

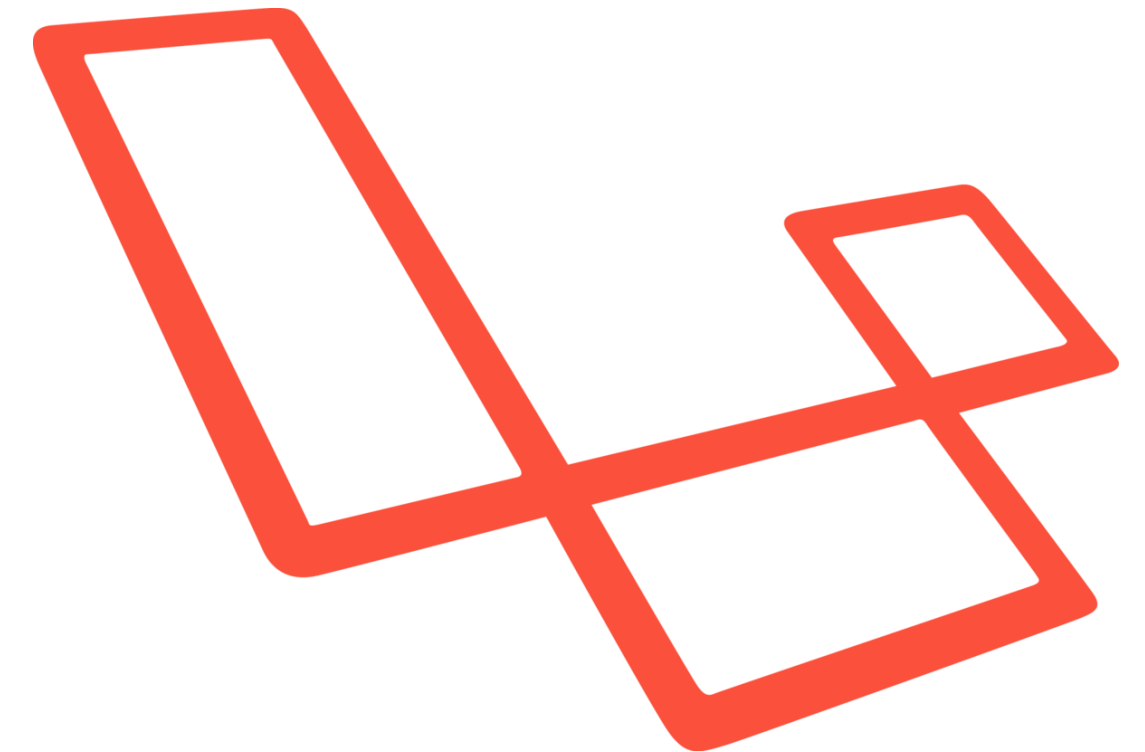


Laravel: Templating and Eloquent (DB)

Web, Mobile and Security
Frédéric Vlummens

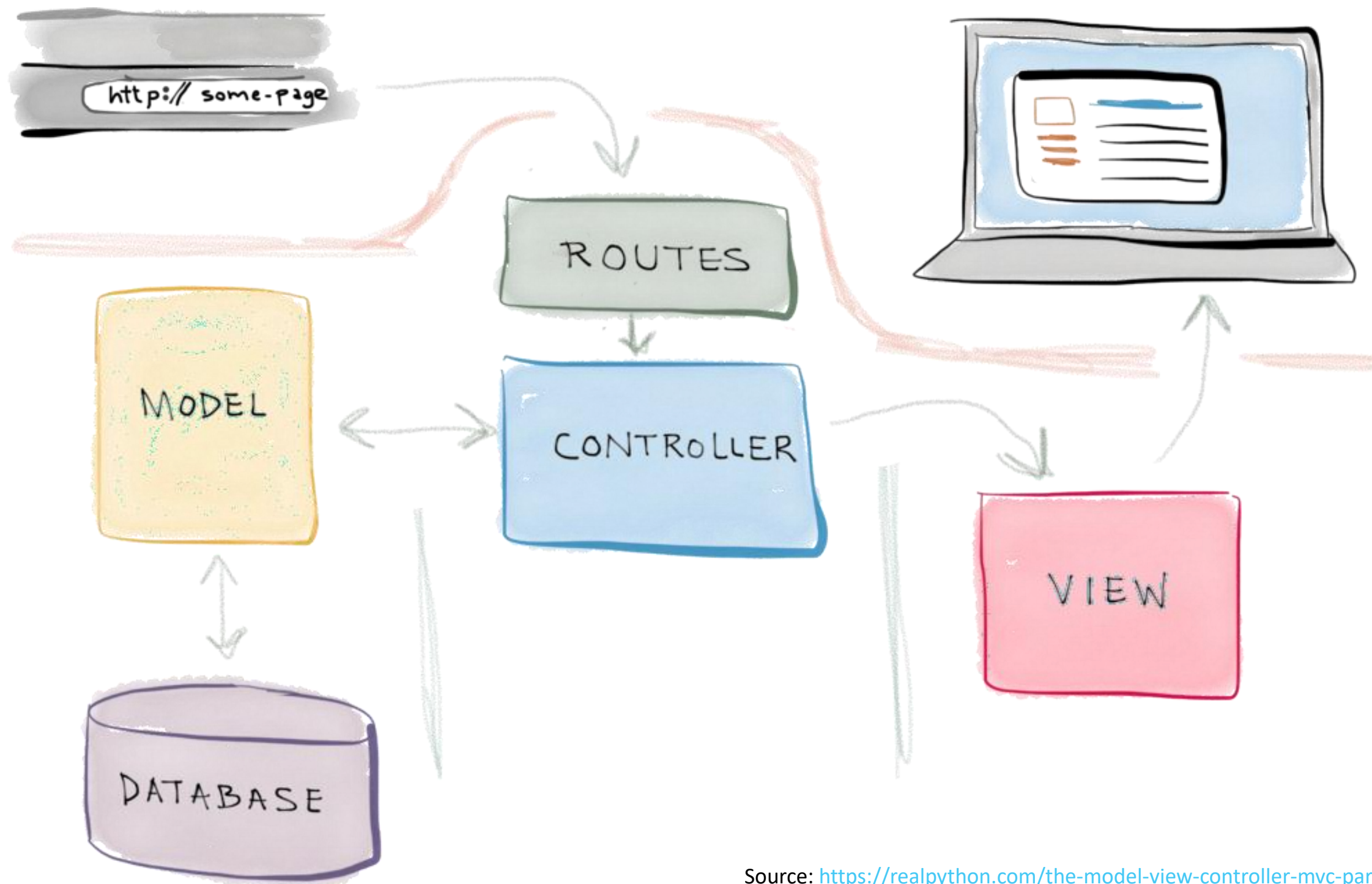
Agenda

- Model – View – Controller: recap
- Blade templates
 - @isset, @for, @foreach, ...
 - Using sections to build modular views
 - Referencing assets (CSS, images, JavaScript, ...)
- Named routes
- Eloquent – Database Access



Model – View – Controller: recap

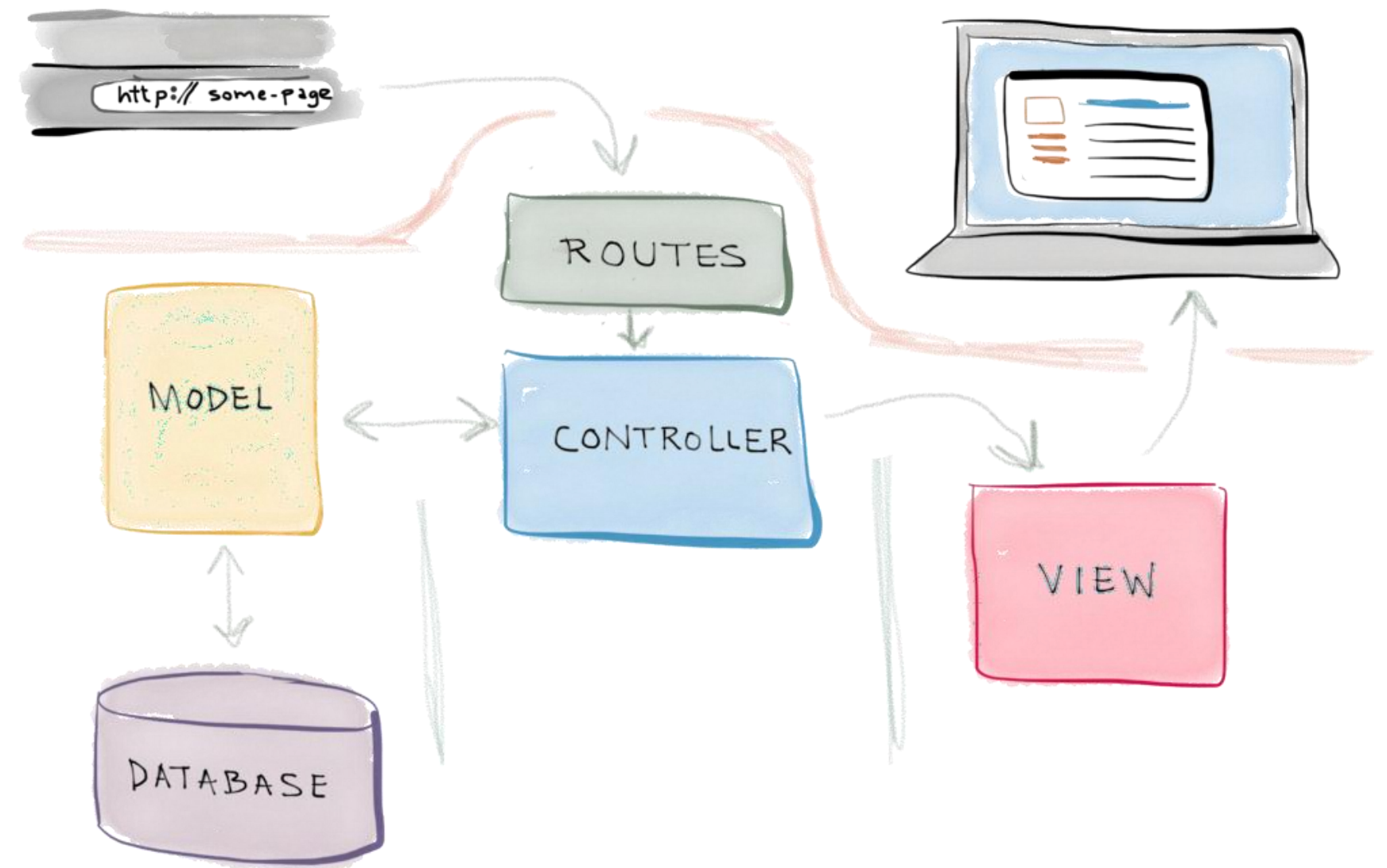
Model – View – Controller: recap



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

Model – View – Controller: recap

- 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



Templating

Blade templates

- Blade = templating mechanism
- Allows us to build modular views
- No or a minimum of PHP code in the views themselves
- Consists mainly of regular HTML + Blade expressions and control structures

Using @if

- Allows you to check a condition
- Comparable to regular if-statements in PHP, but:
 - No { curly braces } to indicate block start and end
 - Instead, done using @if and @endif
 - Optionally, use @elseif and/or @else for multiple checks

```
@if (count($vegetables) == 1)
|   <p>One vegetable available</p>
@elseif (count($vegetables) > 1)
|   <p>Multiple vegetables available</p>
@else
|   <p>No vegetables available</p>
@endif
```


Using @switch

- Allows you to check a condition
- Comparable to regular switch in PHP and JavaScript, but:
 - no { curly braces }, case and break to indicate block start end end
 - Instead, done using @switch, @case, @break, @default and @endswitch

```
@switch (count($vegetables))
    @case (0)
        <p>No vegetables available</p>
        @break
    @case (1)
        <p>One vegetable available</p>
        @break
    @default
        <p>Multiple vegetables available</p>
@endswitch
```

Using @for

- Allows you to execute something n number of times in a Blade template
- Typically, loop over a range of integer numbers
- Comparable to regular for-loops in PHP, Java(Script), but:
 - no { curly braces } to indicate block start and end
 - Instead, done using @for and @endfor

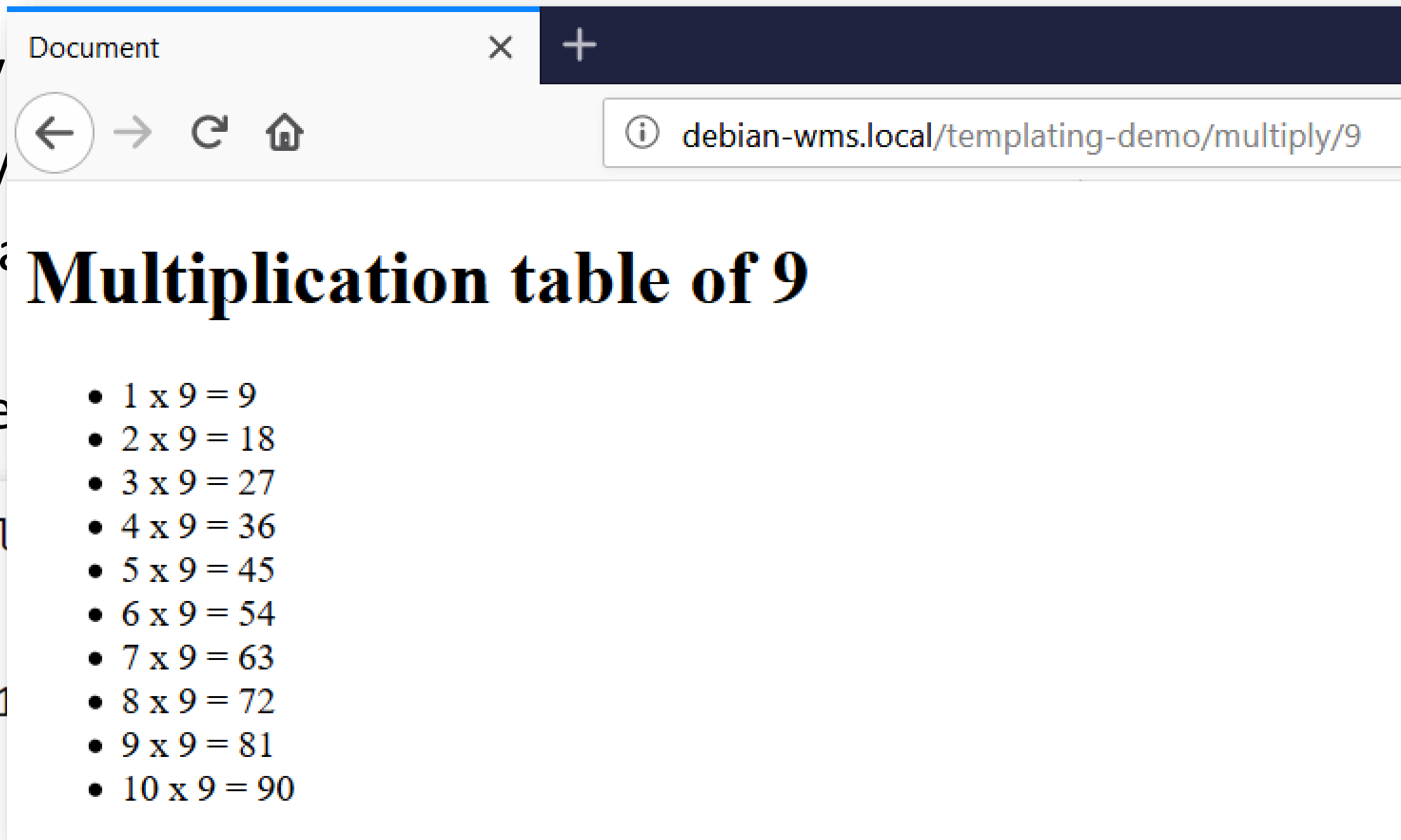
```
<h1>Multiplication table of {{ $number }}</h1>

<ul>
@for($i = 1; $i ≤ 10; $i++)
|   <li>{{ $i }} x {{ $number }} = {{ $i * $number }}
@endfor
</ul>
```

Using @for

- Allows you to loop through a range of numbers
- Typically used for generating lists
- Compared to @foreach
 - no {
 - Instead

```
<h1>Multiplication table of 9</h1>
<ul>
  @for($i = 1; $i <= 10; $i++)
    <li>{
  @endfor
</ul>
```



Using @foreach

- Allows you to loop over an array/collection of items
- Comparable to regular foreach-loops in PHP, but:
 - no { curly braces } to indicate block start and end
 - Instead, done using @foreach and @endforeach

```
class DemoController extends Controller
{
    function multiply($number) {
        return view("multiply", [ "number" => $number ]);
    }

    function vegetables() {
        $vegetables = ["lettuce", "cucumber", "tomato"];
        return view("vegetables", [ "vegetables" => $vegetables ]);
    }
}
```

Using @foreach

- Allows you to loop over an array/collection of items
- Comparable to regular foreach-loops in PHP, but:
 - no { curly braces } to indicate block start and end
 - Instead, done using @foreach and @endforeach

```
<ul>
@foreach($vegetables as $vegetable)
    <li>{{ $vegetable }}</li>
@endforeach
</ul>
```

- lettuce
- cucumber
- tomato

Using @isset

- Allows you to check whether a specific variable passed to /available in the view is defined and not null

```
@isset($name)
|    <p>Hello, {{ $name }}!</p>
@endisset

<form method="post" action="./hello">
|    @csrf
|    <label for="name">Your name:</label>
|    <input type="text" name="name" id="name" />
|    <input type="submit" value="Send" />
|</form>
```

- Comparable to, but easier to write than @if(isset(...)):

```
@if(isset($name))
|    <p>Hello, {{ $name }}!</p>
@endif
```

Using @isset

- Allows you to check whether a specific variable passed to /available in the view is defined and not null

```
@isset($name)
```

@isset always requires a variable name to be passed to it

The system will check on that variable being “set” (=defined and not null) or not

@isset is not a special function to detect whether you have submitted your form or not

```
</form>
```

- Comparable to, but easier to write than @if(isset(...)):

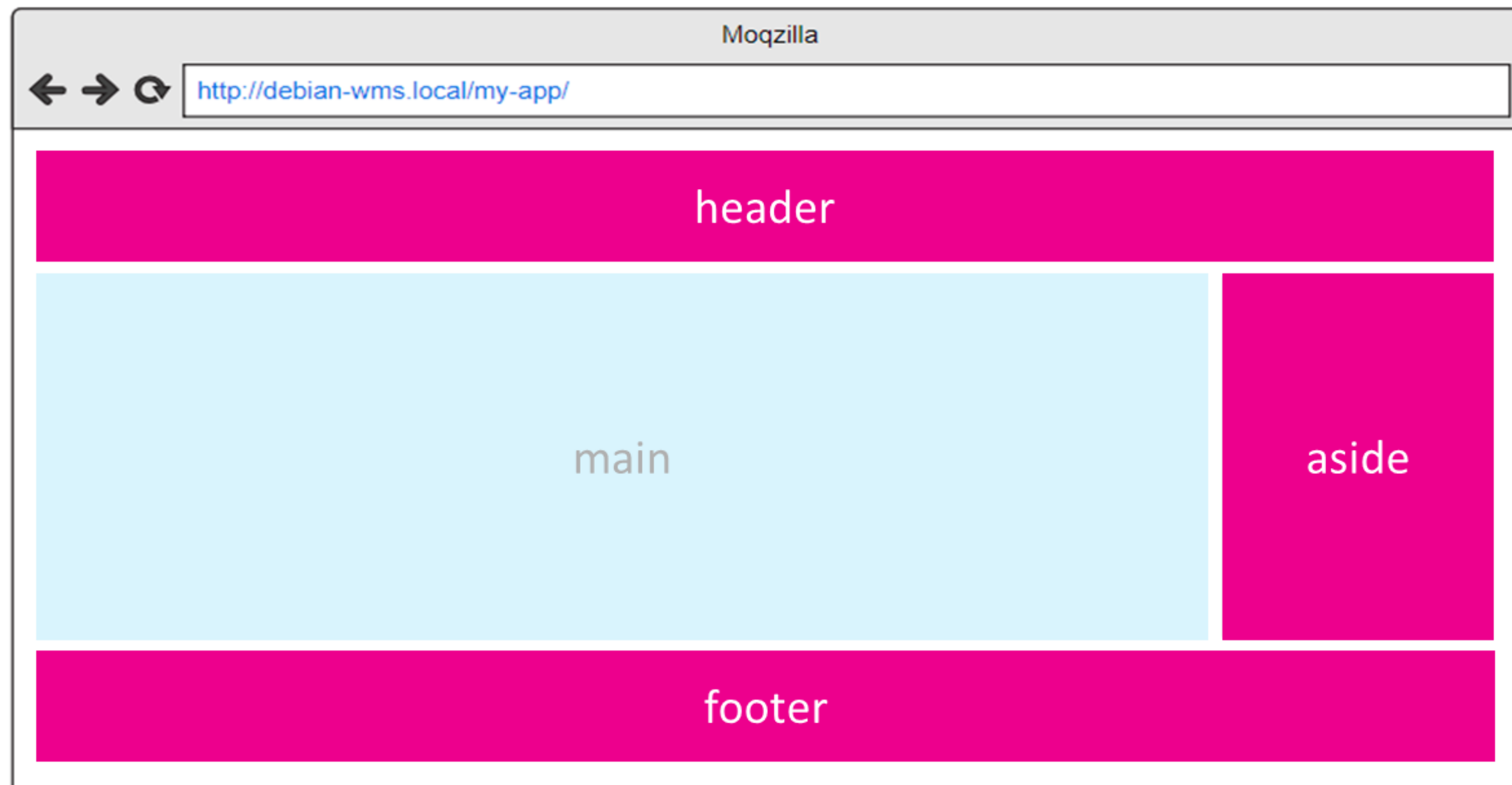
```
@if(isset($name))  
    <p>Hello, {{ $name }}!</p>  
@endif
```

Other directives

- @unless (⇔ @if)
 - @empty (⇔ @isset)
 - @while
-
- More information: <https://laravel.com/docs/master/blade#control-structures>

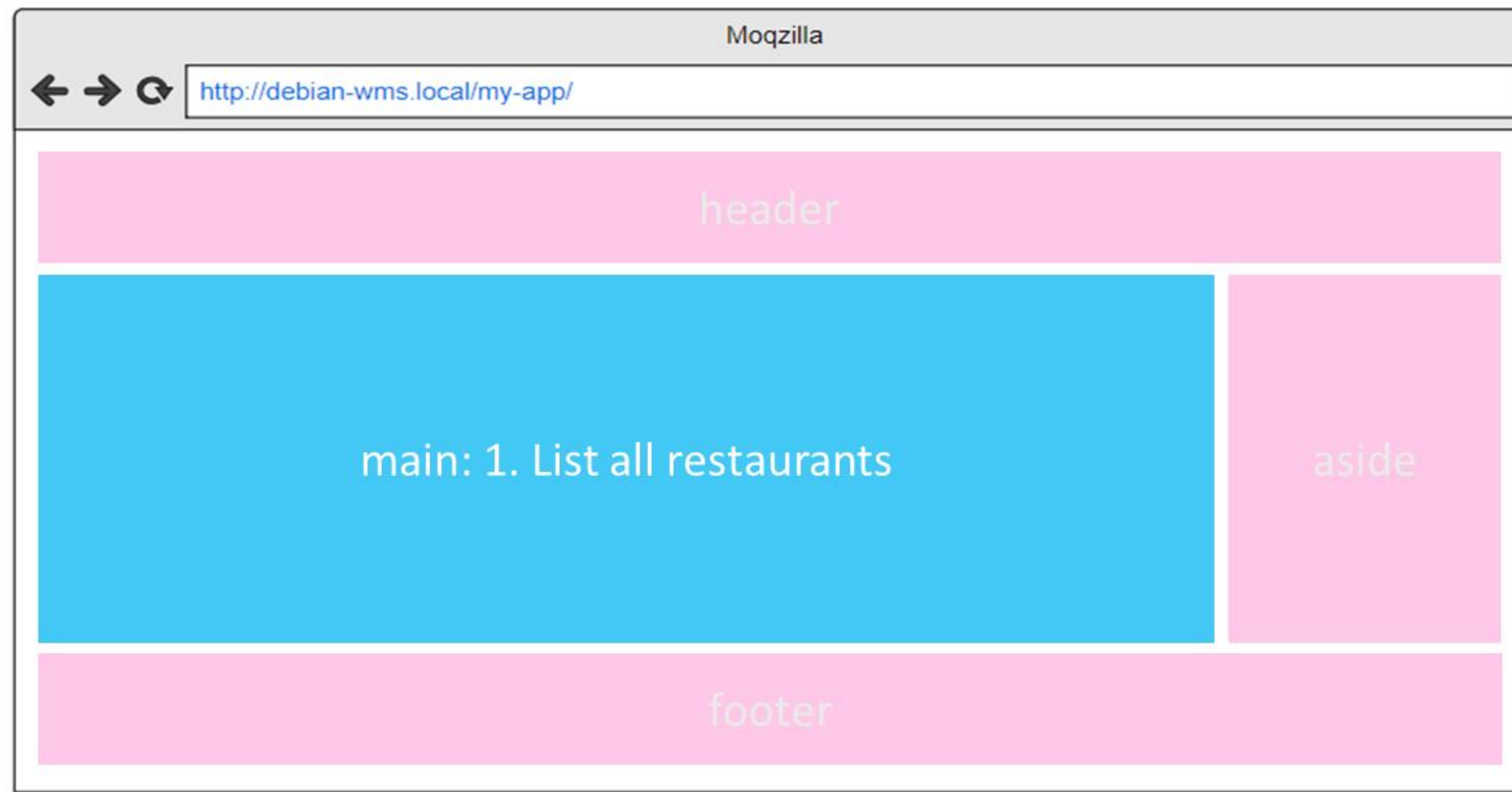
Using sections

- Isolate what stays the same and make it part of the **“master” view**



Using sections

- Isolate what stays the same and make it part of the **“master” view**
- Define **inheriting “child” views** for each situation



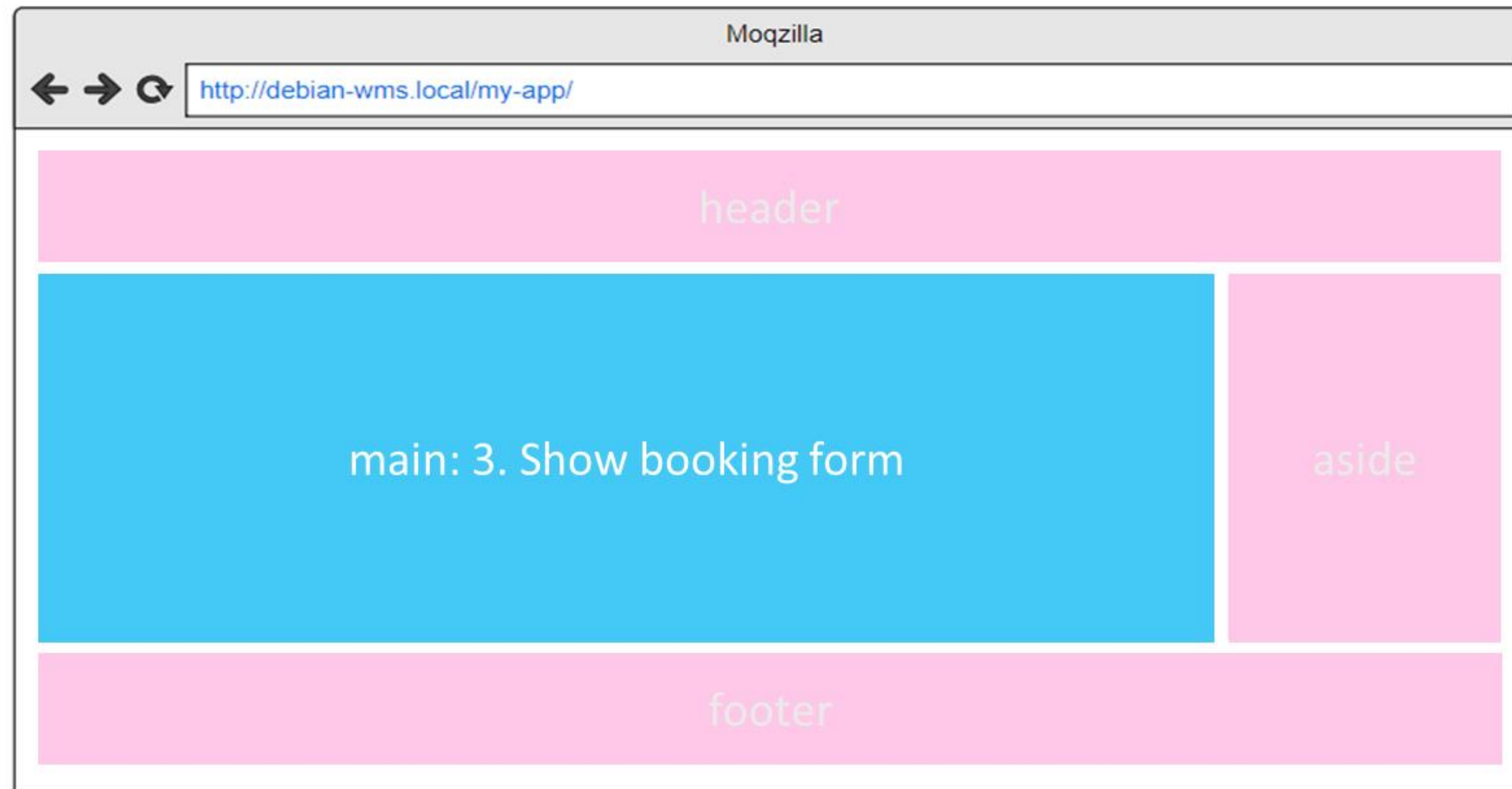
Using sections

- Isolate what stays the same and make it part of the **“master” view**
- Define **inheriting “child” views** for each situation



Using sections

- Isolate what stays the same and make it part of the **“master” view**
- Define **inheriting “child” views** for each situation



Using sections

- Isolate what stays the same and make it part of the **“master” view**
- Define **inheriting “child” views** for each situation



Using sections – building the master view

```
<body>

    <header>
        <h1>Howest.be</h1>
    </header>

    <main>
        @yield("main")
    </main>

    <footer>
        <hr />
        &copy; 2019 Howest University of Applied Sciences
    </footer>

</body>
```

/resources
/views
/master.blade.php

Using sections – building the child view

```
<body>
  <header>
    <h1>Howest.be</h1>
  </header>

  <main>
    @yield("main")
  </main>

  <footer>
    <hr />
    &copy; 2019 Howest University of Applied Sciences
  </footer>
</body>
```

/resources
/views
/master.blade.php

```
Route::get("/howest-ac", function() {
    return view("courses.acs");
});
```

```
@extends("master")
```

```
@section("main")
```

```
    <h2>Applied Computer Sciences</h2>
```

```
    @for($i = 0; $i < 10; $i++)
        {{ $i }}
```

```
    @endfor
```

```
@endsection
```

/resources
/views
/courses
/acs.blade.php

Referencing assets

- CSS files
- Images
- JavaScript files
- Web fonts
- ...



To be stored in the `/public` directory of your app

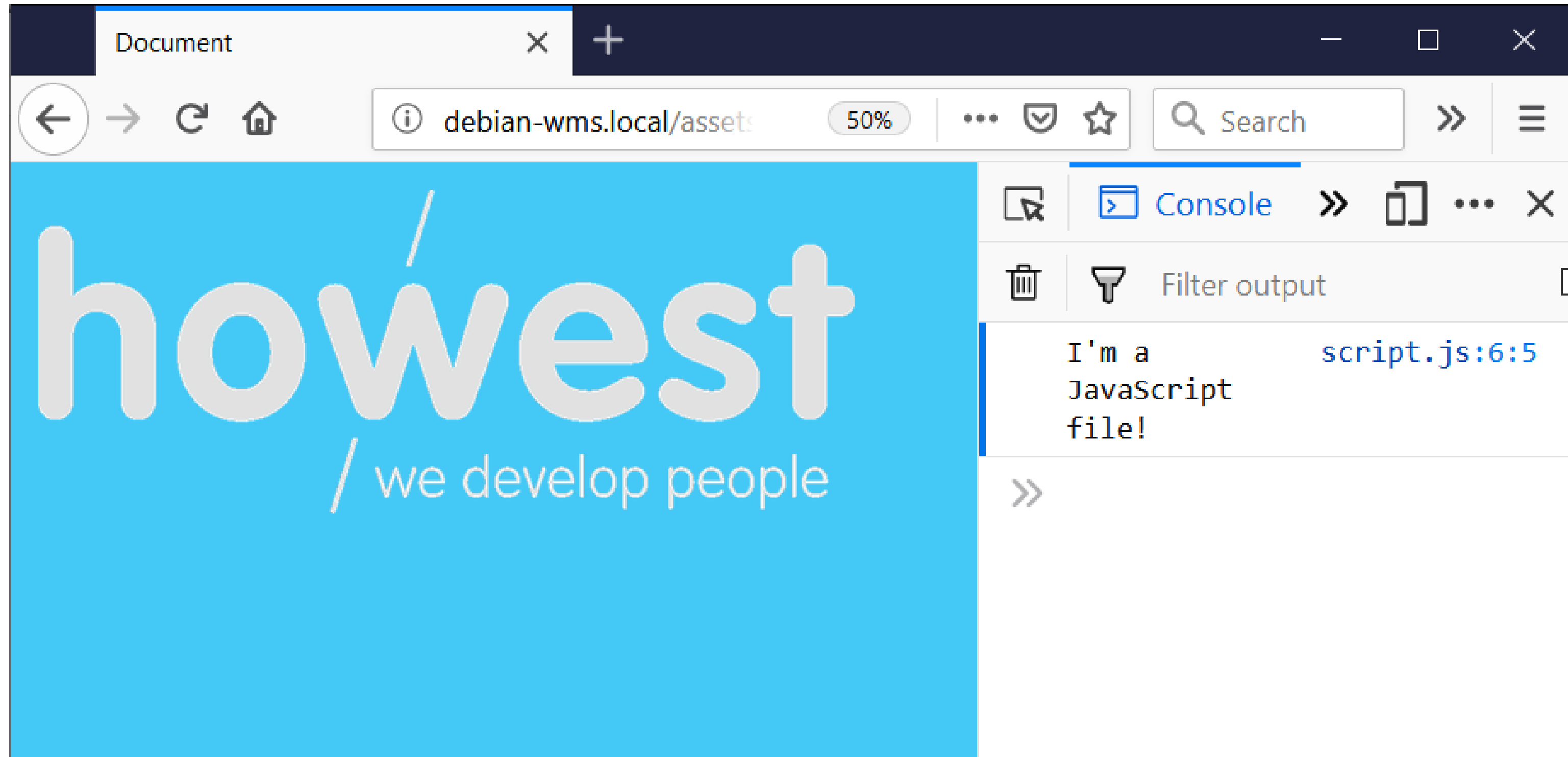
- To reference them from your Blade views, use the **asset** function
- It will automatically create the correct path

Referencing assets

- app
- bootstrap
- config
- database
- ▾ public
 - ▾ css
 - # reset.css
 - # screen.css
 - ▾ images
 - howest_logo.png
 - ▾ js
 - JS script.js
 - ⚙ .htaccess
 - ★ favicon.ico
 - 🐘 index.php
 - ☰ robots.txt
- resources

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <meta http-equiv="X-UA-Compatible" content="ie=edge">
  <title>Document</title>
  <link rel="stylesheet" type="text/css" href="{{ asset('css/reset.css') }}" />
  <link rel="stylesheet" type="text/css" href="{{ asset('css/screen.css') }}" />
</head>
<body>
  
  <script src="{{ asset('js/script.js') }}"></script>
</body>
</html>
```

Referencing assets



Named routes

Named routes

- You can give your routes unique names in web.php using the **name** method:

```
Route::get("/register", "RegController@showForm");

Route::post("/register", "RegController@processForm")
    → name("process-registration");
```

- You can then reference the route by using function **route** with the appropriate name:

```
<form method="post" action="{{ route('process-registration') }}">
    @csrf
    <label for="address">Your email address:</label>
    <input type="email" id="address" name="address" />
    <input type="submit" value="Register" />
</form>
```

Named routes

- The **route** function translates into the correct path:

```
<form method="post" action="{{ route('process-registration') }}">
    @csrf
    <label for="address">Your email address:</label>
    <input type="email" id="address" name="address" />
    <input type="submit" value="Register" />
</form>
```

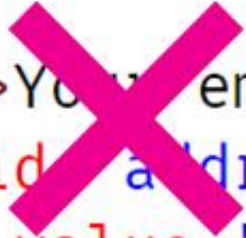


```
<form method="post" action="http://debian-wms.local/named-routes/register">
    <input type="hidden" name="_token" value="HY34vVkYNehxWWVBndkPhDey469NH2Rbwl7yElEo">
    <label for="address">Your email address:</label>
    <input type="email" id="address" name="address" />
    <input type="submit" value="Register" />
</form>
```

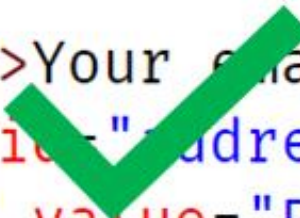

Generating URLs: summary (1/2)

- Use **route** with named routes to generate your URLs, instead of hardcoding:

```
<form method="post" action="./register">
  @csrf
  <label for="address">Your email address:</label>
  <input type="email" id="address" name="address" />
  <input type="submit" value="Register" />
</form>
```



```
<form method="post" action="{{ route('process-registration') }}">
  @csrf
  <label for="address">Your email address:</label>
  <input type="email" id="address" name="address" />
  <input type="submit" value="Register" />
</form>
```



Generating URLs: summary (2/2)

- Use **asset** to reference your assets, instead of hardcoding:

```
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <meta http-equiv="X-UA-Compatible" content="ie=edge">
  <title>Document</title>
  <link rel="stylesheet" type="text/css" href="./css/screen.css" />
</head>
```



```
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <meta http-equiv="X-UA-Compatible" content="ie=edge">
  <title>Document</title>
  <link rel="stylesheet" type="text/css" href="{{ asset('css/screen.css') }}" />
</head>
```



Database access

Database access

- Problem with last week's Restaurant Booker:
 - Information not stored
 - How can the restaurant retrieve a list of bookings?
- Solution:
 - Store individual bookings in a database

Working with models (the M of MVC)

- A model represents the data of our application
- It can be used to perform CRUD operations
 - Create
 - Read
 - Update
 - Deleate
- Laravel uses Eloquent, an ORM (Object-Relational Mapper)
- Database tables have corresponding Model, used to interact with table
- You do not write SQL code yourself, but let Eloquent generate it for you

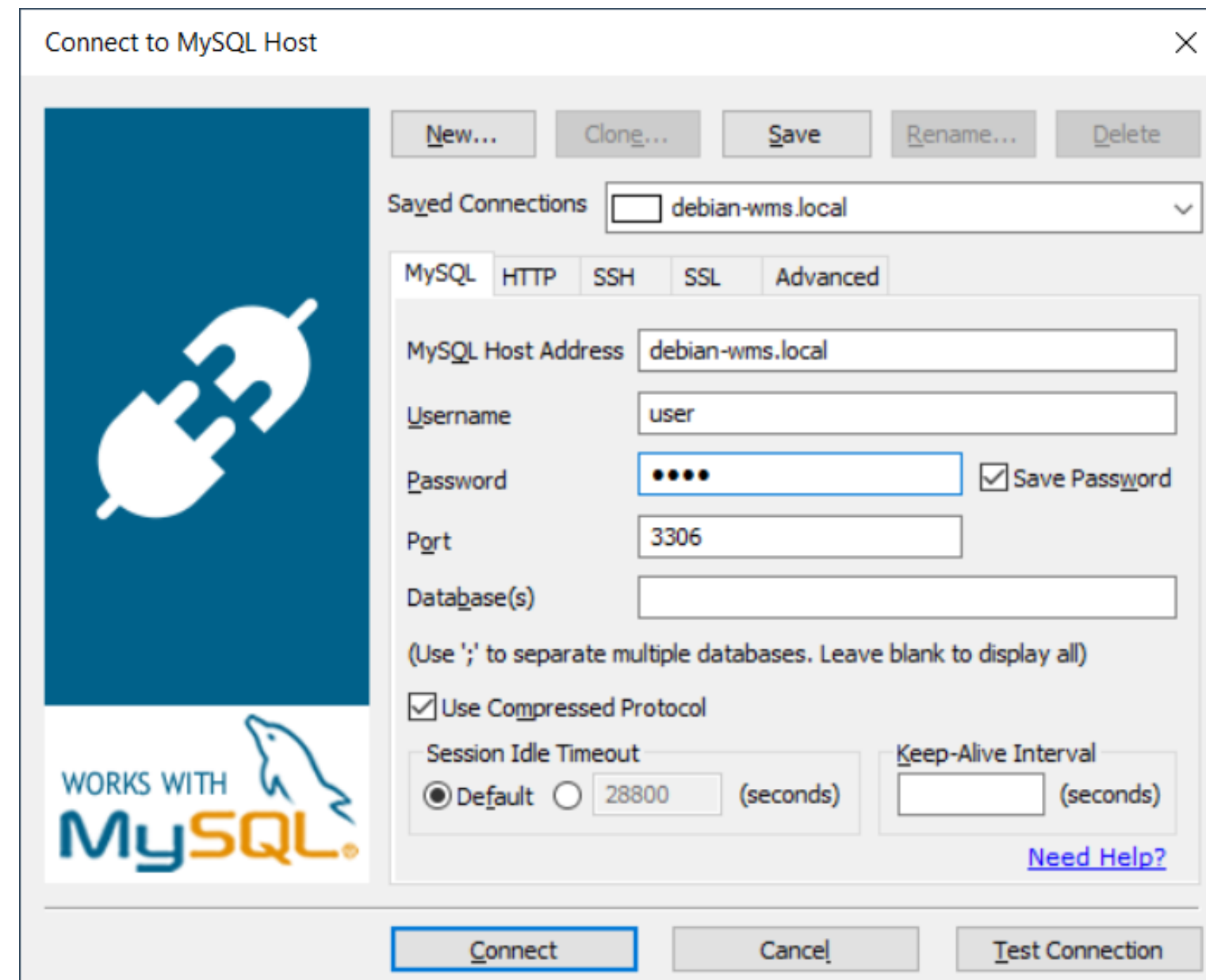
First things first: configuring Laravel to connect to our DB

- Edit **.env** file

<code>DB_CONNECTION=mysql</code>	←	Use MySQL
<code>DB_HOST=127.0.0.1</code>	←	Same server as Apache
<code>DB_PORT=3306</code>	←	Default port
<code>DB_DATABASE=howest-resto</code>	←	Database name
<code>DB_USERNAME=user</code>	←	Database user
<code>DB_PASSWORD=user</code>	←	Database password

Next, build your database

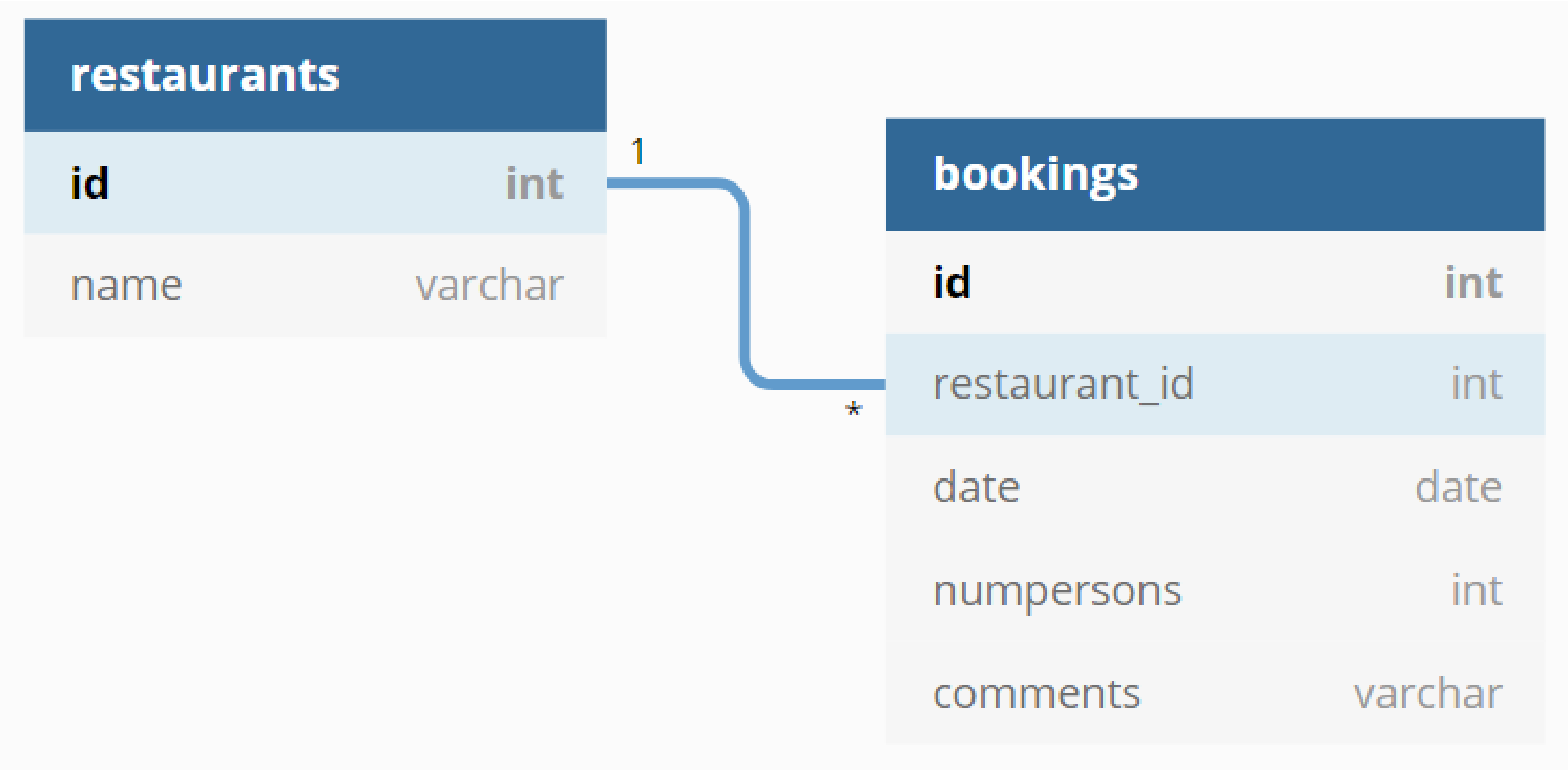
- Your VM contains a MariaDB server (comparable to MySQL)
- How to connect?
 - Make sure your VM is running
 - Using SQLyog, connect using following credentials:
 - Host address: debian-wms.local
 - Username: user
 - Password: user
 - Port: 3306
- **Do not connect to WampServer!**



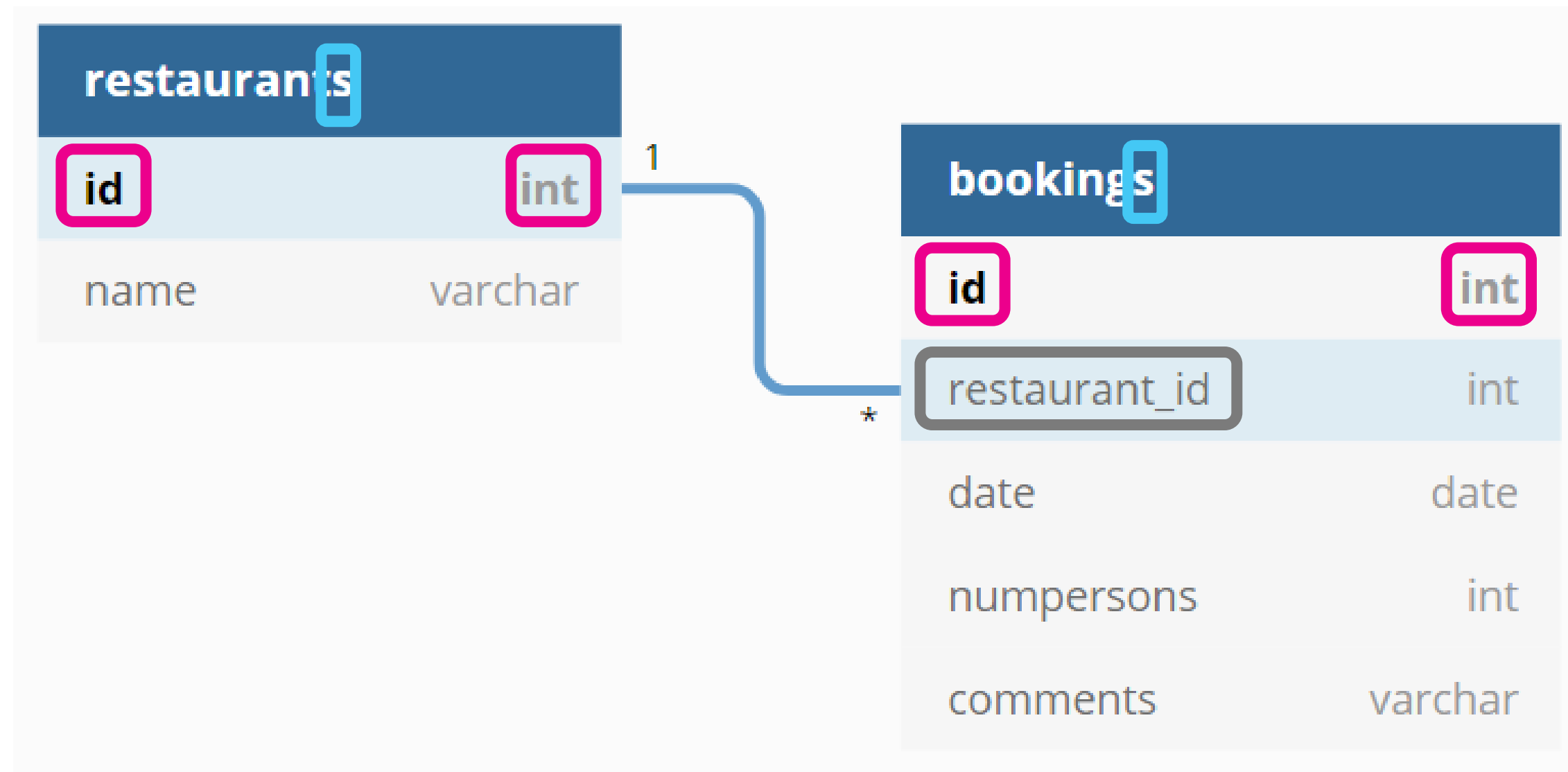
Next, build your database

- Some important conventions for Eloquent to work:
 - Models are singular, e.g. **Restaurant**
 - Corresponding tables are pluralized, e.g. **restaurants**
 - Each table has a primary key called **id** of type **INTEGER AUTO_INCREMENT**
 - One-to-many relationships are handles in the database by taking the singular of the table and suffixing the foreign key field with **_id**
- This is called “convention over configuration”:
 - We do not need to explain Laravel the pluralization rules or the primary keys
 - As long as we follow the rules, Eloquent knows what to do

Our database model



Our database model: conventions



Database table names are pluralized

Primary keys are auto increment integers called **id**

For 1 to many relationships, the foreign key name consists of the related table name in singular, suffixed by **_id**

The 'restaurants' table

Query 1 restaurants x +

Table Name restaurants

Engine InnoDB

Database restobooker

Character Set utf8mb4

Collation utf8mb4_general_ci

1 Columns

2 Indexes

3 Foreign Keys

4 Check Constraint

5 Advanced

6

+ - ▲ ▼

<input type="checkbox"/>	Column Name	Data Type	Length	Default	PK?	Not Null?	Unsigned?	Auto Incr?
<input type="checkbox"/>	id	int	11		<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>
<input type="checkbox"/>	name	varchar	50		<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
<input type="checkbox"/>					<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

The 'bookings' table

Query 1

restaurants

bookings x

New Table

Query 2

+

Table Name

bookings

Engine

InnoDB

Database

restobooker

Character Set

utf8mb4

Collation

utf8mb4_general_ci

1 Columns

2 Indexes

3 Foreign Keys

4 Check Constraint

5 Advanced

6

+ -

▲ ▼

<input type="checkbox"/>	Column Name	Data Type	Length	Default	PK?	Not Null?	Unsigned?	Auto Incr?
<input type="checkbox"/>	id	int	11		<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>
<input type="checkbox"/>	restaurant_id	int	11		<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
<input type="checkbox"/>	date	date			<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
<input type="checkbox"/>	numpersons	int	11		<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
<input type="checkbox"/>	comments	varchar	255		<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
<input type="checkbox"/>	email	varchar	128		<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

Building the corresponding Eloquent Model classes

- Generated using **artisan** (cfr. generation of Controllers)
- Execute the following command **from within the homedir of your application:**
\$ php artisan make:model Restaurant
- This will generate a file called Restaurant.php in ./app
- Inherits Model class → becomes an Eloquent model
- Idem for Booking:
\$ php artisan make:model Booking

```
1  <?php
2
3  namespace App;
4
5  use Illuminate\Database\Eloquent\Model;
6
7  class Restaurant extends Model
8  {
9      //
10 }
11
```

Building our view

```
<form method="post">
  @csrf
  <label for="restaurant">Restaurant:</label>
  <select id="restaurant" name="restaurant">
    @foreach ($restaurants as $restaurant)
      <option value="{{ $restaurant → id }}">{{ $restaurant → name }}</option>
    @endforeach
  </select>
  <label for="numpersons">Number of persons:</label>
  <input type="number" id="numpersons" name="numpersons" />
  <label for="date">Date:</label>
  <input type="date" id="date" name="date" />
  <label for="email">Email address:</label>
  <input type="email" id="email" name="email" />
  <label for="comments">Comments:</label>
  <input type="text" id="comments" name="comments" />
  <input type="submit" value="Add booking" />
</form>
```

In Controller: retrieve restaurants and pass to the view

```
function index() {  
    $restaurants = \App\Restaurant::all();  
  
    return view("index", [ "restaurants" => $restaurants ]);  
}
```

- All you need to do is call the **static method** `all()` on the Restaurant model
- Eloquent will automatically generate the necessary SELECT statement and return the results in the variable `$restaurants`
- Which we then pass to the view

Result

The image shows a side-by-side comparison of a web form and its underlying HTML source code. The left browser window displays the rendered form, while the right window shows the raw HTML code.

Left Window (Rendered Form):

- Restaurant: The Veggie Place ▼
- Number of persons:
- Email address:
- Comments:
-

Right Window (Source Code):

```
11 <form method="post">
12   <input type="hidden" name="_token"
value="ie0klwgcJSmprLbCnNH72kSWBHNsC8azSHgJgCrQ">
13   <label for="restaurant">Restaurant:</label>
14   <select id="restaurant" name="restaurant">
15     <option value="1">The Veggie Place</option>
16     <option value="2">Ribb and More</option>
17     <option value="3">Fishmarket</option>
18   </select>
19   <label for="numpersons">Number of persons:</label>
20   <input type="number" id="numpersons" name="numpersons" />
21   <label for="email">Email address:</label>
22   <input type="email" id="email" name="email" />
23   <label for="comments">Comments:</label>
24   <input type="text" id="comments" name="comments" />
25   <input type="submit" value="Add booking" />
</form>
```

In Controller: process input and add to DB using Eloquent

```
function addBooking(Request $request) {  
    $restaurant_id = $request → input("restaurant");  
    $numpersons = $request → input("numpersons");  
    $email = $request → input("email");  
    $comments = $request → input("comments");  
    $date = $request → input("date");  
  
    $booking = new \App\Booking();  
  
    $booking → restaurant_id = $restaurant_id;  
    $booking → numpersons = $numpersons;  
    $booking → email = $email;  
    $booking → comments = $comments;  
    $booking → date = $date;  
  
    $booking → save();  
}
```


In Controller: process input and add to DB using Eloquent

```
function addBooking(Request $request) {  
    $restaurant_id = $request → input("restaurant");  
    $numpersons = $request → input("numpersons");  
    $email = $request → input("email");  
    $comments = $request → input("comments");  
    $date = $request → input("date");
```

```
$booking = new \App\Booking();
```

Create a new instance of the Booking model

```
$booking → restaurant_id = $restaurant_id;  
$booking → numpersons = $numpersons;  
$booking → email = $email;  
$booking → comments = $comments;  
$booking → date = $date;
```

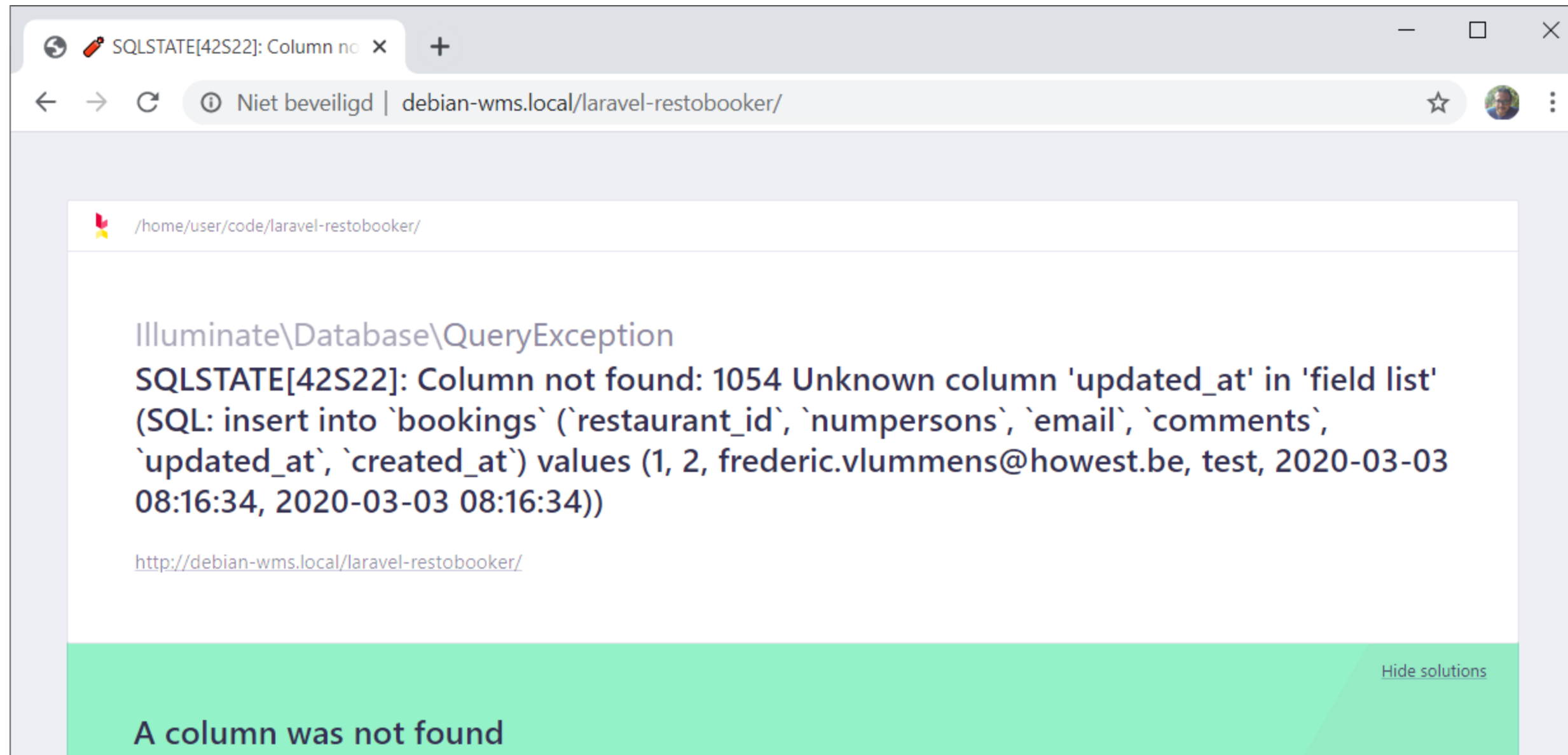
Populate properties

```
$booking → save();
```

Ask Eloquent to save, which generates the necessary SQL
INSERT statement

```
}
```

Trying it out...



Trying it out...

Explanation of this error:

By default, Eloquent also requires columns `updated_at` and `created_at` to exist (for timestamp purposes).

If you do not want them, you need to tell Eloquent so, since we are going against the convention over configuration principle.

Solution: add `$timestamps = false` in the Model classes...

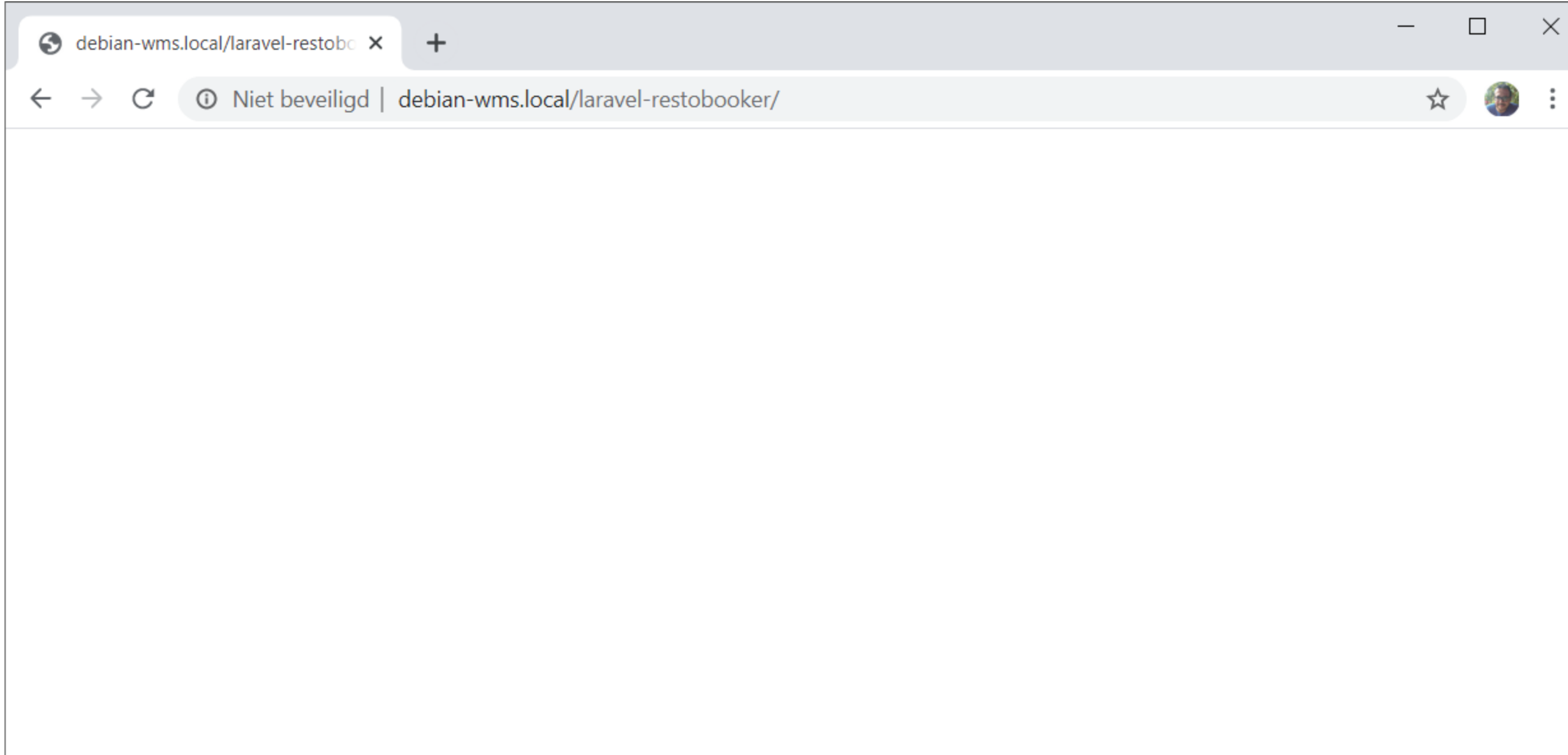
A column was not found

Adding \$timestamps = false property to our Model classes

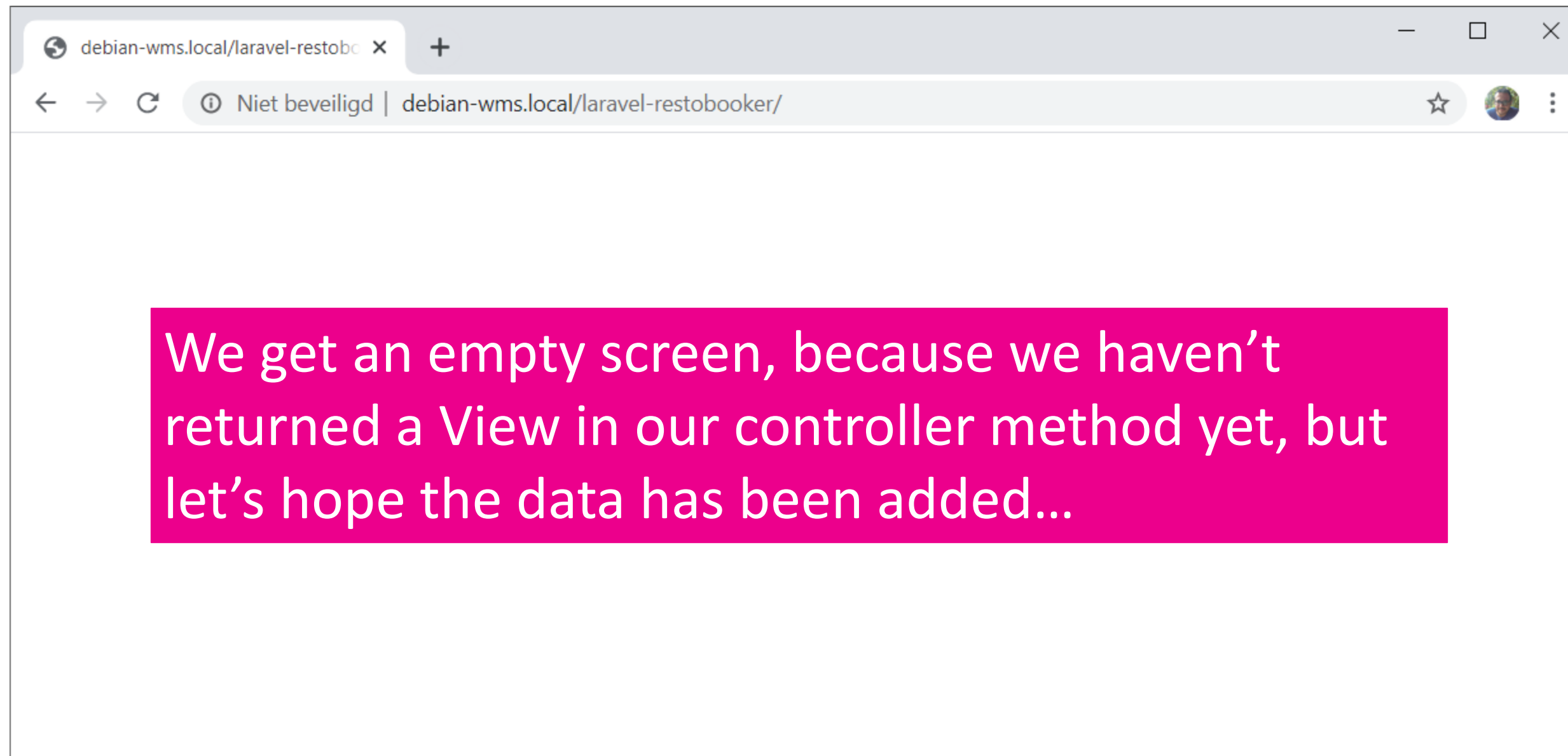
```
class Restaurant extends Model
{
    public $timestamps = false;
}
```

```
class Booking extends Model
{
    public $timestamps = false;
}
```

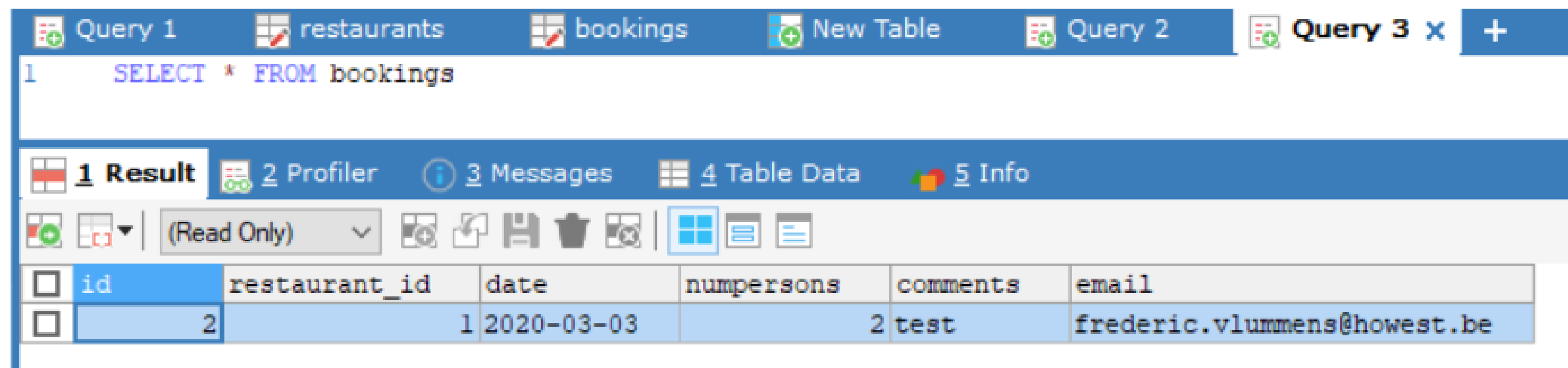
Trying again



Trying again



In the database



The screenshot shows a database management interface with a query editor at the top containing the SQL statement `SELECT * FROM bookings`. Below the editor, a toolbar indicates the query was executed successfully. The main area displays a table with the following data:

<input type="checkbox"/>	id	restaurant_id	date	numpersons	comments	email
<input type="checkbox"/>	2	1	2020-03-03	2	test	frederic.vlummens@howest.be

Success!

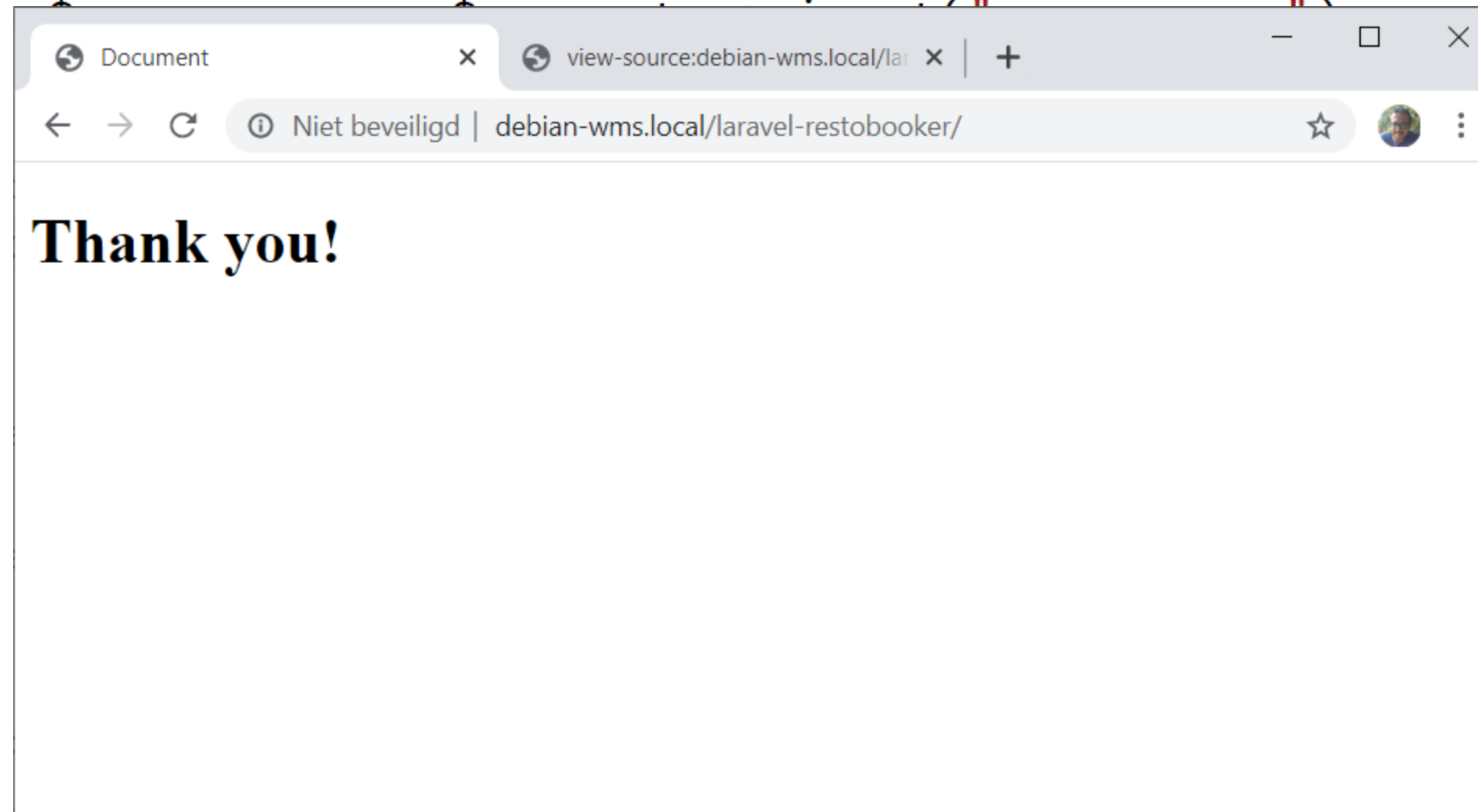
Adding a confirmation message

```
function addBooking(Request $request) {  
    $restaurant_id = $request → input("restaurant");  
    $numpersons = $request → input("numpersons");  
    $email = $request → input("email");  
    $comments = $request → input("comments");  
    $date = $request → input("date");  
  
    $booking = new \App\Booking();  
  
    $booking → restaurant_id = $restaurant_id;  
    $booking → numpersons = $numpersons;  
    $booking → email = $email;  
    $booking → comments = $comments;  
    $booking → date = $date;  
  
    $booking → save();  
  
    return view("thank-you");  
}
```

← If necessary, you can also pass data to the view in the way we are used to.

Adding a confirmation message

```
function addBooking(Request $request) {  
    $restaurant_id = $request → input("restaurant");
```



```
$booking → save();
```

```
return view("thank-you");
```

```
}
```

← If necessary, you can also pass data to the view in the way we are used to.

Questions?

