



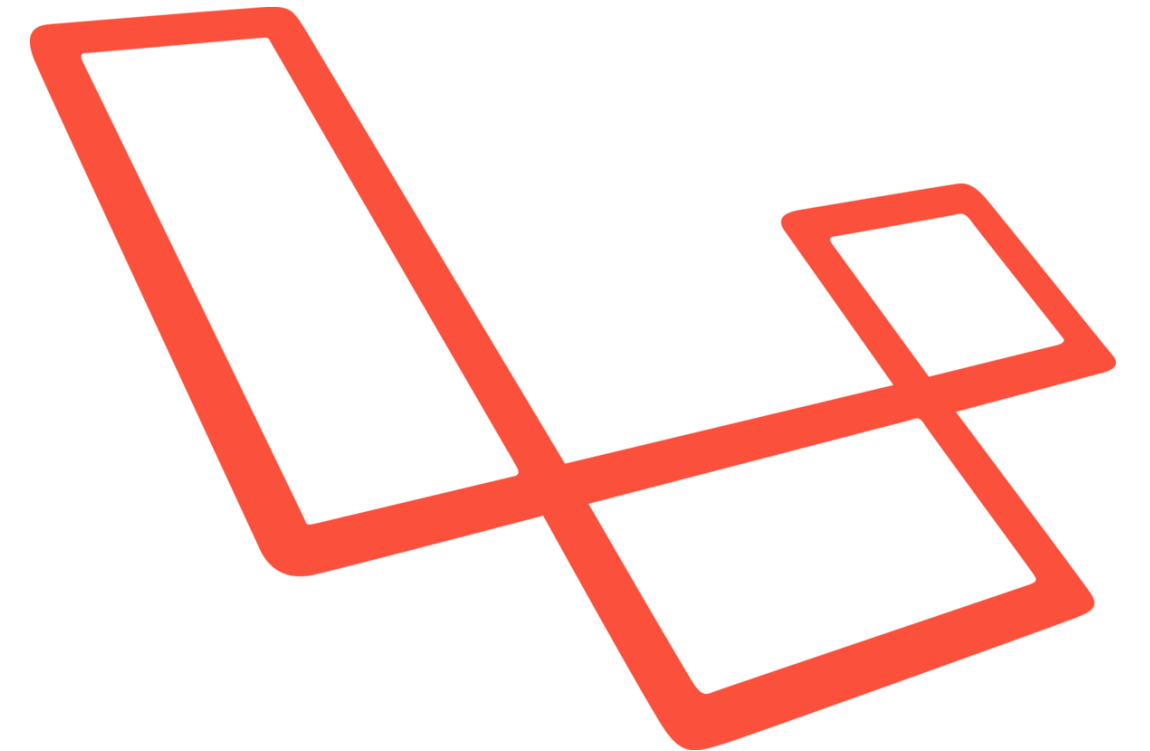
## Laravel: Intro

Web, Mobile and Security  
Frédéric Vlummens

# Agenda

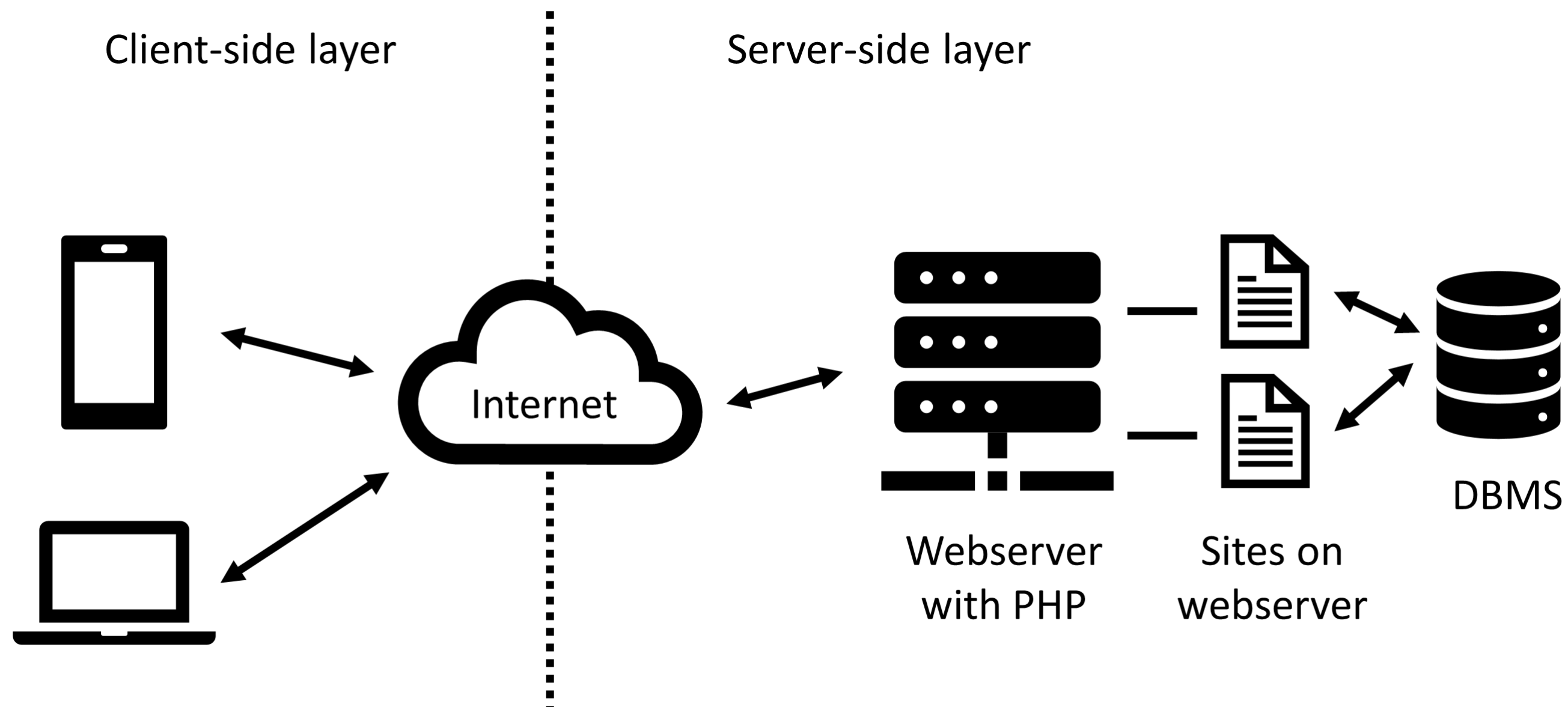
---

- Server-side scripting: recap
- What is Laravel?
- Routes
- Views
- Models
- Controllers
- The MVC pattern



# Server-side scripting: recap

- PHP is a server-side scripting language



# Server-side scripting: recap

---

- PHP code is executed and stays on the server side
- Generates (HTML) code, which in turn is sent to the client side
- The browser (client) does not understand/execute/receive PHP code

# Server-side scripting: recap

```
<?php
function calc_avg($a, $b) {
    return ($a + $b) / 2;
}

$x = 3;
$y = 4;

$avg = calc_avg($x, $y);
?>
```



```
<body>
    <p>
        The average of <?php echo $x; ?> and <?php echo $y; ?> is:
        <?php echo $avg; ?>
    </p>
</body>
```

```
1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4     <meta charset="UTF-8">
5     <meta name="viewport" content="width=device-width, initial-scale=1.0">
6     <title>Document</title>
7 </head>
8 <body>
9
10     <p>
11         The average of 3 and 4 is:
12         3.5     </p>
13
14 </body>
15 </html>
```

# “Classic” PHP scripting

- Mix of HTML and PHP code in same file
- Can become messy

```
2  <?php
3
4  // Random PHP code snippet!
5
6  function create_category_feeds($categories = NULL) {
7
8      global $wpdb, $title, $headcomments;
9
10     if ($categories == NULL) {
11         $sort_column = 'term_id';
12         $query = "SELECT * FROM $wpdb->term_taxonomy
13                 JOIN $wpdb->terms ON ( $wpdb->term_taxonomy.term_id = $wpdb->terms.term_id )
14                 WHERE $wpdb->term_taxonomy.taxonomy = 'category' AND $wpdb->terms.term_id > 0 AND count
15                 ORDER BY $wpdb->terms.name ASC";
16         $categories = $wpdb->get_results($query);
17     }
18
19     $catsnum = count($categories);
20
21     foreach ($categories as $category) {
22         $link = '<link rel="alternate" type="application/rss+xml" title="';
23         $link = $link . $title . ': ' . $category->name;
24         $link = $link . '" href="' . get_category_rss_link(0, $category->term_id, $category->name) . '" />';
25         echo "\t" . $link . "\n";
26     }
27
28     $hcomlink = '<link rel="alternate" type="application/rss+xml" title="';
29     $hcomlink = $hcomlink . $title . ': Comments';
```

# “Classic” PHP scripting

- Mix of HTML and PHP code in same file
- Can become messy

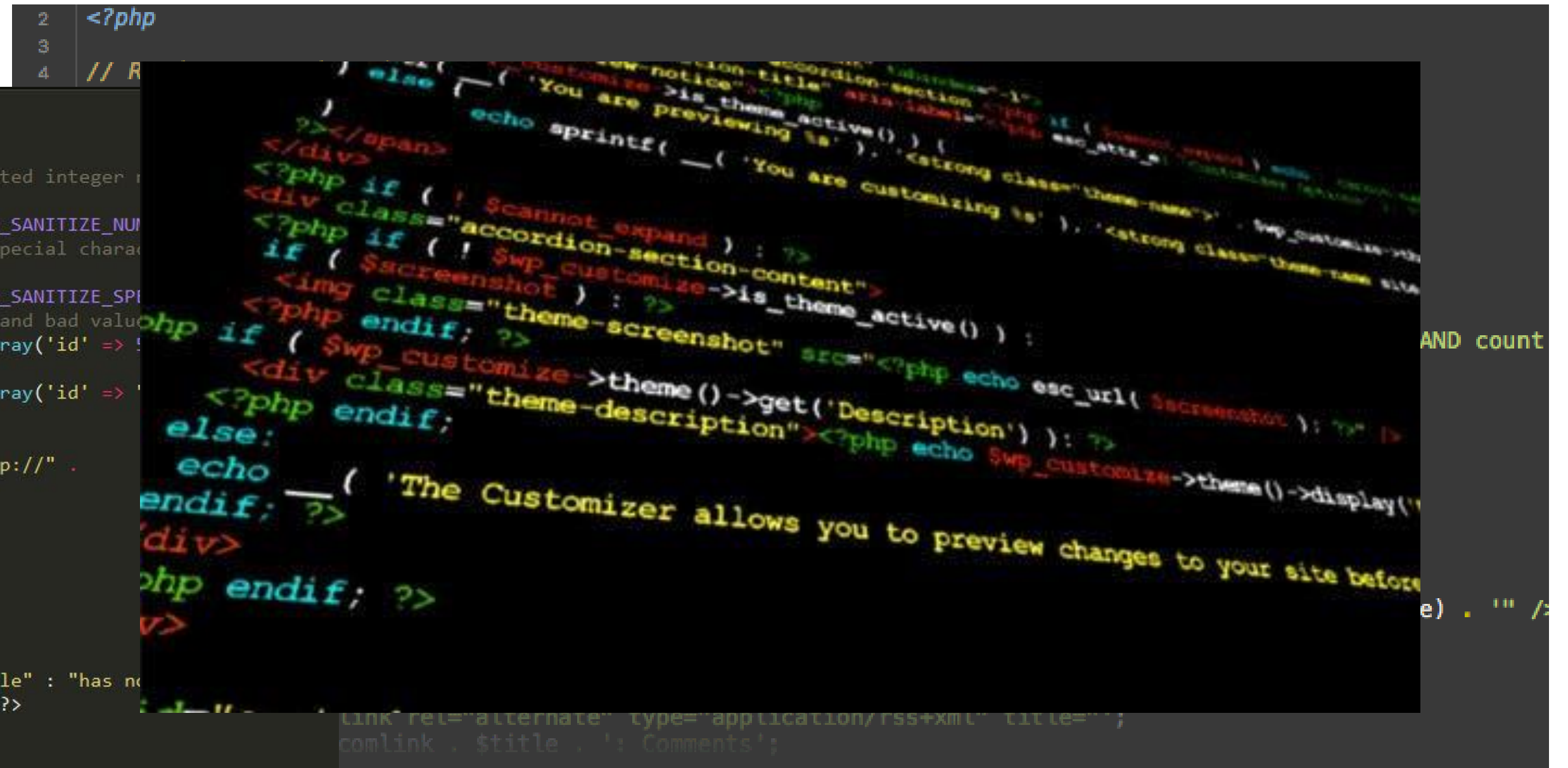
```
1 <?php
2 /*
3  * Extract the 'id' value from the query string.
4  * We're filtering to ensure it matches the expected integer number we expect.
5  */
6 $id = filter_input(INPUT_GET, 'id', FILTER_SANITIZE_NUMBER_INT);
7 // Filter the 'title' value, which can contain special characters (HTML
8 // entities and the like).
9 $title = filter_input(INPUT_GET, 'title', FILTER_SANITIZE_SPECIAL_CHARS);
10 // Build a simple URL with some good (matching) and bad values.
11 $goodURL = baseURL() . '?' . http_build_query(array('id' => 5276,
12     'title' => "Les Misérables"));
13 $badURL = baseURL() . '?' . http_build_query(array('id' => "not a number",
14     'title' => "Les Misérables"));
15
16 function baseURL() {
17     return (@$_SERVER['HTTPS']) ? "https://" : "http://" .
18         $_SERVER["SERVER_NAME"] .
19         ($SERVER['SERVER_PORT'] != 80 ? ":" : "" .
20         $_SERVER['SERVER_PORT'] : null) .
21         $_SERVER['SCRIPT_NAME'];
22 }
23
24 <!-- Our basic HTML output -->
25 <p>
26     Your searched book <?= $title ? "is titled $title" : "has no title" ?> and
27     has <?= $id ? "the ID $id." : "an invalid ID." ?>
28 </p>
29 <a href="<?= baseURL() ?>">Home</a><br/>
30 <a href="<?= $goodURL ?>">Good Link</a><br/>
31 <a href="<?= $badURL ?>">Bad Link</a><br/>
32
33 <?php
34 // Random PHP code snippet!
35
36 category_feeds($categories = NULL) {
37     $title, $headcomments;
38
39     if ($categories == NULL) {
40         $mn = 'term_id';
41         $query = "SELECT * FROM $wpdb->term_taxonomy
42             JOIN $wpdb->terms ON ( $wpdb->term_taxonomy.term_id = $wpdb->terms.term_id )
43             WHERE $wpdb->term_taxonomy.taxonomy = 'category' AND $wpdb->terms.term_id > 0 AND count
44             ORDER BY $wpdb->terms.name ASC";
45         $s = $wpdb->get_results($query);
46
47         foreach ($categories as $category) {
48             $link = $category->link;
49             $link . $title . ': ' . $category->name;
50             $link . " href=" . get_category_rss_link(0, $category->term_id, $category->name) . " />
51             $link . "\n";
52
53             $link . " href=" . get_category_rss_link(0, $category->term_id, $category->name) . " />
54             $link . "\n";
55         }
56     }
57 }
```



# “Classic” PHP scripting

- Mix of HTML and PHP code in same file
- Can become messy

```
1 <?php
2 /*
3  * Extract the 'id' value from the query string.
4  * We're filtering to ensure it matches the expected integer
5  */
6 $id = filter_input(INPUT_GET, 'id', FILTER_SANITIZE_NUMBER_INT);
7 // Filter the 'title' value, which can contain special characters
8 $title = filter_input(INPUT_GET, 'title', FILTER_SANITIZE_SPECIAL_CHARS);
9 // Build a simple URL with some good (matching) and bad values
10 $goodURL = baseURL() . '?' . http_build_query(array('id' => $id, 'title' => "Les Misérables"));
11 $badURL = baseURL() . '?' . http_build_query(array('id' => $id, 'title' => "Les Misérables"));
12 function baseURL() {
13     return (@$_SERVER['HTTPS']) ? "https://" : "http://";
14     $_SERVER['SERVER_NAME'] .
15     ($_SERVER['SERVER_PORT'] != 80 ? ":" : "") .
16     $_SERVER['SERVER_PORT'] : null) .
17     $_SERVER['SCRIPT_NAME'];
18 }
19 ?>
20 <!-- Our basic HTML output -->
21 <p>
22     Your searched book <?= $title ? "is titled $title" : "has no title" .
23     has <?= $id ? "the ID $id." : "an invalid ID." ?>
24 </p>
25 <a href="<?= baseURL() ?>">Home</a><br/>
26 <a href="<?= $goodURL ?>">Good Link</a><br/>
27 <a href="<?= $badURL ?>">Bad Link</a><br/>
```

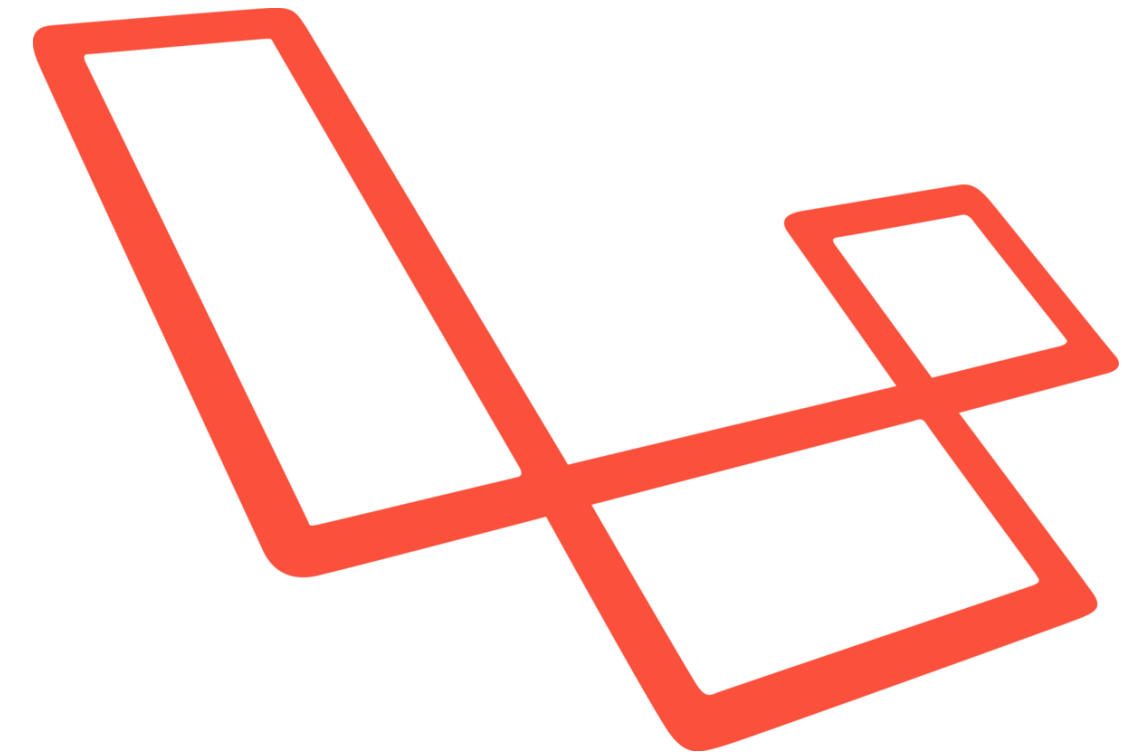




# Laravel to the rescue

---

- PHP Web Framework
- Intended for development of web applications
- Follows Model – View – Controller architectural pattern
- Basically, allows for powerful apps, whilst writing elegant, clean and maintainable code
- Our server-side scripting tool for the reset of the semester

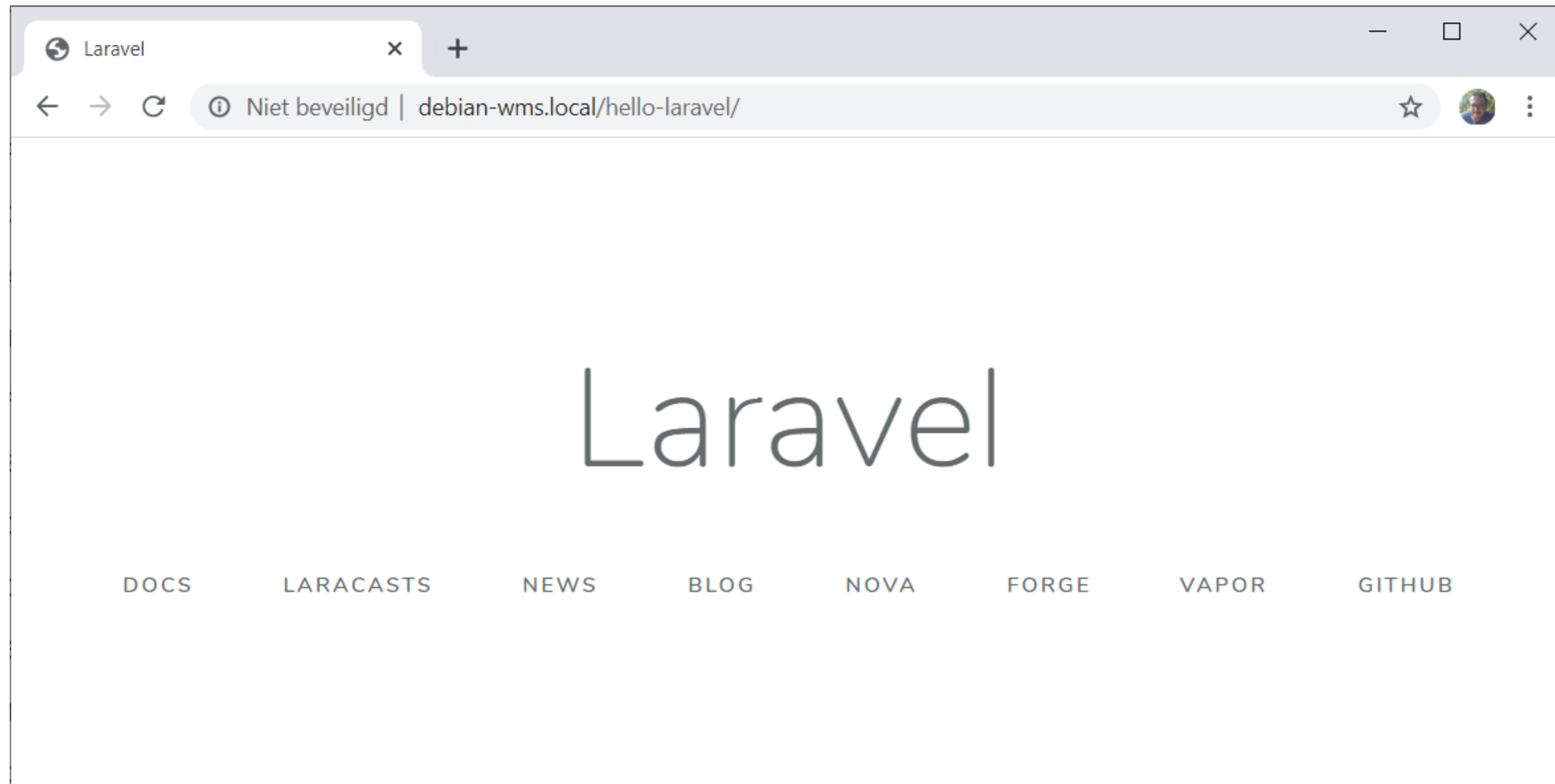


# Creating a new Laravel app

---

- Laravel requires the installation of several technologies
- <https://laravel.com/docs/master#installation>
- Alternative: use Homestead (Ubuntu VM with everything already installed)
- Our Debian VM has been equipped with all the necessary software
- Just use **build-laravel-project** instead of **build-plain-project**

# Creating a new Laravel app



# Routes

---

- Used to tell Laravel which request will trigger which response
- For example:

[GET]        <https://www.fredericvlummens.be/recipes/3/>

Will retrieve recipe with ID = 3

[POST]       <https://www.fredericvlummens.be/recipes/?title=spaghetti&description=cookpasta>

Will create a new recipe with specified title and description

# Web app routes

---

- Defined in the file **./routes/web.php** of your application folder

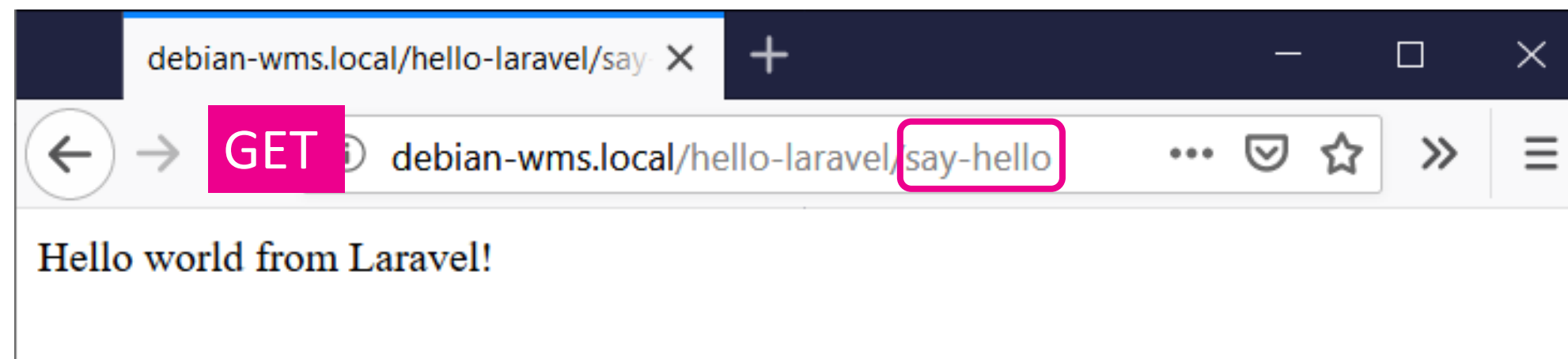
```
web.php x
routes ▸ web.php
1  <?php
2
3  Route::get('/say-hello', function() {
4      return "Hello world from Laravel!";
5  });
6
```



# Web app routes

- Defined in the file `./routes/web.php` of your application folder

```
web.php x
routes ▸ web.php
1  <?php
2
3  Route::get('/say-hello', function() {
4      return "Hello world from Laravel!";
5  });
6
```



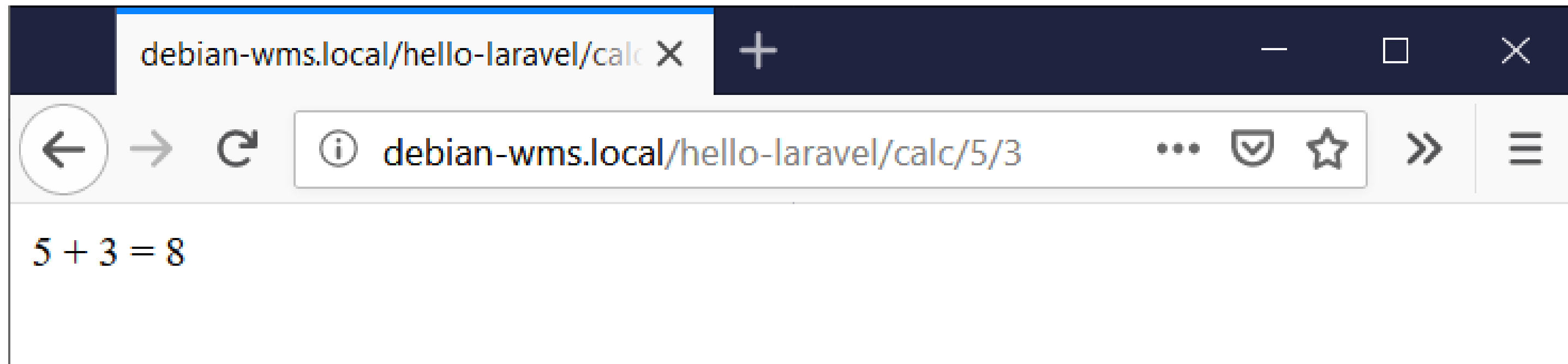
# Defining web app routes: basic recipe

- HTTP verb (get, post, put, delete, ...)
- URL to respond to
- Function to execute if verb and URL are a match

```
Route::get(<<URL>>, function() {  
    // code  
});  
  
Route::post(<<URL>>, function() {  
    // code  
});  
  
Route::put(<<URL>>, function() {  
    // code  
});  
  
Route::delete(<<URL>>, function() {  
    // code  
})  
  
// ...
```

# Defining route parameters

```
Route::get('/calc/{a}/{b}', function($a, $b) {  
    $sum = $a + $b;  
    return "{$a} + {$b} = {$sum}";  
});
```



# Views

---

- Views are the “user interface” part of your application
- Contain the HTML served to and rendered by the browser
- Stored in the folder **/resources/views** of your app
- Carry extension **.blade.php**
- Blade = templating engine
- Allows you to create modular and easily maintainable views

# A simple HTML-only view

hello-world.blade.php x

resources ▸ views ▸ hello-world.blade.php

```
1  <!DOCTYPE html>
2  <html lang="en">
3  <head>
4      <meta charset="UTF-8">
5      <meta name="viewport" content="width=device-width, initial-scale=1.0">
6      <meta http-equiv="X-UA-Compatible" content="ie=edge">
7      <title>Hello World</title>
8  </head>
9  <body>
10     <h1>Hello World</h1>
11 </body>
12 </html>
```



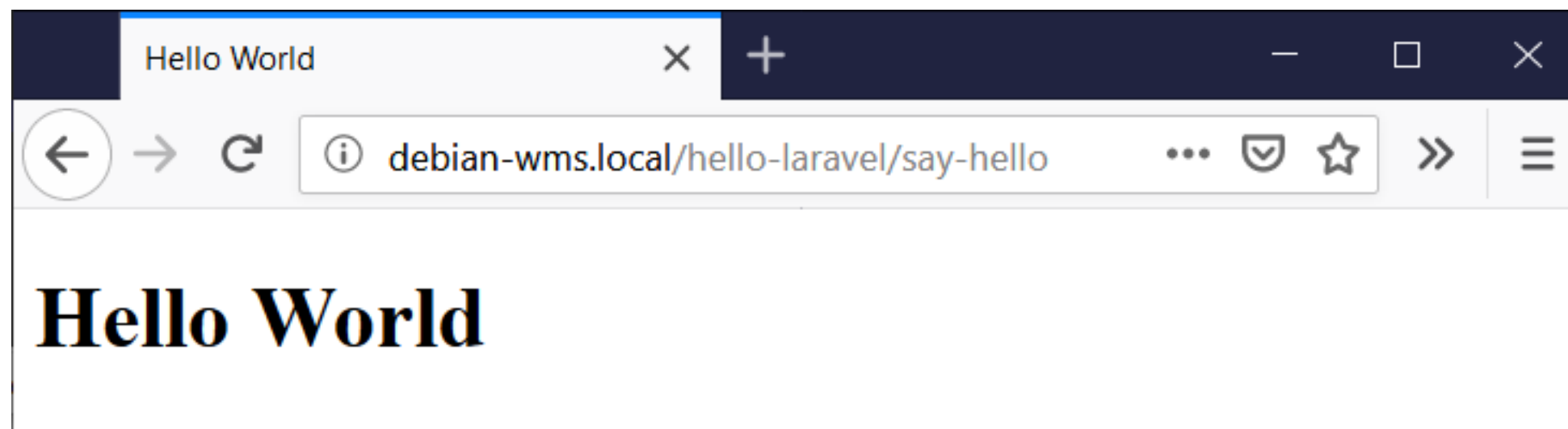
# Linking the view to a route

- In `/routes/web.php`:

```
<?php

Route::get('/say-hello', function() {
    return view("hello-world");
});
```

- Result:



# Passing data to a view

---

- Views are not limited to HTML only
- Let us personalize our hello world app...
- We define a route with one parameter {name}
- Name is passed to view in associative array (key-value pairs)

```
routes ▸ web.php
1  <?php
2
3  Route::get('/say-hello/{name}', function($name) {
4      |      return view("hello-world", [ "name" => $name ]);
5  });
6
```

# Extracting data in the view

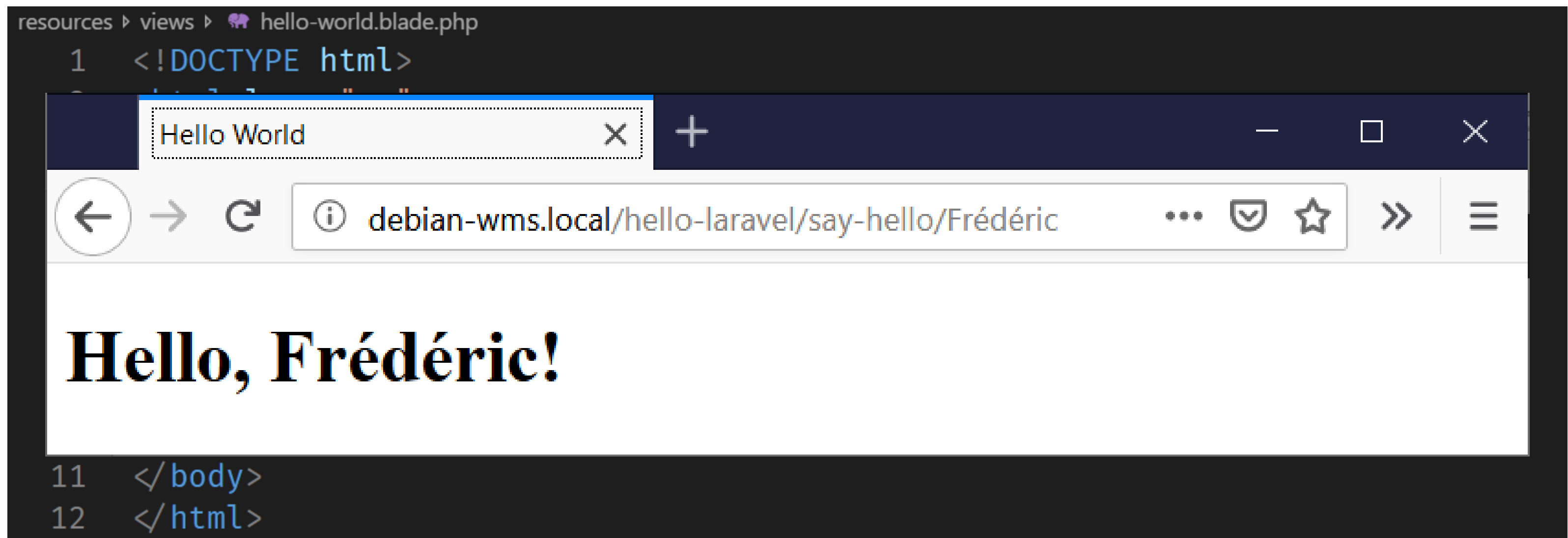
- Using the Blade templating mechanism `{{ $variable_name }}`

resources ▸ views ▸ 🐘 hello-world.blade.php

```
1  <!DOCTYPE html>
2  <html lang="en">
3  <head>
4      <meta charset="UTF-8">
5      <meta name="viewport" content="width=device-width, initial-scale=1.0">
6      <meta http-equiv="X-UA-Compatible" content="ie=edge">
7      <title>Hello World</title>
8  </head>
9  <body>
10     <h1>Hello, {{ $name }}!</h1>
11 </body>
12 </html>
```

# Extracting data in the view

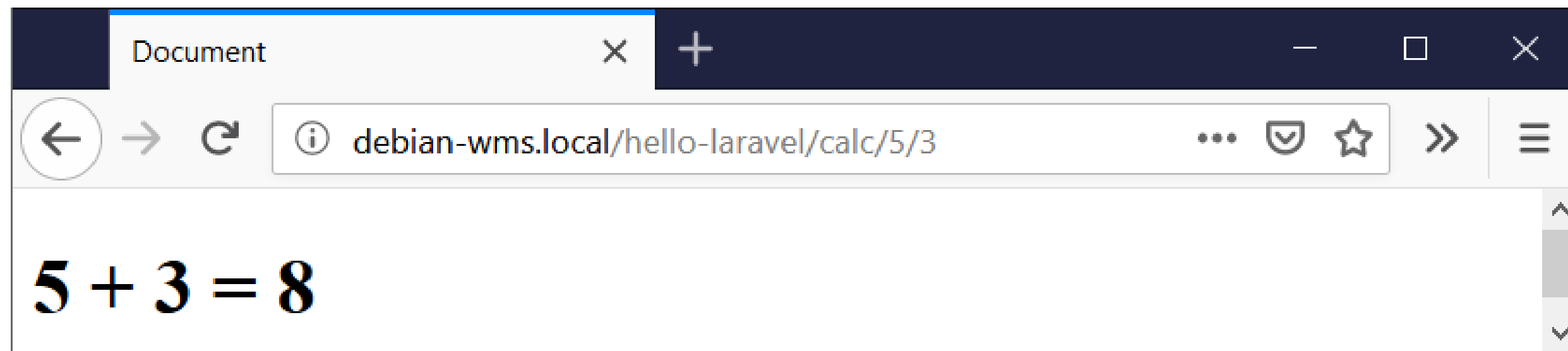
- Using the Blade templating mechanism `{{ $variable_name }}`



## Example 2: passing more than one item

```
Route::get('/calc/{a}/{b}', function($a, $b) {  
    $sum = $a + $b;  
    return view("calc", [ "a" => $a, "b" => $b, "sum" => $sum]);  
});
```

```
<body>  
  
    <h1>{{ $a }} + {{ $b }} = {{ $sum }}</h1>  
  
</body>
```





# Model

---

- The model is the data part of your application
- Retrieval / storage
- Often using (relation) database (e.g. MySQL)
- Will be studied in a future session
- For now, let's stick to data coming in through the URL (or a simple web form)

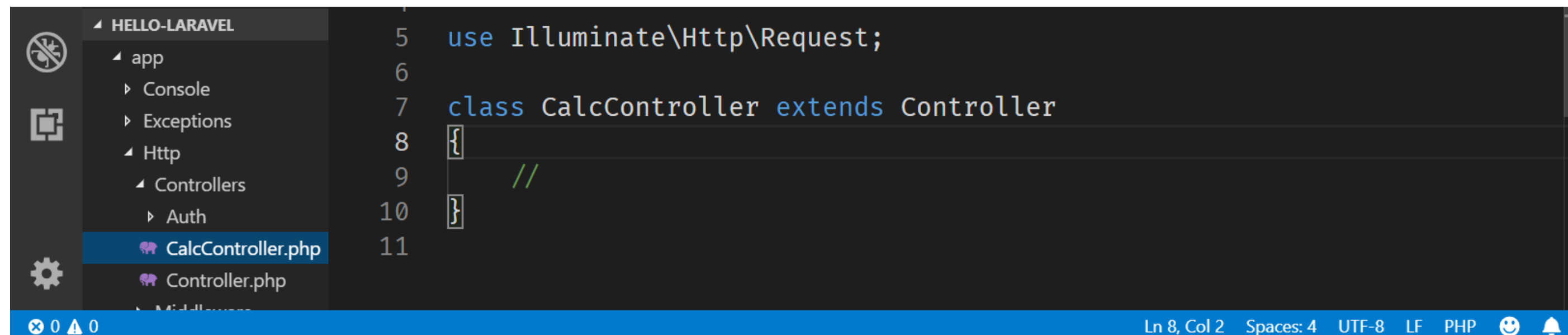
# Controller

---

- Piece of code
- Usually triggered by accessing a certain route (=request)
- Controller receives request
- Can tell the view to update itself / serve a specific page / ...
- Or tell the model to perform some logic (save to DB for example)

# Adding a controller to our example

- In PuTTY, navigate (cd) to the directory of your app:  
**\$ cd ~/code/hello-Laravel**
- Next, execute the following command to generate a controller (here called CalcController):  
**\$ php artisan make:controller CalcController**
- Controller is now available in ~/code/hello-Laravel/app/Http/Controllers:



The screenshot shows an IDE window with a file explorer on the left and a code editor on the right. The file explorer shows the project structure: HELLO-LARAVEL > app > Http > Controllers. The file 'CalcController.php' is selected. The code editor shows the following code:

```
5 use Illuminate\Http\Request;
6
7 class CalcController extends Controller
8 {
9     //
10 }
11
```

The status bar at the bottom indicates 'Ln 8, Col 2', 'Spaces: 4', 'UTF-8', 'LF', 'PHP'.

# Adding a controller to our example (2)

- Writing the controller method:

```
<?php

namespace App\Http\Controllers;

use Illuminate\Http\Request;

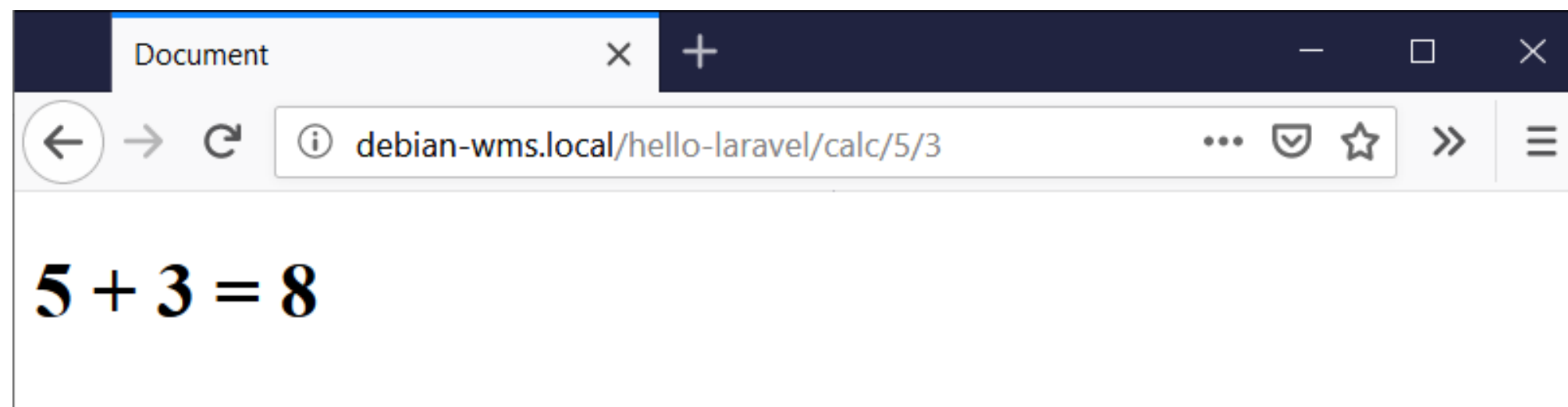
class CalcController extends Controller
{
    function sum($a, $b) {
        $sum = $a + $b;
        return view("calc", ["a" => $a, "b" => $b, "sum" => $sum]);
    }
}
```

# Adding a controller to our example (3)

- Associating a route with the **CalcController's sum** method:

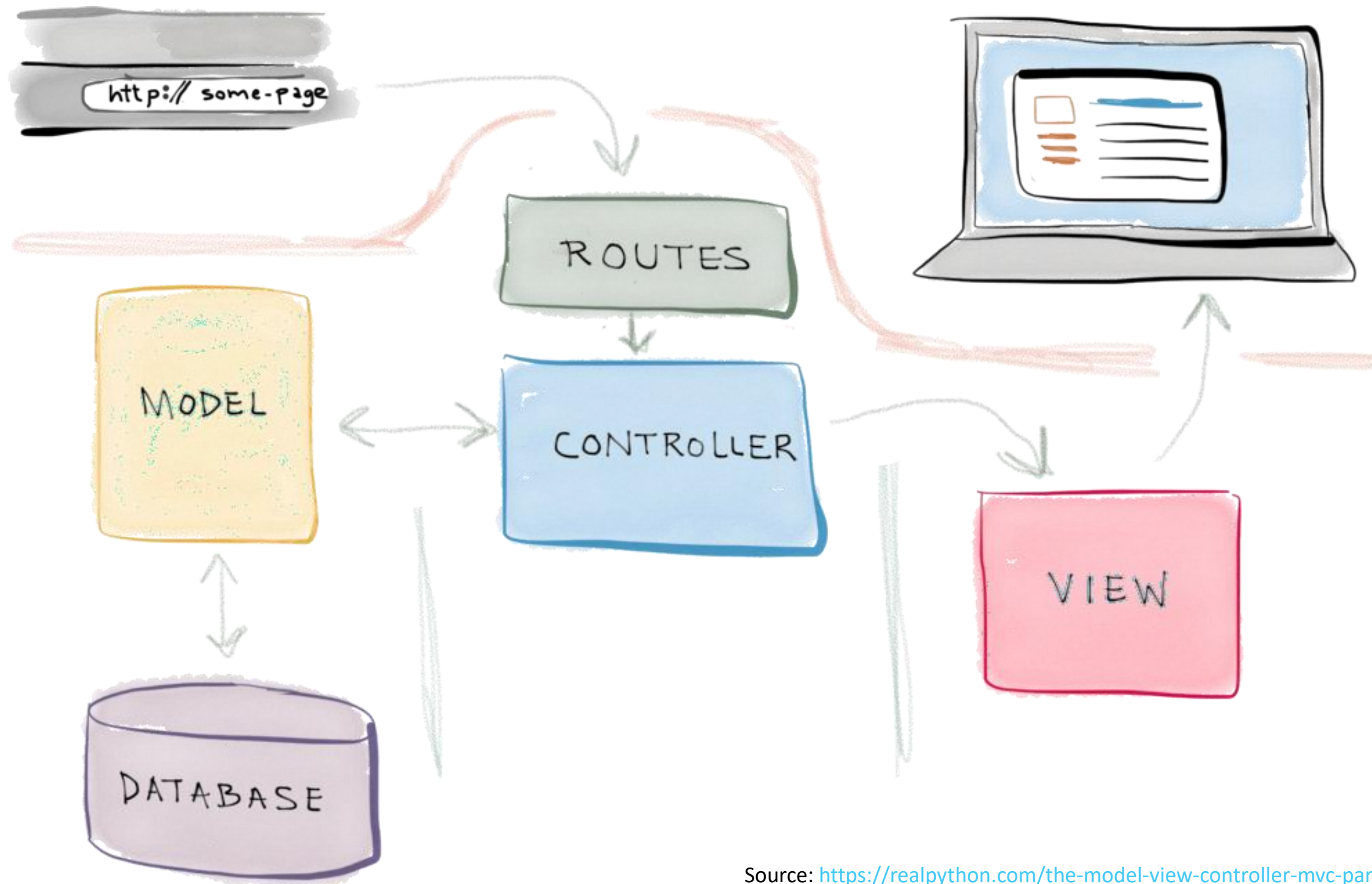
```
routes ▸ web.php
1  <?php
2
3  Route::get('/calc/{a}/{b}', 'CalcController@sum');
4
```

- End result:





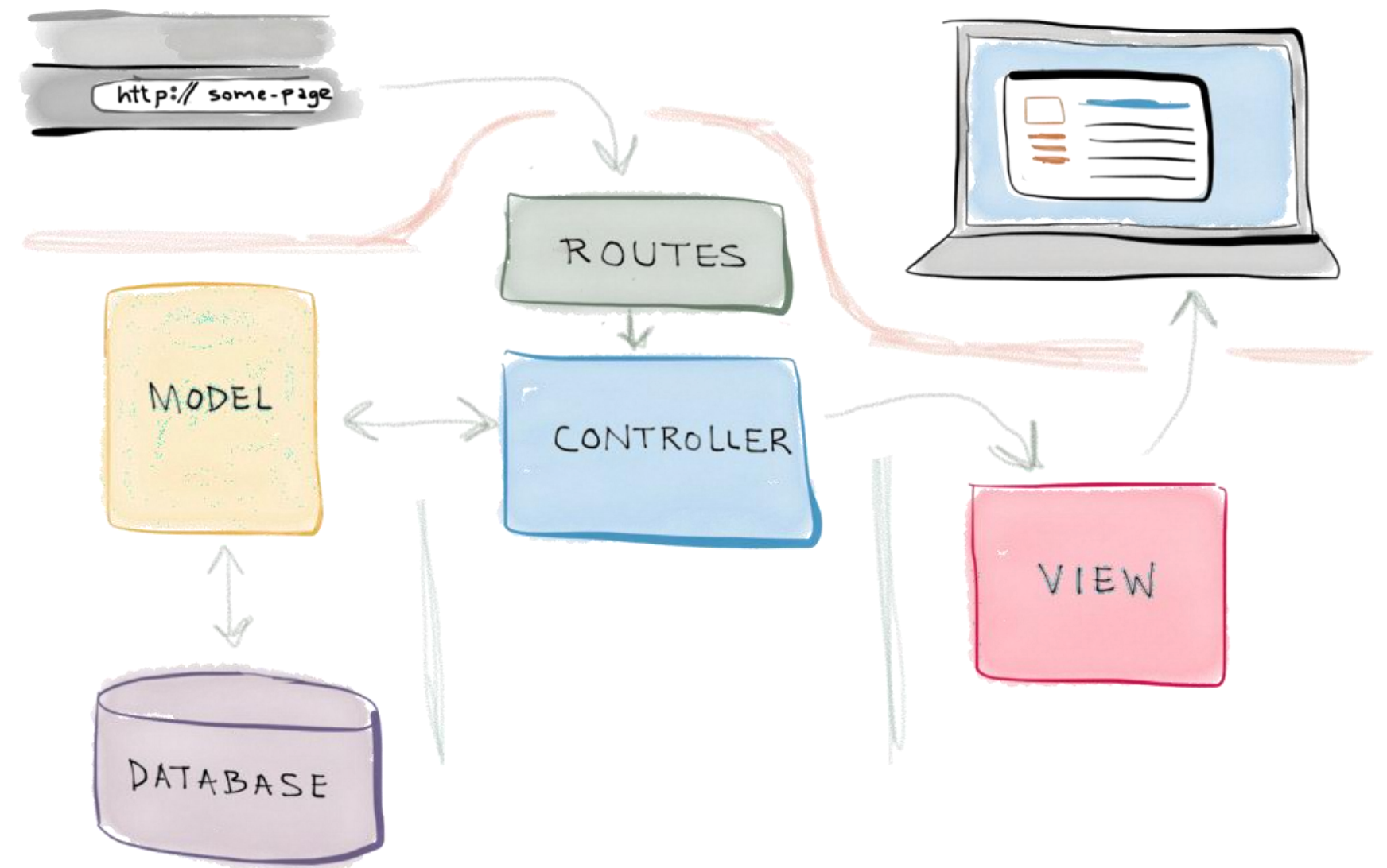
# Model + View + Controller = the MVC pattern



Source: <https://realpython.com/the-model-view-controller-mvc-paradigm-summarized-with-legos/>

# Model + View + Controller = the MVC pattern

- Model
  - Manages data
  - Receives input from controller
- View
  - Representation of model
  - Can be in various formats: web page, JavaFX GUI, command line app, ...
- Controller
  - Responds to user input
  - Manipulates the model
  - Tells view to update itself/serve specific page



# Enhancing our web app

---

- Adding a form in the view
- Allow users to specify numbers to add via form

```
<h1>Calculator app</h1>

<form method="post" action="calc">
    @csrf

    <label for="number1">Number 1:</label>
    <input type="number" id="number1" name="number1" />
    <br />
    <label for="number2">Number 2:</label>
    <input type="number" id="number2" name="number2" />
    <br />
    <input type="submit" value="Add" />
</form>
```

# Enhancing our web app

- Adding a form in the view
- Allow users to specify numbers to add via form

```
<h1>Calculator app</h1>
```

```
<form method="post" action="calc">
```

```
    @csrf
```

Adds protection against cross-site request forgery  
More info: <https://laravel.com/docs/master/csrf>

```
    <label for="number1">Number 1:</label>
```

```
    <input type="number" id="number1" name="number1" />
```

```
    <br />
```

```
    <label for="number2">Number 2:</label>
```

```
    <input type="number" id="number2" name="number2" />
```

```
    <br />
```

```
    <input type="submit" value="Add" />
```

```
</form>
```

# Enhancing our web app

- Adding a form in the view
- Allow users to specify numbers to add via form

```
<h1>Calculator app</h1>

<form method="post" action="calc">
  @csrf
  <label for="number1">Number 1:</label>
  <input type="number" id="number1" name="number1" />
  <br />
  <label for="number2">Number 2:</label>
  <input type="number" id="number2" name="number2" />
  <br />
  <input type="submit" value="Add" />
</form>
```

POST request (cfr. class on HTTP verbs)



# Enhancing our web app

- Adding a form in the view
- Allow users to specify numbers to add via form

```
<h1>Calculator app</h1>

<form method="post" action="calc">
  @csrf

  <label for="number1">Number 1:</label>
  <input type="number" id="number1" name="number1" />
  <br />
  <label for="number2">Number 2:</label>
  <input type="number" id="number2" name="number2" />
  <br />
  <input type="submit" value="Add" />
</form>
```

Route

# Enhancing our web app

---

- Defining the routes:

```
<?php  
  
Route::get('/calc', 'CalcController@showForm');  
  
Route::post('/calc', 'CalcController@sum');
```

- If the resource **/calc** is requested via **GET** → execute function **showForm** of **CalcController**
- If the resource **/calc** is requested via **POST** (=submit) → execute function **sum** of **CalcController**

# Enhancing our web app

---

- Writing the controller functions:

```
class CalcController extends Controller
{
    function showForm() {
        return view("calc");
    }

    function sum(Request $request) {
        $a = $request → input("number1");
        $b = $request → input("number2");

        $sum = $a + $b;
        return view("calc", ["a" ⇒ $a, "b" ⇒ $b, "sum" ⇒ $sum]);
    }
}
```



# Enhancing our web app

- Writing the controller functions:

```
class CalcController extends Controller
{
    function showForm() {
        return view("calc");
    }

    function sum(Request $request) {
        $a = $request → input("number1");
        $b = $request → input("number2");

        $sum = $a + $b;
        return view("calc", ["a" ⇒ $a, "b" ⇒ $b, "sum" ⇒ $sum]);
    }
}
```

Executed upon GET

Executed upon POST

# Enhancing our web app

- Writing the controller functions:

```
class CalcController extends Controller
{
    function showForm() {
        return view("calc");
    }

    function sum(Request $request) {
        $a = $request->input('number1');
        $b = $request->input("number2");

        $sum = $a + $b;
        return view("calc", ["a" => $a, "b" => $b, "sum" => $sum]);
    }
}
```

```
<h1>Calculator app</h1>

<form method="post" action="calc">
    @csrf

    <label for="number1">Number 1:</label>
    <input type="number" id="number1" name="number1" />
    <br />
    <label for="number2">Number 2:</label>
    <input type="number" id="number2" name="number2" />
    <br />
    <input type="submit" value="Add" />
</form>
```

# Enhancing our web app

- Showing the results of the calculation:

```
<h1>Calculator app</h1>

@isset($sum)
    <h2>{{ $a }} + {{ $b }} = {{ $sum }}</h2>
@endisset

<form method="post" action="calc">
    @csrf

    <label for="number1">Number 1:</label>
    <input type="number" id="number1" name="number1" />
    <br />
    <label for="number2">Number 2:</label>
    <input type="number" id="number2" name="number2" />
    <br />
    <input type="submit" value="Add" />
</form>
```

# Enhancing our web app

- Showing the results of the calculation:

```
<h1>Calculator app</h1>
@isset($sum)
    <h2>{{ $a }} + {{ $b }} = {{ $sum }}</h2>
@endisset

<form method="post" action="calc">
    @csrf

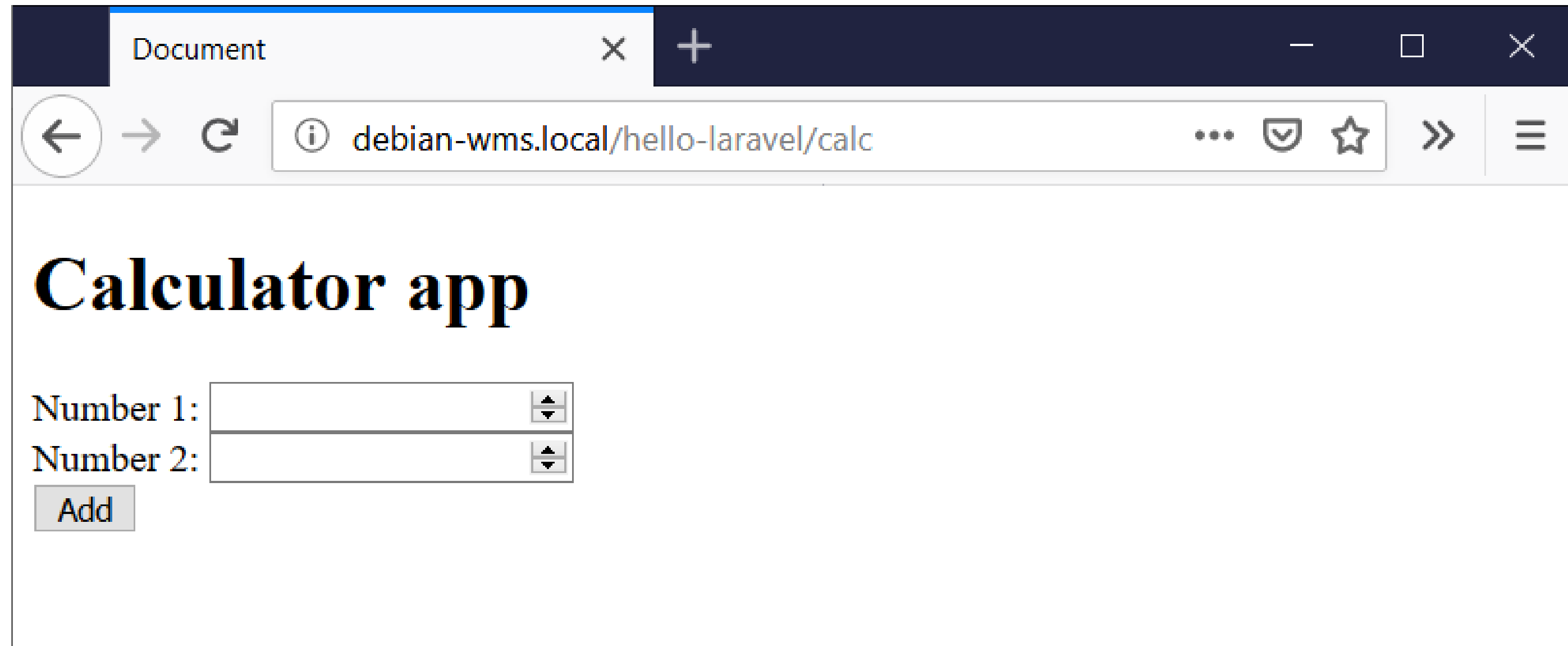
    <label for="number1">Number 1:</label>
    <input type="number" id="number1" name="number1" />
    <br />
    <label for="number2">Number 2:</label>
    <input type="number" id="number2" name="number2" />
    <br />
    <input type="submit" value="Add" />
</form>
```



Condition in Blade syntax

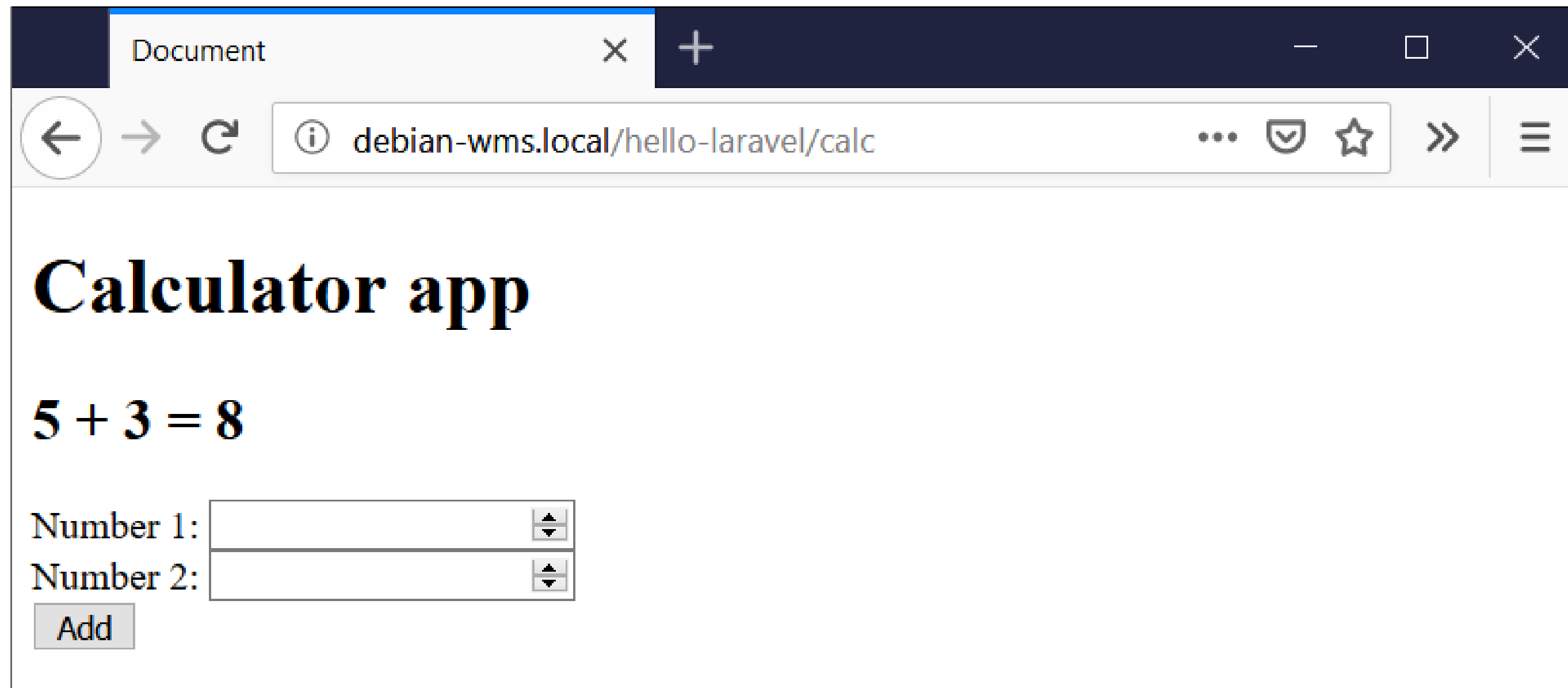
# Enhancing our web app

- First visit (GET request):



# Enhancing our web app

- After submit (POST request):



# Questions?

---

