

Le P'tit Jardinier – Documentation Technique (Symfony)

1. Présentation du Projet

Le P'tit Jardinier est une application web développée en PHP avec le framework Symfony. Ce projet vise à faciliter la gestion des devis pour une entreprise de jardinage.

Fonctionnalités principales :

- Gestion des haies et de leurs catégories
 - Gestion des clients
 - Création de devis (calcul automatique du tarif)
 - Espace membre avec authentification
 - Interface basée sur Twig
-

2. Environnement de Développement

- Symfony CLI
- PHP 8.x
- MAMP (serveur local Apache/MySQL)
- PHPMyAdmin (gestion de la base)
- Visual Studio Code (éditeur de code)
- SGBD : MySQL

Architecture utilisée :

- MVC (Modèle - Vue - Contrôleur)
-

3. Configuration Symfony

Base de données

Dans le fichier .env :

```
DATABASE_URL="mysql://user:password@127.0.0.1:3306/le_ptit_jardinier"
```

Doctrine est utilisé comme ORM (Object Relational Mapper).

4. Routage

Les routes sont définies par annotations :

```
[Route('/haie', name: 'haie')]
```

Elles permettent d'associer une URL à une méthode de contrôleur.

5. Contrôleurs et Vues

Contrôleur : Exemple de création d'une haie

```
#[Route('/new', name: 'app_haie_new', methods: ['GET', 'POST'])]
public function new(Request $request, EntityManagerInterface $entityManager):
    Response
    {
        // verifie si un utilisateur est connecté
        if (!$this->getUser()) {
            return $this->redirectToRoute('app_login');
        }

        $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_haie_index', [],
Response::HTTP_SEE_OTHER);
        }
        return $this->render('haie/new.html.twig', [
            'haie' => $haie,
            'form' => $form,
```

```
    });  
}
```

6. Vue (Twig)

```
{% extends 'base.html.twig' %}  
  
{% block body %}  
    <h1>Ajouter une haie</h1>  
    {{ form(form) }}  
{% endblock %}
```

7. CRUD avec Doctrine

Création

Utilisation de persist() et flush() :

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

Lecture

Utilisation du Repository :

```
return $this->render('haie/index.html.twig', [  
    'haies' => $haieRepository->findAll(),  
]);
```

Modification

Modifier l'objet et appeler flush() :

```
$form = $this->createForm(HaieType::class, $haie);  
$form->handleRequest($request);  
  
if ($form->isSubmitted() && $form->isValid()) {
```

```
$entityManager->flush();  
}
```

Suppression

```
$entityManager->remove($haie);  
$entityManager->flush();
```

8. Entités et Relations

Exemple de relation Haie - Catégorie :

```
#[ORM\ManyToOne(inversedBy: 'haies')]  
#[ORM\JoinColumn(nullable: false)]  
private ?Categorie $categorie = null;
```

9. Formulaires Symfony

Symfony permet la génération automatique de formulaires à partir des entités.

Exemple :

```
$builder  
    ->add('nom')  
    ->add('prix')  
    ->add('categorie', EntityType::class, [  
        'class' => Categorie::class,  
        'choice_label' => 'libelle',  
    ])  
    ;
```

Types de champs utilisés au cours du projet :

- TextType
- NumberType
- DateType
- SubmitType ...

10. Authentification & Espace Membre

Création d'un utilisateur :

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

11. Gestion des Erreurs

Validation des données avec Symfony Validator :

```
use Symfony\Component\Validator\Constraints\Length;
#[Assert\Length(
    min: 8,
    minMessage: "Le code doit contenir au moins {{ limit }} caractères"
)]
```

Affichage des erreurs dans la vue Twig :

```
{{ form_errors(form.codePostal) }}
```

12. Génération des Migrations

Après modification des entités, générer et exécuter les migrations :

```
php bin/console make:migration
php bin/console doctrine:migrations:migrate
```

13. Génération de Devis

- Sélection des haies à inclure dans le devis
- Saisie des dimensions

- Calcul dynamique du prix total