Fiche de Synthèse : Mes Voyages *Février 2021*

Développement :
Il s'agit d'un site web dynamique présentant mes voyages fictifs, avec leur description.

Cadre :
Il s'agit d'un TP proposé dans le cadre du cours "Réalisation et Maintenance de Composants Logiciels ", par Elisabeth Martins Da Silva.

Support :
Développé en PHP avec NetBeans, une bdd gérée par PhpMyAdmin. L'intérêt du TP était de nous faire découvrir le framework Symfony

Difficultés rencontrées :
L'inconvénient de ce type d'outils, très complet … c'est qu'il est difficile de modifier quoi que ce soit à l'existant. J'ai rencontré un souci de version de bundle (pour le bundle-uploader), qui ne fonctionnait pas avec ma version de doctrine-bundle, elle-même choisie par ma version de composer lors de l'installation du framework. Et c'est ainsi que j'ai finalement refaire une installation complète, et propre, afin d'avoir les bonnes versions compatibles entre elles.

Description détaillée du développement

J’ai suivi les étapes suivantes pour le développement:
- introduction
- création du projet sous Symfony
- les routes
- les vues
- configuration accès bdd
- l’administration du site
- ajout de bundle
- sécurité

1. Introduction

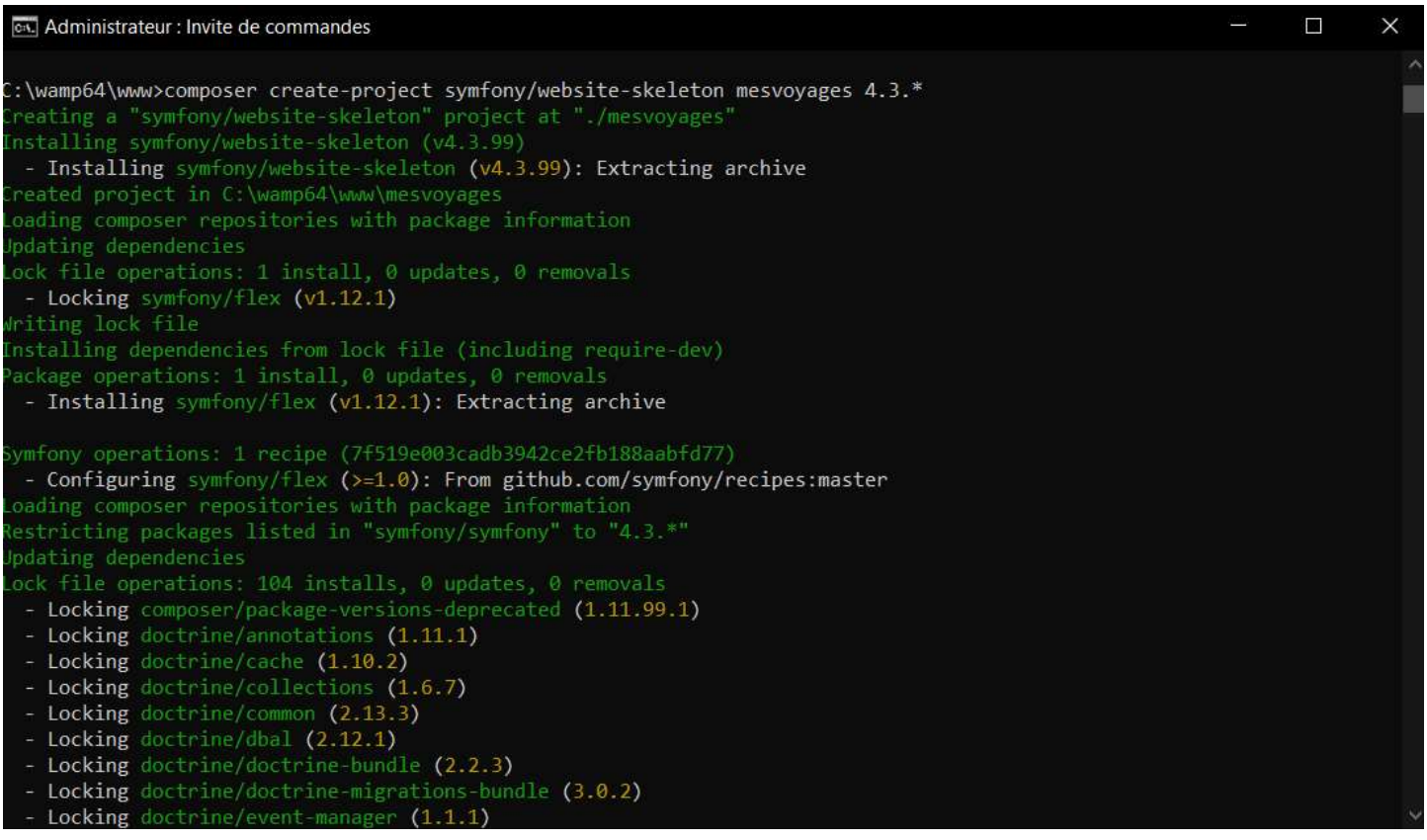
Ce TP est vraiment destiné à nous faire découvrir les possiblités, et les facilités que peuvent offrir un framework. Symfony est très populaire, c’est l’occasion de découvrir pourquoi.

Cependant, utiliser Symfony pour créer un tout petit site web dynamique tel que celui-ci, c’est comme utiliser un bazooka pour tuer un moustique : ça va marcher, pas de doutes, mais ce n’était pas indispensable dans un cas comme celui présenté. Il s’agit simplement d’une illustration.

Si vous êtes un adepte de la green IT, arrêtez votre lecture ici : poursuivre pourrait être douloureux.

2. Création du projet sous Symfony

Tout commence dans l’invite de commande Windows, en mode administrateur. Il faut se mettre dans le dossier approprié (dans mon cas le www de WampServer puisque c’est le serveur que j’utilise en local), et de demander gentiment à *composer* de créer un projet avec Symfony, un projet qui s’appelera donc *mesvoyages* :



Il fait tout le travail, va chercher les composants nécessaires, et à l'ouverture du projet avec l'IDE, vous avez déjà de très très très nombreux dossiers/fichiers, construits sur un modèle MVC (Modèle – Vue – Contrôleur).

3. Les routes

C'est donc un contrôleur qui va dire par où va devoir passer une requête. Pour indiquer la route, en fait.

Pour la page d'accueil du site par exemple, il y a donc une class AccueilController.php : on y instancie le repository (la classe qui sert d'interface avec la bdd) ensuite, pour cette page une fonction findAllLasted() est utilisée pour sélectionner les 2 dernières visites, puis le contrôleur vous dirige vers la page accueil.html.twig (la vue), en y envoyant au passage le résultat de la fonction utilisée, c'est à dire les 2 dernières visites :

```
class AccueilController extends AbstractController {
    /**
     *
     * @var MesvoyagesVisiteRepository
     */
    private $repository;

    /**
     *
     * @param MesvoyagesVisiteRepository $repository
     */
    function __construct(MesvoyagesVisiteRepository $repository) {
        $this->repository = $repository;
    }

    /**
     * @Route("/", name="accueil")
     * @return Response
     */
    public function index(): Response {
        $visites = $this->repository->findAllLasted(2);
        return $this->render("pages/accueil.html.twig", [
            "visites" => $visites
        ]);

    }
}
```

Le plus intéressant, c'est la route, qui est définie au-dessus, en commentaire : Symfony lit et interprête les commentaires (si on respecte la syntaxe évidemment) : donc dans ce cas on souhaite que cette page (la page accueil.html.twig) s'affiche aux chargement du site, c'est à dire avec l'URL de l'index.php.

4. Les vues

Restons sur la page d'accueil : il y a un fichier base.html.twig, sur lequel on affiche ce qu'il y a, classiquement, en haut d'une page web. Voici un (court) extrait :

```
<!DOCTYPE html>
<html lang='fr'>
   <head>
      <meta charset="UTF-8">
      <title>{% block title %}Welcome!{% endblock %}</title>

      <link rel="stylesheet" href="https://cdn.jsdelivr.net/npm/bootstrap@4.6.0/dist/css/bootstrap.min.css"
integrity="sha384-B0vP5xmATw1+K9KRQjQERJvTumQW0nPEzvF6L/Z6nronJ3oUOFUFpCjEUQouq2+l"
crossorigin="anonymous">
      {% block stylesheets %}{% endblock %}

   </head>
   <body>
      <!-- Haut de page -->
```

Rien de très nouveau, le lien vers le CDN de Bootstrap 4.6, et, plus bas dans le code, les liens CDN Javascript.

Cet en-tête est repris, ensuite, dans une page basefront.html.twig : pour cette page, il y a uniquement la barre de navigation :

```
{% extends "base.html.twig" %}

{% block title %}{% endblock %}
{% block stylesheets %}{% endblock %}
{% block top %}
   <div class='container'>
      <!-- titre -->
      <div class='text-center'>
         <h1>Mes voyages</h1>
         <p>Je vais vous présenter les villes que j'ai visitées lors de mes voyages.</p>
      </div>
      <!-- menu -->
      <nav class='navbar navbar-expand-lg navbar-light bg-light'>
         <div class='collapse navbar-collapse' id='navbarSupportedContent'>
            <ul class='navbar-nav mr-auto'>
               <li class='nav-item'>
                  <a class='nav-link' href='{{ path("accueil") }}'>Accueil</a>
               </li>
               <li class='nav-item'>
                  <a class='nav-link' href='{{ path("voyages") }}'>Voyages</a>
               </li>
               <li class="nav-item">
                  <a class="nav-link" href="{{ path('contact') }}">Contact</a>
               </li>
            </ul>
         </div>
      </nav>
   </div>
{% endblock %}
{% block body %}{% endblock %}
{% block javascripts %}{% endblock %}
```

Là, déjà, c'est un peu magique : dans le href du lien, là où normalement on devrait s'embêter à mettre un lien vers la page, il suffit de lui indiquer le nom de la page, celui qu'on a donné dans la route indiquée au contrôleur.
Dans le contrôleur de la page d'accueil, on lui avait dit :

```
 * @Route("/", name="accueil")
```

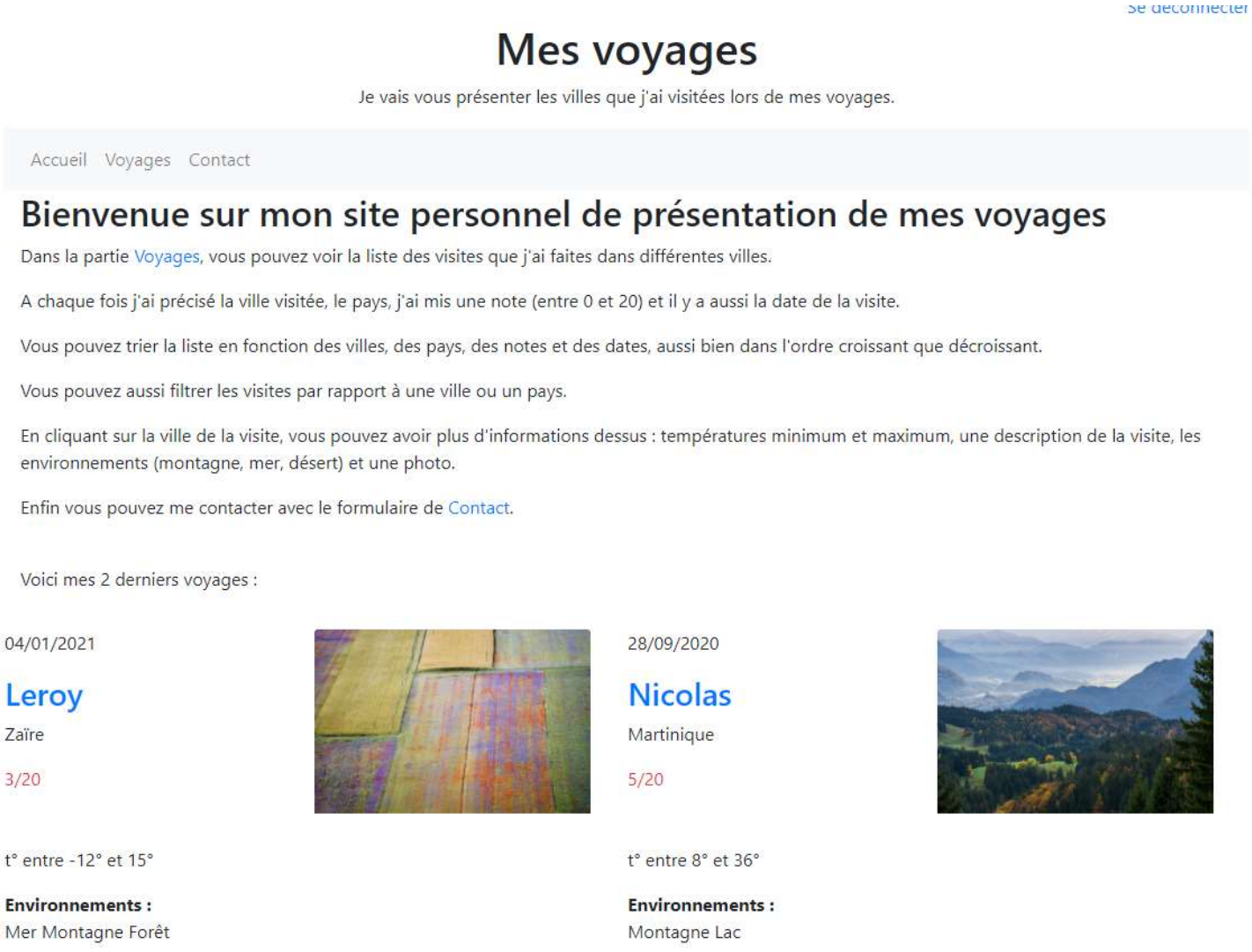
Donc puisqu'on a associé un nom au chemin, il suffit ensuite de l'utiliser dans le fichier twig.

A ce stade, nous avons donc l'en-tête qui est récupérée, et on a mis une barre de navigation, qui sera commune à toutes les pages du site. Reste donc à remplir, enfin, cette page d'accueil.html.twig :

```
{% extends "basefront.html.twig" %}

{% block body %}
  <div class="container">
    <h2>Bienvenue sur mon site personnel de présentation de mes voyages</h2>
    <p>blablabla ...</p>
  </div>

  <table>
    <tr>
      {% for visite in visites %}
        <td style="vertical-align: text-top; width: 50%">
          <div class="row mt-3 mr-3">
            <div class="col">
              <p>{{ visite.datecreationstring }}</p>
              <h3 class="text-primary mt-1">
                <a href="{{ path('voyages.showone', {id:visite.id}) }}">{{ visite.ville }}</a>
              </h3>
              <p>{{ visite.pays }}</p>
              {% if visite.note>=10 %}
                <p class="text-success mt-1">{{ visite.note }}/20</p>
              {% else %}
                <p class="text-danger mt-1">{{ visite.note }}/20</p>
              {% endif %}
              <p class="mt-5">t° entre {{ visite.tempmin }}° et {{ visite.tempmax }}°</p>
              <p class="mt-1"><strong>Environnements :</strong><br/>
                {% for environnement in visite.environnements %}
                  {{ environnement.nom }}
                {% endfor %}
              </p>
            </div>
            <div class="col">
              <!-- emplacement photo -->
              {% if visite.imagename %}
                <img src="{{ app.request.getBasePath()~vich_uploader_asset(visite, 'imageFile') }}"
                  class="card-img-top" style="with:100%;height:auto;">
              {% endif %}
            </div>
          </div>
        </td>
      {% endfor %}
    </tr>
  </table>
{% endblock %}
```

Puisque dans le contrôleur de l'accueil, on est allée chercher les visites de la bdd (je détaillerai plus tard), on peut tranquillement boucler sur une visite, et afficher les champs de la bdd là où on le souhaite dans la page.

Et là … c'est beau :



5. Configuration accès bdd

J’ai mis un peu la charrue avant les bœufs. Il est temps de s’attarder sur une question fondamentale : avec Symfony, comment créée-t-on une base de données ? Des tables ? Des champs ? Et les classes qui vont avec ?

Là encore, il suffit de demander gentiment, c’est à dire avec la syntaxe appropriée, directement en ligne de commande :

```
C:\wamp64\www\mesvoyages>php bin/console doctrine:database:create
Created database `mesvoyages` for connection named default
```

Maintenant que la bdd est créée, ajoutons lui une table, et des champs :

```
C:\wamp64\www\mesvoyages>php bin/console make:entity

Class name of the entity to create or update (e.g. FierceChef):
> mesvoyages_visite

created: src/Entity/MesvoyagesVisite.php
created: src/Repository/MesvoyagesVisiteRepository.php

Entity generated! Now let's add some fields!
You can always add more fields later manually or by re-running this command.

New property name (press <return> to stop adding fields):
> ville

Field type (enter ? to see all types) [string]:
>

Field length [255]:
> 50

Can this field be null in the database (nullable) (yes/no) [no]:
>

updated: src/Entity/MesvoyagesVisite.php

Add another property? Enter the property name (or press <return> to stop adding fields):
>
```

A ce stade, Symfony nous a créée une ‘Entity’, dans le code, prête à travailler avec les propriétés correspondants aux champs qu’on a renseigné à la création :

```
/**
 * @ORM\Entity(repositoryClass=MesvoyagesVisiteRepository::class)
 * @Vich\Uploadable
 */
class MesvoyagesVisite
{
    /**
     * @ORM\Id
     * @ORM\GeneratedValue
     * @ORM\Column(type="integer")
     */
    private $id;

    /**
     * @ORM\Column(type="string", length=50)
     */
    private $ville;

    /**
     * @ORM\Column(type="string", length=50)
     */
    private $pays;

...
```

Les commentaires @ORM vont permettre de créer les champs dans la base de données concernée.

En parallète, une classe dans le ‘repository’ a aussi été créée : celle-ci va servir à faire les requêtes dont on a besoin pour les accès à la bdd, dont voici un petit extrait (avec la méthode utilisée pour la page d’accueil).

```php
/**
 * @method MesvoyagesVisite|null find($id, $lockMode = null, $lockVersion = null)
 * @method MesvoyagesVisite|null findOneBy(array $criteria, array $orderBy = null)
 * @method MesvoyagesVisite[]    findAll()
 * @method MesvoyagesVisite[]    findBy(array $criteria, array $orderBy = null, $limit = null, $offset = null)
 */
class MesvoyagesVisiteRepository extends ServiceEntityRepository
{
    public function __construct(ManagerRegistry $registry)
    {
        parent::__construct($registry, MesvoyagesVisite::class);
    }

                                (...)
    /**
     * Retourne les n visites les plus récentes
     * @param type $nb
     * @return MesvoyagesVisite[]
     */
    public function findAllLasted($nb) : array {
        return $this->createQueryBuilder('v') // alias de la table
            ->orderBy('v.datecreation', 'DESC')
            ->setMaxResults($nb)
            ->getQuery()
            ->getResult();
    }
```

La syntaxe des requêtes est un peu particulière, il faut s'y habituer.

Ces classes sont donc bien créées automatiquement par Symfony.

Ensuite, une fois qu'on s'est assuré qu'il y avait bien tout ce qu'on voulait, on peut demander la migration :

```
C:\wamp64\www\mesvoyages>php bin/console make:migration


  Success!


Next: Review the new migration "migrations/Version20210209140928.php"
Then: Run the migration with php bin/console doctrine:migrations:migrate
See https://symfony.com/doc/current/bundles/DoctrineMigrationsBundle/index.html
```

A ce moment là, un fichier de migration est créé, qui détaille les opérations qui vont être effectuées sur la bdd :

```php
/**
 * Auto-generated Migration: Please modify to your needs!
 */
final class Version20210209140928 extends AbstractMigration
{
    public function getDescription() : string
    {
        return '';
    }

    public function up(Schema $schema) : void
    {
        // this up() migration is auto-generated, please modify it to your needs
        $this->addSql('CREATE TABLE mesvoyages_visite (id INT AUTO_INCREMENT NOT NULL, ville VARCHAR(50) NOT NULL, pays VARCHAR(50) NOT NULL, datecreation DATE DEFAULT NULL, note INT DEFAULT NULL, avis LONGTEXT DEFAULT NULL, PRIMARY KEY(id)) DEFAULT CHARACTER SET utf8mb4 COLLATE `utf8mb4_unicode_ci` ENGINE = InnoDB');
    }

    public function down(Schema $schema) : void
    {
        // this down() migration is auto-generated, please modify it to your needs
        $this->addSql('DROP TABLE mesvoyages_visite');
    }
}
```

On peut donc vérifier/modifier le fichier généré par Symfony avant de l'exécuter.
Si on est d'accord, que cela correspond bien à ce qu'on attend, on peut (enfin) créer la table :

```
C:\wamp64\www\mesvoyages>php bin/console doctrine:migrations:migrate

 WARNING! You are about to execute a database migration that could result in schema changes and data loss. Are you sure
you wish to continue? (yes/no) [yes]:
 > yes

[notice] Migrating up to DoctrineMigrations\Version20210209140928
[notice] finished in 69.6ms, used 16M memory, 1 migrations executed, 1 sql queries
```

A ce stade, la table est créée, on peut le vérifier dans phpMyAdmin :

6. L’administration du site

Bien sûr, il faut pouvoir modifier/supprimer des voyages, ou des environnements.

En réalité, au niveau de la conception, pas vraiment de nouveauté : après avoir créé un dossier ‘admin’ dans le Controller et les vues, on peut y développer un contrôleur pour l’administration des voyages. Par exemple, un extrait de la classe AdminVoyagesController, et simplement la méthode suppr() :

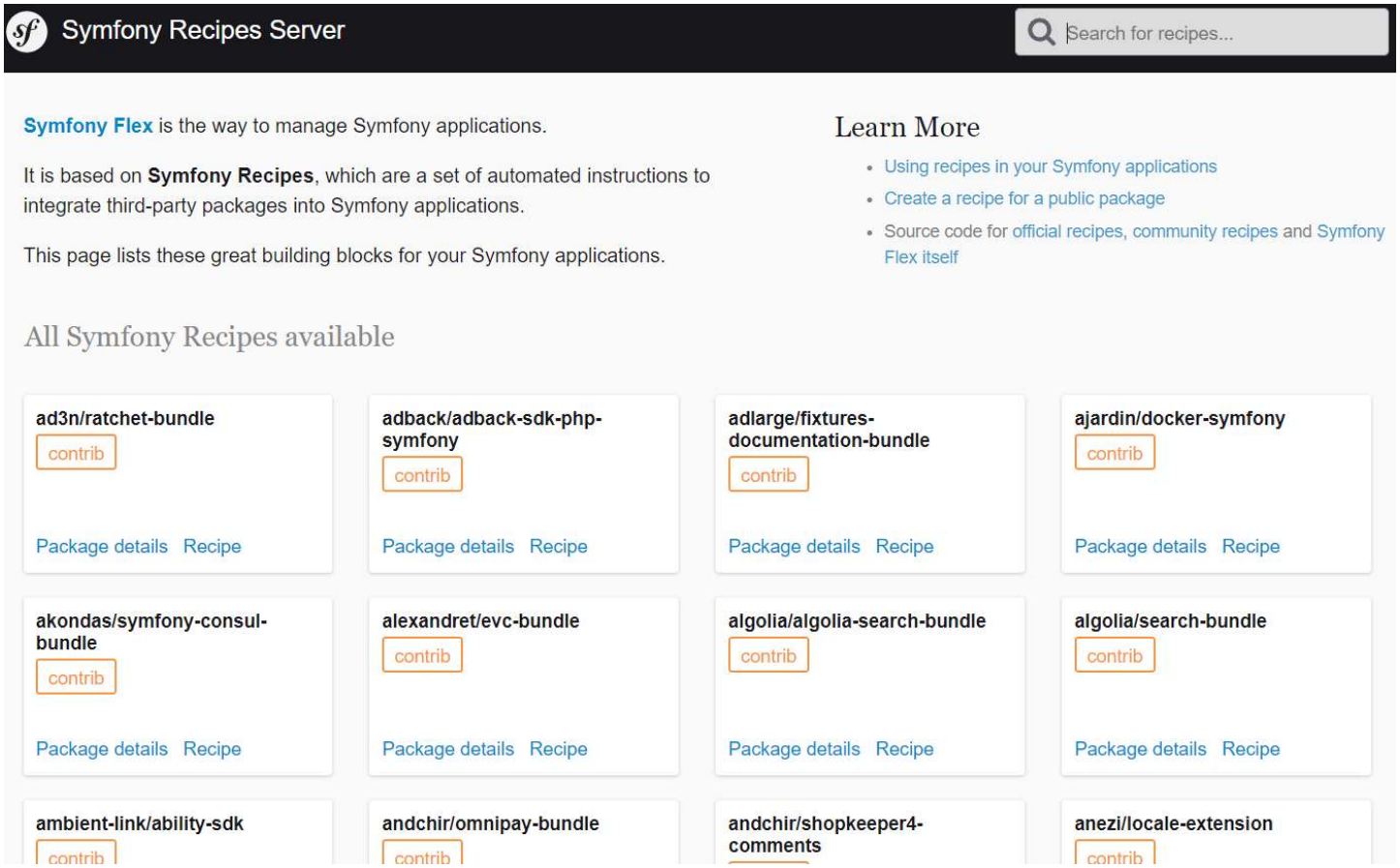
```php
class AdminVoyagesController extends AbstractController{

    /**
     *
     * @var MesvoyagesVisiteRepository
     */
    private $repository;

    /**
     *
     * @var EntityManagerInterface
     */
    private $om;

    /**
     *
     * @param MesvoyagesVisiteRepository $repository
     * @param EntityManagerInterface $om
     */
    function __construct(MesvoyagesVisiteRepository $repository, EntityManagerInterface $om) {
        $this->repository = $repository;
        $this->om = $om;
    }

    /**
     * @Route("/admin", name="admin.voyages")
     * @return Response
     */
    public function index(): Response {
        $visites = $this->repository->findAllOrderBy('datecreation', 'DESC');
        return $this->render("admin/admin.voyages.html.twig", [
            'visites' => $visites
        ]);
    }

    /**
     * @Route("/admin/suppr/{id}", name="admin.voyage.suppr")
     * @param Visite $visite
     * @return Response
     */
    public function suppr(MesvoyagesVisite $visite) : Response {
        $this->om->remove($visite);
        $this->om->flush();
        return $this->redirectToRoute('admin.voyages');
    }
```

Toujours le même principe : au-dessus de la méthode on indique la route ainsi que son nom pour pouvoir l'utiliser tranquillement dans la vue.

7. Ajout de bundle

Quand on a besoin de faire quelque chose qui n'est pas, par défaut, prévu par Symfony, on peut aller chercher si toutefois un bundle correspondant n'existe pas déjà, sur le site https://flex.symfony.com/

Il y en a pour tous les goûts, beaucoup proposés par des contributeurs, certains considérés comme 'officiels'.



Pour le chargement des images, nous avons installé uploader-bundle (pour télécharger des images), en suivant la procédure indiquée dans le dépôt GitHub de l'auteur :

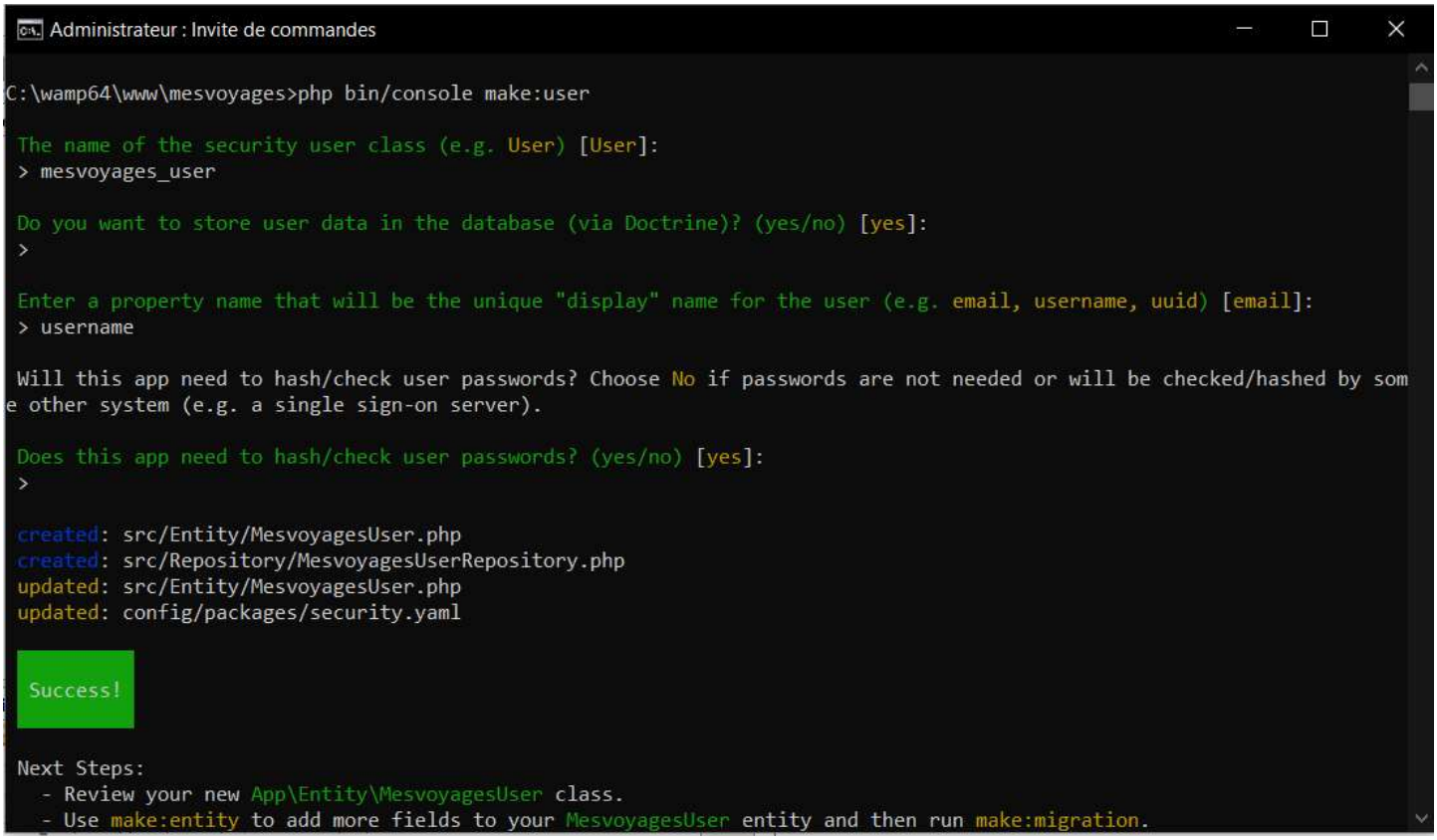
```
C:\wamp64\www\mesvoyages>composer require vich/uploader-bundle 1.10.0
./composer.json has been updated
Running composer update vich/uploader-bundle
Loading composer repositories with package information
Restricting packages listed in "symfony/symfony" to "4.3.*"
Updating dependencies
Lock file operations: 4 installs, 0 updates, 0 removals
  - Locking behat/transliterator (v1.3.0)
  - Locking jms/metadata (2.4.0)
  - Locking symfony/templating (v4.3.11)
  - Locking vich/uploader-bundle (1.10.0)
Writing lock file
Installing dependencies from lock file (including require-dev)
Package operations: 4 installs, 0 updates, 0 removals
  - Installing symfony/templating (v4.3.11): Extracting archive
  - Installing jms/metadata (2.4.0): Extracting archive
  - Installing behat/transliterator (v1.3.0): Extracting archive
```

Pour l'utilisation, là encore tout est indiqué dans le dépôt GitHub, aucune difficulté.

8. Sécurité

Nous avons donc un joli site, que nous pouvons administrer … comme n'importe qui, en tout cas pour l'instant. Il est donc temps de sécuriser l'accès à la partie 'admin' de notre site.

Symfony a la solution bien sûr :



Les classes MesvoyagesUser et MesvoyagesUserRepository sont ainsi créées, ainsi que la table dédiée dans la bdd.

On peut ajouter, pour que ce soit plus pratique, un joli formulaire de connexion, avec la ligne de commande :

```
C:\wamp64\www\mesvoyages\php bin/console make:auth
```

De nombreux fichiers sont générés automatiquement : un fichier SecurityController.php avec les méthodes login() et logout(), un fichier login.html.twig pour l'affichage d'un formulaire d'authentification et un fichier LoginFormAuthenticator.php qui permet de traiter le formulaire. C'est seulement ce dernier que l'on va légèrement modifier, afin de lui indiquer quoi faire en cas d'échec de l'authentification (la page de redirection en fait).

Il faut donc ajouter ça à la méthode onAuthenticationSuccess() :

```
public function onAuthenticationSuccess(Request $request, TokenInterface $token, $providerKey)
    {
        if ($targetPath = $this->getTargetPath($request->getSession(), $providerKey)) {
            return new RedirectResponse($targetPath);
        }

        // For example : return new RedirectResponse($this->urlGenerator->generate('some_route'));
        // throw new \Exception('TODO: provide a valid redirect inside '.__FILE__);
        return new RedirectResponse($this->urlGenerator->generate('accueil'));
    }
```

Ensuite, après une légère modification du formulaire de la vue, on obtiens ceci :