|  |
| --- |
| Documentation pour le Projet Symfony "Le p'tit Jardinier" |
|  |
|  |

# Contexte

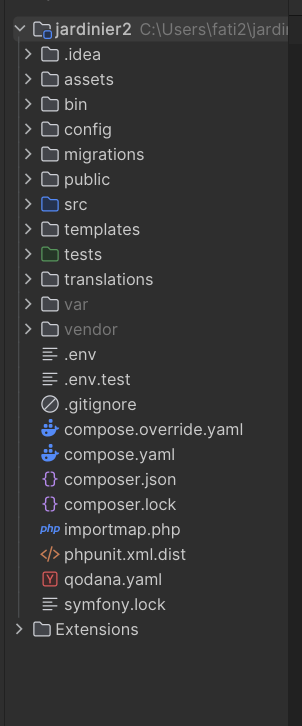
La société "Le p'tit Jardinier" souhaite développer une application web permettant à ses clients de demander des devis pour des services de taille de haies.

# Présentation du Projet

Le projet consiste à créer une application web pour "Le p'tit Jardinier" afin de permettre aux utilisateurs de demander des devis pour des services de taille de haies. Cette application sera développée en utilisant le framework Symfony.

# Prérequis Techniques

Avant de commencer, assurez-vous d'avoir les éléments suivants installés sur votre machine :

* PHP 8.1.x ou supérieur
* Composer
* Symfony CLI
* Serveur Web (WampServer ou similaire)
* Base de données (MySQL)

# Installation et Configuration

1. Mettre à jour votre environnement de développement :
   * Télécharger et installer la dernière version de WampServer : [WampServer](https://wampserver.aviatechno.net/)
   * Mettre à jour PHP et PhpMyAdmin si nécessaire.
2. Installer Composer :
   * Télécharger et installer Composer : Composer
   * Vérifier l'installation avec la commande :

composer --version

1. Installer Symfony CLI :
   * Télécharger et installer Symfony CLI : Symfony CLI
   * Vérifier l'installation avec la commande :

symfony check:requirements

# Création de l'Application

1. Créer un nouveau projet Symfony :
   * Exécuter la commande suivante pour créer un nouveau projet :

symfony new monProjet --webapp

1. Initialisation du projet :
   * Cette commande téléchargera et configurera automatiquement les bundles nécessaires dans le dossier monProjet.

# Démarrage du Serveur Web

1. Démarrer le serveur web intégré :
   * Utiliser l'une des commandes suivantes pour démarrer le serveur web :

symfony server:start

# ou

symfony serve

1. Accéder à l'application dans un navigateur :
   * Ouvrir un navigateur et accéder à l'adresse : <http://localhost:8000>

# Étapes de Réalisation

## Étape 1 : Gestion Contrôleur/Vue

### 1. Page d'accueil : création du contrôleur et de la vue

Générer un contrôleur

Créer un contrôleur, utilisez la commande suivante :

symfony console make:controller

La commande demandera le nom du contrôleur. Entrez Accueil (pas besoin d'ajouter "Controller" car il est ajouté automatiquement).

Résultat de la commande :

* Création du contrôleur AccueilController.php dans le dossier monProjet\src\Controller
* Création de la vue index.html.twig dans le dossier monProjet\templates\accueil

Contenu du contrôleur :

<?php

namespace App\Controller;

use Symfony\Bundle\FrameworkBundle\Controller\AbstractController;

use Symfony\Component\HttpFoundation\Response;

use Symfony\Component\Routing\Annotation\Route;

class AccueilController extends AbstractController

{

#[Route('/accueil', name: 'app\_accueil')]

public function index(): Response

{

return $this->render('accueil/index.html.twig', [

'controller\_name' => 'AccueilController',

]);

}

}

Test de la page d'accueil

Pour tester le contrôleur, accédez à [127.0.0.1:8000/accueil](http://127.0.0.1:8000/accueil). Vous devriez voir apparaître le message "Hello AccueilController".

Modification de la vue

Le fichier correspondant à cette vue est monProjet\templates\accueil\index.html.twig. Modifiez-le pour inclure un lien hypertexte :

html

Copier le code

<!DOCTYPE html>

<html>

<head>

<title>Accueil</title>

</head>

<body>

<h1>Hello {{ controller\_name }}</h1>

<a href="{{ path('app\_choix') }}">Obtenir un devis en ligne</a>

</body>

</html>



<div class="example-wrapper">  
  
 <h1><u>Le P'tit J@rdinier</u></h1>  
 <p>  
 Des professionnels à votre service  
 </p>  
  
 <p><a href="{{ path('app\_choix')}}">obtenir un devis en ligne pour une taille de haie</a></p>  
<img src="{{asset('/images/img.png') }}">  
  
</div>  
{% endblock %}

2. Page de choix du type d'utilisateur : Particulier ou entreprise

Création du contrôleur et de la vue

Créez un nouveau contrôleur ChoixController avec la commande :

symfony console make:controller Choix

Contenu du contrôleur :

<?php

namespace App\Controller;

use Symfony\Bundle\FrameworkBundle\Controller\AbstractController;

use Symfony\Component\HttpFoundation\Response;

use Symfony\Component\Routing\Annotation\Route;

class ChoixController extends AbstractController

{

#[Route('/choix', name: 'app\_choix')]

public function index(): Response

{

return $this->render('choix/index.html.twig');

}

}  
  
Une image contenant texte, capture d’écran, habits, personne

Description générée automatiquement

Modification de la vue d'accueil

Modifiez la vue d'accueil pour appeler le contrôleur ChoixController depuis le lien hypertexte :

<a href="{{ path('app\_choix') }}">Obtenir un devis en ligne</a>

3. Page de choix de la haie, saisie de la longueur et hauteur

Création du contrôleur et de la vue

Créez un nouveau contrôleur MesureController avec la commande :

symfony console make:controller Mesure

Contenu du contrôleur :

{% extends 'base.html.twig' %}  
  
{% block title %}Choix d'utlisateur{% endblock %}  
  
{% block body %}  
<style>  
 .example-wrapper { margin: 1em auto; max-width: 800px; width: 95%; font: 18px/1.5 sans-serif; }  
 .example-wrapper code { background: #F5F5F5; padding: 2px 6px; }  
 .example-wrapper fieldset{  
 color: #7484fd;  
 border: 1px solid #7484fd;  
 border-radius: 5px;  
 padding: 10px;  
 margin: 10px;  
 }  
 .example-wrapper legend{  
 color: #7484fd;  
 font-size: 20px;  
 font-weight: bold;  
 }  
 .example-wrapper input[type="radio"]{  
 margin: 10px;  
 }  
 .example-wrapper input[type="submit"]{  
 margin: 10px;  
 background-color: #7484fd;  
 color: white;  
 border: none;  
 border-radius: 5px;  
 padding: 10px;  
 }  
 .example-wrapper input[type="submit"]:hover{  
 color: #7484fd;  
 border: 1px solid #7484fd;  
 border-radius: 5px;  
 padding: 10px;  
 margin: 10px;  
 background-color: white;  
 color: #7484fd;  
 }  
  
 .example-wrapper select {  
 color: #7484fd;  
 border: 1px solid #7484fd;  
 border-radius: 5px;  
 padding: 10px;  
 margin: 10px;  
 }  
 .example-wrapper button{  
 margin: 10px;  
 padding: 10px;  
 border-radius: 5px;  
 background-color: #7484fd;  
 color: white;  
 border: none;  
 }  
 .example-wrapper button:hover{  
 color: #7484fd;  
 border: 1px solid #7484fd;  
 border-radius: 5px;  
 padding: 10px;  
 margin: 10px;  
 background-color: white;  
 color: #7484fd;  
 }  
  
  
</style>  
<div class="example-wrapper">  
{% if app.user %}  
<form action="{{ path('app\_mesure') }}" method="post">  
<fieldset>  
 <label>  
 <input type="radio" name="typeUtilisateur" value="particulier" >  
 Particulier  
 </label>  
 <label>  
 <input type="radio" name="typeUtilisateur" value="entreprise">  
 Entreprise  
 </label>  
 <button type="submit">Envoyer</button>  
 </fieldset>  
</form>  
{% else %}  
 <h1>Vous devez vous connecter pour accéder à cette page</h1>  
 <a href="{{ path('app\_login') }}">Se connecter</a>  
 <a href="{{ path('app\_register') }}">S'inscrire</a>  
{% endif %}  
  
  
  
</div>  
{% endblock %}

<?php  
  
namespace App\Controller;  
  
use Symfony\Bundle\FrameworkBundle\Controller\AbstractController;  
use Symfony\Component\HttpFoundation\Response;  
use Symfony\Component\Routing\Annotation\Route;  
use Symfony\Component\HttpFoundation\Request;  
use App\Entity\Devis;  
  
  
class ChoixController extends AbstractController  
{  
 #[Route('/choix', name: 'app\_choix')]  
 public function index(): Response  
 {  
 return $this->render('choix/index.html.twig', [  
 'controller\_name' => 'ChoixController',  
 ]);  
 }  
  
 #[Route('/choix/modifier/{id}', name: 'app\_choix\_modifier')]  
 public function choixModifier(Request $request, $id): Response  
 {  
 // Récupérer l'entité Devis correspondant à l'ID  
 $devis = $this->getDoctrine()->getRepository(Devis::class)->find($id);  
  
 // Vérifier si le devis existe  
 if (!$devis) {  
 throw $this->createNotFoundException('Aucun devis trouvé pour l\'ID '.$id);  
 }  
  
 // Maintenant vous pouvez utiliser $devis dans votre vue  
  
 return $this->render('choix/choixModifier.html.twig', [  
 'controller\_name' => 'ChoixController',  
 'devis' => $devis,  
 ]);  
 }  
}

Le ChoixController gère les opérations liées à l'entité Devis. Il permet l'affichage d'une page de choix et la modification d'un devis spécifique.

## Routes

### Route /choix

* **Nom** : app\_choix
* **Méthode HTTP** : GET
* **Description** : Affiche la page principale des choix.
* **Méthode du contrôleur** : index
* **Template** : choix/index.html.twig
* **Paramètres** :
  + Aucun

### Route /choix/modifier/{id}

* **Nom** : app\_choix\_modifier
* **Méthode HTTP** : GET
* **Description** : Affiche la page de modification pour un devis spécifique.
* **Méthode du contrôleur** : choixModifier
* **Template** : choix/choixModifier.html.twig
* **Paramètres** :
  + id : Identifiant du devis à modifier (type : entier)

## Méthodes

### index()

* **Retourne** : Response
* **Description** : Affiche la page principale des choix.
* **Logiciel** :
  + Rend le template choix/index.html.twig
  + Passe le nom du contrôleur comme variable au template

### choixModifier(Request $request, $id)

* **Paramètres** :
  + Request $request : L'objet requête HTTP
  + int $id : Identifiant du devis à modifier
* **Retourne** : Response
* **Description** : Récupère et affiche un devis pour modification.
* **Logiciel** :
  + Récupère l'entité Devis correspondant à l'ID fourni
  + Si le devis n'existe pas, lance une exception NotFoundHttpException
  + Rend le template choix/choixModifier.html.twig en passant l'objet Devis comme variable au template

## Templates

### choix/index.html.twig

Template utilisé pour afficher la page principale des choix. Ce template reçoit la variable controller\_name.

### choix/choixModifier.html.twig

Une image contenant texte, ligne, Police, capture d’écran

Description générée automatiquementTemplate utilisé pour afficher la page de modification d'un devis. Ce template reçoit l'objet devis pour afficher les informations du devis à modifier.

Le MesureController gère les opérations liées aux mesures de haies. Il permet de lister les haies, de sélectionner une haie spécifique et de soumettre des mesures (longueur et hauteur) associées.

## Routes

### Route /mesure

* **Nom** : app\_mesure
* **Méthode HTTP** : GET
* **Description** : Affiche la page de mesure des haies, permet de sélectionner une haie et de saisir des mesures.
* **Méthode du contrôleur** : index
* **Template** : mesure/index.html.twig
* **Paramètres** :
  + Aucun

## Méthodes

### index(Request $request, SessionInterface $session, HaieRepository $haieRepository)

* **Paramètres** :
  + Request $request : L'objet requête HTTP
  + SessionInterface $session : L'interface de session pour gérer les sessions utilisateur
  + HaieRepository $haieRepository : Le repository pour accéder aux entités Haie
* **Retourne** : Response
* **Description** : Affiche la page de mesure des haies. Récupère toutes les haies, le type d'utilisateur depuis la requête, et stocke le type d'utilisateur dans la session.
* **Logiciel** :
  + Récupère toutes les entités Haie depuis le repository
  + Récupère le type d'utilisateur depuis les données du formulaire
  + Stocke le type d'utilisateur dans la session
  + Rend le template mesure/index.html.twig en passant les haies et le type d'utilisateur comme variables au template
* class MesureController extends AbstractController  
  {  
   #[Route('/mesure', name: 'app\_mesure')]  
   public function index(Request $request, SessionInterface $session, HaieRepository $haieRepository): Response  
   {  
   //la liste des haies  
   $haies = $haieRepository->findAll();  
    
   // Récupérer le type d'utilisateur à partir des données du formulaire  
   $typeUtilisateur = $request->request->get('typeUtilisateur');  
    
   // Créer une variable de session pour stocker le type d'utilisateur  
   $session->set('typeUtilisateur', $typeUtilisateur);  
    
   return $this->render('mesure/index.html.twig', [  
   'controller\_name' => 'MesureController',  
   'haies' => $haies,  
   'typeUtilisateur' => $typeUtilisateur,  
    
   ]);  
   }  
  }

## Template

### mesure/index.html.twig

Ce template est utilisé pour afficher la page de mesure des haies. Il permet de sélectionner une haie, de saisir des mesures de longueur et de hauteur, et de soumettre ces données. Le template affiche également un message de connexion si l'utilisateur n'est pas connecté.

## Une image contenant texte, capture d’écran, logiciel, nombre Description générée automatiquement Description

Le DevisController gère les opérations liées aux devis pour les haies, y compris la création, la visualisation, l'édition et la suppression de devis.

## Routes

### Route /devis

* **Nom** : app\_devis
* **Méthode HTTP** : POST
* **Description** : Gère la création de devis et calcule le montant total en fonction des paramètres saisis par l'utilisateur.
* **Méthode du contrôleur** : index
* **Template** : devis/index.html.twig

### Route /devis/show

* **Nom** : app\_devis\_show
* **Méthode HTTP** : GET
* **Description** : Affiche tous les devis et les détails associés.
* **Méthode du contrôleur** : show
* **Template** : devis/show.html.twig

### Route /devis/delete/{id}

* **Nom** : app\_devis\_delete
* **Méthode HTTP** : GET
* **Description** : Supprime un devis et ses détails associés.
* **Méthode du contrôleur** : delete
* **Template** : Aucun (redirection vers app\_devis\_mesdevis)

### Route /devis/edit/{id}

* **Nom** : app\_devis\_edit
* **Méthode HTTP** : POST
* **Description** : Modifie un devis existant.
* **Méthode du contrôleur** : edit
* **Template** : devis/edit.html.twig

### Route /devis/mesdevis

* **Nom** : app\_devis\_mesdevis
* **Méthode HTTP** : GET
* **Description** : Affiche tous les devis de l'utilisateur connecté.
* **Méthode du contrôleur** : mesdevis
* **Template** : devis/mesdevis.html.twig

## Méthodes

### index(Request $request, EntityManagerInterface $entityManager, HaieRepository $haieRepository, SessionInterface $session): Response

* **Description** : Crée un devis en fonction des données saisies par l'utilisateur et calcule le montant total.
* **Paramètres** :
  + Request $request
  + EntityManagerInterface $entityManager
  + HaieRepository $haieRepository
  + SessionInterface $session
* **Retourne** : Response

### show(Request $request, EntityManagerInterface $entityManager, DevisRepository $devisRepository, TaillerRepository $taillerRepository): Response

* **Description** : Affiche tous les devis et les taillers associés.
* **Paramètres** :
  + Request $request
  + EntityManagerInterface $entityManager
  + DevisRepository $devisRepository
  + TaillerRepository $taillerRepository
* **Retourne** : Response

### delete(Request $request, EntityManagerInterface $entityManager, DevisRepository $devisRepository, TaillerRepository $taillerRepository, $id): Response

* **Description** : Supprime un devis et tous les taillers associés.
* **Paramètres** :
  + Request $request
  + EntityManagerInterface $entityManager
  + DevisRepository $devisRepository
  + TaillerRepository $taillerRepository
  + $id
* **Retourne** : Response

### edit(Request $request, EntityManagerInterface $entityManager, DevisRepository $devisRepository, TaillerRepository $taillerRepository, HaieRepository $haieRepository, $id): Response

* **Description** : Modifie un devis existant.
* **Paramètres** :
  + Request $request
  + EntityManagerInterface $entityManager
  + DevisRepository $devisRepository
  + TaillerRepository $taillerRepository
  + HaieRepository $haieRepository
  + $id
* **Retourne** : Response

### mesdevis(Request $request, EntityManagerInterface $entityManager, DevisRepository $devisRepository, TaillerRepository $taillerRepository): Response

* **Description** : Affiche tous les devis de l'utilisateur connecté.
* **Paramètres** :
  + Request $request
  + EntityManagerInterface $entityManager
  + DevisRepository $devisRepository
  + TaillerRepository $taillerRepository
* **Retourne** : Response

Une image contenant texte, capture d’écran, nombre, Parallèle

Description générée automatiquementUne image contenant texte, ligne, Police, diagramme

Description générée automatiquementUne image contenant texte, capture d’écran, Police

Description générée automatiquement

Une image contenant texte, capture d’écran, nombre, Parallèle

Description générée automatiquement



contrôleur CreerHaieController gère la création de nouvelles haies. Voici une documentation technique pour ce contrôleur :

### index(Request $request, EntityManagerInterface $entityManager, HaieRepository $haieRepository): Response

* **Description** : Affiche le formulaire de création de haie et gère la soumission du formulaire pour enregistrer une nouvelle haie dans la base de données.
* **Paramètres** :
  + $request : L'objet Request qui contient les données de la requête HTTP.
  + $entityManager : L'interface EntityManagerInterface pour interagir avec l'ORM Doctrine.
  + $haieRepository : Le repository de l'entité Haie pour effectuer des opérations de recherche.
* **Retourne** : Response contenant la vue Twig rendue avec le formulaire de création de haie.
* class CreerHaieController extends AbstractController  
  {  
    
   #[Route('/creerhaie', name: 'app\_creerhaie')]  
   public function index(Request $request, EntityManagerInterface $entityManager, HaieRepository $haieRepository): Response  
   {  
    
    
   $haie = new Haie();  
   $form = $this->createForm(HaieType::class, $haie);  
   $form->handleRequest($request);  
    
   if ($form->isSubmitted() && $form->isValid()) {  
   $entityManager->persist($haie);  
   $entityManager->flush();  
    
   return $this->redirectToRoute('app\_creerhaie');  
   }  
    
   $haies = $haieRepository->findAll();  
    
   return $this->render('creer\_haie/index.html.twig', [  
   'form' => $form->createView(),  
   ]);  
   }  
  }

### Templates

#### creer\_haie/index.html.twig

<div class="example-wrapper">  
 <h1>Créer une nouvelle haie</h1>  
  
 {% if is\_granted('ROLE\_ADMIN') %}  
 {{ form\_start(form) }}  
 {{ form\_widget(form) }}  
 <button type="submit" class="btn btn-primary">Ajouter</button>  
 {{ form\_end(form) }}  
 {% else %}  
 <div class="alert">  
 Cette action est réservée aux administrateurs.  
 </div>  
 {% endif %}  
</div>  
{% endblock %}

### Une image contenant texte, capture d’écran, Police, nombre Description générée automatiquement index(Request $request, EntityManagerInterface $entityManager, HaieRepository $haieRepository): Response

* **Description** : Affiche la liste des haies existantes et le formulaire pour en créer de nouvelles.
* **Paramètres** :
  + $request : L'objet Request qui contient les données de la requête HTTP.
  + $entityManager : L'interface EntityManagerInterface pour interagir avec l'ORM Doctrine.
  + $haieRepository : Le repository de l'entité Haie pour effectuer des opérations de recherche.
* **Retourne** : Response contenant la vue Twig rendue avec le tableau des haies et le formulaire de création de haie.
* class ListHaieController extends AbstractController  
  {  
   #[Route('/list/haie', name: 'app\_list\_haie')]  
   public function index(Request $request, EntityManagerInterface $entityManager,HaieRepository $haieRepository): Response  
   {  
   $haies = new Haie();  
   $form = $this->createForm(HaieType::class, $haies);  
   $form->handleRequest($request);  
    
   if ($form->isSubmitted() && $form->isValid()) {  
   $entityManager->persist($haies);  
   $entityManager->flush();  
   return $this->redirectToRoute('app\_list\_haie');  
   }  
    
   //affichage du tableau des haies  
   $haies = $haieRepository->findBy([], ['code' => 'DESC']);  
    
   return $this->render('list\_haie/index.html.twig', [  
   'controller\_name' => 'ListHaieController',  
   'form' => $form->createView(),  
   'haies' => $haies,  
   ]);  
   }  
  }

### Template list\_haie/index.html.twig

<div class="example-wrapper">  
 <table>  
 <tr>  
 <th>Code</th>  
 <th>Nom</th>  
 <th>Prix</th>  
 <th></th>  
 </tr>  
 {% for haie in haies %}  
 <tr>  
 <td>{{ haie.code }}</td>  
 <td>{{ haie.nom }}</td>  
 <td>{{ haie.prix }}</td>  
 <td>  
  
 <a href="{{ path('modifier\_haie', { 'code': haie.getCode() }) }}">  
 Consultation/Modification  
 </a>  
 </td>  
 </tr>  
  
 {% endfor %}  
 </table>  
</div>  
{% else %}  
 <h1>Vous devez vous connecter pour accéder à cette page</h1>  
 <a href="{{ path('app\_login') }}">Connexion</a>  
 <a href="{{ path('app\_register') }}">Inscription</a>  
{% endif %}  
{% endblock %}

Une image contenant texte, capture d’écran, nombre, Police

Description générée automatiquement

### index(Request $request, EntityManagerInterface $entityManager, HaieRepository $haieRepository, $code): Response

* **Description** : Cette méthode gère la modification d'une haie existante identifiée par son code.
* **Paramètres** :
  + $request : L'objet Request qui contient les données de la requête HTTP.
  + $entityManager : L'interface EntityManagerInterface pour interagir avec l'ORM Doctrine et gérer les entités.
  + $haieRepository : Le repository de l'entité Haie pour effectuer des opérations de recherche.
  + $code : Le code de la haie à modifier.
* **Retourne** : Response contenant la vue Twig rendue avec le formulaire de modification de la haie.
* Une image contenant texte, capture d’écran, Police, nombre

  Description générée automatiquement<?php  
    
  namespace App\Controller;  
    
  use Symfony\Bundle\FrameworkBundle\Controller\AbstractController;  
  use Symfony\Component\HttpFoundation\Request;  
  use Symfony\Component\HttpFoundation\Response;  
  use Symfony\Component\Routing\Annotation\Route;  
  use App\Form\HaieType;  
  use Doctrine\ORM\EntityManagerInterface;  
  use App\Repository\HaieRepository;  
    
  class ModifierHaieController extends AbstractController  
  {  
   #[Route('/modifier\_haie/{code}', name: 'modifier\_haie')]  
   public function index(Request $request, EntityManagerInterface $entityManager, HaieRepository $haieRepository, $code): Response  
   {  
   //Récupérer la haie à partir du code  
   $haie = $haieRepository->findOneBy(['code' => $code]);  
    
   if (!$haie) {  
   throw $this->createNotFoundException('La haie avec le code ' . $code . ' n\'existe pas.');  
   }  
    
   // Créer le formulaire de modification  
   $form = $this->createForm(HaieType::class, $haie);  
   $form->handleRequest($request);  
    
   if ($form->isSubmitted() && $form->isValid()) {  
   // Enregistrer la modification dans la base de données  
   $entityManager->flush();  
    
   // Rediriger vers la liste des haies après la modification  
   return $this->redirectToRoute('app\_list\_haie');  
   }  
    
   return $this->render('modifier\_haie/index.html.twig', [  
   'form' => $form->createView(),  
   ]);  
   }  
  }

### Template modifier\_haie/index.html.twig

<div class="example-wrapper">  
 <h2>Modifier haie</h2>  
 {{form(form) }}  
  
</div>  
{% endblock %}

Le contrôleur RegistrationController gère à la fois l'inscription de nouveaux utilisateurs et la modification des informations des utilisateurs existants. Voici une documentation pour ce contrôleur :

### Méthode register(Request $request, UserPasswordHasherInterface $userPasswordHasher, UserAuthenticatorInterface $userAuthenticator, UserAuthenticator $authenticator, EntityManagerInterface $entityManager): Response

* **Description** : Cette méthode gère le processus d'inscription d'un nouvel utilisateur.
* **Paramètres** :
  + $request : L'objet Request qui contient les données de la requête HTTP.
  + $userPasswordHasher : L'interface UserPasswordHasherInterface pour hasher le mot de passe de l'utilisateur.
  + $userAuthenticator : L'interface UserAuthenticatorInterface pour authentifier l'utilisateur après son inscription.
  + $authenticator : Une instance de UserAuthenticator pour authentifier l'utilisateur.
  + $entityManager : L'interface EntityManagerInterface pour interagir avec l'ORM Doctrine et gérer les entités.
* **Retourne** : Response contenant la redirection vers une page appropriée après l'inscription de l'utilisateur.

### Méthode edit(Request $request, EntityManagerInterface $entityManager, User $user): Response

* **Description** : Cette méthode gère la modification des informations d'un utilisateur existant.
* **Paramètres** :
  + $request : L'objet Request qui contient les données de la requête HTTP.
  + $entityManager : L'interface EntityManagerInterface pour interagir avec l'ORM Doctrine et gérer les entités.
  + $user : L'utilisateur à modifier.
* **Retourne** : Response contenant la vue Twig rendue avec le formulaire de modification des informations de l'utilisateur.
* <?php  
    
  namespace App\Controller;  
    
  use App\Entity\User;  
  use App\Form\RegistrationFormType;  
  use App\Security\UserAuthenticator;  
  use Doctrine\ORM\EntityManagerInterface;  
  use Symfony\Bundle\FrameworkBundle\Controller\AbstractController;  
  use Symfony\Component\HttpFoundation\Request;  
  use Symfony\Component\HttpFoundation\Response;  
  use Symfony\Component\PasswordHasher\Hasher\UserPasswordHasherInterface;  
  use Symfony\Component\Routing\Annotation\Route;  
  use Symfony\Component\Security\Http\Authentication\UserAuthenticatorInterface;  
  use Symfony\Contracts\Translation\TranslatorInterface;  
    
  class RegistrationController extends AbstractController  
  {  
   #[Route('/register', name: 'app\_register')]  
   public function register(Request $request, UserPasswordHasherInterface $userPasswordHasher, UserAuthenticatorInterface $userAuthenticator, UserAuthenticator $authenticator, EntityManagerInterface $entityManager): Response  
   {  
   $user = new User();  
   $form = $this->createForm(RegistrationFormType::class, $user);  
   $form->handleRequest($request);  
    
   if ($form->isSubmitted() && $form->isValid()) {  
   // encode the plain password  
   $user->setPassword(  
   $userPasswordHasher->hashPassword(  
   $user,  
   $form->get('plainPassword')->getData()  
   )  
   );  
    
   $entityManager->persist($user);  
   $entityManager->flush();  
   // do anything else you need here, like send an email  
    
   return $userAuthenticator->authenticateUser(  
   $user,  
   $authenticator,  
   $request  
   );  
   }  
    
   return $this->render('registration/register.html.twig', [  
   'registrationForm' => $form->createView(),  
   $this->redirectToRoute('app\_accuiel'  
    
   )  
   ]);  
   }  
   #[Route('/client/{id}/edit', name: 'client\_edit')]  
   public function edit(Request $request, EntityManagerInterface $entityManager, User $user): Response  
   {  
   $form = $this->createForm(RegistrationFormType::class, $user);  
   $form->handleRequest($request);  
    
   if ($form->isSubmitted() && $form->isValid()) {  
   // Mettre à jour les informations de l'utilisateur en base de données  
   $entityManager->flush();  
    
   // Rediriger vers une page de confirmation ou une autre page  
   return $this->redirectToRoute('confirmation\_page');  
   }  
    
   return $this->render('client/edit.html.twig', [  
   'form' => $form->createView(),  
   ]);  
   }  
  }

### Template registration/register.html.twig

* **Description** : Ce template affiche le formulaire d'inscription pour les nouveaux utilisateurs.
* {% block body %}  
   <div class="container">  
   <h1>Register</h1>  
    
   {{ form\_errors(registrationForm) }}  
    
   {{ form\_start(registrationForm) }}  
   <div class="cote-a-cote">  
   {{ form\_row(registrationForm.nom) }}  
   {{ form\_row(registrationForm.prenom) }}  
   </div>  
   {{ form\_row(registrationForm.adresse) }}  
   <div class="cote-a-cote">  
    
   {{ form\_row(registrationForm.ville)}}  
   {{ form\_row(registrationForm.cp) }}  
   </div>  
   <div class="cote-a-cote">  
   {{ form\_row(registrationForm.email) }}  
   {{ form\_row(registrationForm.plainPassword, {  
   label: 'Password'  
   }) }}  
   </div>  
   {{ form\_row(registrationForm.agreeTerms) }}  
    
   <button type="submit" class="btn">Register</button>  
   {{ form\_end(registrationForm) }}  
   <div>  
  {% endblock %}

Une image contenant texte, capture d’écran, nombre, Police

Description générée automatiquement

Le contrôleur SecurityController gère l'authentification des utilisateurs avec les méthodes login et logout. Voici une explication pour ce contrôleur :

### Méthode login(AuthenticationUtils $authenticationUtils): Response

* **Description** : Cette méthode gère l'affichage du formulaire de connexion et la gestion des erreurs d'authentification.
* **Paramètres** :
  + $authenticationUtils : Un objet AuthenticationUtils qui fournit des utilitaires pour récupérer les erreurs d'authentification et le dernier nom d'utilisateur utilisé.
* **Retourne** : Une Response avec le rendu du formulaire de connexion.

### Méthode logout(): RedirectResponse

* **Description** : Cette méthode gère la déconnexion de l'utilisateur et le redirige vers la page de connexion.
* **Retourne** : Une RedirectResponse vers la page de connexion.
* class SecurityController extends AbstractController  
  {  
   #[Route(path: '/login', name: 'app\_login')]  
   public function login(AuthenticationUtils $authenticationUtils): Response  
   {  
   // if ($this->getUser()) {  
   // return $this->redirectToRoute('target\_path');  
   // }  
    
   // get the login error if there is one  
   $error = $authenticationUtils->getLastAuthenticationError();  
   // last username entered by the user  
   $lastUsername = $authenticationUtils->getLastUsername();  
    
   return $this->render('security/login.html.twig', ['last\_username' => $lastUsername, 'error' => $error]);  
    
   }  
    
   #[Route(path: '/logout', name: 'app\_logout')]  
   public function logout(): \Symfony\Component\HttpFoundation\RedirectResponse  
   {  
   return $this->redirectToRoute('app\_login');  
   }  
    
    
  }

### Template security/login.html.twig

* **Description** : Ce template affiche le formulaire de connexion.
* {% block body %}  
   <div class="container">  
  <form method="post">  
   {% if error %}  
   <div class="alert alert-danger">{{ error.messageKey|trans(error.messageData, 'security') }}</div>  
   {% endif %}  
    
   {% if app.user %}  
   <div class="mb-3">  
   You are logged in as {{ app.user.userIdentifier }}, <a href="{{ path('app\_logout') }}">Logout</a>  
   </div>  
   {% endif %}  
    
   <h1 class="h3 mb-3 font-weight-normal">Connectez-vous</h1>  
   <label for="inputEmail">Email</label>  
   <input type="email" value="{{ last\_username }}" name="email" id="inputEmail" class="form-control" autocomplete="email" required autofocus>  
   <label for="inputPassword">Mot de passe</label>  
   <input type="password" name="password" id="inputPassword" class="form-control" autocomplete="current-password" required>  
    
   <input type="hidden" name="\_csrf\_token"  
   value="{{ csrf\_token('authenticate') }}"  
   >  
    
   <button class="btn btn-lg btn-primary" type="submit">  
   Connexion  
   </button>  
    
  </form>  
   </div>  
  {% endblock %}

Une image contenant texte, capture d’écran, Police, nombre

Description générée automatiquement  
  
Le ProfileController gère les actions liées au profil de l'utilisateur, notamment l'affichage du profil, la modification des informations de l'utilisateur et la modification du mot de passe. Voici une explication de ce contrôleur :

### Méthode index()

* **Route** : /profile
* **Description** : Cette méthode affiche le profil de l'utilisateur.
* **Retourne** : Une Response avec le rendu du template du profil (profile/index.html.twig).

### Méthode edit(Request $request, EntityManagerInterface $entityManager, User $user)

* **Route** : /profile/{id}/edit
* **Description** : Cette méthode gère la modification des informations de l'utilisateur.
* **Paramètres** :
  + $request : La requête HTTP.
  + $entityManager : L'interface pour interagir avec l'entité User dans la base de données.
  + $user : L'utilisateur à modifier.
* **Retourne** : Une Response avec le rendu du formulaire de modification des informations de l'utilisateur (profile/edit.html.twig).

### Méthode editPassword(Request $request, User $user, UserPasswordHasherInterface $passwordHasher, EntityManagerInterface $entityManager)

* **Route** : /profile/edit\_password/{id}
* **Description** : Cette méthode gère la modification du mot de passe de l'utilisateur.
* **Paramètres** :
  + $request : La requête HTTP.
  + $user : L'utilisateur dont le mot de passe doit être modifié.
  + $passwordHasher : L'interface pour hasher le nouveau mot de passe de l'utilisateur.
  + $entityManager : L'interface pour interagir avec l'entité User dans la base de données.
* **Retourne** : Une Response avec le rendu du formulaire de modification du mot de passe (profile/edit\_password.html.twig).

class ProfileController extends AbstractController  
{  
 #[Route('/profile', name: 'app\_profile')]  
 public function index(): Response  
 {  
 $user = $this->getUser();  
  
 return $this->render('profile/index.html.twig', [  
 'user' => $user,  
 ]);  
 }  
  
 #[Route('/profile/{id}/edit', name: 'edit\_user')]  
 public function edit(Request $request, EntityManagerInterface $entityManager, User $user): Response  
 {  
 $form = $this->createForm(UserEditType::class, $user);  
 $form->handleRequest($request);  
  
 if ($form->isSubmitted() && $form->isValid()) {  
 // If form is submitted and valid, update user information  
 $entityManager->flush(); // Save changes to the database  
  
 $this->addFlash('success', 'Your profile has been updated successfully.');  
  
 return $this->redirectToRoute('app\_profile');  
 }  
  
 return $this->render('profile/edit.html.twig', [  
 'form' => $form->createView(),  
 ]);  
 }  
  
 #[Route('/profile/edit\_password/{id}', name: 'edit\_password')]  
 public function editPassword(Request $request, User $user, UserPasswordHasherInterface $passwordHasher, EntityManagerInterface $entityManager): Response  
 {  
 $form = $this->createForm(ChangePasswordType::class);  
 $form->handleRequest($request);  
  
 if ($form->isSubmitted() && $form->isValid()) {  
 $data = $form->getData();  
 $newPassword = $passwordHasher->hashPassword($user, $data['newPassword']);  
 $user->setPassword($newPassword);  
 $entityManager->persist($user);  
 $entityManager->flush();  
  
 $this->addFlash('success', 'Password updated successfully.');  
  
 return $this->redirectToRoute('profile'); // Adjust the route as necessary  
 }  
  
 return $this->render('profile/edit\_password.html.twig', [  
 'form' => $form->createView(),  
 ]);  
 }  
}

### Templates Twig

* profile/index.html.twig : Affiche les détails du profil de l'utilisateur.

<div class="container">  
 <h2>Profil</h2>  
 <form method="post">  
 <label for="nom">Nom d'utilisateur : </label>  
 <label>{{ user.nom }}</label>  
 <br>  
 <label for="prenom">Prénom : </label>  
 <label>{{ user.prenom }}</label>  
  
 <br>  
 <label for="role">Rôle : </label>  
 <label>{{ user.roles|join(', ') }}</label>  
 <br>  
 <label for="email">Email : </label>  
 <label>{{ user.email }}</label>  
 <br>  
 <label for="adresse">Adresse : </label>  
 <label>{{ user.adresse }}</label>  
 <br>  
 <label for="cp">Code postal : </label>  
 <label>{{ user.cp }}</label>  
 <br>  
 <label for="ville">Ville : </label>  
 <label>{{ user.ville }}</label>  
  
 </form>  
 <button><a href="{{ path('edit\_user', {'id': user.id}) }}">Modifier</a></button>  
 <button><a href="{{ path('edit\_password',{'id':user.id}) }}">Changer le mot de passe</a></button>  
  
</div>

Une image contenant texte, capture d’écran, Police

Description générée automatiquement  
profile/edit.html.twig : Affiche le formulaire de modification des informations de l'utilisateur.

{% block body %}  
 <div class="container">  
 <h2>Change Password</h2>  
 {{ form\_start(form) }}  
 {{ form\_widget(form) }}  
 {{ form\_end(form) }}  
 </div>  
{% endblock %}

Une image contenant texte, capture d’écran, affichage, nombre

Description générée automatiquement

profile/edit\_password.html.twig : Affiche le formulaire de modification du mot de passe de l'utilisateur.

{% block body %}  
 <h1>Modifier l'utilisateur</h1>  
<div class="container">  
{{ form\_start(form) }}  
<div class="cote-a-cote">  
 {{ form\_row(form.email) }}  
 </div>  
<div class="cote-a-cote">  
 {{ form\_row(form.roles) }}  
</div>  
<div class="cote-a-cote">  
 {{ form\_row(form.password) }}  
 </div>  
 <div class="cote-a-cote">  
 {{ form\_row(form.nom) }}  
 {{ form\_row(form.prenom) }}  
 </div>  
 <div class="cote-a-cote">  
 {{ form\_row(form.adresse) }}  
 </div>  
 <div class="cote-a-cote">  
 {{ form\_row(form.ville) }}  
 {{ form\_row(form.cp) }}  
 </div>  
  
 <button type="submit" class="btn btn-primary">Save Changes</button>  
 {{ form\_end(form) }}  
 {% endblock %}

Une image contenant texte, capture d’écran, Police, ligne

Description générée automatiquement