

Pascal CHANTREL

DWWM 2021 - 2022



SOLENEOS

RAFRAICHISSEZ NOS VILLES

Sommaire

Introduction

Soleneos	page 2
Le projet	page 2
Langages, Frameworks et librairies utilisés	page 2
Logiciels utilisés pour ce projet	page 4
Compétences acquises pendant ce projet	page 5

Création du site de l'entreprise

	page 5
Etude de la charte graphique	page 5
Installation de Symfony	page 10
Création du Layout (Header + Footer)	page 13
Création du contenu	page 15
Formulaire de contact et newsletter	page 17
Hébergement	

Création de l'accès client

	page 18
Création des entités avec les relations	page 20
Création de la base de données	page 22
Installation du bundle Easy Admin	page 26
Création du formulaire d'inscription	page 28
Configuration de la page "connexion"	

Mise en oeuvre du serveur

	page 30
Configuration d'un "cloud computer"	page 31
Mise en place du server Apache	

Conclusion

	page 34
Conclusion et remerciements	

Annexes

	page 35
Annexes	

Introduction

SOLENEOS est un bureau d'étude nantais spécialisé dans le traitement des îlots de chaleur urbains qui accompagne les acteurs de l'urbanisme dans la conception de quartiers durables.

Basée sur la réalisation de simulation numérique, le cœur de l'expertise de SOLENEOS permet l'aide à la conception en prenant finement en compte le climat urbain.

Acteur engagé de l'innovation, SOLENEOS contribue à faire évoluer le monde de l'urbanisme vers la création de villes plus résilientes.

Le projet

Objectifs:

Ma première mission était de mettre à jour le site dans l'entreprise, en effet le site avait été produit de manière assez "basique", Le directeur de Soleneos voulant gagner en visibilité, il devait avoir un site présentant les différentes missions de l'entreprise, sa présentation ainsi que le moyen de la contacter.

La seconde mission était de créer un interface pour les clients de l'entreprise, cette interface ayant pour but de donner l'accès à la visualisation des différents projets.

Langages, Frameworks et librairies utilisés pour la réalisation de ce projet



HTML est développé par le W3C (World Wide Web Consortium) et le WHATWG (Web Hypertext Application Technology Working Group), le format ou langage HTML est apparu dans les années 1990. Il a progressivement subi des modifications et propose depuis 2014 une version HTML5 plus aboutie.

HTML est ce qui permet à un créateur de sites Web de gérer la manière dont le contenu de ses pages Web va s'afficher sur un écran, via le navigateur.

Il repose sur un système de balises permettant de titrer, sous-titrer, mettre en gras, etc., du texte et d'introduire des éléments interactifs comme des images, des liens, des vidéos.



CSS est l'acronyme de « Cascading Style Sheets » ce qui signifie « feuille de style en cascade ».

Le CSS correspond à un langage informatique permettant de mettre en forme des pages web (HTML ou XML).

Ce langage est donc composé des fameuses « feuilles de style en cascade » également appelées fichiers CSS (.css) et contient des éléments de codage.



Bootstrap est un framework développé par l'équipe du réseau social Twitter. Proposé en open source (sous licence MIT), ce framework utilisant les langages HTML, CSS et JavaScript fournit aux développeurs des outils pour créer un site facilement. Ce framework est pensé pour développer des sites avec un design responsive, qui s'adapte à tout type d'écran, et en priorité pour les smartphones. Il fournit des outils avec des styles déjà en place pour des typographies, des boutons, des interfaces de navigation et bien d'autres encore. On appelle ce type de framework un "Front-End Framework".



Symfony est un framework qui représente un ensemble de composants (aussi appelés librairies) PHP autonomes qui peuvent être utilisés dans des projets web privés ou open source. Mais c'est également un puissant Framework PHP développé par une société française : SensioLabs. Il permet de réaliser des sites internet dynamiques de manière rapide, structurée, et avec un développement clair. Les développeurs peuvent travailler sur ce Framework très facilement, seuls ou en équipe, grâce à la facilité de prise en main.



jQuery, est une bibliothèque JavaScript gratuite, libre et multiplateforme. Compatible avec l'ensemble des navigateurs Web (Internet Explorer, Safari, Chrome, Firefox, etc.), elle a été conçue et développée en 2006 pour faciliter l'écriture de scripts. Il s'agit du framework JavaScript le plus connu et le plus utilisé. Il permet d'agir sur les codes HTML, CSS, JavaScript et AJAX et s'exécute essentiellement côté client.



Créé en 1995, le serveur **MySQL** peut être utilisé sur de nombreux systèmes d'exploitation (Windows, Mac OS, etc.). Il supporte les langages informatiques SQL et SQL/PSM.

Dans la pratique, le serveur MySQL peut se résumer à un lieu de stockage et d'enregistrement des données, que celles-ci soient ou non cryptées. Il est alors ensuite possible, via une requête SQL, d'aller récupérer des informations sur ce serveur très rapidement.

Logiciels utilisés pour la réalisation de ce projet



Visual studio code ou VS Code est un éditeur de code développé par Microsoft en 2015. Contrairement à ce à quoi Microsoft a eu l'habitude de nous habituer durant des années, il est l'un de ces premiers produits open source et gratuit, et surtout disponible sur les systèmes d'exploitation Windows, Linux et Mac. Vs code est développé avec le framework Électron et conçu principalement pour développer des projets avec Javascript, Node.js ou encore TypeScript.



XAMPP est un ensemble de logiciels permettant de mettre en place facilement un serveur Web et un serveur FTP. Il s'agit d'une distribution de logiciels libres (X Apache MySQL Perl PHP) offrant une bonne souplesse d'utilisation, réputée pour son installation simple et rapide.

Ainsi, il est à la portée d'un grand nombre de personnes puisqu'il ne requiert pas de connaissances particulières et fonctionne, de plus, sur les systèmes d'exploitation les plus répandus.

Cette « distribution » se chargera donc d'installer l'ensemble des outils dont vous pourriez avoir besoin lors de la création d'un site Web.



ParaView est un logiciel libre de visualisation de données (voir par exemple *Représentation graphique de données statistiques*, *Représentation cartographique de données statistiques*, *Visualisation scientifique*). Il est fondé sur la bibliothèque VTK et publié sous licence BSD. Il est développé principalement par le Sandia National Laboratories (Lockheed Martin Corporation), le Los Alamos National Laboratory et la société Kitware Inc.



Figma est une application spécialisée dans la conception des interfaces Web et mobiles. Cet outil a été conçu dans l'esprit d'un Design System, partageable par une équipe de designers.



Composer est un gestionnaire de dépendances pour PHP. C'est un outil simple et fiable que les développeurs utilisent pour gérer et intégrer des paquets ou des bibliothèques externes dans leurs projets basés sur PHP.

Compétences acquises lors de la réalisation de ce projet

Développer la partie Front-end d'une application web ou web mobile en intégrant les recommandations de sécurité

- Maquetter une application
- Réaliser une interface utilisateur web statique et adaptable
- Développer une interface utilisateurs web dynamique

Développer la partie Back-end d'une application web ou web mobile en intégrant les recommandations de sécurité

- Créer une base de données
- Développer les composants d'accès aux données
- Développer la partie back-end d'une application web ou web mobile

Création du site de l'entreprise

Etude de la charte graphique

La charte graphique a été produite via le logiciel Figma, j'ai pris connaissance des différentes caractéristiques, puis j'ai fait le point avec le directeur, un premier problème c'est posé, en effet le site n'avait pas été pensé "responsive" et ne pouvait être visualisé sur un format type smartphone ou tablette.

Après réflexion avec le directeur nous avons convenu de la forme pour ces différents formats.

Voir annexe 1.1

Installation de Symfony

Après étude des différents objectifs à atteindre, j'ai fait le choix d'utiliser le Framework Symfony qui est basé sur l'architecture Modèle-Vue-Contrôleur (MVC).

Qu'est ce qu'un framework?

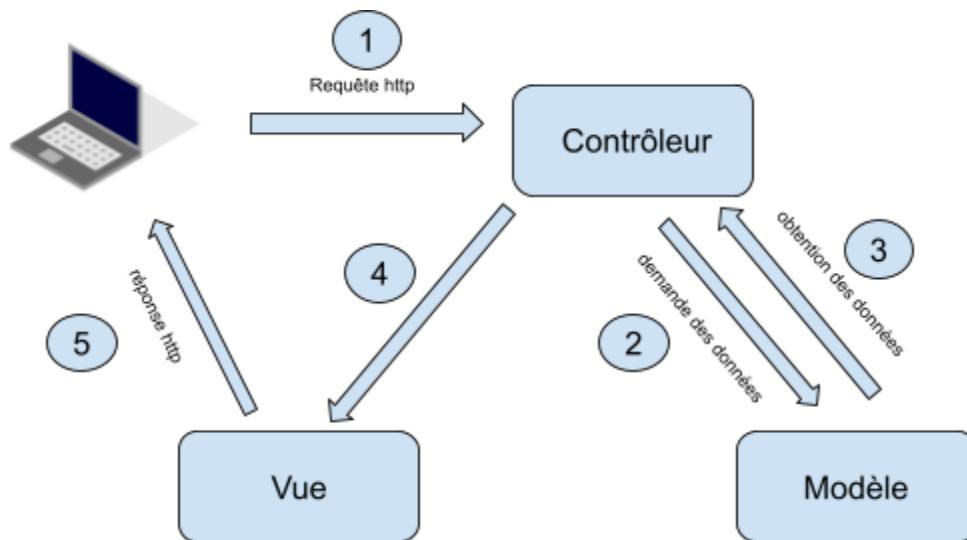
Fondamentalement, un framework se compose de :

- **Une boîte à outils** - un ensemble de composants logiciels préfabriqués et rapidement intégrables. Cela signifie que vous devrez écrire moins de code, avec moins de risque d'erreur. Cela signifie également une plus grande productivité et la possibilité de consacrer plus de temps à faire ce qui apporte une plus grande valeur ajoutée, comme la gestion des principes directeurs, des effets secondaires, etc.
- **Une méthodologie** – un « schéma d'assemblage » pour les applications. Une approche structurée peut sembler contraignante au premier abord. Mais en réalité, cela permet aux développeurs de travailler à la fois efficacement et efficacement sur les aspects les plus complexes d'une tâche, et l'utilisation des meilleures pratiques garantit la stabilité, la maintenabilité et l'évolutivité des applications que vous développez.

Basically, a framework consists of:

- **A toolbox** - *a set of prefabricated, rapidly integratable software components. This means that you will have to write less code, with less risk of error. This also means greater productivity and the ability to devote more time to doing those things which provide greater added value, such as managing guiding principles, side effects, etc.*
- **A methodology** – *an “assembly diagram” for applications. A structured approach may seem constraining at first. But in reality it allows developers to work both efficiently and effectively on the most complex aspects of a task, and the use of Best Practices guarantees the stability, maintainability and upgradeability of the applications you develop.*

Schéma de la logique MVC:



Prérequis:

Les pré-requis techniques

Avant de créer votre première application Symfony vous devez :

- Installez PHP 8.0.2 ou supérieur et ces extensions PHP (qui sont installées et activées par défaut dans la plupart des installations de PHP 8) : [ctype](#), [iconv](#), [PCRE](#), [Session](#), [SimpleXML](#) et [Tokenizer](#) ;
- Notez que toutes les nouvelles versions publiées de PHP seront prises en charge pendant la durée de vie de chaque version de Symfony (y compris les nouvelles versions majeures). Par exemple, PHP 8.1 est pris en charge.
- Installez [Composer](#), qui est utilisé pour installer les packages PHP.

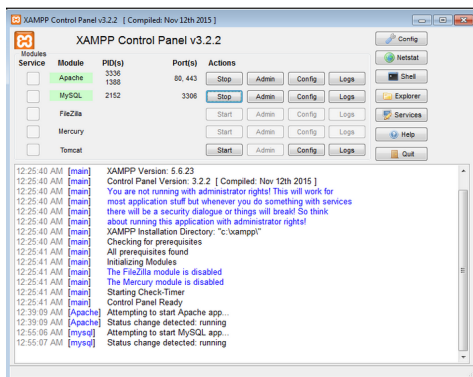
En option, vous pouvez également installer [Symfony CLI](#). Cela crée un binaire appelé `symfony` qui fournit tous les outils dont vous avez besoin pour développer et exécuter votre application Symfony localement.

Le `symfony` binaire fournit également un outil pour vérifier si votre ordinateur répond à toutes les exigences. Ouvrez votre terminal de console et exécutez cette commande :

```
$ symfony check:requirements
```

Symfony nécessite l'installation de logiciels tiers:

- un serveur web (Apache, par exemple).
- un moteur de base de données (MySQL, PostgreSQL, SQLite, ou tout PDO compatible au moteur de base de données).
- Composer.

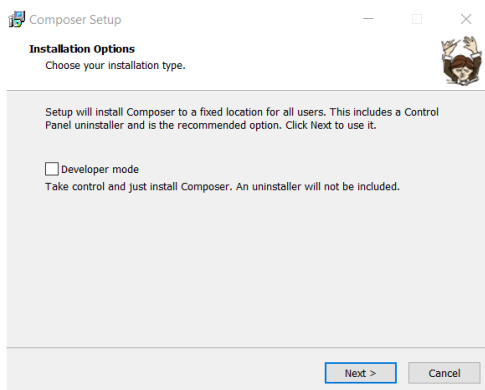


Pour le serveur Apache ainsi que le moteur de base de données j'ai utilisé le logiciel Xampp (logiciel présenté durant notre formation).

En effet ce logiciel est un ensemble de logiciels permettant de mettre en place facilement un serveur Web et moteur de base de données (Apache et MySQL) ainsi que d'autres outils.

Pour l'installation j'ai téléchargé le fichier d'installation depuis le site web <https://www.apachefriends.org/fr/index.html> tout en suivant les recommandations.

Le logiciel est prêt à fonctionner sans configuration particulière.



Composer est utilisé pour installer les packages PHP.

J'ai téléchargé le fichier depuis <https://getcomposer.org/download/> et j'ai suivi les recommandations d'installation.

Une fois les prérequis installés nous pouvons passer à la création du projet

composer create-project symfony/skeleton soleneos

Cette commande va installer le package de base Symfony.

Ensuite l'installation des différents packages utiles pour notre projet:

ORM-PACK (voir annexe)

C'est une librairie qui va faire la correspondance entre nos tables de base de données et les classes PHP. Utile dans le cas où on utilise un CRUD.

composer require symfony orm/pack

APACHE-PACK

La meilleure façon de développer votre application Symfony est d'utiliser le serveur web local de Symfony . Cependant, lors de l'exécution de l'application dans l'environnement de production, vous devez utiliser un serveur Web complet. Ce pack installe un fichier .htaccess dans le répertoire/public qui contient les règles de réécriture nécessaires pour servir l'application Symfony.

composer require symfony apache-pack

PROFILER-PACK

Ajoute la console développeur en environnement de développement dans Symfony.

Le profileur est un puissant outil de développement qui donne des informations détaillées sur l'exécution de toute requête.

```
composer require symfony profiler-pack
```

SECURITY

Le SecurityBundle fournit toutes les fonctionnalités d'authentification et d'autorisation nécessaires pour sécuriser l'application.

```
composer require security
```

MAKER-BUNDLE

Symfony Maker nous aide à créer des commandes vides, des contrôleurs, des classes de formulaires, des tests et plus encore.

Ce bundle fournit plusieurs commandes avec "make:"

```
composer require --dev symfony/maker-bundle
```

ANNOTATIONS

Les annotations ne sont pas implémentées dans PHP lui-même, c'est pourquoi ce composant offre un moyen d'utiliser les blocs de documentation PHP comme emplacement pour la syntaxe d'annotation bien connue utilisant @.

```
composer require annotations
```

ASSET

Le composant Asset gère la génération d'URL et la gestion des versions des ressources Web telles que les feuilles de style CSS, les fichiers JavaScript et les fichiers image.

```
composer require symfony/asset
```

TWIG

Moteur de template flexible, rapide et sécurisé pour PHP.

Twig est à la fois convivial pour les concepteurs et les développeurs en respectant les principes de PHP et en ajoutant des fonctionnalités utiles pour les environnements de modèles.

```
composer require twig
```

Création du Layout (Header + Footer)

Base.html.twig:

Voir annexe

Ce fichier définit le template de base de toutes les pages de notre site, toutes nos pages devront donc hériter de ce fichier.

Nous définissons plusieurs block):

- Title (Le titre qui sera affiché dans l'onglet du navigateur)
- stylesheets (si on veut ajouter des liens CDN ou du contenu css)
- Body (contenu d'une page)
- scripts (si on veut ajouter des liens CDN ou du contenu javascript)

Les blocs vont nous servir à définir des sections dans lesquelles le template enfant va pouvoir écrire.

Tout le contenu qui ne se trouve pas dans un bloc est une méthode privée, on ne peut donc pas le modifier dans les templates enfants, ce qui veut donc dire qu'on ne peut pas modifier le contenu de la balise header dans les templates enfants, parce que ceux-ci ne sont pas définis dans un bloc. Maintenant pour pouvoir voir les modifications, nous allons faire hériter notre page d'accueil (home/index.html.twig) de ce template, nous aurons donc:

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

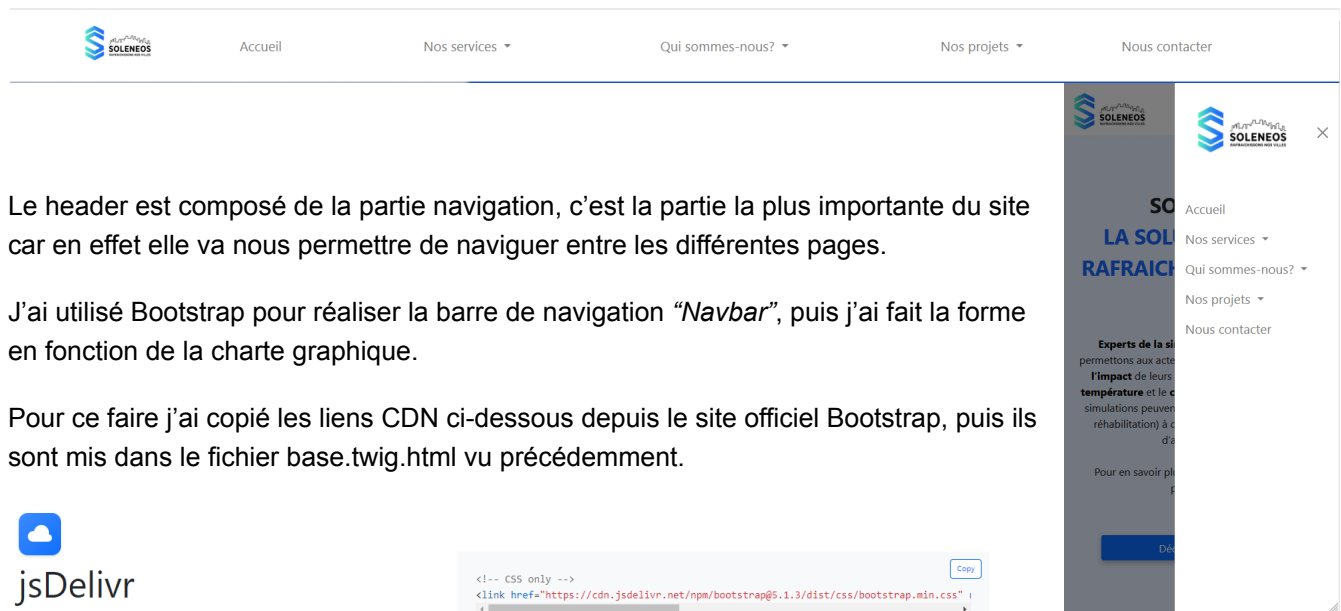
CSS:

- Utilisation du framework bootstrap via ses CDN pour la structure du site web.
- Utilisation d'un fichier personnalisé pour le contenu ainsi que pour les "media queries" (modification de la forme en fonction de la résolution).
- Utilisation d'un fichier css dépendant d'un template pour la partie "connexion".

JAVASCRIPT:

- Utilisation du CDN "Jquery" pour la partie dynamique du site web.
- Utilisation du CDN "google recaptcha" pour éviter les spams via les formulaires.
- Utilisation d'un CDN "ConsentCookie" pour la gestion des cookies.

HEADER:



Le header est composé de la partie navigation, c'est la partie la plus importante du site car en effet elle va nous permettre de naviguer entre les différentes pages.

J'ai utilisé Bootstrap pour réaliser la barre de navigation "Navbar", puis j'ai fait la forme en fonction de la charte graphique.

Pour ce faire j'ai copié les liens CDN ci-dessous depuis le site officiel Bootstrap, puis ils sont mis dans le fichier base.twig.html vu précédemment.



jsDelivr

When you only need to include Bootstrap's compiled CSS or JS, you can use [jsDelivr](#).

See it in action with our simple [starter template](#), or [browse the examples](#) to jumpstart your next project. You can also choose to include Popper and our JS [separately](#).

[Explore the docs](#)

```
<!-- CSS only -->
<link href="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/css/bootstrap.min.css" rel="stylesheet">
```

```
<!-- JavaScript Bundle with Popper -->
<script src="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/js/bootstrap.bundle.min.js" rel="script">
```

Le header est composé de deux parties:

.menu1

.menu2

Chacune dans une <DIV> différente qui sera activée ou non en fonction de la taille, pour mon projet j'ai fait le choix de faire un changement à partir de 992 pixels de largeur. (J'ai suivis les points d'arrêt de la documentation Bootstrap)

Supérieur à 992 pixels = menu1

Inférieur à 992 pixels = menu2

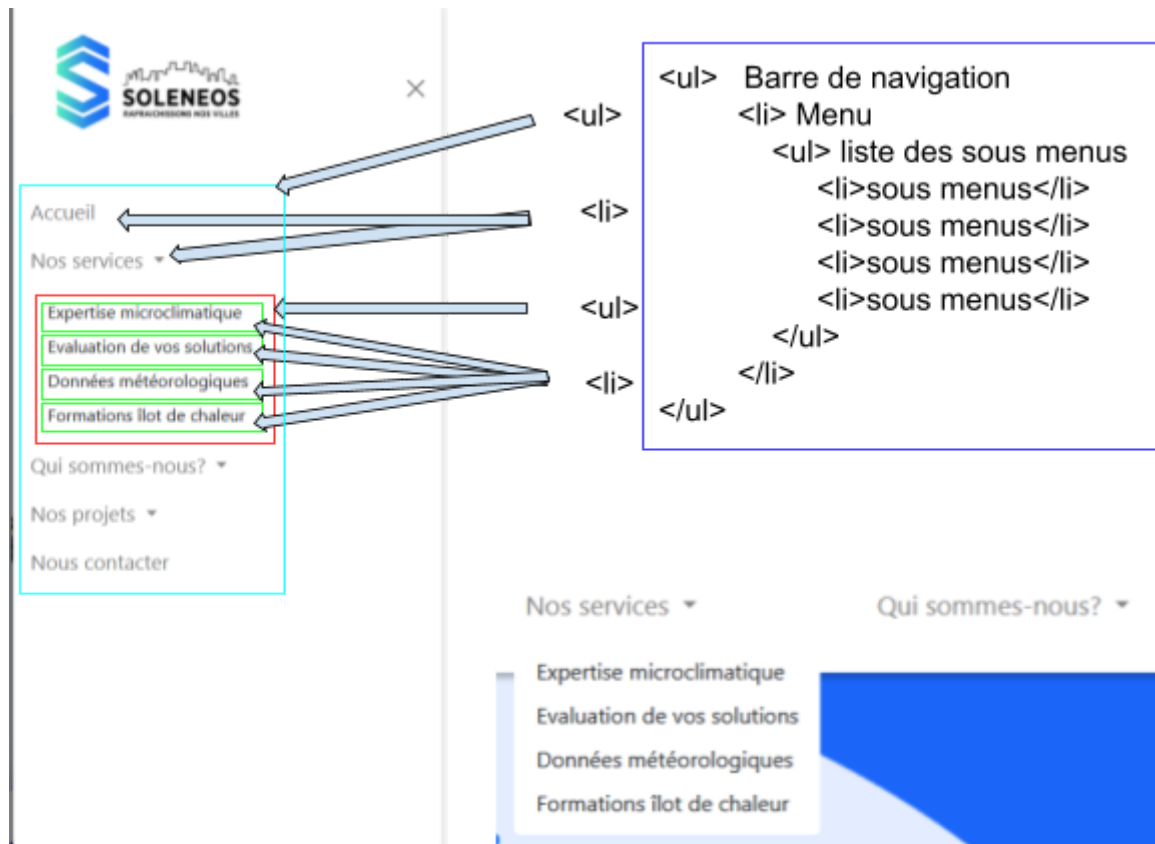
```
.menu1{
  display:none!important;
}

.menu2{
  height: 100px!important;
}
```

```
.menu1{
  height: 100px;
}

.menu2{
  display: none;
}
```

Les différentes pages sont regroupées par menu :



L'ensemble des menus sont dans une balise “``” qui représente une liste non ordonnée.

Chaque menu est une liste “``”.

Au sein de chaque menu nous avons des sous menus qui sont aussi dans une liste non ordonnée “``”.

Chaque sous menu est une liste “``”.

```
<li class="nav-item dropdown">
  <a class="nav-link dropdown-toggle" id="navbarDarkDropdownMenuLink" role="button" data-bs-toggle="dropdown" aria-expanded="false">
    Nos services
  </a>
  <ul class="dropdown-menu dropdown-menu-white" aria-labelledby="navbarDarkDropdownMenuLink">
    <li><a class="dropdown-item bloc" href="{{ path('home') }}#expertise" id="expertise" type="button">Expertise microclimatique</a></li>
    <li><a class="dropdown-item bloc" href="{{ path('home') }}#evaluation" id="evaluation" type="button">Evaluation de vos solutions</a></li>
    <li><a class="dropdown-item bloc" href="{{ path('home') }}#donnees" id="donnees" type="button">Données météorologiques</a></li>
    <li><a class="dropdown-item bloc" href="{{ path('home') }}#formations" id="formations" type="button">Formations îlot de chaleur</a></li>
  </ul>
</li>
```

FOOTER:

Le footer désigne le bas de la page, il doit donner une impression de fermer le site de manière propre et il permet au visiteur de retrouver certains éléments du site sans avoir à remonter.



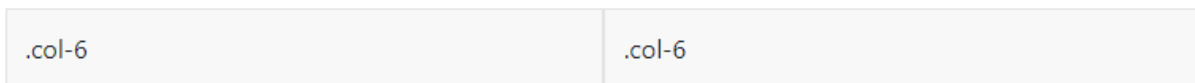
Il est composé de 5 colonnes:

- l'adresse de l'entreprise.
- Liens vers les pages importantes.
- Informations diverses.
- Liens vers les différents réseaux sociaux.
- L'inscription à la Newsletter.

Le framework Bootstrap permet de diviser un page en 12 colonnes.

Pour le footer j'ai divisé en 5 parties comme expliqué ci-dessus, les 4 première parties prennent chacune 2 colonnes et la dernière en occupe 4, ce qui nous fait un totale de 12.

On le spécifie en inscrivant "col-2" par exemple pour une largeur de 2 colonne et ainsi de suite jusqu'à 12 si on veut occuper toute la largeur.

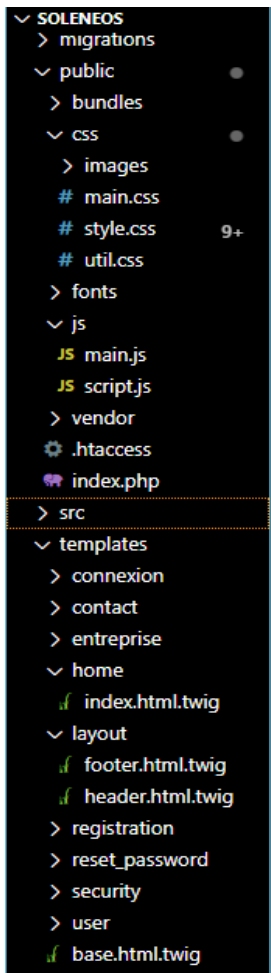


Création du contenu

index.html.twig

Ce fichier contient l'ensemble du contenu présent sur les pages j'ai fait le choix de mettre le contenu sur le même fichier en séparant les différentes pages via des commentaires.

```
{# -----changement de page "notre expertise"----- #}
```



Chaque page est au sein d'une <div> avec un identifiant unique <id>.

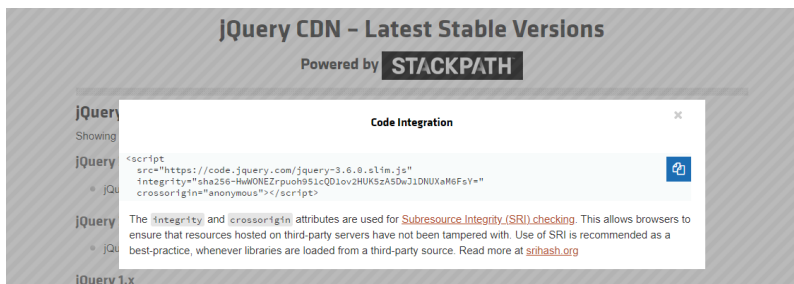
La structure de la page a été réalisée en conformité avec la charte graphique, le style est réalisé via le fichier "style.css", les scripts via le fichier "script.js" et les images sont stockées dans le dossier "public/css/images"

Au chargement du site seul la <div>, avec l'identifiant correspondant au menu accueil, est affichée. Les autres sont cachées.

Pour ce faire j'ai utilisé les technologies :

AJAX (Asynchronous Javascript and XML) permet de communiquer avec le serveur à l'aide de code Javascript en arrière-plan pendant que la page est affichée à l'écran. Ainsi le contenu de la page peut être modifié sans qu'il soit nécessaire de faire transiter et de charger une nouvelle page en entier.

Le framework JQuery qui me permet d'agir sur du code Javascript ainsi que sur l'AJAX. l'utilisation du JQuery se fait via son CDN, disponible sur le site officiel.



Explication du chargement des pages:

- Je définis la variable "\$contents" qui correspond à la classe ".page", cette classe est attribuée à chaque page
- Ensuite je demande d'afficher la première <div> ayant la classe ".page" et de cacher les autres avec "contents.slice(1).hide();"
- Lorsque qu'un visiteur va cliquer sur un des boutons de la barre de navigation, cette action va déclencher la fonction \$(".bloc").click(function()

```
// -----script pour changer de page
var $contents = $('.page');
$contents.slice(1).hide();
$('.bloc').click(function() {
    var $target = $('# ' + this.id + 'show').show();
    $contents.not($target).hide();
});
```

```
<li><a class="dropdown-item bloc" href="{ path('home') }#expertise" id="expertise" type="button">Expertise microclimatique</a></li>
<li><a class="dropdown-item bloc" href="{ path('home') }#evaluation" id="evaluation" type="button">Evaluation de vos solutions</a></li>
<li><a class="dropdown-item bloc" href="{ path('home') }#donnees" id="donnees" type="button">Données météorologiques</a></li>
<li><a class="dropdown-item bloc" href="{ path('home') }#formations" id="formations" type="button">Formations îlot de chaleur</a></li>
```

- En effet chaque bouton <a> à comme classe ".bloc" ainsi qu'un identifiant unique "id" correspondant à une page.
- L'action va définir la variable \$target qui affiche la <div> ayant comme le même "id"+ les caractère "show" à la suite `$('#' + this.id + 'show').show()`

Exemple, j'appuie sur le bouton "expertise climatique", la fonction va chercher une <div> ayant un identifiant "expertise" suivi du mot "show"

```
<div id="expertiseshow" class="page" >
```

Formulaire de contact et inscription à la newsletter

Sur ce projet il m'était demandé de mettre en place deux formulaires:

- un formulaire contact, présent sur la dernière page du site, qui permet au visiteur de prendre contact avec l'entreprise.
- un formulaire d'inscription à la newsletter de l'entreprise.

1ère étape:

Installation du composant google-mailer sur symfony, il forme un système pour créer et envoyer des e-mails - avec prise en charge des messages en plusieurs parties, intégration Twig, inlining CSS et pièces jointes.

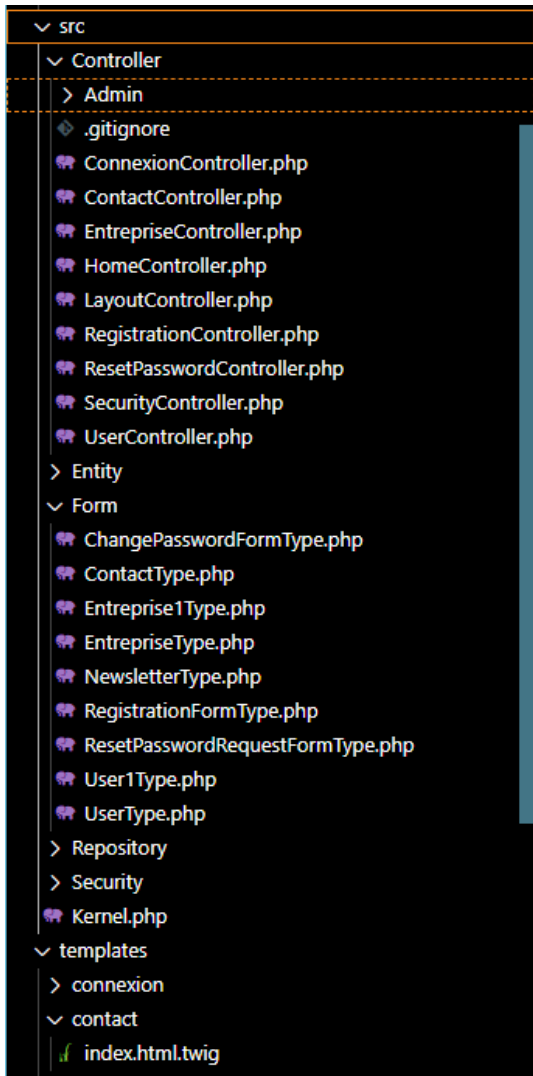
`composer require symfony/google-mailer`

Une fois installé nous pouvons aller voir le fichier ".env" et décommenter la ligne "Mailer" pour y mettre l'adresse mail ainsi que le mot de passe pour que symfony puisse avoir accès à gmail.

```
###> symfony/google-mailer ###
# Gmail SHOULD NOT be used on production, use it in development only.
MAILER_DSN=gmail://soleneos2017:
###< symfony/google-mailer ###
```


2ème étape:

installation du controller "contactController", l'installation va créer plusieurs fichiers:



- Création d'un fichier `src/controller/ContactController`, ce fichier va permettre de, créer le formulaire via le fichier "Contact Type" et rediriger la requête d'un visiteur en cas de soumission du formulaire, si l'ensemble des champs sont correctement remplis alors le message sera envoyé avec une redirection vers la page d'accueil ainsi qu'un message flash de validation, si il y a une erreur le visiteur sera amené à corriger son erreur et à soumettre à nouveau le message. On va aussi pouvoir définir l'email de destination ainsi que celui de l'expéditeur, notamment en récupérant la valeur du champ email et on pourra définir la forme du texte dans l'email de réception.

Pour finir nous allons configurer les messages flash, en déterminant le contenu du message en fonction de l'issue de la requête.

Voir annexe

- Création d'un fichier "`src/form/ContactType`" ce fichier est la classe `ContactType`, symfony y ajoute l'ensemble des champs mappés appartenant à la classe `ContactType` qui sont 'name', 'email' et 'message'.

Voir annexe

- Création du fichier `template/contact/index.html.twig` ce fichier comprend la page contact ainsi que le formulaire contact, ainsi que les différentes options pour la forme correspondant à la carte graphique. Nous ajouterons les messages flash en cas de soumission de formulaire (succès ou erreur).

Voir annexe 1.2

3ème étape :

Mise en place de l'outil Google reCAPTCHA,

L'acronyme CAPTCHA correspond à Completely Automated Public Turing test to tell Computers and Humans Apart ou en français : Test de Turing public entièrement automatique pour distinguer les ordinateurs des humains.

Le but de l'outil, comme son nom le suggère, est de détecter la présence des robots en les différenciant des humains. Pour le faire, CAPTCHA offre aux internautes de petits questionnaires sous différentes formes.

Pour installer l'outil je suis allé sur le site google recaptcha et via le compte gmail dédié à l'entreprise et j'ai configuré des clés pour le site.

Voir annexe 1.3

Pour le formulaire "newsletter" j'ai respecté le même principe de fonctionnement sans créer de Controller spécifique:

- ajout d'une fonction au fichier ContactController.
- création du fichier "src/Form/NewsletterType".
- Ajout du formulaire newsletter dans le footer + l'outil Google reCAPTCHA.

Hébergement

l'hébergement web consiste à rendre un site web accessible aux internautes en permanence et ce, via l'accès à un serveur. En tant que propriétaire du ou des serveurs, l'entreprise avec qui vous faites affaire devient donc l'hébergeur de votre site web.

l'hébergeur met à disposition des propriétaires de sites web des espaces de stockage sur des serveurs sécurisés.

Les différents types d'hébergements:

- **L'hébergement partagé permet** à de nombreux sites (pouvant se compter même en milliers) de se partager les ressources d'un seul et même serveur.
- **L'hébergement VPS** fonctionne sensiblement de la même façon que l'hébergement partagé. La première différence qui l'oppose à ce dernier est que le partage du serveur ne se fait qu'avec une petite quantité de sites web, ce qui est nettement moins que dans l'option précédente.
- **L'hébergement dédié** consiste à n'héberger qu'un seul site par serveur, ce qui est surtout utilisé pour les sites dont le taux d'achalandage par mois est très élevé (plusieurs dizaines de milliers de visiteurs) ainsi que pour ceux dont la confidentialité des informations est vitale.

L'entreprise avait déjà un ancien site sur LWS avec un hébergement de type partagé.

Quelques étapes sont nécessaires pour mettre le site en production avec le framework symfony:

Création du fichier ".env.local.php" qui va remplacer tous les autres fichiers de configuration.

```
composer dump-env prod
```

- Modification du fichier .env : APP_ENV=prod

```
16  ###> symfony/framework-bundle ###
17  APP_ENV=prod
18  APP_SECRET=24456bcffa6c355eaf017d0b29a8a8a9
19  ###< symfony/framework-bundle ###
```

- Vider le cache avant de télécharger le code.

```
php bin/console cache:clear
```

- Copie de l'ensemble des fichiers dans le dossier de l'hébergeur.

Voir annexe 1.4

```
<IfModule mod_rewrite.c>
  Options +FollowSymLinks
  RewriteBase /
  RewriteEngine On
  RewriteCond %{REQUEST_FILENAME} !-f
  RewriteRule ^(.*)$ SoleneosSymfony2/public/$1 [QSA,L]
</IfModule>
```

Modification du chemin vers le fichier index.php dans le fichier "htaccess".

Le fichier htaccess est un fichier de configuration de base utilisé par le serveur web Apache pour vous permettre de créer des règles spéciales qui indiquent à votre serveur web comment fonctionner. Il se trouve dans le dossier racine.

Il ne reste plus qu'à tester l'url pour vérifier que le site est bien en ligne!!

Création de l'accès client

Création des entités avec les relations

Nous allons procéder à la création des différentes entités qui appartiendront à une classe:

Pour l'accès clients j'ai créé plusieurs entités avec leurs attributs respectif:

- User avec ses attributs : (ID, email, rôles, password, username, isVerified).

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

Pour l'entité User on définit le nom de la class "User" par défaut, ensuite si nous souhaitons stocker les données utilisateur en base de donnée "oui" , puis nous devons définir quelle attribut correspondra au nom (ca sera "email") et pour finir, si nous souhaitons que le mot de passe soit crypté.

Deux fichiers vont être créé, le fichier src/entity/User.php et le fichier src/repository/UserRepository.php

Nous ferons la même chose pour les entités Entreprise et Projet

- Entreprise avec ses attributs : (ID, name, ville).
- projet avec ses attributs : (ID, name).

voir annexe 1.5

Relation entre les entités:

Une fois les entités et leurs attributs créés, nous allons pour voir passer à la création de relations.

Il existe les relations suivantes dans doctrine :

- OneToMany (et son inverse ManyToOne)
- ManyToMany
- OneToOne

Les relations nous permettent de lier une entité à une autre, dans mon projet j'ai étudié les besoins de l'entreprise pour définir les relations de la manière suivante:

- les "User" peuvent avoir plusieurs "projets" et un "projet" pourra avoir plusieurs "Users" c'est donc une relation ManyToMany qui sera appliquée dans ce cas là.
- Les "User" appartiendront à une seule "entreprise" et "l'entreprise" pourra avoir plusieurs "User", c'est donc une relation ManyToOne.

```
/**
 * @ORM\ManyToMany(targetEntity=Projet::class, inversedBy="users")
 */
private $projets;

/**
 * @ORM\ManyToOne(targetEntity=Entreprise::class, inversedBy="usersentreprise")
 */
private $entreprise;
```

- Une "entreprise" peut avoir plusieurs "projets" mais un "projet" n'appartient qu'à une seule "entreprise", c'est donc une relation OneToMany.

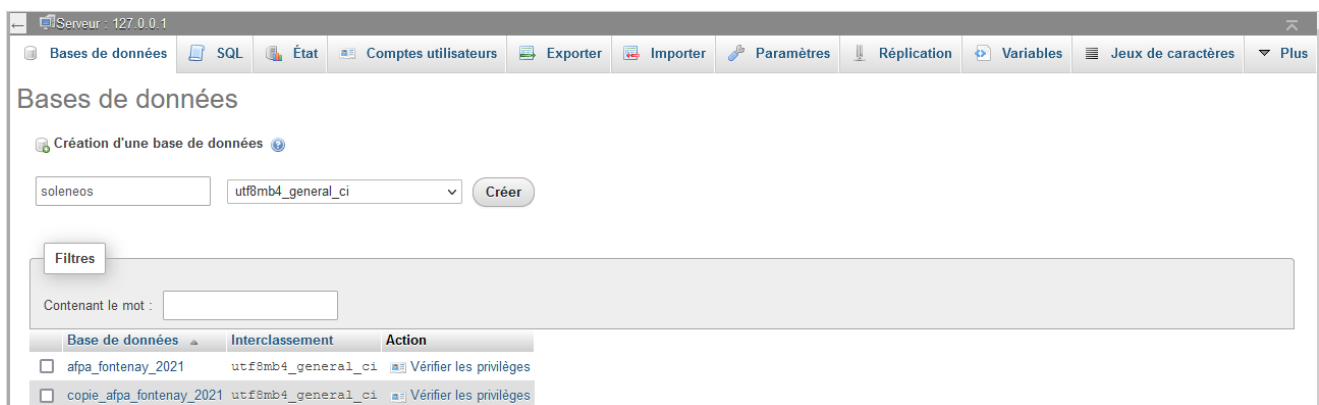
```
/**
 * @ORM\OneToMany(targetEntity=Projet::class, mappedBy="entreprise")
 */
private $Projets;
```

Création de la base de donnée

Une base de données est un ensemble d'informations qui est organisé de manière à être facilement accessible, géré et mis à jour. Elle est utilisée par les organisations comme méthode de stockage, de gestion et de récupération de l'information.

Dans le cadre de mon projet j'ai utilisé le serveur MySQL présent au sein du logiciel Xampp pour mes tests en local et je savais qu'il était aussi utilisé sur l'hébergeur.

Pour administrer et configurer la base de données j'ai utilisé l'interface PHPmyadmin.



Une fois la base créée nous allons devoir configurer symfony pour que le projet puisse communiquer avec la BDD. Pour se faire nous allons dans le fichier ".env".

```
# DATABASE_URL="sqlite:///kernel.project_dir%/var/data.db"
# DATABASE_URL="mysql://solen798828:wahtxewddi@91.216.107.182:3306/solen798828?serverVersion=mariadb-10.3.31"
# DATABASE_URL="mysql://root@127.0.0.1:3306/solen798828"
DATABASE_URL="mysql://root@127.0.0.1:3306/soleneos"
# DATABASE_URL="postgresql://symfony:ChangeMe@127.0.0.1:5432/app?serverVersion=13&charset=utf8"
###< doctrine/doctrine-bundle ###
```

ici nous mettons "root@127.0.0.1:3306" car nous utilisons le serveur mySQL sur l'ordinateur en local pour le développement sur le port 3306 par défaut.

"/soleneos" car c'est le nom de la base de données que nous avons entré dans L'interface d'administration du serveur.

Maintenant que tout est configuré, il nous faut créer un fichier de migration qui contiendra toutes les requêtes relatives à la création ou à la modification des tables.

```

PS C:\Users\AFPA\Desktop\Soleneos> php bin/console make:migration

[WARNING] You have 7 available migrations to execute.

Are you sure you wish to continue? (yes/no) [yes]:
> yes

Success!

Next: Review the new migration "migrations/Version20220217082039.php"
Then: Run the migration with php bin/console doctrine:migrations:migrate
See https://symfony.com/doc/current/bundles/DoctrineMigrationsBundle/index.html
PS C:\Users\AFPA\Desktop\Soleneos>

```

Une fois le fichier créé nous exécutons les requêtes grâce à la commande:

```

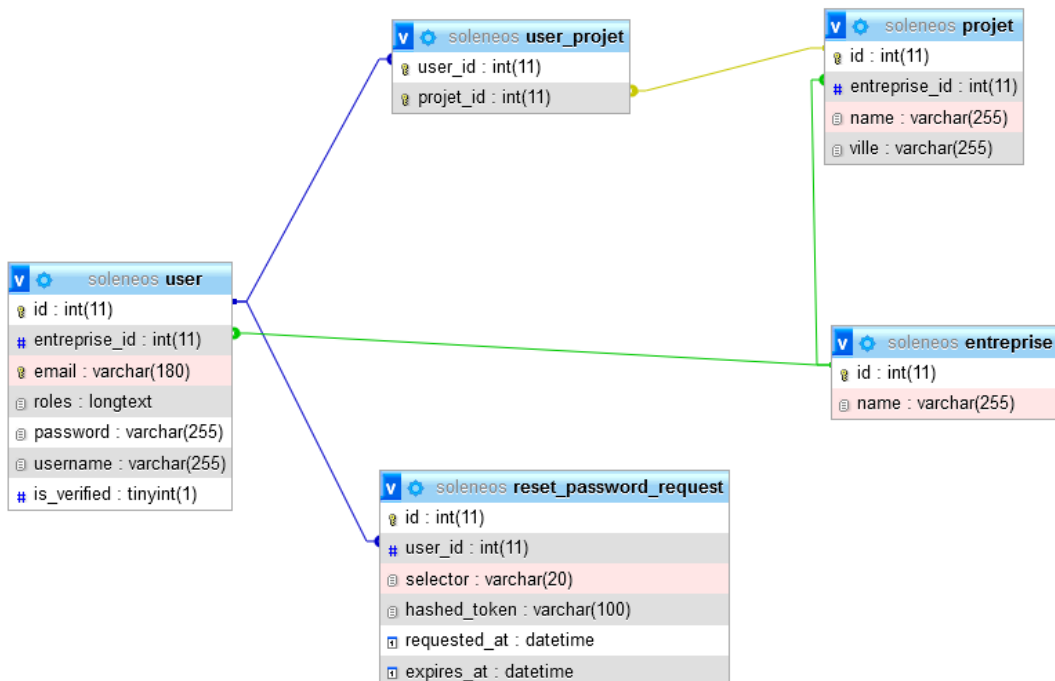
PS C:\Users\AFPA\Desktop\Soleneos> php bin/console doctrine:migrations:migrate

WARNING! You are about to execute a migration in database "soleneos" 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\Version20220217082039
[notice] finished in 458ms, used 20M memory, 1 migrations executed, 11 sql queries

```

Les tables correspondantes sont créées en base de données.



Installation du bundle Easy Admin

Le bundle EasyAdmin permet la création d'un backend d'administration pour site web.

Les tableaux de bord sont le point d'entrée des backends et ils sont liés à une ou plusieurs ressources . Les tableaux de bord affichent également un menu principal pour naviguer dans les ressources et les informations de l'utilisateur connecté.

Dans ce projet je voulais que les membres de Soleneos puissent interagir sur la base donnée de manière intuitive notamment dans la gestion des clients, des projets ainsi que des entreprises.

Le but étant d'ajouter des projets ou entreprises, de modifier les utilisateurs ou les supprimer, et de créer et ajouter les relations entre les différentes classes.

1ère étape Installation du Bundle EasyAdmin:

Pour ce faire, je me suis appuyé sur la documentation Symfony et j'ai suivi l'ensemble des recommandations.

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

2ème étape création des différents CRUD :

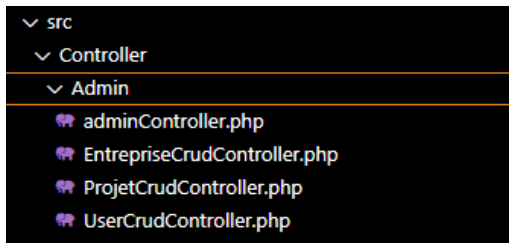
CRUD Signification : *CRUD* est un acronyme issu du monde de la programmation informatique et fait référence aux quatre fonctions considérées comme nécessaires pour implémenter une application de stockage persistant : **créer**, **lire**, **mettre à jour** et **supprimer** .

Dans une base de données relationnelle, chaque ligne d'une table est appelée tuple ou enregistrement. Chaque colonne du tableau représente un attribut ou un champ spécifique. Les quatre fonctions CRUD peuvent être appelées par les utilisateurs pour effectuer différents types d'opérations sur des données sélectionnées dans la base de données. Cela peut être accompli à l'aide de code ou via une interface utilisateur graphique.

Notre but est de pouvoir faire des modifications via une interface graphique.

Nous exécutons la commande pour créer nos différents CRUD

```
PS C:\Users\AFPA\Desktop\Soleneos> php bin/console make:admin:crud
which Doctrine entity are you going to manage with this CRUD controller?:
[0] App\Entity\DoctrineMigrationVersions
[1] App\Entity\Entreprise
[2] App\Entity\Newsletter
[3] App\Entity\Projet
[4] App\Entity\ResetPasswordRequest
[5] App\Entity\User
```



Nous sélectionnons les entités que nous voulons administrer:

- User.
- Projet.
- Entreprise.

Nous pouvons constater la création de plusieurs fichiers:

- Le fichier "adminController.php", ce fichier va nous permettre de définir quelle classe nous voulons administrer depuis le dashboard, nous pouvons aussi personnaliser le dashboard.

```
public function configureMenuItems(): iterable
{
    // yield MenuItem::linkToDashboard('Dashboard', 'fa fa-home');
    // yield MenuItem::linkToCrud('The Label', 'fas fa-list', EntityClass::class);
    yield MenuItem::linkToRoute("Aller vers la page de visualisation", 'fas fa-home', 'connexion');
    yield MenuItem::linkToRoute("Retour vers Soleneos.fr", 'fas fa-home', 'home');
    yield MenuItem::linkToCrud('Utilisateurs', 'fa fa-user', User::class);
    yield MenuItem::linkToCrud('Entreprises', 'fas fa-building', Entreprise::class);
    yield MenuItem::linkToCrud('Projets', 'fas fa-folder-open', Projet::class);
}
```

J'ai ajouté deux liens pour pouvoir naviguer vers le site web de Soleneos ou alors continuer vers la page de connexion, ensuite j'ai ajouté les 3 entités ("user", "projet" et "entreprise").

- Le fichier "UserCrudController.php", ce fichier me permet de définir les informations de l'entité "User" que je veux voir apparaître dans mon dashboard.

```
public function configureFields(string $pageName): iterable
{
    return [
        TextField::new('username'),
        EmailField::new('email'),
        ChoiceField::new('roles')->allowMultipleChoices()->autocomplete()
            ->setChoices([
                'User' => 'ROLE_USER',
                'Admin' => 'ROLE_ADMIN'
            ]),
        AssociationField::new('entreprise'),
        CollectionField::new('projets'),
        AssociationField::new('projets')->hideOnIndex(),
    ];
}
```

Les champs "field" de Symfony permettent d'afficher le contenu des entités Doctrine sur chaque page CRUD. EasyAdmin fournit des champs intégrés pour afficher tous les types de données courants, mais nous pouvons également créer nos propres champs.

J'utilise:

Le champ "TextField" pour faire apparaître nom et prénom de l'utilisateur via l'attribut "username".

Le champ "EmailField" pour faire apparaître l'email du client en appelant l'attribut "email".

Le champ "ChoiceField" qui va me servir à modifier le rôle de l'utilisateur, et je définis la liste des rôles disponible (je reviendrai plus tard sur les "rôles").

le champ "AssociationField" pour faire apparaître le nom de l'entreprise grâce à la relation entre "User" et "Entreprise" en appelant l'attribut "entreprise de ma relation".

Le champ "CollectionField" pour faire apparaître le(s) nom(s) du ou des projet(s) grâce à la relation entre "User" et "Projet" en appelant l'attribut "Projets".

Le champ "AssociationField" pour faire apparaître le(s) nom(s) du ou des projets mais sous forme de liste qui sera visible seulement dans la partie création ou modification d'un utilisateur.

- Le fichier "ProjetCrudController.php", ce fichier me permet de définir les informations de l'entité "Projet" que je veux voir apparaître dans mon dashboard.

```
public function configureFields(string $pageName): iterable
{
    return [
        TextField::new('name'),
        TextField::new('ville'),
        CollectionField::new('users'),
    ];
}
```

J'utilise :

Le champ "TextField" pour faire apparaître le nom des projets via l'attribut "name" ainsi que la ville via l'attribut "ville".

Le champ "CollectionField" pour faire apparaître les utilisateurs en relation avec le projet via l'attribut "users".

- Le fichier "EntrepriseCrudController.php", ce fichier me permet de définir les informations de l'entité "Entreprise" que je veux voir apparaître dans mon dashboard.

```
public function configureFields(string $pageName): iterable
{
    return [
        TextField::new('name'),
        CollectionField::new('projet'),
        CollectionField::new('usersentreprise'),
    ];
}
```

J'utilise:

Le champ "TextField" pour faire apparaître le nom des entreprises via l'attribut "name".

le champ "CollectionField" pour faire apparaître les projets en relation avec l'entreprise via l'attribut "projet" ainsi que les clients appartenant à l'entreprise via l'attribut "userentreprise".

3ème étape, sécurisation du Dashboard:

Le but du dashboard est qu'il doit être accessible pour certain membre de l'entreprise, c'est à ce moment là qu'interviennent les "rôles", nous avons pu voir précédemment l'attribut "rôles" dans l'entité "User". Cet attribut va nous permettre de définir un rôle à chaque utilisateur. De base chaque utilisateur créé a le "role_user", si nous voulons qu'un utilisateur ait accès au dashboard il faudra alors lui attribuer le "role_admin".

Tout d'abord nous allons laisser la sécurité désactivée pour pouvoir attribuer le rôle admin à un utilisateur. Pour ce faire nous allons sur le dashboard via "soleneos.fr/admin", en effet cette route est déterminée dans le fichier "adminController".

```
#[Route('/admin', name: 'admin')]
public function index(): Response
```

Une fois sur le dashboard il nous suffit de sélectionner un utilisateur et grâce à l'Update du CRUD que nous avons réalisé, nous allons donc pouvoir mettre à jour le rôle de l'utilisateur.

Modifier un utilisateur

Via la liste déroulante nous pouvons ajouter le rôle admin.

Ensuite il nous faut définir que l'accès au dashboard ne se fera qu'avec le rôle admin, pour ce faire nous allons configurer le fichier "security.yaml" dans ce fichier je vais spécifier que la route /admin ne sera accessible qu'avec le "ROLE_ADMIN"

```
access_control:
- { path: ^/admin, roles: ROLE_ADMIN }
# - { path: ^/*, roles: ROLE_USER }
```

Cette action redirige tous les utilisateurs qui n'ont pas le rôle admin vers une page avec message d'information.

Voir annexe 1.7

Création du formulaire d'inscription et d'authentification

Pour créer des utilisateurs il nous faut fournir un moyen de créer un compte depuis le site web, pour se faire nous allons donc créer un formulaire d'enregistrement depuis la console.

```
php bin/console make:registration-form
```

Cette commande va générer des fichiers suivants:

- `RegistrationController.php`, ce fichier va nous permettre de déterminer la route pour atteindre la page d'enregistrement et surtout il va nous permettre de configurer les différentes fonctions liées aux soumissions de formulaire lors d'un enregistrement. Notamment en cas de réussite de la soumission nous allons définir l'envoi d'un email de confirmation de compte ainsi qu'un message de confirmation.
- `"RegistrationFormType"`, ce fichier va définir les différents attributs à remplir pour la soumission du formulaire avec les différentes options concernant l'attribut password et on va pouvoir définir le nombre de caractère minimum et maximum ainsi que les messages d'erreur en cas de non validation du password.
- `"register.html.twig"`, ce fichier nous permet de créer le rendu visuel avec bien évidemment la mise en place des différents attributs nécessaire à un enregistrement, mais aussi la possibilité de revenir vers l'accueil ou le site "soleneos.fr".

Le mot de passe est automatiquement crypté via le protocole de cryptage *bcrypt*.

```
password_hashers:
    Symfony\Component\Security\Core\User\PasswordAuthenticatedUserInterface:
        'auto'
    App\Entity\User:
        algorithm: auto
        cost: 4
```

Nous pouvons désormais créer un utilisateur, l'étape suivante va consister à créer le formulaire d'authentification *login* pour accéder à l'espace client.

```
php bin/console make:auth
```

Cette commande va générer les fichiers suivants:

- SecurityController.php, ce fichier permet de définir la route à atteindre pour s'authentifier, pour le site j'ai décidé de mettre *soleneos.fr/login* mais aussi en cas de session déjà activé l'utilisateur sera redirigé vers la page *connexion*.

```
/**
 * @Route("/login", name="app_login")
 */
public function login(AuthenticationUtils $authenticationUtils): Response
{
```

```
    if ($this->getUser()) {
        return $this->redirectToRoute('connexion');
    }
```

- LoginFormAuthenticator.php, ce fichier permet de déterminer les fonctions d'authentification notamment pour la partie sécurité avec le PassportInterface.

Un Passport est une classe qui va contenir les informations ayant besoin d'être validées durant le travail d'authentification et ces informations seront transportées avec la notion de **Badge**, qui sert à ajouter des informations au passeport pour étendre la sécurité.

UserBadge PasswordCredentials CsrfTokenBadge sont des badges qui doivent implémenter une interface *BadgeInterface*. Cette interface a une méthode, *isResolved*, et celle-ci doit retourner true

- UserBadge va résoudre l'utilisateur via un Provider défini dans la configuration ou un callable qu'on peut passer en deuxième argument du constructeur.
- PasswordCredentials va checker le password.
- CsrfTokenBadge va vérifier que le token CSRF est valide.
- Passport va se charger de transporter tout ça.

J'ai aussi configuré la redirection à la suite d'une l'authentification en fonction du rôle de l'utilisateur, la fonction **onAuthenticationSuccess** va vérifier l'attribut "rôles" et en si l'ensemble des caractères "ROLE_ADMIN" sont présent alors l'utilisateur sera redirigé directement vers le dashboard sinon il sera considéré comme un utilisateur et sera dirigé vers la page de "connexion"

- Login.html.twig, ce fichier va permettre de rendre la vue à l'utilisateur on va y insérer, les différents champs pour soumettre l'authentification c'est à dire l'email et le mot de passe, j'ai aussi ajouté un lien pour retourner sur le site "soleneos.fr" ainsi qu'un lien pour réinitialiser le mot de passe.

Nous allons voir le processus de réinitialisation de mot de passe:

```
php bin/console make:reset-password
```

Cette commande va générer les fichiers suivants:

- ResetPasswordController.php, ce fichier va définir la route ainsi que les différentes fonction pour générer le formulaire en cas de requête, plusieurs étapes sont nécessaire, tout d'abord la requête qui nous envoi sur une page où l'on va devoir inscrire notre adresse mail, la fonction checkEmail permet de vérifier si l'email existe dans la base de donnée, si ce n'est pas le cas un faux token sera généré sinon un resettoken sera émis. L'utilisateur reçoit alors un mail avec un liens valide 1 heure ou il va pouvoir renseigner son nouveau mot de passe qui sera validé s'il a bien respecté les conditions..

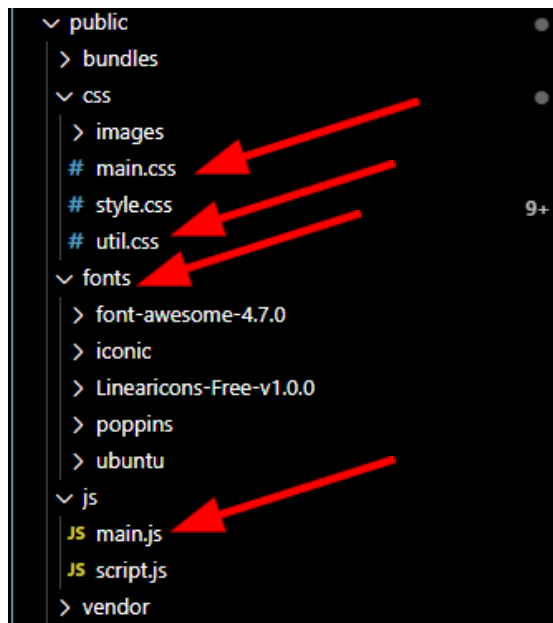
- ResetPasswordRequest.php, qui est une entité avec les différents attributs nécessaires à la réinitialisation, notamment "id" et "user".

- ChangePasswordFormType.php, qui va générer le formulaire avec les attributs nécessaires à la réinitialisation du password, dans ce cas il faudra inscrire deux fois le même mot de passe avec un minimum de 6 caractères et un max de 4096 caractères, j'ai aussi configuré les différents messages en cas d'erreur.
- ResetPasswordRequestFormType.php, qui va générer le formulaire pour renseigner son email lors de la demande de réinitialisation.

Nous avons donc terminé l'ensemble des configurations concernant la création et l'authentification des utilisateurs.

Configuration de la page "connexion"

La page de connexion est la page d'accueil lorsque l'utilisateur se connecte, cette page va lui permettre de choisir le projet qu'il souhaite visionner, si il a plusieurs projets alors il aura la liste de tous ses projets.



j'ai utilisé un template gratuit pour la forme de la page ce même template à été utilisé pour la page d'authentification et d'enregistrement.

J'ai récupéré le contenu css ainsi que le contenu javascript que j'ai inséré dans mon projets (main.css, util.css, dossier fonts, main.js).

j'ai utilisé le code html du template en l'adaptant selon mes besoins.

J'ai ajouter l'ensemble des liens vers ces fichiers dans les blocks de la page connexion

Pour le contenu de la page j'ai rappelé le champ "username" pour pouvoir afficher le nom de l'utilisateur, j'ai aussi ajouté un lien vers le dashboard si l'utilisateur a le rôle admin puis j'ai mis un lien se déconnecter.

```
<div class="bandeaulogin">
  {% if app.user %}
    Bonjour {{ app.user.username }} !<br>
    <a href="{{ path('app_logout') }}">Se déconnecter</a><p>
  {% endif %}

  {% if is_granted('ROLE_ADMIN') %}
    <a href="{{ path('admin') }}">Dashboard</a>
  {% endif %}
</div>
```

Ensuite je décide de faire apparaître la liste des projets en relation avec l'utilisateur en appelant attributs projet en relation avec la classe "user".

```
{% if app.user.projets %}
  <span class="login100-form-title p-b-41">
    Connectez vous à votre espace de visualisation
  </span>
  {% endif %}

  {% for projets in app.user.projets %}

    <div class="container-login100-form-btn m-t-32">
      <button class="login100-form-btn" type="button">
<a href="http://74.82.29.5/?data={{ projets.name }}" target="_blank">{{ projets.name
}}</a></button>
```

Si un projet est présent en base de données alors nous le verrons apparaître, j'ai créé un bouton à destination du serveur de visualisation tout en spécifiant le nom du projet que souhaite voir ce nom sera spécifié dans l'url.

```
{% else %}
  <span class="login100-form-title p-b-41">
    Vous n'avez pas encore de projet ?<br>N'hésitez pas à aller essayer notre exemple de
visualisation!
  </span>
  <div class="container-login100-form-btn m-t-32">
    <button class="login100-form-btn" type="button">
    <a href="http://74.82.29.5/?data=présentation" target="_blank">Voir la présentation</a></button>
  </div>
{% endfor %}
```

Si l'utilisateur n'est pas relié à un projet alors il aura un lien de présentation en attendant que l'entreprise mette en place le projet.

J'ai ajouté l'option "target_blank" qui permet d'ouvrir la page de visualisation sur un autre onglet.

Mise en oeuvre du serveur avec le logiciel Paraview

Le deuxième objectif était de créer un espace de visualisation pour les clients de l'entreprise, en effet l'entreprise travaillait sur des projets d'urbanisme 3D en collaboration avec les différents architectes responsables des projets.

Pour consulter les différentes simulation, les comptes rendus se faisaient via des fichiers pdf sur demande du client ce qui pouvait prendre du temps à l'entreprise.

Nous avons réfléchi à un moyen de permettre aux différents clients d'avoir accès au visuel des projets de manière autonome.

Pour se faire nous nous sommes appuyé sur le framework paraviewWeb du logiciel Paraview, cette bibliothèque JavaScript est un framework Web pour créer des applications avec une visualisation scientifique interactive à l'intérieur du navigateur Web. Ces applications peuvent tirer parti d'un backend VTK et/ou ParaView pour le traitement et le rendu de données volumineuses, mais peuvent également être utilisées sur un serveur Web statique comme Apache ou NGINX, un serveur HTTP hautes performances ou même localement avec une application basée sur la ligne de commande utilisant votre navigateur.

Nous avons utilisé l'application ParaView Visualizer, c'est une application Web qui permet la visualisation scientifique sur le Web à l'aide d'un backend ParaView pour le traitement et le rendu des données.

1ère étape:

Définir sur quelle support nous allons installer le logiciel et mettre en œuvre le serveur, en suivant la documentation du logiciel je me suis rendu compte que le logiciel nécessite certaines ressources comme par exemple une carte graphique, effectivement la visualisation 3D est assez gourmande en terme de ressource. Il existe des versions allégées du logiciel mais plus limitées.

N'ayant pas de serveur au sein de l'entreprise, j'ai fait le choix d'héberger le serveur via un fournisseur de cloud computing, en effet il existe de nombreuses solutions pour louer des ressources via le cloud en fonction des besoins, cependant cela nécessite une bonne connexion internet.

J'ai choisie pour mes tests de passer par "Paperspace" c'est une ESN qui fournit des ressources informatique via le cloud, certains de ses produits correspondant à mes recherches ainsi que la facilité de mise en oeuvre ont permis de valider mon choix.

Voir annexe 1.8

J'ai choisi un serveur sous windows avec une carte graphique, l'installation est très rapide, une fois le compte créé nous avons accès à l'ordinateur via le site ou l'application Paperspace ensuite il nous reste plus qu'à installer les différents composants sur le serveur.

Mise en place du server Apache

Apache est le logiciel de serveur Web le plus utilisé actuellement. En effet, ce dernier est installé sur plus de 46% des serveurs Web actifs. Comme son nom l'indique, il permet de servir des contenus Web, c'est la raison pour laquelle il se nomme serveur Web.

Apache est l'un des plus vieux serveurs Web existant actuellement, créé en 1995 et développé par Apache Software Foundation.

Le serveur Web quant à lui permet d'afficher un site Web. Dès que vous demandez l'affichage d'une page Web, le serveur Web va extraire le contenu et vous le transmettre.

Cependant, l'un des défis les plus grands est de pouvoir servir plusieurs utilisateurs simultanément dans la mesure où chacun va demander l'affichage d'une page différente. De plus, un serveur Web est capable de lire et traiter différents langages de programmation tels que PHP, Python, HTML, Javascript.

Pour mettre en œuvre le serveur apache j'ai installé le logiciel Xampp qui contient une distribution apache complète.

il ne me reste plus qu'à configurer les différents fichiers de la distribution apache pour que mon serveur soit atteignable depuis le web.

J'ai du modifier fichier httpd.conf pour rediriger l'ensemble du trafic vers le logiciel paraview:

Listen 80 J'accepte les connexions entrant sur le port 80.

Ensuite le fichier "httpd-vhosts.conf".

je configure un hôte virtuel qui sera utilisé pour diriger tout le trafic vers ParaviewWeb :

```
<VirtualHost *:80>
    ServerName 74.82.29.5
    DocumentRoot C:\ParaView-5.10.0\share\paraview-5.10\web\visualizer\www
    ErrorLog C:\ParaView-5.10.0\error.log
    CustomLog C:\ParaView-5.10.0\log\apache2\access.log combined
    <Directory "C:\ParaView-5.10.0">

        </Directory>

        # Handle launcher forwarding
        # port and endpoint should match launcher.config
        ProxyPass /paraview http://localhost:9000/paraview

        # Handle WebSocket forwarding
        RewriteEngine On

        # This is the path the mapping file Jetty creates
        # path to proxy should match launcher.config
        RewriteMap session-to-port txt:C:\xampp\proxy.txt

        # This is the rewrite condition. Look for anything with a sessionId= in the
        # query part of the URL and capture the value to use below.
        RewriteCond %{QUERY_STRING} ^sessionId=(.*)&path=(.*)$ [NC]

        # This does the rewrite using the mapping file and the sessionId
        RewriteRule ^/proxy.*$ ws://${session-to-port:%1}/%2 [P]

    </VirtualHost>
```

Une fois les hôtes virtuels configurés, la configuration du lanceur ParaView Web doit être créée afin que les demandes de ParaView Web soient traitées de manière appropriée. Pour cela, créez un fichier nommé **launcher.config** dans le répertoire d'installation de ParaView.

Voir annexe 1.9

Ce fichier va définir le port sur lequel va être configuré le serveur paraview, le chemin où se trouve le contenu de l'application ainsi que les différentes options sur les connexion à distance notamment l'attribution des port 9001 à 9999), mais aussi le délai de tentative de connexion .

dans le script ci-dessus, les propriétés suivantes sont transmises :

- **dataFile** : le nom du fichier à rendre dans le Visualizer.
- **dataDir** : l'emplacement du répertoire de données sous lequel le ou les fichiers de données sont présents.

Si l'emplacement dataDir est fixe, il peut être codé en dur dans le script et seul le nom dataFile doit être transmis. Dans un tel cas, le script suivant doit être utilisé :

Démarrage depuis la ligne de commande

Ouvrez une invite de commande pour exécuter chacun des éléments suivants :

```
C:\Windows\system32>C:\ParaView-5.10.0\bin\pvpython.exe C:\ParaView-5.10.0\bin\Lib\site-packages\wslink\launcher.py  
C:\ParaView-5.10.0\launcher.config  
===== Running on http://localhost:9000 =====  
(Press CTRL+C to quit)
```

J'ai ajouté ce script dans les tâches à lancer au démarrage du serveur pour pouvoir limiter au maximum les actions des membres de l'entreprise.

Conclusion et remerciement

Ce projet m'a permis d'approfondir mes connaissances en développement notamment via le framework Symfony qui est un outil de développement très puissant, il m'a permis de créer un site web structuré.

J'ai dû apprendre à respecter des délais mais aussi à proposer des solutions, j'ai pu travailler en collaboration avec les membres de l'entreprise mais aussi de façon autonome en télétravail. j'ai pu développer de nouvelle façon de travailler, organiser mes recherches sur le web, j'ai très vite compris que la documentation était très importante notamment dans l'utilisation de différents langages, librairies ou frameworks.

Ce stage m'a permis aussi de m'intéresser à de nouvelles technologies dans le but de proposer les solutions les plus pertinentes.

J'ai également beaucoup appris sur les éléments de sécurité web et sur la protection des données d'utilisateurs. La conformité aux règles RGPD mises en place par la CNIL, m'a obligé à revoir ma façon de coder.

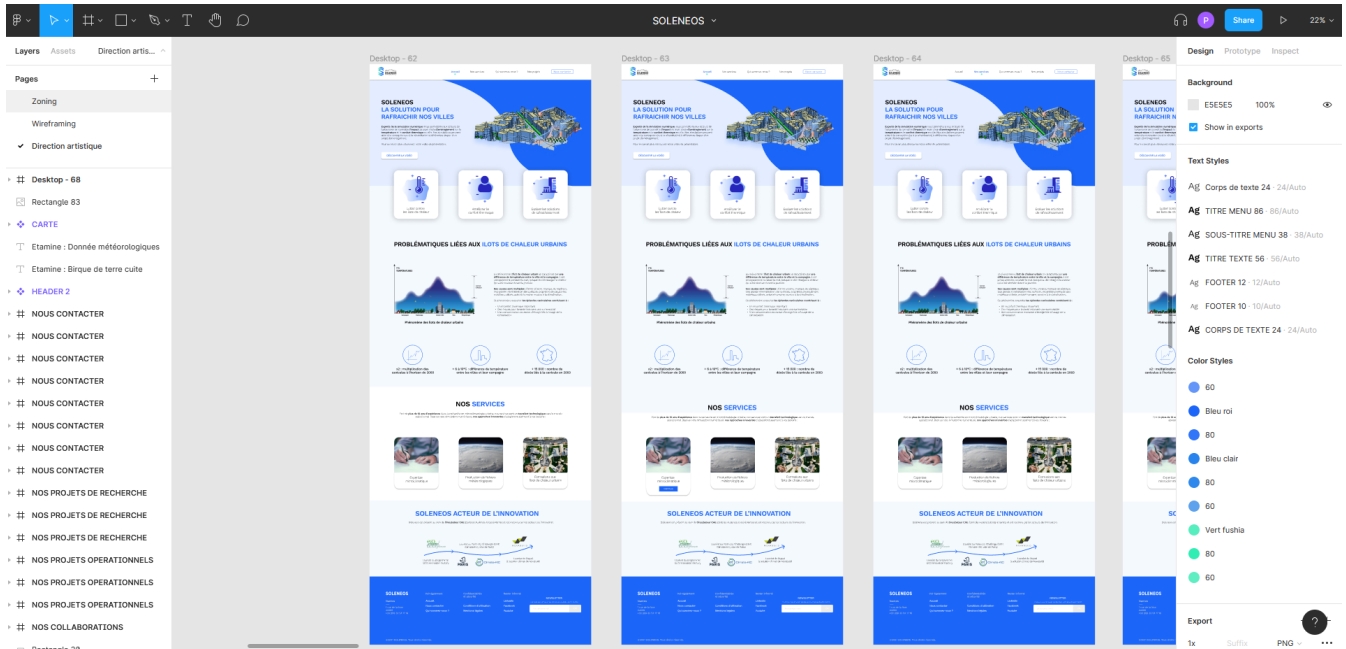
J'ai maintenant une vision globale sur les différentes étapes de construction d'un projets ce qui me permettra avec l'expérience de mieux juger les coûts (humain, technologique et monétaire) ainsi que le temps pour réaliser un projet.

Néanmoins c'est un domaine qui ne cesse d'évoluer et qui demande un investissement personnel continu pour rester cohérent dans la réalisation des projets futurs. En effet, nous pouvons voir que les technologies évoluent très vite car la concurrence entre les acteurs du numérique est très forte et de nouveaux horizons s'ouvrent comme par exemple le métavers ou l'IA de plus en plus présente..

Je remercie Jérôme, notre formateur, de nous avoir guidé tout au long de cette formation ainsi que tous mes camarades de formation avec qui j'ai partagé durant ces 9 mois. Merci à la société Soleneos et notamment le Mr Benjamin Morille, de m'avoir fait confiance pour réaliser ce projet, ils m'ont très bien accueilli et j'ai tout de suite trouvé ma place au sein de cette société. Je remercie également la communauté web qui, à travers les différents forums et documentations, m'a permis d'enrichir mes connaissances.

ANNEXE

1.1 Figma:



1.2 Formulaire contact

ContactType.php

```
1 <?php
2
3 namespace App\Form;
4
5 use Symfony\Component\Form\AbstractType;
6 use Symfony\Component\Form\FormBuilderInterface;
7 use Symfony\Component\OptionsResolver\OptionsResolver;
8 use Symfony\Component\Form\Extension\Core\Type\EmailType;
9 use Symfony\Component\Form\Extension\Core\Type\TextareaType;
10 use Symfony\Component\Form\Extension\Core\Type\TextType;
11
12 class ContactType extends AbstractType
13 {
14     public function buildForm(FormBuilderInterface $builder, array $options): void
15     {
16         $builder
17             ->add('name', TextType::class)
18             ->add('email', EmailType::class)
19             ->add('message', TextareaType::class)
20         ;
21     }
22
23     public function configureOptions(OptionsResolver $resolver): void
24     {
25         $resolver->setDefaults([
26             // Configure your form options here
27         ]);
28     }
29 }
30
```

ContactController

```
/**
 * @Route("/contact", name="contact")
 */
public function contact(Request $request, MailerInterface $mailer)
{
    $form = $this->createForm(ContactType::class);

    $form->handleRequest($request);
    $recaptcha = new ReCaptcha('6LepclUeAAAAAC4cynLmLjkNdnqsLzqR52QPEXYS');
    $resp = $recaptcha->verify($request->request->get('g-recaptcha-response'), $request->getClientIp());

    if($form->isSubmitted() && $form->isValid() && $resp->isSuccess()) {
        $contactFormData = $form->getData();

        $message = (new Email())
            ->from($contactFormData['email'])
            ->to('contact@soleneos.fr')
            ->subject('Vous avez un message depuis la page contact de Soleneos')
            ->text('Contact : '.$contactFormData['email'].'\PHP_EOL.
                | Nom : '.$contactFormData['name'].'\PHP_EOL.
                | Message : '.$contactFormData['message'],
                'text/plain');
        $mailer->send($message);

        $this->addFlash('notice', 'Votre message a été envoyé!');

        return $this->redirectToRoute('home');
    } elseif($form->isSubmitted() && !$resp->isSuccess()) {
        $this->addFlash('error', "Erreur, veuillez recommencer, n'oubliez pas de cocher 'je ne suis pas un robot.'");
    }

    return $this->render('contact/index.html.twig', [
        'our_form' => $form->createView()
    ]);
}
```

contact/index.html

```
{{ form_start(our_form) }}
<div class="row formulaire justify-content-center align-content-center" method="post" action="formulaire.php" id="form">
    <div class="inputEmail col-10">
        {{ form_row(our_form.email, {'attr': {'class': 'form-control', 'placeholder': 'E-mail', 'label': 'none'}, 'label_attr': {'style': 'display: none'}})}}
    </div>
    <div class="InputName col-10">
        {{ form_row(our_form.name, {'attr': {'class': 'form-control', 'placeholder': 'Nom'}, 'label_attr': {'style': 'display: none'}})}}
    </div>
    <div class="InputMessage col-10">
        {{ form_row(our_form.message, {'attr': {'class': 'form-control', 'placeholder': 'Message'}, 'label_attr': {'style': 'display: none'}})}}
    </div>
    <p></p>
    <div class="g-recaptcha" name="captcha" data-sitekey="6LepclUeAAAAAB1RbWxvLN7RtZX0dEKruY6HFYx" required>
    </div>
    <p></p>
    <input type="submit" value="submit" class="btnFormulaire btn btn-primary">
    </div>
{{ form_end(our_form) }}
```

```
</div>
```

1.3 Google REcaptcha

Libellé ⓘ

soleneos

8 / 50

Type de reCAPTCHA : v2 Case à cocher

Clés reCAPTCHA ^

Utilisez cette clé de site dans le code HTML de votre site destiné aux utilisateurs. [Voir l'intégration côté client](#)

🔑 COPIER LA CLÉ DU
SITE

6Lepo[REDACTED]P1BLMMk...LN7D+ZYKdEKruY6HFYx

Utilisez cette clé secrète pour la communication entre votre site et le service reCAPTCHA.

[Voir l'intégration côté serveur](#)

🔑 COPIER LA CLÉ
SECRÈTE

6Lep[REDACTED]IsAAAAA...r5ynHmEjK...f...qR52QPEXYS

Domaines ⓘ

✕ www.soleneos.fr

+ Ajouter un domaine, par exemple, exemple.com

Propriétaires

✕ soleneos2017@gmail.com

+👤 Saisir des adresses e-mail

page contact


E-mail

Nom

Message

☐

Je ne suis pas un robot


reCAPTCHA
Confidentialité - Conditions

submit

1.4 Hébergement:

WS Panel

Gestionnaire de fichiers

soleneos.fr

SoleneosSymfony2

Plein écran

Rechercher un fichier ou un dossier

Réduire tout

Remonter

Sélectionner tout

Désélectionner tout

Recharger

Dossier

Fichier

Copier

Déplacer

Charger

Télécharger

Supprimer

Archiver

Extraire

Renommer

Modifier

Permissions

Nom	Taille	Modifié le	Type	Perms
bin		18/02/2022 17:50:09	directory	0755
config		18/02/2022 17:50:09	directory	0755
migrations		18/02/2022 17:50:09	directory	0755
public		18/02/2022 17:50:10	directory	0755
src		18/02/2022 17:50:10	directory	0755
templates		18/02/2022 17:50:10	directory	0755
translations		18/02/2022 17:50:10	directory	0755
var		18/02/2022 17:50:14	directory	0755
vendor		18/02/2022 17:50:26	directory	0755
env	2.05 KB	16/02/2022 21:29:04	text/plain	0644
.env.local.php	499 B	16/02/2022 21:28:46	text/x-php	0644
.gitignore	189 B	15/02/2022 14:24:14	text/plain	0644
composer.json	2.77 KB	15/02/2022 14:24:14	application/json	0644
composer.lock	257.63 KB	15/02/2022 14:24:14	application/json	0644
docker-compose.override.yml	247 B	15/02/2022 14:24:14	text/plain	0644
docker-compose.yml	717 B	15/02/2022 14:24:14	text/plain	0644
SoleneosSymfony2.zip	51.08 MB	16/02/2022 21:44:57	application/zip	0644
SoleneosSymfony3.zip	50.39 MB	20/02/2022 23:09:39	application/zip	0644
symfony.lock	11.85 KB	15/02/2022 14:24:14	application/json	0644

1.5 Entités:

Entity/User.php

```
/**
 * User
 *
 * @ORM\Table(name="user")
 * @ORM\Entity
 * @UniqueEntity(fields={"email"}, message="There is already an account with this email")
 */
class User implements UserInterface, PasswordAuthenticatedUserInterface
{
    /**
     * @ORM\Id
     * @ORM\GeneratedValue
     * @ORM\Column(type="integer")
     */
    private $id;

    /**
     * @ORM\Column(type="string", length=180, unique=true)
     */
    private $email;

    /**
     * @ORM\Column(type="json")
     */
    private $roles = [];

    /**
     * @var string The hashed password
     * @ORM\Column(type="string")
     */
    private $password;

    /**
     * @ORM\Column(type="string", length=255)
     */
    private $username;
```


Entity/Projet.php

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

    /**
     * @ORM\Column(type="string", length=255)
     */
    private $name;

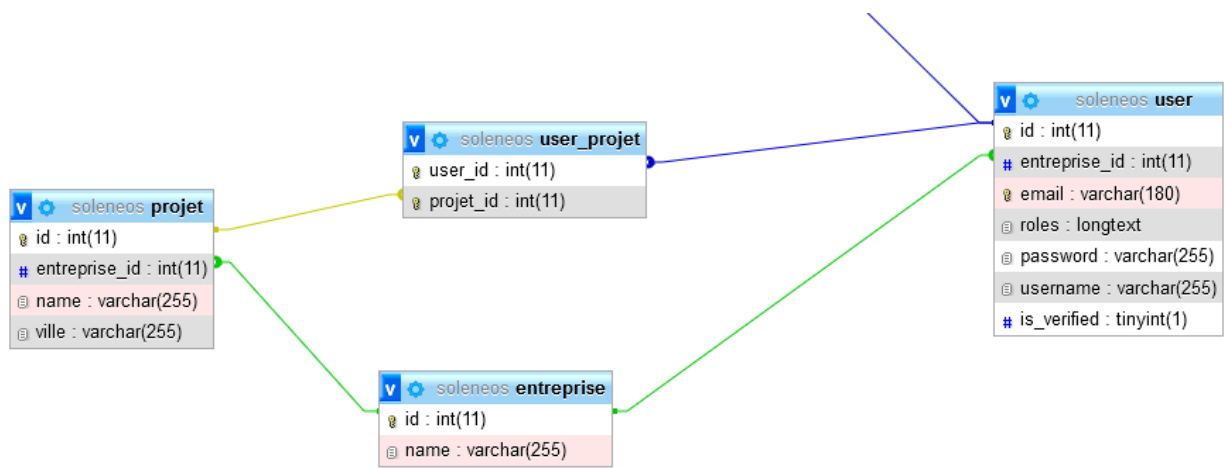
    /**
     * @ORM\Column(type="string", length=255)
     */
    private $ville;
```

Entity/Entreprise.php

```
/**
 * Entreprise
 *
 * @ORM\Table(name="entreprise")
 * @ORM\Entity
 */
class Entreprise
{
    /**
     * @var int
     *
     * @ORM\Column(name="id", type="integer", nullable=false)
     * @ORM\Id
     * @ORM\GeneratedValue(strategy="IDENTITY")
     */
    private $id;

    /**
     * @var string
     *
     * @ORM\Column(name="name", type="string", length=255, nullable=false)
     */
    private $name;
```

1.6 Base de données :



1.7 Dashboard :

Soleneos BACKEND

Allez vers la page de visualisation

Retour vers Soleneos.fr

Utilisateurs

Entreprises

Projets

Utilisateurs

<input type="checkbox"/>	Username	Email	Roles	Entreprise	Projets	
<input type="checkbox"/>	Pascal Chantrel	pascal@gmail.com	Admin, User	Null	marseille	...
<input type="checkbox"/>	administrateur	soleneos2017@gmail.com	User, Admin	Null		...

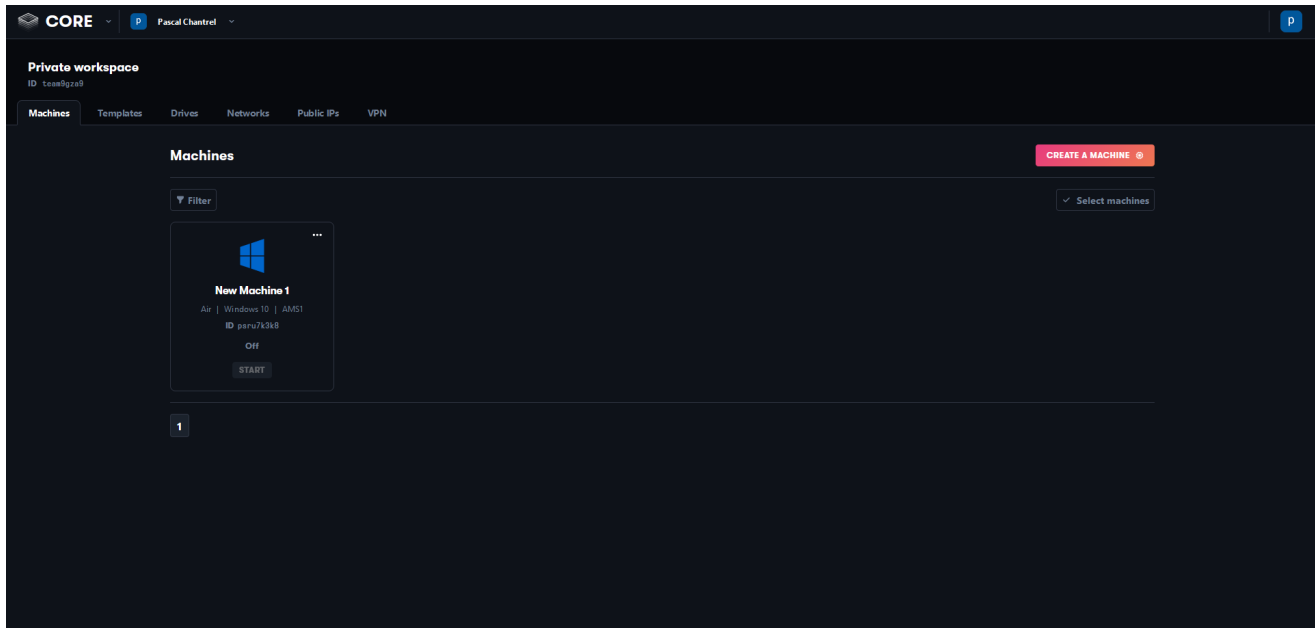
2 results

Previous

1

Next

1.8 Paperspace :



1.9 Launcher.config:

```
{
  "configuration": {
    "host": "localhost",
    "port": 9000,
    "endpoint": "paraview",
    "content": "C:/ParaView-5.10.0/share/paraview-5.10/web/visualizer/www",
    "proxy_file": "C:/xampp/proxy.txt",
    "sessionURL": "ws://74.82.29.5:80/proxy?sessionId=${id}&path=ws",
    "timeout": 30,
    "log_dir": "C:/ParaView-5.10.0/log/",
    "fields": []
  },
  "resources": [ {
    "host": "localhost",
    "port_range": [9001, 9999]
  } ],
  "properties": {
    "python_exec": "C:/ParaView-5.10.0/bin/pvpython.exe",
    "visualizer": "C:/ParaView-5.10.0/share/paraview-5.10/web/visualizer/server/pvw-visualizer.py"
  },
  "apps": {
    "visualizer": {
      "cmd": [
        | "${python_exec}", "--dr", "${visualizer}", "--port", "${port}", "--authKey", "${secret}", "--data", "${dataDir}", "--load-file", "${dataFile}"
        | ],
      "ready_line": "Starting factory"
    }
  }
}
```