



Graduate développeur web full stack

ECF Promo Novembre-Décembre 2024

Adrien CAVET

Logiciels et Frameworks utilisés :

- Symfony
- Twig
- Easyadmin
- Bootstrap
- Sass
- VScode
- Chrome
- Figma
- phpMyAdmin
- Github
- Platform.sh
- Doctrine
- Composer
- Wampserver
- Adobe firefly

Langages utilisés :

- HTML
- Javascript
- PHP
- SQL

Github:

https://github.com/Adrien-1-9-8-6/zoo_弧adia



Identifiants:

Compte administrateur

joselebreton@arcadia.com

Lebreton

José

Labretagne.1\$

Compte employé

josetteemployee@arcadia.com

Employée

Josette

Employée.1\$

Compte vétérinaire

rogerveterinaire@arcadia.com

Vétérinaire

Roger

Docteur.1\$

Maquettage et design:

Lien Figma:

<https://www.figma.com/design/j0fleCx6Y37zM8SOgOm8he/Zoo-Arcadia?node-id=25-682&t=gdOH67dTJCC6eUfs-1>

Choix des couleurs:

L'énoncé nous précise qu'il s'agit d'un zoo avec des tendances écologiques. J'ai donc pensé à utiliser du vert (pour l'écologie) et du marron (qui rappelle la couleur des arbres). J'ai aussi ajouté un jaune qui reflète les couleurs que l'on peut trouver dans la savane.

Choix de la police:

J'ai choisi comme police principale "Baloo" car elle semble être appréciée des enfants. Je pense qu'un zoo cible principalement ce public et les parents doivent le ressentir quand ils visitent notre site.

Comme police secondaire, j'ai choisi "Cardo" car elle crée un contraste avec la première.

Design responsive:

Vu qu'il est maintenant nécessaire d'adapter les sites internet à différents formats d'écran, j'ai créé trois maquettes pour un format mobile et desktop.

Charte graphique:

Vous trouverez via mon lien Figma une charte graphique comprenant les polices, les couleurs ainsi que les composants utilisés

Précisions:

L'organisation des éléments n'est pas optimale, il est prévu de faire quelque chose de plus propre par la suite.

Accessibilité:

J'ai utilisé le plugin "Contrast" afin de vérifier si la taille et les couleurs de polices étaient accessibles. J'ai essayé de trouver un compromis entre accessibilité et design.

The screenshot shows the Contrast plugin interface with two color swatches side-by-side. Both swatches have a dark background and the word 'Handgloves' in a light yellow font.

Color Swatch 1 (Left)	Color Swatch 2 (Right)
#BE9915	#A58952
Contrast Ratio: 5.84 : 1	Contrast Ratio: 4.74 : 1
Normal Text: AA	Normal Text: AA
Large Text: AA	Large Text: AAA
Graphics: AA	Graphics: AA

The screenshot shows the Contrast plugin interface with a single color swatch. It has a dark brown background and the word 'Handgloves' in a light yellow font.

Color Swatch (Left)
#BE9915
Contrast Ratio: 5.45 : 1
Normal Text: AA
Large Text: AA
Graphics: AA

Select Scan

The screenshot shows the Contrast plugin interface with a single color swatch. It has a dark brown background and the word 'Handgloves' in a light yellow font.

Color Swatch (Left)
#37240E
Contrast Ratio: 5.45 : 1
Normal Text: FAIL
Large Text: AA
Graphics: AA

Symfony:

J'ai pris la décision d'installer d'entrée symfony. Cela m'a permis d'avoir des dossiers et des fichiers pré-organisés afin de démarrer dans de bonnes conditions. J'ai pu commencer la partie Front-end via Twig et utiliser les fonctionnalités de Symfony pour construire mes contrôleur.

Création de mon premier contrôleur:

J'ai utilisé la commande make:controller ce qui m'a permis de créer mon premier contrôleur et fichier Twig afin de construire ma page d'accueil.

```
antho@LAPTOP-8UUD7LTL MINGW64 /c/wamp64_3_3_0
$ symfony console make:controller

Choose a name for your controller class (e.g
> HomeController

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

Routage:

J'ai utilisé la même manipulation pour la création des autres pages de mon site, en prenant soin de configurer la route concernée

```
#[Route('/habitat', name: 'app_habitat')]
```

Front-end:

Html:

Il n'y a pas grand chose à dire si ce n'est que j'ai fait usage de nombreuses balises et classes. exemple d'un de mes boutons présents sur le haut de ma page:

```
<button class="btn btn-warning" onclick="window.location.href = '/services#festindestination';"></br>
  Le Festin Du Lion
  </br>
  </br>
  
</button>
```

Base.html.twig

Ce fichier me permet de pouvoir afficher des éléments sur toutes les pages de mon site. J'y ai mis mon header et mon footer

Toutes les autres pages auront un extends pour récupérer la configuration de ma base:

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

Ainsi que mes assets (exemple):

```
ef="{{ asset('scss/main.css') }}" rel="stylesheet">
```

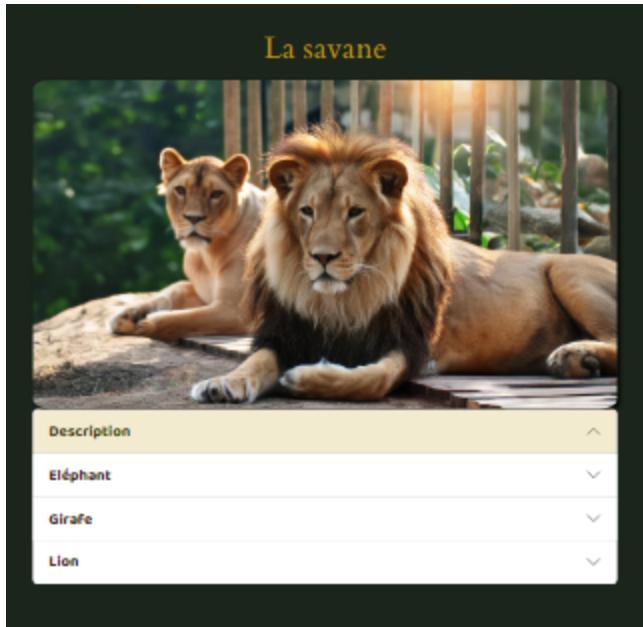
Combinaison Bootstrap plus Sass:

J'ai utilisé cette combinaison pour avoir des éléments préconfigurés et me donner la possibilité de les personnaliser via un fichier SCSS.

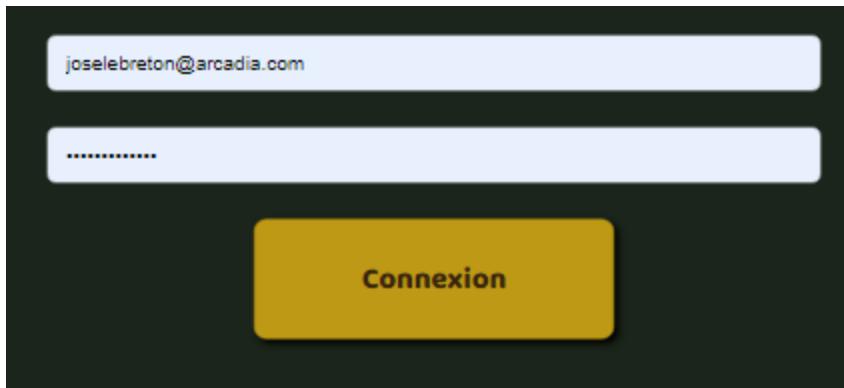
Navbar bootstrap:



Accordéon bootstrap:



Formulaire de connexion plus bouton bootstrap:



J'ai pu via un fichier SCSS créer des variables intégrant les couleurs et la police d'écriture:

```
$primary: #BE9915;
$secondary: #1B251B;
$dark: #A58952;
$black: #37240E;
$font-family-base: 'Baloo 2', sans-serif;
$font-family-secondary: 'Cardo', sans-serif;

@import "../node_modules/bootstrap/scss/bootstrap";
```

Usage des classes bootstrap principalement pour du responsive:

```
ooter class="bg-black text-primary text-center footer">
  <div class="row">
    <div class="col-12 col-lg-4">
```

CSS:

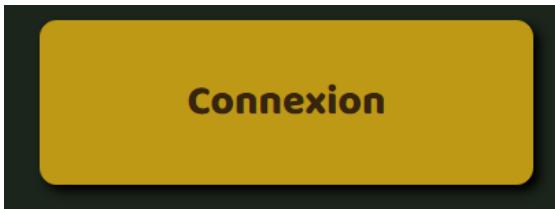
J'ai créé de nombreuses règles CSS, je vais donc parler des plus importantes ou les plus atypiques.

Utilisation d'@media pour un design responsive:

```
@media (max-width: 992px) {
  main {
    margin-bottom: 550px;
  }
}
```

Ombre en arrière plan afin de créer un effet de relief sur mes boutons et photos:

```
.configImage {
  box-shadow: 3px 3px 5px rgba(0, 0, 0, );
  border-radius: 3%;
}
```



Application d'un filtre sur une photo pour en faire ressortir la police superposée:

```
.image-accueil {
    position: relative;

    &::before {
        content: "";
        position: absolute;
        top: 0;
        left: 0;
        width: 100%;
        height: 100%;
        background-image: url(..../images/Accueil.png);
        background-size: cover;
        background-position: 0% 70%;
        filter: brightness(0.8);
    }
}
```

Utilisation de règle en fonction d'une balise:

```
body {
```

d'une classe:

```
.btn-primary {
```

Précisions:

Je me suis rendu compte que plus les règles CSS s'accumulent, plus il était peu judicieux de toutes les réunir dans un seul et même fichier. Actuellement elles sont toutes dans le fichier "main.scss" je compte par la suite les diviser dans plusieurs fichier nominatifs.

Chrome:

Même si l'usage d'un navigateur internet semble être une évidence, son usage m'a été indispensable pour styliser mon site grâce à la fonction "inspecter" et la possibilité de tester différents formats d'écran.

Adobe firefly:

Je tiens aussi à rendre hommage à Adobe firefly qui m'a permis de générer toutes les photos présentes sur mon site.

Back-end:

Wampserver et phpMyAdmin:

Je n'ai pas de raisons particulières de préférer Wamp à Xampp si ce n'est qu'il était utilisé dans les tutoriels que j'ai visionné. J'ai par contre une grande préférence pour phpMyAdmin en comparaison à MySQL Workbench, son interface m'était beaucoup plus intuitive, il est présent dans de nombreux tutoriels aussi et il est intégré à Wamp.

Doctrine:

J'ai utilisé doctrine pour configurer ma base de données. J'ai créé un fichier .env.local pour éviter que ce fichier contenant des informations sensibles se retrouve sur mon git public.

Configuration .env.local:

```
###> symfony/framework-bundle ###
APP_ENV=dev
APP_SECRET=9586eb55a235b0b7963c0569e3d76c14
###< symfony/framework-bundle ###

###> doctrine/doctrine-bundle ###
DATABASE_URL="mysql://Adrien:Studi.1$@127.0.0.1:3306/ZooArcadia?serverVersion=8.0.32&charset=utf8mb4"
###< doctrine/doctrine-bundle ###
```

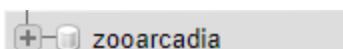
Je vous colle les informations si vous avez besoin d'en faire usage:

```
APP_SECRET=9586eb55a235b0b7963c0569e3d76c14

DATABASE_URL="mysql://Adrien:Studi.1$@127.0.0.1:3306/ZooArcadia?serverVersion=8.0.32&charset=utf8mb4"
```

Création de ma base de données:

```
$ symfony console doctrine:database:create
```



Création de ma première entité et de mon premier repository, cela m'a permis de créer ma table dans un format php:

```
$ symfony console make:entity  
Class name of the entity to create or update (e.g. OrangeChef):  
> █
```

J'ai ensuite suivi les instructions pour configurer les types de données:

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

Construction du script qui va me permettre d'injecter mon entité dans ma base de données en faisant une transition php => sql:

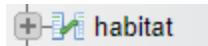
```
$ symfony console make:migration  
created: migrations/Version20231201100323.php
```

Success!

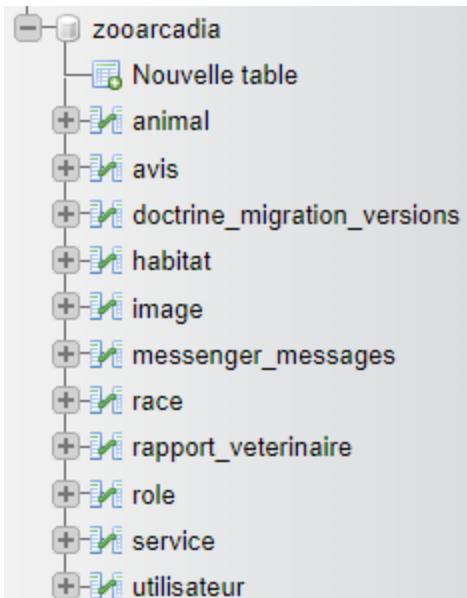
Activation de mon script:

```
$ symfony console doctrine:migration:migrate
```

Ma table est maintenant créée:



Il me reste plus qu'à réitérer pour la création de mes autres tables.



Création de la table utilisateur:

Le fonctionnement est pratiquement le même que pour la création d'une entité classique mais avec des instructions adaptées à une table utilisateur. Cette fois on utilise un `make:user`. Via cette manipulation, des modifications vont aussi être apportées au fichier "security.yaml". Précision importante aussi, un rôle sous format Json va être ajouté à la table utilisateur, il me servira ensuite pour les droits utilisateur.

```
$ php bin/console make:user
```

```
providers:
    # used to reload user from session & other features (e.g. switch_user)
    app_user_provider:
        entity:
            class: App\Entity\User
            property: email
```

roles	json	<input style="width: 150px;" type="text" value='["ROLE_ADMIN"]'/>
-------	------	---

Création des relations entre les tables:

J'utilise à nouveau le make:entity mais je rentre le nom de l'entité déjà existante sur laquelle je veux faire la relation.

```
$ symfony console make:entity
Class name of the entity to create or update (e.g. OrangeChef):
> |
```

Cette fois je choisis comme type "relation"

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

What class should this entity be related to?:
```

puis je choisis la nature de la relation:

Type	Description
ManyToOne	Each Picture relates to (has) one Restaurant. Each Restaurant can relate to (can have) many Picture objects.
OneToMany	Each Picture can relate to (can have) many Restaurant objects. Each Restaurant relates to (has) one Picture.
ManyToMany	Each Picture can relate to (can have) many Restaurant objects. Each Restaurant can also relate to (can also have) many Picture objects.
OneToOne	Each Picture relates to (has) exactly one Restaurant. Each Restaurant also relates to (has) exactly one Picture.

Me voici avec une clé étrangère "role_id" sur ma table utilisateur:

2 role_id  int

Création de ma page d'administration

J'ai installé pour cela "Easyadmin"

```
$ composer require easycorp/easyadmin-bundle
```

Je crée mon Dashboard en suivant les instructions. Un dashboard controller va être créé:

```
$ php bin/console make:admin:dashboard
```

Je peux maintenant accéder à mon dashboard en ajoutant /admin dans mon URL ce qui pose un gros problème de sécurité car tout le monde peut le faire. Nous réglerons cela plus tard.

Création des CRUD

La création de CRUD autour de mes entités va m'être nécessaire si je souhaite pouvoir les configurer depuis ma page d'administration. Si on prend l'exemple d'un utilisateur, le CRUD va me permettre de:

- Créer un utilisateur
- Lire les information concernant un utilisateur
- Mettre à jour les information d'un utilisateur
- Supprimer un utilisateur

J'utilise encore une fois une commande symfony et je choisi l'entité concernée:

```
$ symfony console make:admin:crud

Which Doctrine entity are you going to manage with this CRUD controller?:
[0] App\Entity\Animal
[1] App\Entity\Avis
[2] App\Entity\Habitat
[3] App\Entity\Image
[4] App\Entity\Race
[5] App\Entity\RapportVeterinaire
[6] App\Entity\Role
[7] App\Entity\Service
[8] App\Entity\Utilisateur
> |
```

Configuration de mon Dashboard:

La route:

```
#Route('/admin', name: 'admin')
```

Récupération du CRUD plus un générateur d'URL pour chaque onglets de mon Dashboard:

```
$routeBuilder = $this->container->get(AdminUrlGenerator::class);
$url = $routeBuilder->setController(UtilisateurCrudController::class)->generateUrl();
return $this->redirect($url);      You, il y a 3 jours • Pratiquement tous les CRUD on
```

Récupération de mon entité et personnalisation de mon Dashboard:

```
yield MenuItem::linkToCrud('Utilisateur', 'fas fa-list', Utilisateur::class);
```

Après avoir fait la même chose avec l'entité "rôle" je peux maintenant créer des utilisateurs et leur donner un rôle:

Create Utilisateur

Email*

Nom*

Prenom*

Password*

Role*

Vétérinaire

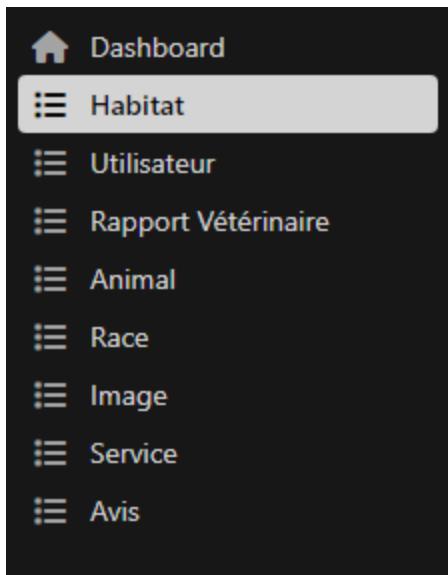
Je peux lire des informations sur mes utilisateurs:

<input type="checkbox"/>	Email	Nom	Prenom
<input type="checkbox"/>	joselebreton@arcadia.com	Lebreton	José
<input type="checkbox"/>	josetteemployee@arcadia.com	Employée	Josette
<input type="checkbox"/>	rogerveterinaire@arcadia.com	Vétérinaire	Roger

et je peux les mettre à jour ou les supprimer:



Il me reste plus qu'à réitérer l'opération pour les autres tables que je veux intégrer à mon Dashboard.



Sécurité:

Hachage du mot de passe:

Pour l'instant, le mdp des utilisateurs apparaît en clair dans ma base de données. Cela représente une faille de sécurité car si une personne arrive à récupérer les données de ma table utilisateur, il connaîtra tous les mdp configurés. J'ai mis en place un script qui permettra de hacher le mdp puis de supprimer le mdp en clair de la variable \$entityInstance. On attaque une partie assez complexe, je vais dans un premier temps mettre l'ensemble du code, puis je vais l'expliquer en détail:

```
//Astuce pour hacher le mdp

private $passwordEncoder;

public function __construct(UserPasswordHasherInterface
$passwordEncoder)

{
    $this->passwordEncoder = $passwordEncoder;
}

public static function getEntityFqcn(): string
{
    return Utilisateur::class;
}

public function persistEntity(EntityManagerInterface $entityManager,
$entityInstance): void
{
    if ($entityInstance instanceof Utilisateur) {
        $plainPassword = $entityInstance->getPassword(); // Obtenir le
mot de passe brut
```



```

$encodedPassword =
$this->passwordEncoder->hashPassword($entityInstance, $plainPassword); // Hacher le mot de passe

    $entityInstance->setPassword($encodedPassword); // Définir le mot de passe haché

}

$entityManager->persist($entityInstance);

$entityManager->flush();

$entityInstance->eraseCredentials(); //Suppression du mdp en clair une fois haché

}

//Fin de l'astuce pour hacher le mdp

```

Explication du code:

UserPasswordHasherInterface

Interface pour hacher les mdp

\$passwordEncoder

Variable pour stocker l'instance de UserPasswordHasherInterface

public function __construct(UserPasswordHasherInterface \$passwordEncoder)

Utilisation d'un constructeur de classe pour prendre une instance de UserPasswordHasherInterface en paramètre et l'assigner à la propriété \$passwordEncoder

public static function getEntityFqcn()

Je retourne le nom complet de la classe Utilisateur

```
public function persistEntity(EntityManagerInterface $entityManager,  
$entityInstance): void
```

Je persiste l'entité `Utilisateur` avec deux paramètres, une instance de `EntityManagerInterface` et une instance de l'entité à persister.

```
if ($entityInstance instanceof Utilisateur) {  
  
    $plainPassword = $entityInstance->getPassword(); // Obtenir le  
mot de passe brut  
  
    $encodedPassword =  
$this->passwordEncoder->hashPassword($entityInstance, $plainPassword); //  
Hacher le mot de passe  
  
    $entityInstance->setPassword($encodedPassword); // Définir le  
mot de passe haché  
  
}
```

On lance une condition "if" pour vérifier qu'on utilise bien un instance de `Utilisateur` puis on lance le processus de hachage.

```
$entityManager->persist($entityInstance);  
  
$entityManager->flush();
```

Je persiste et je flush

```
$entityInstance->eraseCredentials(); //Suppression du mdp en clair une  
fois haché
```

Je suis peut-être dans l'excès, mais je préfère vider la variable vu qu'elle contient le mdp en clair.

Le script fonctionne mais...

Je me suis rendu compte par hasard que quand on edit un utilisateur et qu'on lui change son mdp, le nouveau mdp apparaît en clair ce qui présente là encore une faille de sécurité.
J'ai dû ajouter un nouveau script pour régler ce problème:

```
//Astuce pour hacher le mdp quand on change le mdp

    public function updateEntity(EntityManagerInterface $entityManager,
$entityInstance): void

{

    if ($entityInstance instanceof Utilisateur) {

        $plainPassword = $entityInstance->getPassword(); // Obtenir le mot
de passe brut

        $encodedPassword =
$this->passwordEncoder->hashPassword($entityInstance, $plainPassword); // Hacher le mot de passe

        $entityInstance->setPassword($encodedPassword); // Définir le mot
de passe haché

    }

    $entityManager->persist($entityInstance);

    $entityManager->flush();

    $entityInstance->eraseCredentials(); //Suppression du mdp en clair une
fois haché

}

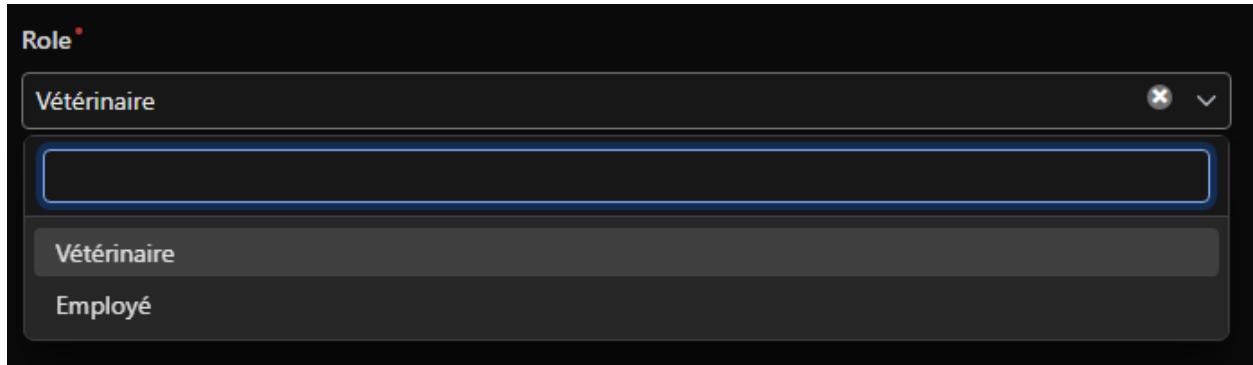
//Fin de l'astuce pour hacher le mdp quand on change le mdp
```

Supprimer la possibilité de créer un compte administrateur depuis le Dashboard:

J'ai créé le compte administrateur de José depuis le Dashboard pour bénéficier du hachage de mdp, j'ai ensuite supprimé cette possibilité avec ce script:

```
public function configureFields(string $pageName): iterable
{
    return [
        TextField::new('email'),
        TextField::new('nom'),
        TextField::new('prenom'),
        TextField::new('password'),
        AssociationField::new('role') //Ligne pour le champ de
        sélection de rôle
        ->setFormTypeOptions([ //suppression de la création du compte
            admin depuis la dashboard
            'query_builder' => function (RoleRepository $er) {
                return $er->createQueryBuilder('r')
                    ->where('r.label != :admin')
                    ->setParameter('admin', 'Administrateur');
            },
        ]),
    ];
}
```

Le rôle administrateur n'apparaît plus dans la liste:



Formulaire de connexion:

Dans un premier temps j'ai créé un contrôleur qui m'a aussi créé un nouveau fichier .twig qui me servira de page de connexion.

Ensuite, dans la documentation de Symfony on nous dit d'ajouter dans le fichier security.yaml ceci:

```
form_login:
    # "app_login" is the name of the route created previously
    login_path: app_login
    check_path: app_login
```

Je récupère le code de la documentation symfony pour configurer mon contrôleur:

```
#[Route('/login', name: 'app_login')]

public function index(AuthenticationUtils $authenticationUtils):
Response

{
    // get the login error if there is one
    $error = $authenticationUtils->getLastAuthenticationError();

    // last username entered by the user
    $lastUsername = $authenticationUtils->getLastUsername();

    return $this->render('login/index.html.twig', [
        'last_username' => $lastUsername,
        'error'           => $error,
    ]);
}

#[Route('/logout', name: 'app_logout')]

public function logout()
{
    $error= "";
    $lastUsername= "";

    return $this->render('login/index.html.twig', [
        'last_username' => $lastUsername,
        'error'           => $error,
    ]);
}
```

J'ajoute ensuite un formulaire bootstrap que j'ai personnalisé:

```
<form action="{{ path('app_login') }}" method="post"
class="form-signin">

    <label for="username" class="sr-only"></label>

    <input type="text" id="username" name="_username" value="{{ last_username }}"
required autofocus class="form-control"
placeholder="Adresse mail" >

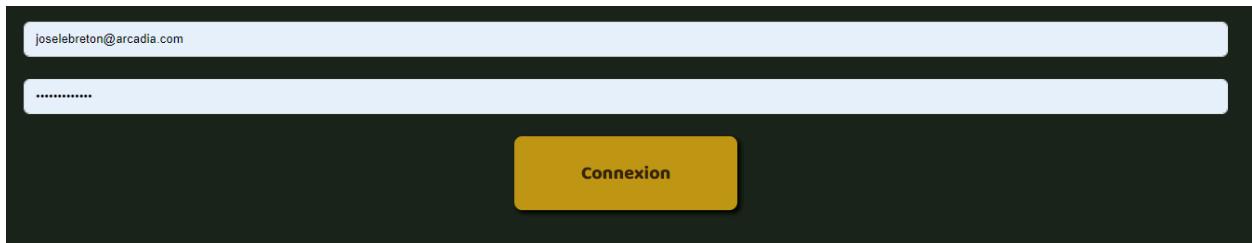
    <label for="password" class="sr-only"></label>

    <input type="password" id="password" name="_password" required
class="form-control" placeholder="Mot de passe">

    <input type="hidden" name="_target_path" value="/admin">

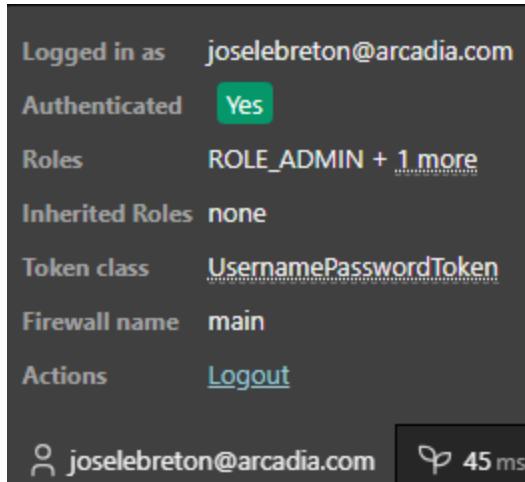
    <button class="btn btn-lg btn-primary btn-block mt-4"
type="submit">Connexion</button>

</form>
```



Attribution des rôles:

Quand je me connecte, je remarque que Symfony m'attribue un rôle:



Cela ressemble fortement au rôle que je dois rentrer dans ma table de données utilisateur en format Json. J'ai donc eu l'idée de me baser sur ces rôles-là grâce à ce script:

```
// Astuce pour attribuer le rôle en fonction du label

$roleLabel = $entityInstance->getRole()->getLabel();

if ($roleLabel == 'administrateur') {
    $entityInstance->setRoles(['ROLE_ADMIN']);
} elseif ($roleLabel == 'Vétérinaire') {
    $entityInstance->setRoles(['ROLE_VETERINAIRE']);
} else {
    $entityInstance->setRoles(['ROLE_EMPLOYE']);
}

//
```

Restriction en fonction des rôle:

Un vétérinaire n'a pas à accéder à la création d'un utilisateur par exemple. Il fallait donc que j'impose des restrictions d'accès. J'ai pris la décision de créer trois Dashboard:

-Administrateur

-Employé

-Vétérinaire

Je personnaliserai celles-ci en fonction des besoins de chacun.

J'ai ensuite configuré les conditions d'accès à mes Dashboard:

```
access_control:
    - { path: ^/admin, roles: [ROLE_ADMIN] }
    - { path: ^/employe, roles: [ROLE_ADMIN, ROLE_EMPLOYE] }
    - { path: ^/veterinaire, roles: [ROLE_ADMIN, ROLE_VETERINAIRE] }
```

Précision: j'ai mis le rôle administrateur sur toutes les pages car il est plus facile d'effectuer mes tests sur toutes les pages avec un seul compte (celui de José).

Déploiement du site:

Platform.sh:

J'ai pris la décision de déployer mon site sur platform.sh. Je vous épargne toutes les manipulations que j'ai faites, ça me semble peu pertinent. Le site s'est bien déployé mais il rencontre plusieurs problèmes:

- Les objets bootstrap fonctionnent mal
- La fameuse erreur 500 est présente

A l'heure où j'écris ces lignes, je suis sur le point de rendre mon projet car je m'approche de la date limite. Je me suis donc empressé de déployer mon site afin que vous ayez un visuel. Je corrigerai les problèmes de déploiement par la suite, en attendant voici le lien:

<https://master-7rqtwti-shiyb27n7edjk.fr-4.platformsh.site/services>

En local:

Vous trouverez dans le fichier Readme.md une explication claire et précise pour déployer mon application en local. J'ai effectué plusieurs tests afin de m'assurer que tout était fonctionnel.

Important:

Il manque encore plusieurs éléments afin de compléter mon projet mais la date limite m'impose de vous l'envoyer en l'état. Pour la suite, rien ne me semble insurmontable, il me faudra simplement plus de temps.