SEMESTER PROJECT
E-SHOP 'OM NOM NOM'

Lubomir Dlhy
Anastasiia Solomiia Hrytsyna

Course: Introduction to Web Technologies
Semester: WS 2021/2022
Supervisor: Ing. Eduard Kuric, PhD.
Year of study: 3

CONTENT

ASSIGNMENT TASK

With your teammate create your own web application (eshop) in the chosen domain (electrical, clothing, footwear, furniture etc.), which solves all of the specified use cases.

GENERAL IDEA

The main theme of our project was an **online shop for candies and other sweet stuff** called 'OM NOM NOM'. We have successfully implemented the **sign up**, **login** and **logout** part, where each user of our web page passes the authentication in case he or she is registered in our eshop. All of the customers have an opportunity to **view categorized products**, **filter** or **sort** them, **search** essential ones and, obviously, **check their details**. Besides, each eshop client may **add any available product to the cart**, **change its quantity**… Afterwards the client may fulfill all necessary **shipping and payment information** and complete an order.
At the same time only **authorized** page **admin** may **check all of the existing products**, **add new** or **delete** needless **products**, **change some information** about the existing ones.
For the user **role**, we used Laravel's auth middleware and for the admin we have developed custom **AuthAdmin middleware** which checks if the authenticated user is admin (attribute is_admin). The whole AdminController uses this middleware so unauthorized users or guests will not be able to call methods inside this class. We haven't used Laravel's Policies because the implementation of custom middleware was easier and fulfills the requirements too. For managing (saving/deleting) **images** we used **Laravel's File Storage API**. For **filling the database** with all information we used Laravel's **Seeders** & **Factories** for generating artificial data.

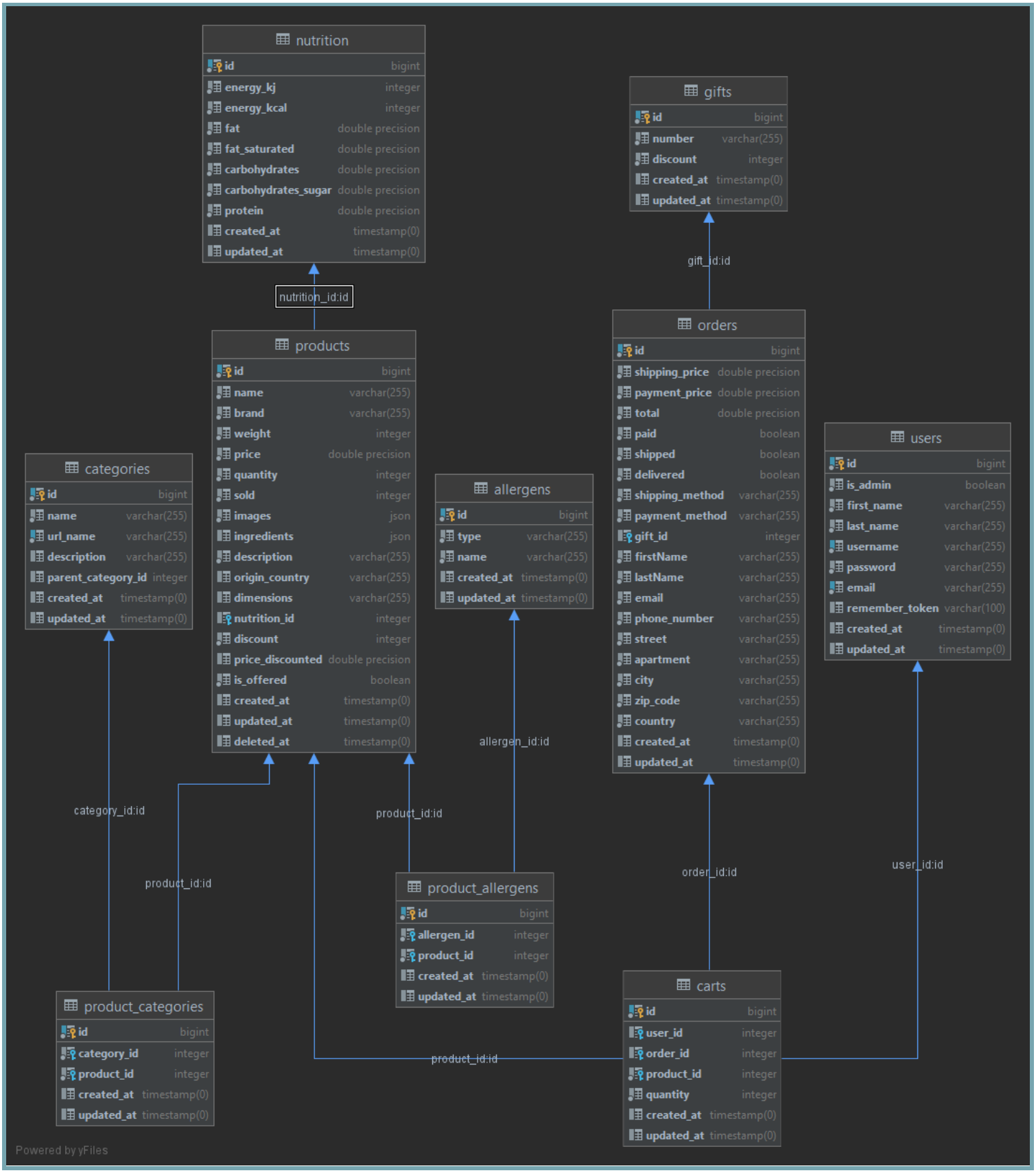
PROGRAMMING ENVIRONMENT AND LIBRARIES

In our project we used **Visual Studio Code** IDE and **pgAdmin** to work with the PostgreSQL v13 database. Also, for the better implementation of our 'OM NOM NOM' eshop we have operated with **Laravel 8** PHP web framework. Additionally we used the next libraries:

- **Bootstrap v4** (for the better and out-of-box responsive design, typography, text, forms, buttons, navigation, filters, pagination and other user interface components)
- **Jquery** (depency to Owl Carousel, price slider and some Bootstrap functionalities (e.g. modals or dropdowns) and was also used to event handling)
- **Jquery-UI** (for 2-handle point price slider)
- **Font Awesome** (for the icon insertion in our project)
- **Owl Carousel plugin** (for the improved responsive carousel slider)

PHYSICAL DATA MODEL

Our physical data model **did not change a lot** since we created the logical model. **Each table contains the main information** about the products/users/orders/categories... The main modification at this phase was **a few more tables** that we decided to add for a better **allergenes** and **nutrition** description. At the nutrition table we have made fixed columns, so that each product points at the special row in this table. By contrast, the allergen table contains names of possible allergens with their type, while the product_allergens table maps products with all these allergens that product do or may include in its containing.
In addition, we also decided to store a part of **users shipping information (address, phone number, full name…) in the orders table**, so that non-authorized users can make some orders too. Moreover, shipping details can be different for each order, so it will be better to store this type of user info in the order table and do not override it each time the information changes. Therefore the user table covers the main information only for the authentification, but not shipping.


nutrition
id bigint
energy_kj integer
energy_kcal integer
fat double precision
fat_saturated double precision
carbohydrates double precision
carbohydrates_sugar double precision
protein double precision
created_at timestamp(0)
updated_at timestamp(0)
gifts
id bigint
number varchar(255)
discount integer
created_at timestamp(0)
updated_at timestamp(0)
gift_id:id
nutrition_id:id
products
id bigint
name varchar(255)
brand varchar(255)
weight integer
price double precision
quantity integer
sold integer
images json
ingredients json
description varchar(255)
origin_country varchar(255)
dimensions varchar(255)
nutrition_id integer
discount integer
price_discounted double precision
is_offered boolean
created_at timestamp(0)
updated_at timestamp(0)
deleted_at timestamp(0)
orders
id bigint
shipping_price double precision
payment_price double precision
total double precision
paid boolean
shipped boolean
delivered boolean
shipping_method varchar(255)
payment_method varchar(255)
gift_id integer
firstName varchar(255)
lastName varchar(255)
email varchar(255)
phone_number varchar(255)
street varchar(255)
apartment varchar(255)
city varchar(255)
zip_code varchar(255)
country varchar(255)
created_at timestamp(0)
updated_at timestamp(0)
users
id bigint
is_admin boolean
first_name varchar(255)
last_name varchar(255)
username varchar(255)
password varchar(255)
email varchar(255)
remember_token varchar(100)
created_at timestamp(0)
updated_at timestamp(0)
categories
id bigint
name varchar(255)
url_name varchar(255)
description varchar(255)
parent_category_id integer
created_at timestamp(0)
updated_at timestamp(0)
allergens
id bigint
type varchar(255)
name varchar(255)
created_at timestamp(0)
updated_at timestamp(0)
allergen_id:id
category_id:id
product_id:id
product_id:id
order_id:id
user_id:id
product_allergens
id bigint
allergen_id integer
product_id integer
created_at timestamp(0)
updated_at timestamp(0)
product_categories
id bigint
category_id integer
product_id integer
created_at timestamp(0)
updated_at timestamp(0)
product_id:id
carts
id bigint
user_id integer
order_id integer
product_id integer
quantity integer
created_at timestamp(0)
updated_at timestamp(0)
Powered by yFiles

IMPLEMENTATION SOME OF THE USE CASES

- **Quantity changes of an exact product** `(CartController@update)`

Firstly, we validated the user inputs (product id & quantity). For authenticated users we retrieved chosen products from the cart in the database and then changed their quantity. In case the new quantity was zero, the record was deleted from the table. For guests (all non-authenticated users), we retrieved the product from our custom session object representing the cart, changed its quantity and recalculated the subtotal of the cart. In case the new quantity was zero, the product was removed from the session object. If there weren't any other products in the cart, the whole object cart was deleted from the user's session.

- **Login** `(AuthController@login_form)`

We validate all of the inputs from the login form (username is required and has to be alphanumeric, password is required too and min length is 6 characters). Then we try to authenticate the user using the Laravel built-in Auth facade. In case it completes successfully, the cart from the session is transferred to the database and the session ID is regenerated. Otherwise, a validation error is thrown and the user gets a message of a bad username or password entering.

- **Search** `(App\Http\Filters\SearchFilter)`

For this part we have implemented a custom class SearchFilter, which applies full text search rules from the request query on Laravel's Eloquent Query Builder. We have also added a scope 'filter' on the product model which just calls the custom filter. Whenever getting products, we just use our product model with the search filter inside the filter method `Product::filter($search)` and full text search on the brand and product name is applied.

- **Add product to the cart** `(CartController@add)`

The same as at the beginning, firstly, we validated the user inputs (product id & quantity). For all authenticated users we added the product with the corresponding quantity to the user's cart inside the database. And for all other users (guests), we created (or obtained in case it has already existed) the cart object in the session and, subsequently, added the new product into it.

- **Pagination**

For limiting the number of products shown on one page, we have used Laravel's Eloquent paginate method with 12 products per page. Moreover, as we are using Bootstrap framework, we have also applied Laravel builtin generator for the UI paginator element.

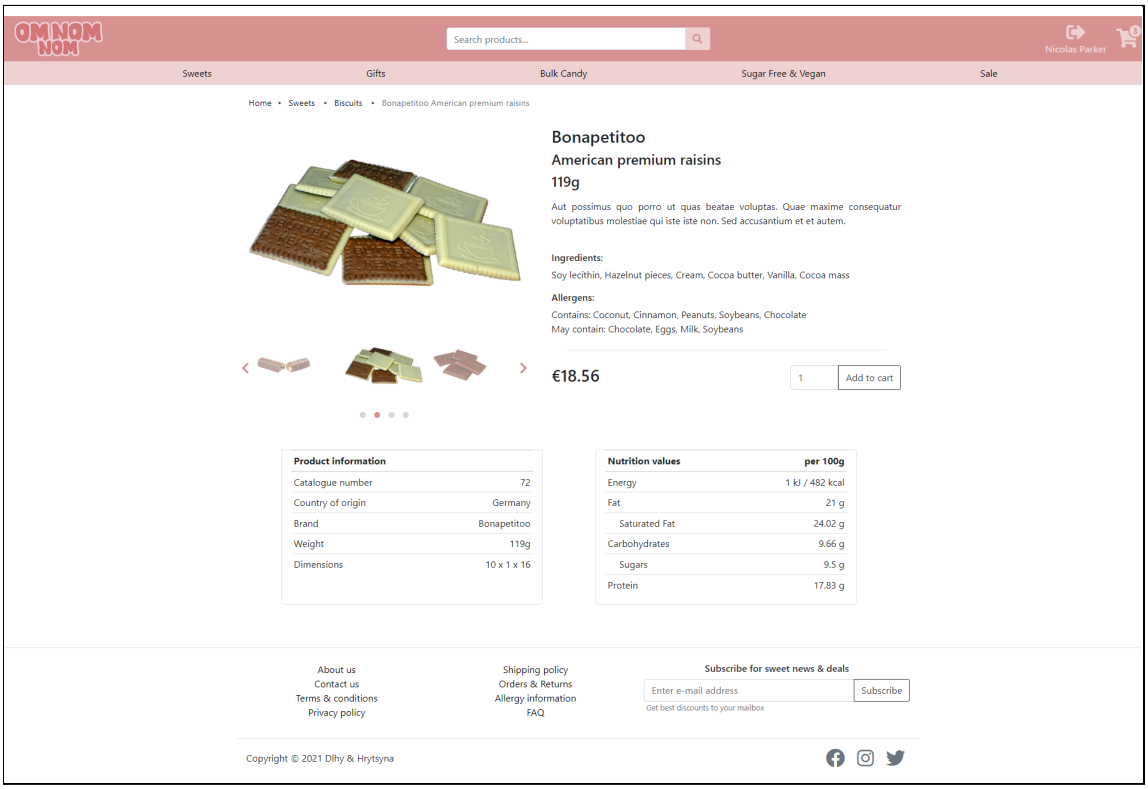
- **Filters** `(App\Http\Filters\ProductFilter)`

In our eshop we use 3 types of filters: price, country and brand. All of the parameters for each filter are values extracted from the database. Similarly to SearchFilter, we have implemented a custom class ProductFilter which applies filtering rules from the request query on Laravel's Eloquent Query Builder. Whenever getting products, we just use our product model with the product filter inside the filter method `Product::filter($filter)` and all possible (price, country, brand) filters are applied.

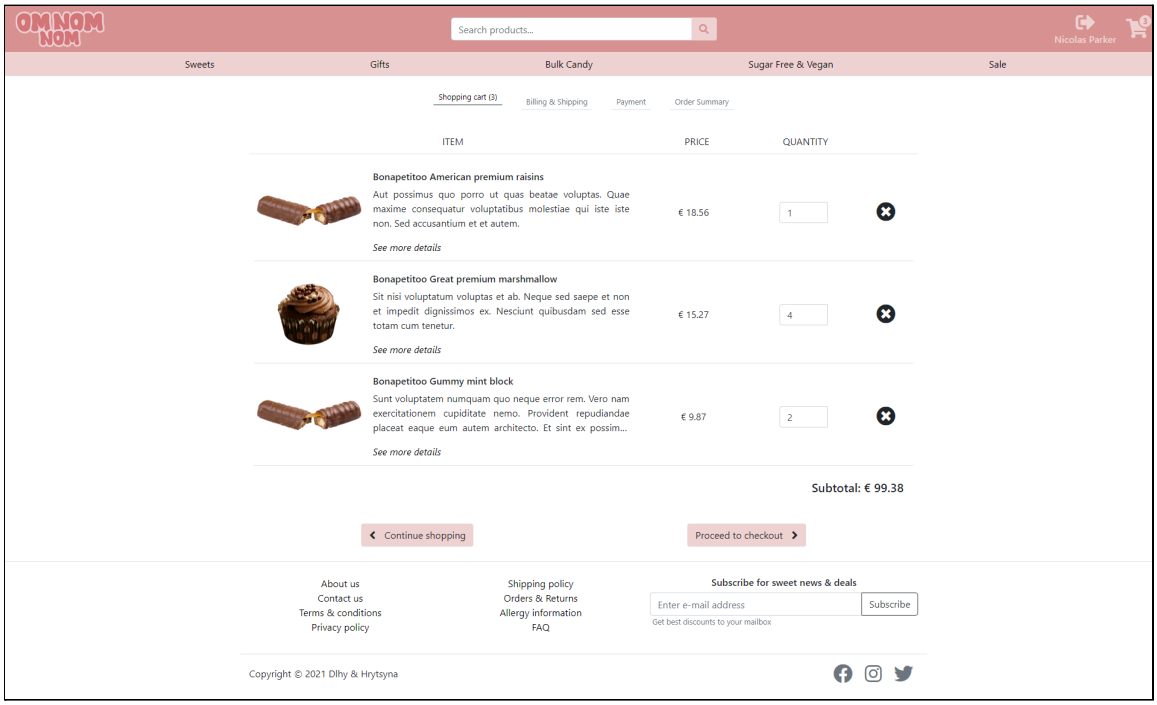
SCREENSHOTS

Also, for a better visualization we decided to add a few more screenshots of our web page.

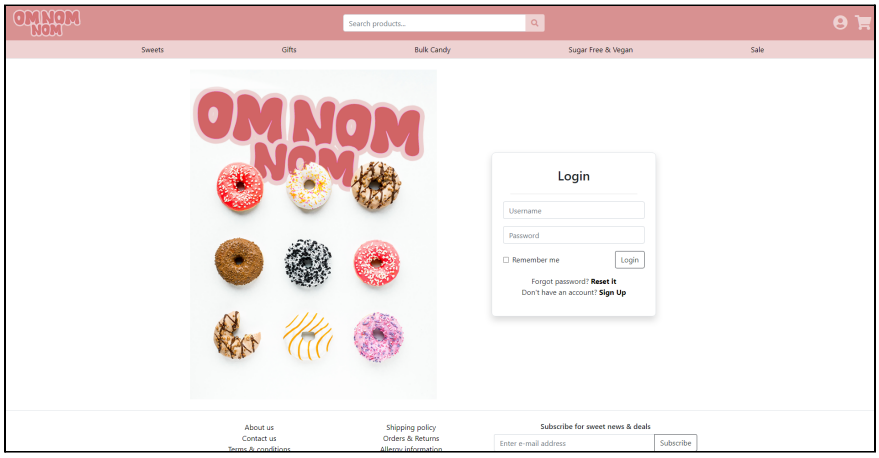
- **Product details**

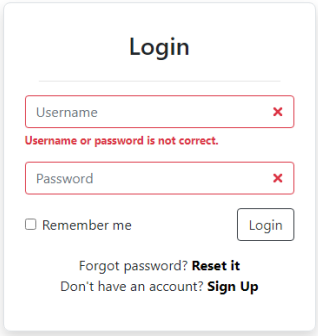


- **Cart with some products**



- **Login**





- **Homepage**