ASSIGNMENT FRONT SHEET

Qualification	**BTEC Level 4 HND Diploma in Computing**		
Unit number and title	**WEBG301. Project Web**		
Submission date	25/03/2022	**Date Received 1st submission**	
Re-submission Date		**Date Received 2nd submission**	
Group number:	**Student names & codes**	**Final scores**	**Signatures**
	1.Do Phu Cuong | GCC200122		Cuong
	2. Lam Phi | GCC200011		Phi
	3. Huynh Hoang Khang | GCC200338		Khang
Class	GCC0901	**Assessor name**	Nguyen Trung Viet

Student declaration
I certify that the assignment submission is entirely my own work and I fully understand the consequences of plagiarism. I understand that making a false declaration is a form of malpractice.

P1	P2	M1	M2	D1

OBSERVATION RECORD

Student 1	**Do Phu Cuong**		
Description of activity undertaken			
About his activity undertaken which are create entity such as Product, Category and show customer. Moreover, he created add, edit, delete and show function for his entity. He created, update database and PowerPoint for group presentation			
Assessment & grading criteria			
How the activity meets the requirements of the criteria			
Student signature:	**Cuong**	**Date:**	**25/03/2022**
Assessor signature:		**Date:**	
Assessor name:	**Nguyen Trung Viet**		

Student 2	**Lam Phi**
Description of activity undertaken	
About his activity undertaken which are make CRUD for comment, customer, order. Adding the function adding image for product, interface of homepage, create some controller in home page, creating factory, session storage, cart manager.	
Assessment & grading criteria	
How the activity meets the requirements of the criteria	

Student signature:	**Phi**	**Date:**	**25/03/2022**
Assessor signature:		**Date:**	
Assessor name:	**Nguyen Trung Viet**		

Student 3	**Huynh Hoang Khang**
Description of activity undertaken	
About his activity undertaken which are make CRUD for supplier. Login and register function, update database adjust relationship in database, edit interface in login and register, and make some controller in home page.	
Assessment & grading criteria	
How the activity meets the requirements of the criteria	

Student signature:	**Khang**	**Date:**	**25/03/2022**
Assessor signature:		**Date:**	
Assessor name:	**Nguyen Trung Viet**		

❒ Summative Feedback:		❒ Resubmission Feedback:
Grade:	Assessor Signature:	Date:
Internal Verifier's Comments:		
Signature & Date:		

Assignment Brief 1 (RQF)

Higher National Certificate/Diploma in Computing

Student Name/ID Number:	
Unit Number and Title:	
Academic Year:	
Unit Assessor:	**Hoang Nhu Vinh**
Assignment Title:	
Issue Date:	**01 April 2021**
Submission Date:	
Internal Verifier Name:	
Date:	

Submission Format:

Format: The submission is in the form of documents

- An individual evaluation document in PDF format. Writing must be professional.

Submission

- Students are compulsory to submit the assignment in due date and in a way requested by the Tutor.
- The form of submission will be a soft copy posted on http://cms.greenwich.edu.vn/.
- Remember to convert the word file into PDF file before the submission on CMS.

Note:

-

Unit Learning Outcomes:

LO1 Use MVC (Model - View - Controller) pattern with Symfony framework to develop dynamic Web Applications

LO2 Use Doctrine ORM framework to query information from database

LO3 Create Web Api services to handle users' request and return JSON responses via Postman REST client

LO4 Apply best practices such as: Dependency Injection, Git and GitHub, Web security (CSRF, DDoS attack, SQL Injection) in Web development

Assignment Brief and Guidance:

Scenario: Design and implement a Student Management/Inventory Management/Shopping cart/CMS/RPG Game/Conference Scheduler using PHP (Symfony) and HTML / CSS / JavaScript. Your project must be approved by Lecturer and meet all the requirements listed below.
The project will be implemented by a team with 3 members maximum.

General Requirements

- **Use PHP** – the major part of your work should be PHP written
    - You **must use Symfony Framework**
        - The application must have at least **10 web pages (views**)
        - The application must have at least **4 independent entity models**
        - The application must have at least **4 controllers**
    - You have to additionally use **HTML5, CSS3** to create the content and to stylize your web application
    - You may optionally use **JavaScript, jQuery, Bootstrap**
    - Use **PHP 7**

Forbidden Techniques and Tools

- Using **CMS / blog systems** (like WordPress, Drupal and Joomla) is forbidden.
- Using **Shopping cart systems** (like OpenCart) is forbidden.

Source Control

Use a **source control system** by choice, e.g., **GitHub**, **BitBucket**

- Submit a link to your public source code repository
- Each student should have at least **10 meaningful commits**
- Each student should have **commits** in at least **3 DIFFERENT** days

Tasks

Group Report

Each group will be required to deliver a Group Report, which have following items:

- Produce a set of Users' requirements by using User Story template
- Site map of the project
- Entity Relationship Diagram (ERD)
- Final Result of the project with evidences
  + GitHub repository evidences
  + Sample Source code with brief explanation
  Images of final application
- Conclusion and Lessons Learned
- The Document should be less than 20 pages

Public Project Defense

Each group will have to deliver a public defense of its work in front of the Lecturer.
Each group will have only 20 minutes for the following:

- Each student demonstrates his/her part in the project
- Each students shows the source code and explain how it works
- Answer questions related to the project (and best practices in general)

Please be strict in timing!
Be well prepared for presenting maximum of your work for a minimum time. Bring your OWN LAPTOP. Test it preliminarily with the multimedia projector. Open the project assets beforehand to save time.

Learning Outcomes and Assessment Criteria (Assignment):			
Assessment Criteria			
Group Report – (30% of final mark) • Produce a set of Users’ requirements by using User Story template – **0.5 points** • Site map of the project – **0.5 points** • Entity Relationship Diagram (ERD) – **1 points** • Final Result of the project with evidence – **1 points** **Individual Assessment – (70% of final mark)** • Functionality and Presentation – **2 points** • Answering correctly technical questions asked by Lecturer – **3 points** • Each student has at least 10 meaningful commits in 3 different days – **1 points** • Implementing views and controllers correctly (controllers should do only their work, using display and editor templates) **– 1 points**			
Assessment Range			
Learning Outcome	Pass (5 – 6.5 points)	Merit (7 – 8.5 points)	Distinction (9 – 10 points)
LO1 - LO2 - LO3 - LO4	P1 Produce a set of Users’ requirements by using User Story template. Tables must have relationships with each other. Site map of the project.	**M1** Can perform Register / Login	**D1** Implement API and consume it with JavaScript. Create Web Api services to handle users' request and return JSON responses via Postman REST client
	P2 Can perform CRUD actions. Use Doctrine ORM framework to query information from database. Final Result of the project with evidence	**M2** Can Perform Authorization	

Contents

1. User' requirements

1.1. Project specification

We create website for selling coffee and dessert. On this website admin can create, update, view, delete on product, category, supplier, customer. Customer can register account, login, and logout, they can view cart before order items. The website should be working well on Chrome, Safari, Firefox, and so on. The main user it is customer that visit our website and admin.

- Input data from Users
  - Information of users: username, password, roles
  - Information of order: status, created_at, updated_at
  - Information of order_item: product id, order_ref_id, date, deliverydate, quatity, customer_id
  - Information of comments: customer_id, date, description
  - Customer information: name, gmail, address, telephone, birth,
- Output data for Users
  - List of items in shop
  - Information of the website such as about us, event, policy, and so on
- Input data for admin
  - Information of category: name, description
  - Information of supplier: name, address, email, telephone
  - Information of product: supplier id, category id, name, date, stock, supplierprice, exportprice, description, image
- Output data for admin
  - List of category
  - List of product
  - List of supplier
  - List of customer
- Function for users
  - Save, clear, remove in shopping cart
  - View details of product
- Function for admin
  - Admin can add, update, remove, view category
  - Admin can add, update, remove, view product
  - Admin can add, update, remove, view customer
  - Admin can add, update, remove, view supplier

2. System design

2.1. Symfony

According to (Symfony, n.d) Symfony is a collection of PHP components, a web application framework, a philosophy, and a community that all work in tandem. On October 18, 2005, it was released as open-source software under the MIT license. Symfony community have a group of over 600,000 developers from more than 120 countries, all committed to helping PHP surpass the impossible.

According to (Chamat, 2020) benefits of Symfony

- **Time-saving capabilities**: Symfony reduces time-to-market by speeding up the development process. It has a number of built-in features that all help to speed up the app development process.
- **Flexibility**: Whether you're just starting out in development or have a lot of experience, Symfony is a popular choice. It gives you complete control over configuration and other key development features, and it works with a variety of database systems. Symfony checks all the boxes, from the simplest to the most complex applications. It is extremely adaptable, and additional functionalities can be added with ease, ensuring that the users' specific requirements are met.
- **Ease of use**: Symfony is a flexible framework that is also simple to use. Beginners who are new to the development community benefit from best practices that are built into the process to make it easier for them to learn quickly. Even if you are an experienced developer, there is in-depth documentation that is easy to access and full of useful information. Everything is explained in detail with easy-to-follow examples. Developers can easily create web applications, and the platform includes a variety of command tools to aid in the project management phase.
- **Extensive community support**: Symfony also has a fantastic community of 45 partners (supported by SensioLabs) where users can learn about the latest web development tools and methods. This thriving community of web developers has over 13 years of experience, allowing users to benefit from long-term support and scalability. This means that if you use Symfony, you can expect long-term stability, thorough testing, frequent updates, and improved application performance.
- **Full customizability**:
  - **Full stack**: If you want to design a sophisticated product with several functions, this is the way to go.
  - **Brick by brick**: It entails developing a tailored application with restricted chosen functions based on specific aspects depending on your particular requirements – but not the full framework.
  - **Micro framework**: Utilize certain characteristics to create a custom functionality comprised of modular bricks that may be utilized independently.

- **Easy testing**: Unit testing has never been easier thanks to the PHP Unit Independent Library. This stimulates HTTP requests, allowing testing tools to check the result without the use of a script. Functional testing is also automated, which saves time and effort for developers. Finally, several excellent tools for functional, behavioral, and unit testing are available.
- **Ultimate convenience**: This high-end programming framework provides developers with comfort and ease. There are tools for resolving code problems and security issues, and product development is frictionless and extremely customer-centric

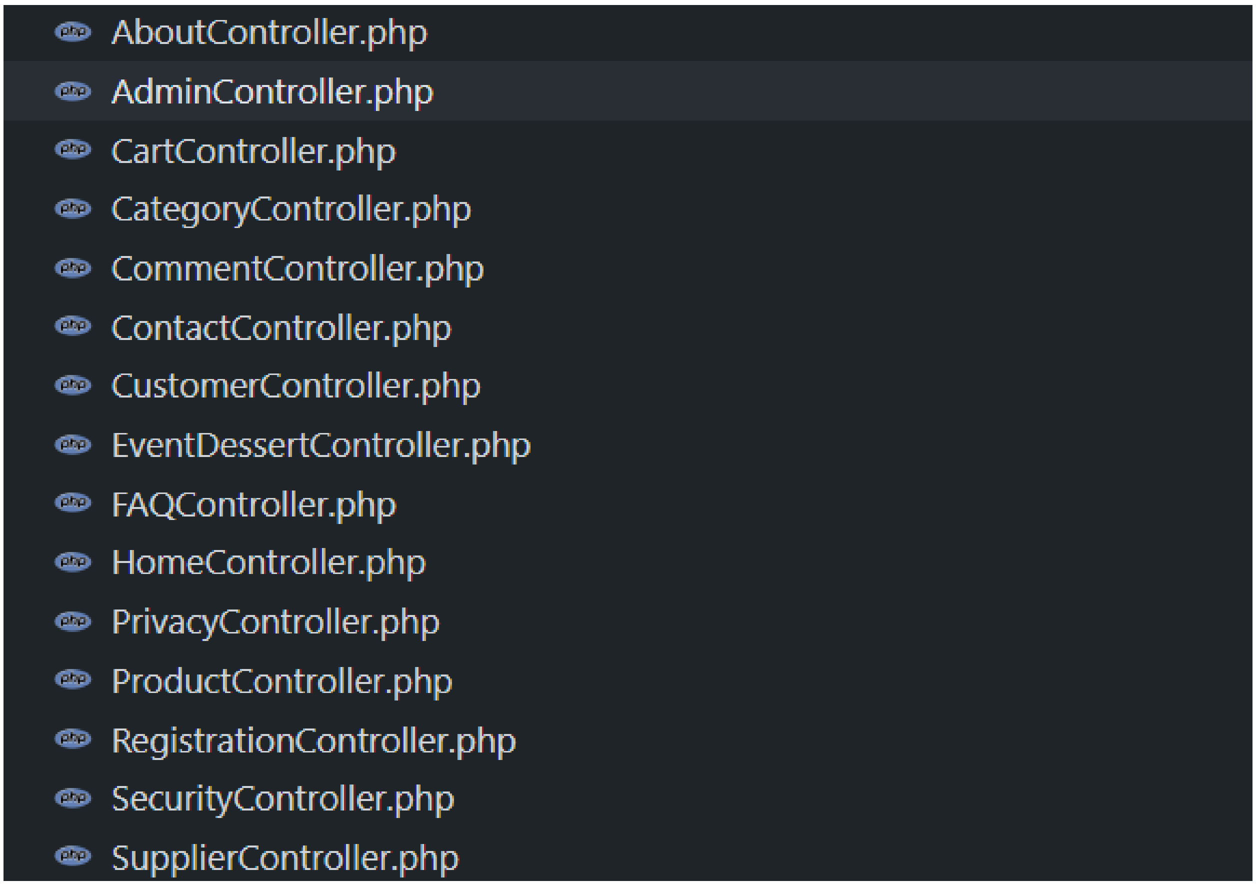
2.2. Controller entity

Controller we have about, admin, home, category and so on. It uses to route the link for website.

We create Controller in symfony by "php bin/console make:Controller" such as

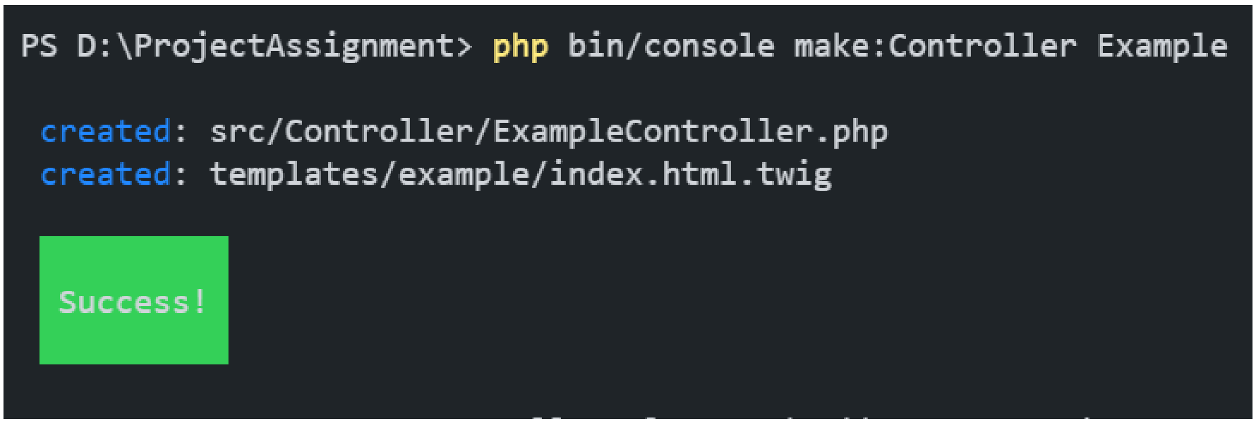
-php bin/console make:Controller Product

-php bin/console make:Controller Supplier



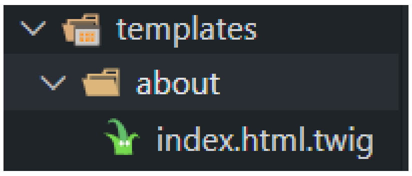
```
AboutController.php
AdminController.php
CartController.php
CategoryController.php
CommentController.php
ContactController.php
CustomerController.php
EventDessertController.php
FAQController.php
HomeController.php
PrivacyController.php
ProductController.php
RegistrationController.php
SecurityController.php
SupplierController.php
```

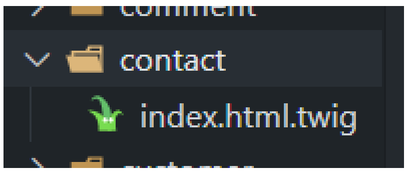
-execute:



```
PS D:\ProjectAssignment> php bin/console make:Controller Example

 created: src/Controller/ExampleController.php
 created: templates/example/index.html.twig

 Success!
```

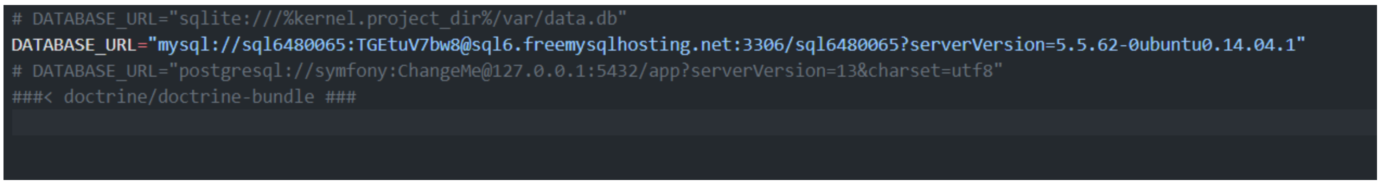
After we create a controller, inside the template will automatically create a folder with the same name, inside including index.html.twig For example, we create AboutController and ContactController, inside template there are 2 folder about and contact, and both includes index.html.twig inside it


```
templates
  about
    index.html.twig
```


```
contact
  index.html.twig
```

At first, we will create a project with a symfony command: symfony new –full ProjectAssignment –version=5.4. *

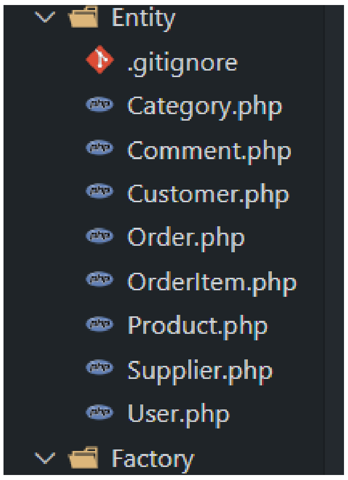
After, we have created a project, we will change. env in my project with the purpose is to connect to the database of phpmyadmin.co. With the account have created before


```
# DATABASE_URL="sqlite:///%kernel.project_dir%/var/data.db"
DATABASE_URL="mysql://sql6480065:TGEtuV7bw8@sql6.freemysqlhosting.net:3306/sql6480065?serverVersion=5.5.62-0ubuntu0.14.04.1"
# DATABASE_URL="postgresql://symfony:ChangeMe@127.0.0.1:5432/app?serverVersion=13&charset=utf8"
###< doctrine/doctrine-bundle ###
```

Next, using symfony command to create database: php bin/console doctrine:database:create

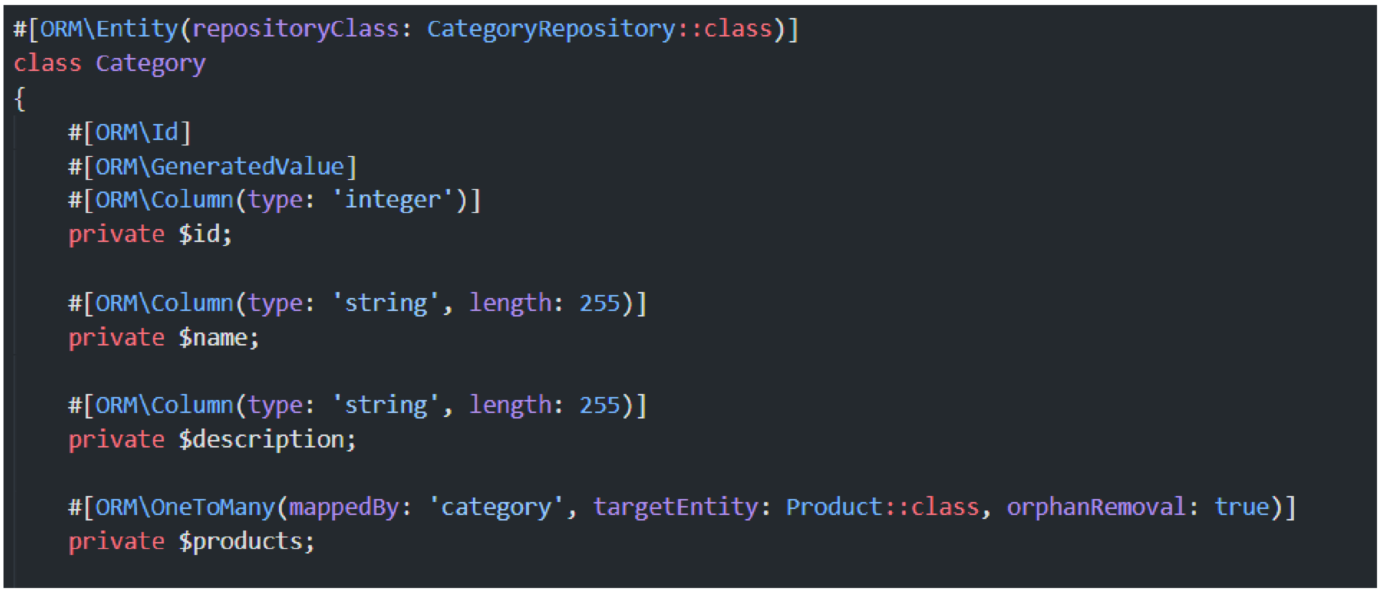
Then, we continue to create entities such as Category, Comment, Customer, Order, OrderItem, Product, Supplier, and User through a command: php bin\console make:entity [Entityname]

From that, we have entities:


```
Entity
  .gitignore
  Category.php
  Comment.php
  Customer.php
  Order.php
  OrderItem.php
  Product.php
  Supplier.php
  User.php
Factory
```

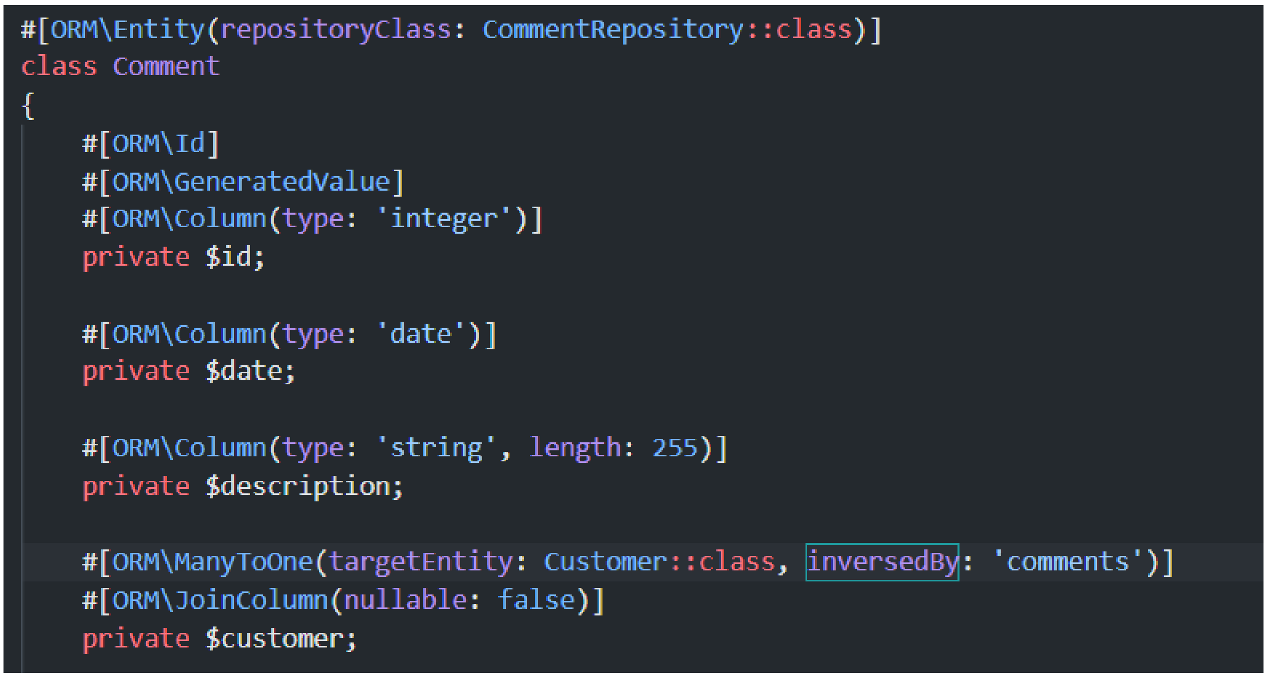
The source code of each Entity:

- The source code of Category Entity: (it includes properties)



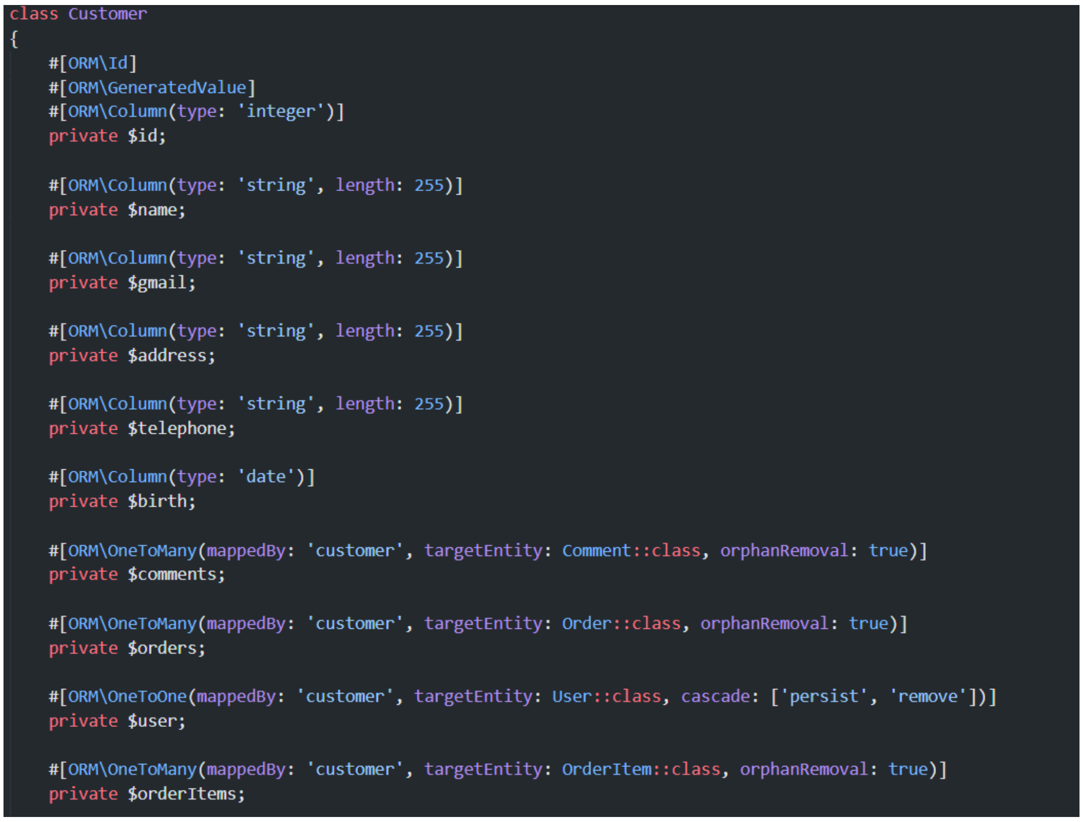
```php
#[ORM\Entity(repositoryClass: CategoryRepository::class)]
class Category
{
    #[ORM\Id]
    #[ORM\GeneratedValue]
    #[ORM\Column(type: 'integer')]
    private $id;

    #[ORM\Column(type: 'string', length: 255)]
    private $name;

    #[ORM\Column(type: 'string', length: 255)]
    private $description;

    #[ORM\OneToMany(mappedBy: 'category', targetEntity: Product::class, orphanRemoval: true)]
    private $products;
```

- The source code of Comment Entity: (it includes properties)



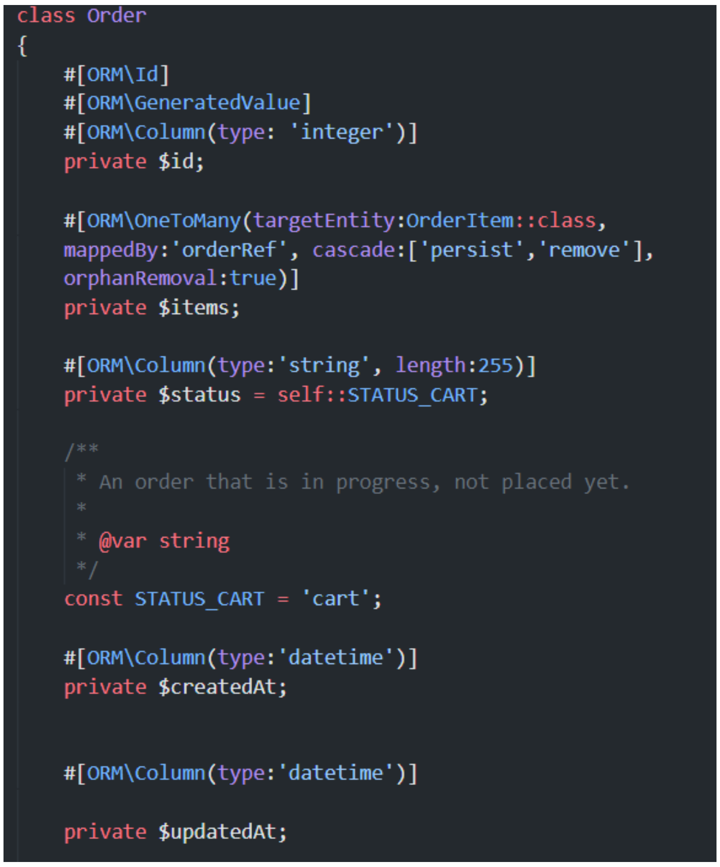
```php
#[ORM\Entity(repositoryClass: CommentRepository::class)]
class Comment
{
    #[ORM\Id]
    #[ORM\GeneratedValue]
    #[ORM\Column(type: 'integer')]
    private $id;

    #[ORM\Column(type: 'date')]
    private $date;

    #[ORM\Column(type: 'string', length: 255)]
    private $description;

    #[ORM\ManyToOne(targetEntity: Customer::class, inversedBy: 'comments')]
    #[ORM\JoinColumn(nullable: false)]
    private $customer;
```

- The source code of Customer Entity: (it includes properties)



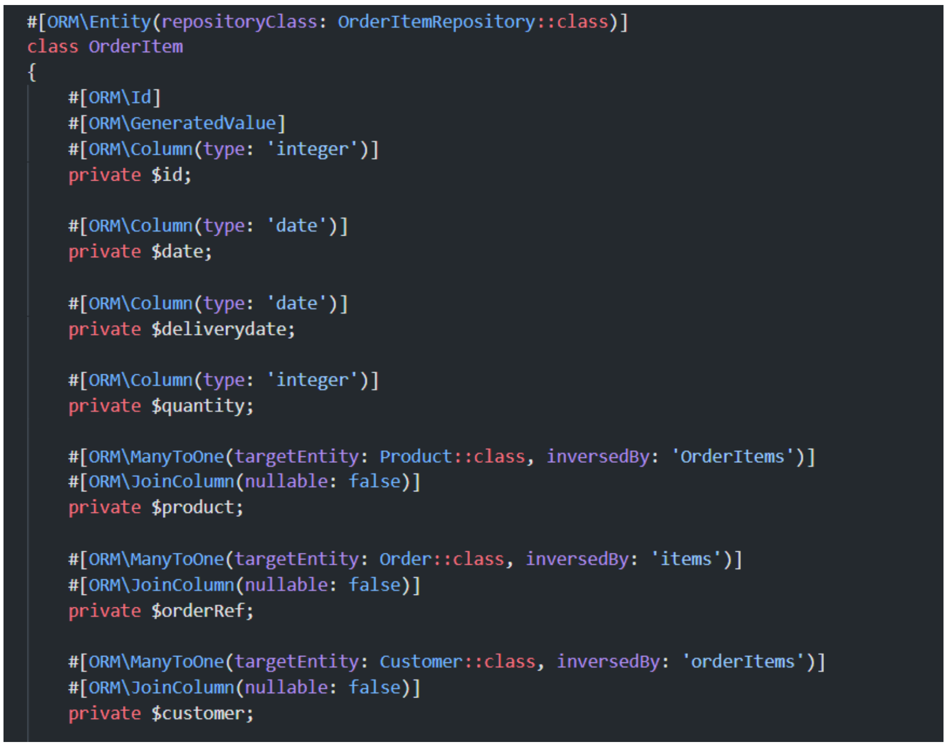
```
class Customer
{
    #[ORM\Id]
    #[ORM\GeneratedValue]
    #[ORM\Column(type: 'integer')]
    private $id;

    #[ORM\Column(type: 'string', length: 255)]
    private $name;

    #[ORM\Column(type: 'string', length: 255)]
    private $gmail;

    #[ORM\Column(type: 'string', length: 255)]
    private $address;

    #[ORM\Column(type: 'string', length: 255)]
    private $telephone;

    #[ORM\Column(type: 'date')]
    private $birth;

    #[ORM\OneToMany(mappedBy: 'customer', targetEntity: Comment::class, orphanRemoval: true)]
    private $comments;

    #[ORM\OneToMany(mappedBy: 'customer', targetEntity: Order::class, orphanRemoval: true)]
    private $orders;

    #[ORM\OneToOne(mappedBy: 'customer', targetEntity: User::class, cascade: ['persist', 'remove'])]
    private $user;

    #[ORM\OneToMany(mappedBy: 'customer', targetEntity: OrderItem::class, orphanRemoval: true)]
    private $orderItems;
```

- Order Entity:

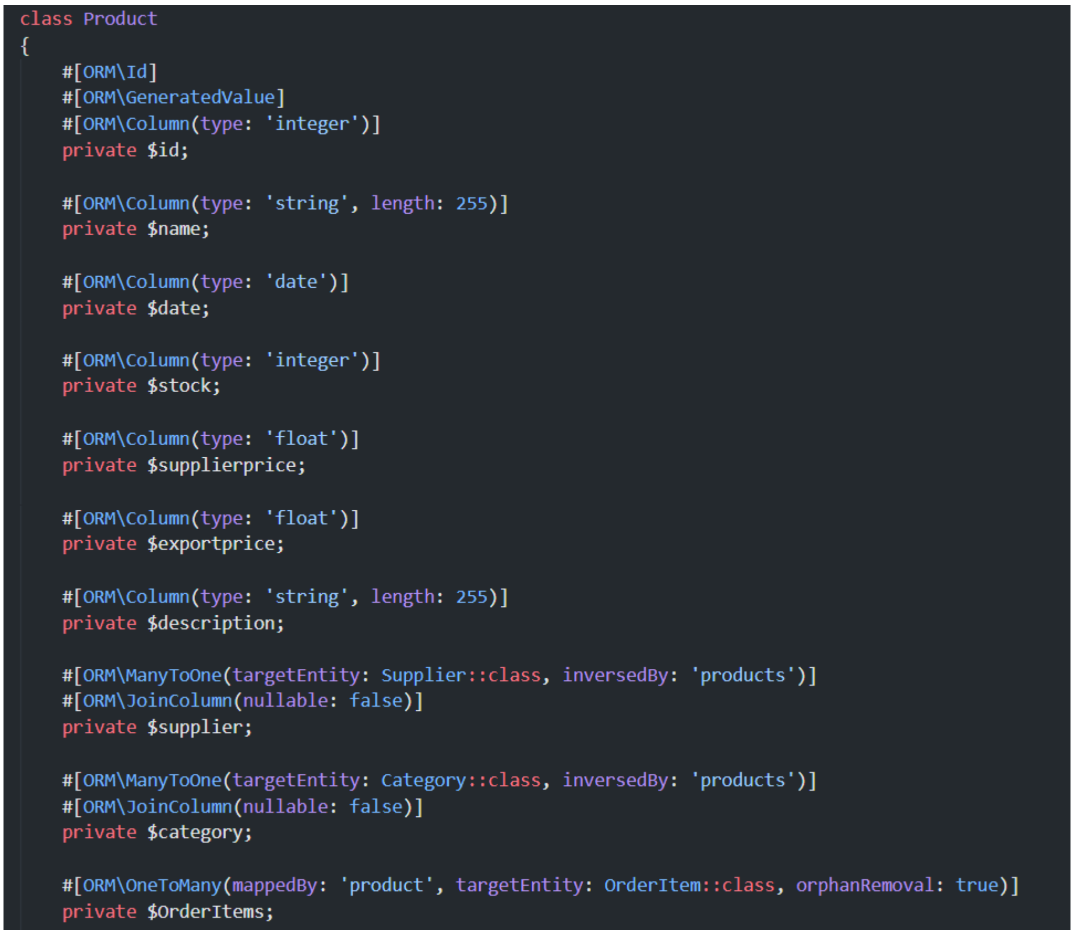


```
class Order
{
    #[ORM\Id]
    #[ORM\GeneratedValue]
    #[ORM\Column(type: 'integer')]
    private $id;

    #[ORM\OneToMany(targetEntity:OrderItem::class,
    mappedBy:'orderRef', cascade:['persist','remove'],
    orphanRemoval:true)]
    private $items;

    #[ORM\Column(type:'string', length:255)]
    private $status = self::STATUS_CART;

    /**
     * An order that is in progress, not placed yet.
     *
     * @var string
     */
    const STATUS_CART = 'cart';

    #[ORM\Column(type:'datetime')]
    private $createdAt;


    #[ORM\Column(type:'datetime')]

    private $updatedAt;
```

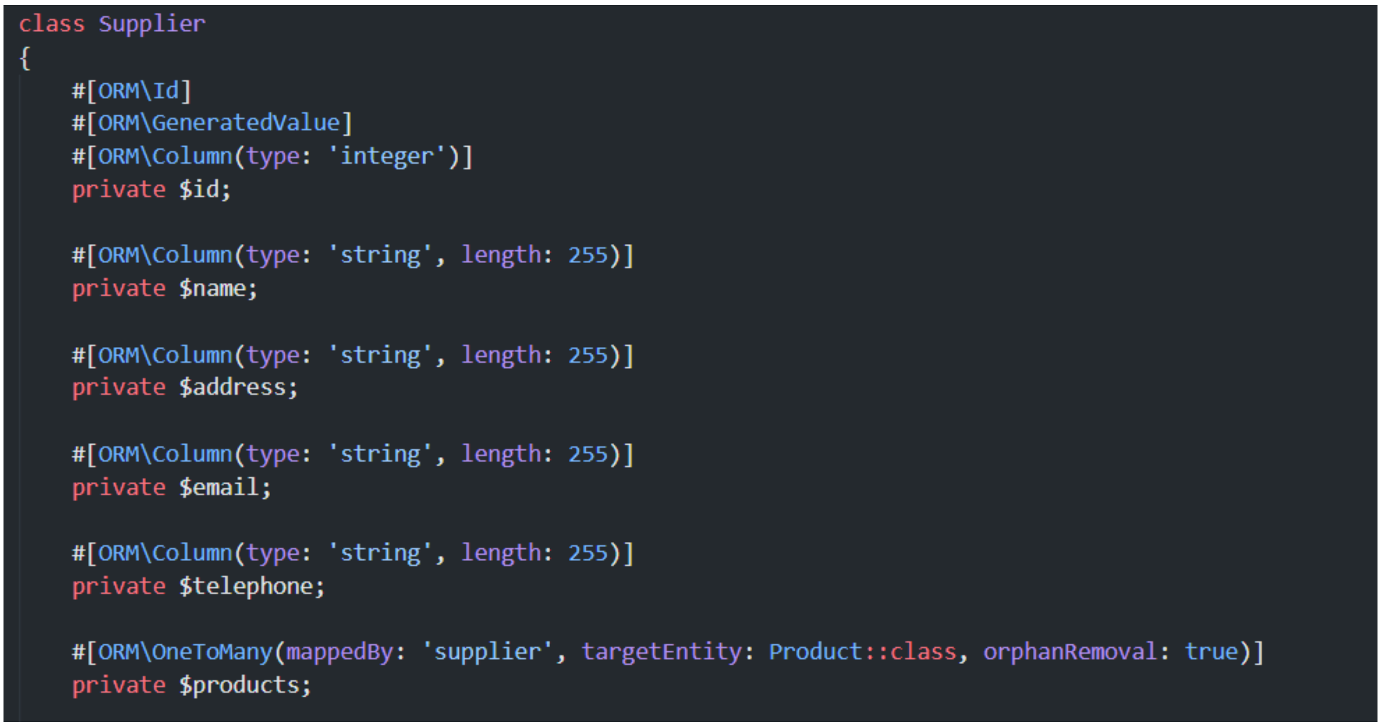
- OrderItem entity:


```
#[ORM\Entity(repositoryClass: OrderItemRepository::class)]
class OrderItem
{
    #[ORM\Id]
    #[ORM\GeneratedValue]
    #[ORM\Column(type: 'integer')]
    private $id;

    #[ORM\Column(type: 'date')]
    private $date;

    #[ORM\Column(type: 'date')]
    private $deliverydate;

    #[ORM\Column(type: 'integer')]
    private $quantity;

    #[ORM\ManyToOne(targetEntity: Product::class, inversedBy: 'OrderItems')]
    #[ORM\JoinColumn(nullable: false)]
    private $product;

    #[ORM\ManyToOne(targetEntity: Order::class, inversedBy: 'items')]
    #[ORM\JoinColumn(nullable: false)]
    private $orderRef;

    #[ORM\ManyToOne(targetEntity: Customer::class, inversedBy: 'orderItems')]
    #[ORM\JoinColumn(nullable: false)]
    private $customer;
```

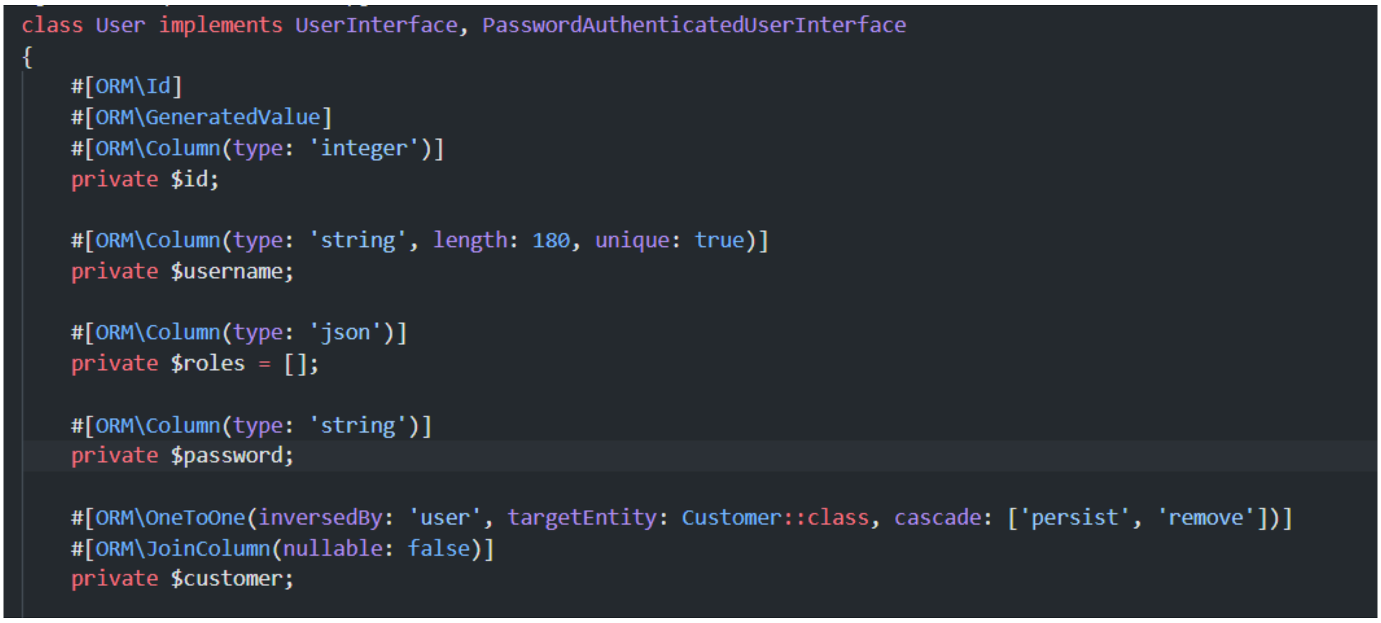
- Product Entity:


```
class Product
{
    #[ORM\Id]
    #[ORM\GeneratedValue]
    #[ORM\Column(type: 'integer')]
    private $id;

    #[ORM\Column(type: 'string', length: 255)]
    private $name;

    #[ORM\Column(type: 'date')]
    private $date;

    #[ORM\Column(type: 'integer')]
    private $stock;

    #[ORM\Column(type: 'float')]
    private $supplierprice;

    #[ORM\Column(type: 'float')]
    private $exportprice;

    #[ORM\Column(type: 'string', length: 255)]
    private $description;

    #[ORM\ManyToOne(targetEntity: Supplier::class, inversedBy: 'products')]
    #[ORM\JoinColumn(nullable: false)]
    private $supplier;

    #[ORM\ManyToOne(targetEntity: Category::class, inversedBy: 'products')]
    #[ORM\JoinColumn(nullable: false)]
    private $category;

    #[ORM\OneToMany(mappedBy: 'product', targetEntity: OrderItem::class, orphanRemoval: true)]
    private $OrderItems;
```

- Supplier Entity:

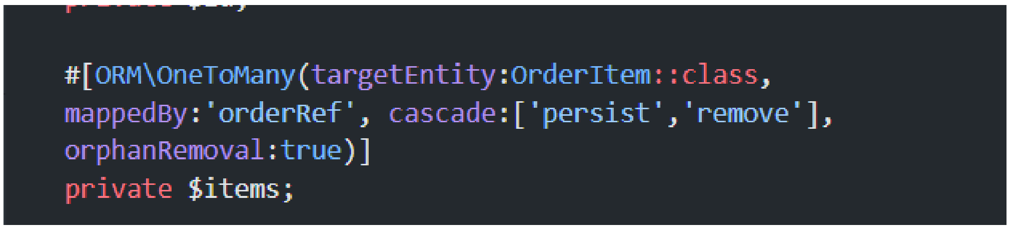

```
class Supplier
{
    #[ORM\Id]
    #[ORM\GeneratedValue]
    #[ORM\Column(type: 'integer')]
    private $id;

    #[ORM\Column(type: 'string', length: 255)]
    private $name;

    #[ORM\Column(type: 'string', length: 255)]
    private $address;

    #[ORM\Column(type: 'string', length: 255)]
    private $email;

    #[ORM\Column(type: 'string', length: 255)]
    private $telephone;

    #[ORM\OneToMany(mappedBy: 'supplier', targetEntity: Product::class, orphanRemoval: true)]
    private $products;
```

- User entity:


```
class User implements UserInterface, PasswordAuthenticatedUserInterface
{
    #[ORM\Id]
    #[ORM\GeneratedValue]
    #[ORM\Column(type: 'integer')]
    private $id;

    #[ORM\Column(type: 'string', length: 180, unique: true)]
    private $username;

    #[ORM\Column(type: 'json')]
    private $roles = [];

    #[ORM\Column(type: 'string')]
    private $password;

    #[ORM\OneToOne(inversedBy: 'user', targetEntity: Customer::class, cascade: ['persist', 'remove'])]
    #[ORM\JoinColumn(nullable: false)]
    private $customer;
```

- After creating 8 Entity, we will use a command php bin/console make:migration to create a folder migration and use a command php bin/console doctrine:migrations:migrate created in the database

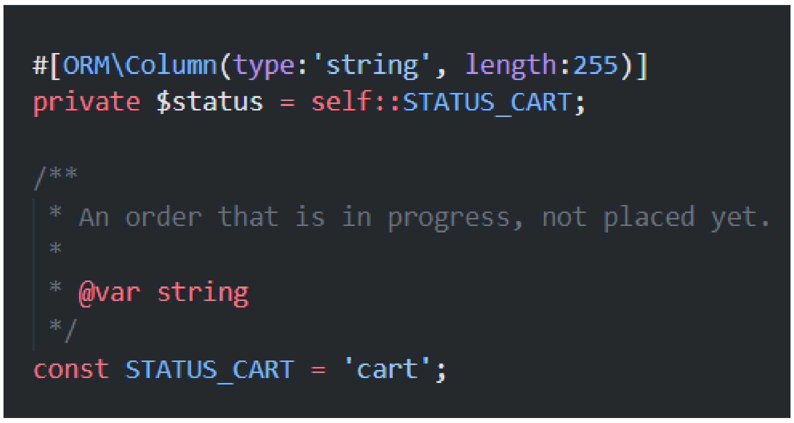
To execute a shopping cart in the website, firstly, It is can be seen that when we add a new OrderItem entity to an Order entity, we should persist it before persisting the Order entity. In addition, when we will want to remove an Order entity, we should remove all OrderItem entities before removing the Order entity. It can be really boring, so we will let Doctrine take care of this internally by using the cascade operations. To do that, add a new cascade option on the items property of the Order entity.



```
    #[ORM\OneToMany(targetEntity:OrderItem::class,
    mappedBy:'orderRef', cascade:['persist','remove'],
    orphanRemoval:true)]
    private $items;
```

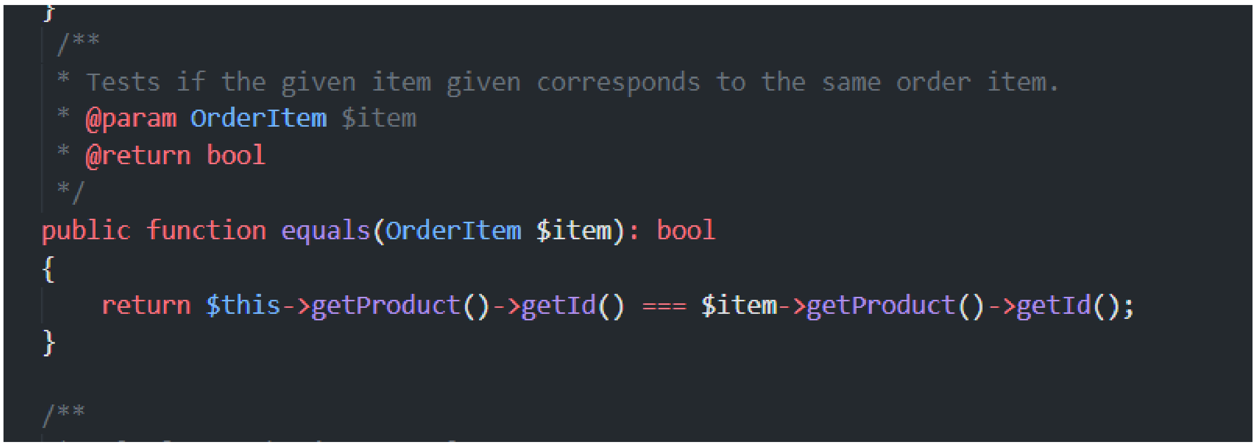
Therefore, when we will persist/remove an Order entity, it is can be seen that Doctrine will automatically call persist/remove on each of the OrderItem objects linked with the Order entity.

Secondly, all new orders of customers must be in the default cart state. Therefore, In the Order entity that created, initialize the status using a STATUS_CART constant:



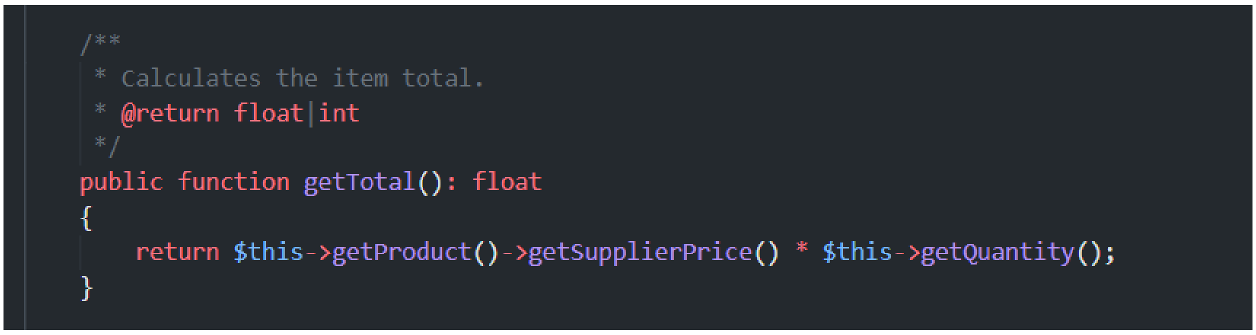
```
#[ORM\Column(type:'string', length:255)]
private $status = self::STATUS_CART;

/**
 * An order that is in progress, not placed yet.
 *
 * @var string
 */
const STATUS_CART = 'cart';
```

Duplicate OrderItem entities (with the same Product) can now be added to the Order entity. Let's see what we can do about that. To know if the item corresponds to an item given as an argument, add an equals() method to the OrderItem class.



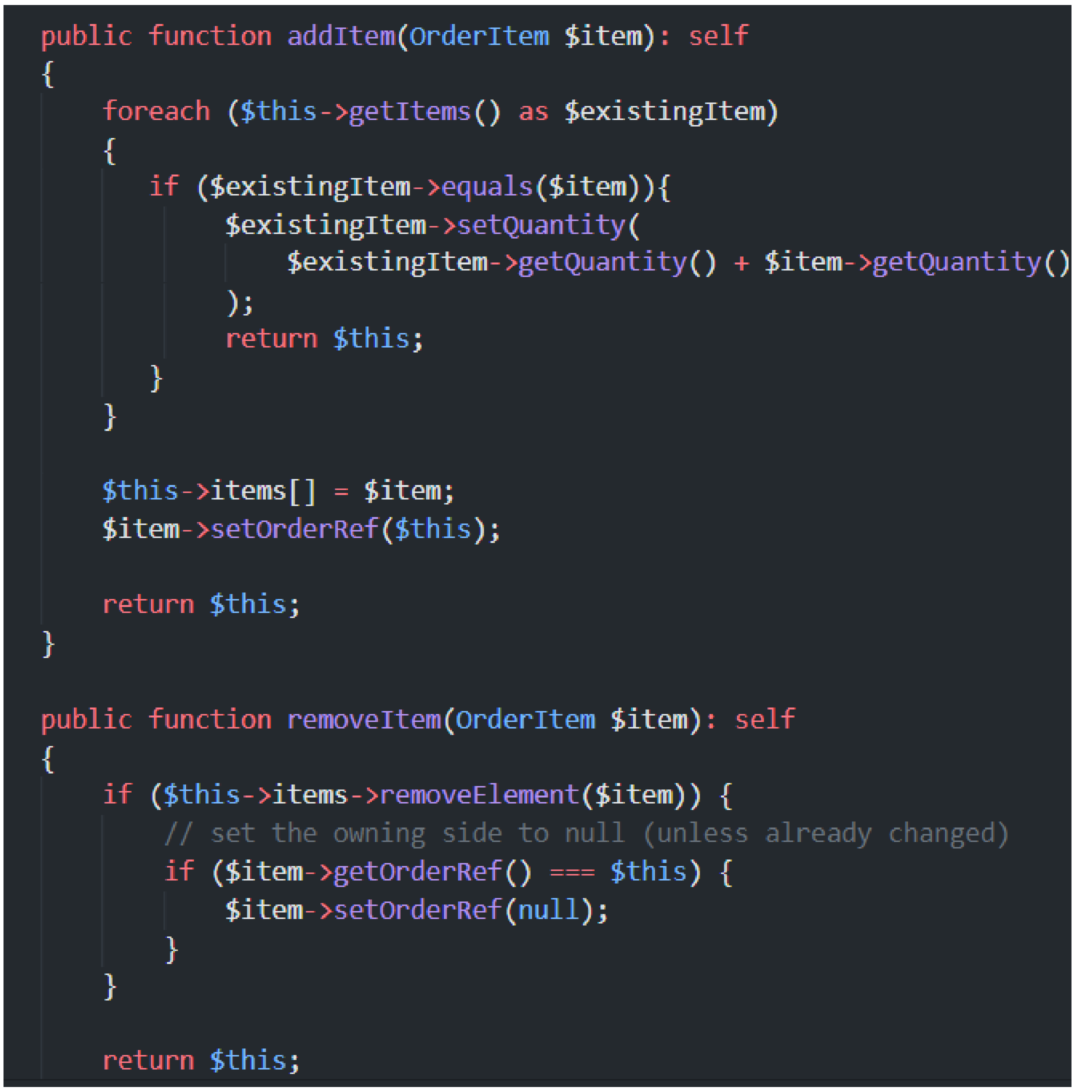
```
    /**
     * Tests if the given item given corresponds to the same order item.
     * @param OrderItem $item
     * @return bool
     */
public function equals(OrderItem $item): bool
{
    return $this->getProduct()->getId() === $item->getProduct()->getId();
}

/**
```

Next, the cart summary will have to be shown. It is can be seen that in the cart, we ignore adjustments that order could have such as shipping cost, promo code, and others, To calculate the item total, first add a getTotal() function to the OrderItem entity:

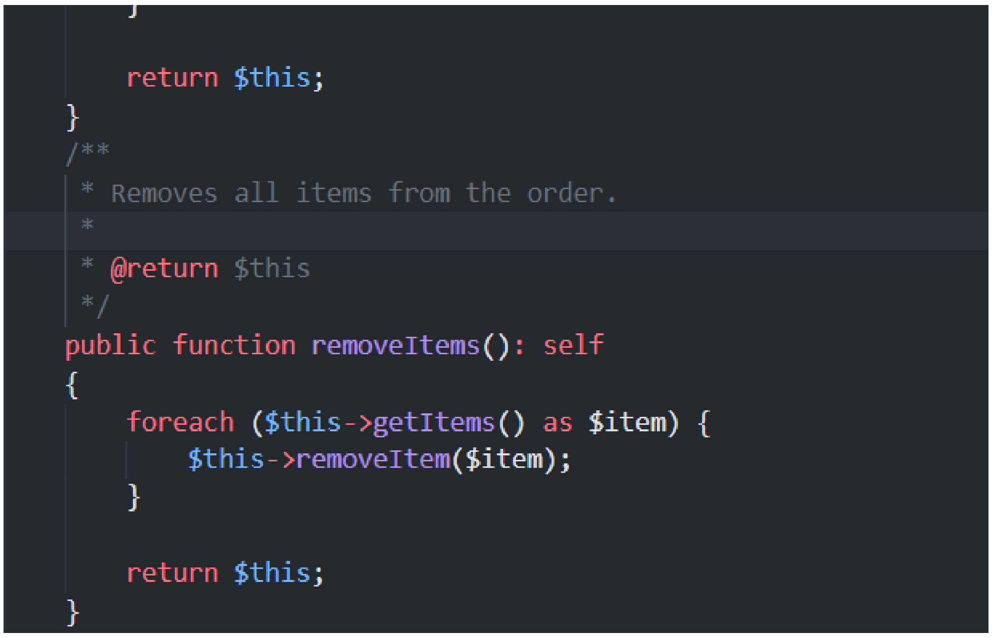


```
/**
 * Calculates the item total.
 * @return float|int
 */
public function getTotal(): float
{
    return $this->getProduct()->getSupplierPrice() * $this->getQuantity();
}
```

Then we will update the addItem function and removeItem function and removeItems function

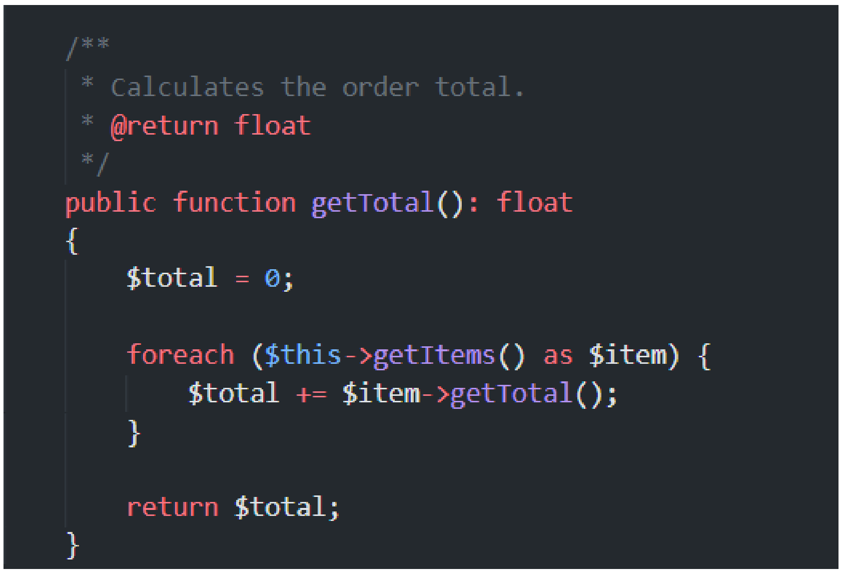


```
public function addItem(OrderItem $item): self
{
    foreach ($this->getItems() as $existingItem)
    {
       if ($existingItem->equals($item)){
            $existingItem->setQuantity(
                $existingItem->getQuantity() + $item->getQuantity()
            );
            return $this;
       }
    }

    $this->items[] = $item;
    $item->setOrderRef($this);

    return $this;
}

public function removeItem(OrderItem $item): self
{
    if ($this->items->removeElement($item)) {
        // set the owning side to null (unless already changed)
        if ($item->getOrderRef() === $this) {
            $item->setOrderRef(null);
        }
    }

    return $this;
```



```
        }

        return $this;
    }
    /**
     * Removes all items from the order.
     *
     * @return $this
     */
    public function removeItems(): self
    {
        foreach ($this->getItems() as $item) {
            $this->removeItem($item);
        }

        return $this;
    }
```

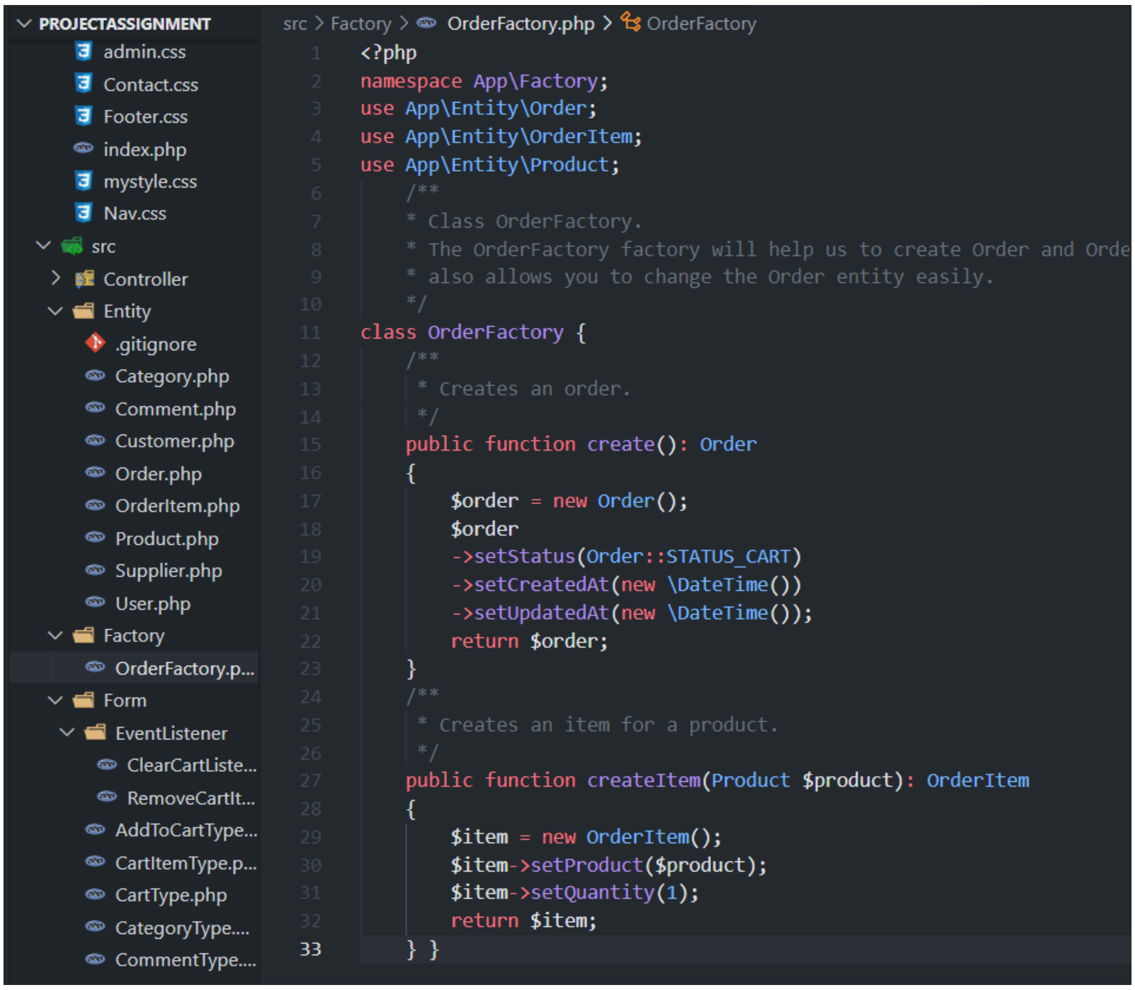
Finally, add a getTotal() Function to the Order entity to calculate the order total:



```
    /**
     * Calculates the order total.
     * @return float
     */
    public function getTotal(): float
    {
        $total = 0;

        foreach ($this->getItems() as $item) {
            $total += $item->getTotal();
        }

        return $total;
    }
```

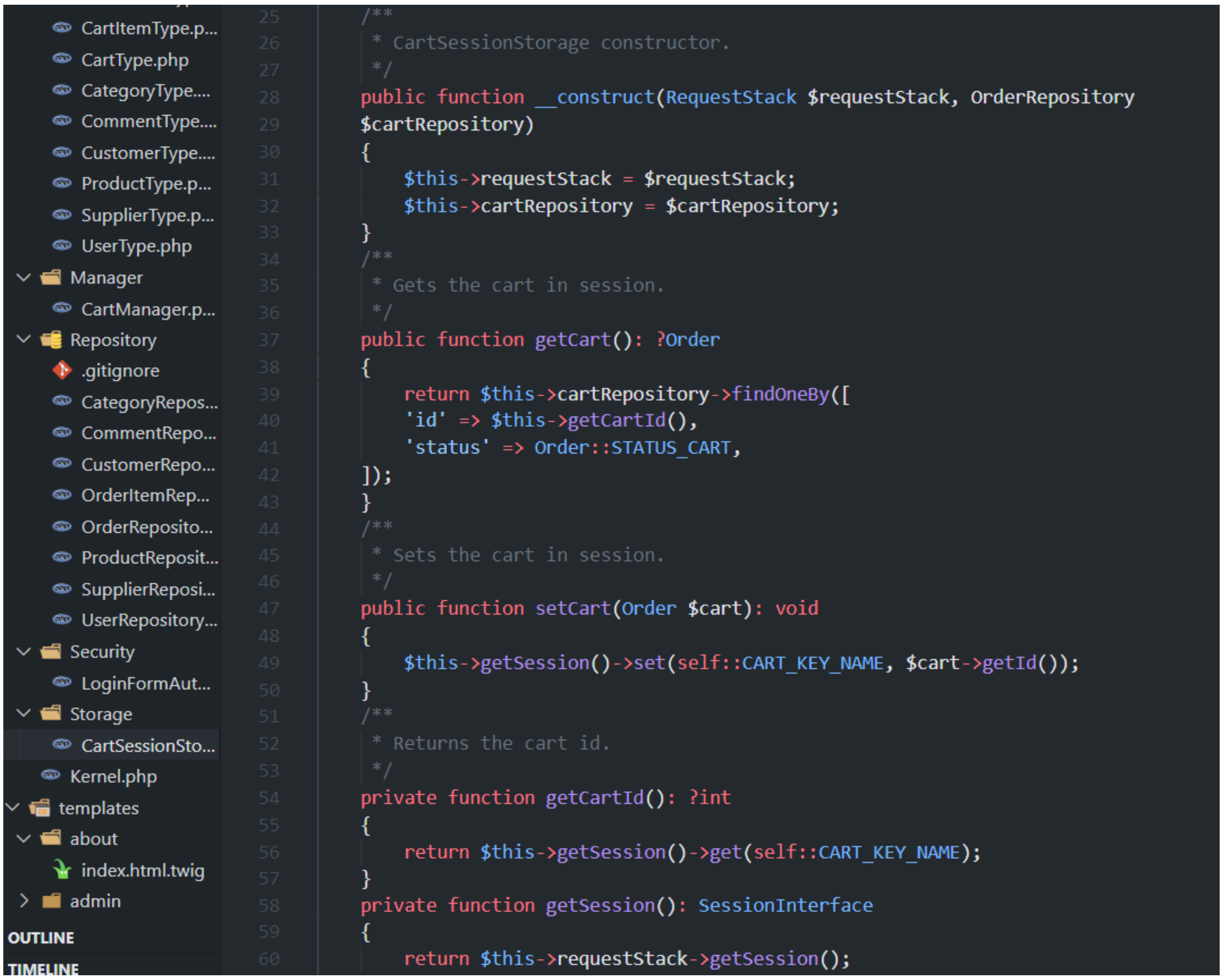
The OrderFactory factory will assist us in generating default data for Order and OrderItem entities. It also makes it simple to alter the Order entity.

Create the Factory folder and the OrderFactory.php file.



```php
<?php
namespace App\Factory;
use App\Entity\Order;
use App\Entity\OrderItem;
use App\Entity\Product;
    /**
    * Class OrderFactory.
    * The OrderFactory factory will help us to create Order and Orde
    * also allows you to change the Order entity easily.
    */
class OrderFactory {
    /**
     * Creates an order.
     */
    public function create(): Order
    {
        $order = new Order();
        $order
        ->setStatus(Order::STATUS_CART)
        ->setCreatedAt(new \DateTime())
        ->setUpdatedAt(new \DateTime());
        return $order;
    }
    /**
     * Creates an item for a product.
     */
    public function createItem(Product $product): OrderItem
    {
        $item = new OrderItem();
        $item->setProduct($product);
        $item->setQuantity(1);
        return $item;
    } }
```

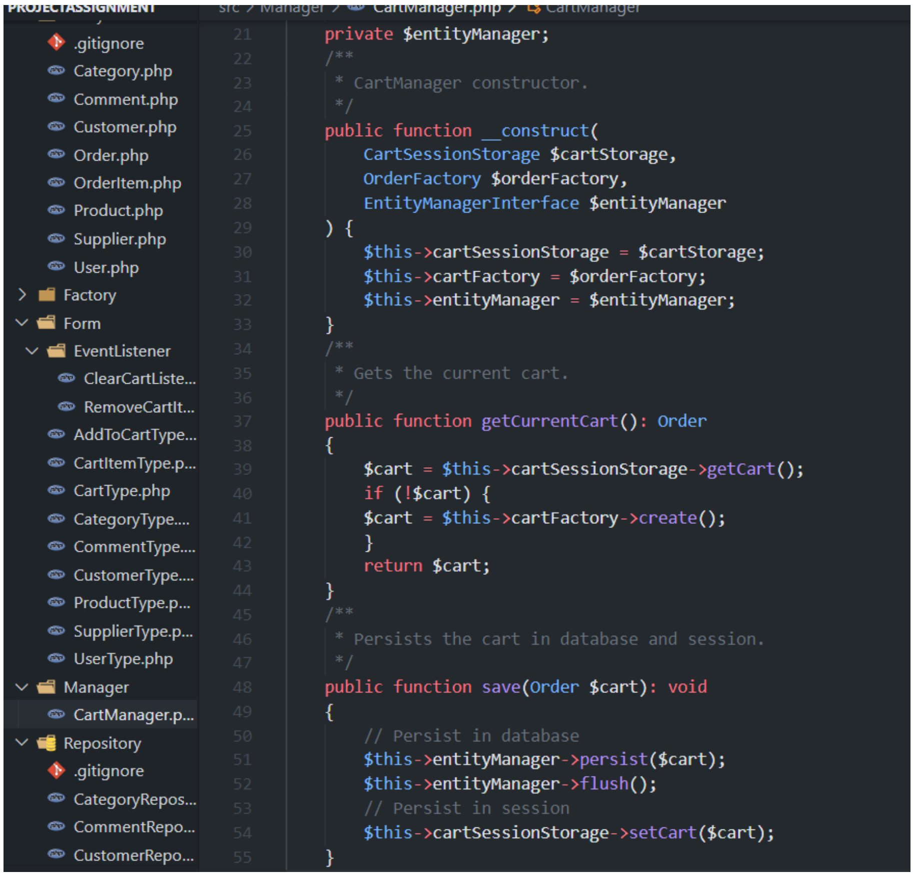
Create folder Storage and file CartSessionStorage.php with the purpose to manage the session of Order



```php
/**
 * CartSessionStorage constructor.
 */
public function __construct(RequestStack $requestStack, OrderRepository
$cartRepository)
{
    $this->requestStack = $requestStack;
    $this->cartRepository = $cartRepository;
}
/**
 * Gets the cart in session.
 */
public function getCart(): ?Order
{
    return $this->cartRepository->findOneBy([
    'id' => $this->getCartId(),
    'status' => Order::STATUS_CART,
]);
}
/**
 * Sets the cart in session.
 */
public function setCart(Order $cart): void
{
    $this->getSession()->set(self::CART_KEY_NAME, $cart->getId());
}
/**
 * Returns the cart id.
 */
private function getCartId(): ?int
{
    return $this->getSession()->get(self::CART_KEY_NAME);
}
private function getSession(): SessionInterface
{
    return $this->requestStack->getSession();
```

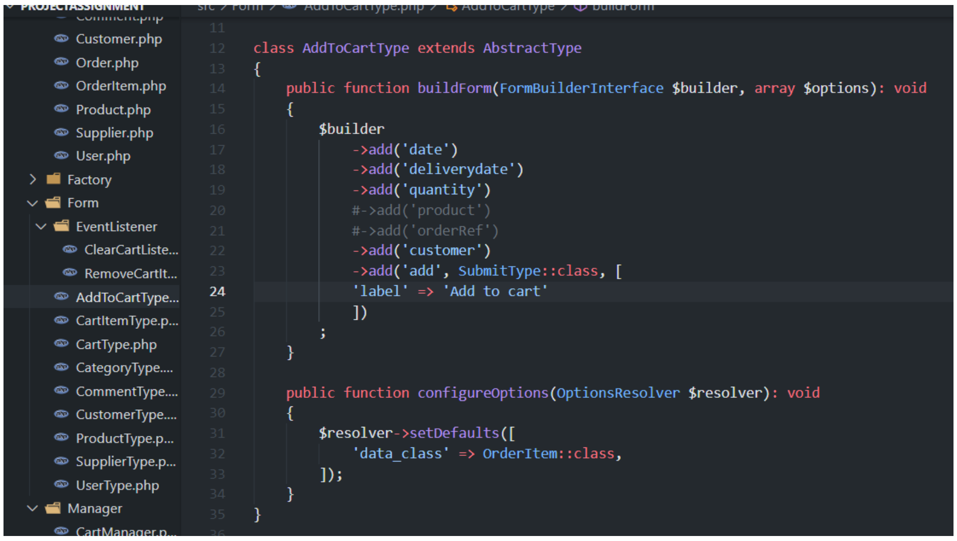
Next, Create the CartManager class that helps us retrieve the current cart of a user and saving the cart.

Make a folder called Manager and a file called CartManager.php in it to can perform Save function

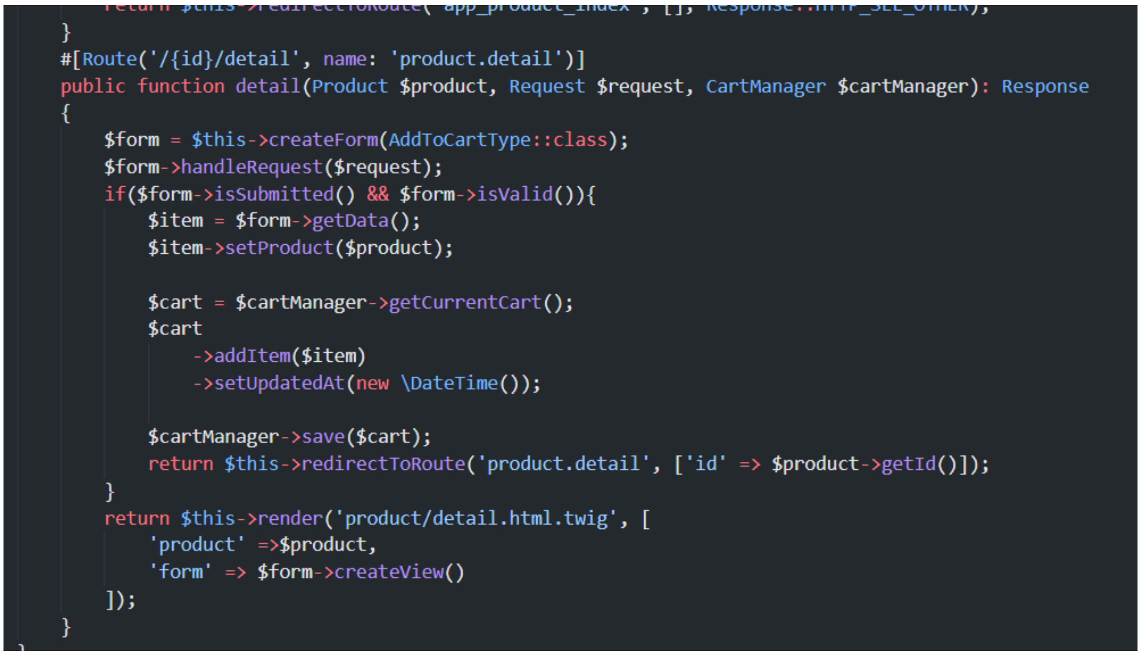


```php
private $entityManager;
/**
 * CartManager constructor.
 */
public function __construct(
    CartSessionStorage $cartStorage,
    OrderFactory $orderFactory,
    EntityManagerInterface $entityManager
) {
    $this->cartSessionStorage = $cartStorage;
    $this->cartFactory = $orderFactory;
    $this->entityManager = $entityManager;
}
/**
 * Gets the current cart.
 */
public function getCurrentCart(): Order
{
    $cart = $this->cartSessionStorage->getCart();
    if (!$cart) {
    $cart = $this->cartFactory->create();
    }
    return $cart;
}
/**
 * Persists the cart in database and session.
 */
public function save(Order $cart): void
{
    // Persist in database
    $this->entityManager->persist($cart);
    $this->entityManager->flush();
    // Persist in session
    $this->cartSessionStorage->setCart($cart);
}
```

Then using a symfony command to create AddToCartType: symfony console make: form AddToCartType OrderItem and display it in detail.html.twig

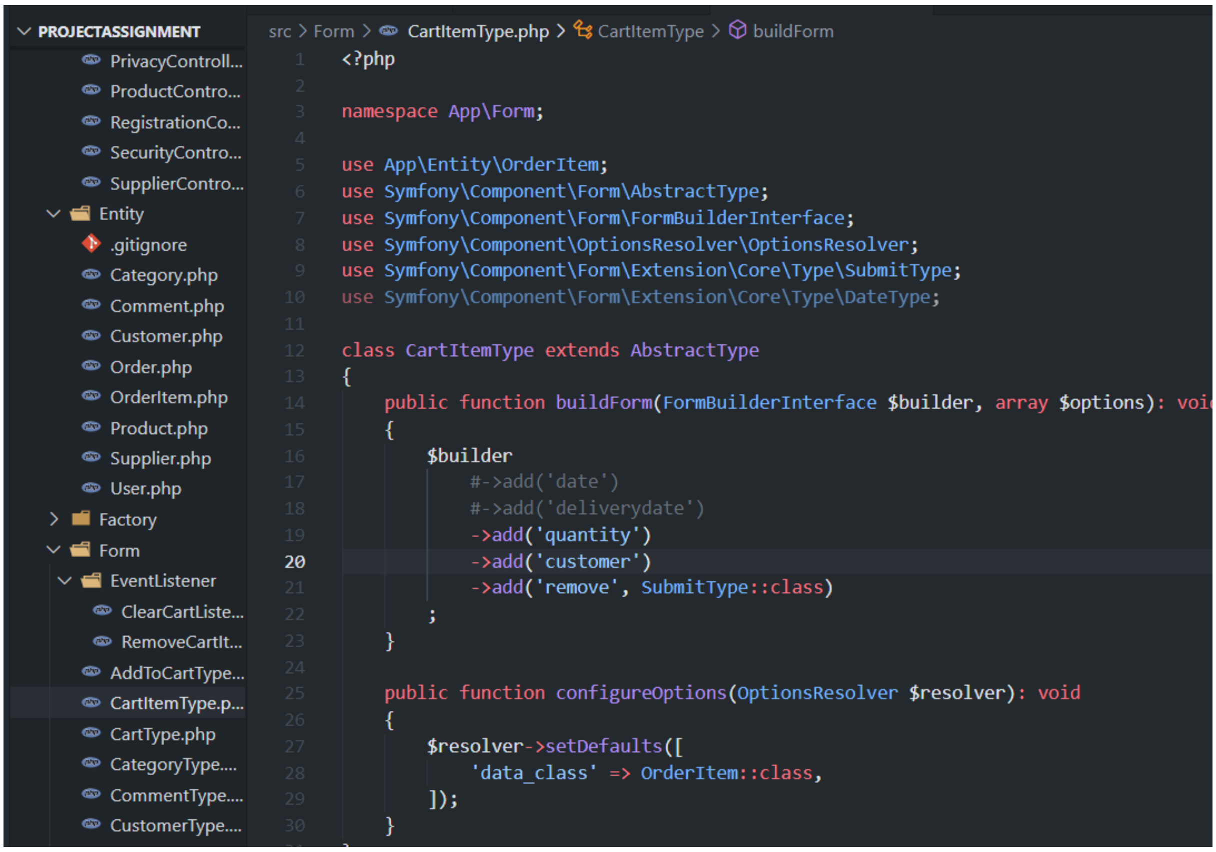


```php
class AddToCartType extends AbstractType
{
    public function buildForm(FormBuilderInterface $builder, array $options): void
    {
        $builder
            ->add('date')
            ->add('deliverydate')
            ->add('quantity')
            #->add('product')
            #->add('orderRef')
            ->add('customer')
            ->add('add', SubmitType::class, [
            'label' => 'Add to cart'
            ])
        ;
    }

    public function configureOptions(OptionsResolver $resolver): void
    {
        $resolver->setDefaults([
            'data_class' => OrderItem::class,
        ]);
    }
}
```



```
        return $this->redirectToRoute('app_product_index', [], Response::HTTP_SEE_OTHER);
    }
    #[Route('/{id}/detail', name: 'product.detail')]
    public function detail(Product $product, Request $request, CartManager $cartManager): Response
    {
        $form = $this->createForm(AddToCartType::class);
        $form->handleRequest($request);
        if($form->isSubmitted() && $form->isValid()){
            $item = $form->getData();
            $item->setProduct($product);

            $cart = $cartManager->getCurrentCart();
            $cart
                ->addItem($item)
                ->setUpdatedAt(new \DateTime());

            $cartManager->save($cart);
            return $this->redirectToRoute('product.detail', ['id' => $product->getId()]);
        }
        return $this->render('product/detail.html.twig', [
            'product' =>$product,
            'form' => $form->createView()
        ]);
    }
```

Then using a symfony command to create CartItemType: symfony console make: form CartItemType OrderItem

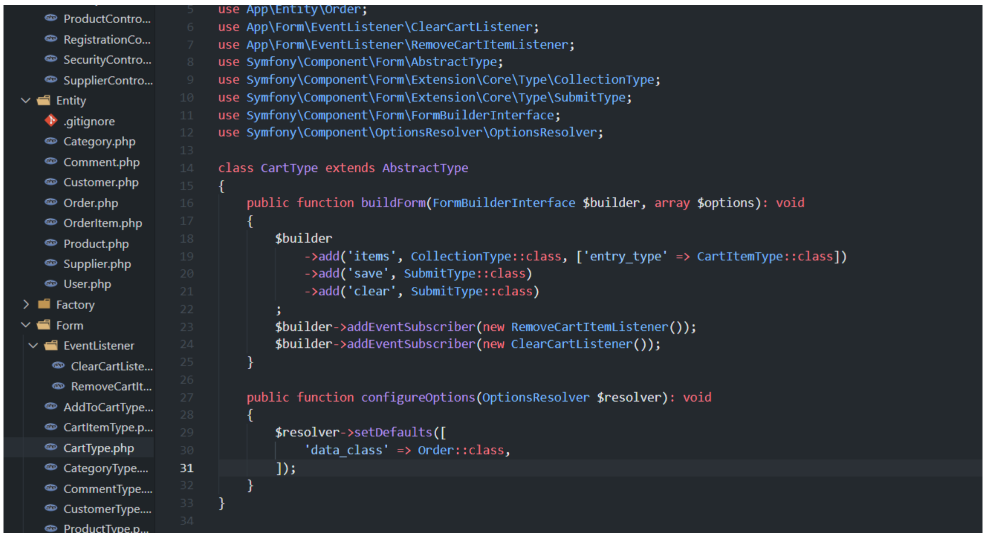


```
PROJECTASSIGNMENT
    PrivacyControll...
    ProductContro...
    RegistrationCo...
    SecurityContro...
    SupplierContro...
  Entity
    .gitignore
    Category.php
    Comment.php
    Customer.php
    Order.php
    OrderItem.php
    Product.php
    Supplier.php
    User.php
  Factory
  Form
    EventListener
      ClearCartListe...
      RemoveCartIt...
    AddToCartType...
    CartItemType.p...
    CartType.php
    CategoryType....
    CommentType....
    CustomerType....
```

src > Form > CartItemType.php > CartItemType > buildForm

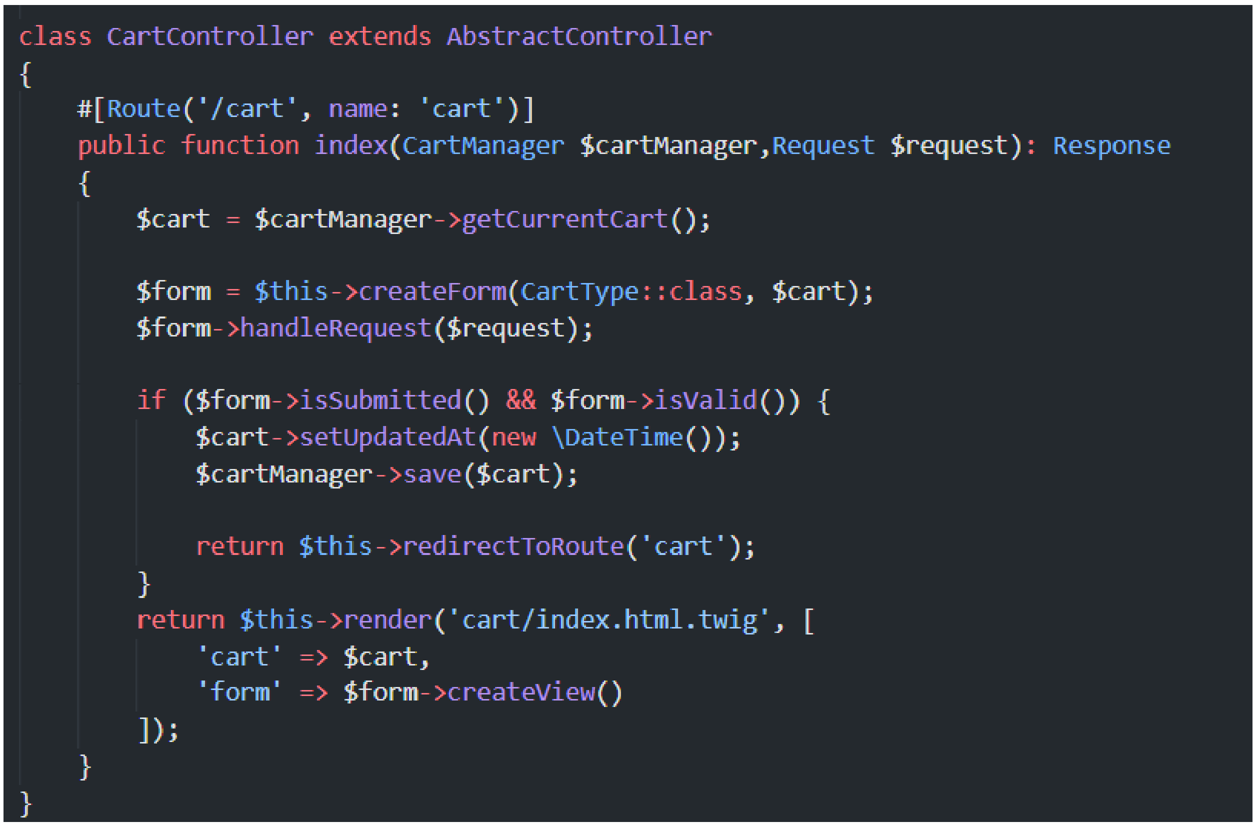
```
<?php

namespace App\Form;

use App\Entity\OrderItem;
use Symfony\Component\Form\AbstractType;
use Symfony\Component\Form\FormBuilderInterface;
use Symfony\Component\OptionsResolver\OptionsResolver;
use Symfony\Component\Form\Extension\Core\Type\SubmitType;
use Symfony\Component\Form\Extension\Core\Type\DateType;

class CartItemType extends AbstractType
{
    public function buildForm(FormBuilderInterface $builder, array $options): voi
    {
        $builder
            #->add('date')
            #->add('deliverydate')
            ->add('quantity')
            ->add('customer')
            ->add('remove', SubmitType::class)
        ;
    }

    public function configureOptions(OptionsResolver $resolver): void
    {
        $resolver->setDefaults([
            'data_class' => OrderItem::class,
        ]);
    }
```

Using a symfony command to create CartType: symfony console make: form CartType Order



```
use App\Entity\Order;
use App\Form\EventListener\ClearCartListener;
use App\Form\EventListener\RemoveCartItemListener;
use Symfony\Component\Form\AbstractType;
use Symfony\Component\Form\Extension\Core\Type\CollectionType;
use Symfony\Component\Form\Extension\Core\Type\SubmitType;
use Symfony\Component\Form\FormBuilderInterface;
use Symfony\Component\OptionsResolver\OptionsResolver;

class CartType extends AbstractType
{
    public function buildForm(FormBuilderInterface $builder, array $options): void
    {
        $builder
            ->add('items', CollectionType::class, ['entry_type' => CartItemType::class])
            ->add('save', SubmitType::class)
            ->add('clear', SubmitType::class)
        ;
        $builder->addEventSubscriber(new RemoveCartItemListener());
        $builder->addEventSubscriber(new ClearCartListener());
    }

    public function configureOptions(OptionsResolver $resolver): void
    {
        $resolver->setDefaults([
            'data_class' => Order::class,
        ]);
    }
}
```

Create the CartController: symfony console make:controller CartController

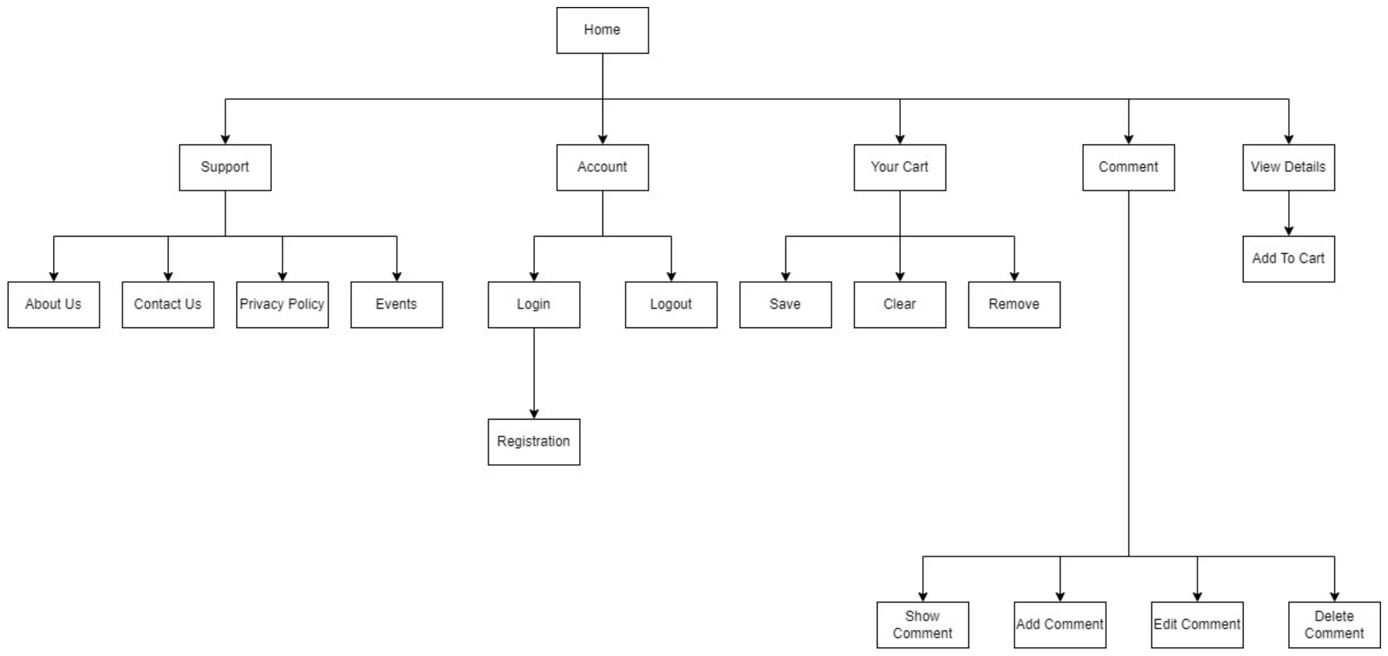


```
class CartController extends AbstractController
{
    #[Route('/cart', name: 'cart')]
    public function index(CartManager $cartManager,Request $request): Response
    {
        $cart = $cartManager->getCurrentCart();

        $form = $this->createForm(CartType::class, $cart);
        $form->handleRequest($request);

        if ($form->isSubmitted() && $form->isValid()) {
            $cart->setUpdatedAt(new \DateTime());
            $cartManager->save($cart);

            return $this->redirectToRoute('cart');
        }
        return $this->render('cart/index.html.twig', [
            'cart' => $cart,
            'form' => $form->createView()
        ]);
    }
}
```

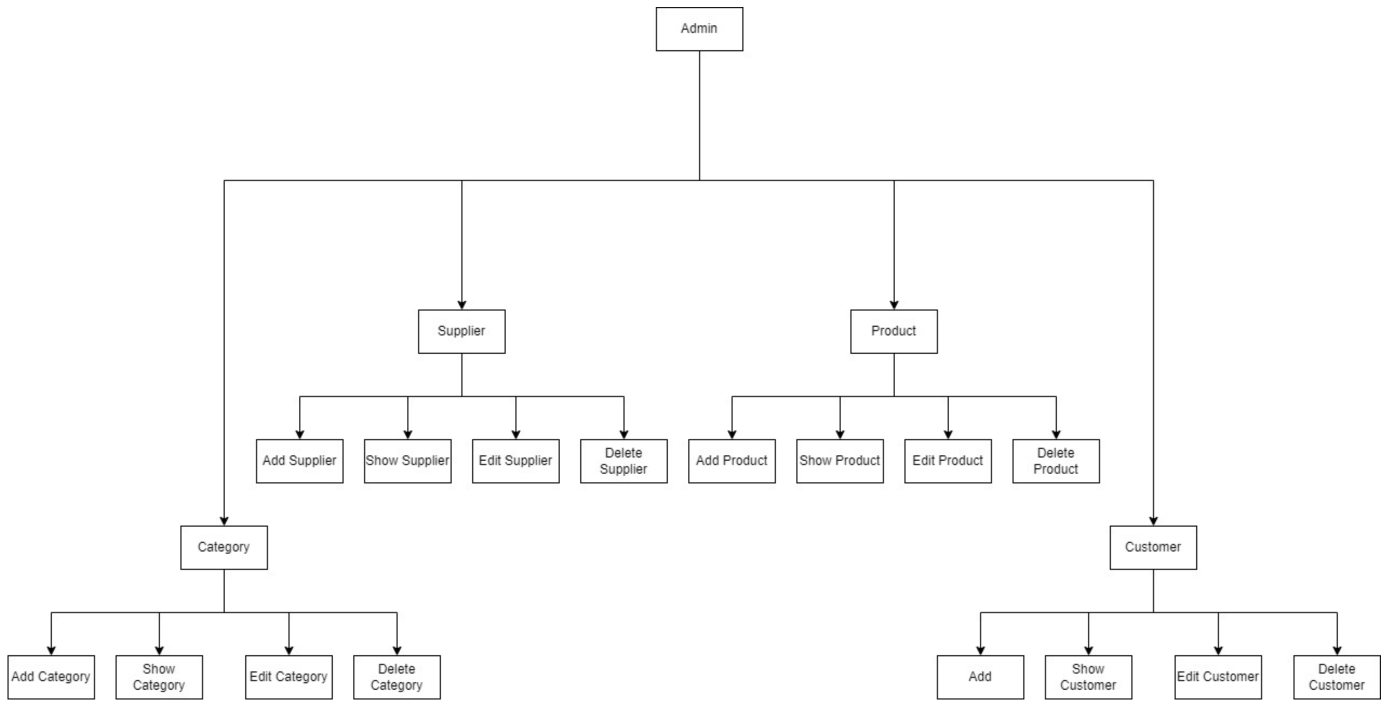
And rendering the cart page

2.3. Sitemap

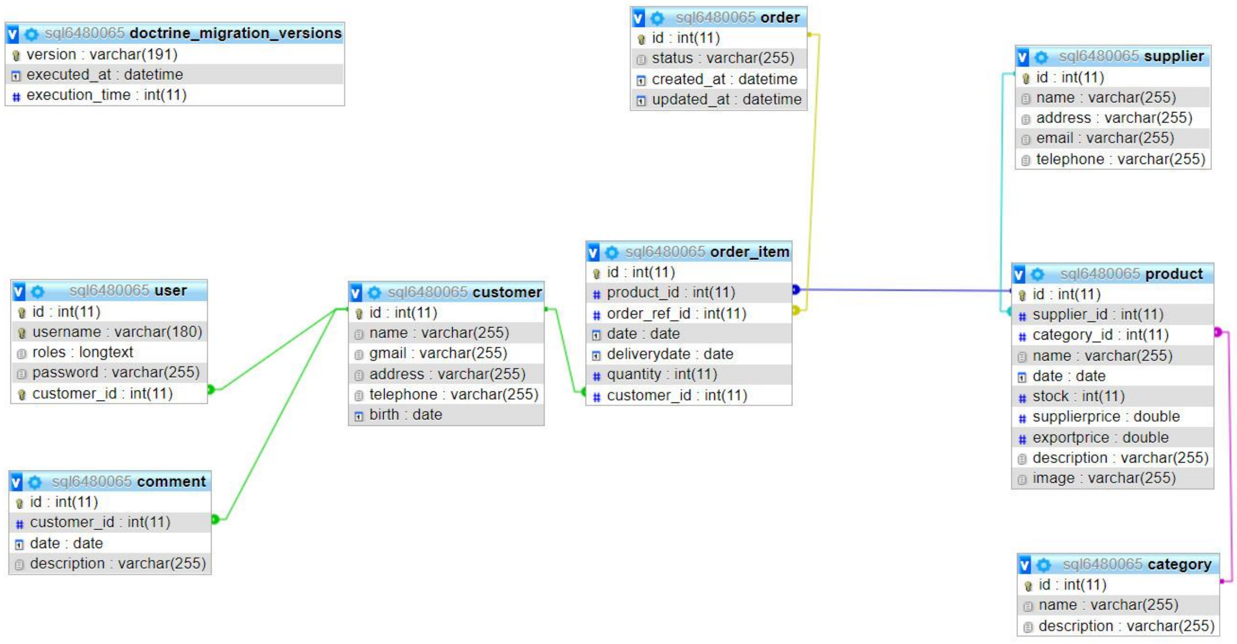
Sitemap of home page



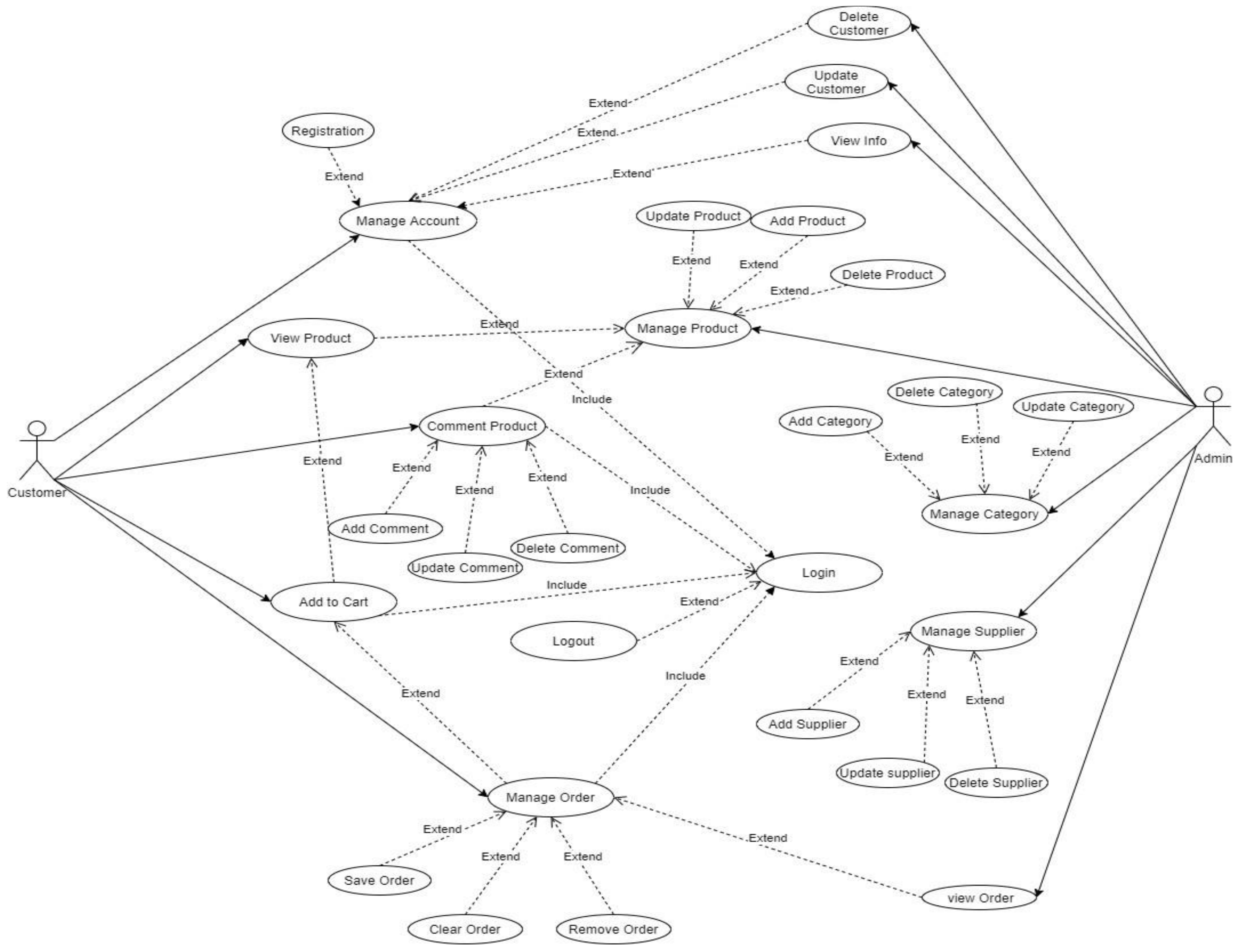
Sitemap of admin page



2.4. Entity Relationship Diagram



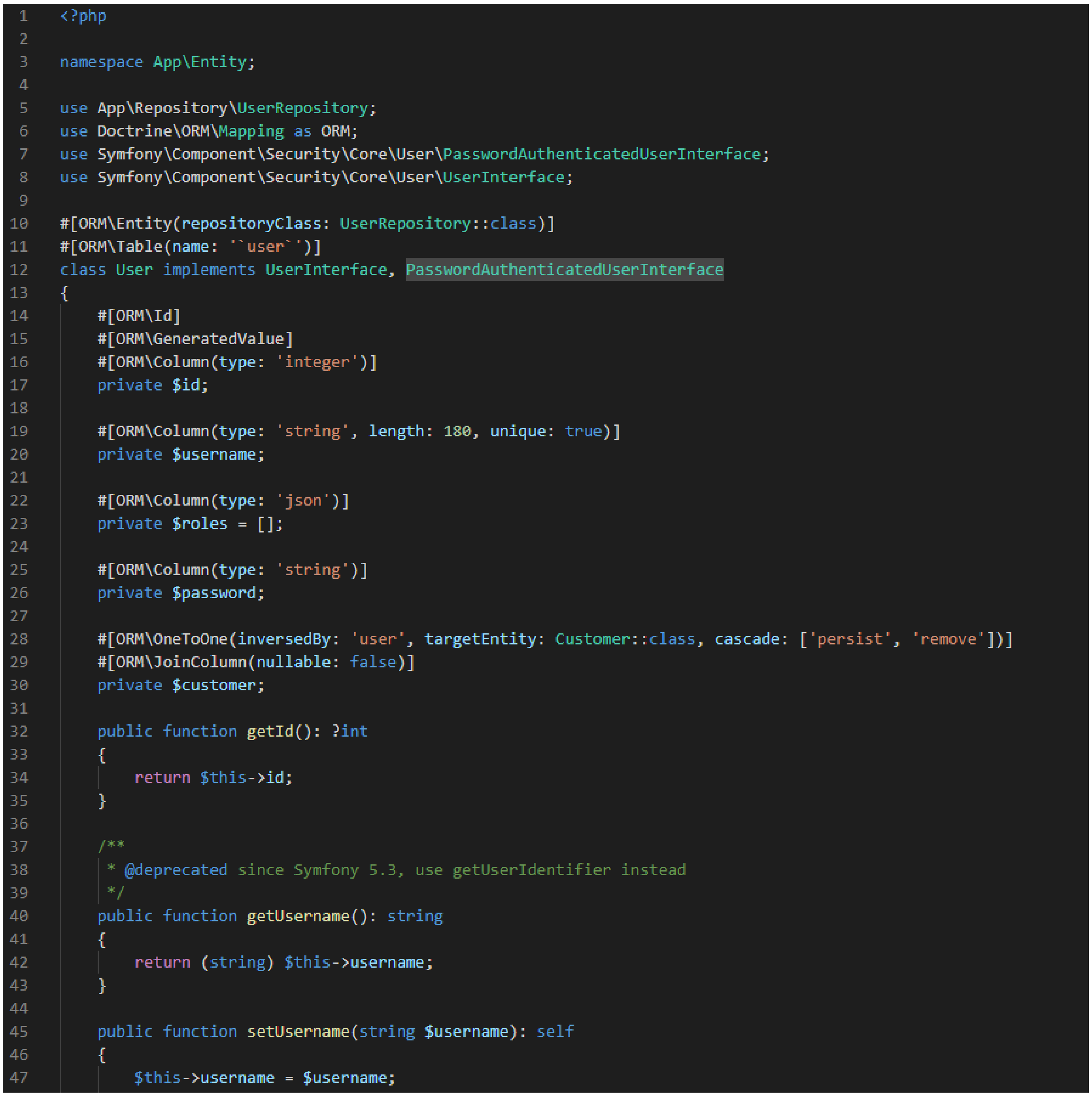
2.5. Use case



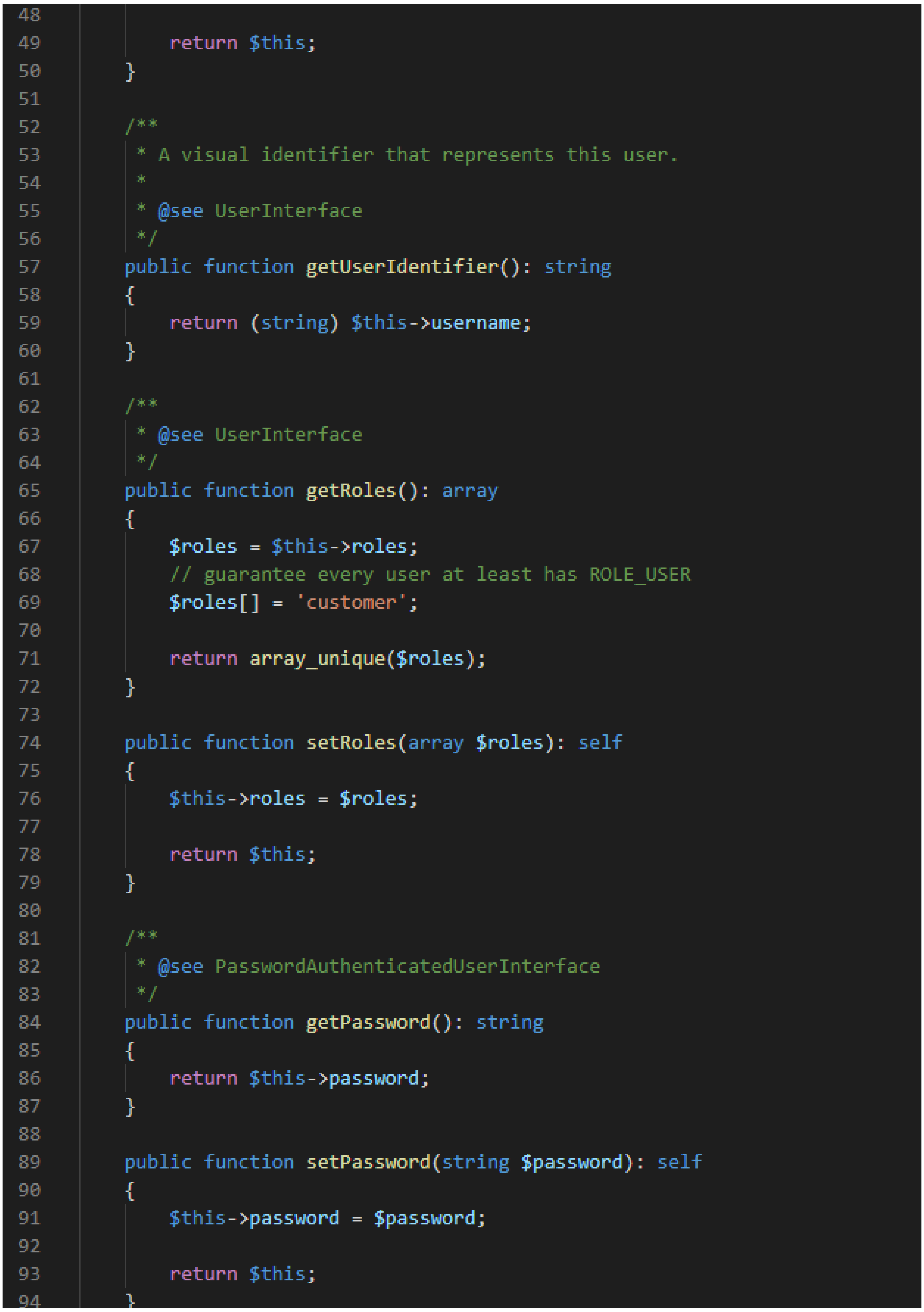
3. Implementation

3.1. Sample source code

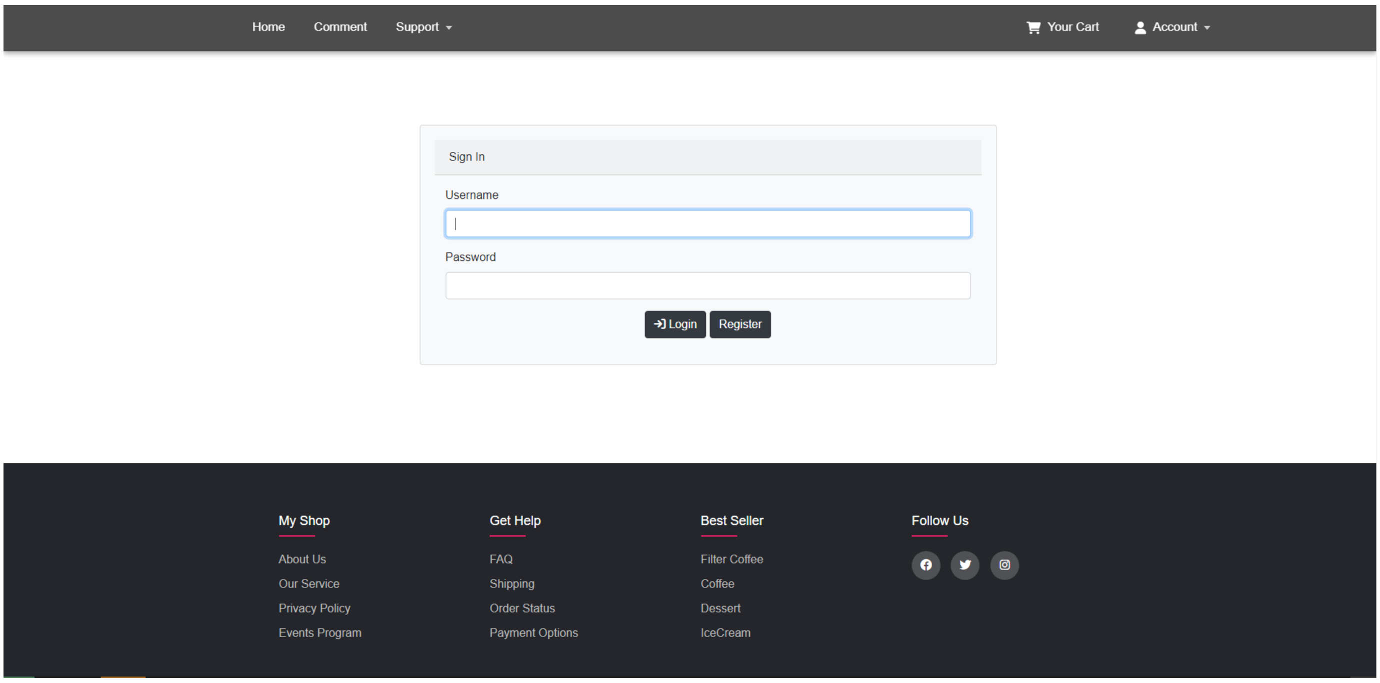
Here are some examples in our source code



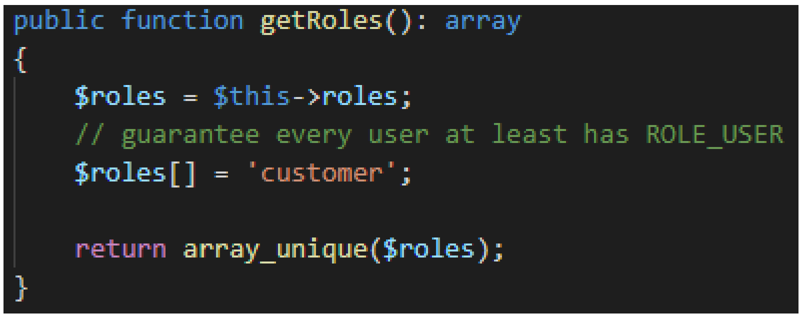
```php
<?php

namespace App\Entity;

use App\Repository\UserRepository;
use Doctrine\ORM\Mapping as ORM;
use Symfony\Component\Security\Core\User\PasswordAuthenticatedUserInterface;
use Symfony\Component\Security\Core\User\UserInterface;

#[ORM\Entity(repositoryClass: UserRepository::class)]
#[ORM\Table(name: '`user`')]
class User implements UserInterface, PasswordAuthenticatedUserInterface
{
    #[ORM\Id]
    #[ORM\GeneratedValue]
    #[ORM\Column(type: 'integer')]
    private $id;

    #[ORM\Column(type: 'string', length: 180, unique: true)]
    private $username;

    #[ORM\Column(type: 'json')]
    private $roles = [];

    #[ORM\Column(type: 'string')]
    private $password;

    #[ORM\OneToOne(inversedBy: 'user', targetEntity: Customer::class, cascade: ['persist', 'remove'])]
    #[ORM\JoinColumn(nullable: false)]
    private $customer;

    public function getId(): ?int
    {
        return $this->id;
    }

    /**
     * @deprecated since Symfony 5.3, use getUserIdentifier instead
     */
    public function getUsername(): string
    {
        return (string) $this->username;
    }

    public function setUsername(string $username): self
    {
        $this->username = $username;
```


```php

        return $this;
    }

    /**
     * A visual identifier that represents this user.
     *
     * @see UserInterface
     */
    public function getUserIdentifier(): string
    {
        return (string) $this->username;
    }

    /**
     * @see UserInterface
     */
    public function getRoles(): array
    {
        $roles = $this->roles;
        // guarantee every user at least has ROLE_USER
        $roles[] = 'customer';

        return array_unique($roles);
    }

    public function setRoles(array $roles): self
    {
        $this->roles = $roles;

        return $this;
    }

    /**
     * @see PasswordAuthenticatedUserInterface
     */
    public function getPassword(): string
    {
        return $this->password;
    }

    public function setPassword(string $password): self
    {
        $this->password = $password;

        return $this;
    }
```

Login and register function, with these function users can create an account and login the website. Users will have two roles: customer and admin. Account with customer role can login and do some basic interaction such as view details of the product, add them to cart, order items. With admin role, admin can perform CRUD in their admin interface

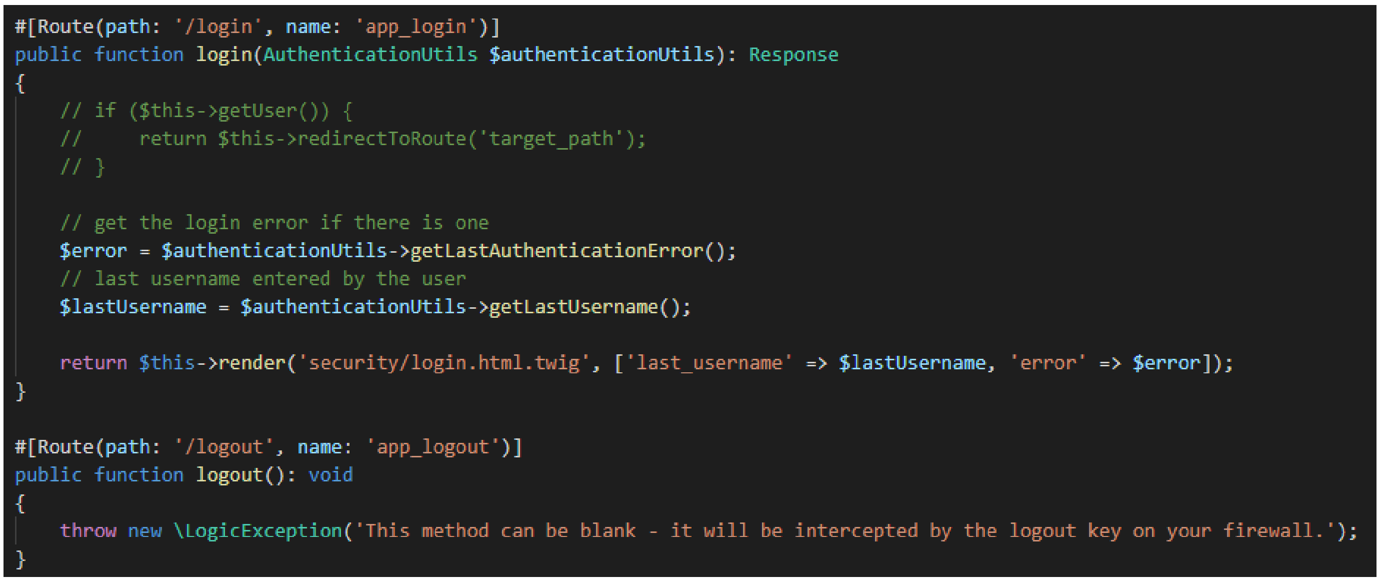


Each account create will have a default role it is customer



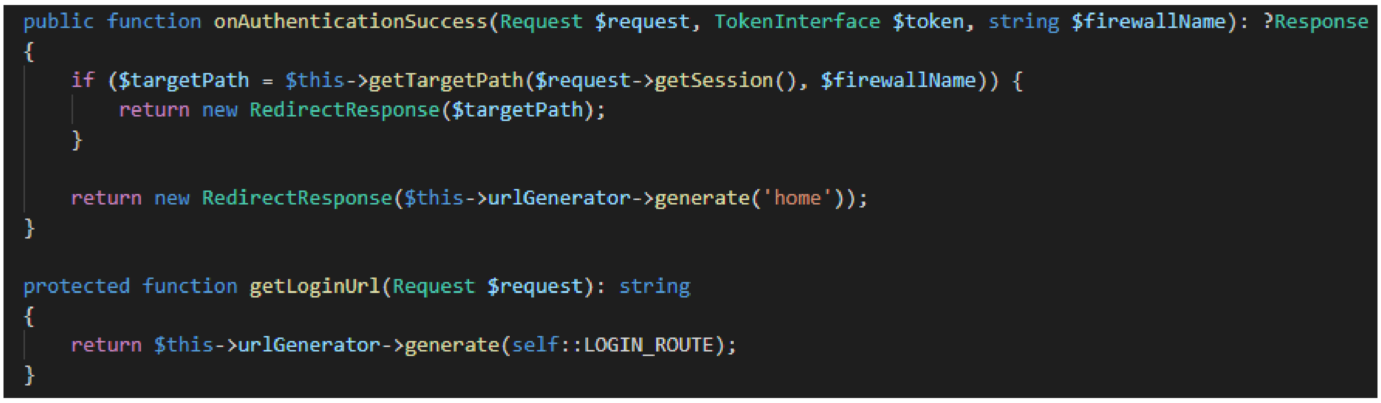
```
public function getRoles(): array
{
    $roles = $this->roles;
    // guarantee every user at least has ROLE_USER
    $roles[] = 'customer';

    return array_unique($roles);
}
```

In this figure bellow, those code perform login function, you can understand simply it will take information from user input and compare it to information in database if it matches to any user, it will send back to end user. And function logout it used to logout we user want to logout.



```
#[Route(path: '/login', name: 'app_login')]
public function login(AuthenticationUtils $authenticationUtils): Response
{
    // if ($this->getUser()) {
    //     return $this->redirectToRoute('target_path');
    // }

    // get the login error if there is one
    $error = $authenticationUtils->getLastAuthenticationError();
    // last username entered by the user
    $lastUsername = $authenticationUtils->getLastUsername();

    return $this->render('security/login.html.twig', ['last_username' => $lastUsername, 'error' => $error]);
}

#[Route(path: '/logout', name: 'app_logout')]
public function logout(): void
{
    throw new \LogicException('This method can be blank - it will be intercepted by the logout key on your firewall.');
}
```

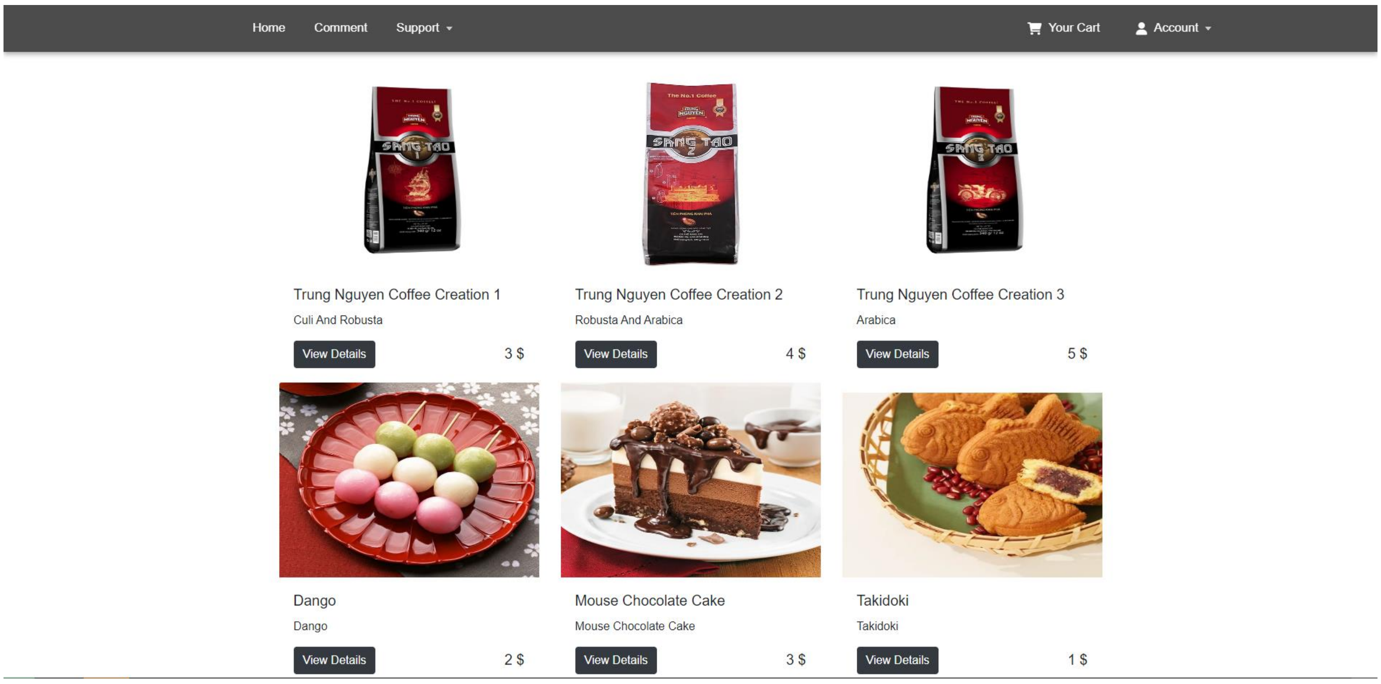
To create LoginFormAuthenticator, run this command in terminal “php bin/console make:auth”. This function will identify user, and if the user login success it will bring user back to home page



```
public function onAuthenticationSuccess(Request $request, TokenInterface $token, string $firewallName): ?Response
{
    if ($targetPath = $this->getTargetPath($request->getSession(), $firewallName)) {
        return new RedirectResponse($targetPath);
    }

    return new RedirectResponse($this->urlGenerator->generate('home'));
}

protected function getLoginUrl(Request $request): string
{
    return $this->urlGenerator->generate(self::LOGIN_ROUTE);
}
```

3.2. Images of final application

In home page, we use “php bin/console make:controller” to create controller for our entity. After that we config the index file. And here is the result.



Trung Nguyen Coffee Creation 1
Culi And Robusta
View Details 3 $

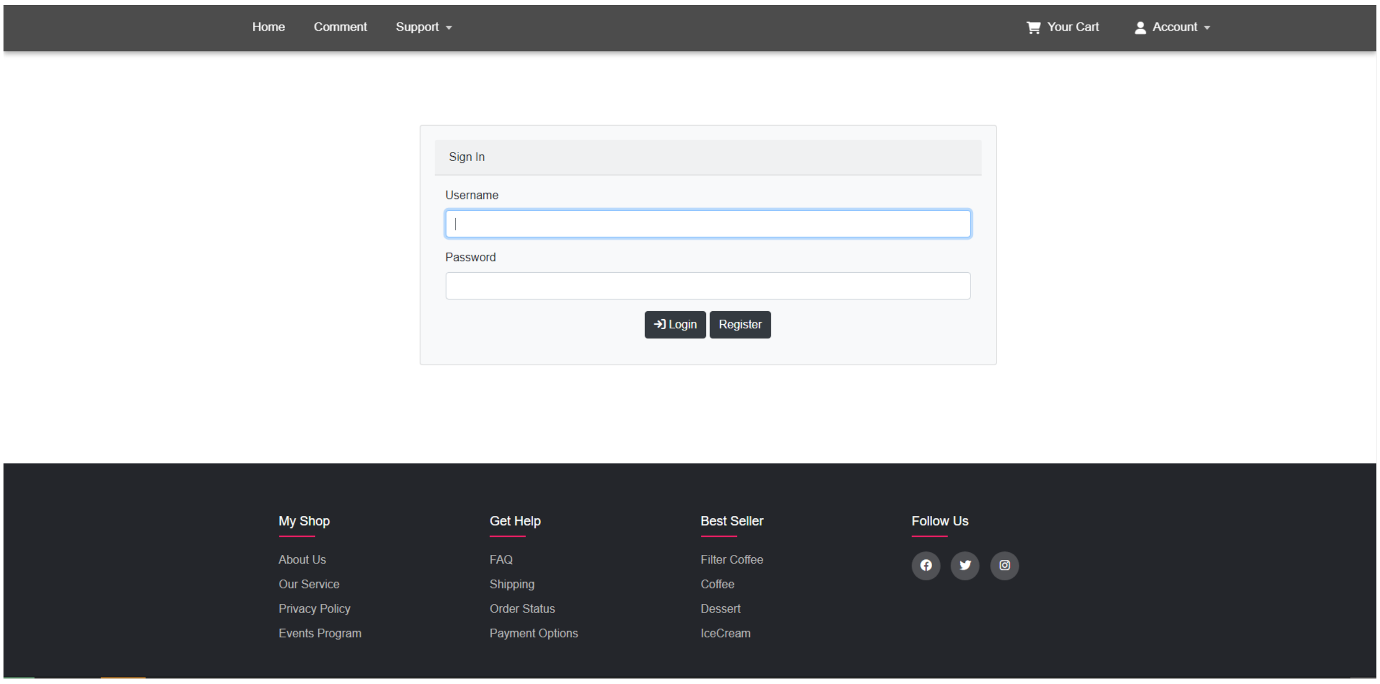
Trung Nguyen Coffee Creation 2
Robusta And Arabica
View Details 4 $

Trung Nguyen Coffee Creation 3
Arabica
View Details 5 $

Dango
Dango
View Details 2 $

Mouse Chocolate Cake
Mouse Chocolate Cake
View Details 3 $

Takidoki
Takidoki
View Details 1 $

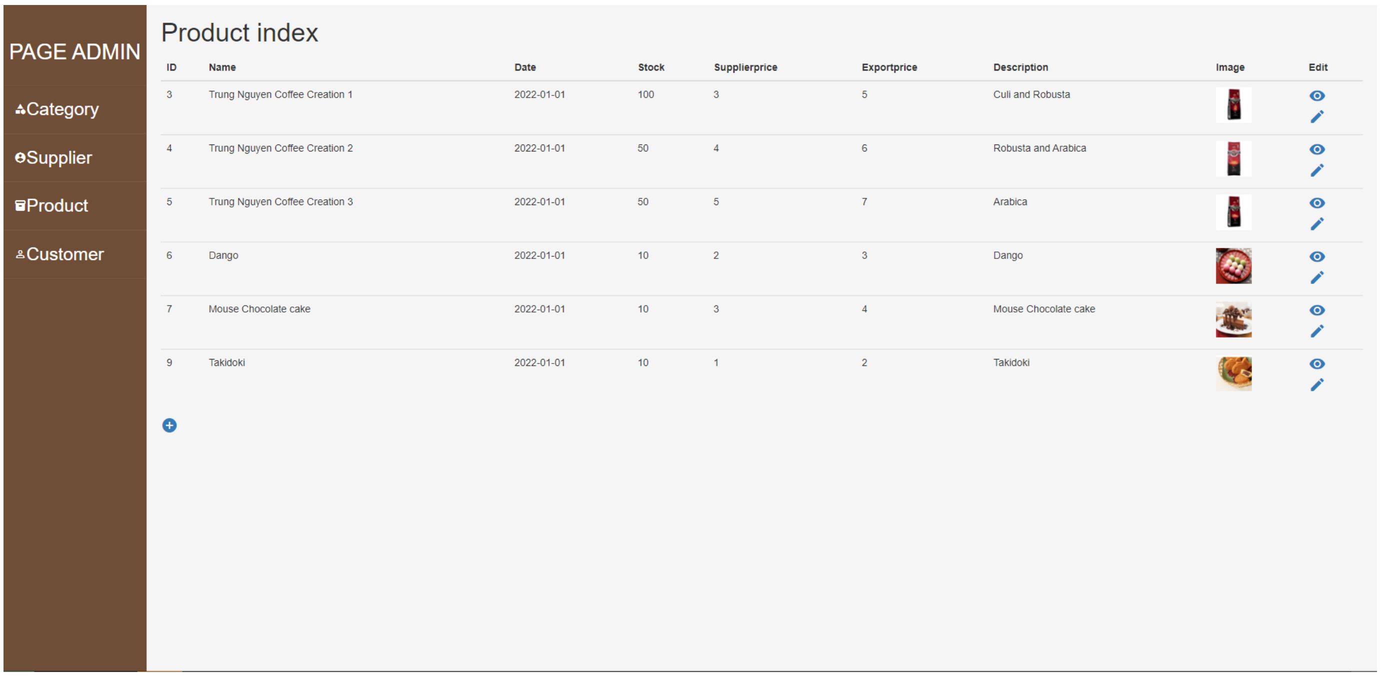


My Shop
About Us
Our Service
Privacy Policy
Events Program

Get Help
FAQ
Shipping
Order Status
Payment Options

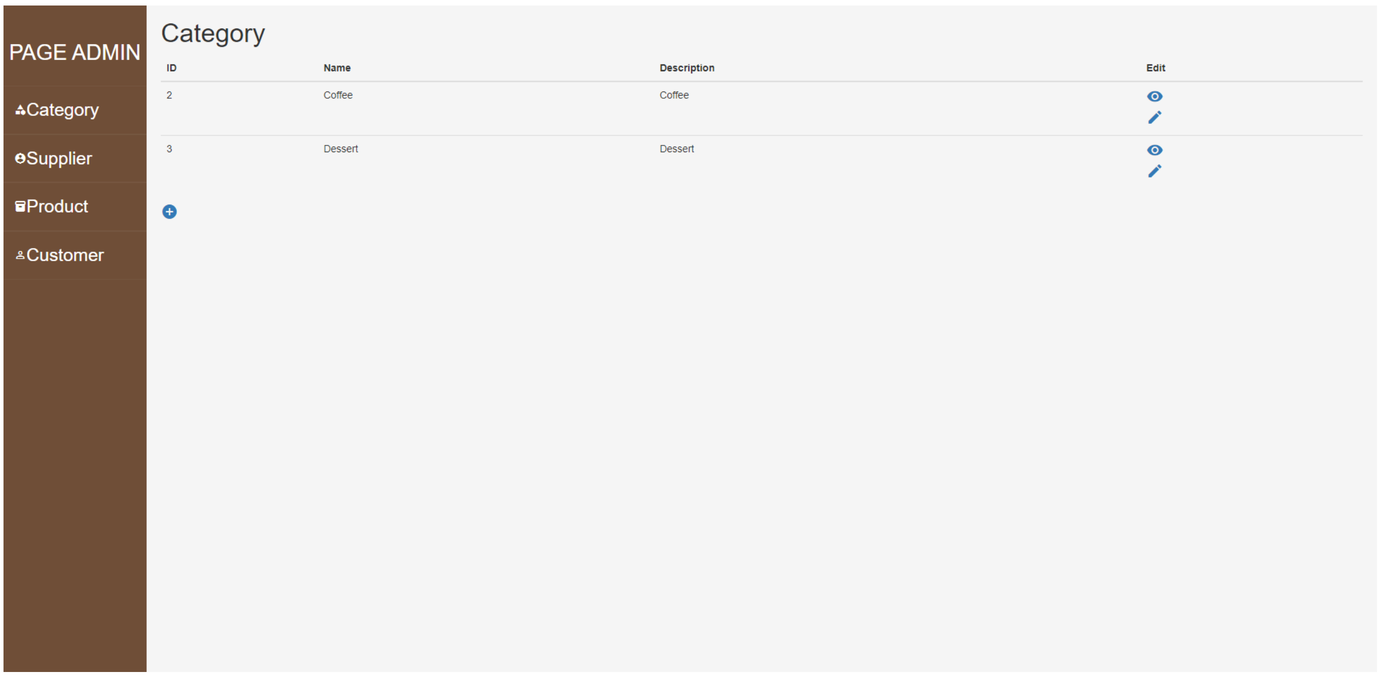
Best Seller
Filter Coffee
Coffee
Dessert
IceCream

Follow Us



Product index

ID	Name	Date	Stock	Supplierprice	Exportprice	Description	Image	Edit
3	Trung Nguyen Coffee Creation 1	2022-01-01	100	3	5	Culi and Robusta		👁 ✏
4	Trung Nguyen Coffee Creation 2	2022-01-01	50	4	6	Robusta and Arabica		👁 ✏
5	Trung Nguyen Coffee Creation 3	2022-01-01	50	5	7	Arabica		👁 ✏
6	Dango	2022-01-01	10	2	3	Dango		👁 ✏
7	Mouse Chocolate cake	2022-01-01	10	3	4	Mouse Chocolate cake		👁 ✏
9	Takidoki	2022-01-01	10	1	2	Takidoki		👁 ✏



Category

ID	Name	Description	Edit
2	Coffee	Coffee	👁 ✏
3	Dessert	Dessert	👁 ✏

3.3. GitHub repository evidences

Our GitHub link: https://github.com/PhuCuong020623/ProjectAssignment

4. Conclusion

4.1. What went well

- The interface of the website can operate smoothly and beautifully. The paths to other pages (such as Homepage, Comment Page, Contact Us page, about page, Events Page, Privacy page and so on) are correctly

- The website can perform features such as show, add, edit and, delete comments

- We can perform login and registration in the website

- We create the detail page, the cart page, place the product in the cart, and process the order.

- Besides, we create a detail page about product (including name, price, supplier, and other information), from that, users can read in more detail, adjust quantity and add to cart

- Creating a Cart page to list all products that users add to cart, it will calculate the quantity and total price of all products, besides, when the users order new products into the shopping cart, it will update automatically, and it will calculate the new quantity and total price. In addition, The Shopping cart can perform features such as save, remove or clear the cart

- We can perform CRUD operations in other pages such as Category, Customer, Supplier, Product.

4.2. What did no go well?

- In the interface of the admin page, some features such as add, show, update and remove that not display according to our wishes

- In the detail Page and Shopping cart, although we login in a specific account, we must still choose an account to add to cart

- In login function we can login a specific account with different role but admin role cannot access to admin page

4.3. Lessons learned and further improvements

Lesson learned:

- Understanding the structure and operation of MVC (Model – View - Controller)

- Using Symfony to design a website base on the structure and operation of MVC

- Understanding commands of Symfony and its functions.

- Using GitHub to work together.

Further improvements:

- In our website, adding some features and functions such as a Search tool to research for products follows name and checkout to payment, and pagination.

- The interface of the Admin Page will be modified to the interface can be beautiful and appropriate

- The customers can comment below of each product to express their opinion

- We can solve problems that relate to accounts of customers

-Try to catch bugs maybe happen in website

5. References

Chamat, R. (2020, June 3). *ewm.* Retrieved from https://ewm.swiss/en/blog/why-use-php-framework-symfony

Symfony. (n.d). *Symfony*. Retrieved from https://symfony.com/what-is-symfony