Projet_Web2

Enseignants: M.Subrenant & Mme. Zrour

Ce projet est à réaliser avant le 16 avril 2021 et est à déposer sur Updago dans le cadre de l'Unité d'Enseignement **Technologies du Web 2**. Ce projet porte sur l'utilisation de Php, Symfony, Twig, et certaines libraires.

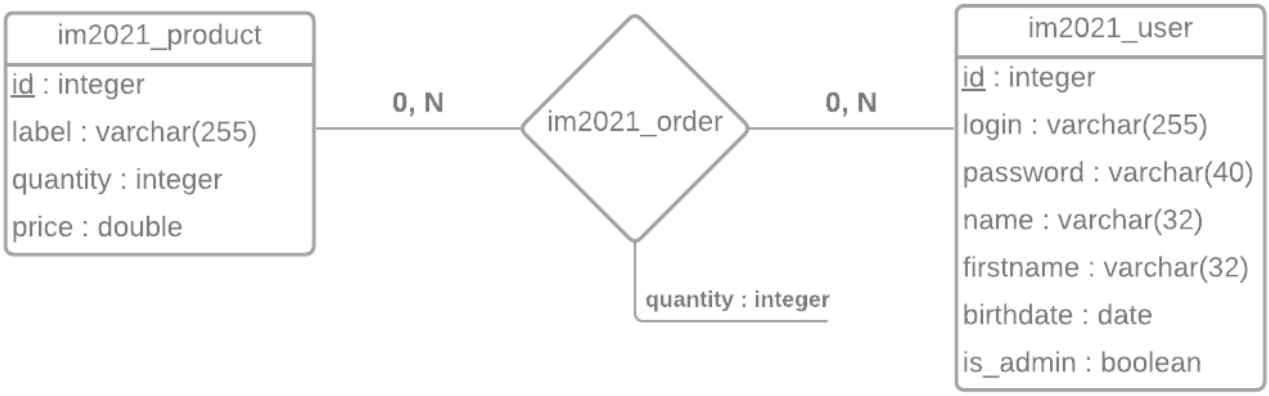
Notre groupe est composé des membres suivants :

- Vincent COMMIN
- Louis LEENART

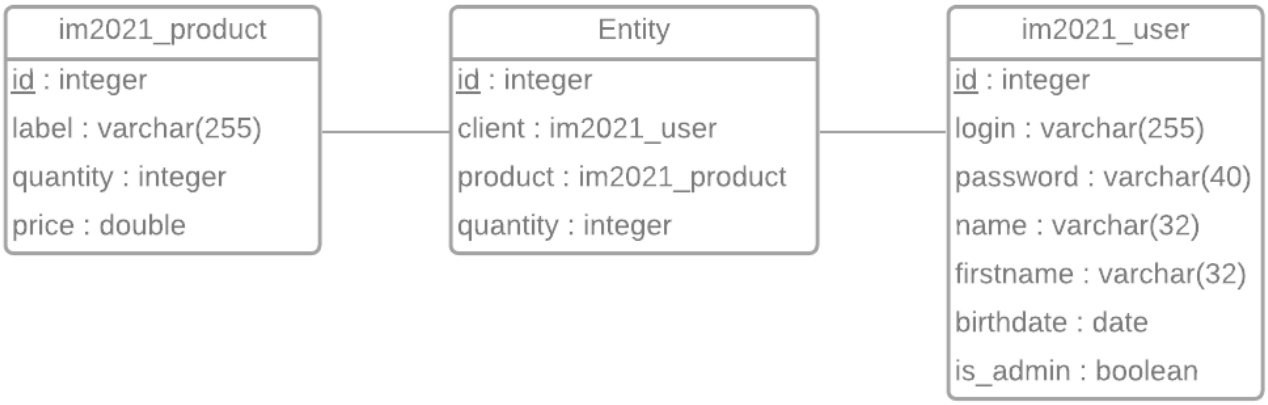
SOMMAIRE

Schémas de la base de données utilisée

i. Entité association



ii. Relationnel



Entity.client est une clé étrangère de **im2021_user**

Entity.product est une clé étrangère de **im2021_product**

Symfony ajoutant une clé primaire automatiquement, nous avons dû ajouter une contrainte unique sur les champs `client` et `produit` afin d'en faire notre clé primaire.

Hiérarchie du projet

```
src/
|   Kernel.php
|
├──Controller
|    ├── .gitignore
|    ├── AccountController.php     # Contient toutes les actions relatives
aux comptes
|    ├── AdminController.php       # Contient les actions accessibles
uniquement par l'admin
|    ├── MenuController.php        # Contient l'action pour le menu
(inclut dans le twig)
|    └── ProductController.php     # Contient les actions relatives à la
gestion des produits
|
├──DataFixtures
|    └── AppFixtures.php
|
├──Entity                          # Contient les classes relatives aux
tables de la base de données
|    ├── .gitignore
|    ├── Order.php
|    ├── Product.php
|    └── User.php
|
├──Form                            # Contient les formulaires des entités
|    ├── OrderType.php
|    ├── ProductType.php
|    └── UserType.php
|
```

```
├──Repository                       # Contient les répertoires des entités
│    ├── .gitignore
│    ├── OrderRepository.php
│    ├── ProductRepository.php
│    └── UserRepository.php
│
└──Services                         # Contient les services
     ├── EmptyOrders.php
     └── InvertString.php
```

```
templates/
├── base.html.twig
│
├──commons                          # Contient tous les templates twig
desquelles nous allons hériter
│    ├── _base.html.twig
│    ├── _body.html.twig
│    ├── _flash.html.twig
│    ├── _footer.html.twig
│    ├── _layout.html.twig
│    └── _menu.html.twig
│
├──menu                             # Contient le template du menu
│    └── index.html.twig
│
└──vues                             # Contient les vues des actions
    ├──account                      # du controller account
    │    ├── connect.html.twig
    │    ├── createAccount.html.twig
    │    ├── disconnect.html.twig
    │    ├── editProfile.html.twig
    │    └── welcome.html.twig
    │
    ├──admin                        # du controller admin
    │    ├── addProduct.html.twig
    │    ├── editProduct.html.twig
    │    ├── editProducts.html.twig
    │    ├── editUser.html.twig
    │    └── listUsers.html.twig
    │
    └──product                      # du controller product
         ├── listOrders.html.twig
         ├── orders.html.twig
         └── productList.html.twig
```

```
public/
├──css                              # Contient toutes les feuilles de
styles globales et pour chaque controller
```

```
|   ├──account
|   ├──admin
|   └──product
└──images                          # Contient les images du site
```

Création d'un service sur Symfony

Avant de procéder à la création d'un service sous Symfony, nous allons expliquer briévement ce à quoi cela correspond.

Un service (selon M. Achref El Mouelhi) est une classe PHP ne réalisant qu'une seule fonctionnalité (envoi de mail, manipulation dans la base de donnée, ...) qui se veut accessible partout dans le code et injectable dans les classes qui en ont besoin. Il a un identifiant qui est son nom de classe.

Pour notre projet nous avons décidé de faire un service inversant une chaîne de caractères et de l'afficher sur la page d'accueil.

Dans un premier temps créé notre service dans le dossier `src/Services` :

```php
// src/Services/InvertString.php
namespace App\Services;

class InvertString
{
    // retourne la chaîne de caractère inversée
    public function getInvertString(string $str) : string
    {
        return strrev($str);
    }
}
```

Ensuite, pour pouvoir utiliser le service nous devons passer un objet du type du service en paramètre d'une action qui souhaite l'utiliser :

```php
// src/Controller/unController.php
public function action(InvertString $invertString)
{
    ...

    $inverted = $invertString->getInvertString("string"); // retourne
"gnirts"

    ...
}
```

Points particuliers du framework

Nous n'avons pas eu de gros problèmes durant la réalisation de ce projet hormis aux points abordés ci-dessous.

Ajout d'un service pour vider le panier

Durant le développement de notre site, nous nous sommes confrontés à un dilemme. Nous devions vider le panier d'un utilisateur mais cette action pouvait être réalisée à plusieurs endroits. Dans un premier temps nous avions donc dupliqué un peu de code car la fonction gérant cela devait être appelée dans deux controllers différents. Heureusement, grâce au sujet nous demandant de créer un service, nous avons pu nous renseigner et créer un deuxième service vidant le panier.

```
class EmptyOrders
{
    /**
     * @var EntityManagerInterface
     */
    private $em;

    public function __construct(EntityManagerInterface $em)
    {
        $this->em = $em; // On utilise une instance de l'entity manager
pour pouvoir gérer doctrine
    }

    public function emptyOrders($id, $isBuyed = false)
    {
        ... // vide le panier suivant l'id et si les produits sont achetés
    }
}
```

Formulaire de la liste des produits

Un autre point qui nous a posé problème est la mise en place de la page principale de notre site, l'ajout au panier des articles. En effet, mettre en place un formulaire 'Symfony' permettant de gérer non pas un seul article, mais une liste entière, a été source de nombreux essais et recherches en vain. Nous avons alors décidé d'utiliser la solution moins 'propre' en créant un formulaire classique à la main. Le détail de l'action est alors décrit dans le fichier `src/Controller/ProductController.php` dans `productListAction`.

Service de mailing

Pour ajouter l'action d'envoyer un mail à l'utilisateur contenant le nombre d'articles en stock, nous avons utilisé la librairie `SwiftMailer`. En effet, sur la page listant les produits pour les ajouter au panier, nous avons ajouter un formulaire pour saisir une adresse mail vers laquelle nous envoyons le message. L'action qui s'occupe de gérer ce formulaire est dans `src/Controller/ProductController.php` dans `mailProductAction`. La librairie s'occupe de l'intégralité des détails d'envoie du mail, cependant nous avons 'mis en clair' les identifiants de l'adresse mail créée pour l'occasion (pour ce projet). Il serait idéal, pour ne pas dire impératif, de stocker ces identifiants autre part pour ne pas que n'importe qui ait accès à la boite mail.