R4.01 - TP 01 : Route, Contrôleur, Template Twig
1 séance de 2 heures

Suivez scrupuleusement les instructions ci-dessous. Lisez bien ce qui s'affiche et n'hésitez pas à appeler l'enseignant si l'installation semble avoir échoué.

1.1 – Installation du projet Symfony fourni

- **Prérequis : vérifiez que vous disposez bien d'une centaine de Mo disponible dans votre quota disque. Sinon, faites du ménage ou vous ne pourrez pas installer correctement votre projet symfony !**
- Placez-vous dans le répertoire qui contient le projet symfony (version 7.2) qui a déjà été créé pour vous (remplacez **`votre_login`** par votre login UGA habituel) :

  ```
  cd /users/info/pub/2a/R4.01/TP/votre_login
  ```

  C'est dans ce répertoire que vous devez, au fil des séances, développer votre projet. Ne le déplacez pas ! C'est ici qui nous viendrons contrôler et évaluer le travail que vous aurez réalisé en TP.

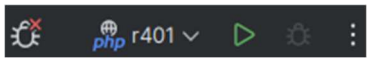
- Depuis votre répertoire, lancez la commande d'installation des dépendances :

  ```
  composer install
  ```

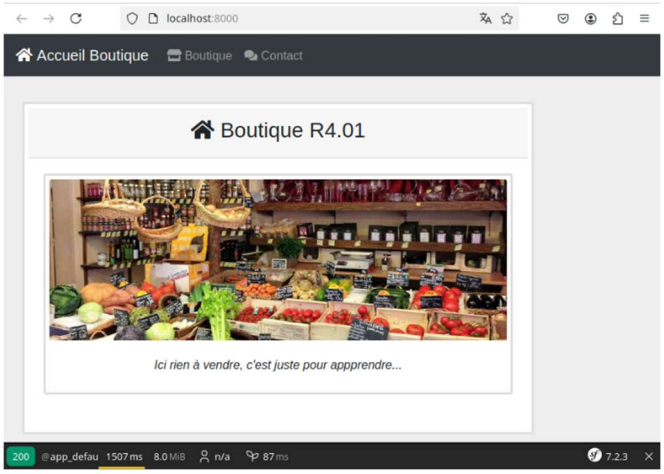
- Lancez l'IDE **`PhpStorm`** avec lequel nous travaillerons. C'est le meilleur pour développer en Symfony, en particulier pour avoir une coloration syntaxique et un formatage du code justes dans les *templates* **`Twig`** (vous constaterez par la suite que c'est très utile) :

  ```
  phpstorm &
  ```

- Ouvrez votre projet : `/users/info/pub/2a/R4.01/TP/votre_login`
- Depuis **`PhpStorm`**, lancez le serveur de développement de php qui fera tourner notre projet. Ce serveur permet de déboguer le code PHP grâce au débogueur très intuitif de PhpStorm (même fonctionnement que sous Clion ou IntelliJ). Cliquez sur la flèche verte :



- Lancez **`firefox`** et naviguez vers l'URL http://localhost:8000/
- Si tout s'est bien passé, vous devriez voir dans votre navigateur la page d'accueil par défaut du squelette de projet Symfony fourni :



Travail à réaliser au fil des TP : site de e-commerce

Le site devra offrir un *front-office* permettant à des utilisateurs de :
- Consulter le catalogue d'un commerçant : catégories de produits, liste des produits d'une catégorie donnée, caractéristiques d'un produit (libellé, photo, prix, *etc*…).
- Constituer un « panier » et passer commande en s'inscrivant sur le site.
- Revenir ensuite consulter l'état d'avancement de leurs commandes.

- Une démo du site à réaliser est visible :
  https://iut2-dg-scolarite.iut2.univ-grenoble-alpes.fr/info/R401/

Nota Bene : *L'objectif de ce module n'est pas de consacrer son temps à faire du « design » avec Bootstrap 4 (le framework css utilisé pour ce projet). Vous pouvez donc vous inspirer sans vergogne du code HTML du site de démo dans vos templates* ***Twig***

1.2 - Mise en place du Layout et de sa barre de navigation

- Un exemple de fichier **templates/base.html.twig**, déjà fourni, contient le *layout* des pages de votre site. Ce *layout* « factorise » le code HTML commun à vos différentes pages et définit des blocs qui doivent ensuite être redéfinis dans les *templates* qui hériteront de lui. Sur note exemple, on propose :
  - Un bloc **`title`** pour pouvoir remplir la balise `<title>` dans chaque *template* fils
  - Un bloc **`body`** pour pouvoir remplir la `<div>` qui héberge le contenu principal d'un *template* fils
  - Un bloc **`script`** pour pouvoir ajouter, à chaque *template* fils, les éventuels scripts `js` dont il aurait spécifiquement besoin
  - Un bloc **`style`** pour pouvoir ajouter, à chaque *template* fils, les éventuels styles `css` dont il aurait spécifiquement besoin

  On pourrait bien sûr définir d'autres blocs !

- Par souci de modularité, la barre de navigation n'est pas codée dans **base.html.twig** mais dans un autre *template*, **templates/navbar.html.twig**. Cette barre de navigation est incluse dans le *layout* grâce à une balise **`Twig {% include … %}`**

- Au fur et à mesure que vous créerez les différentes pages de votre site, des liens vers les nouvelles pages devront être correctement mis en place dans la barre de navigation en utilisant des noms de routes **`{{ path("un_nom_de_route", …) }}`**

1.3 - Pages statiques : créer la page contact

- Un squelette de contrôleur **`DefaultController`** a déjà été créé avec la commande :

  `bin/console make:controller` *(ne la relancez pas pour l'instant !)*

- Cette commande a produit 2 fichiers :
  - Une classe **`DefaultController.php`** dans le répertoire **`src/Controller/`**, qui contient déjà une méthode **`index`** avec sa route associé (en annotation)
  - Un *template* **`index.html.twig`** dans le répertoire **`templates/default`**

- Dans **`src/Controller/DefaultController.php`**, la route **`app_default`** est définie pour que la méthode **`index`** soit déclenchée par l'URL **`/`**
- Le fichier **`templates/default/index.html.twig`** affiche l'image de la page d'accueil de votre site (en héritant de **`base.html.twig`**)
- Dans **`src/Controller/DefaultController.php`**, créez une nouvelle méthode **`contact`** avec une route nommée **`app_default_contact`** associée à l'URL **`/contact`**

- La méthode **`contact`** devra rendre le *template* contact.html.twig que vous créerez dans le répertoire **`templates/default/`** et qui devra afficher (en héritant lui aussi de **`base.html.twig`**) la page de contact de votre site.
- Faites en sorte que, dans la barre de navigation, le lien `contact` du menu permette de naviguer vers cette nouvelle page.

1.4 - Pages dynamiques : catégories et produits par catégorie

Nous n'avons pas encore modélisé les données persistantes de notre application (ce sera au TP4). En attendant, pour pouvoir réaliser une maquette de la partie « boutique » de notre site, on vous fournit un service Symfony (une classe PHP) qui permettra de manipuler des **`Catégories`** de produits et des **`Produits`** (stockés « en dur » dans des tableaux PHP). L'interface de ce service, que vous devrez utiliser, est la suivante :

```php
// Un service pour manipuler le contenu de la Boutique
//  qui est composée de catégories et de produits stockés "en dur"
class BoutiqueService {
    // renvoie toutes les catégories
    public function findAllCategories() : array {...}
    // renvoie la categorie dont id == $idCategorie (null si pas trouvée)
    public function findCategorieById(int $idCategorie) : object {...}
    // renvoie le produit dont id == $idProduit, null si pas trouvé
    public function findProduitById(int $idProduit) : object {...}
    // renvoie untableau de produits dont idCategorie == $idCategorie
    public function findProduitsByCategorie(int $idCategorie) : array {...}
    // renvoie un tableau de produits dont libelle+texte contient $search
    public function findProduitsByLibelleOrTexte(string $search) : array {...}
    // Le catalogue de la boutique, codé en dur dans un tableau associatif
```

Les méthodes fournissent :

- Soit un objet standard PHP qui représente un **`Produit`** ou une **`Catégorie`**
- Soit un tableau PHP d'objets standards de type **`Produit`** ou de **`Catégorie`**

Ces objets sont construits à partir de données stockées « en dur », au format JSON, dans le service lui-même. Observez bien les différents attributs qui sont disponibles dans ces 2 types d'objet :

```php
private $categories = <<<JSON
    [   {
            "id"      : 1,
            "libelle" : "Fruits",
            "visuel"  : "images/categories/fruits.jpg",
            "texte"   : "De la passion ou de ton imagination"
        }, ...
    ]
JSON;
private $produits = <<<JSON
    [   {
            "id"          : 1,
            "idCategorie" : 1,
            "libelle"     : "Pomme",
            "texte"       : "Elle est bonne pour la tienne",
            "visuel"      : "images/produits/pommes.jpg",
            "prix"        : 3.42
        }, ...
    ]
JSON;
}
```

- Le fichier **`BoutiqueService.php`** est disponible dans le répertoire **`src/Service`**

Lorsque vous voudrez utiliser ce service dans une méthode d'un contrôleur, il vous suffira de l'injecter par les paramètres du contrôleur. Par exemple :

```php
namespace App\Controller;
use Symfony\Component\HttpFoundation\Response;
use Symfony\Bundle\FrameworkBundle\Controller\AbstractController;
use Symfony\Component\Routing\Attribute\Route;
use App\Service\BoutiqueService;

class BoutiqueController extends AbstractController {

    #[Route('/boutique', name : 'app_boutique')]
    public function index(BoutiqueService $boutique) : Response {

        // Utiliser le service pour récupérer les catégories
        $categories = $boutique->findAllCategories();

        // Rendre un template auquel on transmet ces catégories
        // …
    }
}
```

- Avec la commande **`bin/console make:controller`** évoquée précédemment, créez un nouveau contrôleur **`BoutiqueController`** qui sera chargé d'afficher la page des « rayons » (toutes les catégories disponibles) et la page de détail d'un rayon (tous les produits d'une catégorie donnée)
- Définissez une route **`app_boutique`** pour l'URL **`/boutique`**. Cette route exécutera la méthode **`index`** du contrôleur **`BoutiqueController`** et devra afficher la liste des **`catégories`** de notre boutique.
- Définissez une route **`app_boutique_rayon`** pour l'URL **`/rayon/{idCategorie}`**. Cette route exécutera la méthode **`rayon(int $idCategorie)`** du contrôleur **`BoutiqueController`** et devra afficher la liste des **`produits`** de la catégorie **`idCategorie`**.
- Ecrivez le *template* **`templates/boutique/index.html.twig`** qui affichera les catégories, chaque catégorie étant un lien pour naviguer vers les produits de cette catégorie
- Ecrivez le *template* **`templates/boutique/rayon.html.twig`** qui affiche les produits d'une catégorie
- Complétez la barre de navigation pour pouvoir atteindre la page **`app_boutique`**