

LeP'titJardinier - Roche Sébastien



Présentation et objectifs

Cet atelier professionnel a pour but de réaliser une application web, à l'aide du Framework PHP Symfony. Cette application reprend le contexte d'un site de jardinerie, qui doit permettre de pouvoir créer des devis avec un ou plusieurs taillages de haies dans le même devis. Voici la liste des contraintes à respecter :

- Gestion des clients : Recherche/Ajout/Modification/Suppression des clients
- Gestion des devis : Calcul du devis en fonction du type de client (particulier ou entreprise) pour obtenir la remise et des différents types de haie du devis (hauteur et longueur)

Environnement de travail & Symfony

Lors de cette atelier professionnel, j'ai utilisé une base de donnée MySQL générée par l'ORM Doctrine. Comme indiqué précédemment, cet AP a été consacré à l'utilisation du framework PHP Symfony. J'utilise l'IDE Visual Studio code pour développer, car il est à mon goût, le plus complet des IDE et il a également la possibilité de rajouter des extensions créées d'une grande communauté.



Concernant Symfony, voici de plus amples précisions :

Symfony est un ensemble de composants PHP ainsi qu'un framework MVC libre écrit en PHP. Il fournit des fonctionnalités modulables et adaptables qui permettent de faciliter et d'accélérer le développement d'un site web.

Ce framework permet de gérer facilement un système de routage, le passage de données des contrôleurs à la vue grâce au langage Twig.

De plus nous avons la possibilité d'utiliser l'outil de Mapping Doctrine. Cet ORM orienté PHP, permet d'interagir avec la base de donnée (sélectionner des valeurs/créer des tables) sans requête SQL, ni aucune action sur Phpmyadmin.

Découvrir les fonctionnalités du site

- [Capture d'écrans de l'application](#)

Base de donnée

- [Base de donnée](#)

[BONUS] Système de connexion & Clients

- [Connexion, Inscription & Permissions](#)
- [Gestion des clients avec CRUD](#)

Gestion des devis

- [Type de client et Devis](#)
- [Relation entre devis & haie](#)
- [Choix des dimensions et de la haie - Devis](#)
- [Validation du devis et liste des devis](#)

Gestion des haies

- [Gestion des haies avec CRUD](#)

Conclusion

Cet atelier professionnel m'a permis de découvrir un nouveau framework et d'en maîtriser les points forts, notamment grâce aux multiples options et commandes disponibles. L'ensemble de ses fonctionnalités sont des acteurs clés dans le développement back-end en PHP. J'ai beaucoup aimé cet atelier sur Symfony

From:

<http://ppe.boonum.fr/> - **AP.SIO**

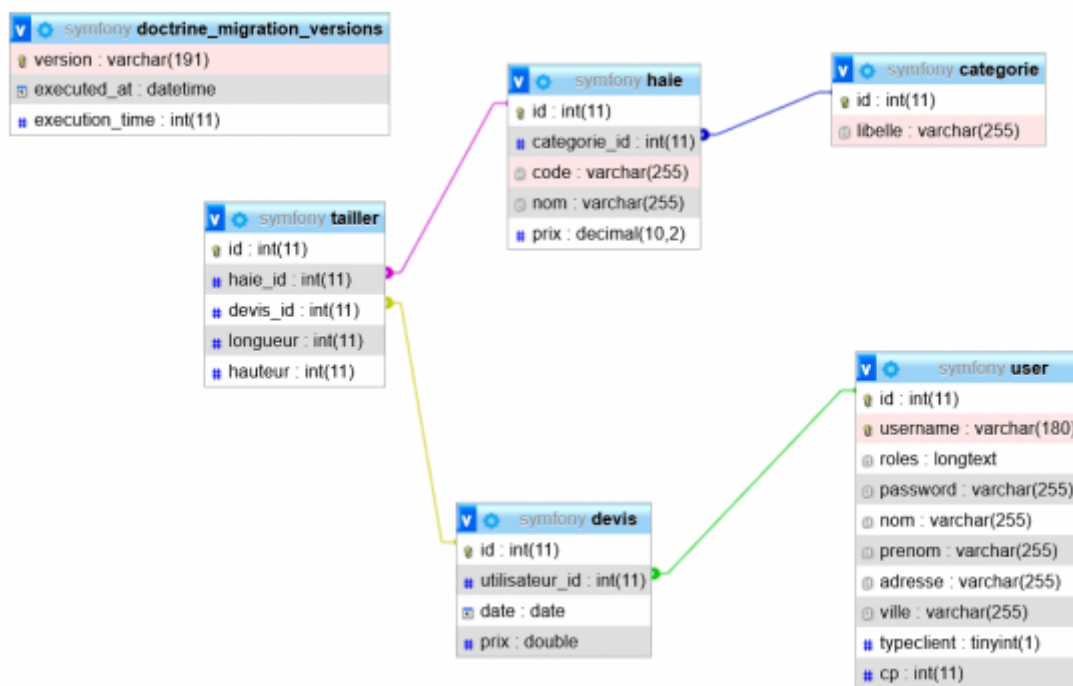
Permanent link:

<http://ppe.boonum.fr/doku.php?id=slam:ws:2021:sio:ap3.2:equipe2:accueil>

Last update: **2022/03/30 08:20**



LeP'titJardinier - Ma Base de donnée



From:
<http://ppe.boonum.fr/> - **AP.SIO**

Permanent link:
<http://ppe.boonum.fr/doku.php?id=slam:ws:2021:sio:ap3.2:equipe2:bdd>

Last update: **2022/03/30 00:22**



LeP'titJardinier - Connexion



[Retour](#)

Nous devons ajouter la fonctionnalité de **gestion des clients** :

- Gestion des clients : recherche/ajout/modification/suppression des clients

Or en implémentant la table **client**, j'ai décidé de transformer cette entité en **système de connexion**. Grâce à cette méthode, un **utilisateur** pourra effectuer **plusieurs devis** et donc par conséquent être relié à ce dernier sans effectuer de sélection dans une liste déroulante par exemple.

Pour ce faire, nous commençons avec la commande :

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

Cette commande permet de créer tous les champs utiles à un système de connexion lambda, que l'on retrouve dans les applications web modernes :

- Nom d'utilisateur
- Mot de passe
- Des rôles

Pour des soucis de simplicité de conception, j'ai **réuni les entités Clients et Utilisateurs**, j'ai donc rajouté les champs de **client** (les informations personnelles) dans **user** grâce à la commande :

```
php bin/console make:entity
```

Connexion

J'ai ensuite créé la page de connexion grâce à la fonction, elle me permet de générer un formulaire, un contrôleur et une vue en twig pour la page de login :

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

Inscription

Pour le système d'inscription, Symfony propose également une commande qui permet de lier une entité User à un système d'inscription.

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

Gestion des roles & Permissions

Ce système de connexion et d'inscription me permet d'effectuer un contrôle de connexion et de rôle sur cette page. Rendons nous dans le fichier **security.yaml** :

```
access_control:
    - { path: ^/client, roles: ROLE_ADMIN }
    - { path: ^/devis, roles: ROLE_USER }
    - { path: ^/haie, roles: ROLE_USER }
```

Comme on peut le voir ici, les pages de **devis**, **clients**, et **haie** nécessitent une **connexion**. Cependant, la page de **client** nécessite une permission particulière, celle d'**administrateur**. Cette fonctionnalité de d'authentification permet aussi de récupérer n'importe quelle information de l'utilisateur connecté grâce

- à la fonction (en PHP) :

```
$user = $this->getUser();
```

- à la ligne suivante (en twig) (app.user.nom pour le nom de l'utilisateur) :

```
{{app.user}}
```

Dans la page du panier, un utilisateur ayant le role ADMIN possède un bouton supplémentaire lui permettant de voir la liste des devis, tout ceci grace à cette condition :

```
{% if is_granted('ROLE_ADMIN') %}
    <div class="optn_line_ctn">
        <a class="optn_btn_link" href="{{ path('devis_liste') }}">
            Administrer les devis
            
        </a>
    </div>
{% endif %}
```

From:

<http://ppe.boonum.fr/> - **AP.SIO**

Permanent link:

<http://ppe.boonum.fr/doku.php?id=slam:ws:2021:sio:ap3.2:equipe2:connexion>

Last update: **2022/03/30 01:33**



LeP'titJardinier - Utilisateurs



[Retour](#)

Entité dans Symfony

Tout d'abord, voici l'entité Utilisateurs : [Entité Utilisateur](#)

Création du CRUD

La **commande suivante** m'a permis de générer **toutes les pages** et **fonctions** utiles à la **gestion des utilisateur** :

- Affichage en liste
- Affichage individuel
- Modification
- Suppression

```
php bin/console make:crud
```

Contrôleur des utilisateurs

```
#[Route('/client')]
class UserController extends AbstractController
{
    #[Route('/', name: 'user_index', methods: ['GET'])]
    public function index(UserRepository $userRepository): Response
    {
        return $this->render('user/index.html.twig', [
            'users' => $userRepository->findAll(),
        ]);
    }
}
```



```
    });
}

#[Route('/new', name: 'user_new', methods: ['GET','POST'])]
public function new(Request $request): Response
{
    $user = new User();
    $form = $this->createForm(UserType::class, $user);
    $form->handleRequest($request);

    if ($form->isSubmitted() && $form->isValid()) {
        $entityManager = $this->getDoctrine()->getManager();
        $entityManager->persist($user);
        $entityManager->flush();

        return $this->redirectToRoute('user_index', [],
Response::HTTP_SEE_OTHER);
    }

    return $this->renderForm('user/new.html.twig', [
        'user' => $user,
        'form' => $form,
    ]);
}

#[Route('/{id}', name: 'user_show', methods: ['GET'])]
public function show(User $user): Response
{
    return $this->render('user/show.html.twig', [
        'user' => $user,
    ]);
}

#[Route('/{id}/edit', name: 'user_edit', methods: ['GET','POST'])]
public function edit(Request $request, User $user): Response
{
    $form = $this->createForm(UserType::class, $user);
    $form->handleRequest($request);

    if ($form->isSubmitted() && $form->isValid()) {
        $this->getDoctrine()->getManager()->flush();

        return $this->redirectToRoute('user_index', [],
Response::HTTP_SEE_OTHER);
    }

    return $this->renderForm('user/edit.html.twig', [
        'user' => $user,
        'form' => $form,
        // 'registrationForm' => $form,
    ]);
}
```

```
    });
}

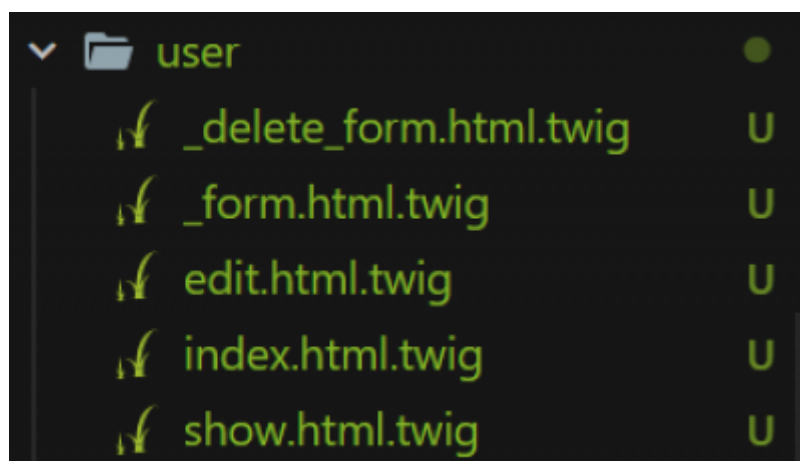
#[Route('/{id}', name: 'user_delete', methods: ['POST'])]
public function delete(Request $request, User $user): Response
{
    if ($this->isCsrfTokenValid('delete'.$user->getId(),
$request->request->get('_token'))) {
        $entityManager = $this->getDoctrine()->getManager();
        $entityManager->remove($user);
        $entityManager->flush();
    }

    return $this->redirectToRoute('user_index', [],
Response::HTTP_SEE_OTHER);
}

#[Route('/', name: 'user_search', methods: ['POST'])]
public function search(UserRepository $userRepository): Response
{
    $request = Request::createFromGlobals();
    $user_recherche=$request->get('search_user');
    return $this->render('user/index.html.twig', [
        'users' => $userRepository->findBy(['nom' => $user_recherche]),
    ]);
}
}
```

Comme on peut le voir, j'ai rajouter une fonction **search** qui n'était pas présent dans le **CRUD**, cette dernière permet de **recupérer le nom** écrit dans le champ de recherche de la vue et de faire une **recherche par nom** à l'aide de **doctrine**.

Vues générées par CRUD



Last update: 2022/03/30
01:51

slam:ws:2021:sio:ap3.2:equipe2:clients <http://ppe.boonum.fr/doku.php?id=slam:ws:2021:sio:ap3.2:equipe2:clients>

From:

<http://ppe.boonum.fr/> - **AP.SIO**

Permanent link:

<http://ppe.boonum.fr/doku.php?id=slam:ws:2021:sio:ap3.2:equipe2:clients>

Last update: **2022/03/30 01:51**



LeP'titJardinier - Entité Utilisateur



[Retour](#)

```
class User implements UserInterface, PasswordAuthenticatedUserInterface
{
    /**
     * @ORM\Id
     * @ORM\GeneratedValue
     * @ORM\Column(type="integer")
     */
    private $id;

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

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

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

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

    /**
     * @ORM\Column(type="string", length=255)
     */
}
```

```
private $prenom;

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

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

/**
 * @ORM\Column(type="boolean")
 */
private $typeclient;

/**
 * @ORM\Column(type="integer")
 */
private $cp;

/**
 * @ORM\OneToMany(targetEntity=Devis::class, mappedBy="Utilisateur")
 */
private $devis;

public function __construct()
{
    $this->devis = new ArrayCollection();
}

public function getId(): ?int
{
    return $this->id;
}

/**
 * @deprecated since Symfony 5.3, use getUserIdentifier instead
 */
public function getUsername(): string
{
    return (string) $this->username;
}

public function setUsername(string $username): self
{
    $this->username = $username;
}
```

```
        return $this;
    }

    /**
     * A visual identifier that represents this user.
     *
     * @see UserInterface
     */
    public function getUserIdentifier(): string
    {
        return (string) $this->username;
    }

    /**
     * @see UserInterface
     */
    public function getRoles(): array
    {
        $roles = $this->roles;
        // guarantee every user at least has ROLE_USER
        $roles[] = 'ROLE_USER';

        return array_unique($roles);
    }

    public function setRoles(array $roles): self
    {
        $this->roles = $roles;

        return $this;
    }

    /**
     * @see PasswordAuthenticatedUserInterface
     */
    public function getPassword(): string
    {
        return $this->password;
    }

    public function setPassword(string $password): self
    {
        $this->password = $password;

        return $this;
    }

    /**
     * Returning a salt is only needed, if you are not using a modern
     * hashing algorithm (e.g. bcrypt or sodium) in your security.yaml.
     */
```

```
    * @see UserInterface
    */
    public function getSalt(): ?string
    {
        return null;
    }

    /**
     * @see UserInterface
     */
    public function eraseCredentials()
    {
        // If you store any temporary, sensitive data on the user, clear it
here        // $this->plainPassword = null;
    }

    public function getNom(): ?string
    {
        return $this->nom;
    }

    public function setNom(string $nom): self
    {
        $this->nom = $nom;

        return $this;
    }

    public function getPrenom(): ?string
    {
        return $this->prenom;
    }

    public function setPrenom(string $prenom): self
    {
        $this->prenom = $prenom;

        return $this;
    }

    public function getAdresse(): ?string
    {
        return $this->adresse;
    }

    public function setAdresse(string $adresse): self
    {
        $this->adresse = $adresse;
    }
}
```

```
        return $this;
    }

    public function getVille(): ?string
    {
        return $this->ville;
    }

    public function setVille(string $ville): self
    {
        $this->ville = $ville;

        return $this;
    }

    public function getTypeClient(): ?bool
    {
        return $this->typeclient;
    }

    public function setTypeClient(bool $type_client): self
    {
        $this->typeclient = $type_client;

        return $this;
    }

    public function getCp(): ?int
    {
        return $this->cp;
    }

    public function setCp(int $cp): self
    {
        $this->cp = $cp;

        return $this;
    }

    /**
     * @return Collection|Devis[]
     */
    public function getDevis(): Collection
    {
        return $this->devis;
    }

    public function addDevi(Devis $devi): self
    {
        if (!$this->devis->contains($devi)) {
```



```
        $this->devis[] = $devi;  
        $devi->setUtilisateur($this);  
    }  
  
    return $this;  
}  
  
public function removeDevi(Devis $devi): self  
{  
    if ($this->devis->removeElement($devi)) {  
        // set the owning side to null (unless already changed)  
        if ($devi->getUtilisateur() === $this) {  
            $devi->setUtilisateur(null);  
        }  
    }  
  
    return $this;  
}  
}
```

From:
<http://ppe.boonum.fr/> - **AP.SIO**

Permanent link:
<http://ppe.boonum.fr/doku.php?id=slam:ws:2021:sio:ap3.2:equipe2:clients:entite>

Last update: **2022/03/30 00:59**



LeP'titJardinier - Type de client & Devis



[Retour](#)

Un utilisateur peut posséder 2 types de status :

- Entreprise : 10% de promotion sur le devis
- Particulier : Aucune promotion sur le devis

Pour effectuer cette promotion, je récupère la variable `typeclient` de l'utilisateur connecté, cette variable est de type boolean, si elle est vraie alors l'utilisateur est une entreprise sinon c'est un particulier. Dans ce cas précis, le type boolean est plus utile, mais si il faut rajouter d'autres types dans le futur, cela va poser problème, j'ai donc fait un choix.

Si l'utilisateur est une entreprise, une indication de promotion de 10% s'affiche :



PROMOTION DE 10% POUR LES ENTREPRISES

```
<div class="promo_ctn">
  
  <h1>
    {% if app.user.typeclient %}
    Promotion de <span>10%</span> pour les <span>entreprises</span>
    {% else %}
    Aucune promotion pour les particuliers
    {% endif %}
  </h1>
</div>
```


Panier de Sébastien :

x1 Sapin

↔ 1m | ↑ 1m

40 €



PRIX : ~~40€~~ (-10%) **36€**

Valider le panier



```
<p class="cart_info"> Prix :
    {% if app.user.typeclient %}
    <strike>{{prixT}}€ </strike>
    (-10%)
    <span class="cart_prix">{{prixT - (prixT * 0.1)}}€</span>
    {% else %}
    <span class="cart_prix">{{prixT}}€</span>
    {% endif %}
</p>
```

From:

<http://ppe.boonum.fr/> - **AP.SIO**

Permanent link:

<http://ppe.boonum.fr/doku.php?id=slam:ws:2021:sio:ap3.2:equipe2:devisclient>Last update: **2022/03/30 01:30**

LeP'titJardinier - Ajout d'une commande au panier

[Retour](#)

Ajout d'une commande

Nous avons un formulaire qui nous permet de sélectionner le type de haie ainsi que la hauteur et la longueur que l'on veut tailler.

```
<form action="{{ path('devis') }}" method="post">

    <h1>Ajouter une haie à votre panier !</h1>
    <select name="haie_select_devis" required>
        <option value="" selected disabled>Selectionner votre
haie....</option>
        {% for haie in haies %}
            <option value="{{ haie.getId() }}">{{ haie.getNom() }}</option>
        {% endfor %}
    </select>
    <input type="text" value="Hauteur" />
    <input type="text" value="Longueur" />
    <button type="submit" value="Ajouter au panier" />
</form>
```

```
        </select>
        <input type="text" placeholder="Hauteur" name="hauteur"
required>
        <input type="text" placeholder="Longueur" name="longueur"
required>
        <button class="btn btn-success" type="submit">Ajouter au
panier</button>
    </form>
```

La liste des haies est récupéré grâce au contrôleur et à une fonction simple de Doctrine :

```
$mesHaies = $this->getDoctrine()
    ->getRepository(Haie::class)
    ->findAll();
```

Voici donc le moment de valider notre choix de haie, l'ensemble de nos données sont récupérées en **POST** et sont **ajoutées dans le panier** (sous forme d'objet), qui est **une variable session** :

```
// Session initialisation
    $session = new Session();
    if($session->get('kart')){
        $kart = $session->get('kart');
    } else {
        $kart = array();
    }

    if($request->isMethod('POST')){
        $haie_id = $request->get('haie_select_devis');
        $haie =
$this->getDoctrine()->getRepository(Haie::class)->find($haie_id);

        $tailler = new Tailler();
        $tailler->setHauteur($request->get('hauteur'));
        $tailler->setLongueur($request->get('longueur'));
        $tailler->setHaie($haie);

        array_push($kart,$tailler);
        $session->set('kart',$kart);
    }
```

Le **prix total** (sans promotion) est donc ensuite très facilement calculé grâce à ce **tableau d'objet**.

Le prix doit respecter les **contraintes suivantes** :

- Prix = Prix unitaire(*) x Longueur haie
- Si Hauteur > 1m50, multiplier le prix par 1.5

```
$prixTotal= 0;
    foreach ($kart as $product) {
        if($product->getHauteur() > 1.5) {
```

```
                $prixTotal = $prixTotal + $product->getHaie()->getPrix() *  
$product->getLongueur()*1.5;  
            } else {  
                $prixTotal = $prixTotal + $product->getHaie()->getPrix() *  
$product->getLongueur();  
            }  
        }  
    }
```

Suppression d'une commande

J'ai également fait un **système de suppression de commande**. Dans le code de la vue, on pose un lien qui envoie vers le contrôleur de suppression avec la position dans la boucle de **tailler**, en **GET**.

```
<a class="link_icon" href="{{ path('devis_remove', {'id': loop.index - 1})  
}}">  
      
</a>
```

Cette position **loop.index-1** (le -1 est dû au décalage de la boucle for du contrôleur qui commence à 0) me permet d'aller **calquer la position sur l'array du panier** dans la session et de **supprimer l'objet** correspondant à cet indice.

From:

<http://ppe.boonum.fr/> - **AP.SIO**

Permanent link:

<http://ppe.boonum.fr/doku.php?id=slam:ws:2021:sio:ap3.2:equipe2:devis>

Last update: **2022/03/30 02:12**



LeP'titJardinier - Validation du devis & Liste



[Retour](#)

Validation du panier & Calcul du devis

Lorsque l'on valide le panier, nous sommes envoyés vers la validation du devis :

```
#[Route('/devis/validation', name: 'devis_add')]
public function devis_valid(Request $request, AuthenticationUtils
$authenticationUtils): Response
{
    if ($request->isMethod('POST')){
        $entityManager = $this->getDoctrine()->getManager();
        $date = date("d/m/Y");
        $session = new Session();
        if($session->get('kart')){
            $kart = $session->get('kart');
            $prixT = 0;
            foreach($kart as $haie ){
                if($haie->getHauteur() > 1.5){
                    $prixT = $prixT + $haie->getHaie()->getPrix() *
$haie->getLongueur() * 1.5;
                } else {
                    $prixT = $prixT + $haie->getHaie()->getPrix() *
$haie->getLongueur();
                }
            }
            if($this->getUser()->getTypeClient()){
                $prixT = $prixT - $prixT * 0.1;
            }

            $devis = new Devis();
            date_default_timezone_set('Europe/Paris');
```

```
$devis->setDate(new DateTime());
$devis->setPrix($prixT);
$devis->setUtilisateur($this->getUser());
$entityManager->persist($devis);
$entityManager->flush();

foreach ($kart as $haie) {
    $tailler = new Tailler();
    $tailler->setHauteur($haie->getHauteur());
    $tailler->setLongueur($haie->getLongueur());
    $tailler->setHaie($haie->getHaie());
    $tailler->setDevis($devis);

    $entityManager->merge($tailler);
    $entityManager->flush();
}

$kart = $session->remove('kart');
} else {
    return $this->redirectToRoute('devis');
}

}
return $this->render('devis/validation.html.twig', [
    'controller_name' => 'Mon devis',
    'haies'=> $kart,
    'prixT'=>$prixT,
    'date'=>$date,
]);
}
```

Symfony récupère la variable session du panier et l'exploite il calcule le montant du devis (avec promotion ou pas) et l'ajoute dans la base de donnée. Il ajoute aussi les entités tailler relatif à ce devis. Puis après avoir effectuer tous les calculs, la variable session du panier est vidé.



P'TIT JARDINIER

SINGH
Paul
truc
87000
Limoges

Haie	Prix unitaire	Longueur	Hauteur	Prix
Sapin	40.00	10	10	600

Créé le : 29/03/2022

Prix à payer : 600 €

Affichage des devis

L'affichage de la **liste des devis** est soumis à une **certaine permission de rôle** (voir : [Permissions Utilisateurs](#)).

Grace à la **liaison précédemment effectuée**, le code PHP est **assez simple**, il nous suffit juste de récupérer l'ensemble des **devis**, ce qui va récupérer l'ensemble des **tailler** et des **haies** correspondantes :


```
#[Route('/devis/liste', name: 'devis_liste')]
public function listeDevis(Request $request): Response
{
    $mesDevis = $this->getDoctrine()
        ->getRepository(Devis::class)
        ->findAll();

    return $this->render('devis/liste.html.twig', [
        'controller_name' => 'DevisController',
        'devis'=>$mesDevis,
    ]);
}
```

C'est dans le twig que la magie opère :

```
<table id="customclient_table">
  <thead>
    <tr>
      <th>Date</th>
      <th>Utilisateur</th>
      <th>Haie(s)</th>
      <th>Montant</th>
```

```
</tr>
</thead>
<tbody>
{% for devi in devis %}
  <tr class="table_client_line">
    <td>{{ devi.date |date('d-m-Y') }}</td>
    <td>{{devi.utilisateur.username}}</td>
    <td>
      <details>
        <summary>Voir les haies</summary>
        {% for tailler in devi.taillers %}
          {{tailler.getHaie().getNom}} : ↔ {{tailler.getLongueur}}m †
          {{tailler.Hauteur}}m<br>
        {% endfor %}
      </details>
    </td>
    <td>{{devi.prix}}€</td>
  </tr>
{% else %}
  <tr>
    <td colspan="4">Aucun devis</td>
  </tr>
{% endfor %}
</tbody>
</table>
```

[Retour à la gestion du panier](#) 

Date	Utilisateur	Haie(s)	Montant
29-03-2022	admin	► Voir les haies	540€
29-03-2022	admin	► Voir les haies	702€
29-03-2022	admin	► Voir les haies	702€
29-03-2022	user	► Voir les haies	40€
29-03-2022	admin	► Voir les haies	9€
29-03-2022	admin	► Voir les haies	540€
29-03-2022	admin	► Voir les haies	9€
29-03-2022	admin	► Voir les haies	180€
29-03-2022	admin	► Voir les haies	450€
29-03-2022	admin	► Voir les haies	180€

(Nous avons la possibilité d'ouvrir le menu contextuel **Voir les haies**, qui fait apparaître la **liste des haies** commandées dans le **devis correspondant**

Date	Utilisateur	Haie(s)	Montant
29-03-2022	admin	▼ Voir les haies Sapin : ⇐ 10m ⇑ 3m	540€
29-03-2022	admin	► Voir les haies	702€
29-03-2022	admin	► Voir les haies	702€

From:

<http://ppe.boonum.fr/> - **AP.SIO**

Permanent link:

<http://ppe.boonum.fr/doku.php?id=slam:ws:2021:sio:ap3.2:equipe2:devisvalid>

Last update: **2022/03/30 02:26**



LeP'titJardinier - Haies



[Retour](#)

Entité dans Symfony

Tout d'abord, voici l'entité haie : [Entité haie](#)

Création du CRUD

La **commande suivante** m'a permis de générer **toutes les pages** et **fonctions** utiles à la **gestion des utilisateur** :

- Affichage en liste
- Affichage individuel
- Modification
- Suppression

```
php bin/console make:crud
```

Contrôleur des haies

```
#[Route('/haie')]
class HaieController extends AbstractController
{
    #[Route('/', name: 'haie_index', methods: ['GET'])]
    public function index(HaieRepository $haieRepository): Response
    {
        return $this->render('haie/index.html.twig', [
            'haies' => $haieRepository->findAll(),
        ]);
    }
}
```

```
    });
}

#[Route('/cate_add', name: 'cate_add')]
public function add_catego(Request $request): Response
{
    if ($request->isMethod('POST')){
        $categorie = new Categorie();
        $categorie->setLibelle($request->get('libelle'));
        $entityManager = $this->getDoctrine()->getManager();

        $entityManager->persist($categorie);
        $entityManager->flush();
    }

    return $this->render('haie/categorie.html.twig', [
    ]);
}

#[Route('/new', name: 'haie_new', methods: ['GET','POST'])]
public function new(Request $request): Response
{
    $haie = new Haie();
    $form = $this->createForm(HaieType::class, $haie);
    $form->handleRequest($request);

    if ($form->isSubmitted() && $form->isValid()) {
        $entityManager = $this->getDoctrine()->getManager();
        $entityManager->persist($haie);
        $entityManager->flush();

        return $this->redirectToRoute('haie_index', [],
Response::HTTP_SEE_OTHER);
    }

    return $this->renderForm('haie/new.html.twig', [
        'haie' => $haie,
        'form' => $form,
    ]);
}

#[Route('/{id}', name: 'haie_show', methods: ['GET'])]
public function show(Haie $haie): Response
{
    return $this->render('haie/show.html.twig', [
        'haie' => $haie,
    ]);
}

#[Route('/{id}/edit', name: 'haie_edit', methods: ['GET','POST'])]
```

```
public function edit(Request $request, Haie $haie): Response
{
    $form = $this->createForm(HaieType::class, $haie);
    $form->handleRequest($request);

    if ($form->isSubmitted() && $form->isValid()) {
        $this->getDoctrine()->getManager()->flush();

        return $this->redirectToRoute('haie_index', [],
Response::HTTP_SEE_OTHER);
    }

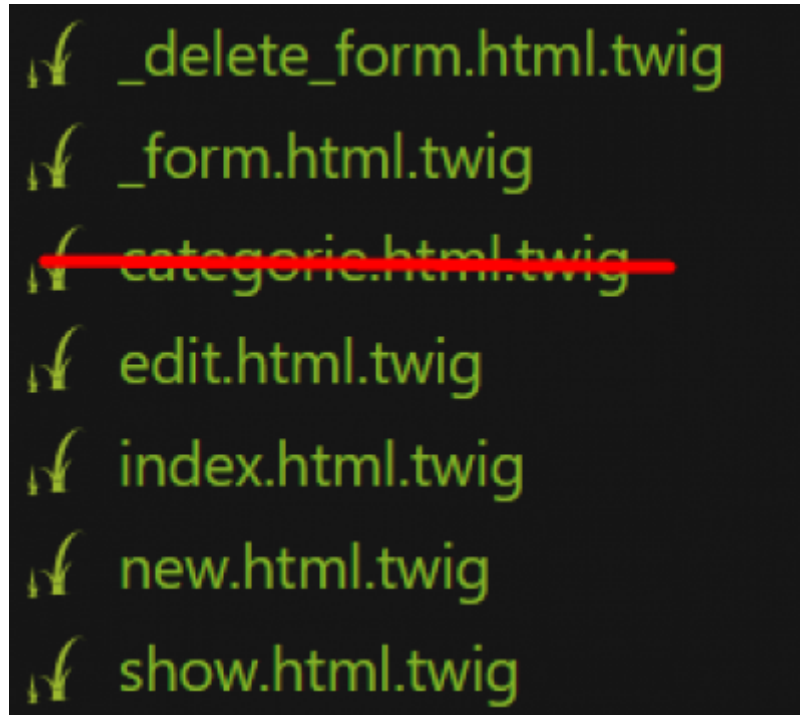
    return $this->renderForm('haie/edit.html.twig', [
        'haie' => $haie,
        'form' => $form,
    ]);
}

#[Route('/{id}', name: 'haie_delete', methods: ['POST'])]
public function delete(Request $request, Haie $haie): Response
{
    if ($this->isCsrfTokenValid('delete'.$haie->getCode(),
$request->request->get('_token'))) {
        $entityManager = $this->getDoctrine()->getManager();
        $entityManager->remove($haie);
        $entityManager->flush();
    }

    return $this->redirectToRoute('haie_index', [],
Response::HTTP_SEE_OTHER);
}
}
```

Comme on peut le voir, j'ai rajouter une fonction **add_catego** qui n'était pas présent dans le **CRUD**, cette dernière permet de **créer** une catégorie grâce à un simple formulaire en POST.

Vues générées par CRUD



From:
<http://ppe.boonum.fr/> - **AP.SIO**

Permanent link:
<http://ppe.boonum.fr/doku.php?id=slam:ws:2021:sio:ap3.2:equipe2:haies>

Last update: **2022/03/30 02:29**



LeP'titJardinier - Entité Haie



[Retour](#)

```
class Haie
{
    /**
     * @ORM\Id
     * @ORM\GeneratedValue
     * @ORM\Column(type="integer")
     */
    private $id;

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

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

    /**
     * @ORM\Column(type="decimal", precision=10, scale=2)
     */
    private $prix;

    /**
     * @ORM\ManyToOne(targetEntity=Categorie::class, inversedBy="haies")
     * @ORM\JoinColumn(nullable=false)
     */
    private $categorie;

    /**
     * @ORM\OneToMany(targetEntity=Tailler::class, mappedBy="haie")
     */
}
```

```
private $tailleurs;

public function __construct()
{
    $this->tailleurs = new ArrayCollection();
}

public function __toString()
{
    return $this->nom;
}

public function getId(): ?int
{
    return $this->id;
}

public function getCode(): ?string
{
    return $this->code;
}

public function setCode(string $code): self
{
    $this->code = $code;

    return $this;
}

public function getNom(): ?string
{
    return $this->nom;
}

public function setNom(string $nom): self
{
    $this->nom = $nom;

    return $this;
}

public function getPrix(): ?string
{
    return $this->prix;
}

public function setPrix(string $prix): self
{

```

```
        $this->prix = $prix;

        return $this;
    }

    public function getCategorie(): ?Categorie
    {
        return $this->categorie;
    }

    public function setCategorie(?Categorie $categorie): self
    {
        $this->categorie = $categorie;

        return $this;
    }

    /**
     * @return Collection|Tailler[]
     */
    public function getTaillers(): Collection
    {
        return $this->taillers;
    }

    public function addTailler(Tailler $tailler): self
    {
        if (!$this->taillers->contains($tailler)) {
            $this->taillers[] = $tailler;
            $tailler->setHaie($this);
        }

        return $this;
    }

    public function removeTailler(Tailler $tailler): self
    {
        if ($this->taillers->removeElement($tailler)) {
            // set the owning side to null (unless already changed)
            if ($tailler->getHaie() === $this) {
                $tailler->setHaie(null);
            }
        }

        return $this;
    }
}
```

Last update:

2022/03/30 02:31 slam:ws:2021:sio:ap3.2:equipe2:haies:entite <http://ppe.boonum.fr/doku.php?id=slam:ws:2021:sio:ap3.2:equipe2:haies:entite>

From:

<http://ppe.boonum.fr/> - **AP.SIO**

Permanent link:

<http://ppe.boonum.fr/doku.php?id=slam:ws:2021:sio:ap3.2:equipe2:haies:entite>

Last update: **2022/03/30 02:31**

