

TD PHP MVC

Étape 15 : Formulaires de création de données pour product

L'objectif de cette étape est d'ajouter une route permettant la création d'un produit (par l'administrateur du site)

La route vers cette vue sera : /product/create. Il faut donc compléter notre controller ProductController avec une méthode create et créer le template associé : template/product/create.view.php.

Pour pouvoir créer un produit en base de données, nous allons avoir besoin d'un formulaire dans le template qui enverra ce qui est saisi par l'admin en POST (sur la même route).

Nous allons stocker le contenu de \$_POST en session et rediriger vers la même route afin de supprimer le popup qui s'affiche lors d'un refresh du navigateur (suite à un submit de formulaire)

Modifications sur le point d'entrée de l'application : index.php

Pour pouvoir stocker en session, il faut « lancer » la session avec session_start(), on le fait tout en haut d'index.php

```
1+ <?php session_start();
```

Puis nous allons voir si lors d'une requête, quelque chose est stocké en \$_POST et si c'est le cas, nous sauvegardons le contenu de \$_POST en session (\$_SESSION['post']) avant de rediriger vers la même route.

```
18
19 $route = $_SERVER["REQUEST_URI"];
20+
21+ if(count($_POST) > 0){
22+     $_SESSION['post'] = $_POST;
23+     $_SESSION['files'] = $_FILES;
24+     foreach($_SESSION['files'] as &$file){
25+         if($file['error'] == 0){
26+             $ext = explode('.', $file['name']);
27+             $ext = array_pop($ext);
28+             $name = "img".(microtime(true)*10000).".$ext";
29+             move_uploaded_file($file['tmp_name'], $_SERVER["DOCUMENT_ROOT"]."/assets/img/temp/".$name);
30+             $file['tmp_name'] = $_SERVER["DOCUMENT_ROOT"]."/assets/img/temp/".$name;
31+         }
32+     }
33+
34+     header("Location: $route");
35+     die;
36+ }
37+
38 require_once 'router.php';
39 $router = new Router($route);
40
```

Dans le code ci-dessus, en plus de stocker le contenu de \$_POST, nous récupérons les éventuels fichiers uploadés (dans \$_FILES) que nous venons stocker dans \$_SESSION['files']. Nous faisons également une copie du fichier dans un répertoire temporaire : /assets/img/temp avec un nom unique pour chaque fichier (grâce à microtime)

Voyons maintenant le contenu de la méthode create de ProductController :

```
controller > product.controller.php > ProductController > create

32 function create(){
33     $errors = [];
34     $posted = [];
35
36     if(isset($_SESSION['post'])){
37
38         $posted = $_SESSION['post'];
39         unset($_SESSION['post']);
40         $files = $_SESSION['files'];
41         unset($_SESSION['files']);
42
43         if(isset($posted['cancel'])){
44             //TODO Delete temp image file if exists
45             header("Location: /product");
46             die;
47         }
48
49         if(isset($files['image_path']) && $files['image_path']['error'] == 0){
50             $file = $files['image_path'];
51             $posted['image_path'] = str_replace("/temp","", $file['tmp_name']);
52             $bp = 0;
53         }
54         if(isset($posted['image_path'])){
55             $temp_image_path = explode('/', $posted['image_path']);
56             $indexToInsert = count($temp_image_path) - 1;
57             array_splice( $temp_image_path, $indexToInsert, 0, "temp" );
58             $temp_image_path = implode('/', $temp_image_path);
59         }
60     }
```

```
controller > product.controller.php > ProductController > create

60
61     $repo = new MainRepository("product");
62     $errors = $repo->validate($posted);
63     if(count($errors) == 0){
64         $result = $repo->insertOne($posted);
65         if($result != false){
66             if (isset($posted['image_path'])) {
67                 $test = rename($temp_image_path, $posted['image_path']);
68             }
69             header("Location: /product/read/$result->id");
70             die;
71         }
72     }
73
74 }
75
76 $repository = new MainRepository("category");
77 $categories = $repository->getAll();
78
79 $this->entities = [ 'categories' => $categories,
80                   'errors' => $errors,
81                   'posted' => $posted ];
82
83 $this->render();
84 }
```

Explications :

L 33-34 : créations de 2 tableaux pour stocker les erreurs de saisie dans le formulaire (par exemple le prix doit être un nombre de type float) et les données postées.

L 36 : Si le formulaire a déjà été envoyé, `$_SESSION['post']` contient les données envoyées, il n'est pas vide, le code entre les lignes 37 et 73 sera alors exécuté. Nous reviendrons sur cette partie ensuite ...


L 76-77 : Nous récupérons les catégories existantes pour afficher les options du select permettant de choisir la catégorie du produit en cours de création

L 79-83 : nous passons les données dont nous allons avoir besoin dans le template et appelons la méthode `render` pour remplir ce template avec ces données.

Avant de se pencher sur les lignes 37 à 73 (exécutées après le submit de la form), voyons le code du template.

(Voir page suivante ...)

```

template > product >  create.view.php
1 <h2>Nouveau Produit</h2>
2 <form action="" method="post" enctype="multipart/form-data">
3   <label for="name">Nom</label><br>
4   <input type="text" name="name" id="name" value="<?=$posted['name'] ?? '' ; ?>">
5   <br><br>
6   <label for="description">Description</label><br>
7   <textarea name="description" id="description" cols="30" rows="10"><?=$posted['description'] ?? '' ; ?></textarea><br><br>
8   <label for="price">Prix</label><br>
9   <input type="text" name="price" id="price" value="<?=$posted['price'] ?? '' ; ?>">
10  <label class="text-danger"><?=$errors['price'] ? 'Champ invalide, saisir un prix SVP.' : '' ; ?></label><br><br>
11  <label for="image_path">Image</label><br>
12  <input type="file" name="image_path" id="image_path"><br><br>
13  <label for="category_id">Catégorie</label><br>
14  <select id="category_id" name="category_id">
15    <option value="null">Choisir ...</option>
16    <?php foreach($categories as $category){ ?>
17      <option value="<?=$category->id?" <?=$category->id == ($posted['category_id'] ?? '') ? 'selected' : '' ?> >
18        <?=$category->name?>
19      </option>
20    <?php } ?>
21  </select><br><br>
22  <div class="d-flex">
23    <input type="submit" name="create" id="create" value="Valider">
24    <input type="submit" name="cancel" id="cancel" value="Annuler" class="ml-2">
25  </div><br><br>
26  <label class="text-danger"><?=$errors ? count($errors) > 0 ? 'Erreur de saisie dans le formulaire.' : '' ; ?></label>
27 </form>

```

Les `label.text-danger` servent à afficher d'éventuelles erreurs de saisie.

`$posted` contient les valeurs précédemment postées à ré-afficher à l'utilisateur s'il a commis des erreurs lors de la saisie et qui doit modifier quelque chose (ça lui évite de devoir tout re-saisir)

Pour les options (ligne 17) correspondant aux catégories possibles, nous ajoutons `selected` sur celle correspondant à l'éventuelle saisie antérieure (toujours en cas d'erreur de saisie).

Revenons sur le code de la méthode `create` du `ProductController`, celui exécuté une fois la form envoyée (`submit`)

L 38-41 : Nous récupérons les données stockées en session ainsi que les fichiers dans des variables locales et nous vidons la session.

L 43-47 : Si le bouton pressé est `cancel`, nous redirigeons vers la liste des produits, sinon nous continuons le traitement.

L 49-59 : Nous faisons des traitements sur le nom du fichier uploadé pour pouvoir insérer par la suite en base de donnée le chemin définitif du fichier (`image_path`)

L 61-62 : Nous créons le repository pour insérer en base dans la table `product` et nous validons les données avant insertion avec la méthode `validate`.

L 63-64 : Si aucune erreur n'est détectée par `validate`, nous insérons le nouveau produit en base avec `insertOne`

L 65-71 : Si l'insertion en base s'est bien passée, `$result` contient l'entity `Product` correspondante. Nous pouvons déplacer l'image uploadée vers son emplacement définitif. Nous pouvons ensuite rediriger vers la page de détail `/product/read` avec l'id du produit que l'on vient de créer.

Nous allons voir maintenant les fonctions `validate` et `insertOne` de la classe `MainRepository`.

(Voir page suivante ...)

```

repository > main.repository.php > MainRepository > validate
146 function validate(&$inputs){
147     $errors = [];
148     $columns = $this->describe();
149     foreach($inputs as $k => $v){
150         $value = filter_var($v, FILTER_SANITIZE_STRING);
151         $column = $columns[$k] ?? null;
152         if(!isset($column) || $value == 'null'){
153             unset($inputs[$k]);
154             continue;
155         }
156         $type = $column['Type'];
157         $type = explode(' ', $type)[0];
158         if(in_array($type, ["float"])){
159             $filtered = filter_var($value, FILTER_SANITIZE_NUMBER_FLOAT, FILTER_FLAG_ALLOW_FRACTION);
160             if($filtered != $value || empty($filtered)){
161                 $errors[$k] = true;
162             }
163         }
164         if(in_array($type, ["int"])){
165             $filtered = filter_var($value, FILTER_SANITIZE_NUMBER_INT);
166             if($filtered != $value || empty($filtered)){
167                 $errors[$k] = true;
168             }
169         }
170     }
171     return $errors;
172 }

```

Cette fonction doit vérifier que les données saisies dans le formulaire sont conformes à ce qui est attendu en base de données. Elle sert également à prévenir contre les injections SQL et les failles XSS (voir sur Google pour explications) en « nettoyant » les données.

L 146 : le & devant le paramètre \$inputs permet de passer ce paramètre par référence pour qu'il soit modifié par la fonction.

L 147 : initialisation du tableau qui contiendra les erreurs s'il y en a.

L 148 : nous récupérerons des infos sur les colonnes de la table dans laquelle nous allons insérer pour vérifier que les données correspondent avec une méthode describe dont le code est ci dessous.

```

repository > main.repository.php > MainRepository > describe
134
135 private function describe(){
136     $sql = "DESCRIBE $this->table";
137     $resp = $this->connect()->query($sql);
138     $results = $resp->fetchAll(PDO::FETCH_ASSOC);
139     $columns = [];
140     foreach($results as $result){
141         $columns[$result['Field']] = $result;
142     }
143     return $columns;
144 }

```

(Exécutez en debug pas à pas pour comprendre le fonctionnement de cette méthode)

L 149-170 : Nous allons boucler sur les inputs qui sont les données du formulaire pour le vérifier.
L 150 : Nettoyage des inputs (injections SQL, failles XSS)
L 151-155 : Si aucune colonne n'existe dans la table pour cet input ou si la valeur de l'input est null, nous la supprimons du tableau (car inutile pour l'insert en base de données)
L 156-157 : nous récupérons le type de la colonne en base de données
L 158-163 : si le type attendu est int nous le vérifions et si ça ne correspond pas nous ajoutons une erreur dans le tableau pour la colonne correspondante
L 164-169 : idem pour un float
L 171 : nous retournons le tableau d'erreur.

Si aucune erreur n'est retournée au controller (ProductController>create L 62) nous pouvons insérer en base avec la fonction insertOne dont voici le code :

```
repository > main.repository.php > MainRepository > describe
101 function insertOne($fields = []){ //TODO insertMany
102     $columns = "";
103     $values = "";
104     if(isset($fields['id'])){
105         unset($fields['id']);
106     }
107     $valuesToBind = array();
108     foreach ($fields as $k => $v) {
109         $columns .= $k . ",";
110         $values .= "?,";
111         array_push($valuesToBind, $v);
112     }
113     $columns = trim($columns, ',');
114     $values = trim($values, ',');
115     $sql = "INSERT INTO $this->table ($columns) VALUES ($values)";
116     $statement = $this->connect()->prepare($sql);
117     $result = $statement->execute($valuesToBind);
118     $test = $statement->rowCount() == 1;
119     if($result && $test){
120         $insertedId = $this->db->lastInsertId();
121         $fields['id'] = $insertedId;
122         $entityClass = $this->entity;
123         $entity = new $entityClass($fields);
124         return $entity;
125     }
126     return false;
127 }
```

L 101 : \$fields contiendra les données nettoyées et validées pour l'insertion.
L 102-115 : Nous construisons la requête SQL pour l'insertion.
L 116-117 : Nous préparons et exécutons la requête.
L 118 : Nous vérifions qu'une ligne a effectivement été insérée.
L 119-126 : Si la requête s'est bien passée, nous retournons une instance de la classe entity Product correspondant à la ligne insérée sinon nous retournons false.
(Exécutez en debug pas à pas pour comprendre le fonctionnement de cette méthode)

Testez le formulaire (route /product/create) avec un jeu de test couvrant tous les cas

- 1. sans fichier image avec un mauvais prix puis avec un bon prix (second submit une fois le prix corrigé)
- 2. sans fichier image avec un bon prix directement
- 3. avec fichier image et un mauvais prix puis avec un bon prix (second submit une fois le prix corrigé)
- 4. avec fichier image et bon prix directement
- 5. en cliquant sur annuler

Vous pouvez ajouter un bouton « nouveau produit » sur la route /product (liste des produits)

```
template > product > index.view.php
1 | <h1 class="d-inline">
2 |     Liste des produits
3 | </h1>
4 | <a href="product/create" class="btn btn-sm btn-success mb-3">Nouveau Produit</a>
5 | <ul>
```

Un peu de cosmétique sur index.php (bootstrap + index.css)

```
43+ <link rel="stylesheet" href="https://cdn.jsdelivr.net/npm/bootstrap@4.6.0/dist/css/bootstrap.min.css"
44+     integrity="sha384-B0vP5xmATw1+K9KRQjQERjvTumQW0nPEzvF6L/Z6nronJ3oUOFUfPcJEUQouq2+1" crossorigin="anonymous"
45+ <link rel="stylesheet" href="/assets/css/index.css">
46+
47+ <body>
48+     <div class="container-fluid">
49+         <header>
50+             HEADER
51+         </header>
52+
53+         <main>
54+             <?= $router->render() ?>
55+         </main>
56+
57+         <footer>
58+             FOOTER
59+         </footer>
60+     </div>
61+ </body>
62+
```

Le petit fichier css

```
assets > css > # index.css > textarea
1 | textarea {
2 |     resize: none;
3 | }
```