

## TD PHP MVC

### Étape 20 : Un panier pour passer commande.

Un panier est une commande non validée contenant une ou plusieurs lignes. Pour réaliser notre panier, nous allons créer une classe Cart, qui ne correspond à aucune table en base de données mais qui regroupe une commande et un tableau de lignes (command\_product)

Nous créons un dossier helper à la racine du projet pour y ranger notre classe Cart, nous devons donc modifier notre autoload des classes dans l'index.php pour qu'il puisse trouver les classes de type helper

```
index.php > autoload

3  function autoload($className){
4      if (strpos($className, "Controller")) {
5          $fileName = lcfirst(str_replace("Controller", "", $className));
6          require_once "controller/$fileName.controller.php";
7      }
8      else if (strpos($className, "Repository")){
9          $fileName = lcfirst(str_replace("Repository", "", $className));
10         require_once "repository/$fileName.repository.php";
11     }
12     else {
13         $fileName = lcfirst($className);
14         if(file_exists("entity/$fileName.entity.php")){
15             require_once "entity/$fileName.entity.php";
16         }
17         elseif (file_exists("helper/$fileName.helper.php")){
18             require_once "helper/$fileName.helper.php";
19         }
20     }
21 }
22 spl_autoload_register("autoload");
```

Le panier sera stocké en session jusqu'à la validation de la commande par l'utilisateur (il y aura alors création en base de données d'une command et de une ou plusieurs lignes command\_product)

Le panier état unique nous allons en profiter pour aborder les design patterns et plus particulièrement le design pattern « singleton »

Première version de notre classe Cart

```

helper > 🐘 cart.helper.php > 📁 Cart
1  <?php
2
3  class Cart
4  {
5      private static $_instance = null;
6      public static function getInstance()
7      {
8          if (is_null(self::$_instance)) {
9              self::$_instance = new Cart();
10         }
11
12         return self::$_instance;
13     }
14
15     private function __construct()
16     {
17         if (!isset($_SESSION['cart'])) {
18             $customer_id = null;
19             if (isset($_SESSION['logged'])) {
20                 $logged = unserialize($_SESSION['logged']);
21                 $customer_id = $logged->customer_id;
22             }
23             $command = new Command(['customer_id'=>$customer_id]);
24             $_SESSION['cart'] = ['command' => serialize($command),
25                                 'lines' => serialize([])];
26         }
27     }
28

```

C'est quoi un singleton ?

<https://apprendre-php.com/tutoriels/tutoriel-45-singleton-instance-unique-d-une-classe.html>

Dans le constructeur, nous initialisons ce dont nous avons besoin en session, le constructeur est private si bien que l'on ne peut pas faire de `new Cart()`. Pour récupérer un objet de type `Cart`, on passe par la méthode `getInstance` qui instancie et stocke dans la propriété (elle aussi private) `$_instance`.

Nous ajoutons 3 méthodes à notre singleton, une pour récupérer la commande stockée en session, une pour récupérer les lignes et une autre pour vider le panier une fois la commande validée.

```

helper > 🐼 cart.helper.php > 📁 Cart
28
29     function getCommand(){
30         return unserialize($_SESSION['cart']['command']);
31     }
32
33     function getLines(){
34         return unserialize($_SESSION['cart']['lines']);
35     }
36
37     function reset(){
38         unset($_SESSION['cart']);
39         $customer_id = null;
40         if (isset($_SESSION['logged'])) {
41             $logged = unserialize($_SESSION['logged']);
42             $customer_id = $logged->customer_id;
43         }
44         $command = new Command(['customer_id'=>$customer_id]);
45         $_SESSION['cart'] = ['command' => serialize($command),
46                             'lines' => serialize([])];
47     }
48

```

Enfin, la méthode ajoutant un produit ou mettant à jour la quantité d'un produit déjà présent.

```

helper > 🐼 cart.helper.php > 📁 Cart > 📁 addOrUpdateLine
49     function addOrUpdateLine($product_id, $quantity){
50         $cart_lines = unserialize($_SESSION['cart']['lines']);
51
52         $exists = false;
53         foreach($cart_lines as $line){
54             if($line->product_id == $product_id){
55                 $line->quantity += $quantity;
56                 $line->price += $quantity * $line->product->price;
57                 $exists = true;
58                 break;
59             }
60         }
61         if(!$exists){
62             $repo = new MainRepository('product');
63             $product = $repo->getOne($product_id);
64             $line = new Command_product(['product_id' => $product_id,
65                                         'quantity'   => $quantity,
66                                         'price'       => $quantity * $product->price,
67                                         'product'     => $product]);
68             array_push($cart_lines, $line);
69         }
70
71         $_SESSION['cart']['lines'] = serialize($cart_lines);
72     }
73

```

Nous créons le contrôleur CartController et l'action par défaut index pour afficher le détail du panier ainsi que le template associé.

```
controller > 🐞 cart.controller.php > 🛠️ CartController > 📦 index
1  <?php
2
3  class CartController extends BaseController
4  {
5      function index(){
6          $cart = Cart::getInstance();
7          $cart_lines = $cart->getLines();
8          $this->entities = ['cart_lines' => $cart_lines];
9          $this->render();
10     }
11
```

```
template > cart > 🐞 index.view.php
1  <?php $cartTotalPrice = 0; ?>
2  <?php foreach($cart_lines as $line){
3      $cartTotalPrice += $line->price;
4  } ?>
5      <?= $line->product->name ?> x <?= $line->quantity ?> : <?= $line->price ?> € <br>
6  <?php }?>
7  <b><u>Total Panier :</u> <?= $cartTotalPrice ?>€</b>
8  <hr>
9  <a href="/cart/command">Commander</a>
```

On ajoute un lien vers cette route dans le header de notre index.php

```
<header>
    <?= isset($_SESSION['logged']) ?
        '<a href="/user/logout">Logout</a>' :
        '<a href="/user/login">Login</a>'?>
    <a href="/cart" class="ml-3">Panier</a>
</header>
```

Il nous reste à ajouter la fonction d'ajout d'un produit (et de sa quantité) au panier. Nous ajoutons un petit formulaire dans le template /product/read.view.php

```
template > product > 🐞 read.view.php
17  <a href="/product/update/<?= $product->id ?>" class="btn btn-sm btn-success mb-3">
18      Modifier
19  </a>
20  <br>
21  <form action="/cart/add/<?= $product->id ?>" method="post">
22      <input type="number" name="quantity" id="quantity" value="1" min="1" max="9" step="1" >
23      <input type="submit" name="addToCart" id="addToCart" value="+" class="btn btn-sm btn-primary mb-1">
24  </form>
```

L'action lors du submit pointe vers la route /cart/add/id. Nous allons donc créer cette méthode dans notre CartController

```
controller > cart.controller.php > CartController > add
11
12     function add(){
13         $redirect = $_SERVER['HTTP_REFERER'];
14         $errors = [];
15         $posted = [];
16
17         if (isset($_SESSION['post'])) {
18             $posted = $_SESSION['post'];
19             unset($_SESSION['post']);
20
21             $product_id = $this->id;
22             $quantity = $posted['quantity'];
23
24             $cart = Cart::getInstance();
25             $cart->addOrUpdateLine($product_id, $quantity);
26         }
27
28         header("Location: $redirect");
29         die;
30     }
```




Nous redirigeons vers la page d'où provient le submit donc pas besoin de template pour cette action.

Il nous reste à coder la validation du panier (sa transformation en commande). Nous avons mis sur notre template /cart/index.view.php servant à afficher le contenu du panier un lien vers /cart/command

```
template > cart > index.view.php
8     <hr>
9     <a href="/cart/command">Commander</a>
10
```

Nous allons créer l'action correspondante dans notre CartController (ici aussi, nous redirigerons vers user/commands ou /cart/ en cas d'échec, donc pas besoin de template)

Debuggez en pas à pas pour bien comprendre le principe de l'action command :-)

controller >  cart.controller.php >  CartController >  add

```
31
32     function command(){
33         if(!isset($_SESSION['logged'])){
34             header('Location: /user/login');
35             die;
36         }
37         $logged = unserialize($_SESSION['logged']);
38         $customer_id = $logged->customer_id;
39         $cart = Cart::getInstance();
40         $command = $cart->getCommand();
41         $command->customer_id = $customer_id;
42         $cart_lines = $cart->getLines();
43         $repo = new MainRepository('command');
44         $fields = [];
45         foreach($command as $k => $v){
46             $fields[$k] = $v;
47         }
48         //TODO command.date //.state //.number
49         $repo->validate($fields);
50         $command = $repo->insertOne($fields);
51         foreach($cart_lines as $line){
52             $line->command_id = $command->id;
53         }
54         $repo = new MainRepository('command_product');
55         $lines = [];
56         foreach($cart_lines as $line){
57             $lineFields = [];
58             foreach($line as $k => $v){
59                 $lineFields[$k] = $v;
60             }
61             $repo->validate($lineFields);
62             array_push($lines, $lineFields);
63         }
64         $result = $repo->insertMany($lines);
65         if($result == true){
66             $cart = Cart::getInstance();
67             $cart->reset();
68             header('Location: /user/commands');
69             die;
70         }
71         header('Location: /cart');
72         die;
73     }
```

**THE END**