TD Symfony

Étape 5 : Les routes de notre application

Nous allons créer toutes les routes de notre application en commençant par la page d'accueil. Nous allons utiliser un bundle de Symfony, normalement déjà installé par defaut dans notre projet : MakerBundle qui permet de générer des Controller et des Template type.

Pour vérifier que le projet intègre bien ce bundle rendez-vous dans /config/bundles.php. Ce fichier comporte l'ensemble des bundles installés. 'dev' signifie que le bundle est disponible en mode développement, 'all' en mode développement et production également.

config > bundles.php

```php
<?php

return [
    Symfony\Bundle\FrameworkBundle\FrameworkBundle::class => ['all' => true],
    Sensio\Bundle\FrameworkExtraBundle\SensioFrameworkExtraBundle::class => ['all' => true],
    Symfony\Bundle\TwigBundle\TwigBundle::class => ['all' => true],
    Symfony\Bundle\WebProfilerBundle\WebProfilerBundle::class => ['dev' => true, 'test' => true],
    Symfony\Bundle\MonologBundle\MonologBundle::class => ['all' => true],
    Symfony\Bundle\DebugBundle\DebugBundle::class => ['dev' => true],
    Symfony\Bundle\MakerBundle\MakerBundle::class => ['dev' => true],
    Doctrine\Bundle\DoctrineBundle\DoctrineBundle::class => ['all' => true],
    Doctrine\Bundle\MigrationsBundle\DoctrineMigrationsBundle::class => ['all' => true],
    Symfony\Bundle\SecurityBundle\SecurityBundle::class => ['all' => true],
    Twig\Extra\TwigExtraBundle\TwigExtraBundle::class => ['all' => true],
];
```

Nous saisissons le commande suivante dans le terminal VSCode

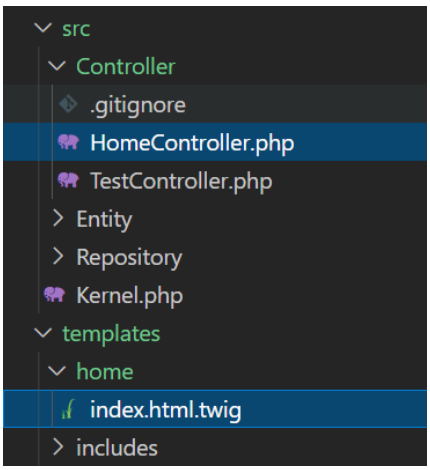
```
td_symfony> symfony console make:controller HomeController
```

...

```
created: src/Controller/HomeController.php
created: templates/home/index.html.twig

Success!
```

Maker a créé un contrôleur et un template associé à l'action index qu'il nous reste à customiser :



Ce contrôleur va permettre d'accéder à la page d'accueil de notre site, nous ajoutons une 2ème route à celle définie par Maker lors de la création afin d'y accéder directement.

src > Controller > HomeController.php > HomeController > index

```
9   class HomeController extends AbstractController
10  {
11      /**
12       * @Route("/home", name="home")
13       * @Route("/", name="root")
14       */
15      public function index(): Response
16      {
```

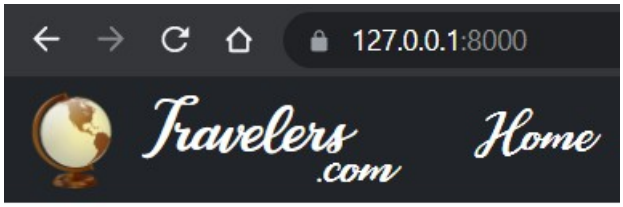
Attention, les 2 routes doivent avoir un name différent.

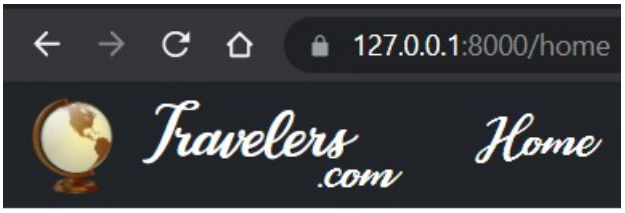
Nous allons modifier 2 liens dans notre menu pour tester ces 2 routes

templates > includes > header.html.twig

```
1   <nav class="navbar navbar-expand-lg navbar-dark bg-dark fixed-top">
2     <div class="container-fluid">
3       <a class="navbar-brand" href="/">
4           <img src="/img/globe-icon.png" alt"logo travelers.com">
5           <h1 id="site_name">Travelers<span>.com</span></h1>
6       </a>
7       <button class="navbar-toggler" type="button" data-bs-toggle="collapse" data-
8         <span class="navbar-toggler-icon"></span>
9       </button>
10      <div class="collapse navbar-collapse" id="navbarSupportedContent">
11        <ul class="navbar-nav me-auto mb-2 mb-lg-0">
12          <li class="nav-item">
13            <a class="nav-link active" aria-current="page" href="/home">Home</a>
14          </li>
15          <li class="nav-item">
16            <a class="nav-link" href="#">Link</a>
17          </li>
```

Les routes fonctionnent.





Par contre, le contenu du template associé n'est pas affiché. Ceci est dû au bloc body que nous avons renommé en content dans la page de base. Pour visualiser le contenu de notre template, il faut modifier le nom du bloc dans celui-ci, nous en profitons pour vider le contenu généré par défaut dans ce bloc, ainsi que le titre de la page (balise title)

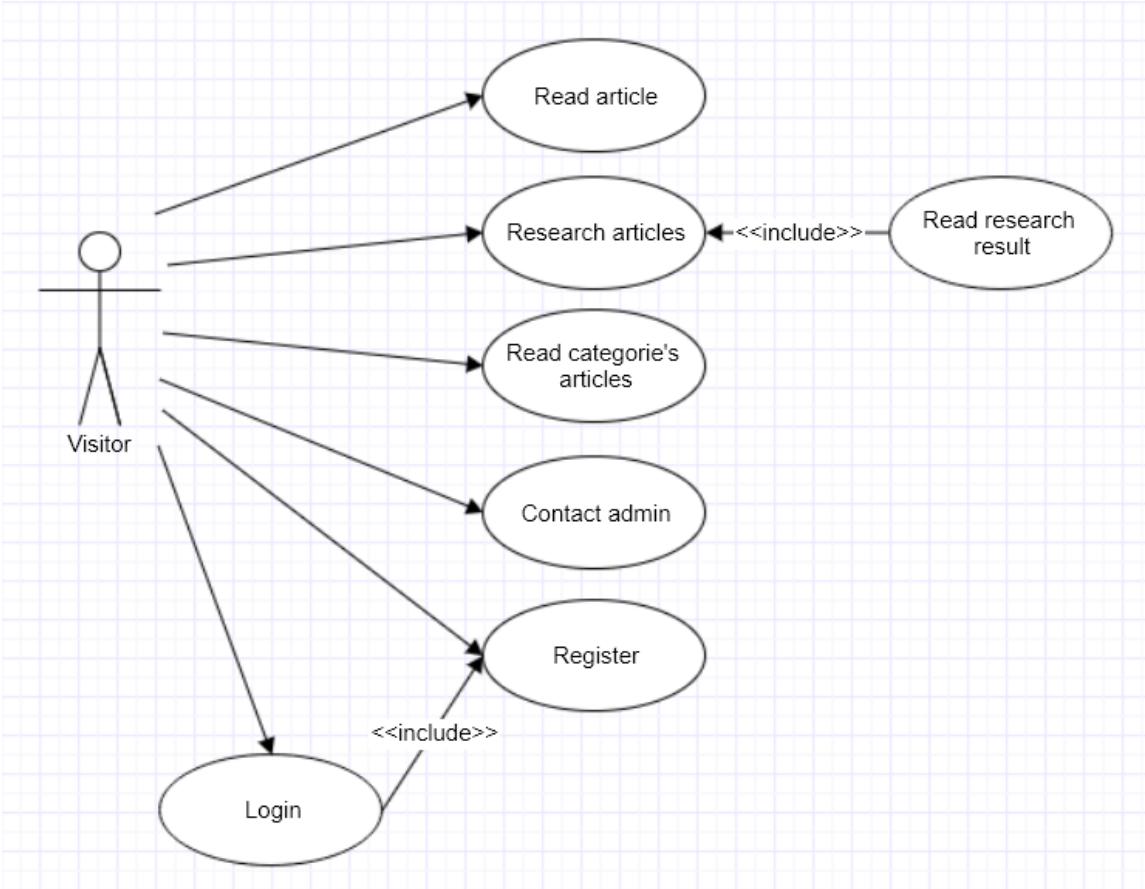
templates > home > index.html.twig

```
1   {% extends 'base.html.twig' %}
2
3   {% block title %}Travelers.com : Blogs de globetrotters{% endblock %}
4
5   {% block content %}
6
7
8
9   {% endblock %}
10
```

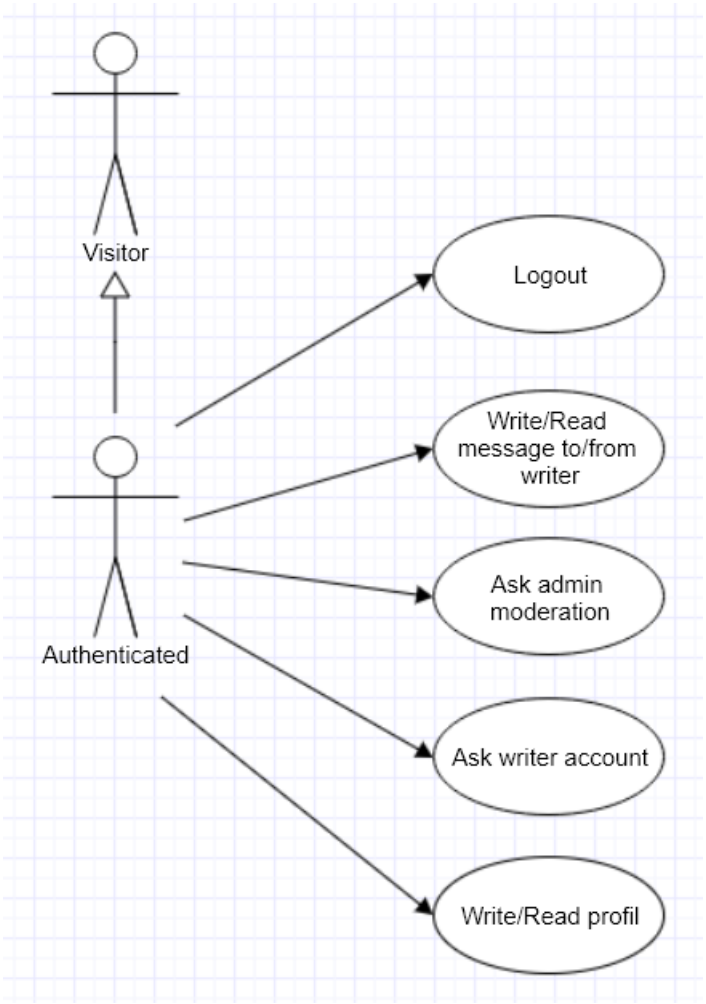
Le but de notre application : un espace de blogs sur le thème du voyage. Avant de créer toutes nos routes , il faut les identifier. Nous allons réaliser un **UML Uses Cases** qui nous permettra d'identifier toutes nos vues et de créer toutes nos routes.

Nous commençons par le Uses Cases pour le profil Visiteur (non authentifié). Il pourra :

- consulter tous les articles (déjà modérés)
- rechercher un article (barre de recherche)
- parcourir différentes catégories regroupant des articles sur le même thème (par continent)
- contacter l'administrateur du site
- créer un compte utilisateur et/ou s'authentifier



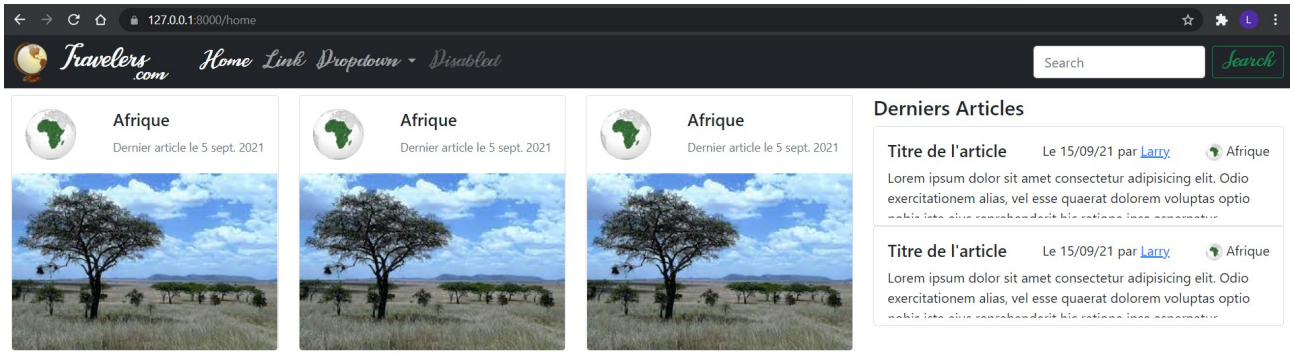
Pour le profil User (authentifié), il pourra faire tout ce que le simple visiteur à le droit de faire. Il pourra en plus :
Envoyer un message au rédacteur d'un article (et en lire la réponse)
Indiquer à l'admin un contenu inapproprié (pour modération)
Demander à l'admin pour devenir rédacteur d'articles
Consulter et modifier son profil
Faire un logout (redevenir un simple visiteur non authentifié)



Avant d'aller plus loin dans les Uses Cases en fonction du profil, nous allons déjà créer quelques routes correspondantes à ces Uses Cases.

Commençons par la page d'accueil qui affichera les derniers articles publiés, ainsi que les catégories d'articles (les continents). Nous avons déjà créé cette route, nous allons écrire notre html/css en « dur » dans un premier temps (pas de données pour l'instant).

Voir le code sur git : https://github.com/laurentbedu/td_symfony_5/tree/TD-5



Version mobile :