Le Petit Jardinier – Document Technique

Framework : Symfony

---

1. Vue d'Ensemble du Projet

Le Jardin Numérique est une application web construite à l'aide du framework **Symfony** en PHP.
Son objectif est de fournir une interface de gestion simplifiée pour les **devis d'une société de jardinage**.
L'application prend en charge les opérations courantes telles que l'administration des haies, des clients, des catégories, et des devis automatisés.

✨ Fonctions Clés :

- Gestion des **haies** et de leurs **catégories**.
- Enregistrement et suivi des **clients**.
- **Création de devis** avec calcul automatique des tarifs selon les dimensions.
- Authentification des membres via un **espace personnel sécurisé**.
- Interface dynamique développée avec **Twig**.

---

2. ⚙ Environnement Technique

- **Langage :** PHP 8.x
- **Framework :** Symfony (CLI)
- **SGBD :** MySQL
- **Serveur local :** WAMP (Apache/MySQL)
- **Éditeur :** Visual Studio Code
- **Administration BD :** phpMyAdmin
- **Architecture :** MVC (Modèle / Vue / Contrôleur)
- **ORM :** Doctrine

---

3. 🔧 Configuration de la Base de Données

Connexion configurée dans le fichier .env :

DATABASE_URL="mysql://user:password@127.0.0.1:3306/le_jardin_numerique"

Doctrine gère les entités via des annotations, assurant le mapping entre objets PHP et tables SQL.

---

4. 🌐 Routage

Le routage est défini directement à l'aide d'**annotations PHP** dans les contrôleurs. Cela permet de lier une URL à une méthode spécifique.

Exemple :

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

#[Route('/nouvelle', name: 'haie_nouvelle', methods: ['GET', 'POST'])]

---

5. 🎮 Contrôleurs

Extrait de méthode pour la création d'une haie :

```
public function ajouter(Request $request, EntityManagerInterface $em): Response
{
    if (!$this->getUser()) {
        return $this->redirectToRoute('app_login');
    }

    $haie = new Haie();
    $formulaire = $this->createForm(HaieType::class, $haie);
    $formulaire->handleRequest($request);

    if ($formulaire->isSubmitted() && $formulaire->isValid()) {
        $em->persist($haie);
        $em->flush();
        return $this->redirectToRoute('haie_liste');
    }

    return $this->render('haie/ajouter.html.twig', [
```

```
        'form' => $formulaire,
    ]);
}
```

---

6. 🎨 Gabarits Twig

Exemple d'affichage de formulaire :

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

{% block body %}
    <h1>Créer une Haie</h1>
    {{ form(form) }}
{% endblock %}
```

---

7. 🍣 Opérations CRUD avec Doctrine

- **Créer** : persist() + flush()
- **Lire** : Requête via le Repository
- **Mettre à jour** : Modifier puis flush()
- **Supprimer** :

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

---

8. 🔗 Modélisation des Entités

Exemple de relation entre Haie et Categorie :

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

---

9. 📝 Formulaires Symfony

Symfony permet la **génération automatique de formulaires** basés sur les entités. Champs utilisés :

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

Types courants :

- TextType
- NumberType
- DateType
- SubmitType

---

10. 🔐 Authentification & Espace Membre

Création d'un utilisateur via console Symfony :

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

---

11. ⚠️ Validation & Gestion des Erreurs

Symfony utilise les **annotations Validator** pour valider les champs :

```
php
CopierModifier
#[Assert\Length(
    min: 8,
    minMessage: "Le mot de passe doit contenir au moins {{ limit }} caractères."
)]
```

Affichage d'erreurs dans Twig :

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

---

12. 🛠 Migration des Schémas

Après modification d’une entité, exécuter :

php bin/console make:migration

php bin/console doctrine:migrations:migrate

---

13. Génération Dynamique de Devis

- Sélectionner plusieurs haies à intégrer.
- Saisir les dimensions souhaitées.
- Le prix total est **calculé automatiquement** selon les règles métier définies.

✅ Conclusion

Le petit Jardinier est une solution pratique pour gérer efficacement les clients, les devis et les haies d’un prestataire paysagiste.
Il s’agit d’un projet complet permettant de découvrir le fonctionnement du framework **Symfony**, les entités Doctrine, la gestion de la base et l’authentification utilisateur.