Portrait Florian

Raad Josué

Projet de WEB

Rapport de projet sur la création d'un site web avec le framework Symfony

Université de Poitiers - L3 informatique - Technologies du Web 2

Année universitaire 2020 - 2021

Sommaire

Le site

Installation

Pour démarrer le site sur un hôte local, se placer avec un terminal dans `/site/public/` et exécuter la commande `symfony serve -d`. Ensuite se rendre sur **`http://localhost:8000/`** .

Le port par défaut est 8000. Pour arrêter le serveur : `symfony server:stop`.

Pensez à activer sqlite dans `php.ini` (`extension=pdo_sqlite`)

Afin de changer l'utilisateur actuel vous pouvez vous rendre dans le fichier `/site/config/services.yaml` et modifier le paramètre `currentUser` (0 : anonyme, 1 : admin, 2 : Gilles, …)

Liens

Accueil - (/)
Error - (/error)
Template menu - (/menu)
Template header - (/header)
Connexion - (/login)
Déconnexion - (/logout)
Service inversion - (/reverse/{str})

Affichage de la liste des produits - (/produit/list)
Affichage produits pour l'administrateur - (/produit/listAdmin)
Créer un nouveau produit - (/produit/new)
Editer le produits sélectionné - (/produit/edit/{id})(2)

Affichage du panier - (/panier)
Ajouter les produits au panier - (/panier/add)(1)
Supprimer un produit du panier - (/panier/delete{id})(2)
Vider le Panier - (/panier/delete)
Passer commande - (/panier/acheter)

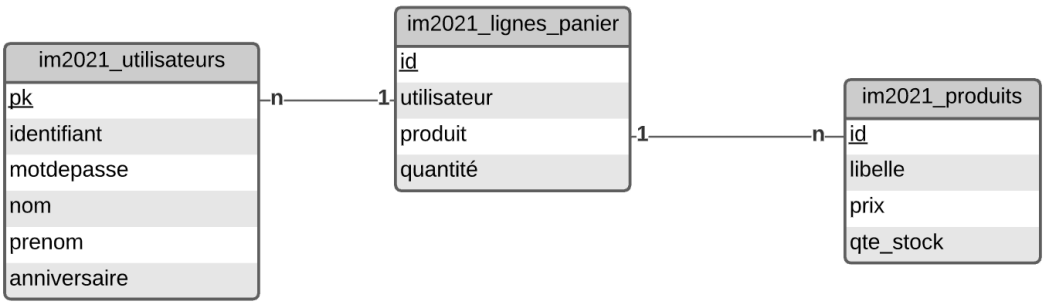
Créer un compte - (/utilisateur/create_account)
Modifier son profil - (/utilisateur/edit)
Supprimer un utilisateur - (/utilisateur/delete/{id})(2)

Les liens peuvent être :

- Accessible par tout le monde
- Accessible seulement par un utilisateur authentifié
- Accessible seulement par l'admin
- Accessible seulement par un anonyme
- (1) Accessible uniquement par envoie d'un formulaire
- (2) Accessible uniquement si l'id est valide

La base de données

Schéma



La table `im2021_lignes_panier` est une table de jointure entre les *utilisateurs* et les *produits*. Pour rendre le code plus simple avec symfony chaque ligne du panier a une clé primaire *id* et les couples (utilisateur, produit) sont des clés étrangères uniques.

Un utilisateur crée une ou plusieurs lignes panier, et une ligne panier n'est créé que par un et un seul utilisateur (cardinalités **1,n** et **1,1**).
Un produit apparaît dans une ou plusieurs lignes panier, et dans une ligne panier n'apparaît qu'un et un seul produit (cardinalités **1,n** et **1,1**).

Doctrine

Toutes les entités ont été créées via le terminal et des annotations supplémentaires ont été ajoutées afin de correspondre au mieux au sujet.

Les noms de la plupart des colonnes ont été définis ainsi que les champs uniques et possiblement null, comme cela : `@ORM\Column([...], unique=true)`

Pour la table ligne_panier, nous avons dû ajouter manuellement `referencedColumnName="pk"` afin que la clé primaire de Utilisateur soit reconnue.

Nous avons défini une contrainte sur la table ligne_panier pour que les couples (utilisateur, produit) soient uniques. Mais on garde la clé primaire afin de pouvoir facilement identifier une ligne (pour simplifier les routes par exemple) :

```
@ORM\Table([...], uniqueConstraints={@UniqueConstraint(name="usr_prod",
columns={"utilisateur","produit"})})
```

Organisation du code

Liste de nos fichiers dans le site (créés et modifiés)

```
./site:
|   .env
|
|---config
|   |   services.yaml
|
|---public
|   |   admin.png
|   |   anon.png
|   |   footer.png
|   |   index.php
|   |   logged.png
|   |
|   |---css
|   |   |   global.css
|   |   |   index.css
|   |   |   site.css
|
|---sqlite
|   |   projectBase.db
|
|---src
|   |---Controller
|   |   |   PanierController.php
|   |   |   ProduitController.php
|   |   |   SiteController.php
|   |   |   UtilisateurController.php
|   |
|   |---Entity
|   |   |   LignePanier.php
|   |   |   Produit.php
|   |   |   Utilisateur.php
|   |
|   |---Form
|   |   |   LignePanierType.php
|   |   |   UtilisateurType.php
|   |   |   ProduitType.php
|   |
|   |---Repository
|   |   |   LignePanierRepository.php
|   |   |   ProduitRepository.php
|   |   |   UtilisateurRepository.php
|   |
|   |---Service
|   |   |   UserAuth.php
|   |   |   ReverseService.php
|
|---templates
|   |---default
|   |   |   _display_flashes.html.twig
|   |   |   _footer.html.twig
|   |   |   _header.html.twig
|   |   |   _menu.html.twig
|   |   |   base.html.twig
|   |   |   site.html.twig
|   |
|   |---panier
|   |   |   show.html.twig
|   |
|   |---produit
|   |   |   edit.html.twig
|   |   |   list.html.twig
|   |   |   list_admin.html.twig
|   |   |   new.html.twig
|   |
|   |---site
|   |   |   index.html.twig
|   |   |   login.html.twig
|   |   |   reverse.html.twig
|   |
|   |---utilisateur
|   |   |   _form.html.twig
|   |   |   create_account.html.twig
|   |   |   edit.html.twig
|   |   |   list.html.twig
```

Config : services.yaml permet de changer l'utilisateur du site

Sqlite : nous retrouvons ici le fichier .db de la base de donnée sqlLite

Controller : les fichiers controller pour les routes et leurs actions

Entity : les fichiers entités doctrine de la base de donnée

Form : les fichiers générés par symfony pour la création des formulaires

Repository : les fichiers repository générés par doctrine

Service : on retrouve le service symfony qui permet d'inverser la chaîne en entrée

Templates : ici se trouvent les vues de chaque page

Public : c'est ici que l'on trouve les images et le css du site

Gestion de l’utilisateur actuel

Nous avons créé un service permettant de récupérer l’utilisateur actuel défini dans le fichier `services.yaml`. Ce service `‘UserAuth’` permet de connaître la nature de l’utilisateur (anonyme, admin, authentifié).

Il est instancié au début de chaque contrôleur qui en a besoin. Nous aurions tout aussi bien pu le passer en paramètre des actions, mais la quasi-totalité des actions y font appel dès la première ligne, donc c’était plus pratique de l’instancier dans les constructeurs.

Les contrôleurs et routes

Nous avons 4 contrôleurs principaux, chacun gérant une partie du site et les routes qui y sont associées. Dans le constructeur de chaque contrôleur, on instancie le service d’authentification pour y accéder plus facilement.

- SiteController gère l’accueil, l’authentification, l'exemple de service, les fragments de template, et les erreurs.
- UtilisateurController, PanierController, et ProduitController se chargent de leurs routes et formulaires respectifs.

Au début de chaque action, on vérifie que l’utilisateur actuel a le droit d’emprunter la route.

Les templates twig

Les templates sont organisés dans des dossiers qui correspondent aux contrôleurs.

Dans le dossier /default sont rangées les modèles de base et du site, ainsi que les fragments de template.

Les fragments du header et du menu sont associés à des routes On aurait pu écrire 'en dur' les actions à appeler pour afficher les fragments de template. Cela aurait évité que l'utilisateur puisse accéder à '/menu' par exemple. Mais par soucis de simplicité du code on a opté pour l'affichage des fragments via les routes

Formulaires et validation

Gestion par symfony

La plupart des formulaires sont gérés par doctrine et ont été créés dans le terminal. Une partie de la validation se fait par symfony dans le formulaire lui-même (champs requis, format d’entrée …).

L’autre partie de la validation est faite par des asserts.

Nous avons ajouté des asserts pour les tables, afin de vérifier / limiter les entrées dans celles-ci (vérifications liés à la classe et non au formulaire), par exemple :

```
@Assert\Type (type="alnum", message="Identifiant alphanumérique uniquement")
```

Au moment de la validation lors de la réception d’un formulaire, ces assertions sont testées avant que les entités soient modifiées et ne provoquent une erreur. On peut définir un message qui sera affiché à l’utilisateur.

Si malgré cela des données incohérentes avec la base arrivent à passer, alors une exception est levée par symfony et la base reste intacte

Gestion manuelle

Dans le cas du formulaire personnalisé de la liste des produits à ajouter au panier, il s'agit du seul point d'entrée des données dans la base pour la table *lignes_panier* (les autres ne sont que des suppressions) . Ainsi on a seulement besoin de vérifier les données dans le contrôleur correspondant (`PanierController::addPanierAction`).

On se contente de vérifier que le produit est en stock et que l'on a pas déjà ce produit dans le panier.

Si on détecte que le formulaire est mal remplis (produit déjà dans le panier) on redirige vers le panier afin de plus simplement modifier l'erreur (au lieu de lever une exception)

Ainsi on ne se retrouve jamais dans le cas où on doit afficher les valeurs précédemment saisies par l'utilisateur dans le cas du formulaire manuel.

Création d'un service dans Symfony

L'autowiring permet au framework de connaître les classes de service (du moment qu'elles sont dans les bons dossiers comme défini dans les fichiers de configuration).

Ainsi, pour créer un service, on définit une classe php dans le dossier `'Services'` de `'src'`.

Pour utiliser un service, il suffit de passer en paramètre d'une fonction la classe du service. Symfony se charge de récupérer une instance de cette classe pour qu'elle puisse être utilisée directement dans la fonction.

Les services dans notre site sont `UserAuth` (service gérant l'authentification) et `ReverseService` (service simple demandé dans le sujet).

Points particuliers

Choix de design dans le projet :

- Au lieu de lever une exception 404 lorsqu'un utilisateur accède à une page interdite, on lance une redirection vers la route `/error` qui peut se charger de lever l'exception. Mais actuellement cette route redirige vers la page d'accueil avec un simple message flash. Pour que le comportement corresponde à ce qui est demandé dans le sujet il suffit de décommenter la première ligne de l'action liée à la route.

- Nous avons ajouté la possibilité pour un admin d'éditer les stocks des produits (en plus de pouvoir ajouter de nouveaux produits). Nous trouvons cela plus cohérent que les stocks puissent augmenter. Par contre, il ne peut pas éditer les caractéristiques des produits.

- On ne vérifie pas systématiquement les id manuellement dans chaque route qui en a besoin. En effet Symfony produit automatiquement une erreur 404 lorsque l'id est inexistant dans la base. (cf. `UtilisateurController::deleteAction` pour une gestion manuelle de l'id)

- Pas besoin de traitement supplémentaire pour le formulaire d'ajout au panier ni d'annotation d'assertion dans la BDD, car si l'utilisateur fait une requête incohérente, elle n'est pas prise en compte. En somme la route /panier/add est le seul point duquel on peut entrer des données dans la table lignes_panier. Et les données sont toujours vérifiées.

- On a inclus le fragment de template `_display_flashes.html.twig` au template `site.html.twig` afin que toutes les pages du site puissent afficher des messages flash, les erreurs sont stylées en rouge et les infos en vert par `site.css`.

- On a choisi une restriction alphanumérique pour l'identifiant des utilisateurs et pas de restrictions pour les noms/prénoms. On peut supprimer son nom, entrer des symboles etc… Tout est permis !

- Nous n'avons modifié que `config/services.yaml`. Il est envisageable de modifier également `config/package/dev/web_profier.yaml` pour intercepter les redirections, mais cela n'a pas été nécessaire dans le cadre du projet.

- Nous avons fait preuve *d'originalité* dans la conception du header et du footer du site.

- Quelques interrogations à propos du projet de WEB :
    - Avec la configuration actuelle du sujet, on ne peut pas avoir un admin que celui par défaut sur le site. (sauf en éditant directement la BDD). Est-ce que nous avons bien compris le sujet ? Y a-t-il quelque chose qui nous échappe ? Un admin ne devrait-il pas être capable de créer un autre admin ou modifier son mot de passe ?

Points non opérationnels

Génération d'un code sql identique à celui fourni

Nous n'avons pas réussi à générer un code identique à celui fourni en sql

Nous avons suivi la documentation de doctrine pour le mapping des types sql vers les entités doctrine.

Seulement sur les types nous n'arrivons pas à indiquer à doctrine que "pk" doit être un int(11) ou encore que isadmin est un tinyint(1).

`columnDefinition="…"` produit une erreur de syntaxe lors de la migration

Pour les commentaires, `options{"comments":"commentaires"}` ne marche pas non plus.