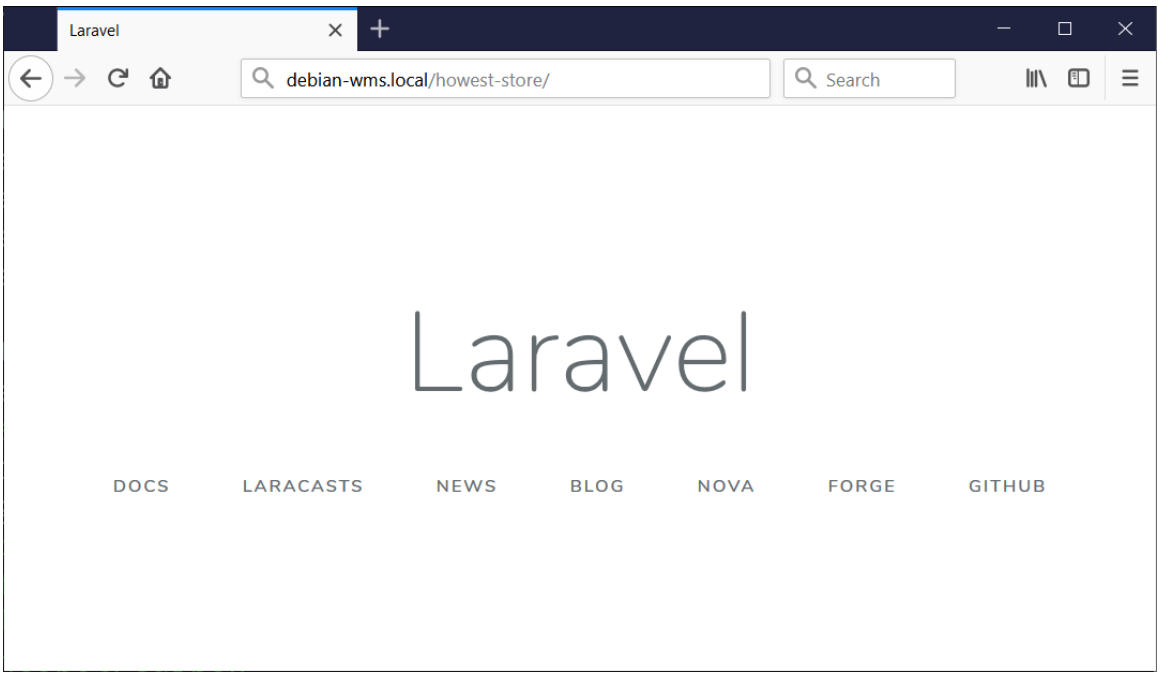
Web, Mobile and Security

Lab: Howest Store

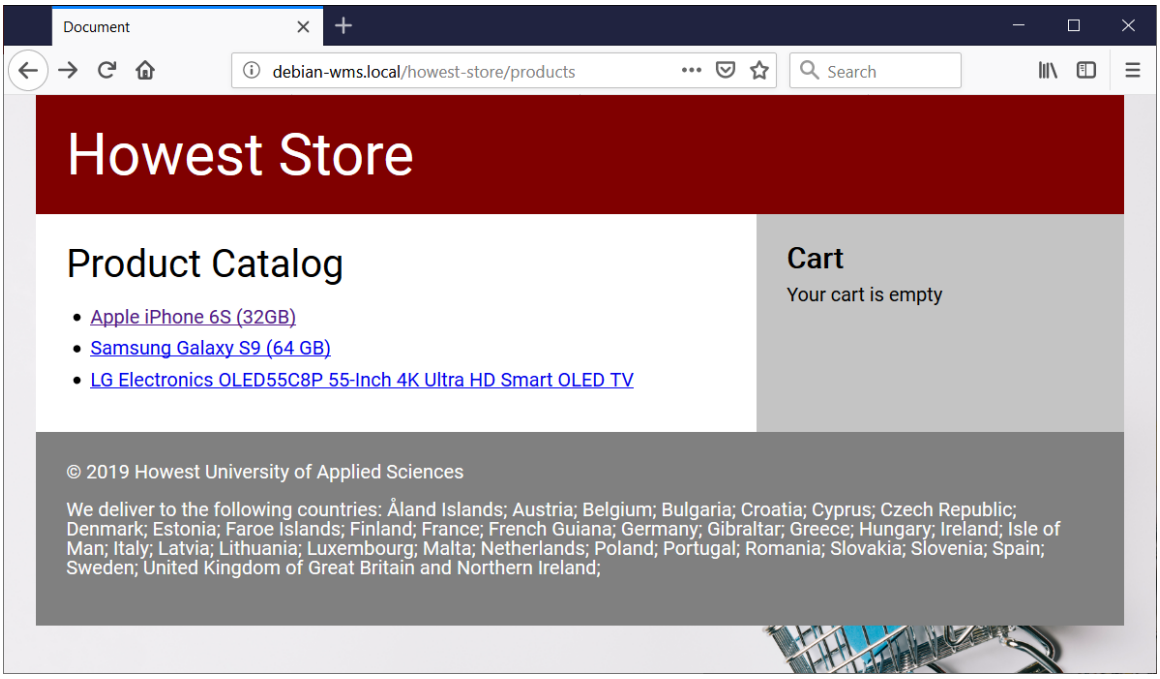
1. Creating the project

- Create a new Laravel project called `howest-store`.
- Try surfing to http://debian-wms.local/howest-store and make sure you get the default Laravel start page:

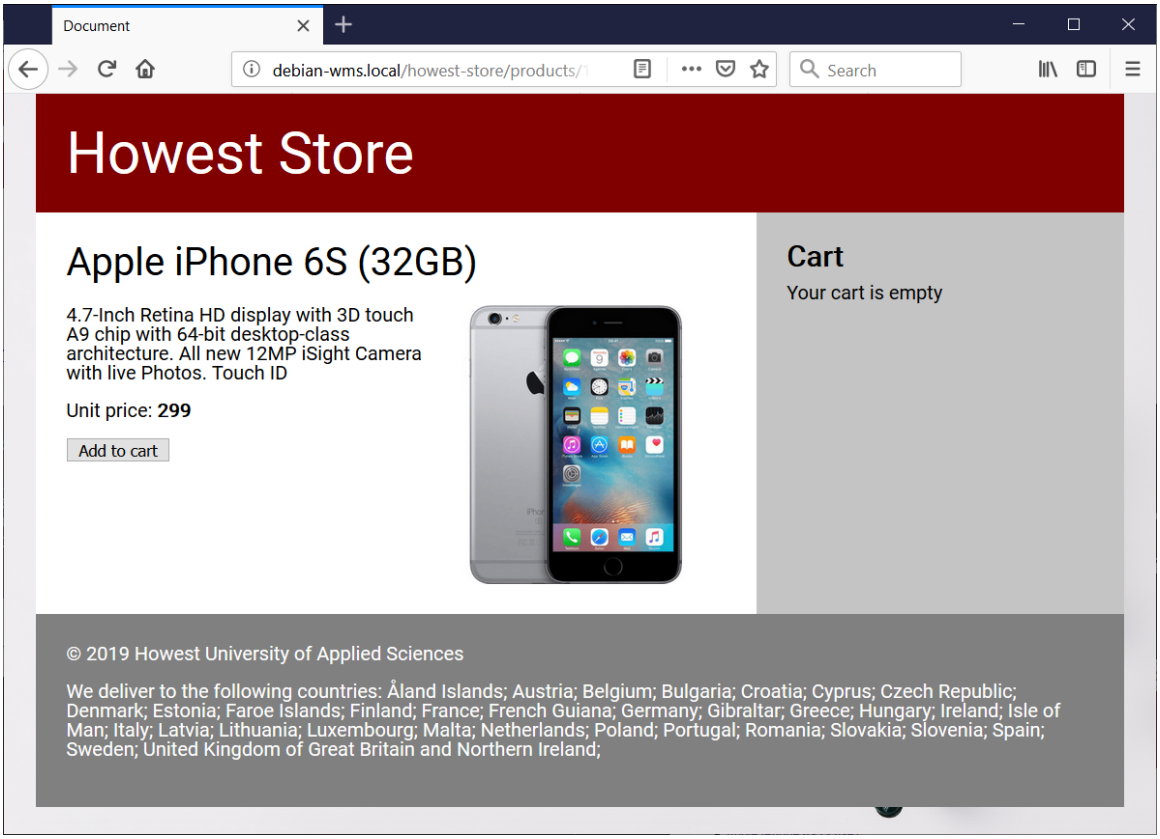


2. General overview of the application

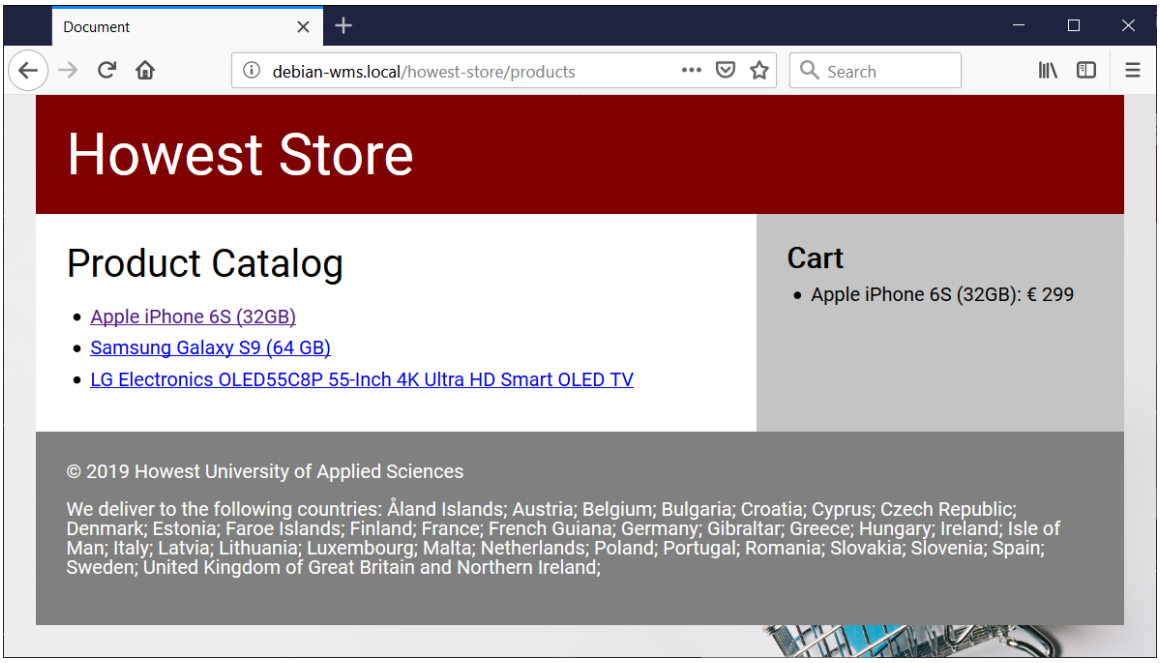
- We are creating the brand new e-commerce site for the *Howest Store*.
- Upon launch of the application, the user gets an overview of all products offered by the store:



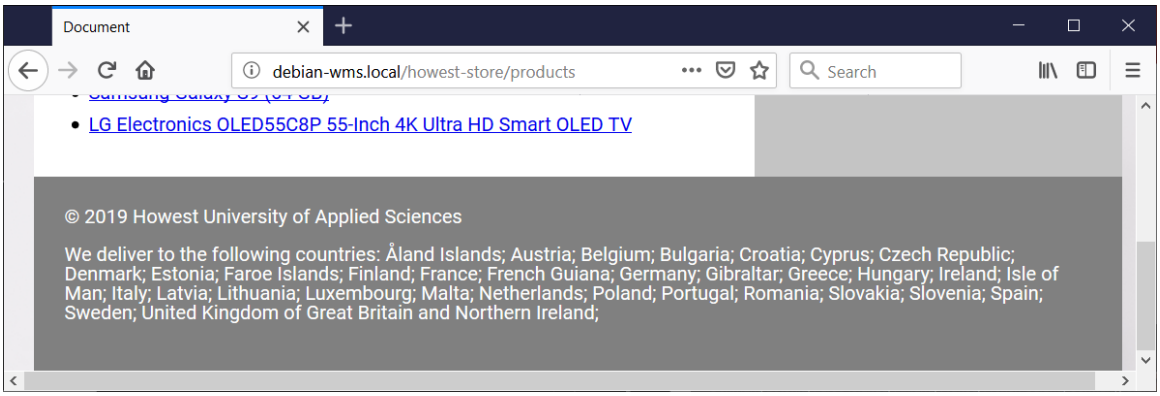
- When clicking a product, its details are displayed and the user may opt to add it to the cart:



- Once a product has been added to the cart, it is displayed at all times at the right hand side of the page:



- The Howest Store only delivers to EU countries. The list of countries is displayed at the bottom of the page, but retrieved dynamically from an API:

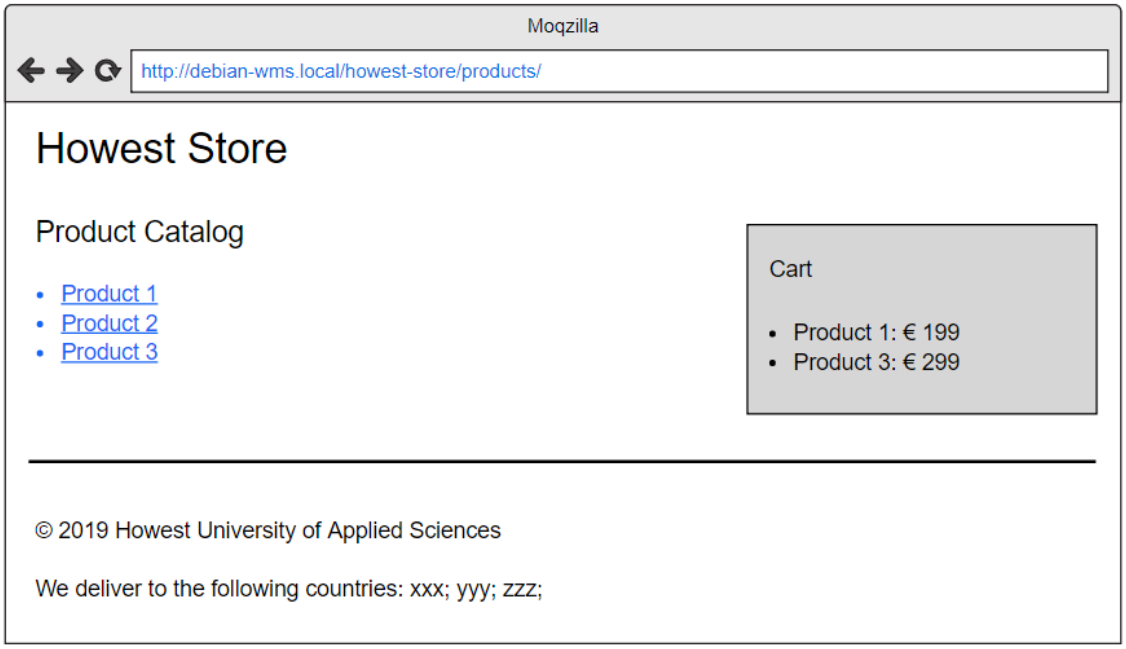


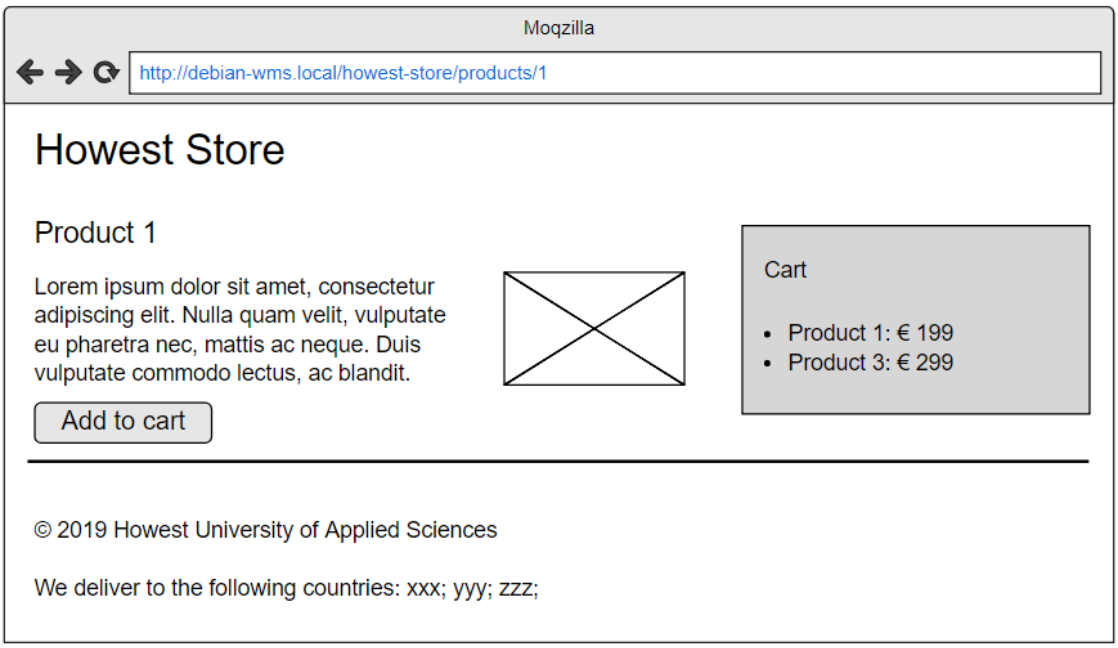
3. Building the model

- Build a database `howest-store`, with one table `products`, which will contain the individual products of the store. Each product has the following properties:
    - `id`: INTEGER, primary key, auto_increment
    - `name`: VARCHAR(255), friendly name
    - `description`: VARCHAR(1024), detail of the product
    - `price`: DOUBLE, unit price of the product
- Create a model `Product` for this table.

4. 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 as well as the wireframes below to build this page. Which part(s) will be different per view? Isolate them using a `@yield` instruction.



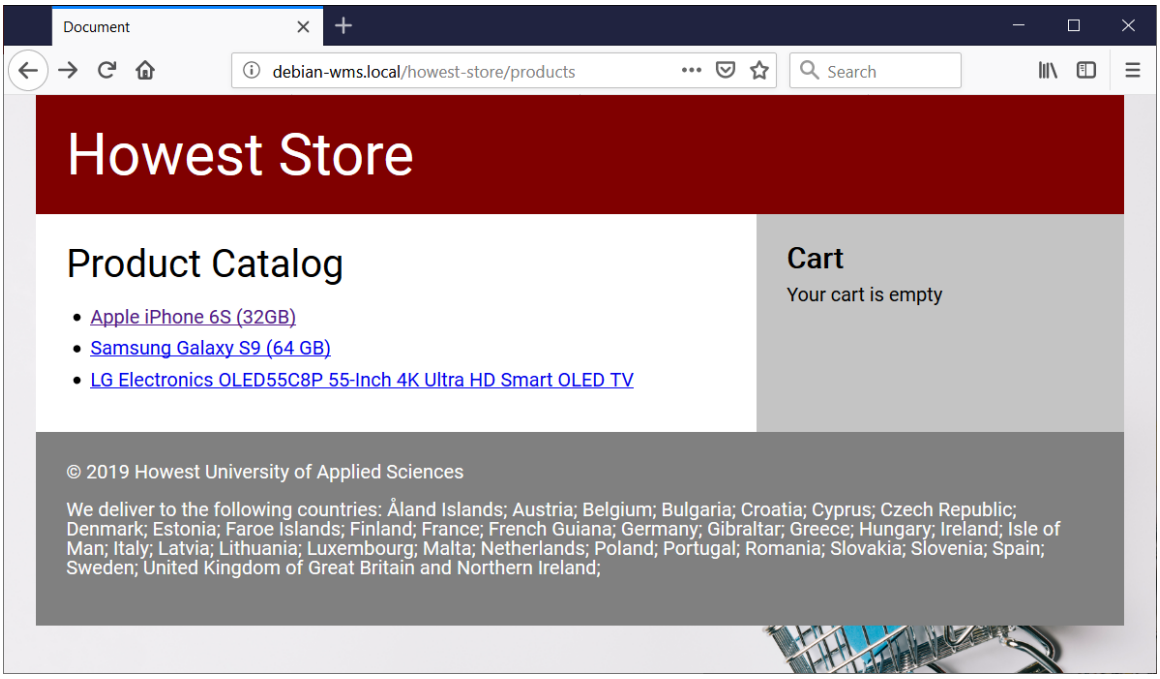


- Your CSS files (`reset.css` and `screen.css`) should be placed in the subdirectory `/public/css` of your application.
- To reference your CSS file(s), use the Blade function `asset`, for example:

```
<link rel="stylesheet" type="text/css" href="{{ asset('css/reset.css') }}" />
<link rel="stylesheet" type="text/css" href="{{ asset('css/screen.css') }}" />
```

5. Creating the view "Product Catalog" (and associated routing and controller code)

- This view is activated when navigating to the route `/products`.
- It displays all products in the catalog as a bulleted list.
- Each product is a hyperlink to a details page, existing under the route `/products/x` with `x` replaced by the selected product's ID.



- To build the hyperlink to the second route, use the `route` function.

  Hint: Take a look at https://laravel.com/docs/master/routing#named-routes for more information on how you can pass arguments to the `route` function.

- The cart is -as always- displayed at the right hand side of the page. Do not focus yet on the contents of the cart. We will tackle this later.

- The footer is -as always- displayed on the bottom of the page. Do not focus yet on the list of countries. We will tackle this later.

- **Important:**

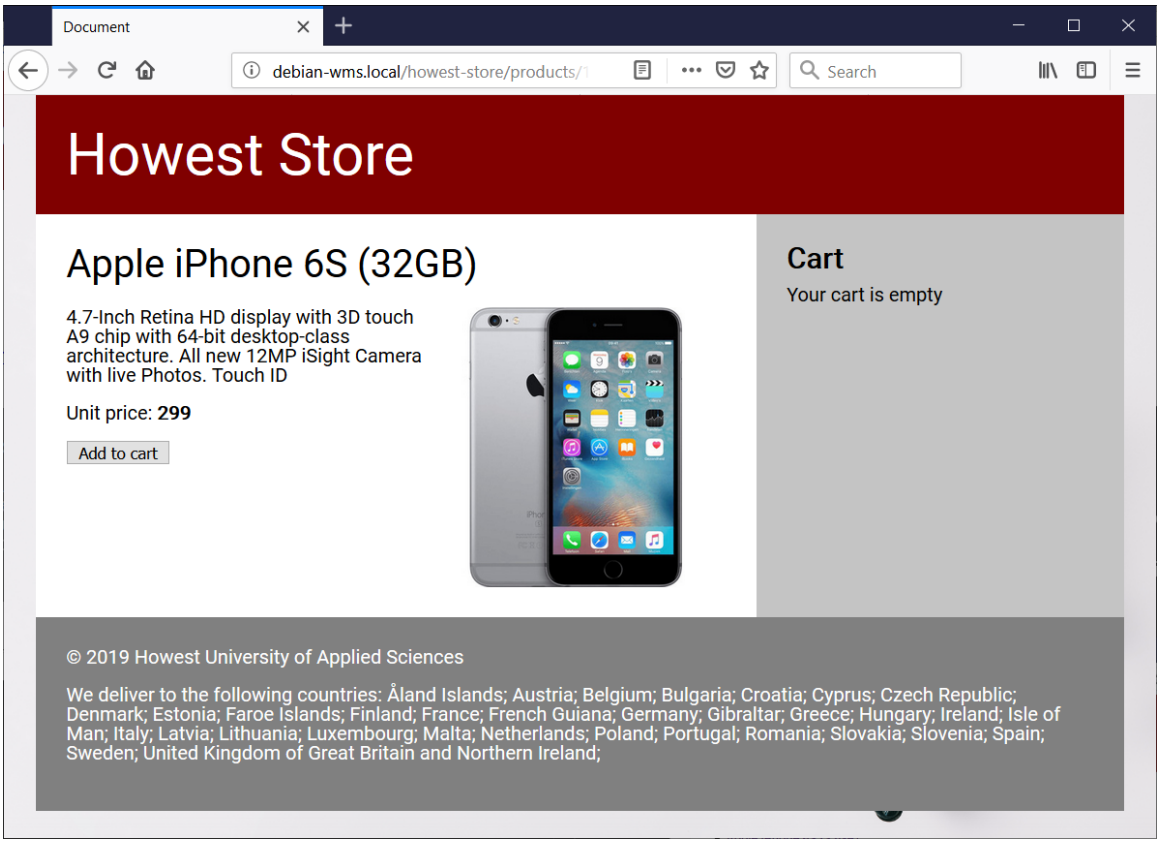
  - The products are rendered *dynamically* based on what the model (database) returns. Should we add a fourth product into the table `products`, the page should automatically also display said product.

  - Therefore, you will not include static HTML in the view with three list items and product names, **everything** comes from the model.

6. Creating the view "Product Details" (and associated routing and controller code)

- This view is activated when navigating to the route `/products/x`, with `x` the unique ID of the product.

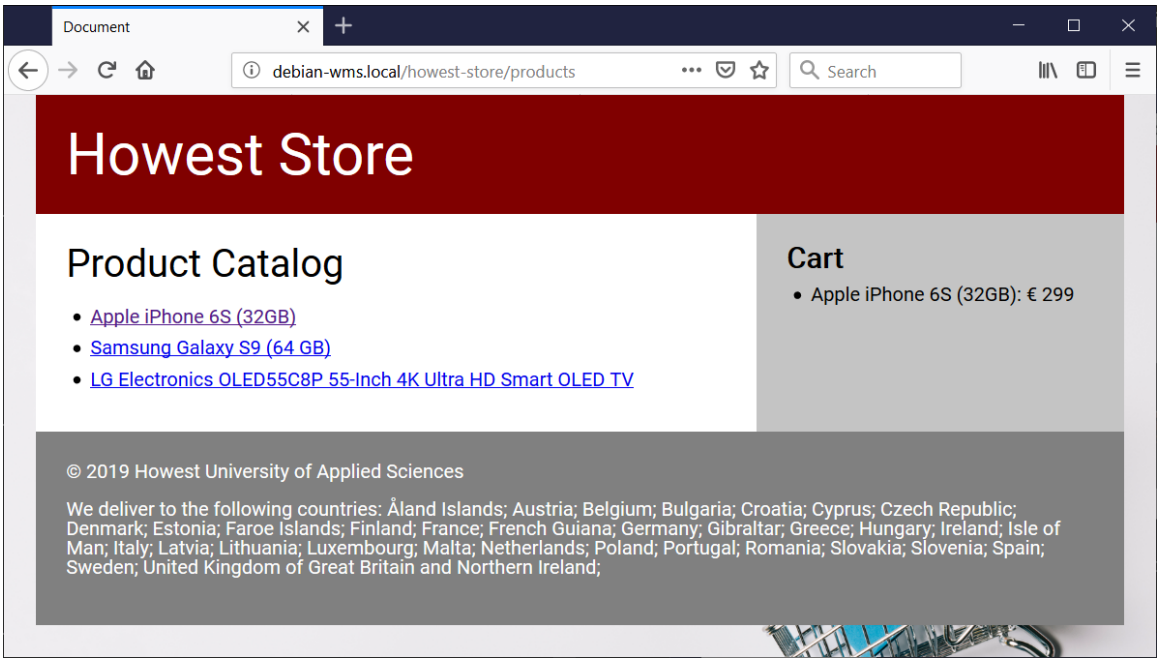
- It displays the product's title, description, image as well as the unit price.



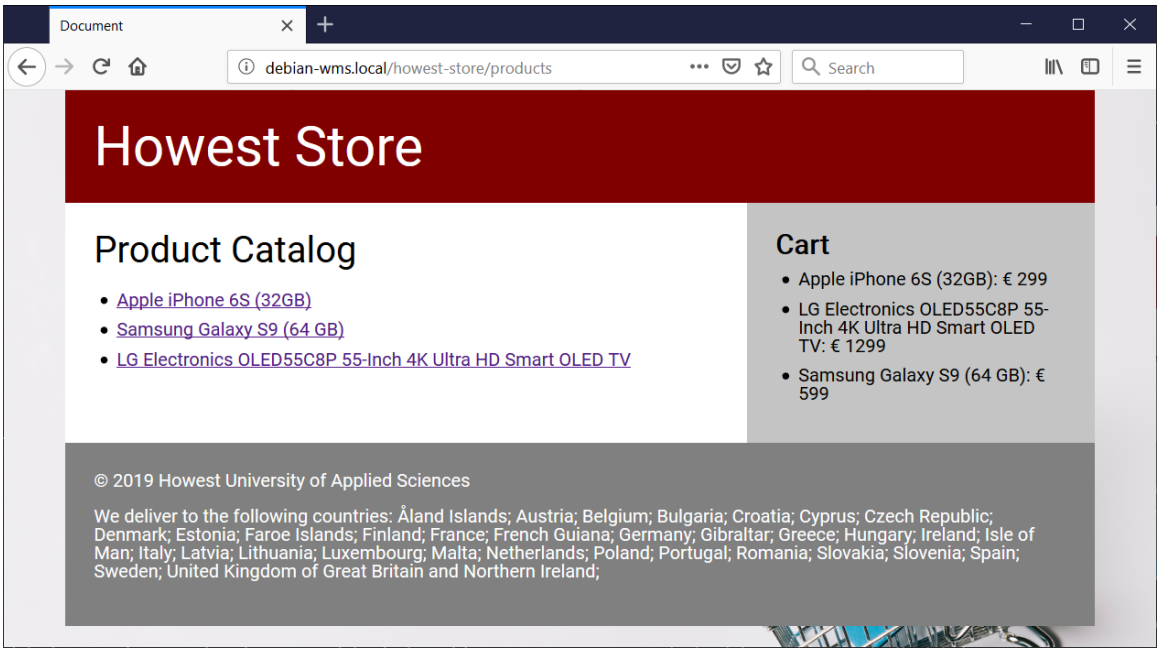
- The product images (whose names correspond to the product IDs), can be stored in the directory `/public/images` of your application.
- To reference them, use the same `asset` function as for the CSS file.
- The button `Add to cart` issues a request to the server to effectively add the selected product to the cart. **Choose the appropriate HTTP action method (verb) for your form.**
- Hint: remember you can use `input type="hidden"` to pass hidden data to the server side.
- Refer to the next section (§7) to develop the actual cart mechanism.

7. Developing the cart mechanism

- Once you are able to consult the catalog and view a product's details, it is time to develop the actual cart mechanism.
- When the user clicks the `Add to cart` button, your controller should:
    1. determine the product to add to the cart;
    2. add it to the cart in such a fashion that it remains there until further notice.
- Choose the appropriate mechanism to obtain this behaviour.
- If the cart exists and contains one or more products, the products are to be displayed at the right hand side of the page:



- The products remain there, regardless of whether you navigate to the main product catalog or to a product's details page.
- Each time the `Add to cart` button is clicked, an additional product is added to the cart, without losing the previous one:



8. Retrieving the list of countries

- Once the shopping mechanism works, it is time to add the list of countries to the footer.
- The EU countries can be retrieved using the API hosted at https://restcountries.eu/.
- Take a look at the documentation and find the endpoint/URL which delivers you **only the EU countries**.
- Using **Fetch**, load these countries from the API and display them in the footer of your views.

- Your JavaScript file can be placed in the directory `/public/js`. You may remove any existing JavaScript files Laravel pre-created in this directory, as we will only be using our own JavaScript.
- To reference your JavaScript file, once again use the `asset` function.

9. Order functionality (extra challenge!)

- Up til now, the user is not able to actually **place** an order.
- Extend the application to support this functionality, which should at least offer the following:
    - At the end of the process the user enters his/her email address.
    - The email address is validated at server side upon form submission.
    - An error message is displayed in case the email address is missing/not valid.
    - When the validation succeeds, the email address is stored in a table `clients` (create a corresponding model for it)
    - The orders themselves (i.e. individual products) are stored in a table `orders` (create a corresponding model for it).
    - Make sure there is a foreign key in the table `orders` that links back to the corresponding `client`.
    - In the end, your database diagram should look something like this: