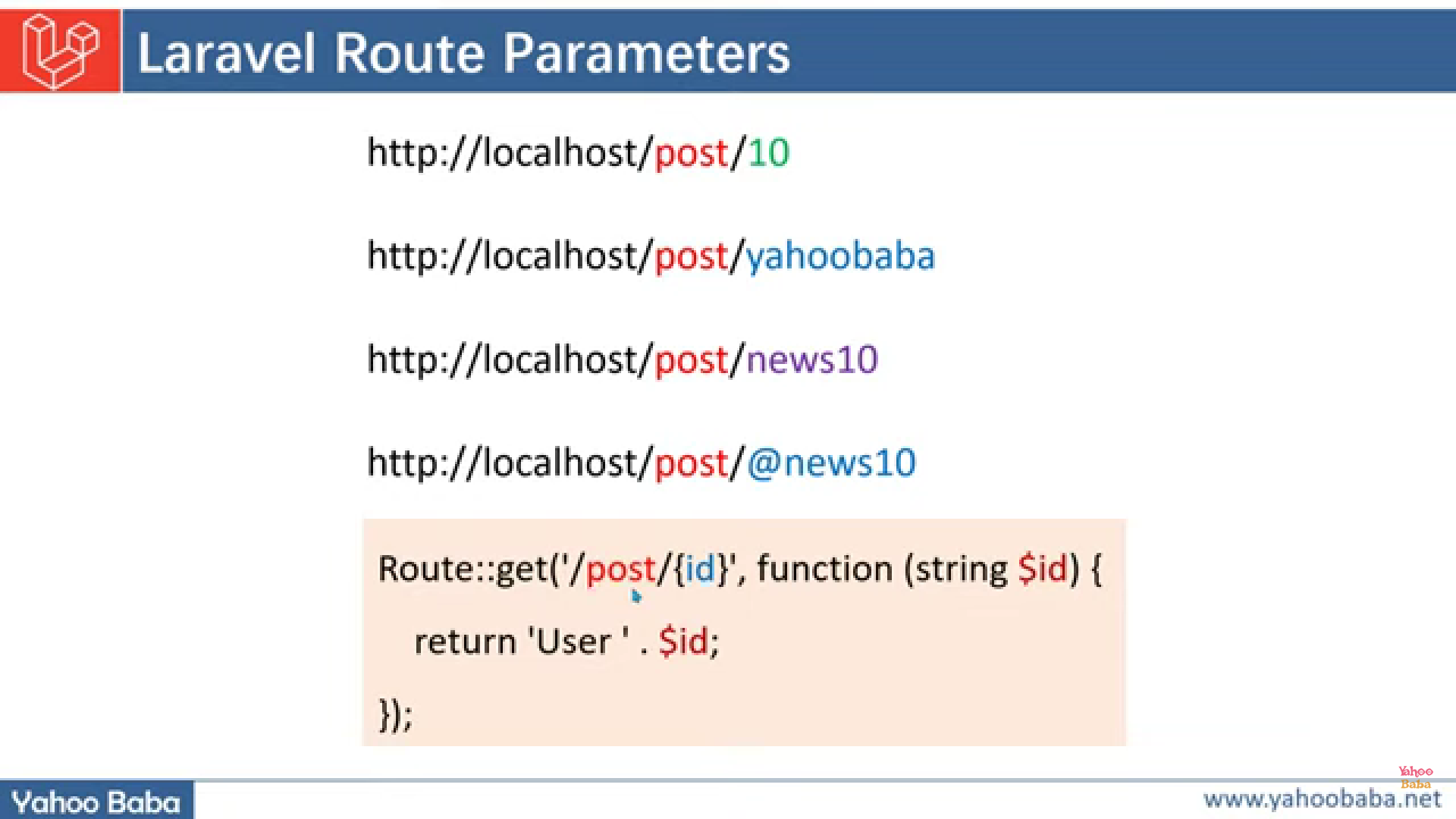
Route::get('/post',function(){

*return* view('post');

});

Route::view('/post','post');

Laravel Route Parameters:



Example of passing parameter in routes

*//passing parameters in routes*

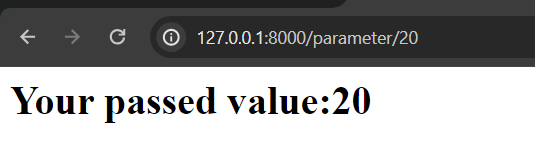
Route::get('/parameter/{id}',function(String $id){

*return* "<h1>Your passed value:". $id ."</h1>";

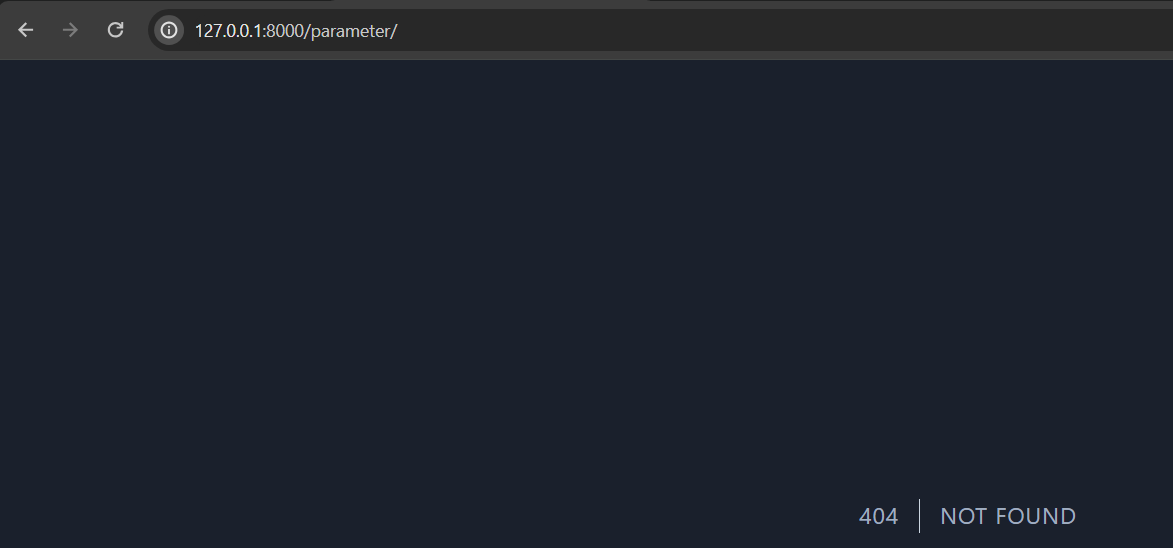
});

o/p

1.if we insert the value



2.if we forgot to insert value



To overcome this problem we create a null or with value route

*//passing parameters in routes way2 : if we forgot to give id then also pgm runs*

Route::get('/WithOrNonparameter/{id?}',function(String $id = null){

*if*($id){

*return* "<h1>Your passed value:". $id ."</h1>";

    }

*else*{

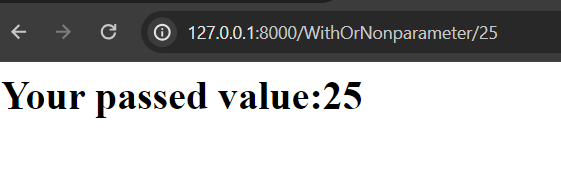
*return* "<h1>you not passed any id</h1>";

    }

});

o/p

1.



2.



Laravel Route multiple parameters

*//passing multiple parameters in route*

Route::get('/FirstPara/{id1?}/SecPara/{id2?}',function($id1=null,$id2=null){

*if*($id1&&$id2){

        echo "first parameter: $id1 <br> secound paramater: $id2";

    }

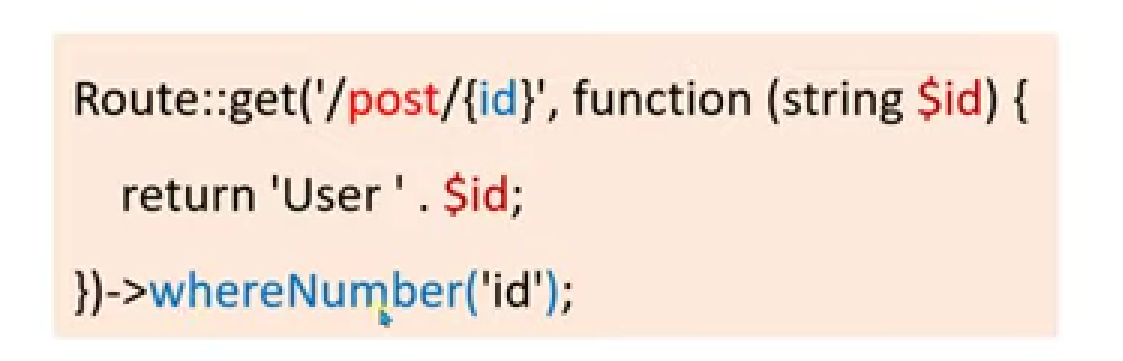
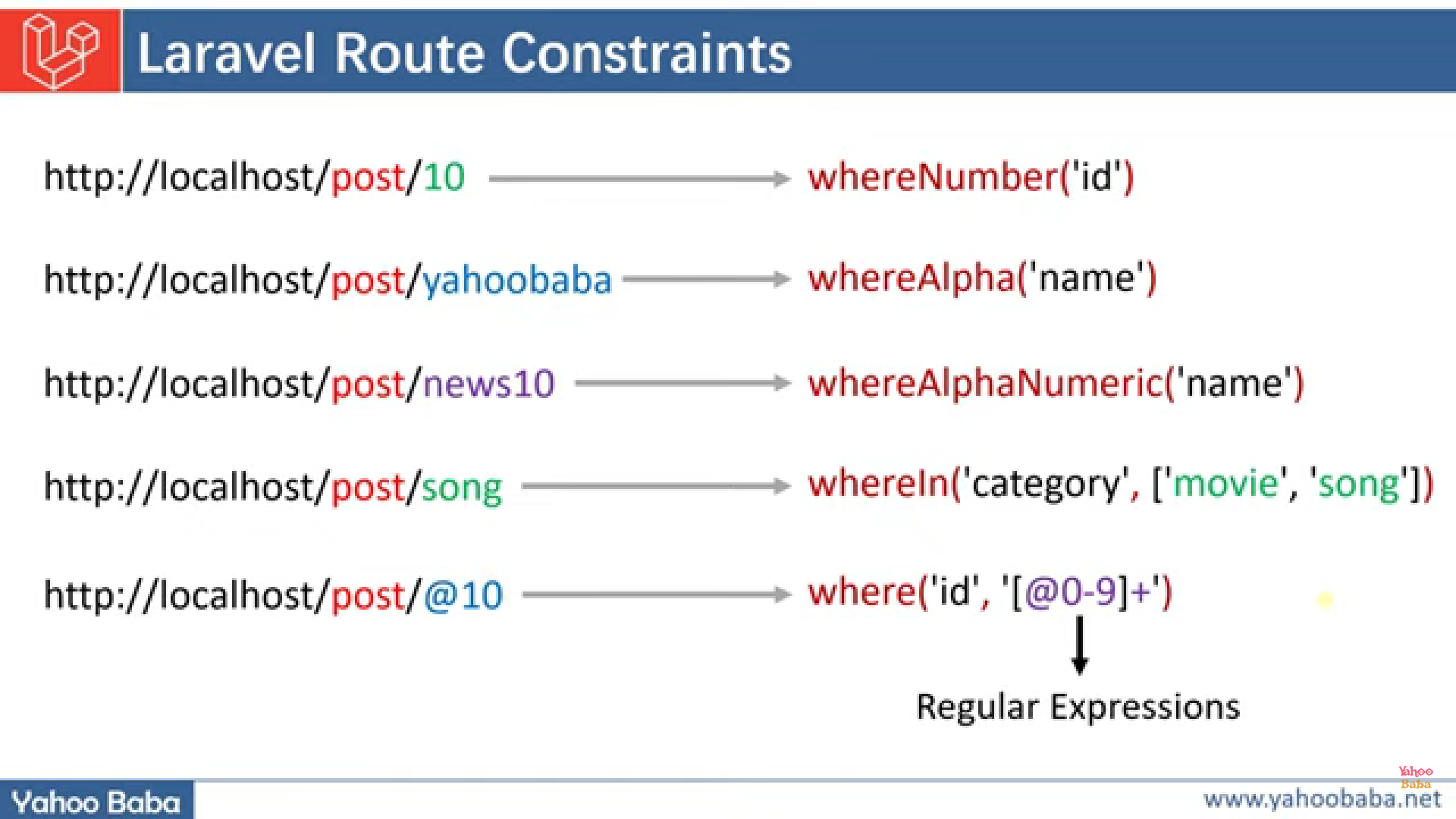
*else*{

        echo "you are not passing any parameter";

    }

});

16 March Laravel Route Constrains:



Example:

*//parameter constraints*

*//1.passing only number*

Route::get('/insertNum/{id}',function(string $id){

    echo "your inserted number is: $id";

})->whereNumber('id');

*//2.passing only string*

Route::get('/insertStr/{id}',function(string $id){

    echo "your inserted string is: $id";

})->whereAlpha('id');

*//3.passing only string+number*

Route::get('/insertNumStr/{id}',function(string $id){

    echo "your inserted number+string is: $id";

})->whereAlphaNumeric('id');

*//4.with custom category*

Route::get('/insertCustom/{id}',function(string $id){

    echo "your inserted my custom value is $id";

})->whereIn('id',["rowdy"]);

*//5.custom digit number*

Route::get('/insertCustomDigit/{id}',function(string $id){

    echo "your inserted my custom value is $id";

})->where('id','[0-9]+');

*//5.custom alpha*

Route::get('/insertCustomAlpha/{id}',function(string $id){

    echo "your inserted my custom value is $id";

})->where('id','[a-zA-Z]+');

Laravel Route multi parameterized Constrains

**Ex1:**

Route::get('/post/{id}/comment/{commentid}', function (string $id, string $comment) {

*if*($id) {

*return* "<h1>Post ID : " . $id . "& Comment :" . $comment."</h1>";

}*else*{

*return* "<h1>No ID Found</h1>";

}

})->where('id', '[0-9]+')->whereAlpha('commentid');

Ex2:

Route::get('/one/{a?}/two/{b?}',function(string $a=null,string $b=null){

*if*($a){

*return* "<h1>".$a." and ".$b."</h1>";

    }

*else*{

*return* "<h1>please enter parameters</h1>";

    }

*// })->whereAlpha('a')->whereNumber('b');*

*// })->whereAlpha('a')->whereAlphaNumeric('b');*

*// })->whereAlpha('a')->where('b','[a-zA-z]+');*

*// })->whereAlpha('a')->whereNumber('b','[0-9]+');*

*// })->whereAlpha('a')->whereIn('b',["suraj","khot"]);*

})->whereAlpha('a')->whereIn('b',["suraj",20,-5]);

Code redirection in case we change route name but user want to access by its old name:

*//redirection*

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

*return* view('carrer');

});

*//synatax for redirecting password\_get\_info*

*// Route:: redirect('old route name','new route name',HTTP status codes 3xx code);*

Route::redirect('carrer','test');

**HTTP status codes 3xx**[[edit](https://en.wikipedia.org/w/index.php?title=URL_redirection&action=edit&section=15)]

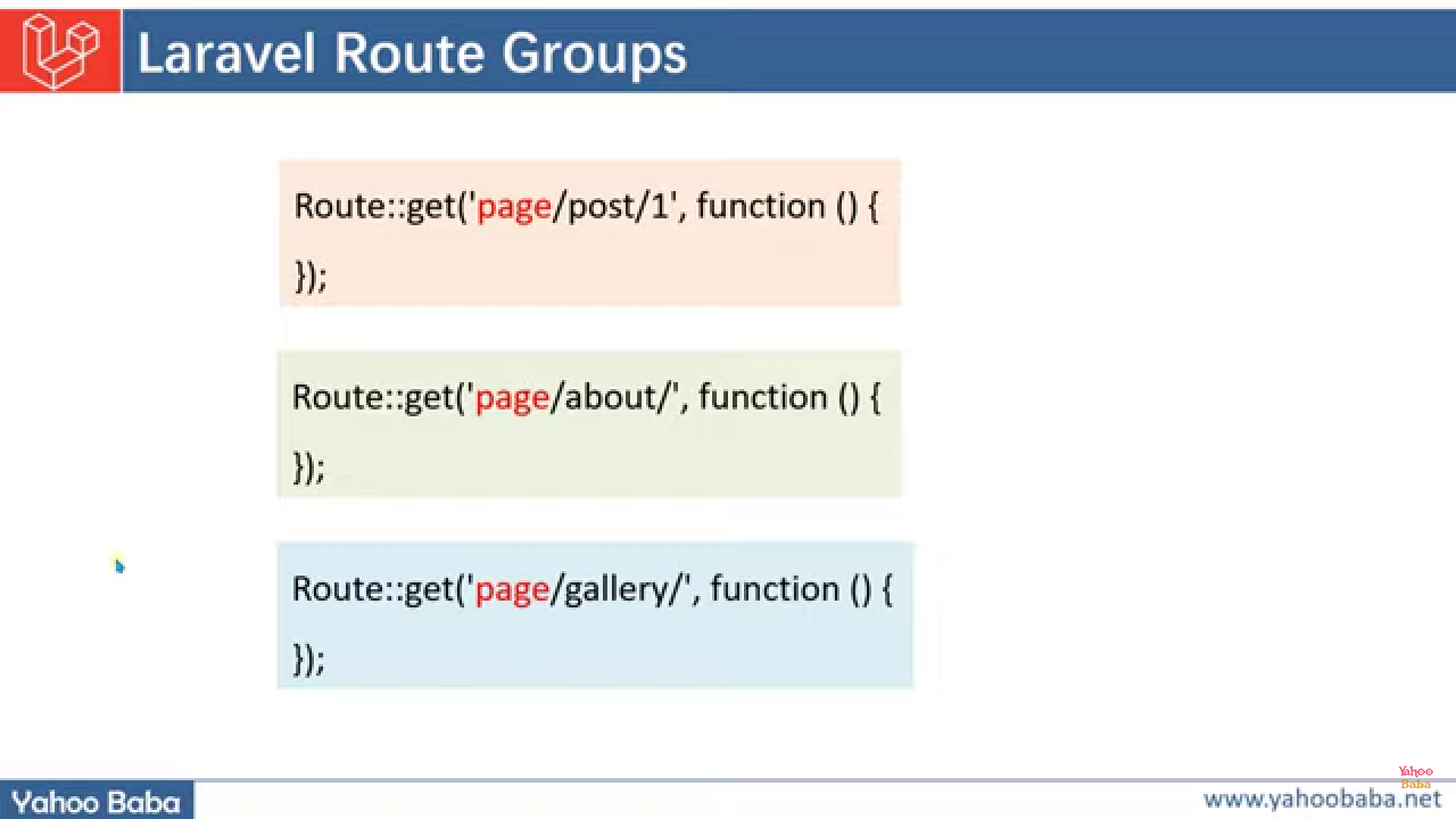
In the [HTTP](https://en.wikipedia.org/wiki/HTTP) [protocol](https://en.wikipedia.org/wiki/Protocol_(computing)) used by the [World Wide Web](https://en.wikipedia.org/wiki/World_Wide_Web), a **redirect** is a response with a [status code](https://en.wikipedia.org/wiki/List_of_HTTP_status_codes) beginning with *3* that causes a browser to display a different page. If a client encounters a redirect, it needs to make a number of decisions how to handle the redirect. Different status codes are used by clients to understand the purpose of the redirect, how to handle caching and which request method to use for the subsequent request.

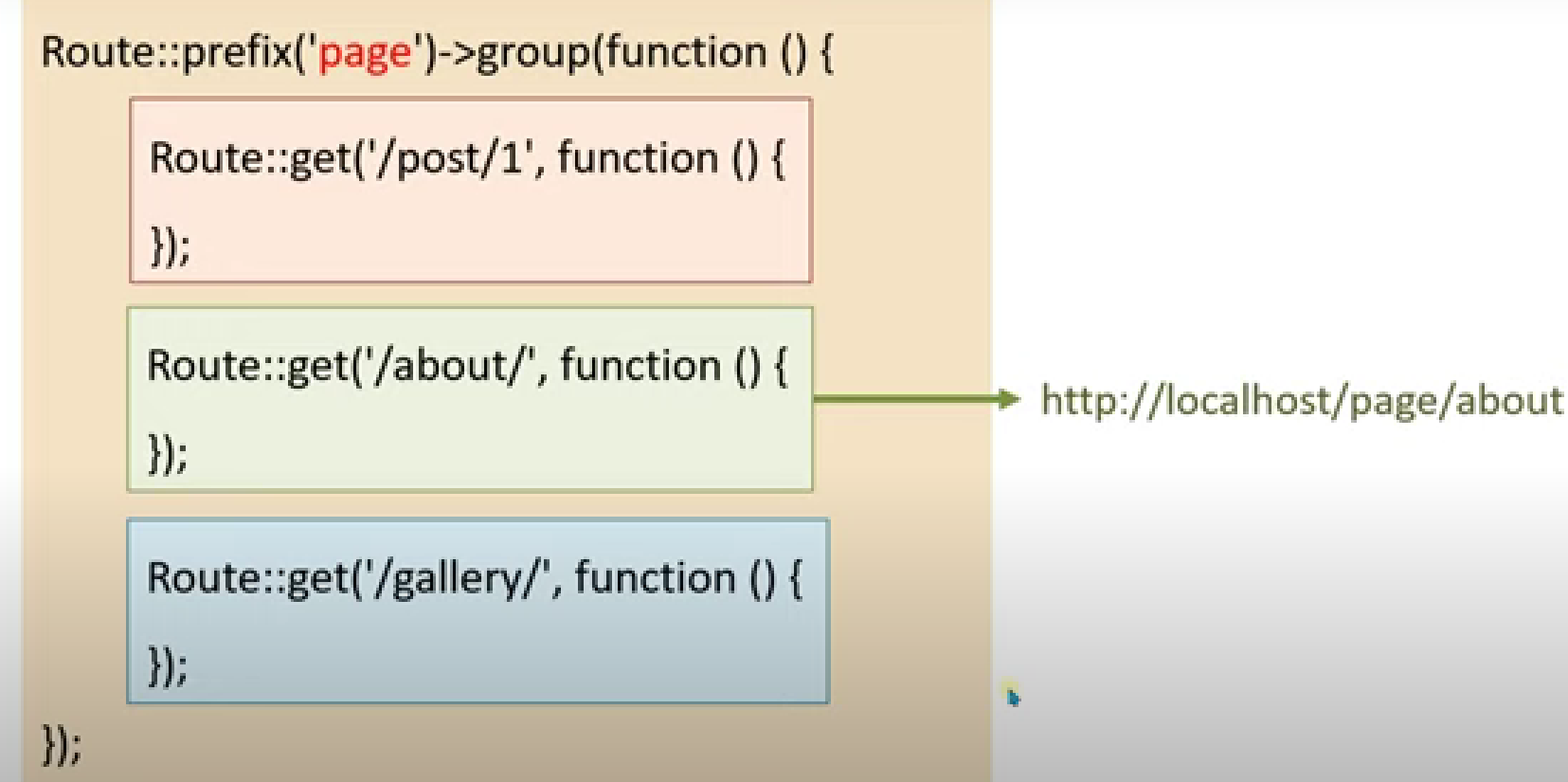
HTTP/1.1 defines several status codes for redirection (RFC 7231):

* [300 multiple choices](https://en.wikipedia.org/wiki/HTTP_300) (e.g. offer different languages)
* [301 moved permanently](https://en.wikipedia.org/wiki/HTTP_301) (redirects permanently from one URL to another passing link equity to the redirected page)
* [302 found](https://en.wikipedia.org/wiki/HTTP_302) (originally "temporary redirect" in HTTP/1.0 and popularly used for CGI scripts; superseded by 303 and 307 in HTTP/1.1 but preserved for backward compatibility)
* [303 see other](https://en.wikipedia.org/wiki/HTTP_303) (forces a GET request to the new URL even if original request was POST)
* [305 use proxy](https://en.wikipedia.org/wiki/305_Use_Proxy) (indicates that the client's requested resource is only available through a proxy)
* [307 temporary redirect](https://en.wikipedia.org/wiki/HTTP_307) (provides a new URL for the browser to resubmit a GET or POST request)
* [308 permanent redirect](https://en.wikipedia.org/wiki/HTTP_308) (provides a new URL for the browser to resubmit a GET or POST request)

Status codes [304 not modified](https://en.wikipedia.org/wiki/HTTP_304) and [305 use proxy](https://en.wikipedia.org/wiki/HTTP_305) are not redirects.

Laravel grouping routes:





*//grouping route*

Route::prefix('page')->group(function(){

    Route::get('/page1',function(){

*return* "<h1><b></b>page1 page</b></h1>";

    });

    Route::get('/page2',function(){

*return* "<h1><b></b>page2 page</b></h1>";

    });

    Route::get('/page3/page3.1',function(){

*return* "<h1><b></b>page3.1 page</b></h1>";

    });

    Route::get('/page4/page4.1/page4.2',function(){

*return* "<h1><b></b>page4.2 page</b></h1>";

    });

});

Showing custom view to user if he/she accessing a unknown/un-exist route

Syntax:

Route::fallback(function(){

return view(‘view name’);

});

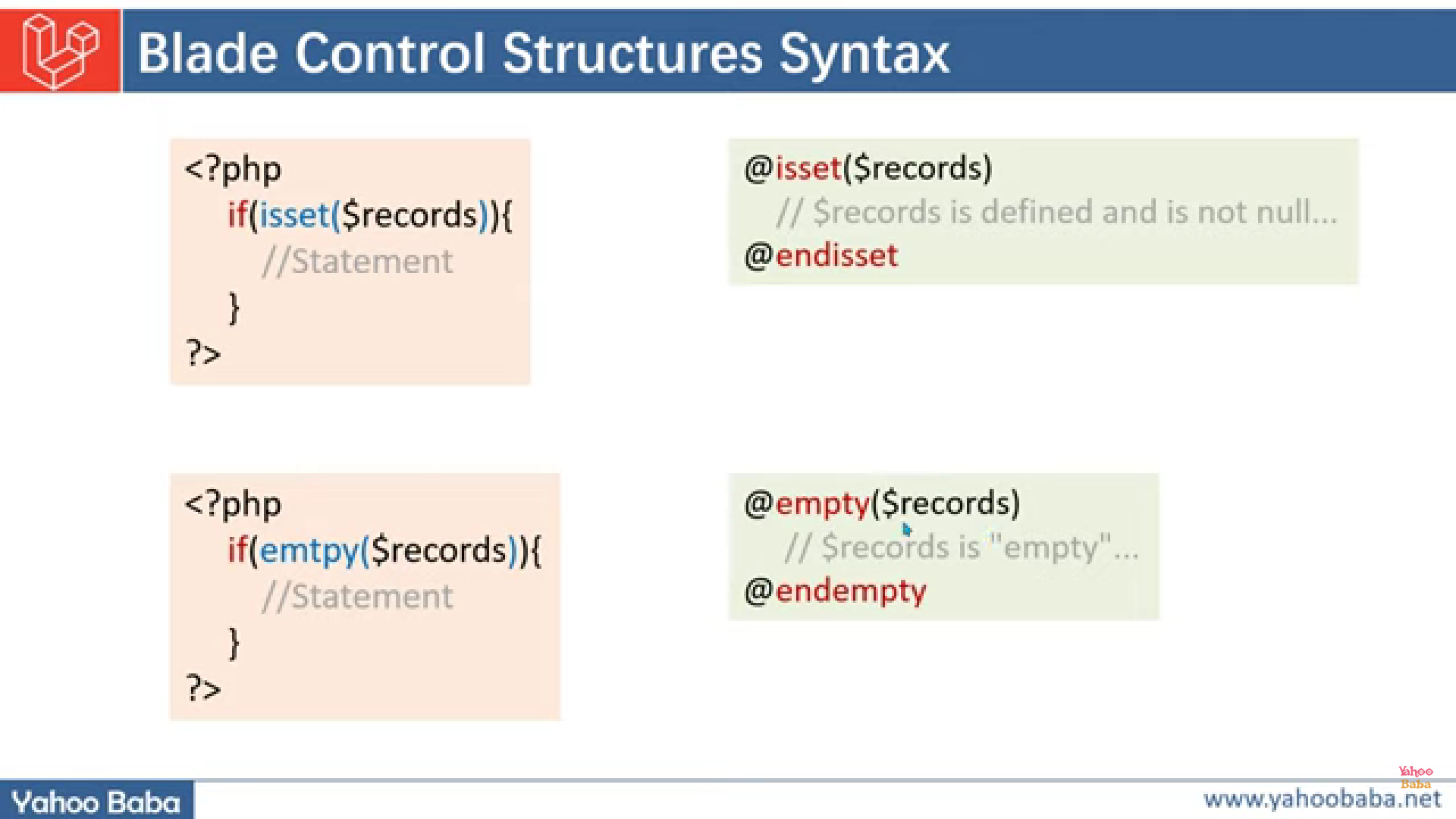
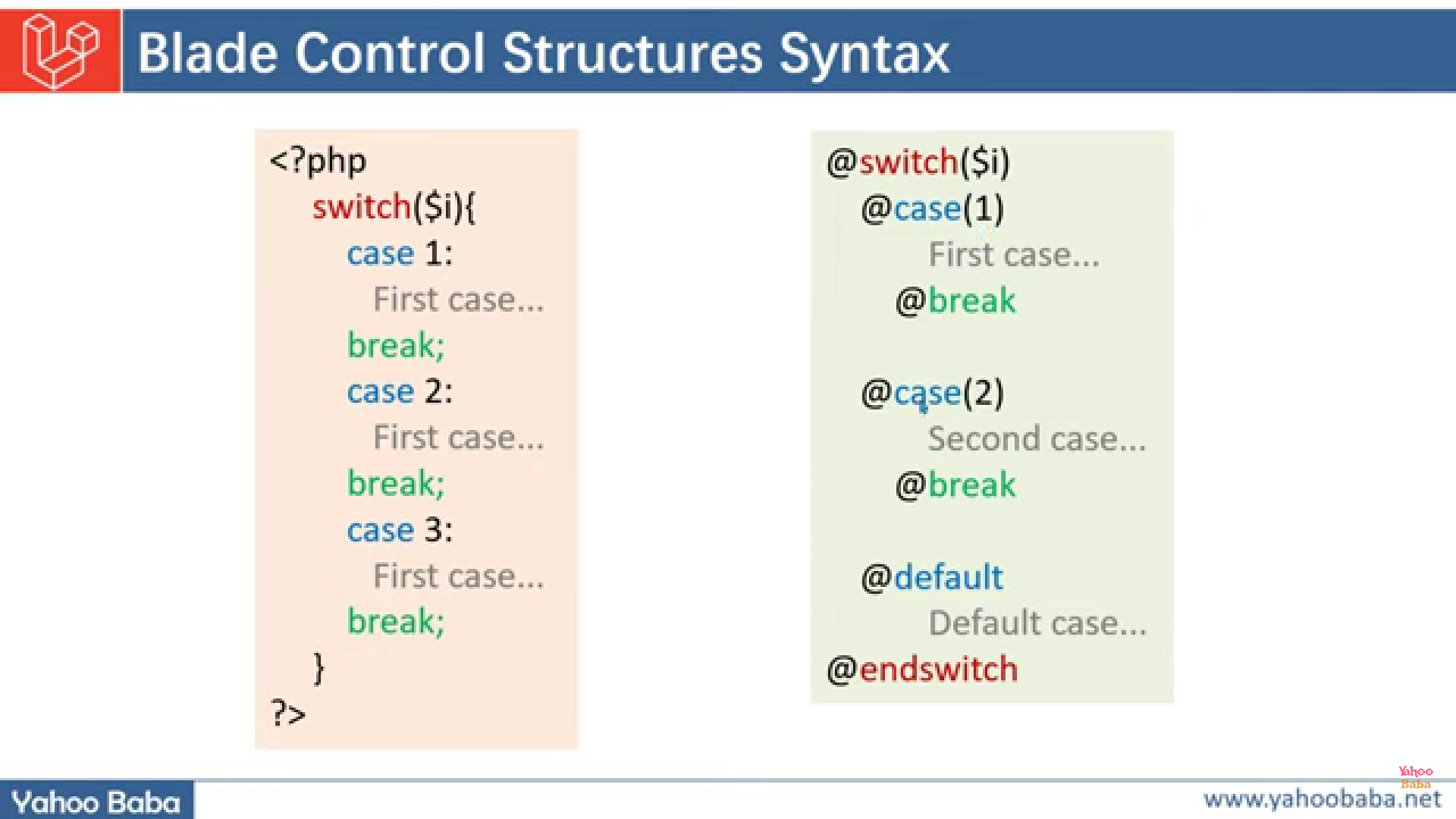
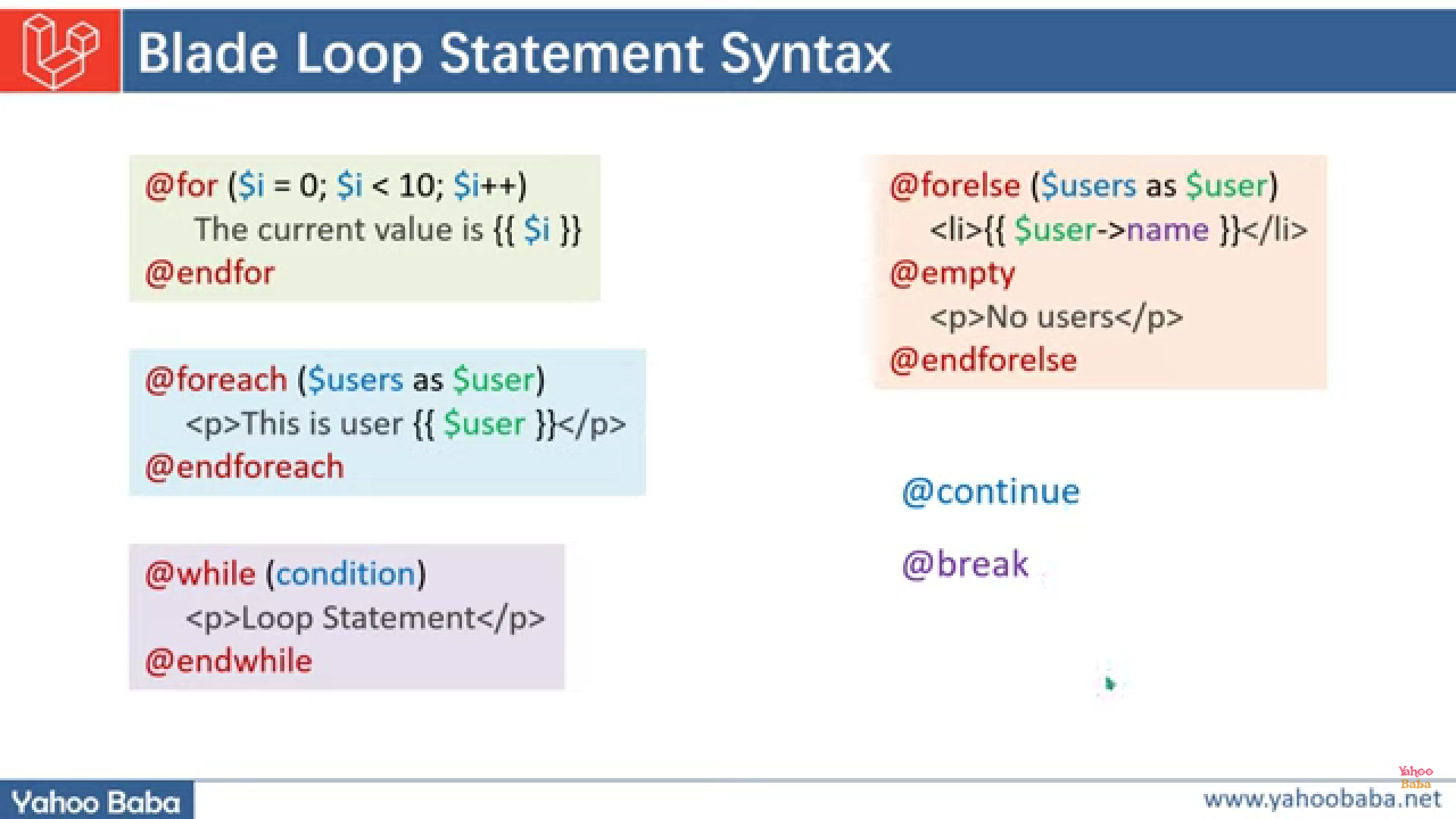
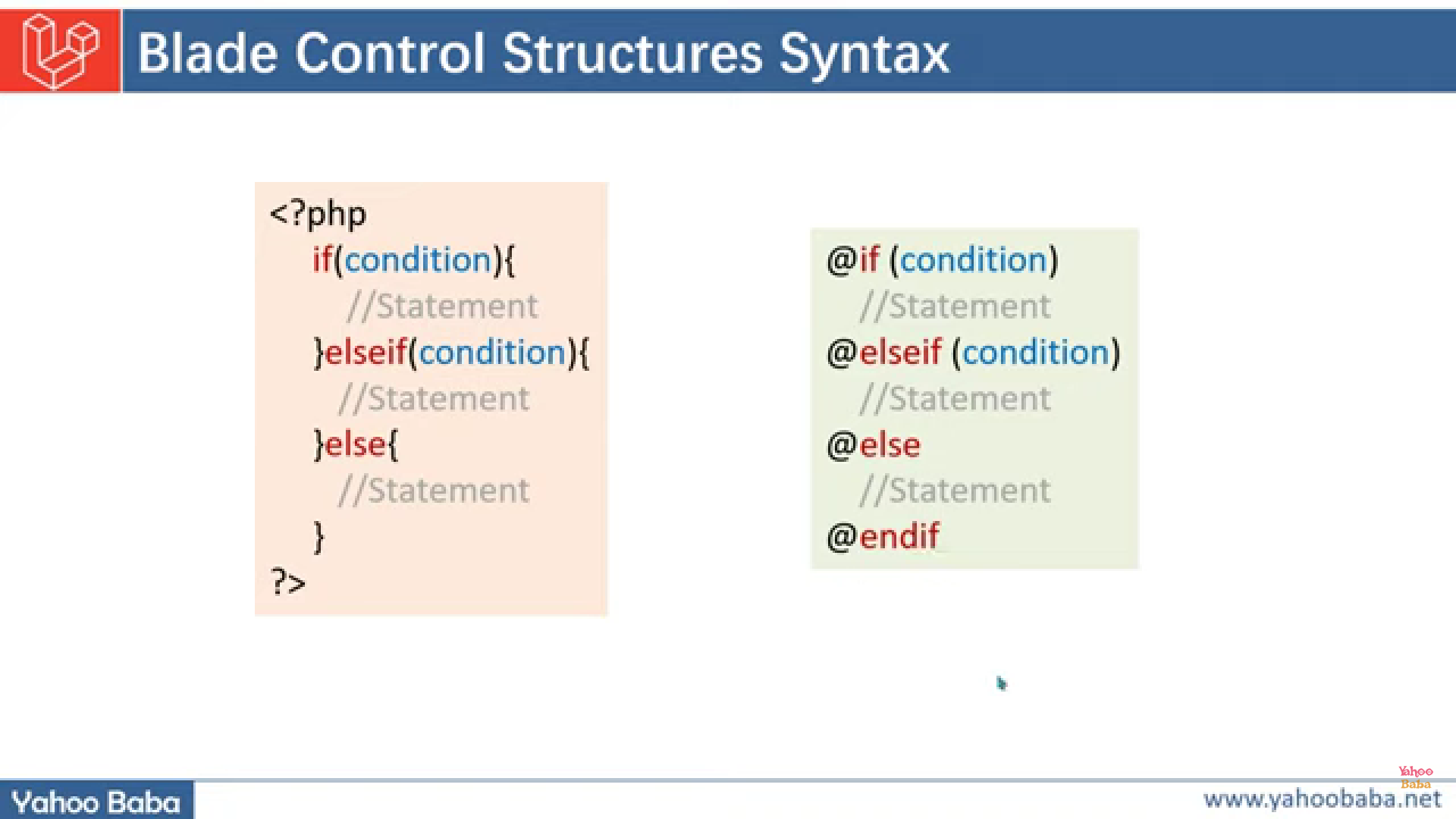
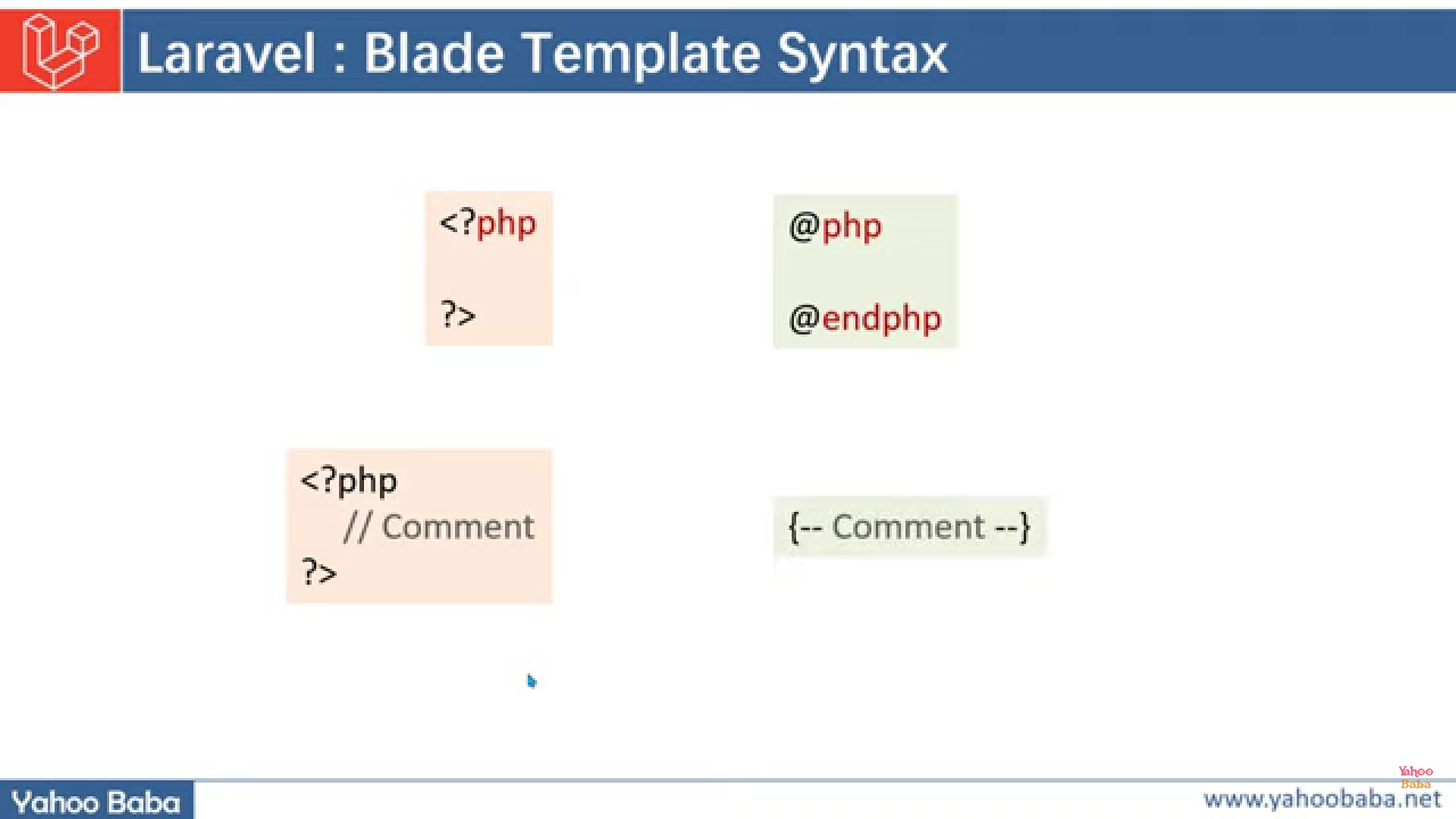
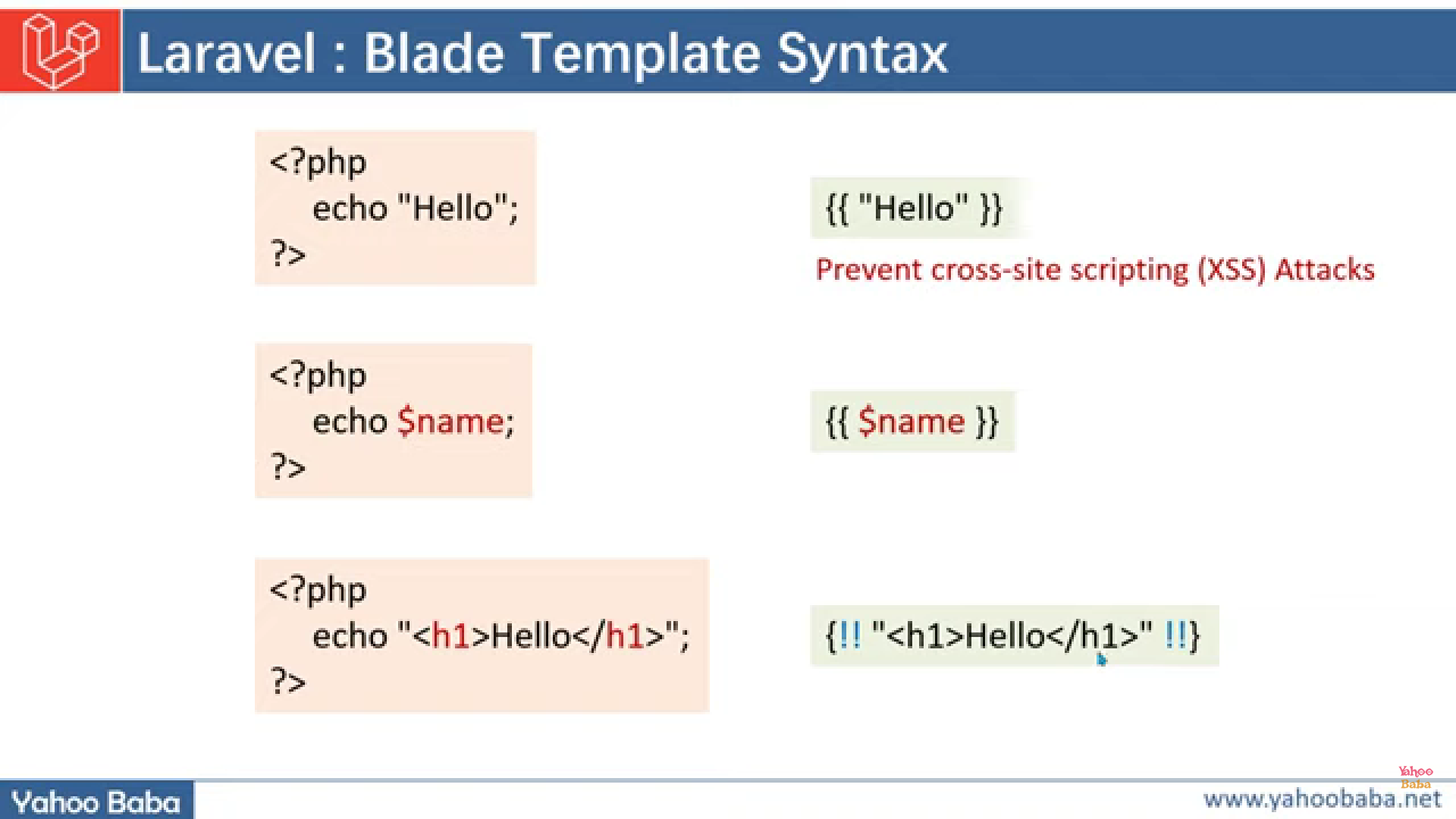
*//if user want to access undefined route then we can show our custom view to user like:*

    Route::fallback(function(){

*return* "<h1>page not found buddy</h1>";

    });

Blade template syntax’s:



Ex:

1.comment:

*{{-- this is my comment --}}*

2.echo command :

{{5+2+10}}

{{"hello this is like echo command"}}

3.html tags in echo:

{!!"<p>hello this is like html tags in echo command</p>"!!}

{!!"<b>hello this is like <li>html</li> tags in echo command</b></b></p>"!!}

4.js commands in echo

{!!"<script>alert('this is js tags in blade')</script>"!!}

5.<?php……?> command and printing value of variable

@php

$x=20;

@endphp

{{"value of x: "}}{{$x}}

6.if-else condition

*{{-- if else condition --}}*

@if($x==10)

    x is equals to 10

@elseif($x==20)

    x is equals to 20

@else

    value of x:{{$x}}

@endif

7. while loop:

*{{-- while loop --}}*

@while ($x<25)

    {{$x}}

    @php

        $x++;

    @endphp

@endwhile

8: for and for each loop:

@php

    $arr=[1,2,3,4,5,6,7,8,9,10];

@endphp

*{{-- for loop  --}}*

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

    {{$arr[$i]}}

@endfor

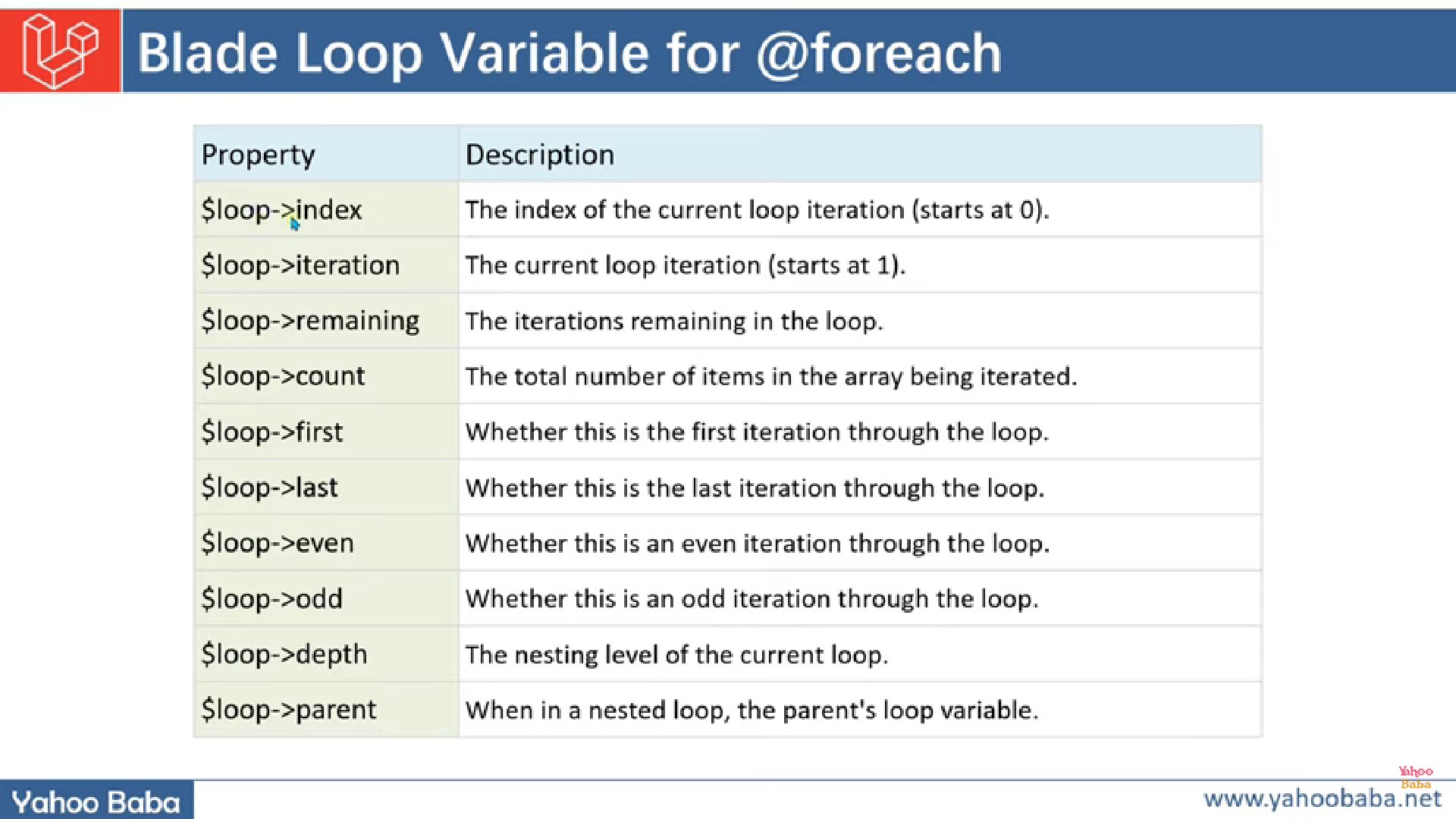
*{{-- for each loop  --}}*

@foreach ($arr as $value )

    {{$value}}

@endforeach

9:functions for for-each loop



Ex:

@php

    $arr=[1,2,3,4,5,6,7,8,9,10];

@endphp

*{{-- for each loop functions --}}*

@foreach($arr as $m)

    @if ($loop->first)

        <h1 style="color:red"> {{$m}} </h1>

    @endif

    @if ($loop->last)

        <h1 style="color:blue"> {{$m}} </h1>

    @endif

    @if ($loop->even)

        <i style="color:blueviolet"> {{$m}} </i>

    @endif

    @if ($loop->odd)

        <i style="color:yellow"> {{$m}} </i>

    @endif

    {{"index:"}}{{$loop->index}}

    <br>

    {{"itration:"}}{{$loop->iteration}}

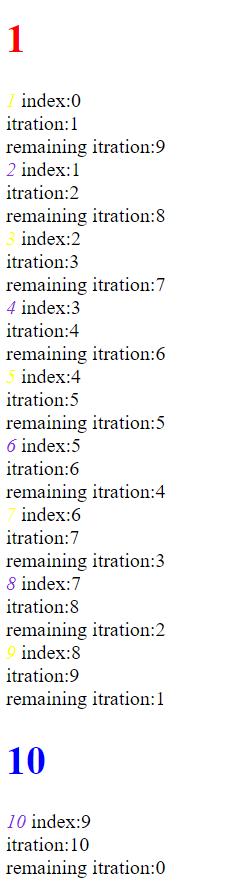
    <br>

    {{"remaining itration:"}}{{$loop->remaining}}

    <br>

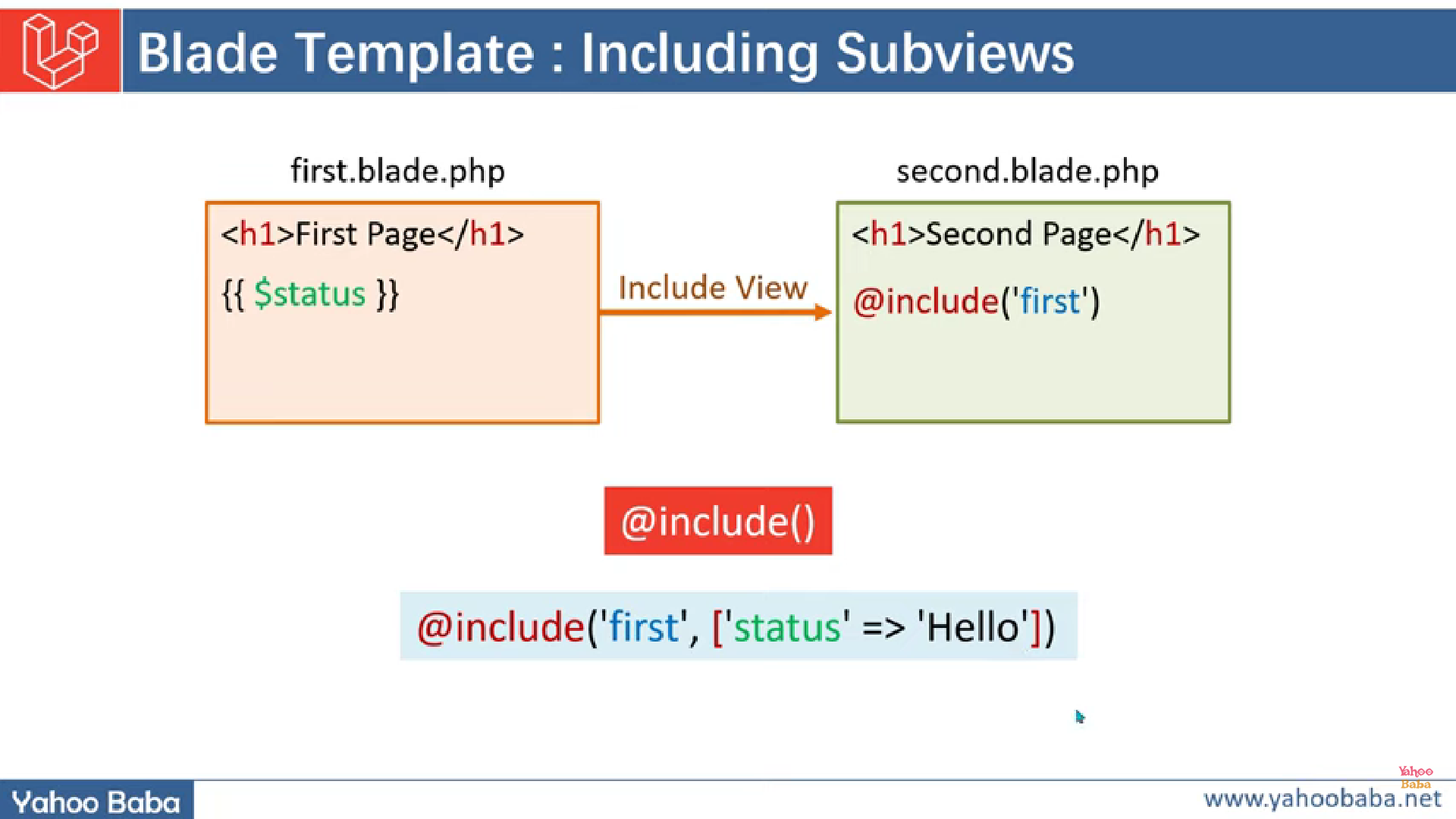
@endforeach

o/p



17 March 2024 Blade template main directives:

1. @include:



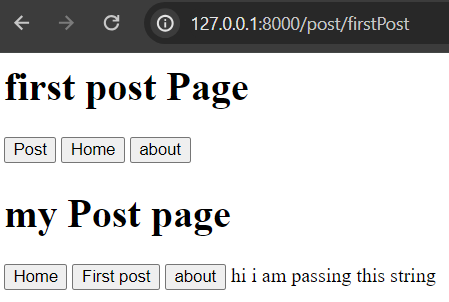
Example:

*{{-- @include() directive in blade --}}*

@include('post')

*{{-- passing a parameter with include directive --}}*

@include('post',['pass'=>"hi i am passing this string"])



Passing array

firstPost.blade.php;

*{{-- passing array as a values --}}*

@php

    $myarr=["suraj","rowdy","patil","khot","xyz"];

@endphp

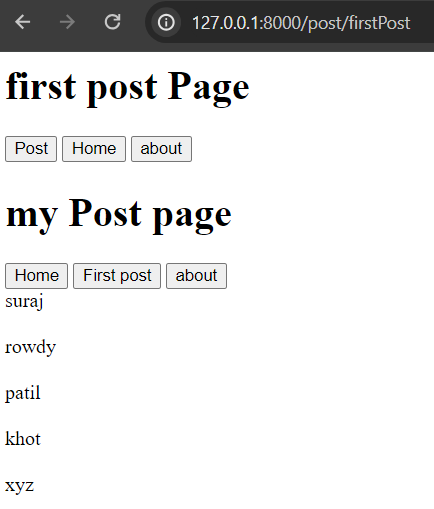
@include('post',["array"=>$myarr])

Post.blade.php;

@foreach ($array as $x )

    {!! "<br> $x <br>"!!}

@endforeach



If array having null values:

firstPost.blade.php:

*{{-- if array dont have any value --}}*

@php

    $myarr=[];

@endphp

@include('post',["array"=>$myarr])

post.blade.php:

*{{-- using for else loop alse we can done the same task properly --}}*

@forelse($array as $key=>$value)

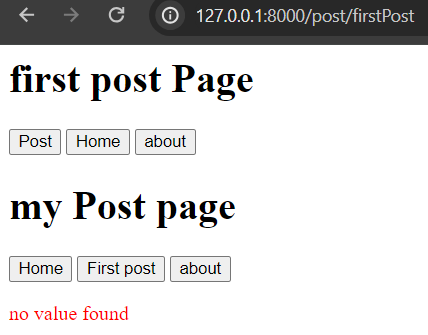
    <p>{{$key}}-{{$value}}</p>

@empty

    <p style="color:red">no value found</p>

@endforelse

o/p:



1. @includeIf();

If there is no view found then we got error while using @include() but we can use @includeIf() directive and we get view only if that exists

Ex:

*{{-- @includeIf() in blade --}}*

@php

    $myarr=[];

@endphp

@includeIf('posti',["array"=>$myarr])

There is no posti view then also pgm runs



3. @includeWhen(): it execute view-file if condition is true or 1

*{{-- @includeWhen(, ) in blade --}}*

@php

    $myarr=[];

    $bool=true;

@endphp

@includeWhen($bool,'post',['array'=>$myarr])

4. @includeUnless() : it execute view-file if condition is false or 0

*{{-- @includeUnless(, ) in blade --}}*

@php

    $myarr=[];

    $bool=true;

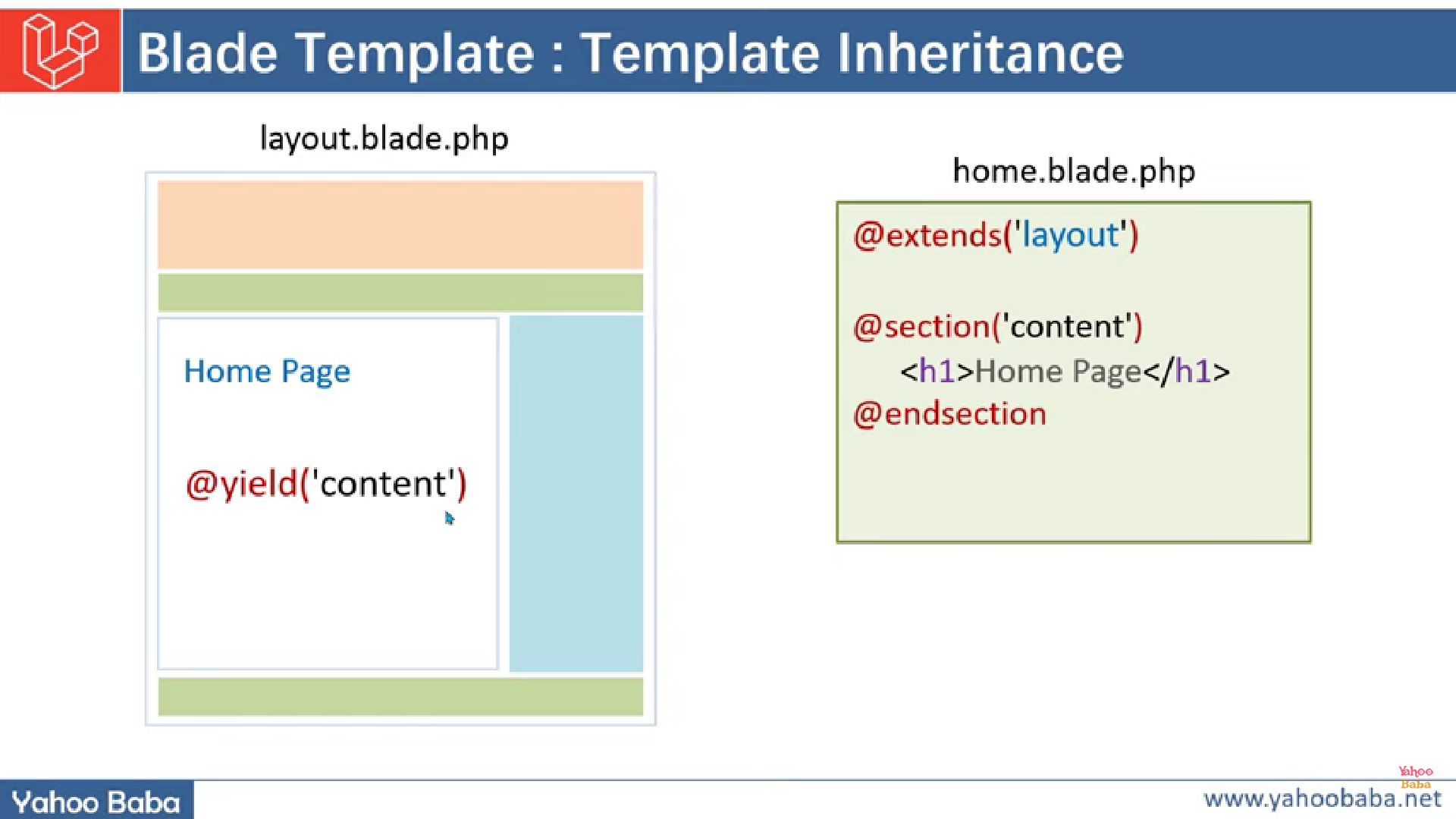
@endphp

@includeWhen($bool,'post',['array'=>$myarr])

It will not return view

Blade template inheritance:

1:@yield() and @section



Syntax:

Using yield in one view:

//header

@yield(‘yield-name’);

//footer

And using inherit in other view:

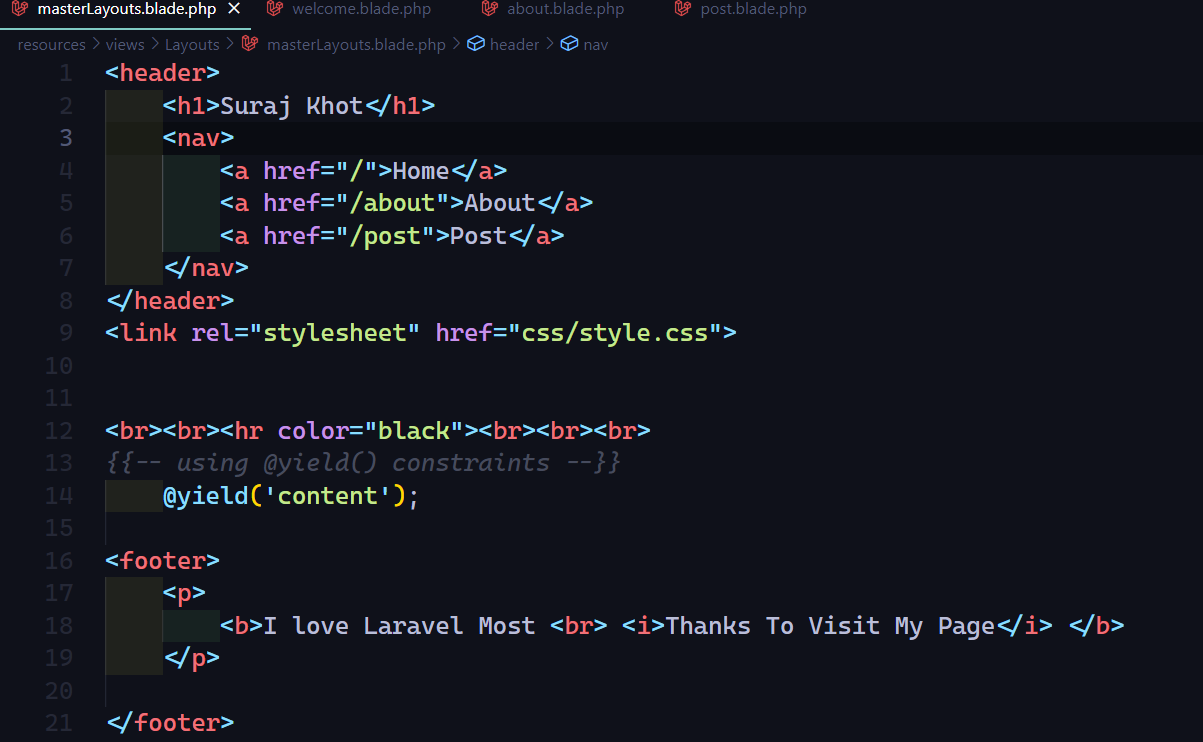
@extend(‘pathtoview\_where@yaildpresent’);

@section(‘yield-name’)

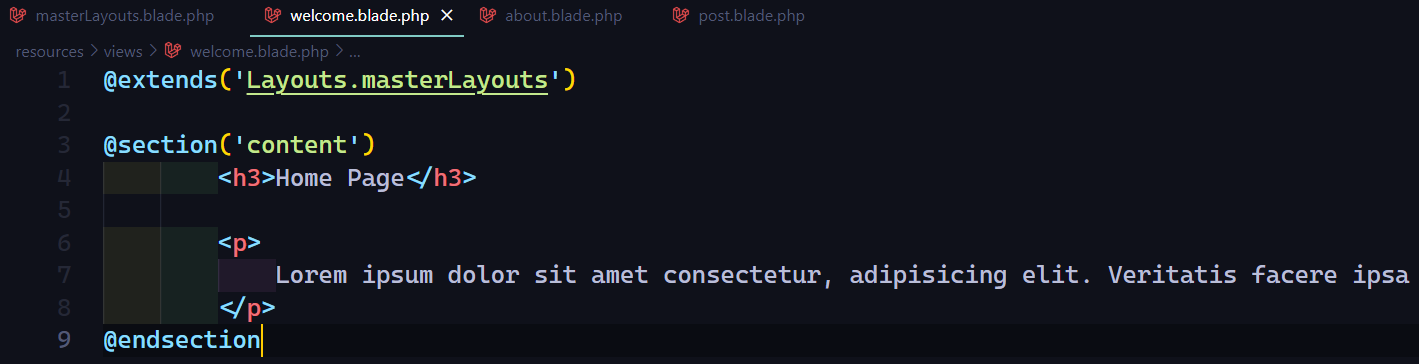
//content

@endsection

Main file where @yield(); present



Files where extend yield() and showing content using @section



NOTE:

IF WE ADD SECTION TWICE THEN IT WILL SHOW ONLY FIRST SECTION ONLY

@section('content')

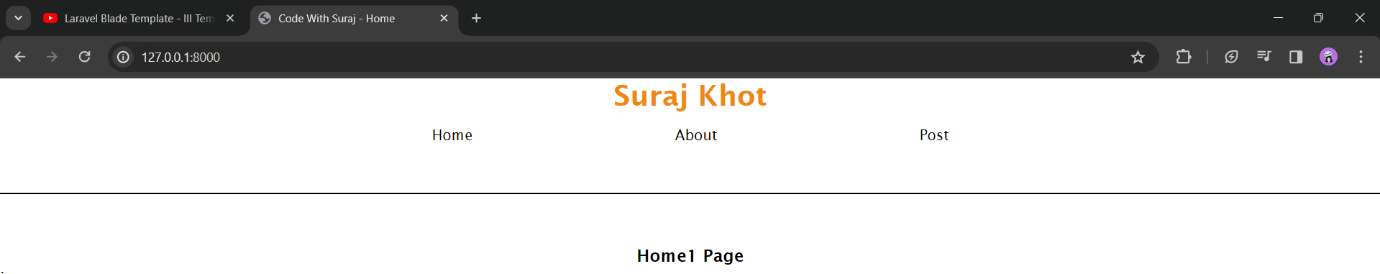
        <h3>Home1 Page</h3>

@endsection

@section('content')

        <h3>Home2 Page</h3>

@endsection

It will show first content only

2.@hasSection():

Case if declare yield and forgot to add section then to go with program flow there is a constraint called @hasSection():

Syntax:

@hasSection(‘yeild-name’)

@yield(‘yield-name’)

@else

//statement

@endif

Example:

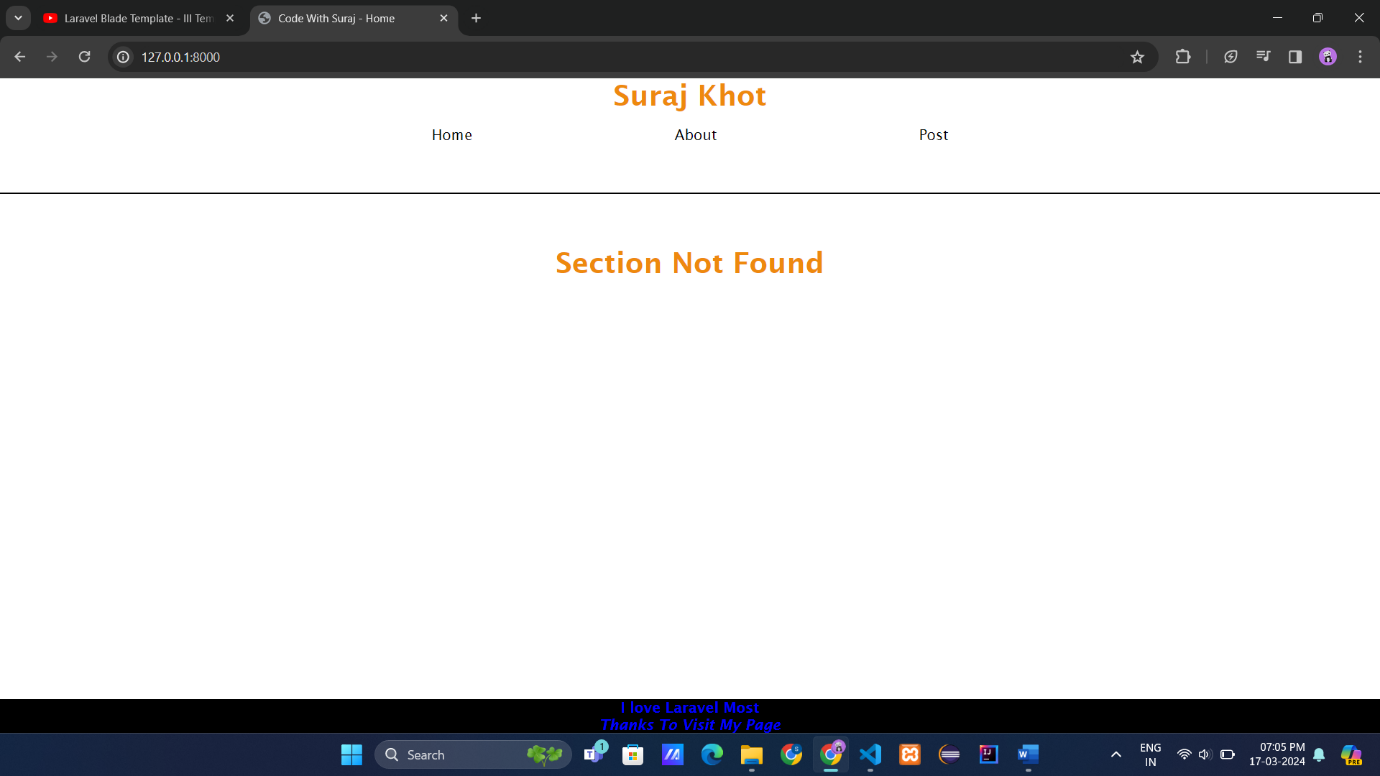
@hasSection('content')

    @yield('content');

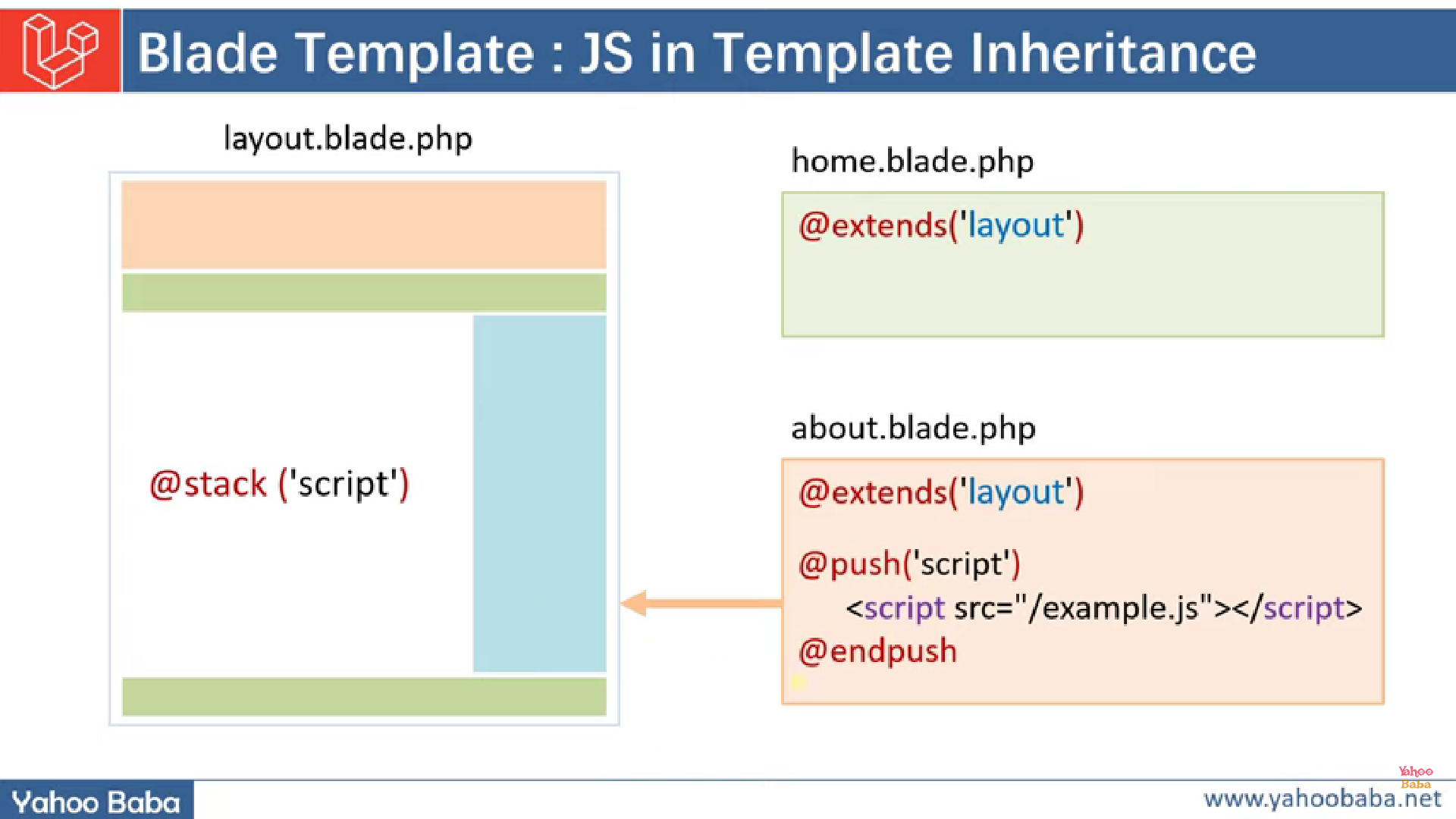
@else

    <h1>Section Not Found</h1>

@endif



Java script in template inheritance:



To add js file in blade we use @stack(‘’) and @push(‘’) directives:

MasterTemplate.blade.php

@stack('JsFiles')

@stack('JsFiles')

Welcome.blade.php

@push('JsFiles')

        <script src="/f1.js"></script>

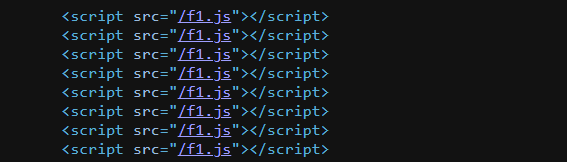
        <script src="/f1.js"></script>

        <script src="/f1.js"></script>

        <script src="/f1.js"></script>

@endpush

o/p in browser on page source:



Hence we can create once use many

We can create @stack(‘’) twice

Also we can add many @push(‘’) for one stack

@stack('JsFiles')

@push('JsFiles')

        <script src="/f1.js"></script>

        <script src="/f1.1.js"></script>

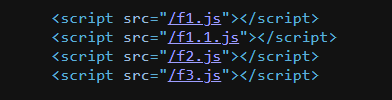
@endpush

@push('JsFiles')

        <script src="/f2.js"></script>

        <script src="/f3.js"></script>

@endpush



We can also add css file inside push:

@push('CssFiles')

        <link rel="stylesheet" href="/css/bootsrap.css">

@endpush

We Can add default css using @prepend(‘stack name’) directive it will add content **before** specified stack name:

Sytnax:

@prepend(‘stack-name’)

//statement

@endprepend

Note : “it will add content before stack-name ”

Ex:

@push('CssFiles')

        <link rel="stylesheet" href="/css/bootsrap.css">

@endpush

@prepend('CssFiles')

        <style>

                h1{

                        color: darkmagenta;

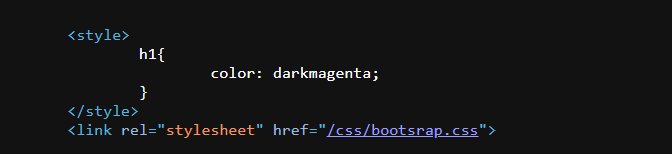
                }

        </style>

@endprepend

It will add content before CssFiles and see result **page source view**:

It added before CssFiles stack



CssFiles stack

To Show absolute path of css file in page source on browser

Using asset() function

Syntax:

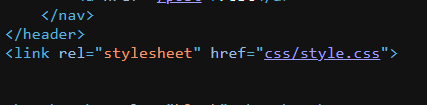
{{ asset(‘path to css file’) }}

Before:

In blade file

<link rel="stylesheet" href="css/style.css">

On page source

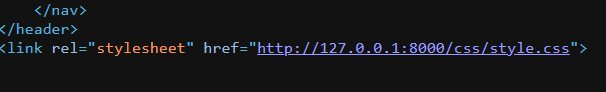


After:

On blade file

<link rel="stylesheet" href="{{ asset('css/style.css') }}">

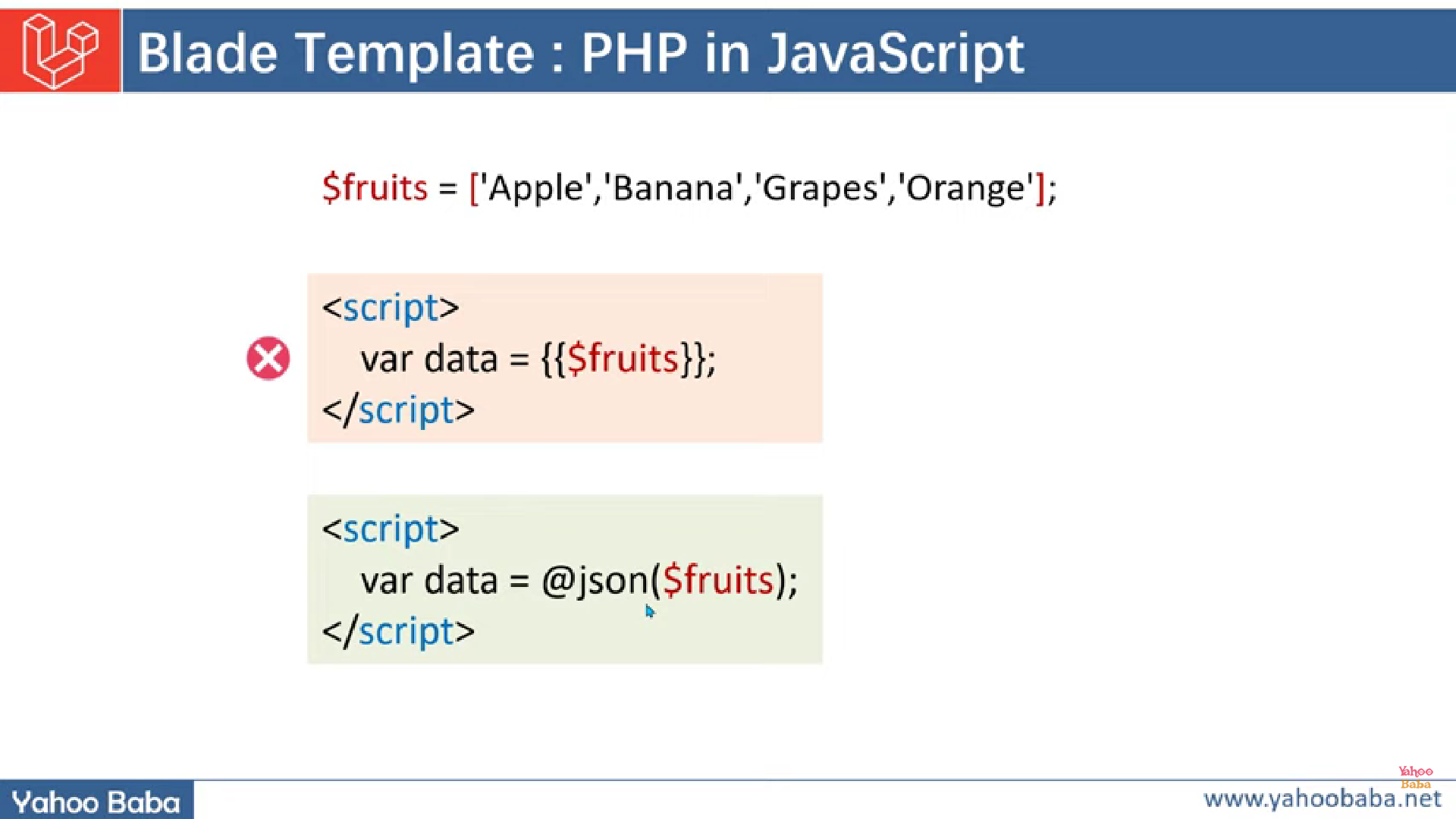
On page source



Printing php variable in console i.e using javascript :

Synatx:

@json(‘name-of-variable’);



Ex:

@php

    $name="suraj khot";

@endphp

*{{-- printing variable using script tag --}}*

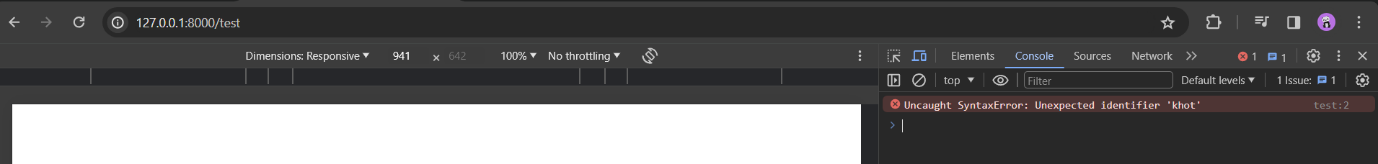
<script>

        var name={{$name}};

        console.log(name);

</script>

Error:



To resolve this error we are going to use **@json()** directive

@php

    $name="suraj khot";

@endphp

*{{-- printing variable using script tag --}}*

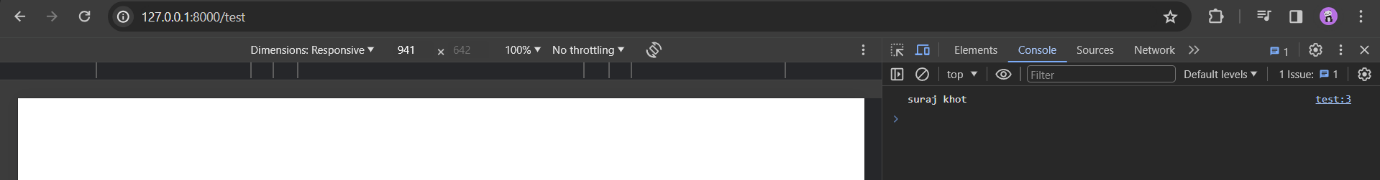
<script>

        var name=@json($name);

        console.log(name);

</script>

Running:



Passing array:

Way 1:

@php

$arr=["suraj","sanjay","khot","BCA","81.72%"];

@endphp

*{{-- printing variable using script tag --}}*

<script>

        var array=@json($arr);

        console.log(name);

*//first way*

        array.forEach(*element* => {

            console.log(element);

        });

*//secound way*

        array.forEach(function(*element*){

            console.log(element);

        });

</script>

Way 2:

@php

$arr=["suraj","sanjay","khot","BCA","81.72%"];

@endphp

*{{-- printing variable using script tag --}}*

<script>

*//using Js::from() method*

var array={{ Js::from($arr)  }}

        console.log(name);

*//first way*

        array.forEach(*element* => {

            console.log(element);

        });

*//secound way*

        array.forEach(function(*element*){

            console.log(element);

        });

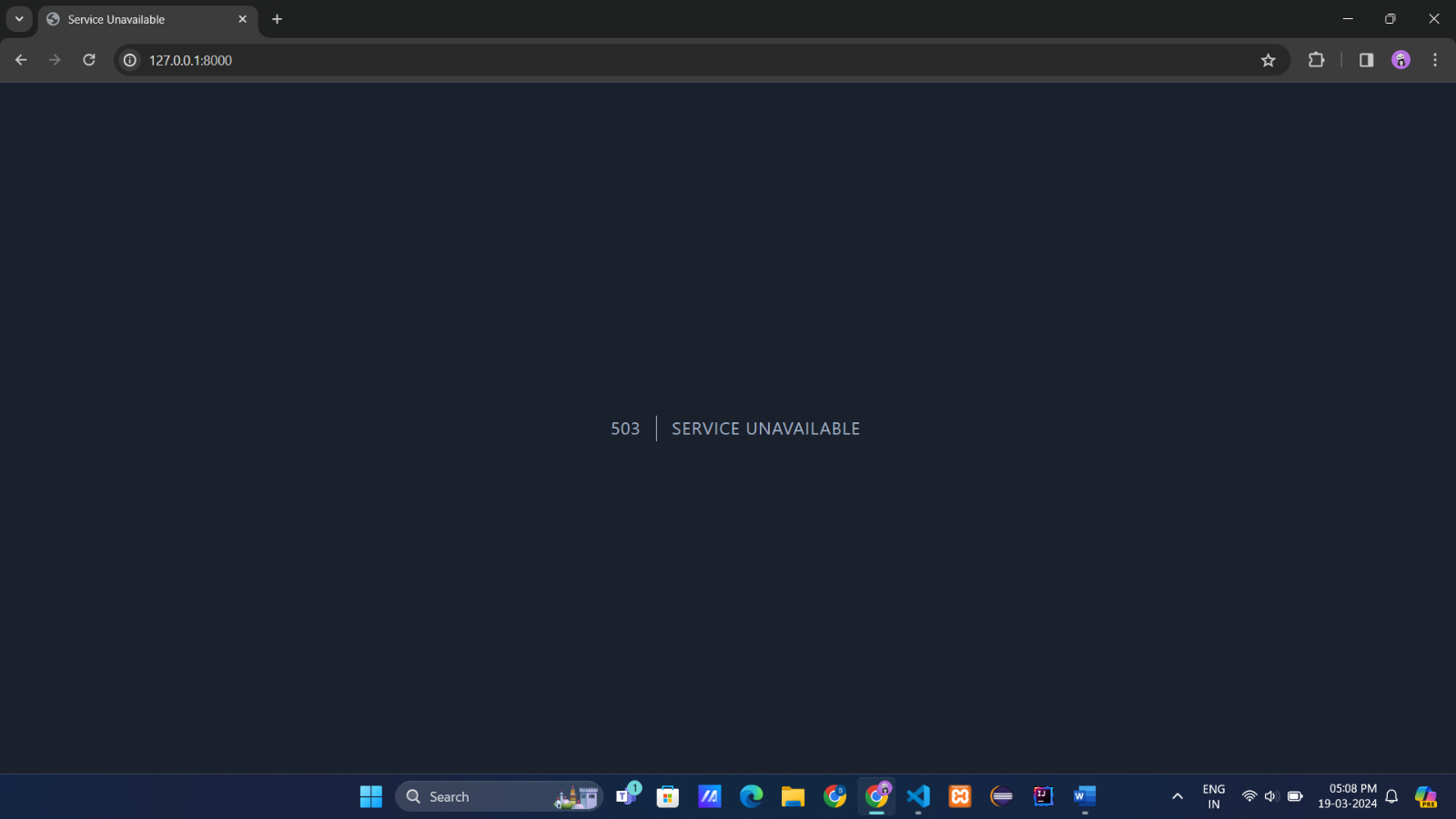
</script>

How to our project in Maintenance mode

We just fire an artisan command to on maintenance mode of our project

$ php artisan down

INFO Application is now in maintenance mode.



To make live we fire:

$ php artisan up

INFO Application is now live.

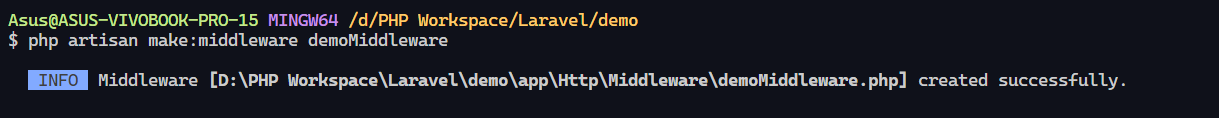
Middleware in Laravel

To create middleware (act as middle man between request and response )we fire command :

Syntax : $php artisan make : middleware <name>

After creating it will found in path : project-name\app\http\middleware\<name-middleware>

Ex:



Controller

