

# Web, Mobile and Security

---

## Lab: Product Review Site

### 1. Creating the project

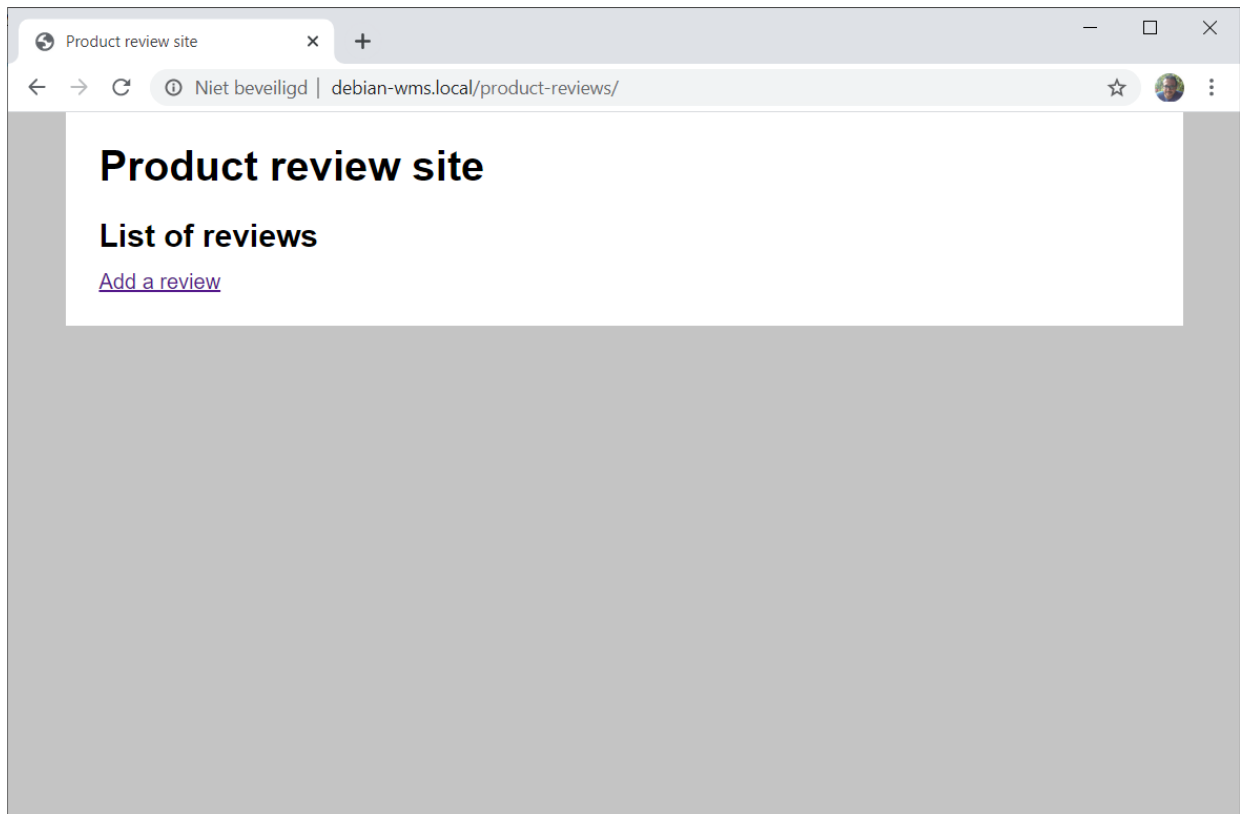
- Create a new Laravel project called **product-reviews**.
- You will also need an account to Mailtrap, which is a service to mimic a real SMTP server.
- Therefore, go to <https://mailtrap.io/> and sign up for a new **-free-** account. More details on the usage itself will be detailed later on in this assignment.

### 2. General overview of the application

- We are building a website that allows a user to write a review for a product.
- The user is also able to upload an image of the product (s)he reviews.
- When the review is submitted, an automatic e-mail message is sent.
- The list of reviews can be consulted by other visitors of the website as well.
- In a second phase, we will also implement authentication:
  - all users (including guests) can read existing reviews;
  - only authenticated users can actually add a new review;
  - the automatic e-mail message is sent to the authenticated user's mail address.

### 3. Screenshots (version without authentication)

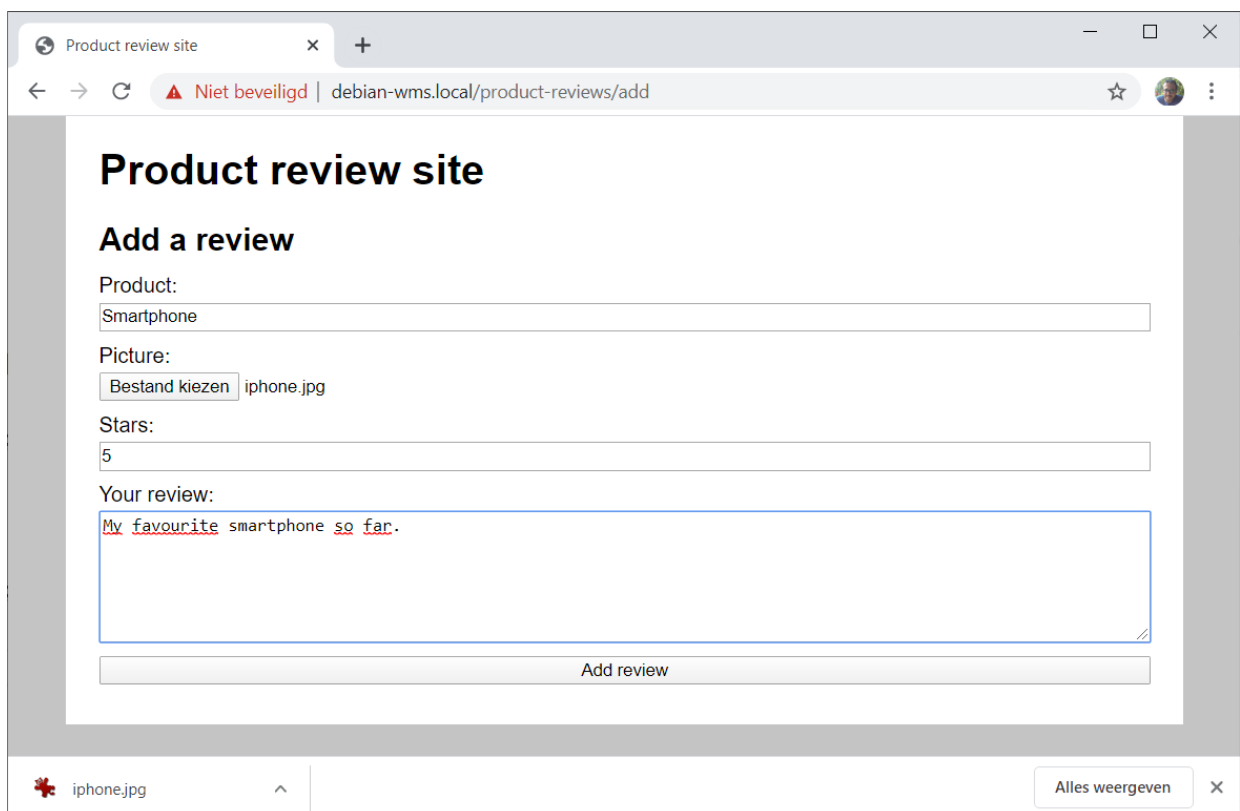
#### 3.1. The initial view, no existing reviews



Upon application launch, the user is presented with a list of existing reviews. The first time, logically, there are no reviews.

At the bottom of the list, a link **Add a review** enables the user to go to the next screen.

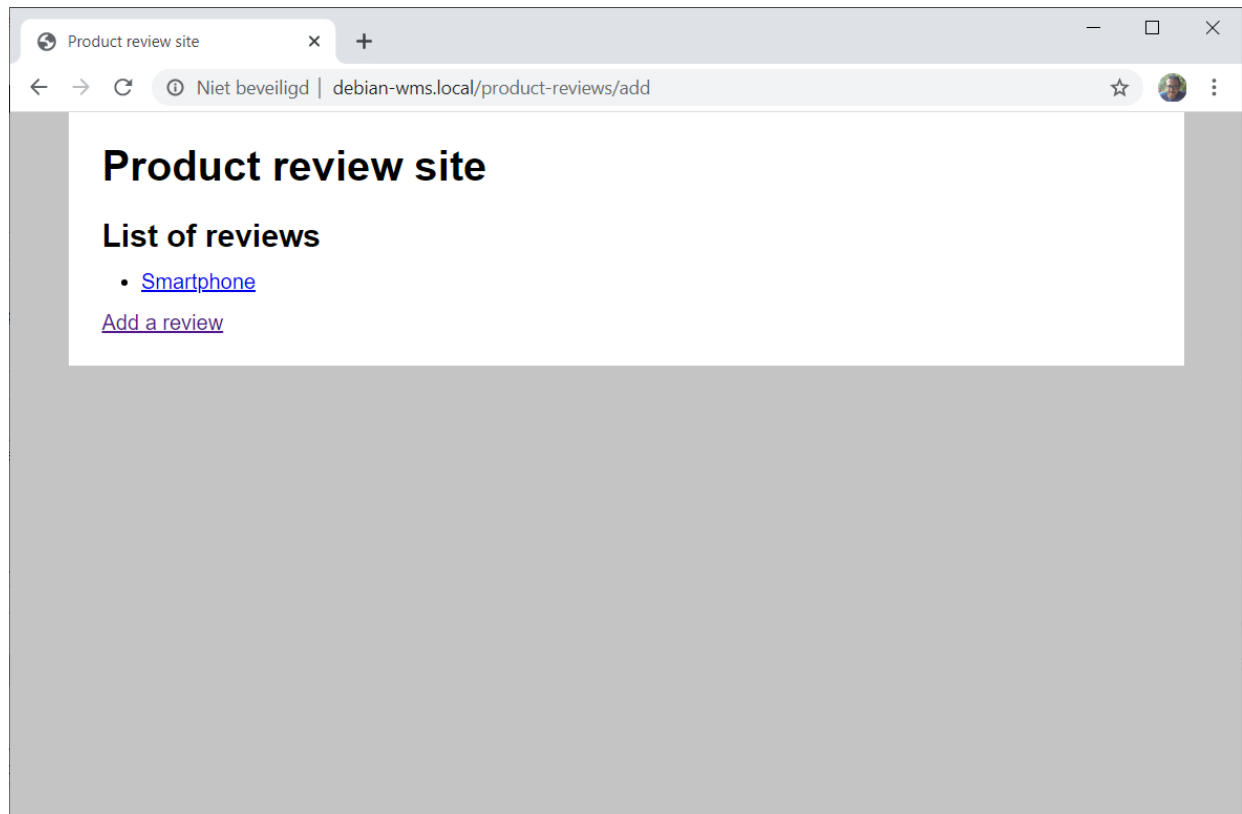
### 3.2. Adding a review



The user can then specify the product, select a picture for upload, rate the product with a number of stars between 0 and 5 and write the actual review text.

When submitting the review, it is added to the database and the user is redirected to the list of reviews.

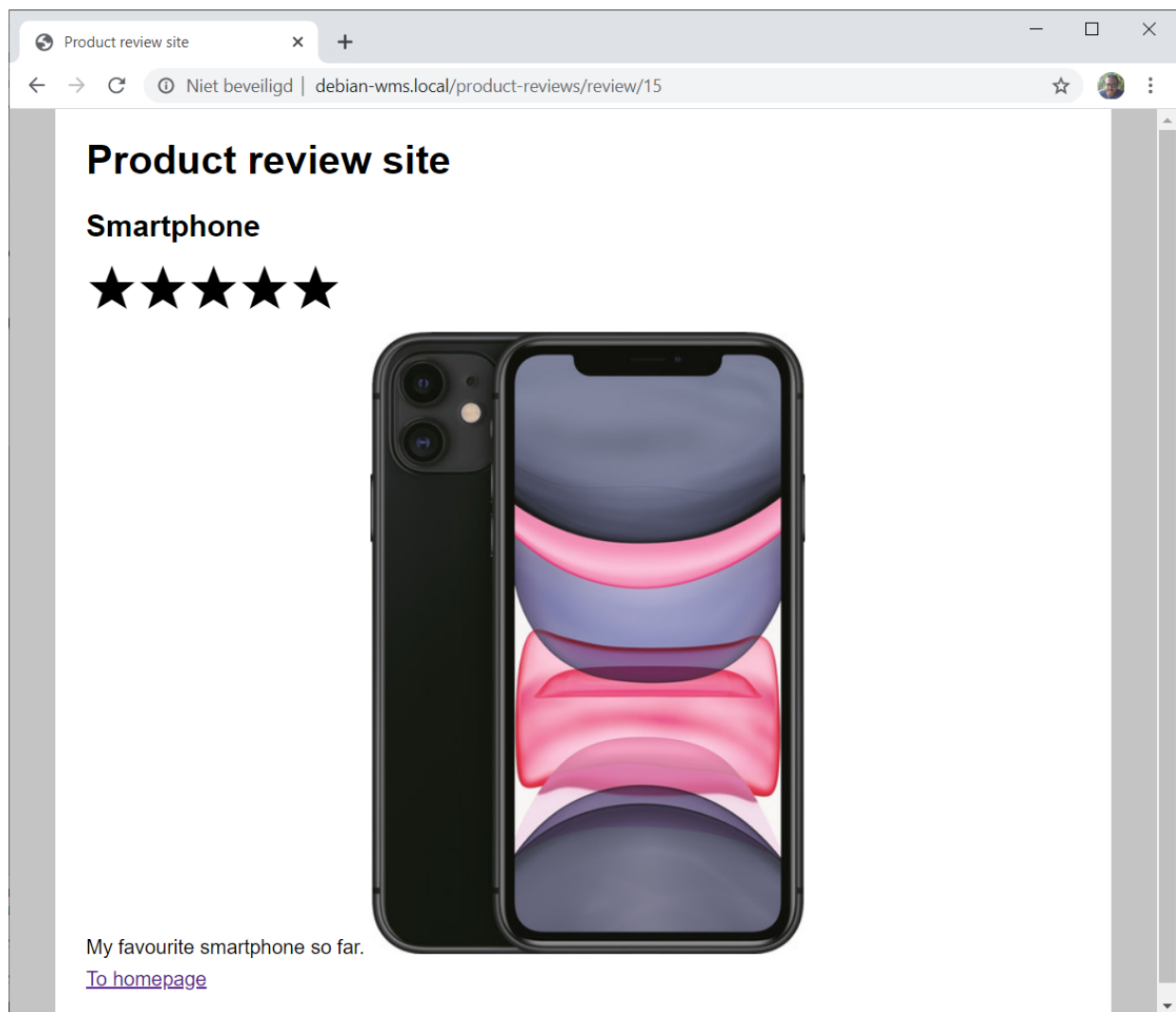
### 3.3. Consulting the existing reviews



After a product has been added, it is displayed in the list of reviews.

A user can click the product title (as seen above) and is then taken to the review page.

### 3.4. Displaying an existing review



On the product review page, the title of the product is displayed, together with the number of stars rated, as well as the uploaded image and the actual review text.

The page also has a link to go back to the homepage with all the reviews.

#### 4. Building the model

- Build a database **product-reviews**, with one table **reviews**, which will contain the individual product reviews. Each review has the following properties:
  - **id**: INTEGER, primary key, auto\_increment
  - **product**: VARCHAR(255), product name
  - **rating**: TINYINT, the number of stars
  - **image**: VARCHAR(255), the filename of the uploaded image (see later)
  - **review**: VARCHAR(2048), the actual product review
- Create a model **Review** for this table. We will not be needing timestamps.

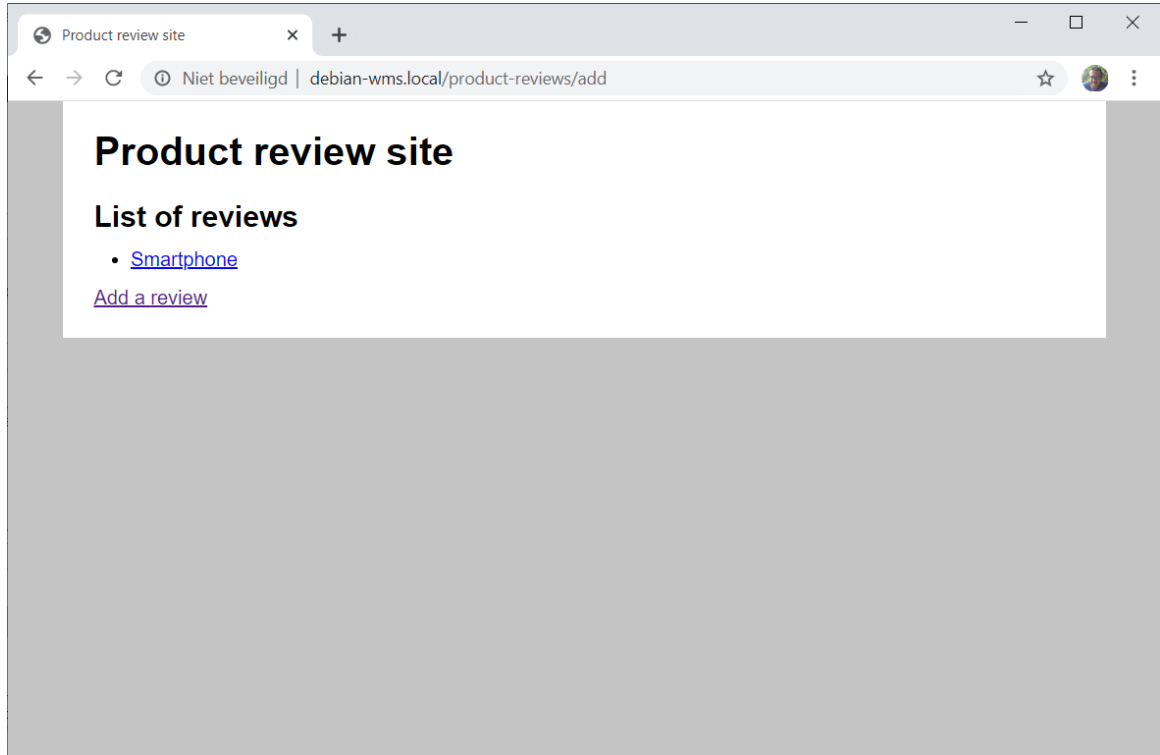
#### 5. Building the master layout page

- Create a blade "master page" called **master.blade.php**, which will function as template for the various views.
- Base yourself on the screenshots above. Which part(s) will be different per view? Isolate them using a **@yield** instruction.

- Place Your CSS files ([reset.css](#) and [screen.css](#)) in the correct directory and reference them using the appropriate Blade function.

## 6. Creating the "Index" view (and associated routing and controller code)

- This view is activated when navigating to the route [/](#).
- It displays all reviews in the database as well as a bulleted list.



- Each review is a hyperlink to a details page, existing under the route [/reviews/x](#) with [x](#) replaced by the selected review's ID.

## 7. Creating the view "Add Review" (and associated routing and controller code)

- This view is activated when navigating to the route [/add](#).
- It displays the required form, including file upload.

Product review site

## Add a review

Product:  
Smartphone

Picture:  
Bestand kiezen | iphone.jpg

Stars:  
5

Your review:  
My favourite smartphone so far.

Add review

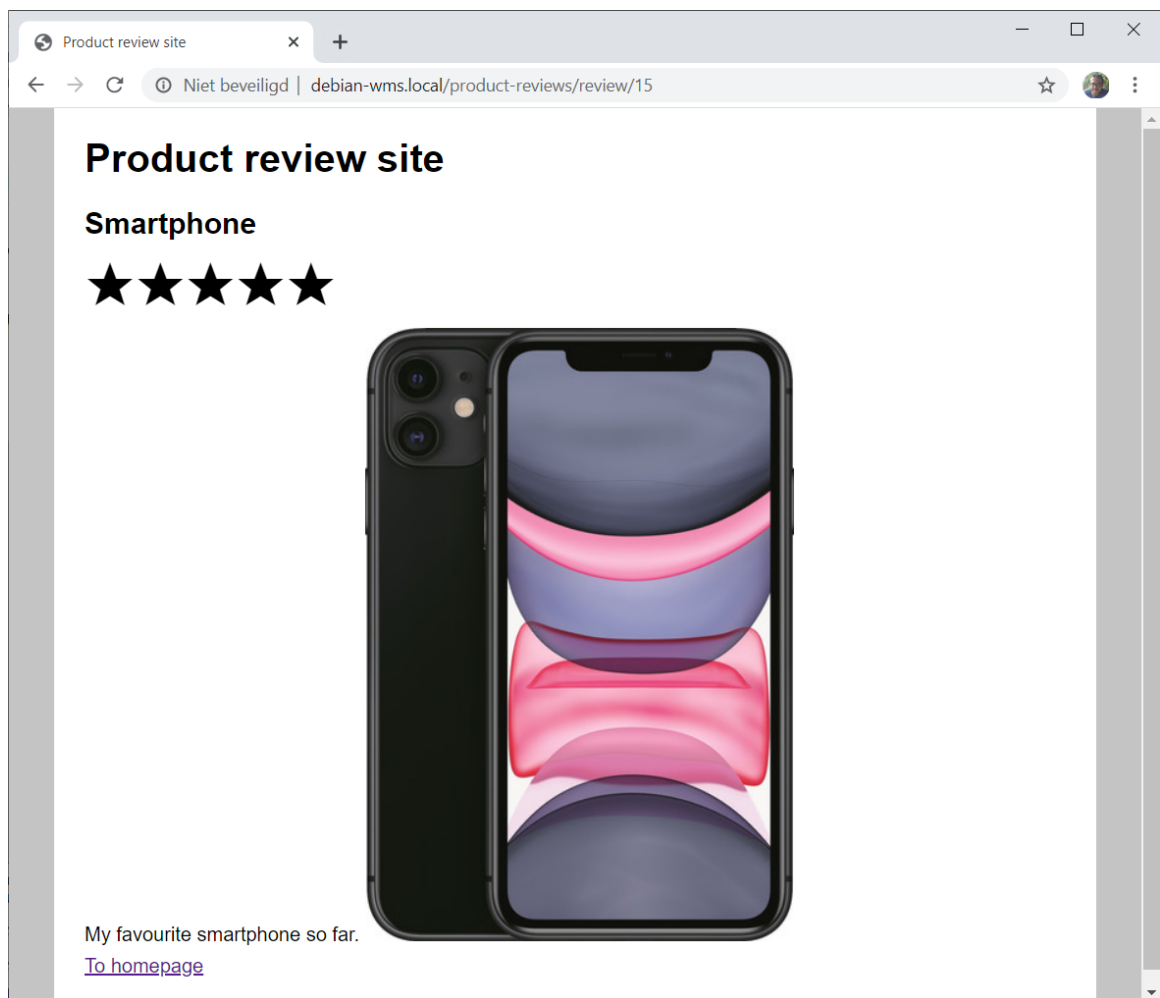
iphone.jpg

Alles weergeven

- Implement both client-side and server-side validation.
- Limit the file upload to pictures. Take a look at <https://laravel.com/docs/master/validation#rule-mimes> to help you.
- The uploaded pictures should be stored in the `./storage/app/products`. Write your controller code to do so.
- When saving the uploaded picture, you will get a filename. Make sure to store this in your model in the appropriate property (see §4).

## 8. Creating the view "Review Details" (and associated routing and controller code)

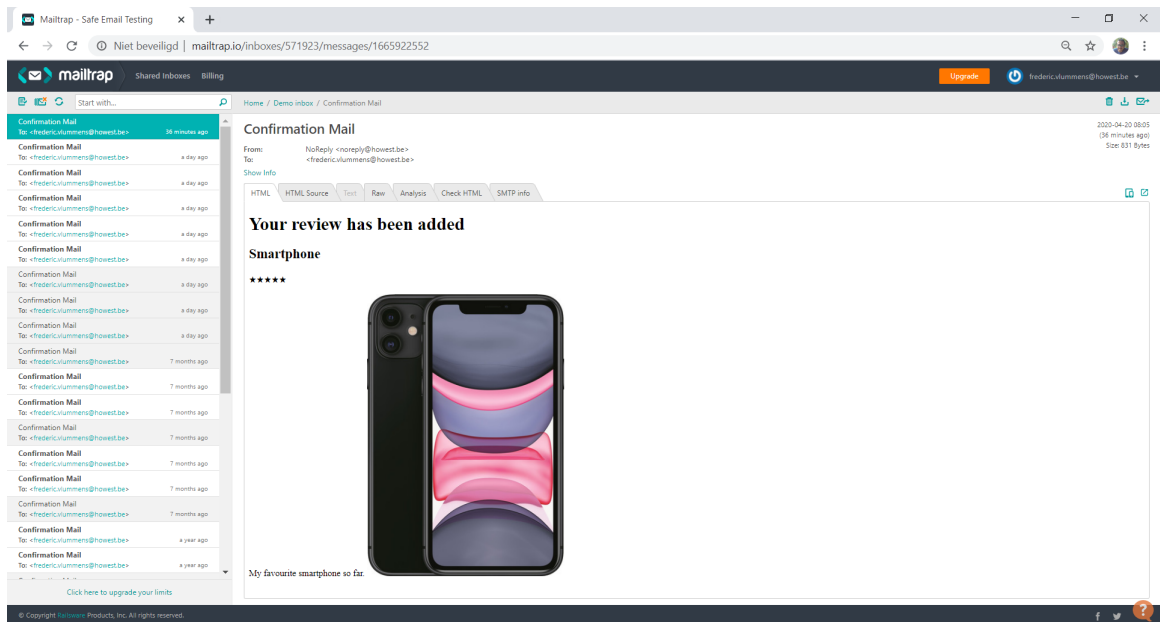
- This view is activated when navigating to the route `/reviews/x`, with `x` the unique ID of the review.
- It seen earlier, it displays the product, stars rating, uploading image, review text and a link to the homepage.



- The uploaded product images are to be stored in the directory `./storage/app/products`.
- Make sure they are also accessible publicly by defining the appropriate symlink using `php artisan storage:link`.
- To reference them, use the same `asset` function as for the CSS files.
- To obtain the correct number of stars, take a look at the PHP function `str_repeat`. Remember, you can always consult the PHP documentation on <https://www.php.net/manual/en/>.

## 9. Sending out the e-mail message

- Each time a new review is added to the database, a mail message is sent out.
- First, create a new view called `./resources\views\confirmation-mail.blade.php`, corresponding to the following layout:

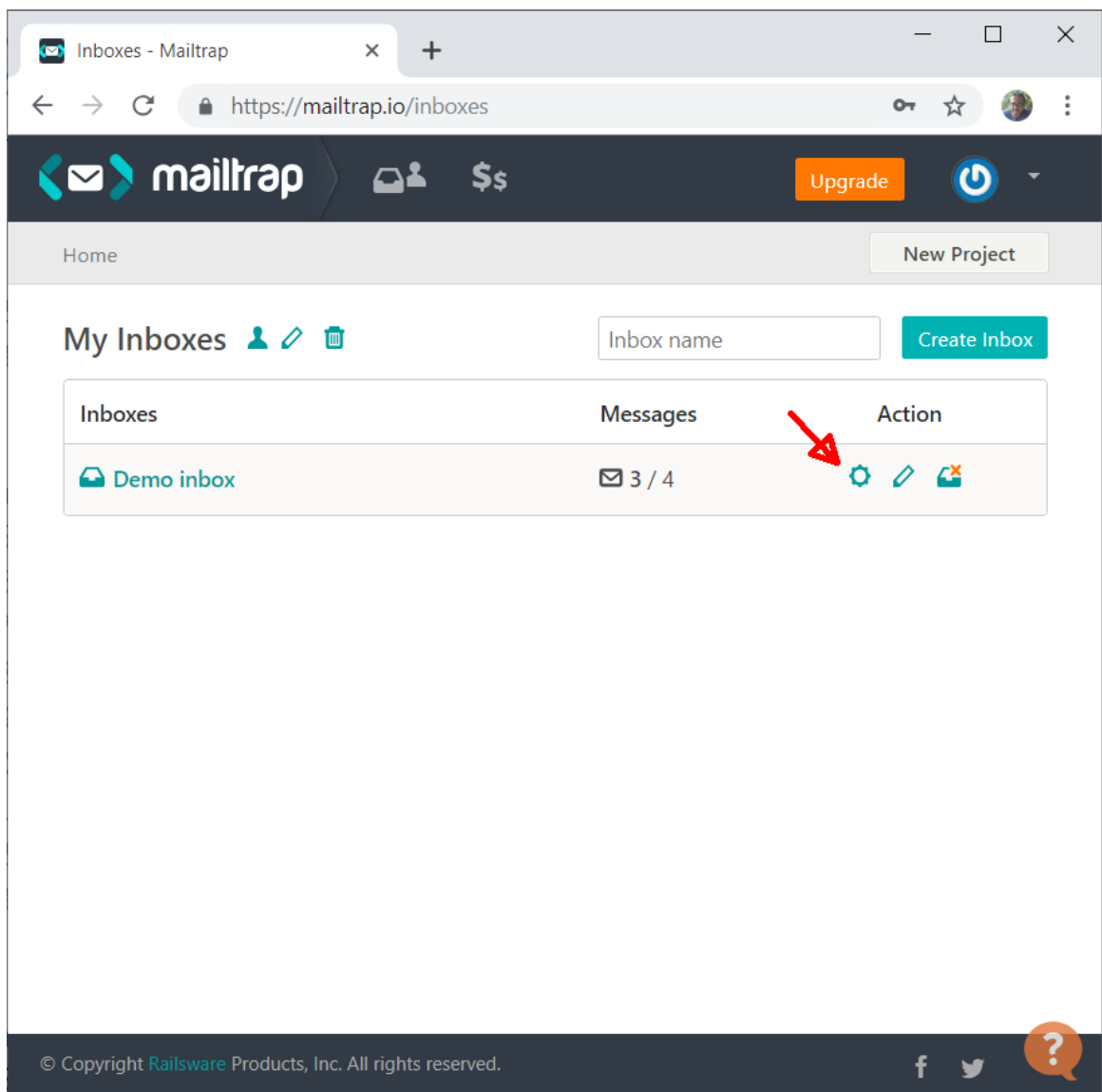


- Next, we need to make sure we correctly configure the mail mechanism in Laravel. As usual, configuration is done in the `.env` file.
- Locate the `MAIL_*` settings in said file and modify according to your Mailtrap credentials.

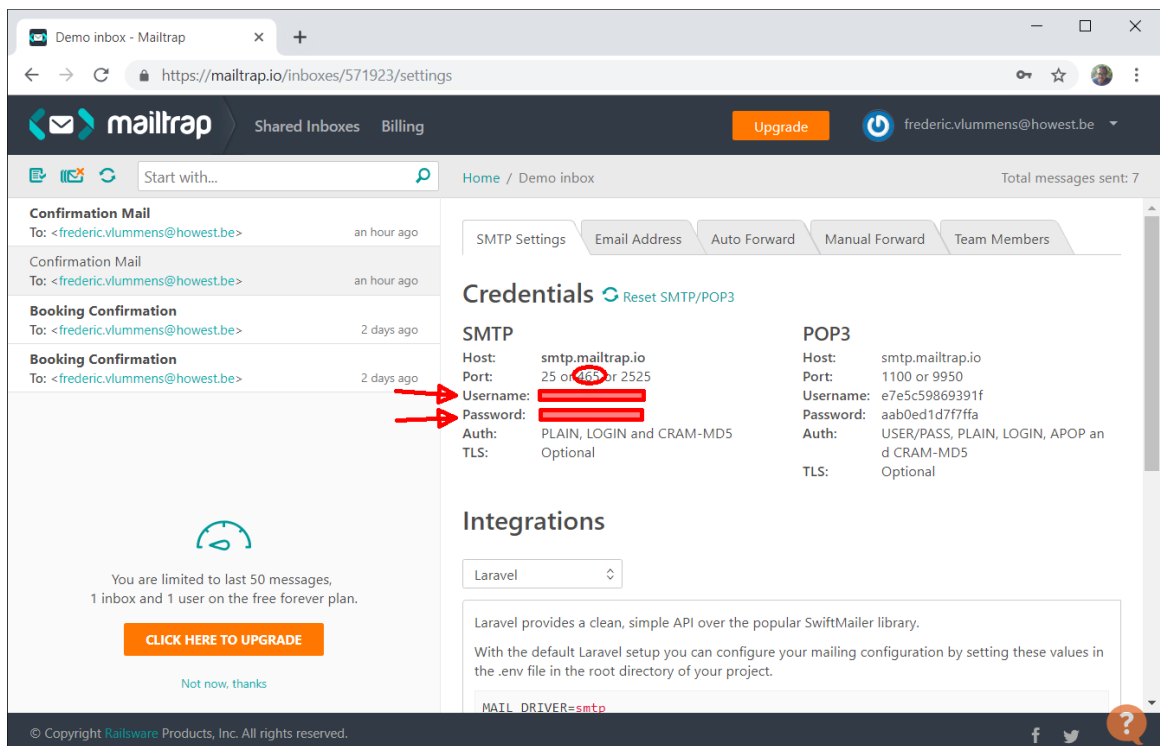
```
MAIL_DRIVER=smtp
MAIL_HOST=smtp.mailtrap.io
MAIL_PORT=465
MAIL_USERNAME=<<your own username>>
MAIL_PASSWORD=<<your own password>>
MAIL_FROM_ADDRESS=noreply@howest.be
MAIL_FROM_NAME="No Reply"
```

- To find your own username and password, go to Mailtrap and click the cogwheel icon next to your "inbox":





- You will find the username and password as in the screenshot below (marked by arrows):



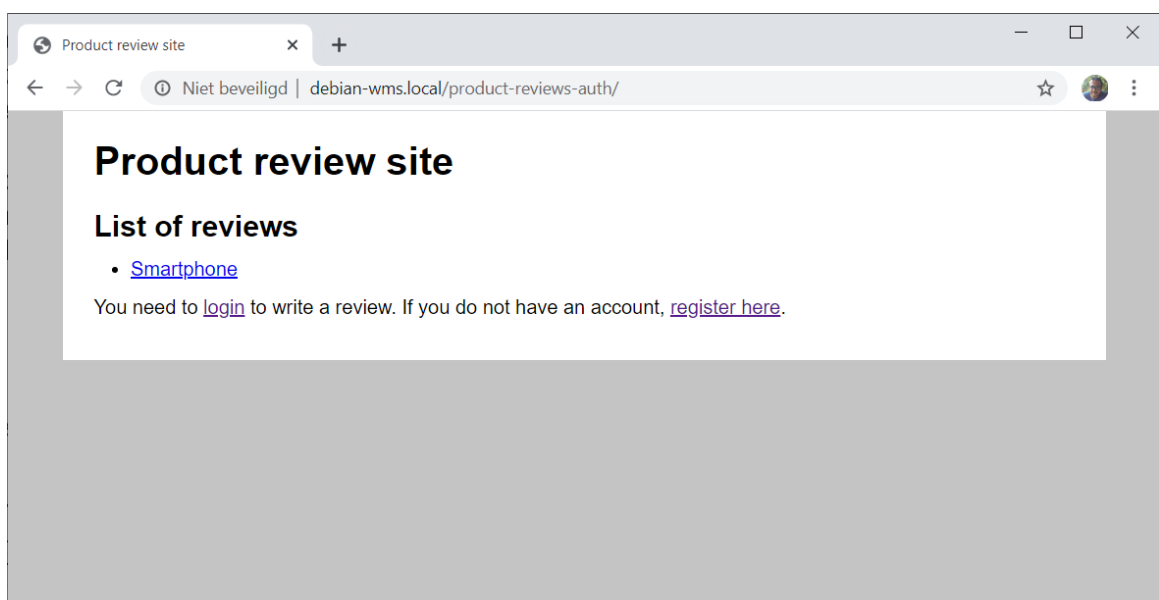
- **Attention:** on the Howest network, make sure to use mail port 465. The default port 2525 or the alternative 25 do not work. **At home, port 2525 will possibly work. You need to try this out.**
- Now that the configuration is correct, it is time to create a Mail class. To do so, in PuTTY, execute the following command, **from within your application folder:**

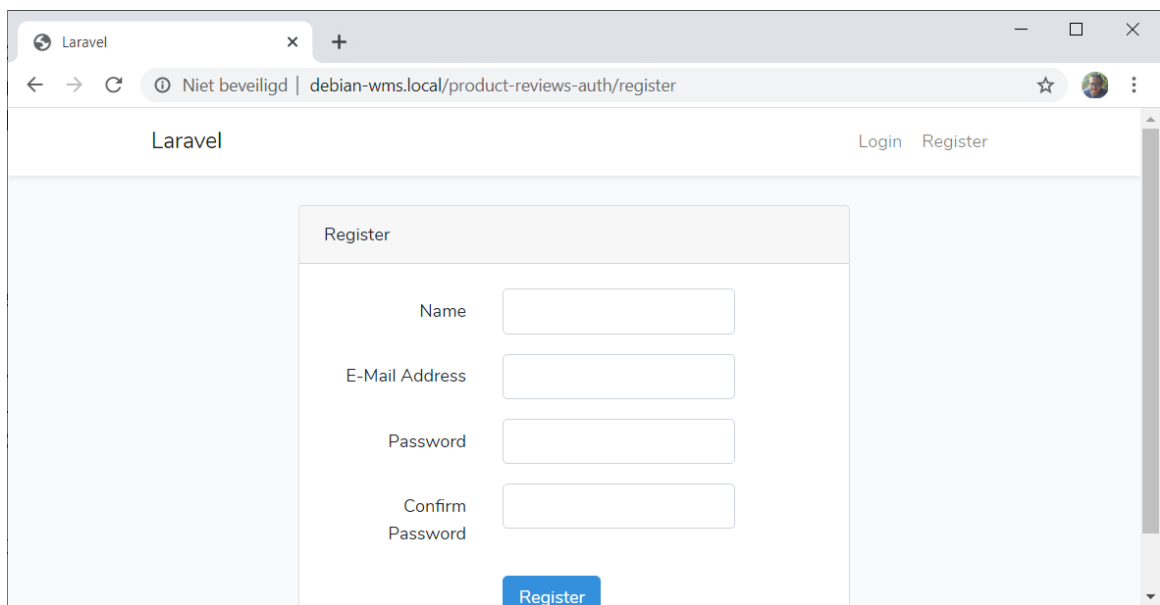
```
$ php artisan make:mail ConfirmationMail
```

- This will create a class `ConfirmationMail.php` in the folder `app\Mail`.
- Next, write the code to link the `ConfirmationMail` class to the `confirmation-mail.blade.php` file, whilst also keeping a reference to the actual `Review` to include in the mail.
- From within your controller, make sure the mail gets actually sent to your own e-mail address.
- Verify in Mailtrap that the mail has been "received".
- Not sure how to perform these steps? Take a look at the class demo on Mailables.
- Also use the Logging mechanism to log an informational message towards text file prior to *and* after sending the mail message. **There is no need to log towards Slack.**
- **Note:** for a real-life application, we would not use Mailtrap, but a real SMTP server to send out actual messages.

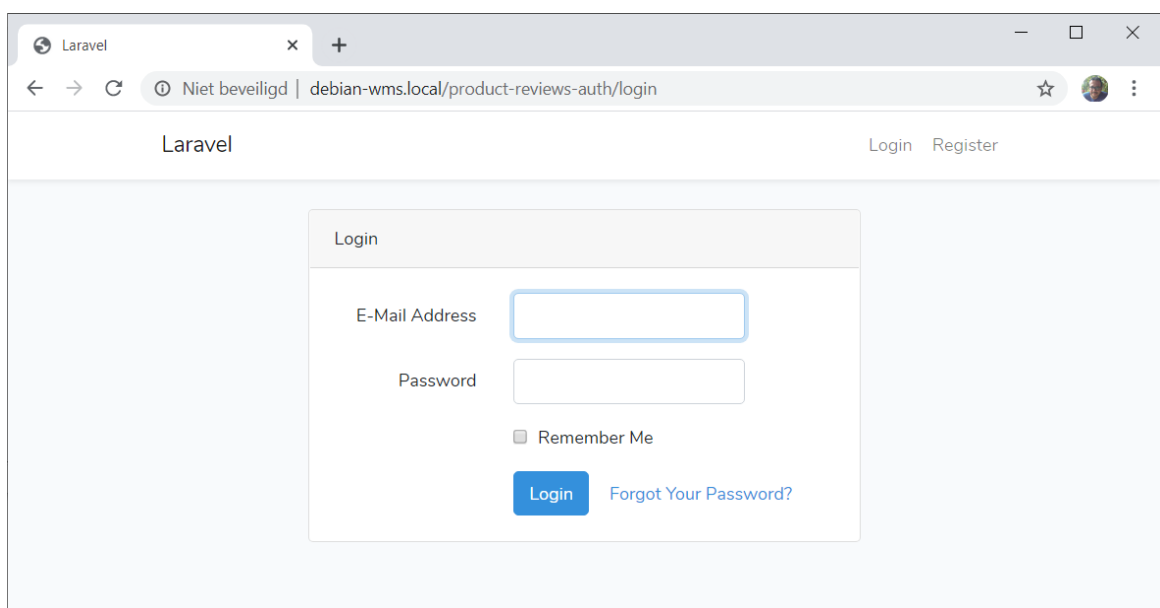
## 10. Adding authentication: general overview

- Once you are ready with the previous steps, implement authentication in your solution.
- **All** visitors should be able to consult the existing reviews, but **only authenticated users can access the route to add a review.**
- Make sure users can also login/register using the auto-generated Laravel views.





The screenshot shows a web browser window with the URL `debian-wms.local/product-reviews-auth/register`. The page has a header with the 'Laravel' logo and links for 'Login' and 'Register'. The main content area features a 'Register' form with four input fields: 'Name', 'E-Mail Address', 'Password', and 'Confirm Password'. A blue 'Register' button is located at the bottom of the form.



The screenshot shows a web browser window with the URL `debian-wms.local/product-reviews-auth/login`. The page has a header with the 'Laravel' logo and links for 'Login' and 'Register'. The main content area features a 'Login' form with two input fields: 'E-Mail Address' and 'Password'. Below the password field is a checkbox labeled 'Remember Me'. At the bottom of the form are a blue 'Login' button and a link that says 'Forgot Your Password?'.

## 11. Adding authentication: individual steps

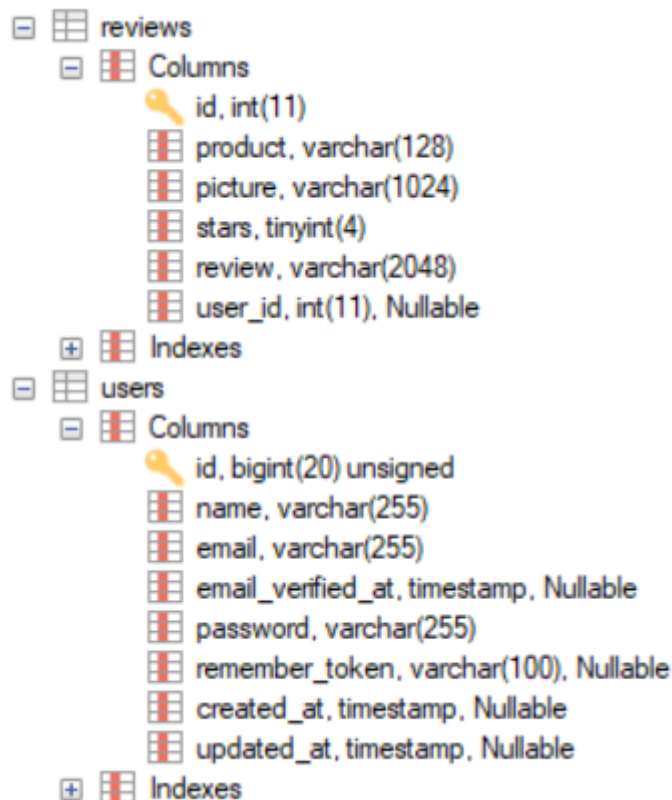
- First, execute the necessary commands to generate/scaffold the models, views and controllers.
- Execute the database migrations to generate the necessary tables.
- Next, make sure each **Review** can be linked to a **User**. We will be implementing a **1-N** relationship in our Eloquent model: each User can have multiple Reviews.
  - To do so, add the following code in the **User** model:

```
public function reviews() {  
    return $this -> hasMany("App\Review");  
}
```

- This will tell Eloquent that one User can have multiple Reviews.
- Next, add the following code in the **Review** model:

```
public function user() {
    return $this -> belongsTo("App\User");
}
```

- This will tell Eloquent that one Review belongs to one specific User.
- Of course, we will also need to model this 1-N relationship in our database.
  - Add a column `user_id` in the `reviews` table.
  - The idea is that each time a user adds a review, his/her ID is added to this column, through Eloquent.
  - Your database model should now look something like this:



- When adding the review (in your controller), you should now also associate the logged in user with the review:

```
$review = new Review();

// your existing code

$review -> user() -> associate(Auth::user());
```

- As you can see in the code above, we associate the authenticated user (`Auth::user()`) with the review through the `user` relationship.
- Make sure the route `/add` is only accessible for authenticated users.

- Finally, make sure the email is now no longer sent to your mail address, but to the mail address of the authenticated user itself. You can query the **Review** model's **user** property as follows to obtain the entire **User** object:

```
$user = $review -> user;
```

This User object has various properties, including one for the mail address.

## 12. Ready? Some time left?

Feel free to restyle the auto-generated views to use your CSS instead of Bootstrap.