À présent, le formulaire nous permet bien d'ajouter des enregistrements en base de données.

1. Afficher le détail d'un produit

Nous allons maintenant permettre la consultation d'un produit en particulier. Pour ce faire, nous allons donner la possibilité à l'utilisateur de cliquer sur le bouton « Voir le produit » pour le rediriger vers une page contenant tout le détail.

Commencer par créer un nouveau contrôleur « ShowProductController » dans lequel il faut ajouter une nouvelle fonction « showProductController » (attention à la casse). Cette fonction prendra en paramètre l'objet « twig » et l'objet « pdo ».

Dans la fonction du contrôleur, voici le code à écrire :

```
function showProductController($twig, $db) {

    include_once '../src/model/ProductModel.php';

    $product = null;
    if (isset($_GET['product'])) {
        $product = getOneProduct($db, $_GET['product']);
    }

    echo $twig->render('show_product.html.twig', [
        'product' => $product
    ]);

}
```

Explications :

- La ligne « include_once » permet d'importer le fichier « ProductModel.php »
- Par défaut, la variable « $product » est définie sur nulle
- La condition permet de vérifier que l'identifiant du produit est bien défini dans l'URL. S'il est bien défini, la variable « $product » reçoit le résultat de la requête SQL effectuée dans la fonction « getOneProduct() ».
- Il est à savoir que la requête SQL effectuée par la fonction « getOneProduct() » peut renvoyer la valeur « false » si l'identifiant n'existe pas dans la base de données.
- La fonction « getOneProduct() » est une fonction que nous avons déjà créée au début de ce chapitre.
- Enfin, la ligne contenant « echo » se charge d'afficher notre vue Twig tout en lui passant la variable « $product » qui contient l'ensemble des données de la requête SQL.

Créer la vue :

```
{% extends "layout.html.twig" %}

{% block title %}Fiche de produit{% endblock %}

{% block content %}
  <div class="container">
    {% if product %}
      <div class="col-12">
        <h1 class="mb-5">{{ product.label }}</h1>
      </div>
      <div class="row">
        <div class="col-md-4">
          <div class="card p-2">
            <img src="https://fakeimg.pl/350x200/?text=Image" class="img-fluid
rounded-start" alt="...">
          </div>
        </div>

        <div class="col-md-5 text-secondary">
          <p>{{ product.description }}</p>
        </div>

        <div class="col-md-3 bg-light text-center p-5">
          <h5 class="h1 text-primary">{{ product.price }} €</h5>
        </div>
      </div>
    {% else %}
      <p>Aucun produit n'est défini !</p>
    {% endif %}
  </div>
{% endblock %}
```

Explications :

- « {% if product %} » se charge ici de vérifier que la valeur de la variable passée dans le contrôleur n'est pas « null » ni égale à « false ». Dans ce cas les informations du produit peuvent être affichées.
- Dans le cas où la variable « product » est « null » ou égale à « false », un message indique que le produit n'existe pas.

N'oubliez pas d'ajouter une nouvelle route, celle-ci se nommera « showProduct » et fera référence à la fonction du contrôleur que nous venons de créer.

Essayez maintenant d'accéder à l'URL en indiquant les paramètres suivants : ?page=showProduct&product=1
Vous devriez voir apparaitre le détail du produit.

© 2022 – Formateur : Grégory Boudringhin

Vous pouvez également renseigner un identifiant de produit qui n'existe pas pour observer le comportement du code.
Pour relier chaque produit de notre liste avec sa page de détail, nous allons modifier le bouton dans le fichier home.html.twig. Faites les ajustements nécessaires en vous appuyant de l'extrait du code suivant.

Extrait du code :

```
...
<div class="card-body">
        <h5 class="card-title">{{ product.label }}</h5>
        <p class="card-text">{{ product.description }}</p>
        <a href="?page=showProduct&product={{ product.id }}" class="btn btn-
primary">Voir le produit</a>
</div>
...
```

2. Gestion des messages d'erreurs et de succès

Actuellement la gestion des erreurs de nos formulaires ou de nos requêtes n'est pas forcément très esthétique.

Nous allons améliorer le formulaire d'ajout de produit en intégrant un système qui va gérer l'affichage des messages. L'idée étant d'afficher plus esthétiquement les messages de succès et d'erreur pour avertir efficacement l'utilisateur de notre site.

Dans le fichier « AddProductController.php » ajustez les lignes de code pour obtenir ce résultat :

```
function addProductController($twig, $db) {

    include_once '../src/model/ProductModel.php';

    $form = [];

    if (isset($_POST['btnAddProduct'])) {
        $label = htmlspecialchars($_POST['productLabel']);
        $description = htmlspecialchars($_POST['productDescription']);
        $price = htmlspecialchars($_POST['productPrice']);

        if(empty($price)) {
            $price = 0.00;
        }

        $category = htmlspecialchars($_POST['productCategory']);
```


© 2022 – Formateur : Grégory Boudringhin

```
        if (!empty($label) && !empty($description) && !empty($category)) {
            $form = [
                'state' => 'success',
                'message' => 'Votre produit a bien été ajouté !'
            ];
            saveProduct($db, $label, $description, $price, $category);
        } else {
            $form = [
                'state' => 'danger',
                'message' => 'Votre produit n\'a pas été ajouté car les champs obligatoires
n\'ont pas été remplis !'
            ];
        }
    }

    echo $twig->render('form_product.html.twig', [
        'form' => $form
    ]);
}
```

Les modifications apportées sont très simples.

Explication :

- Une variable « $form » est créée et définie en tant que tableau vide « [] ».
- Dans les deux cas de la condition, la variable « $form » est complétée avec des informations que nous pourrons réutiliser dans la vue Twig.
- Ici la variable « $form » se constitue d'une clé « state » pour définir l'état du formulaire. Les valeurs utilisées correspondent aux variables CSS (success, warning, danger, etc). Le tableau est aussi constitué d'une clé « message » dans laquelle nous renseignerons un message personnalisé pour informer l'utilisateur.
- Enfin, la variable « $form » est passée à la vue Twig sous le nom « form ».

Vous pouvez maintenant essayer d'ajouter un produit, un message devrait s'afficher pour informer que le produit a bien été ajouté. Vous pouvez également tester avec un champ manquant, un message d'erreur s'affichera pour l'indiquer.

Note :
Vous pouvez améliorer cette partie en ajoutant plus de contrôle pour personnaliser le message pour chaque cas.
Par exemple : la taille du libellé, etc.

Afin de clôturer cette partie de chapitre, vous allez mettre en pratique ce que nous venons de voir.

© 2022 – Formateur : Grégory Boudringhin

Exercice pratique :

1. Créer une nouvelle page permettant d'ajouter des catégories en base de données. La page sera accessible grâce à la route « addCategory ».
   Le formulaire se constitue d'un champ texte permettant de renseigner le libellé de la catégorie (« label »).
   Les requêtes SQL liées à la catégorie devront être placées dans le fichier « CategoryModel.php » au même titre que ce qui a été fait pour les produits.

2. Modifier le formulaire d'ajout de produit (la page « addProduct ») afin que la liste des catégories se modifie automatique lorsque nous en ajoutons.
   La liste de ces catégories se retrouve sous la forme d'un champ de sélection.

3. Sur la page d'accueil, plus particulièrement dans la liste des produits, ajouter le libellé de la catégorie au-dessus du nom du produit.

4. Intégrez un système de message pour traiter les erreurs du formulaire.