R4.01 - TP 03 – Contrôleur, Service, Session
1 séance

3.0. Travail à réaliser – Vue d'ensemble

Pour mettre en place la notion de « panier » sur notre site, nous allons :

- Développer une nouvelle **fonctionnalité métier**, le panier, dans un service **`PanierService`** qui devra :
  - Proposer les opérations de base à réaliser sur un panier :
    - Ajouter/Enlever un produit au/du panier
    - Supprimer un produit du panier
    - Calculer le montant du panier, le nombre de produits dans le panier
    - Vider tout le panier
    - Consulter le contenu du panier
  - Assurer la persistance du panier de l'utilisateur en stockant son contenu (identifiants des Produits concernés et quantité) en **session**
- Mettre en place les routes, contrôleurs et le *template* nécessaires pour offrir à l'utilisateur une interface qui lui permettra d'interagir avec ce service
- Exemple : https://iut2-dg-scolarite.iut2.univ-grenoble-alpes.fr/info/R401/

3.1. Développez le service PanierService

Le service **`PanierService`** va devoir utiliser le service **`BoutiqueService`** afin d'accéder aux Produits. Il va également devoir aussi utiliser la **`Session`** afin de pouvoir stocker le contenu du panier en session.
On vous fournit ci-dessous l'interface de ce nouveau service qu'il vous faudra compléter (téléchargez le fichier `PanierService.php` sur Chamilo) :

```php
<?php // src/Service/PanierService.php
namespace App\Service;
use Symfony\Component\HttpFoundation\RequestStack;
use App\Service\BoutiqueService;
// Service pour manipuler le panier et le stocker en session
class PanierService {
    private $session;   // Le service session
    private $boutique;  // Le service boutique
    private $panier;    // Tableau associatif, la clé est un idProduit, la valeur une quantité
                        //   $this->panier[$idProduit] = quantité du produit $idProduit
    const PANIER_SESSION = 'panier'; // Nom de la var. de session pour persister $this->panier
    // Constructeur du service
    public function __construct(RequestStack $requestStack, BoutiqueService $boutique) {
        // Récupération des services session et BoutiqueService
        $this->boutique = $boutique;
        $this->session  = $requestStack->getSession();
        // Récupération du panier en session s'il existe, initialisation à tableau vide sinon
        $this->panier   = // à compléter… ;
    }
    // Ajouter au panier le produit $idProduit en quantite $quantite
    public function ajouterProduit(int $idProduit, int $quantite = 1) : void { // à compléter… }
    // Enlever du panier le produit $idProduit en quantite $quantite
    public function enleverProduit(int $idProduit, int $quantite = 1) : void { // à compléter… }
    // Supprimer le produit $idProduit du panier
    public function supprimerProduit(int $idProduit) : void { // à compléter… }
    // Renvoyer le montant total du panier
    public function getTotal() : float { // à compléter… }
    // Renvoyer le nombre de produits dans le panier
    public function getNombreProduits() : int { // à compléter… }
    // Vider complètement le panier
    public function vider() : void { // à compléter… }
    // Renvoyer le contenu du panier (dans le but de l'afficher dans un template)
    //   => un tableau d'éléments [ "produit" => un objet produit, "quantite" => sa quantite ]
    public function getContenu() : array { // à compléter… }
}
```

Lisez les indications page suivante avant de vous lancer dans le code !

Quelques indications utiles :

- Rappels sur la manipulation des tableaux en PHP… Soit `$tableau` un tableau PHP :
  - `foreach($tableau as $cle => $val)` est l'itération qui permet de parcourir `$tableau`
  - `isset($tableau[$i])` renvoie vrai si l'élément d'indice `$i` est bien présent dans `$tableau`, faux sinon
  - `unset($tableau[$i])` supprime l'élément d'indice `$i` de `$tableau`
  - `$tableau = []` vide le contenu de `$tableau`
- Pour l'utilisation de la **session** en symfony, voir **cours 03, page 11.**

3.2. Routes, Contrôleurs, *Template*

Dans le répertoire **`src/Controller`**, créez une nouvelle classe contrôleur appelé **`PanierController`** (en utilisant la commande **`php bin/console make:controller`**)
Cette classe contiendra tous les contrôleurs (méthodes) permettant à l'utilisateur d'interagir avec son panier *via* des requêtes HTTP. Ces contrôleurs devront bien sûr utiliser les services **`PanierService`** et **`BoutiqueService`**. Vous ferez en sorte de lever des exceptions si une requête n'a pas de sens, par exemple si on essaye d'ajouter au panier un produit dont l'identifiant n'existe pas (et pour déterminer cela, il faudra utiliser le service **`BoutiqueService`**). Pour lever une exception et renvoyer une réponse http 404, voir **cours 03 page 10**.
Dans cette classe contrôleur, vous devrez proposer les routes et contrôleurs (méthodes) suivants :

- Une route **`app_panier_index`** pour l'URL **`/{_locale}/panier/`**
  - Cette route sera associée à la méthode **`index`**
  - La méthode **`index`** devra utiliser le service **`PanierService`** pour récupérer le contenu du panier et le transmettre au *template* **`panier/index.html.twig`**
  - Ce *template* devra afficher le contenu du panier qui lui a été transmis
- Une route **`app_panier_ajouter`** pour l'URL **`/{_locale}/panier/ajouter/{idProduit}/{quantite}`**
  - Cette route sera associée à la méthode **`ajouter`** et devra utiliser le service **`PanierService`** pour ajouter au panier le produit reçu en paramètre, s'il existe (dans la quantité précisée).
  - La méthode **`ajouter`** devra **rediriger** l'utilisateur vers la route **`panier_index`** (voir **cours 03 page 7**)
  - Modifiez le *template* qui affiche les produits (**`boutique/rayon.html.twig`**) afin qu'un clic sur un produit déclenche la route **`app_panier_ajouter`**
  - Modifiez le *template* **`panier/index.html.twig`** pour offrir sur chaque produit du panier un bouton ⊞ qui déclenchera la route **`app_panier_ajouter`**
- Une route **`app_panier_enlever`** pour l'URL **`/{_locale}/panier/enlever/{idProduit}/{quantite}`**
  - Cette route sera associée à la méthode **`enlever`** et devra utiliser le service **`PanierService`** pour enlever au panier le produit reçu en paramètre, s'il existe (dans la quantité précisée).
  - La méthode **`enlever`** devra rediriger l'utilisateur vers la route **`app_panier_index`**
  - Modifiez le *template* **`panier/index.html.twig`** pour offrir sur chaque produit du panier un bouton ⊟ qui déclenchera la route **`app_panier_enlever`**
- Une route **`app_panier_supprimer`** pour l'URL **`/{_locale}/panier/supprimer/{idProduit}`**
  - Cette route sera associée à la méthode **`supprimer`** et devra utiliser le service **`PanierService`** pour enlever au panier le produit reçu en paramètre, s'il existe.
  - L'action **`supprimer`** devra rediriger l'utilisateur vers la route **`panier_index`**
  - Modifiez le *template* **`panier/index.html.twig`** pour offrir sur chaque produit du panier un bouton 🗑 qui déclenchera la route **`app_panier_supprimer`**
- Définissez une route **`app_panier_vider`** pour l'URL **`/{_locale}/panier/vider`**
  - Cette route sera associée à la méthode **`vider`** et devra utiliser le service **`PanierService`** pour vider complètement le panier.
  - L'action **`vider`** devra rediriger l'utilisateur vers la route **`app_panier_index`**
  - Modifiez le *template* **`panier/index.html.twig`** pour offrir un bouton 🗑 qui déclenchera la route **`app_panier_vider`**

Quelques indications utiles :

- Si vous trouvez pénible de devoir rajouter le paramètre `_locale` dans chaque URL d'une nouvelle route à l'intérieur d'une classe contrôleur, vous avez la possibilité de préciser, par une annotation `Route` au niveau de la classe, un préfixe qui se rajoutera automatiquement devant l'URL de chaque route définie dans la classe :
```
#[Route(path: '/{_locale}/panier', requirements: ['_locale'=>'%app.supported_locales%'])]
class PanierController extends AbstractController { … }
```
- Dans une route, on peut contraindre un paramètre et vérifier qu'il respecte bien une expression régulière. Par exemple, pour vérifier qu'un paramètre de type **`id`** doit être composé uniquement de chiffres, on écrirait :
```
#[Route('/afficher/{id}', name: 'une_route', requirements: ['id' => '\d+'])]
```

3.3. Contrôleur Imbriqué dans la Barre de Navigation

On souhaite maintenant rendre dynamique la barre de navigation du site afin que le nombre de produits ajoutés au panier soit en permanence affiché. Pour cela, vous devrez :

- Développer, dans la classe contrôleur **`PanierController`**, un nouveau contrôleur :
  ```
  public function nombreProduits(PanierService $panier): Response { // à compléter… }
  ```

  Ce contrôleur devra renvoyer une réponse contenant uniquement le nombre de produits présents dans le panier.

  Ce contrôleur sera un « contrôleur imbriqué » destiné uniquement à être appelé depuis un *template* et il ne sera donc associé à aucune route (voir **cours 03, pages 15-17**).

  ✋ **Rappel** : un contrôleur peut renvoyer simplement une réponse, sans faire appel à un *template*, en exécutant :
  ```
  return new Response("Hello World") ;
  ```

- Modifier le *template* **`navBar.html.twig`** afin que la barre de navigation comporte un nouveau bouton permettant d'afficher le contenu du panier, et donc de déclencher la route **`app_panier_index`**.

  Le texte de ce bouton devra afficher le nombre de produits dans le panier et ceci pourra être réalisé en appelant, depuis le *template*, le contrôleur imbriqué **`PanierController::nombreProduits`**

  ✋ **Rappel** : nous avions convenu de ne plus traduire les nouvelles pages, mais nous avions aussi convenu que le contenu de la barre de navigation devait, lui, être traduit… Il faudra donc faire le nécessaire (*cf* TP 02) pour pouvoir traduire le texte de ce nouveau bouton de navigation vers le panier !