

# Web, Mobile and Security

---

## Extra exercise: Ticket Booker

### About this exercise

In this exercise, we will be building a web application that allows users to book tickets for concerts.

### Required outcome

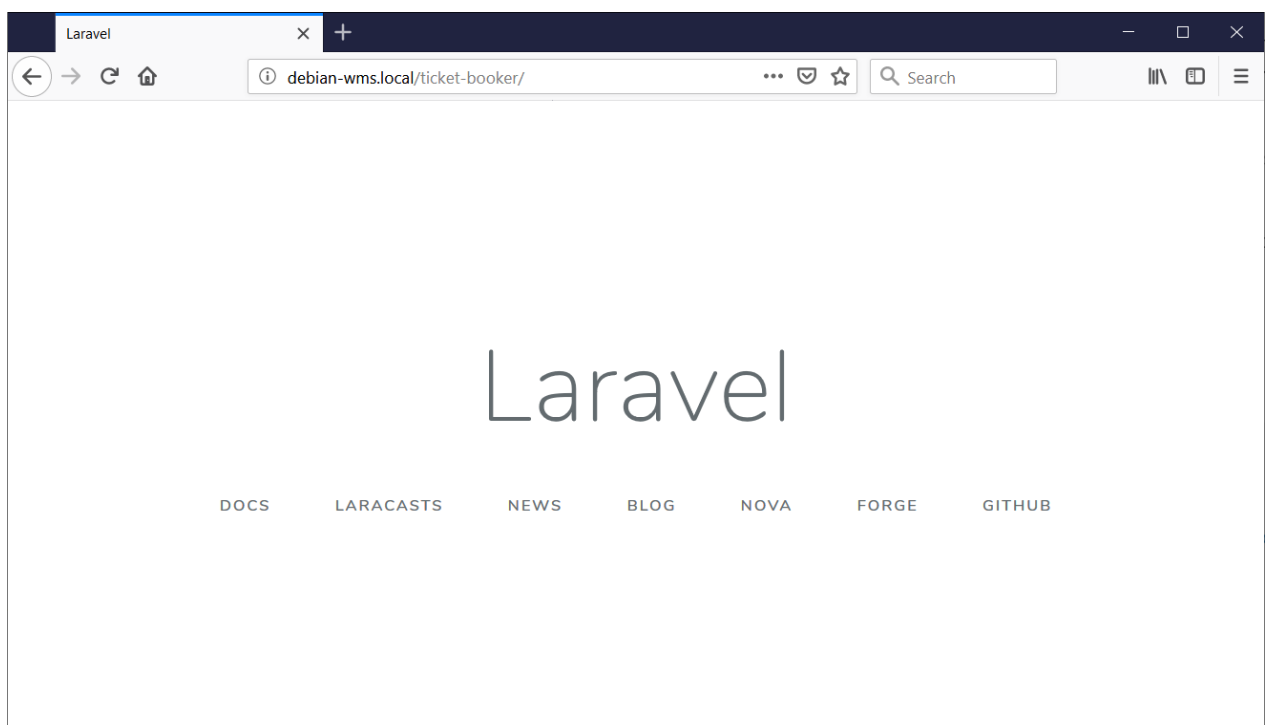
- When the user launches the website, a list of concerts is displayed.
- When clicking a concert, its details are displayed, and a link is displayed to book the concert.
- When clicking the link, the user is taken to a new page where (s)he can fill in personal details such as lastname, firstname, e-mail address and number of tickets desired.
- Upon completion of that form, the user receives a confirmation message in the browser.
- An admin page also exists, where an overview of all requested tickets can be displayed.

### Steps

1. Create a new app using the following command in the VM (PuTTY):

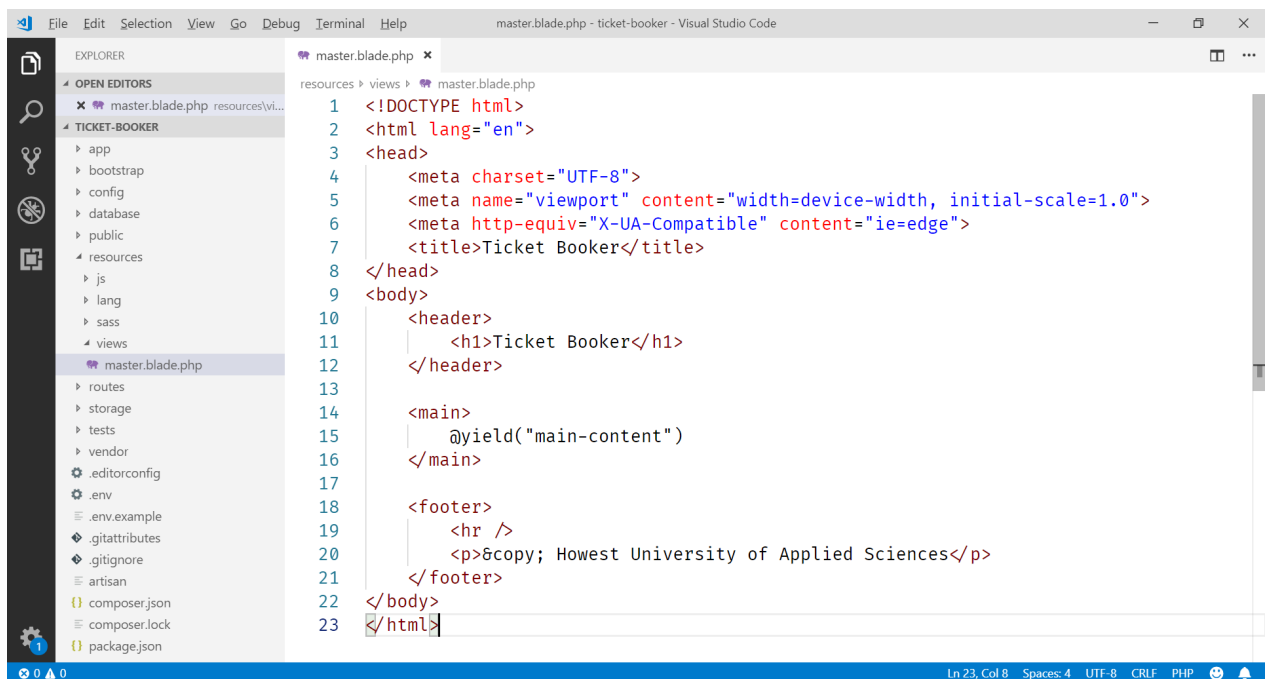
```
./build-laravel-app ticket-booker
```

2. Try out the new app by surfing to <http://debian-wms.local/ticket-booker>:



3. Next, we will determine how many views we will need:

1. List of all concerts
  2. Concert details
  3. Registration form
  4. Confirmation message
  5. Admin page containing all registrations
4. Each view will be accessed using a route, so we need to determine those as well. Per case, we ask ourselves which HTTP verb to use: GET/POST/...
1. List of all concerts → GET (we request a resource)
  2. Concert details → GET (we request a resource)
  3. Registration form → GET (we request a resource)
  4. Processing the registration and showing the confirmation message → POST (we submit/create a resource)
  5. Admin page → GET (we request a resource)
5. We will be using a master page, containing the basic layout of the site, referencing CSS if any, ... Only what is to differ per view/situation, will be defined in each individual child view.
6. Save a file called **master.blade.php** in your folder **/resources/views**. It will contain the following code:

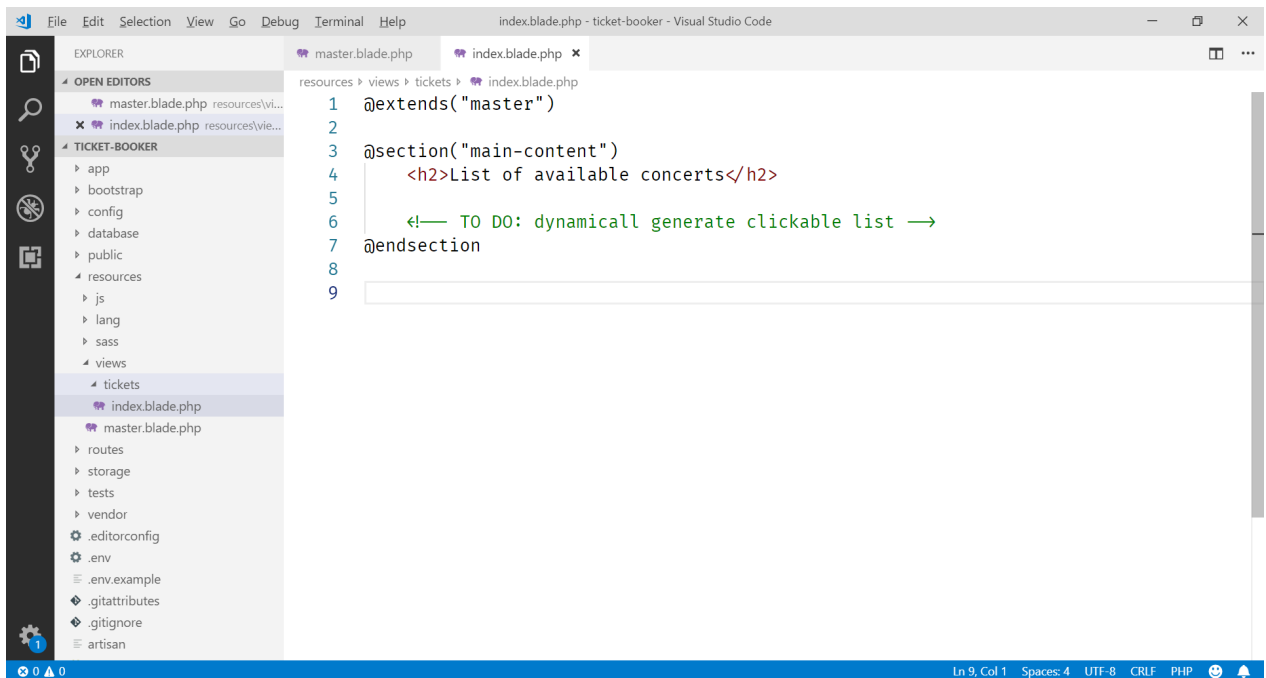


```

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>Ticket Booker</title>
8  </head>
9  <body>
10     <header>
11         <h1>Ticket Booker</h1>
12     </header>
13
14     <main>
15         @yield("main-content")
16     </main>
17
18     <footer>
19         <hr />
20         <p>©copy; Howest University of Applied Sciences</p>
21     </footer>
22 </body>
23 </html>

```

7. As you can see in the screen above, **@yield** defines a 'block' that will be filled in per individual child view.
8. Next, we define our first child view, the one displaying the list of available concerts.
9. Create a file called **index.blade.php** in a subfolder **/resources/views/tickets**. It will contain the following code:



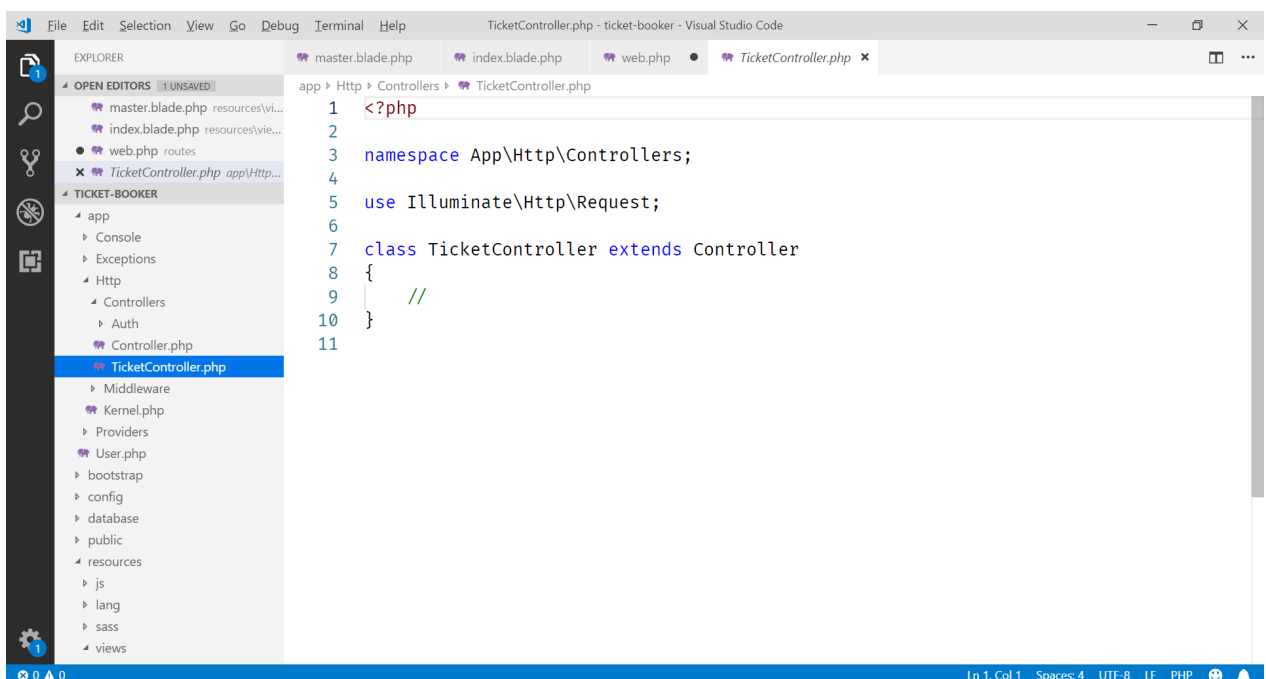
10. As we see, it inherits (**extends**) from the master view. Within the **@section-@endsection** block, we define what is to be displayed instead of the **@yield** instruction.
11. Time to test our first view. Of course, we will need an associated route. But in a Model-View-Controller application, handling the views is done by the controller;
12. So let's generate our **TicketController** by issuing the following commands within PuTTY:

```

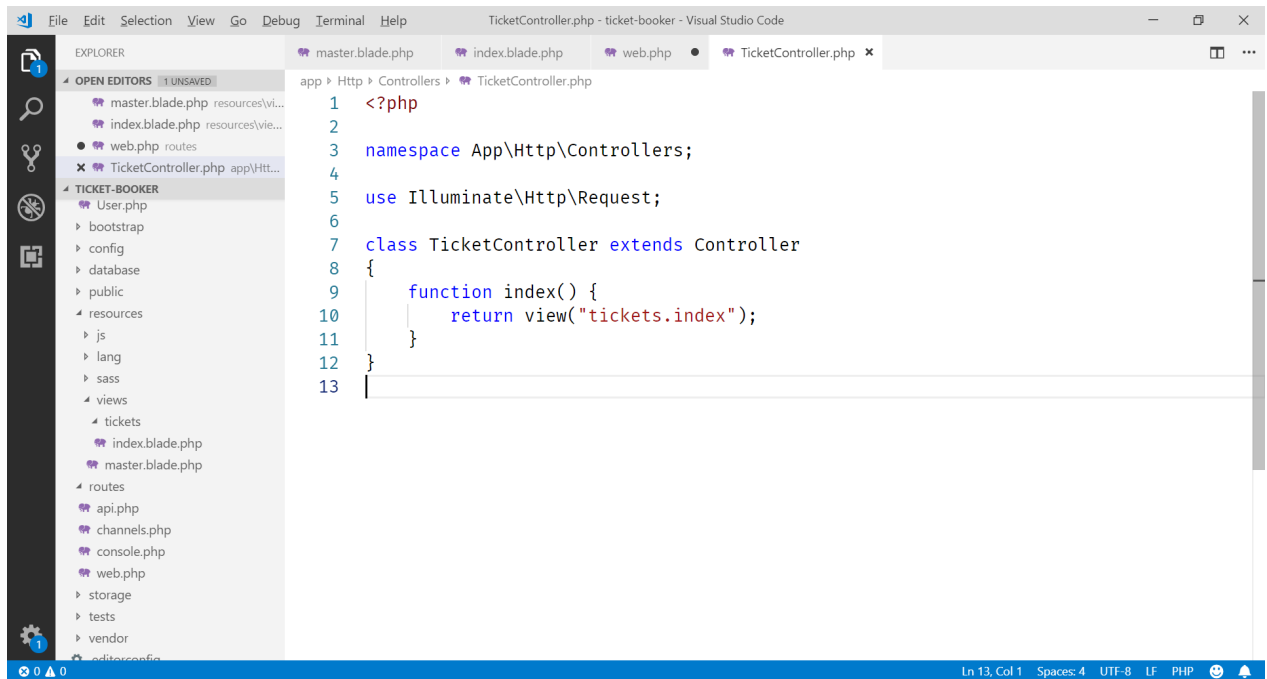
user@debian-wms:~$ cd code/ticket-booker
user@debian-wms:~/code/ticket-booker$ php artisan make:controller
TicketController

```

13. This will generate a file called **TicketController.php** in the folder **/app/Http/Controllers/**:

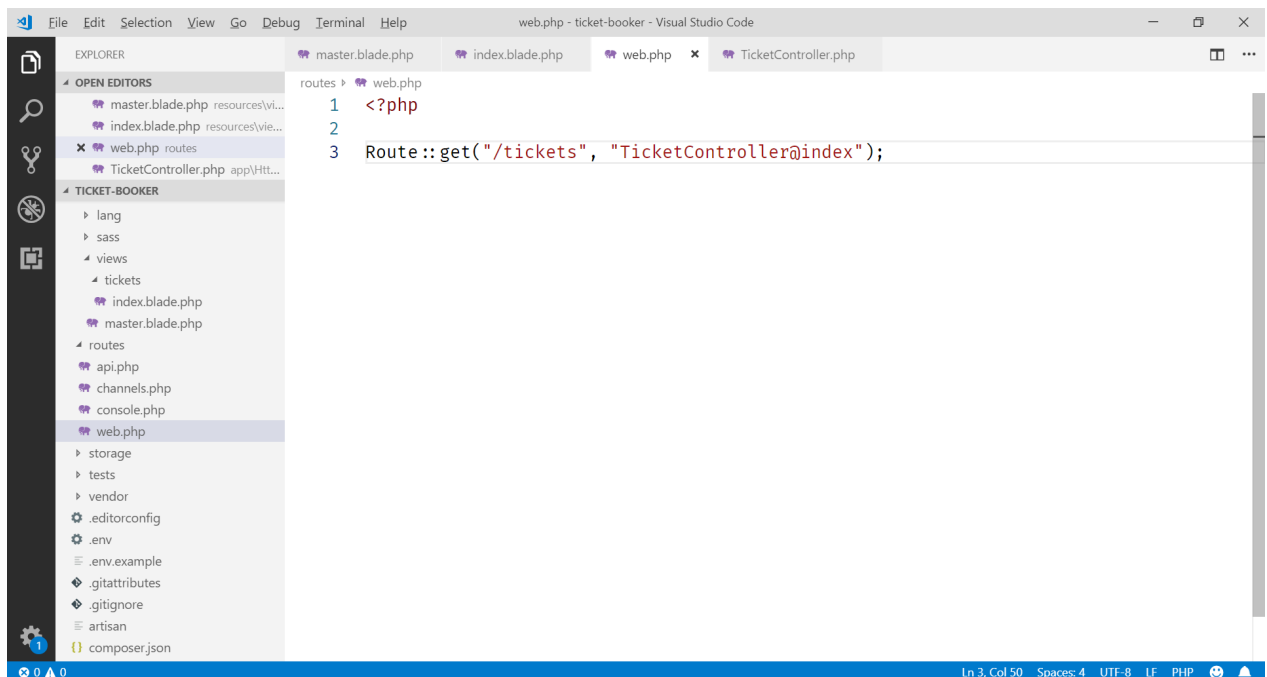


14. Our first controller method will be responsible for displaying the view with concerts. Since the view `index.blade.php` is in a subfolder `tickets`, we use the notation `tickets.index` to reference the view:



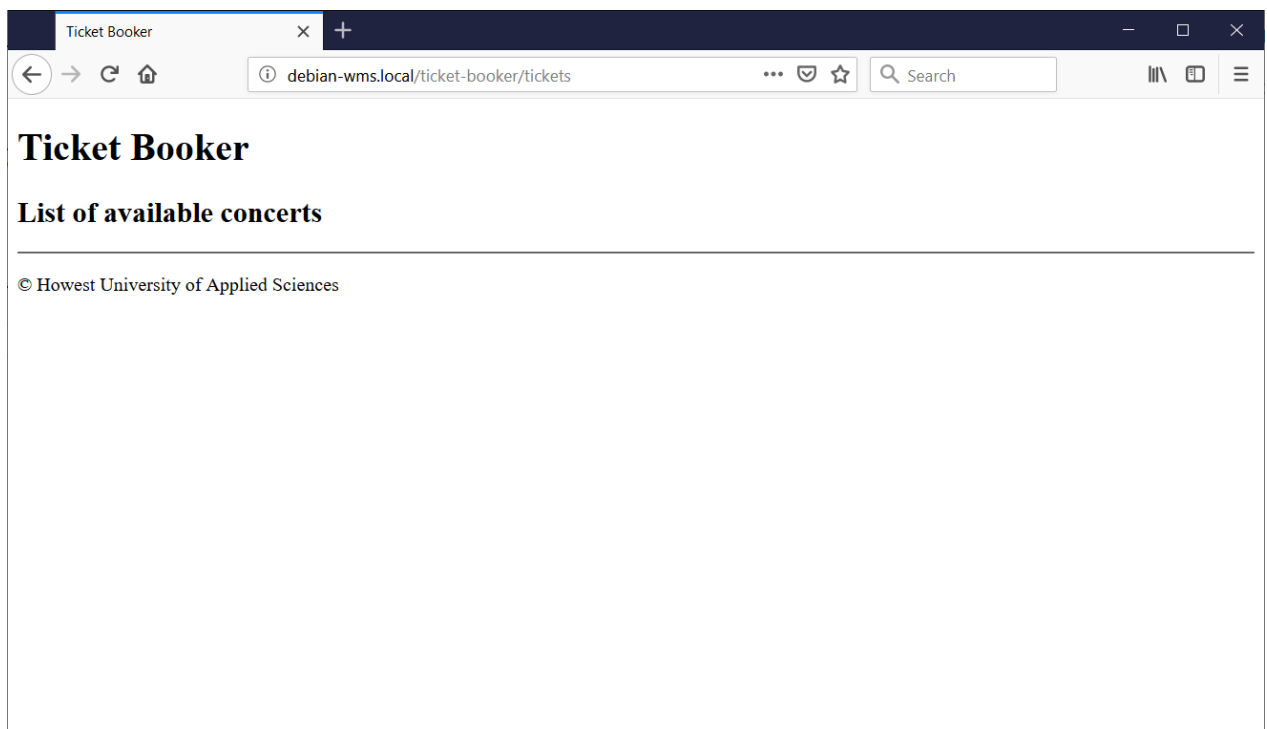
```
1 <?php
2
3 namespace App\Http\Controllers;
4
5 use Illuminate\Http\Request;
6
7 class TicketController extends Controller
8 {
9     function index() {
10         return view("tickets.index");
11     }
12 }
13
```

15. Next, we need to associate a route with this controller method. To do so, go to the file `/routes/web.php` and add the following:

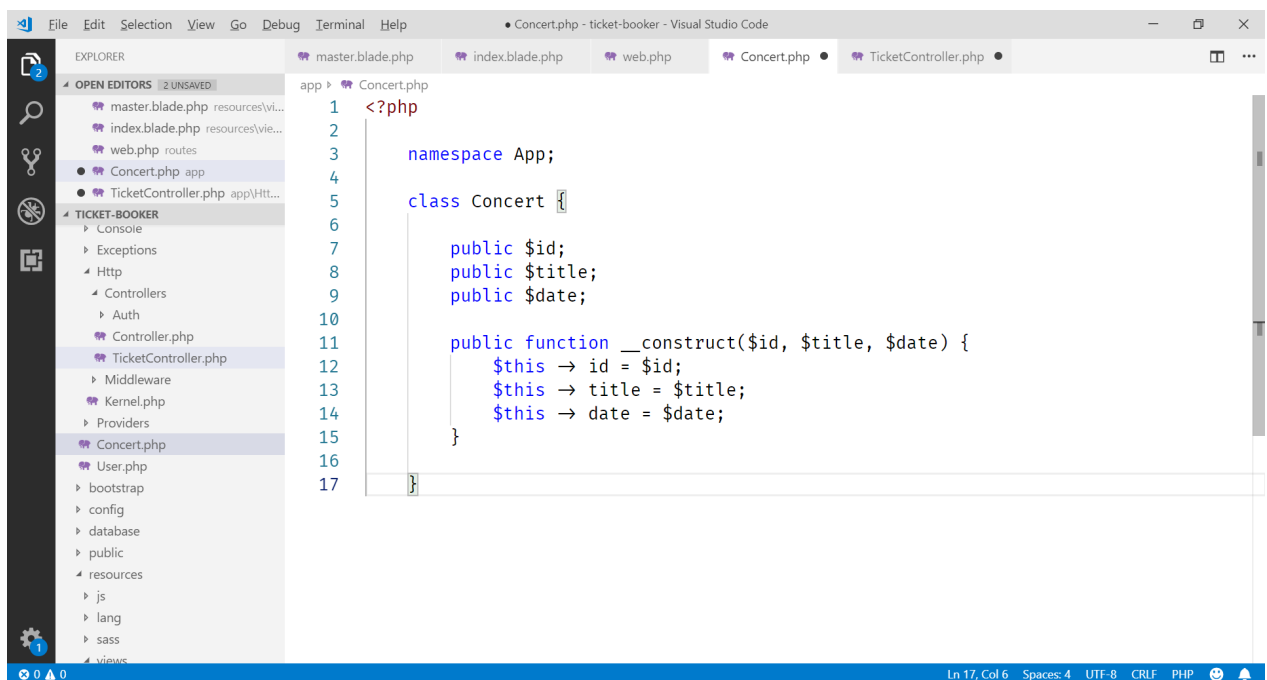


```
1 <?php
2
3 Route::get("/tickets", "TicketController@index");
```

16. Now when we navigate to <http://debian-wms.local/ticket-booker/tickets> we get the child view, inheriting from the master page:



17. Time to add the concerts themselves. In a Model-View-Controller application, data is retrieved from the model.
18. So, we create a model class called **Concert** in the folder **/app**. This class has a constructor and also a few fields:
  - id
  - title
  - date



19. Normally, the data (=list of concerts) would come from the database. Since we haven't seen database access yet, we'll 'hardcode' the concerts in the model.
20. We will build a static method (part of the class itself, instead of an instance of the class) that returns all the concerts:

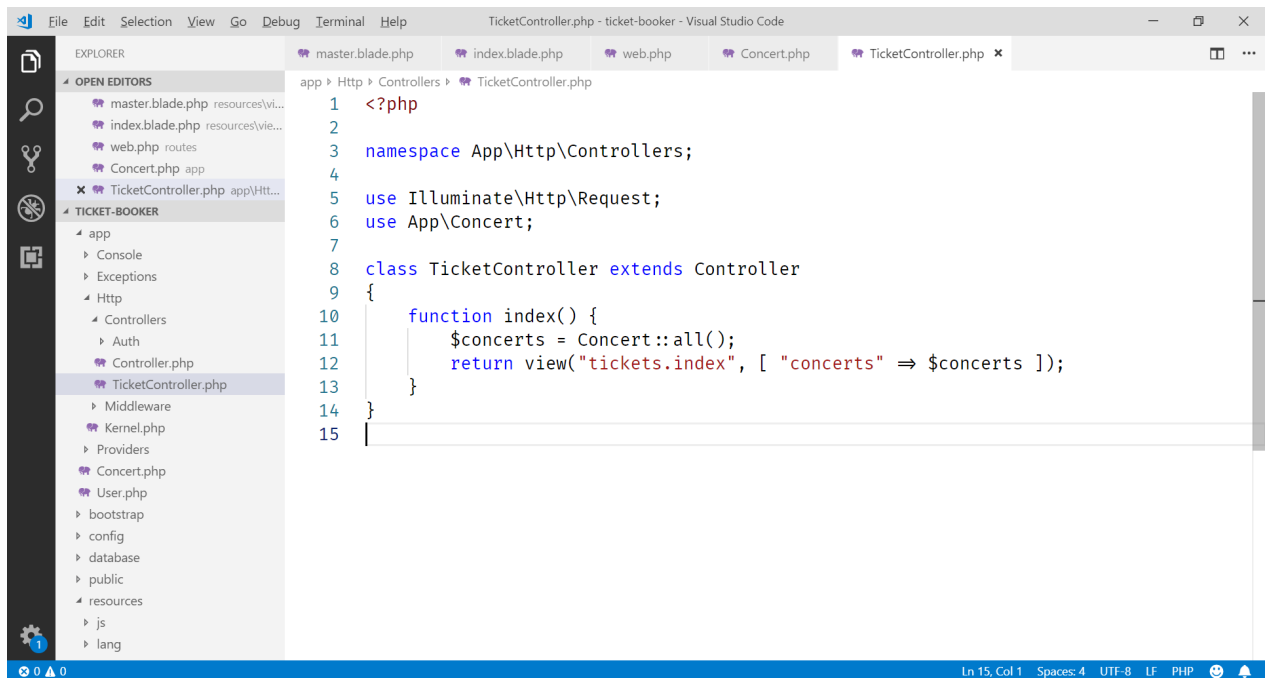
```
4 class Concert {
5     public $id;
6     public $title;
7     public $date;
8
9
10    public function __construct($id, $title, $date) {
11        $this->id = $id;
12        $this->title = $title;
13        $this->date = $date;
14    }
15
16    public static function all() {
17        $concerts = []; // new empty array
18
19        $concerts[101] = new Concert(101, "Rock in Bruges", "30-07-2019");
20        $concerts[102] = new Concert(102, "Fun Beats", "02-04-2019");
21        $concerts[103] = new Concert(103, "Classica", "28-02-2019");
22
23        return $concerts;
24    }
25 }
```

We initialize an empty array, push three elements in it and return it as result of the method **all**.

21. We might also need the details of a concert based on its unique id. So, we write a corresponding method **find**:

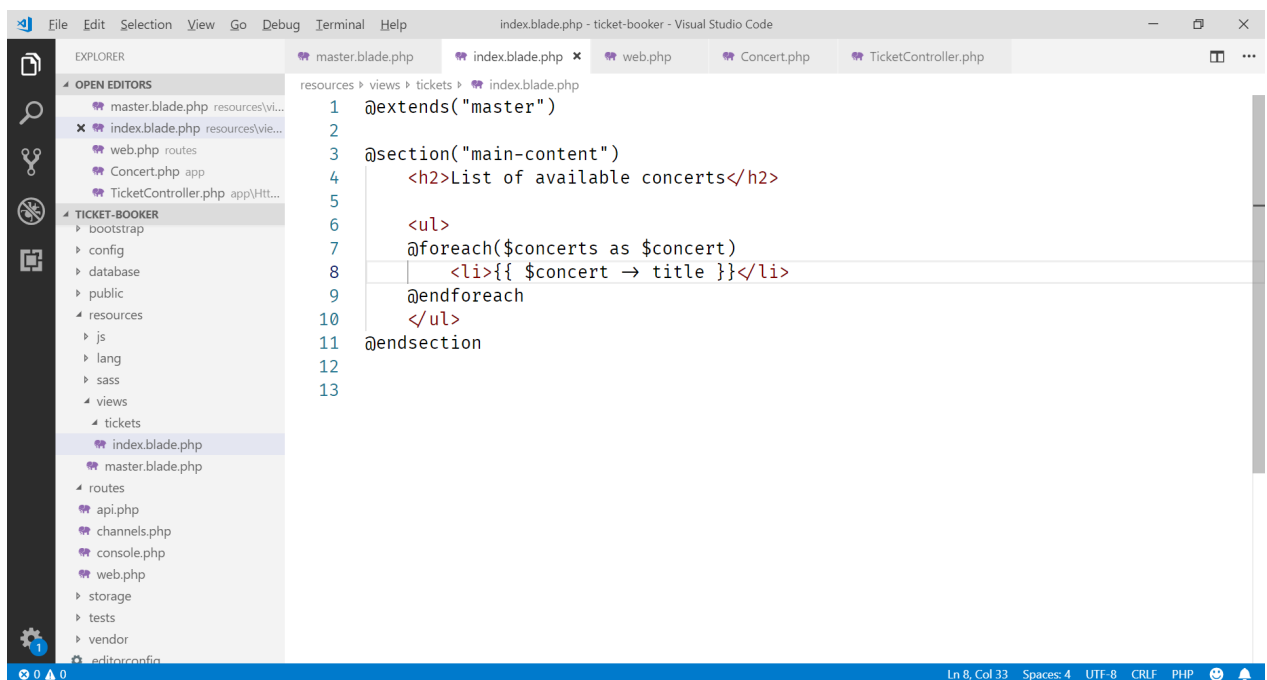
```
17
18 $concerts[101] = new Concert(101, "Rock in Bruges", "30-07-2019");
19 $concerts[102] = new Concert(102, "Fun Beats", "02-04-2019");
20 $concerts[103] = new Concert(103, "Classica", "28-02-2019");
21
22 return $concerts;
23 }
24
25 public static function find($id) {
26     $concerts = Concert::all(); // retrieve all concerts
27
28     // next ,look for the id in the array
29     if (array_key_exists($id, $concerts)) {
30         // found, so return it
31         return $concerts[$id];
32     } else {
33         // not found, return null;
34         return null;
35     }
36 }
37
38 }
```

22. Back to our controller, where we will pass the list of concerts to the view. So we update our method accordingly:



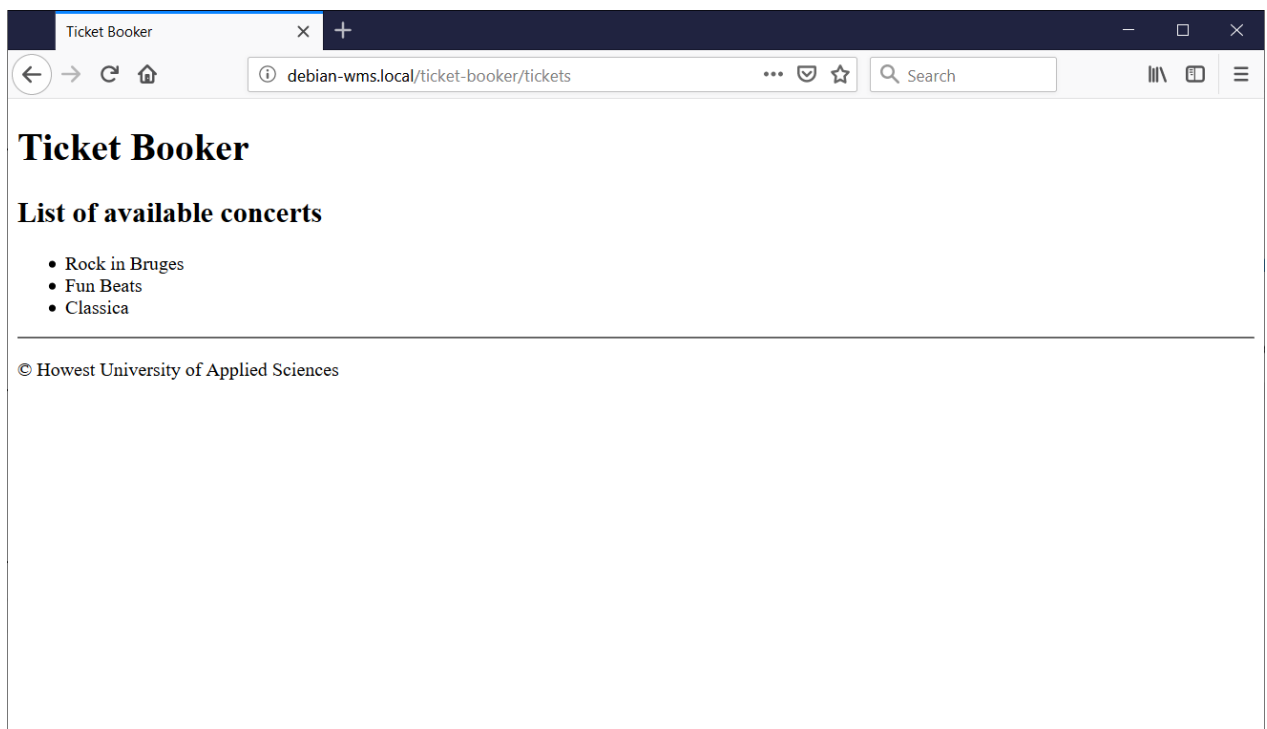
```
1 <?php
2
3 namespace App\Http\Controllers;
4
5 use Illuminate\Http\Request;
6 use App\Concert;
7
8 class TicketController extends Controller
9 {
10     function index() {
11         $concerts = Concert::all();
12         return view("tickets.index", [ "concerts" => $concerts ]);
13     }
14 }
15
```

23. Since we now (using an associative array) pass the concerts in using the key **"concerts"**, we should be able to access this variable from within our view:

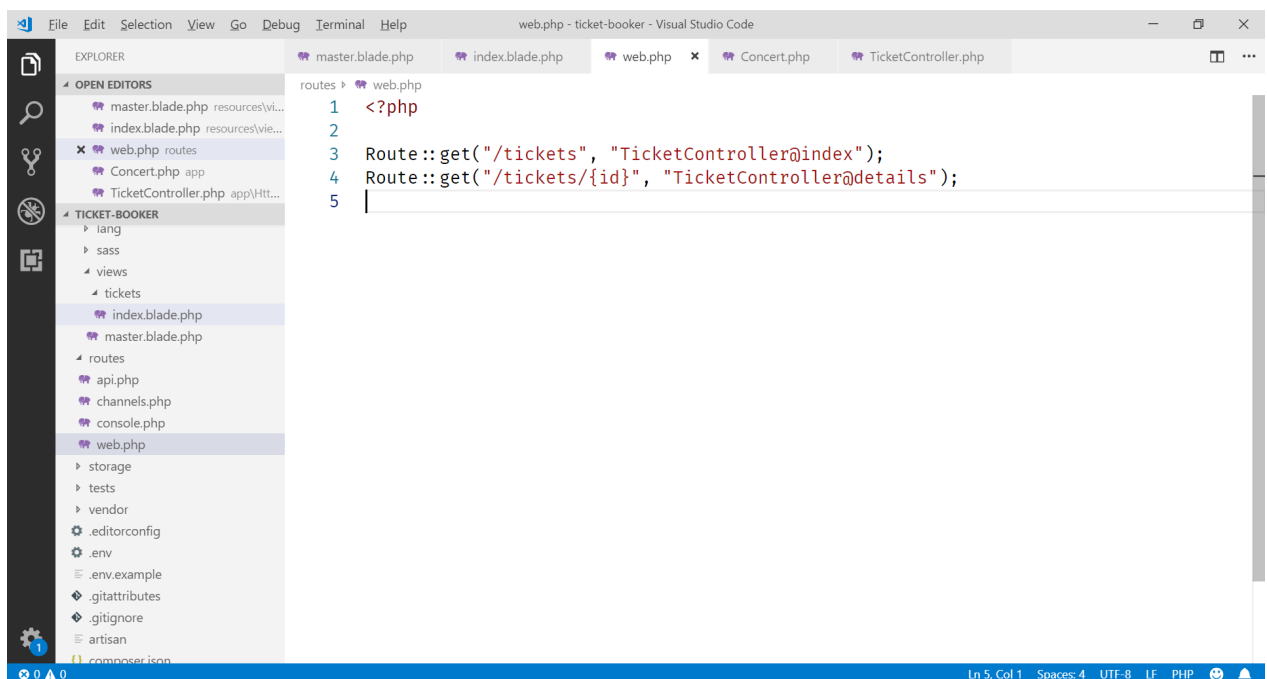


```
1 @extends("master")
2
3 @section("main-content")
4     <h2>List of available concerts</h2>
5
6     <ul>
7         @foreach($concerts as $concert)
8             <li>{{ $concert -> title }}</li>
9         @endforeach
10     </ul>
11 @endsection
12
13
```

24. And when we try this out, we get the desired result:

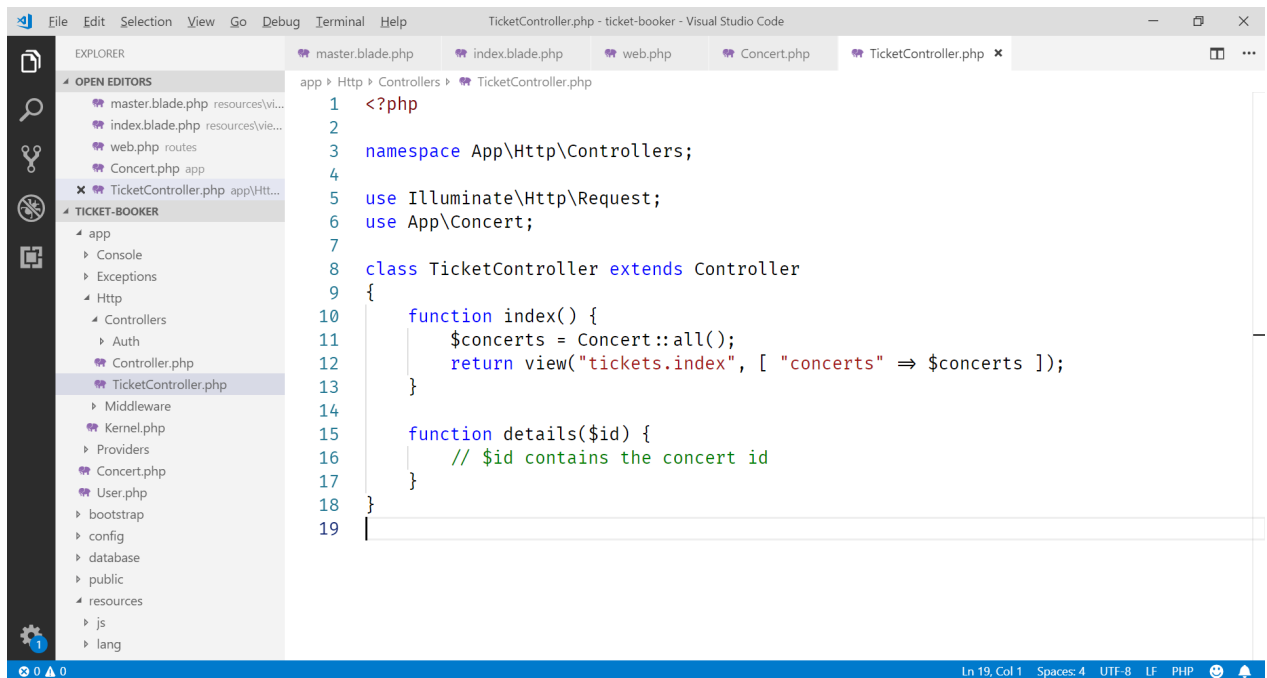


25. Next, we want to make our concerts clickable. And when clicking a concert, we want to display the concert's details.
26. As we discussed earlier, this involves a GET route. With the hyperlink, we will pass the concert id in the URL, for example:
- <http://debian-wms.local/ticket-booker/tickets/101> for concert 101 (Rock in Bruges)
  - <http://debian-wms.local/ticket-booker/tickets/103> for concert 103 (Classica)
  - ...
27. So, in our routes, we define a new route, with a required parameter:



28. Of course, our controller now needs a method `details` with a parameter in which `{id}` will be passed:

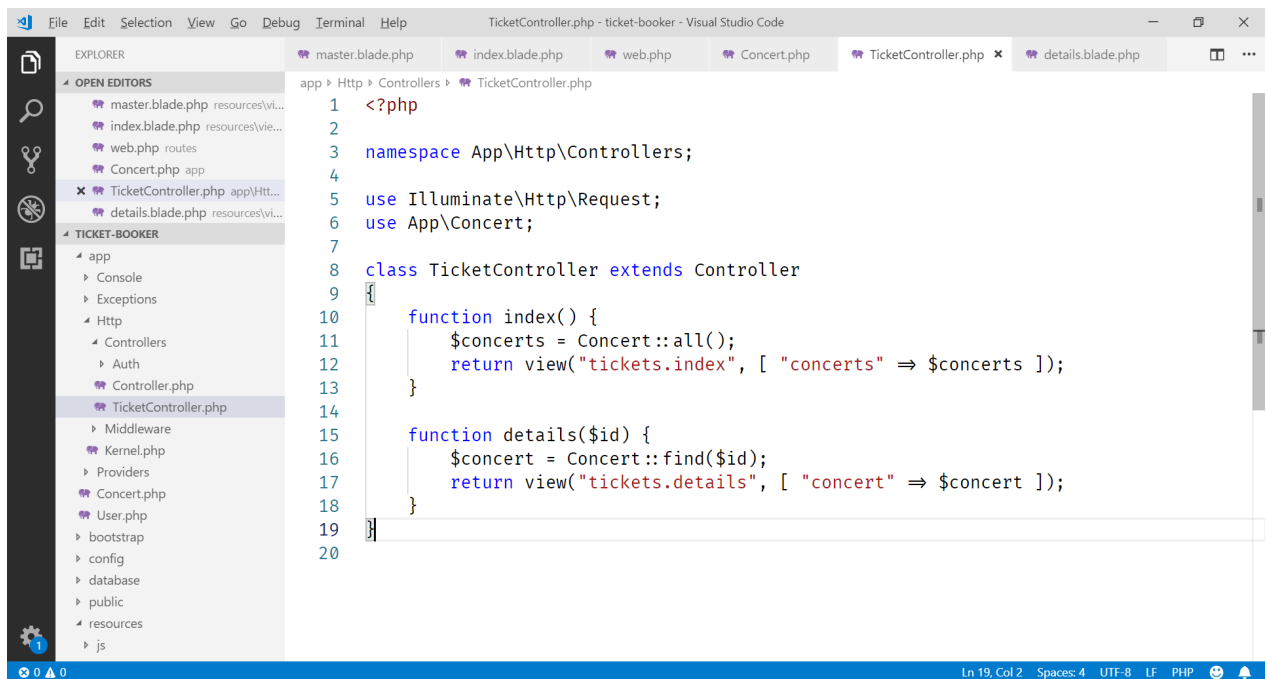




```
1 <?php
2
3 namespace App\Http\Controllers;
4
5 use Illuminate\Http\Request;
6 use App\Concert;
7
8 class TicketController extends Controller
9 {
10     function index() {
11         $concerts = Concert::all();
12         return view("tickets.index", [ "concerts" => $concerts ]);
13     }
14
15     function details($id) {
16         // $id contains the concert id
17     }
18 }
19
```

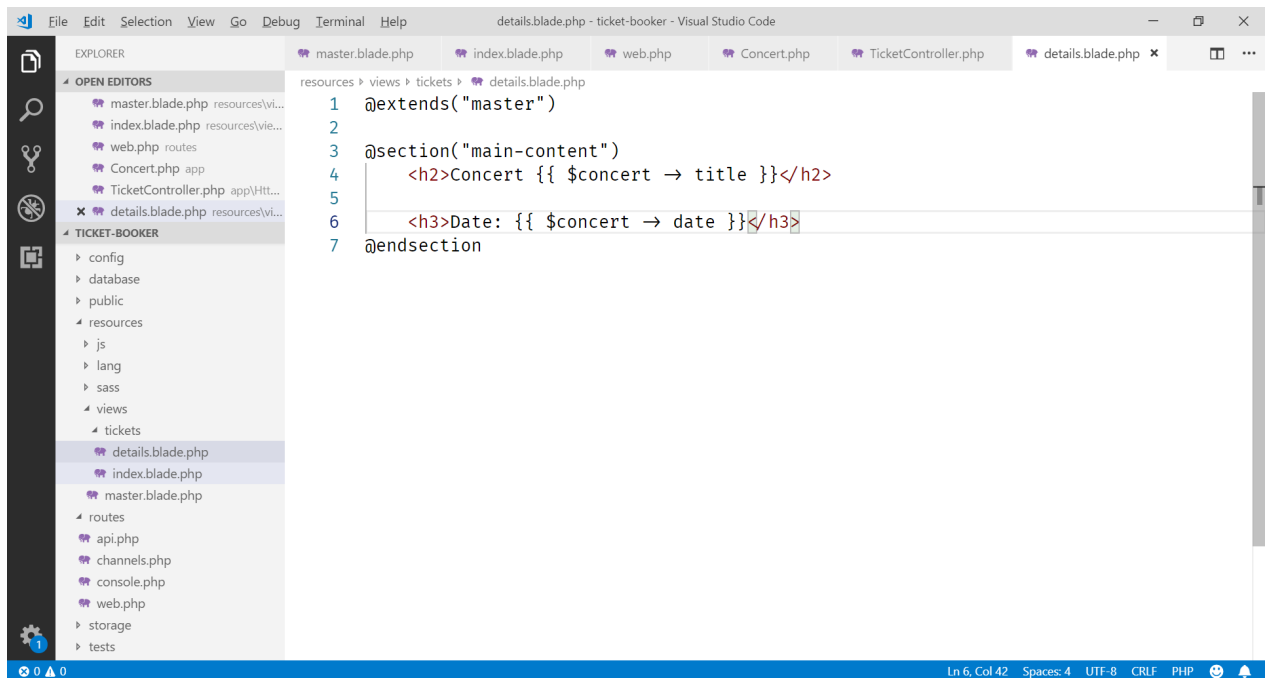
29. We will need a view that can display the concert details, and we'll pass the concert object to it.

30. So first, we need to retrieve the concert object based on its id. Luckily, we can rely on our method **find** we wrote earlier:



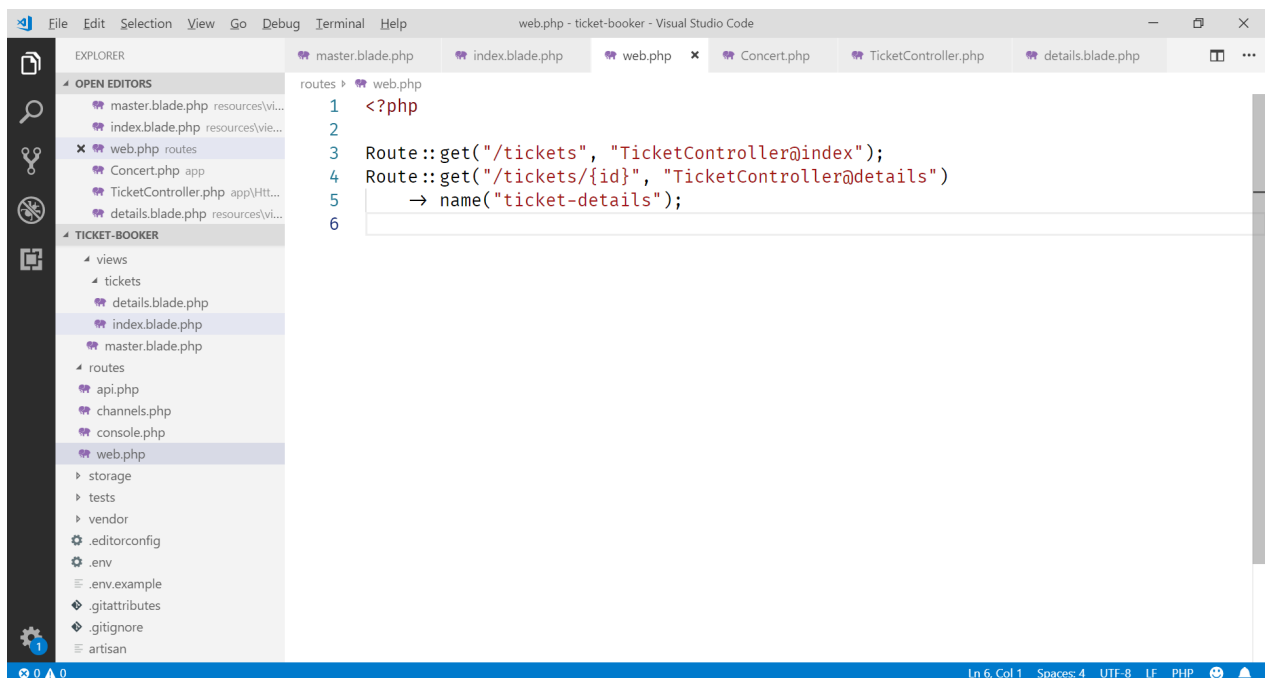
```
1 <?php
2
3 namespace App\Http\Controllers;
4
5 use Illuminate\Http\Request;
6 use App\Concert;
7
8 class TicketController extends Controller
9 {
10     function index() {
11         $concerts = Concert::all();
12         return view("tickets.index", [ "concerts" => $concerts ]);
13     }
14
15     function details($id) {
16         $concert = Concert::find($id);
17         return view("tickets.details", [ "concert" => $concert ]);
18     }
19 }
20
```

31. And now we can write the view itself, where we write out the concert's details using curly braces:



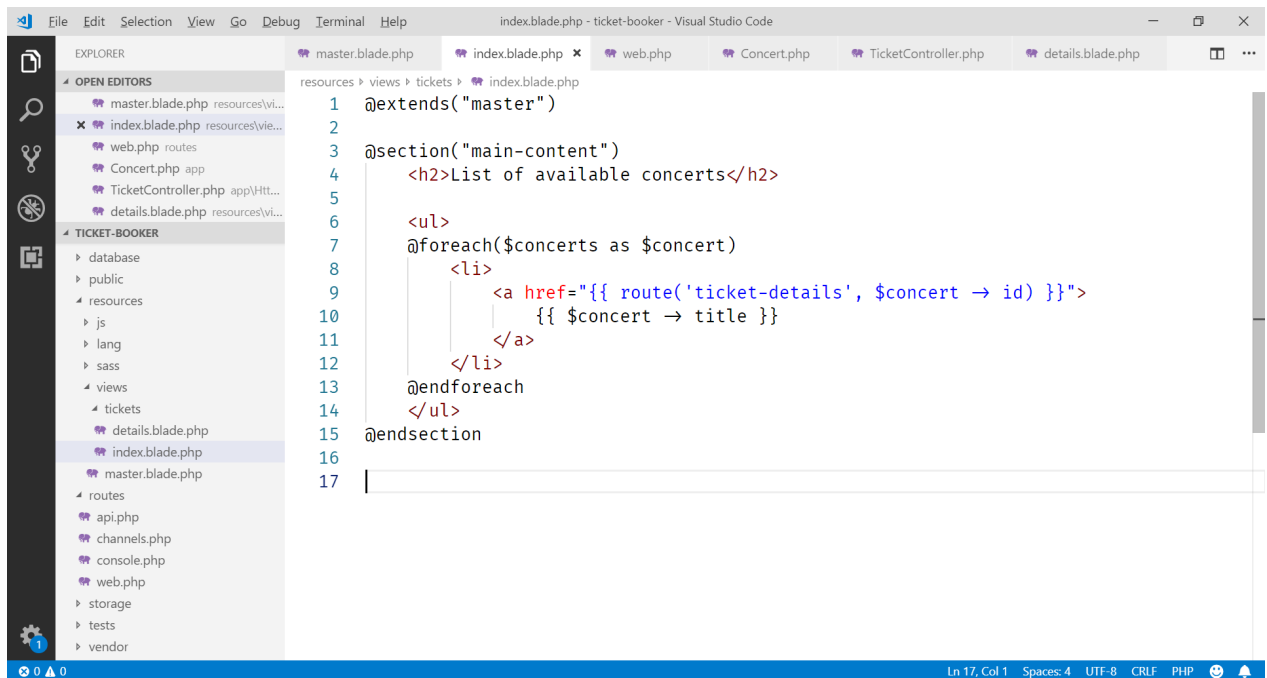
```
1 @extends("master")
2
3 @section("main-content")
4     <h2>Concert {{ $concert → title }}</h2>
5
6     <h3>Date: {{ $concert → date }}</h3>
7 @endsection
```

32. One piece is missing: we need to make sure our list of concerts in `index.blade.php` is clickable and leads to the correct route with the parameter `{id}`.
33. To generate the correct route/link, we can rely on the `route` function. For this function to work, we need to give our route a name in `web.php`:



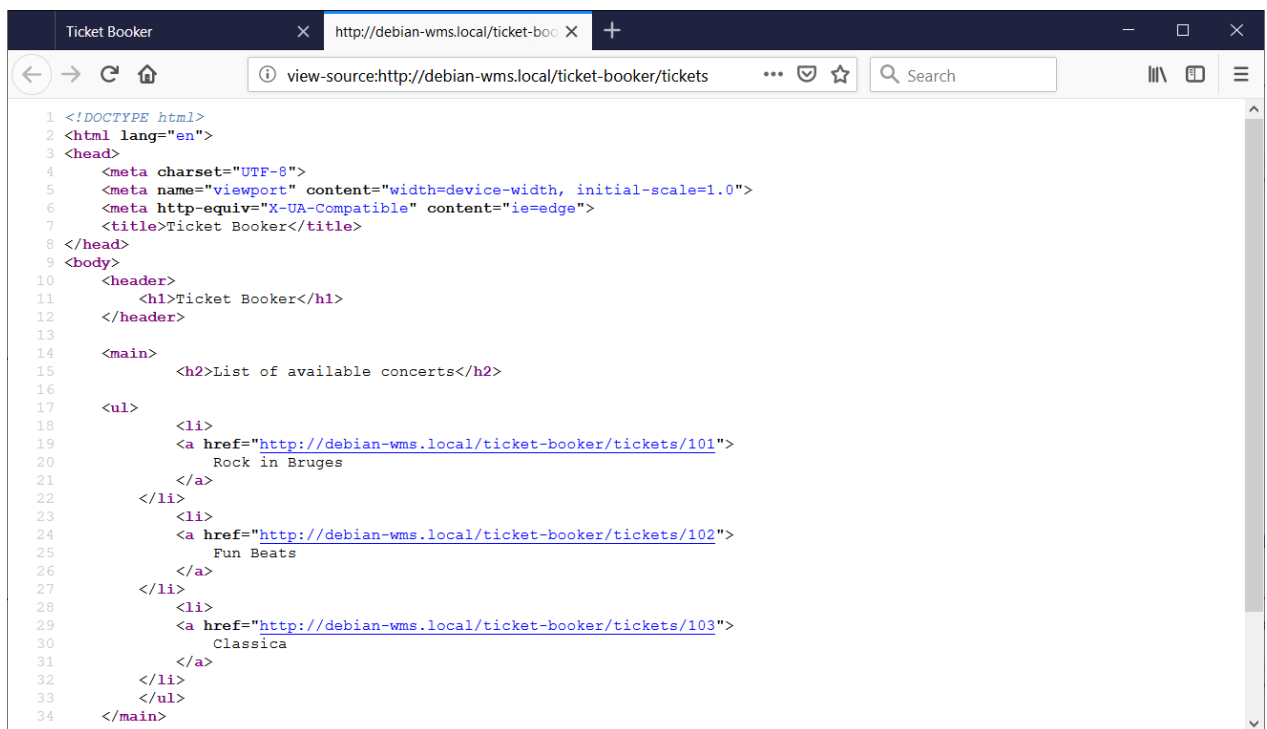
```
1 <?php
2
3 Route::get("/tickets", "TicketController@index");
4 Route::get("/tickets/{id}", "TicketController@details")
5     → name("ticket-details");
6
```

34. Because our route has a unique name, we can now pass it to the `route` function. The second argument of the function call will contain the id of the concert:



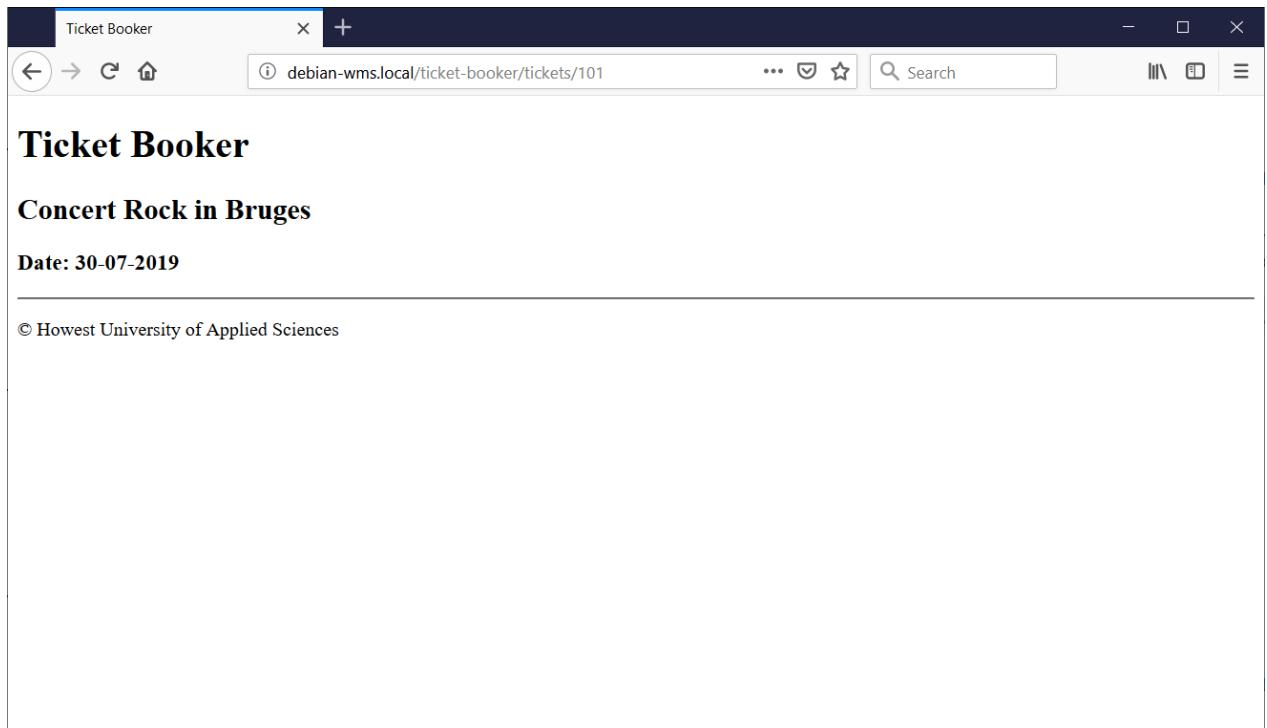
```
1 @extends("master")
2
3 @section("main-content")
4     <h2>List of available concerts</h2>
5
6     <ul>
7         @foreach($concerts as $concert)
8             <li>
9                 <a href="{{ route('ticket-details', $concert -> id) }}">
10                     {{ $concert -> title }}
11                 </a>
12             </li>
13         @endforeach
14     </ul>
15 @endsection
16
17
```

35. The `route` function will append the id to the URL, which we can clearly see if we go into the source of the generated HTML:

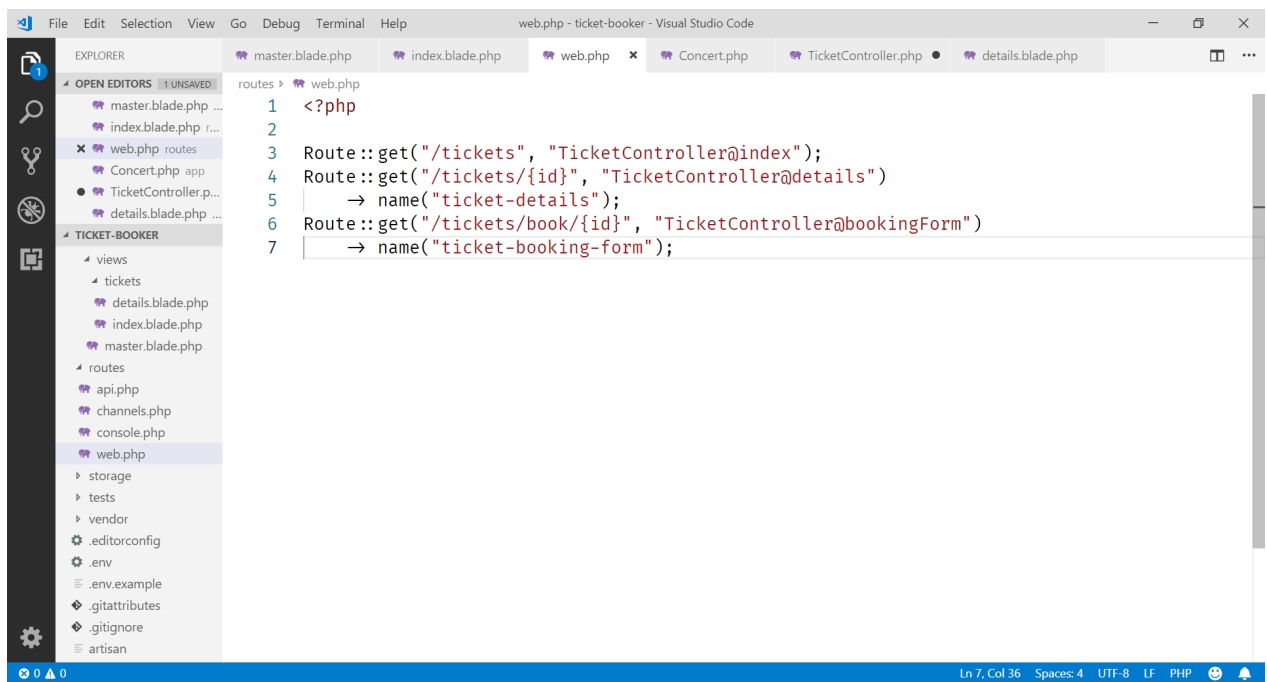


```
1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4     <meta charset="UTF-8">
5     <meta name="viewport" content="width=device-width, initial-scale=1.0">
6     <meta http-equiv="X-UA-Compatible" content="ie=edge">
7     <title>Ticket Booker</title>
8 </head>
9 <body>
10     <header>
11         <h1>Ticket Booker</h1>
12     </header>
13
14     <main>
15         <h2>List of available concerts</h2>
16
17         <ul>
18             <li>
19                 <a href="http://debian-wms.local/ticket-booker/tickets/101">
20                     Rock in Bruges
21                 </a>
22             </li>
23             <li>
24                 <a href="http://debian-wms.local/ticket-booker/tickets/102">
25                     Fun Beats
26                 </a>
27             </li>
28             <li>
29                 <a href="http://debian-wms.local/ticket-booker/tickets/103">
30                     Classica
31                 </a>
32             </li>
33         </ul>
34     </main>
35
```

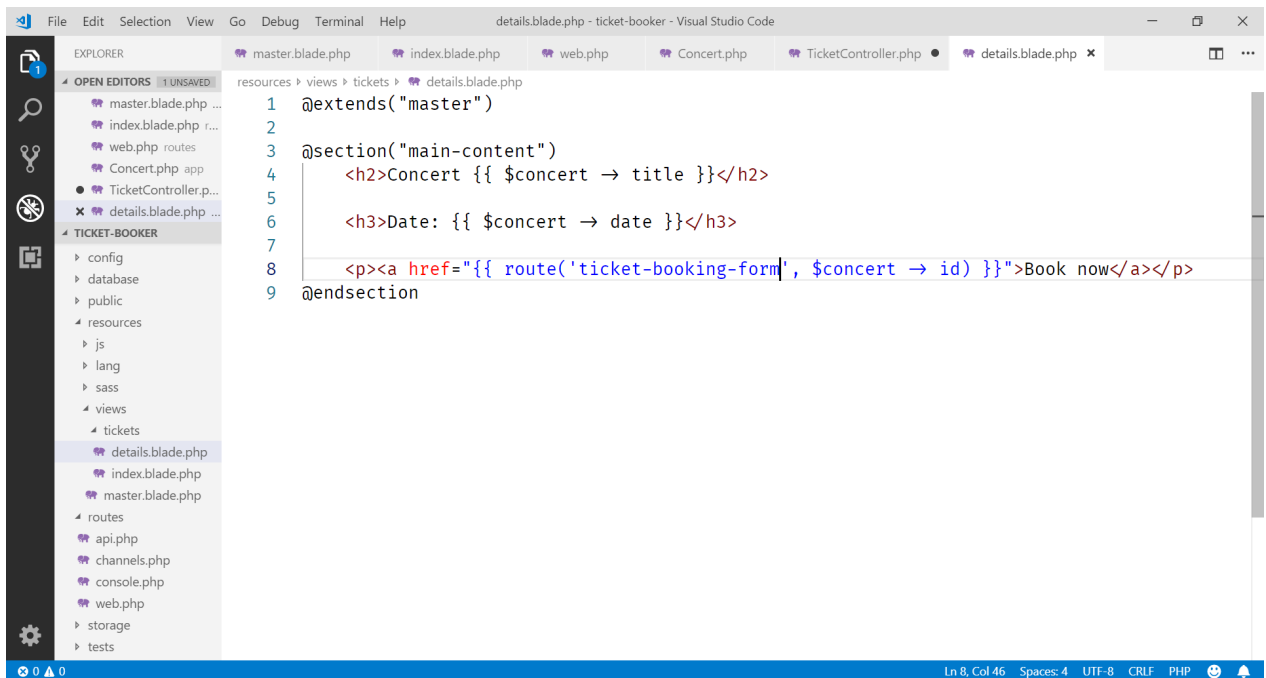
36. All pieces now come together, and when we click a concert, we get the details view:



37. Next, we want a link to a booking page. So, we need a new named route:

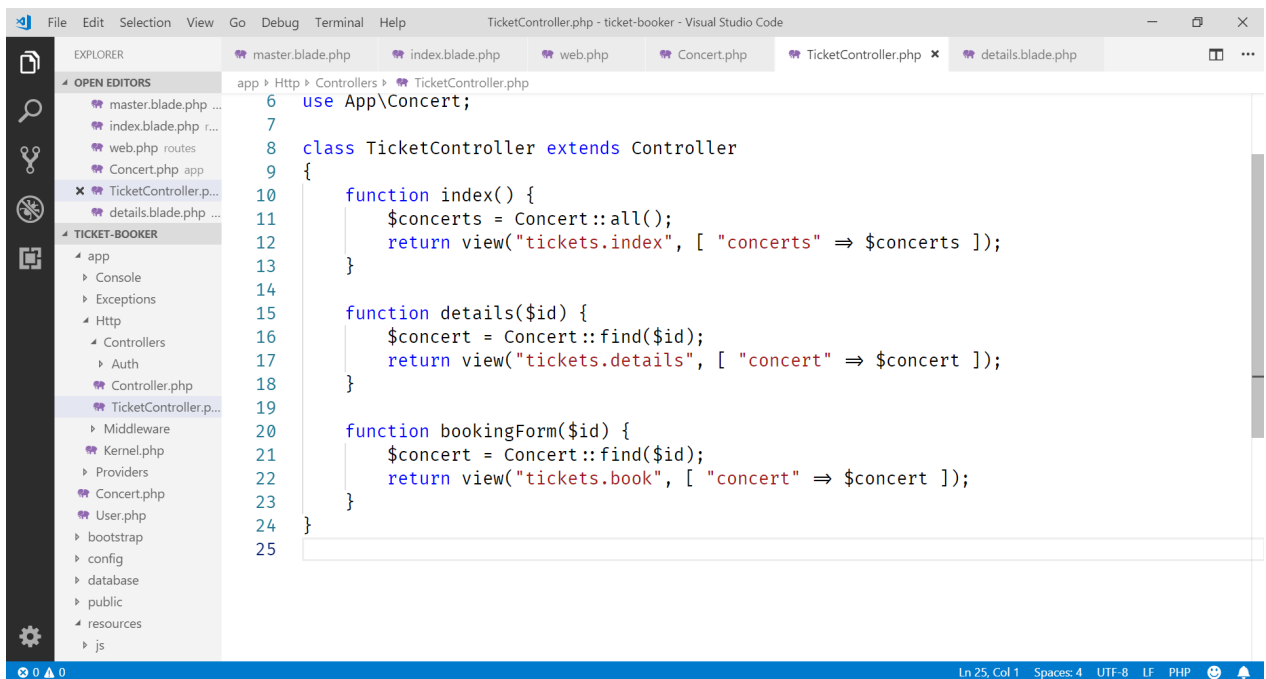


38. This route name can now be used in combination with the `route` function in our view, to build the actual link:



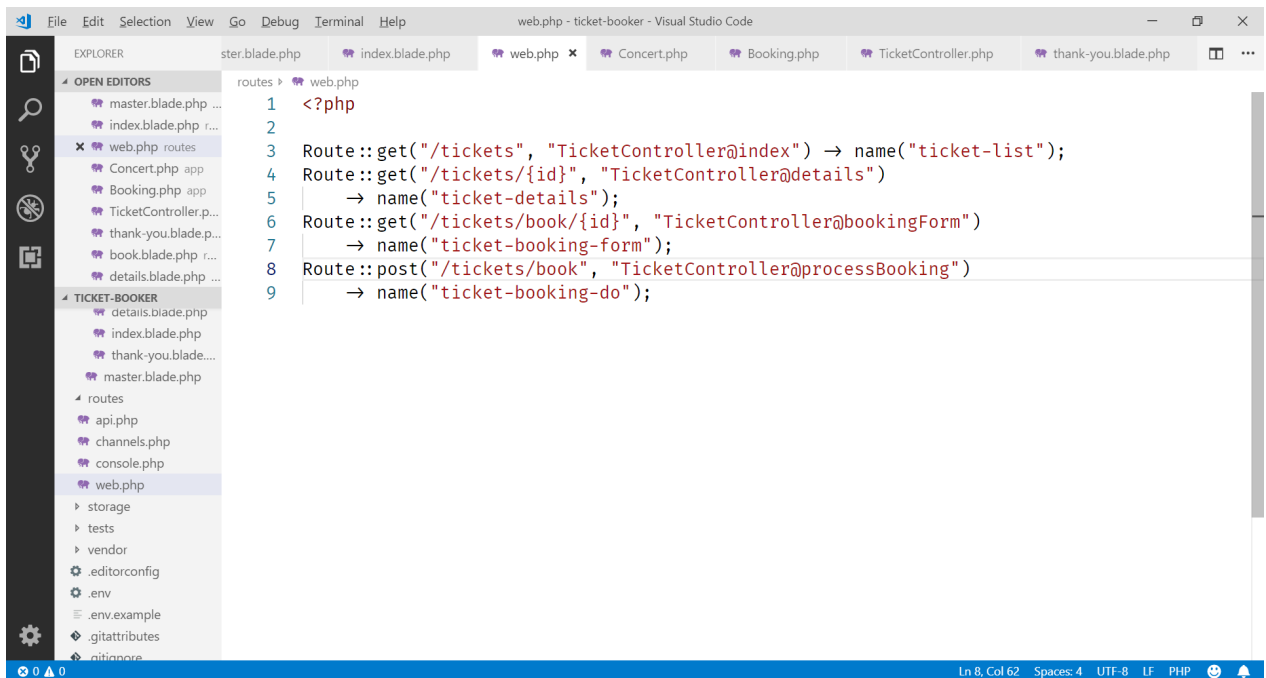
```
1 @extends("master")
2
3 @section("main-content")
4     <h2>Concert {{ $concert → title }}</h2>
5
6     <h3>Date: {{ $concert → date }}</h3>
7
8     <p><a href="{{ route('ticket-booking-form', $concert → id) }}">Book now</a></p>
9 @endsection
```

39. In our controller, we now of course need to write the corresponding method:



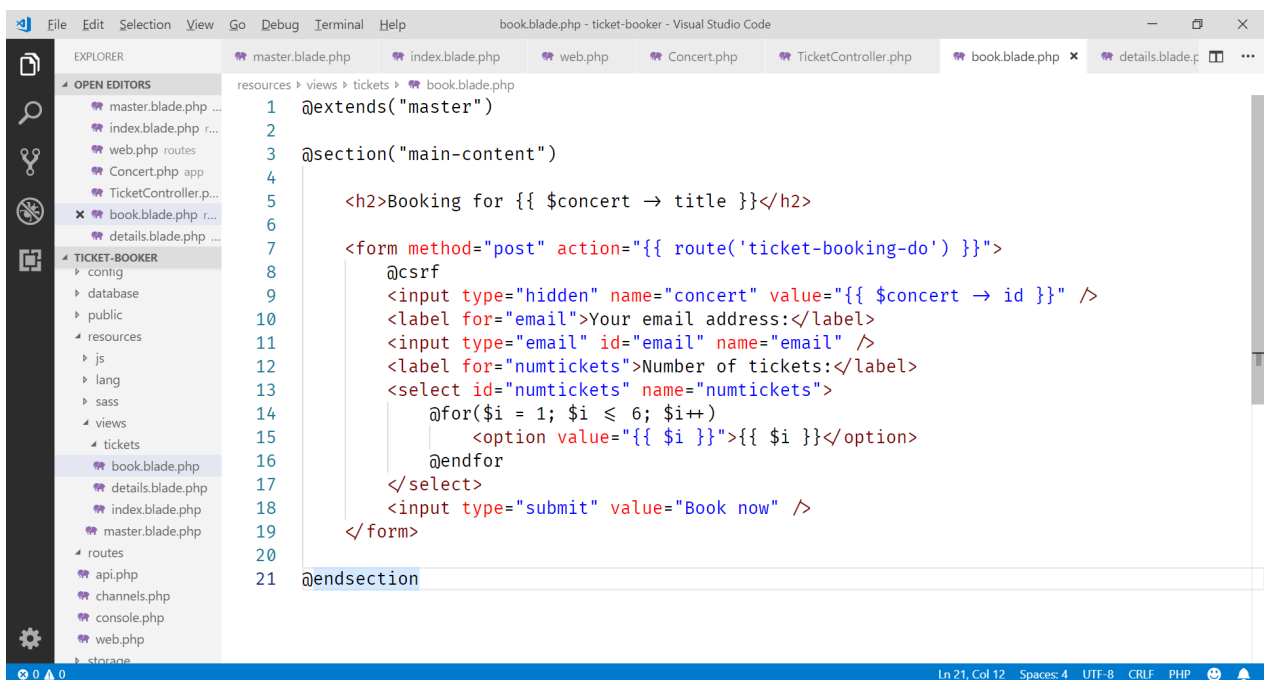
```
6 use App\Concert;
7
8 class TicketController extends Controller
9 {
10     function index() {
11         $concerts = Concert::all();
12         return view("tickets.index", [ "concerts" => $concerts ]);
13     }
14
15     function details($id) {
16         $concert = Concert::find($id);
17         return view("tickets.details", [ "concert" => $concert ]);
18     }
19
20     function bookingForm($id) {
21         $concert = Concert::find($id);
22         return view("tickets.book", [ "concert" => $concert ]);
23     }
24 }
25
```

40. And we develop the corresponding view, which will display the booking form. Of course we need a route to process the POST submission of the booking, so we define that in web.php:



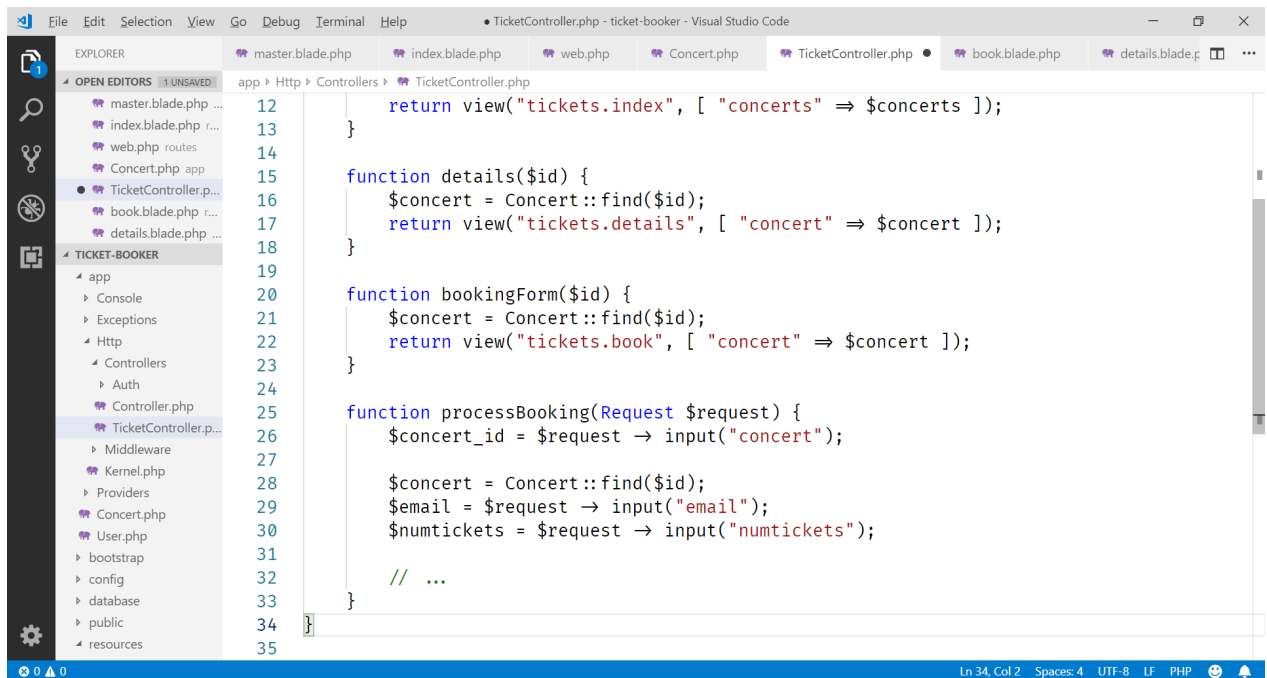
```
1 <?php
2
3 Route::get("/tickets", "TicketController@index") → name("ticket-list");
4 Route::get("/tickets/{id}", "TicketController@details")
5     → name("ticket-details");
6 Route::get("/tickets/book/{id}", "TicketController@bookingForm")
7     → name("ticket-booking-form");
8 Route::post("/tickets/book", "TicketController@processBooking")
9     → name("ticket-booking-do");
```

41. And then we develop the actual view. Note the usage of a hidden input type to submit the concert ID via POST:



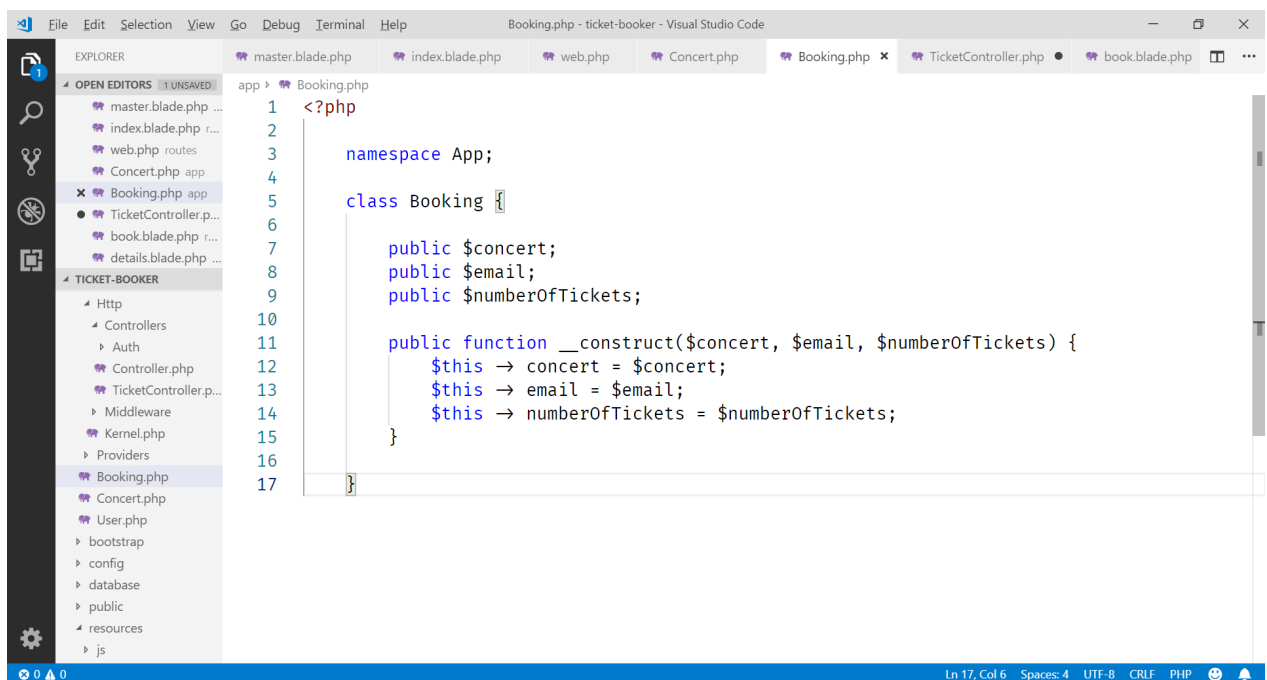
```
1 @extends("master")
2
3 @section("main-content")
4
5     <h2>Booking for {{ $concert → title }}</h2>
6
7     <form method="post" action="{{ route('ticket-booking-do') }}">
8         @csrf
9         <input type="hidden" name="concert" value="{{ $concert → id }}" />
10        <label for="email">Your email address:</label>
11        <input type="email" id="email" name="email" />
12        <label for="numtickets">Number of tickets:</label>
13        <select id="numtickets" name="numtickets">
14            @for($i = 1; $i ≤ 6; $i++)
15                <option value="{{ $i }}">{{ $i }}</option>
16            @endfor
17        </select>
18        <input type="submit" value="Book now" />
19    </form>
20
21 @endsection
```

42. In the corresponding controller method `doBooking`, we can now retrieve the concert ID, email address and number of tickets from the `Request` object:

A screenshot of the Visual Studio Code editor showing the TicketController.php file. The Explorer sidebar on the left shows the project structure with folders like app, Http, Controllers, and Ticket-BOOKER. The main editor area displays PHP code for the TicketController. The code includes a return statement for the 'tickets.index' view, and three functions: details(\$id), bookingForm(\$id), and processBooking(Request \$request). The processBooking function sets concert\_id, email, and numtickets from the request input.

```
12     return view("tickets.index", [ "concerts" => $concerts ]);
13 }
14
15 function details($id) {
16     $concert = Concert::find($id);
17     return view("tickets.details", [ "concert" => $concert ]);
18 }
19
20 function bookingForm($id) {
21     $concert = Concert::find($id);
22     return view("tickets.book", [ "concert" => $concert ]);
23 }
24
25 function processBooking(Request $request) {
26     $concert_id = $request -> input("concert");
27
28     $concert = Concert::find($id);
29     $email = $request -> input("email");
30     $numtickets = $request -> input("numtickets");
31
32     // ...
33 }
34
35 }
```

43. Before we can put our booking in the session (so it is remembered), we will group it in a **Booking** object. So, we develop a model class **Booking** with the appropriate properties:

A screenshot of the Visual Studio Code editor showing the Booking.php file. The Explorer sidebar on the left shows the project structure with folders like app, Http, Controllers, and Ticket-BOOKER. The main editor area displays PHP code for the Booking class. The code includes a namespace declaration, class declaration, and public properties for \$concert, \$email, and \$numberOfTickets. It also includes a \_\_construct method that initializes these properties.

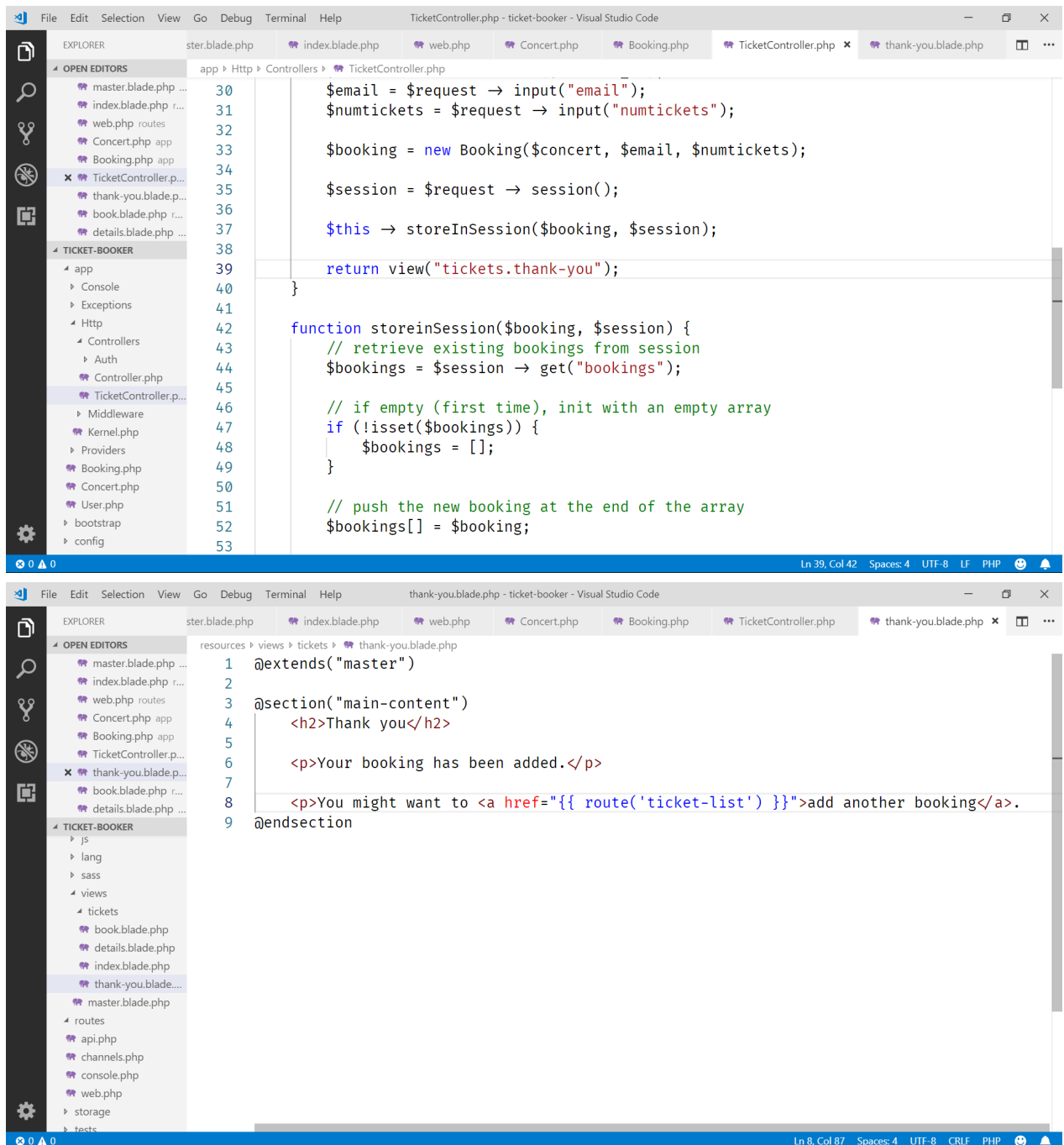
```
1 <?php
2
3 namespace App;
4
5 class Booking {
6
7     public $concert;
8     public $email;
9     public $numberOfTickets;
10
11     public function __construct($concert, $email, $numberOfTickets) {
12         $this -> concert = $concert;
13         $this -> email = $email;
14         $this -> numberOfTickets = $numberOfTickets;
15     }
16
17 }
```

44. We can now instantiate a **Booking** object and store it with the other bookings in our session:

```
File Edit Selection View Go Debug Terminal Help TicketController.php - ticket-booker - Visual Studio Code
master.blade.php index.blade.php web.php Concert.php Booking.php TicketController.php x book.blade.php
OPEN EDITORS
master.blade.php ... 33
index.blade.php r... 34
web.php routes 35
Concert.php app 36
Booking.php app 37
TicketController.p... 38
book.blade.php r... 39
details.blade.php ... 40
TICKET-BOOKER
app
  Console
  Exceptions
  Http
    Controllers
      Auth
      Controller.php
      TicketController.p...
    Middleware
    Kernel.php
    Providers
      Booking.php
      Concert.php
      User.php
    bootstrap
    config
    database
33 $booking = new Booking($concert, $email, $numtickets);
34
35 $session = $request -> session();
36
37 $this -> storeInSession($booking, $session);
38 }
39
40 function storeInSession($booking, $session) {
41     // retrieve existing bookings from session
42     $bookings = $session -> get("bookings");
43
44     // if empty (first time), init with an empty array
45     if (!isset($bookings)) {
46         $bookings = [];
47     }
48
49     // push the new booking at the end of the array
50     $bookings[] = $booking;
51
52     // store in the session
53     $session -> put("bookings", $bookings);
54 }
55
56
```

45. And finally, we make sure we are shown a thank you message, by returning a dedicated view:





Note that the hyperlink back to the index page also takes advantages of the named routing mechanism.

46. We are nearly there. Now, we need to write the "admin" page, which displays the bookings in the session. So, first we define a new route, linking to a method in the controller:

```

1 <?php
2
3 Route::get("/tickets", "TicketController@index") → name("ticket-list");
4 Route::get("/tickets/{id}", "TicketController@details")
5     → name("ticket-details");
6 Route::get("/tickets/book/{id}", "TicketController@bookingForm")
7     → name("ticket-booking-form");
8 Route::post("/tickets/book", "TicketController@processBooking")
9     → name("ticket-booking-do");
10 Route::get("/listBookings", "TicketController@listBookings")
11     → name("ticket-bookings-list");

```

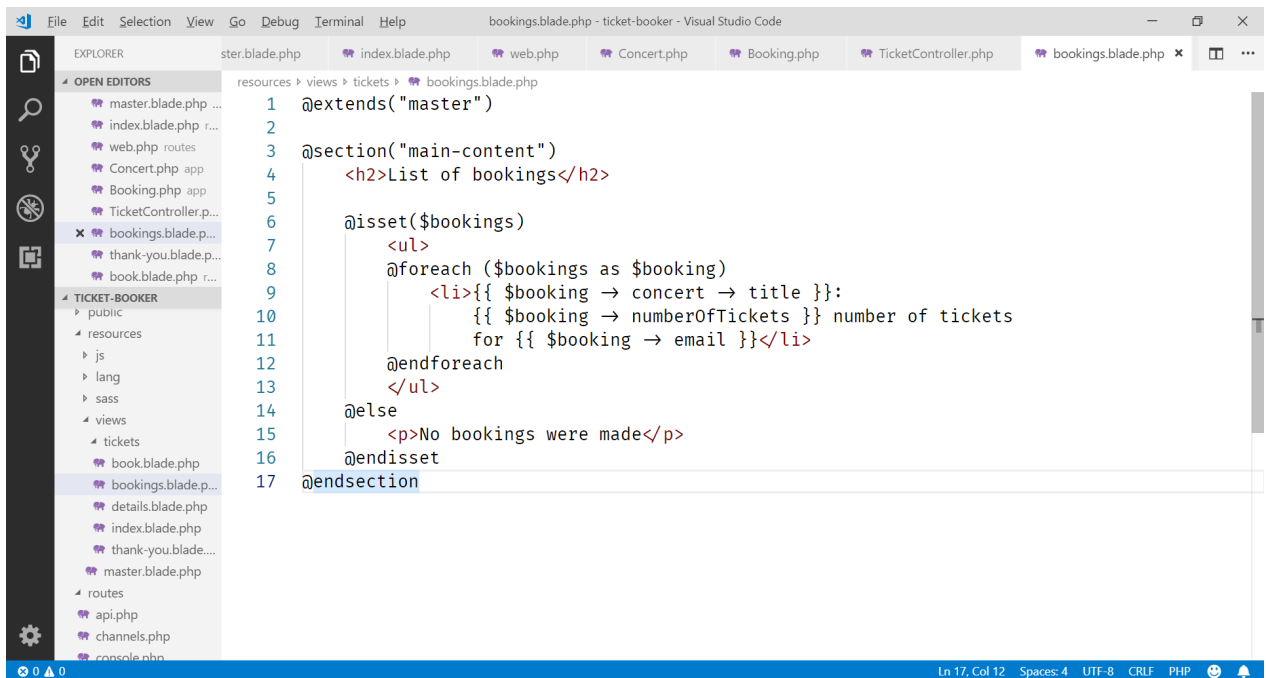
47. And we implement the controller method, in which we will retrieve all bookings from the session and pass them to a view:

```

42 function storeInSession($booking, $session) {
43     // retrieve existing bookings from session
44     $bookings = $session → get("bookings");
45
46     // if empty (first time), init with an empty array
47     if (!isset($bookings)) {
48         $bookings = [];
49     }
50
51     // push the new booking at the end of the array
52     $bookings[] = $booking;
53
54     // store in the session
55     $session → put("bookings", $bookings);
56 }
57
58 function listBookings(Request $request) {
59     $bookings = $request → session() → get("bookings");
60
61     return view("tickets.bookings", [ "bookings" ⇒ $bookings ]);
62 }
63
64

```

48. In the view, we loop over the bookings using a `@foreach` construct:



```
1 @extends("master")
2
3 @section("main-content")
4     <h2>List of bookings</h2>
5
6     @isset($bookings)
7         <ul>
8             @foreach ($bookings as $booking)
9                 <li>{{ $booking → concert → title }}:
10                    {{ $booking → numberOfTickets }} number of tickets
11                    for {{ $booking → email }}</li>
12             @endforeach
13         </ul>
14     @else
15         <p>No bookings were made</p>
16     @endisset
17 @endsection
```

49. And we are finished. When we now navigate to <http://debian-wms.local/ticket-booker/listBookings>, we get an overview of all bookings:

