Projet Maya

(***en cours de formation, travail en équipe***)

Contexte

La Ferme Maya propose des fruits et légumes de saison issus d'une agriculture raisonnée ou biologique. Les clients de la ferme peuvent composer leur propre panier ou choisir parmi les paniers de la semaine. Leurs achats sont mis à disposition à la Ferme Maya le jour de leur choix.

Le site vitrine de la ferme est ancien et ne donne plus satisfaction. La ferme fait appel à nos services pour effectuer la refonte du site.

Méthodologie de la gestion du projet Maya

Nous utilisons la méthode Agile Scrum sur Trello pour assurer le travail collaboratif.

La méthode Scrum propose de catégoriser les projets en fonction des critères "À faire", "En cours" et "Terminés" de manière à pouvoir accompagner en temps réel leur avancement.

Trello nous permet d'implémenter la méthode Scrum en toute simplicité. L'objectif principal étant défini nous utilisons Trello pour subdiviser cet objectif en liste de tâches individuelles.

Cette méthode se compose d'étapes nommées « sprint ». Un sprint est une itération agile d'une durée de 1 à 4 semaines pendant laquelle l'équipe projet effectue un travail donné et livre un incrément, afin de répondre à un objectif donné. Ce travail en cycle court permet de s'adapter rapidement aux évolutions et de recueillir le feedback des utilisateurs.

Gestion des versions

La gestion des versions se fait par Git et GitHub. Chaque développeur travaille sur son dépôt Git. Le dépôt GitHub permet la synchronisation.

Ressources logicielles

Framework Symfony, Serveur web PHP, serveur de base de données MySql

SPRINT 1

Sécurité, Authentification, autorisations et gestion des utilisateurs

Pour développer le back office c'est-à-dire l'application web d'administration des données, nous utilisons le Framework Symfony.

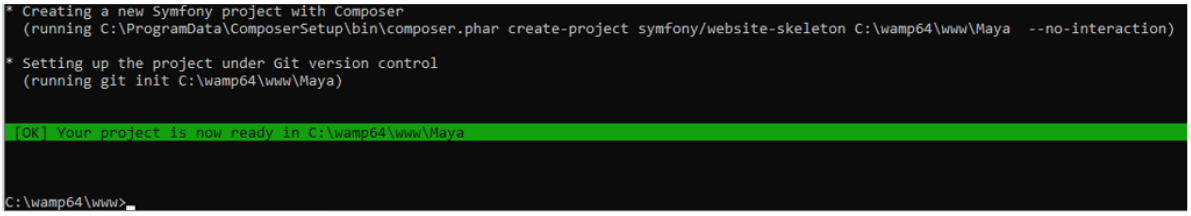
Il s'agit donc de créer Symfony et de mettre en place le mécanisme d'authentification avec Symfony.

L'administration des données de l'application Maya ne sera pas accessible à tous les utilisateurs. Il convient de mettre en place un système d'authentification avec 2 profils d'utilisateurs à gérer dans le back office :

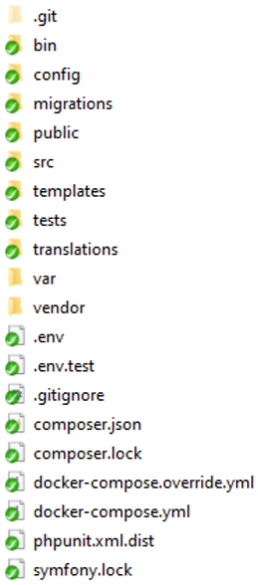
- Administrateur : accès total
- Utilisateur : accès à tout sauf gestion des catégories et gestion des utilisateurs

1- Création de l'application

```
C:\wamp64\www>Symfony new Maya --version=5.4 --webapp
* Creating a new Symfony project with Composer
  (running C:\ProgramData\ComposerSetup\bin\composer.phar create-project symfony/website-skeleton
C:\wamp64\www\Maya --no-interaction)

* Setting up the project under Git version control
  (running git init C:\wamp64\www\Maya)

 [OK] Your project is now ready in C:\wamp64\www\Maya

C:\wamp64\www>
```



Contenu du dossier Maya ajouté



2- Création du template de base

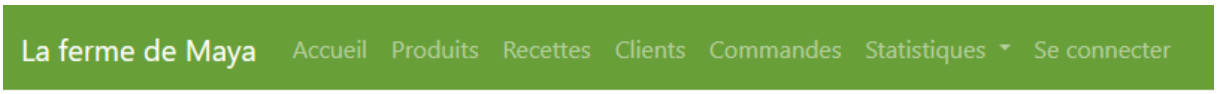
Un template de base est généré lors de la création de l'application. Il faut le modifier et définir le design commun à toutes les pages. Dans notre cas il s'agit de définir les feuilles de styles, la barre de navigation et le conteneur principal.

3- Création des contrôleurs Accueil, Produits et Catégorie

Ils permettront de tester l'affichage du template de base dans un premier temps et tester les autorisations d'accès par la suite.

```
C:\wamp64\www\Maya>php bin/console make:controller AccueilController

 created: src/Controller/AccueilController.php
 created: templates/accueil/index.html.twig


  Success!


 Next: Open your new controller class and add some pages!

C:\wamp64\www\Maya>
```

Test du template



4- Sécurité et authentification avec Symfony

Dans le contexte Maya l'identité numérique sera stockée dans la base de données. Un formulaire d'authentification permettra aux utilisateurs de saisir leurs données d'identification (email, mot de passe) pour accéder à l'application. La gestion des données des utilisateurs (CRUD) sera également prise en charge par l'application.

Nous allons donc utiliser plusieurs bundles (package) de l'écosystème Symfony :

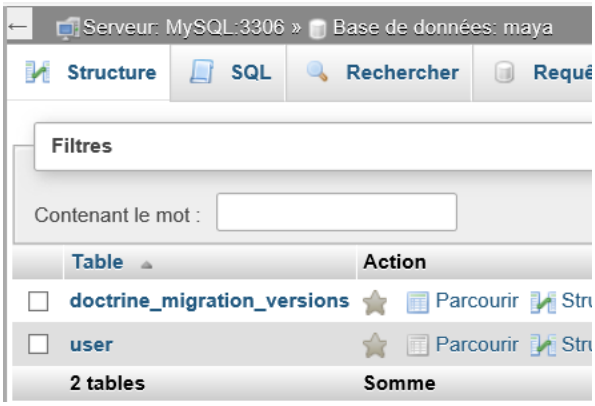
- Le FOSUserBundle
- Le MakerBundle
- Les bundles Doctrine :
- doctrine/annotations, doctrine/doctrine-bundle, doctrine/doctrine-migrations-bundle, doctrine/orm et doctrine/doctrine-fixtures-bundle

5- Mise en place du mécanisme d'authentification dans Maya

Le mécanisme d'authentification consiste à établir un lien irréfutable entre une personne et une identité numérique.

Les différentes étapes :

- Configurer et créer la base de données
- Créer la classe User (l'entité User)
- Créer la table User
- Créer des utilisateurs de test avec DoctrineFixturesBundle

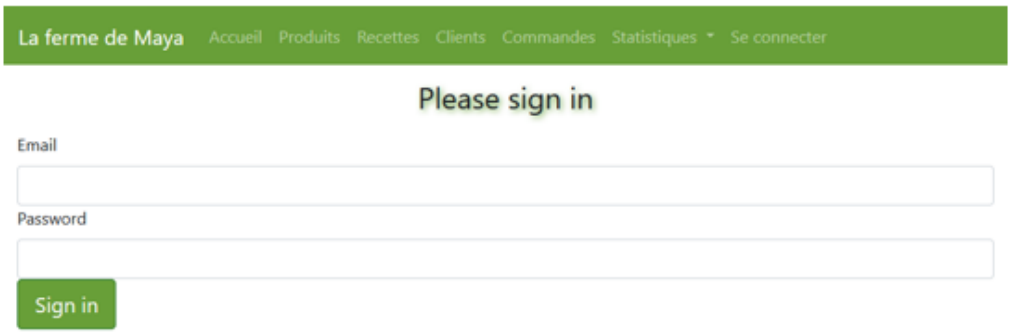


6- Gérer l'authentification

S'authentifier c'est confirmer son identité en fournissant par exemple un nom d'utilisateur (username) ou un email et un mot de passe sur un système.

Tâches :

- Créer un formulaire de connexion
- Afficher le nom de l'utilisateur
- Gérer la déconnexion



7- Gérer les autorisations

Les autorisations permettent d'interdire ou d'autoriser l'accès à certaines pages. En s'authentifiant on décline son identité et en fonction de cette identité on est autorisé ou non à accéder à certaines pages.

Pour Maya, 2 rôles :

- ROLE_ADMIN : administrateur, accès total
- ROLE_USER : utilisateur, accès à toutes les pages sauf celles de la gestion des catégories

8 Gérer les utilisateurs

Ce n'est pas l'utilisateur qui peut s''inscrire mais l'administrateur qui inscrira les utilisateurs. Seul le rôle ROLE_ADMIN aura accès à la gestion des utilisateurs.

SPRINT 2

L'ORM Doctrine, les formulaires Doctrine

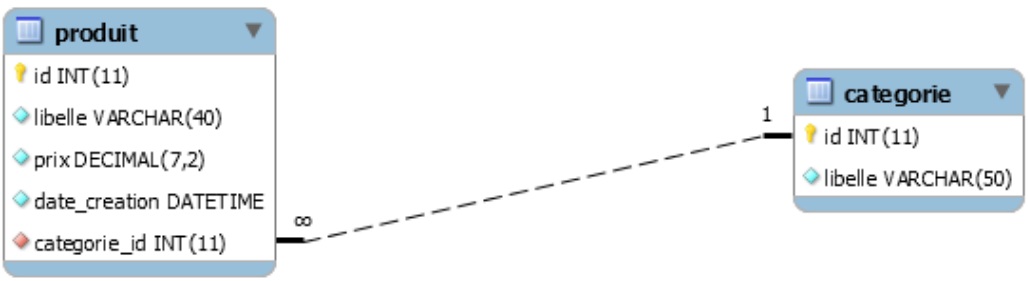
Le but de cette mission est de pouvoir créer des formulaires et de manipuler les données de la base en effectuant des opérations CRUD.

L'acronyme informatique anglais CRUD (*pour Create, Read, Update, Delete*) désigne les quatre opérations de base pour la persistance des données, en particulier le stockage d'informations en base de données.

Nous abordons la notion d'entité (Les données de la base de données sont des objets. Un objet dont l'enregistrement est confié à l'ORM se nomme une entité (entity en anglais). On dit également persister une entité, plutôt qu'enregistrer une entité.

Dans ce sprint nous utilisons l'un des composants de Symfony, l'ORM Doctrine, en utilisant un extrait des données de l'application future, les produits (pomme, poire, cerise, courgette, aubergine, …) et leur catégorie (Fruits, Légumes, …) ainsi que les formulaires Doctrine.

Schéma relationnel :



Les opérations de base avec une entité (une classe métier, une table)

- Création de l'entité Produit

```
C:\wamp64\www\Maya>php bin/console make:entity

Class name of the entity to create or update (e.g. GentlePizza):
> Produit

created: src/Entity/Produit.php
created: src/Repository/ProduitRepository.php

Entity generated! Now let's add some fields!
```

- Création de la table Produit

```
C:\wamp64\www\Maya>php bin/console make:migration


  Success!
```

- Ajout du constructeur à la classe Produit

```php
    public function __construct()
    {
        $this->dateCreation = new \DateTime('now');
    }
```

- Persister les objets dans le contrôleur

Nouveau produit enregistré, son id est : 1

- Lire un produit de la base de données

Voici le libellé du produit : haricots verts

- Modifier un produit de la base de données

Voici le libellé du produit : haricots verts fins

- Supprimer un produit de la base de données

Le produit a été supprimé, id : 2

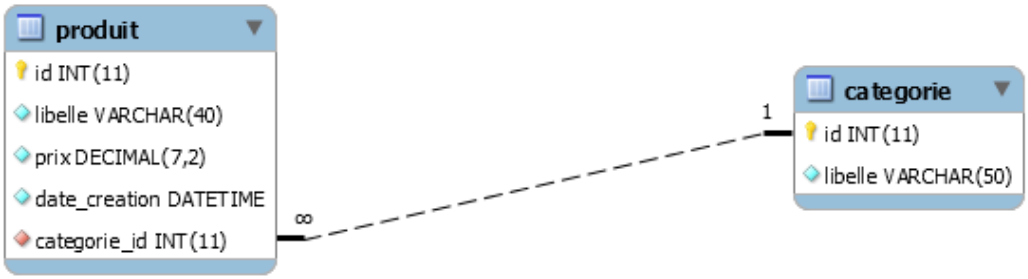
- Création d'une nouvelle requête dans le repository

Ici ajout d'une méthode pour récupérer tous les produits dont le prix est supérieur à un paramètre

```
Voici le nombre de produits : 3
```

Les opérations de base avec deux entités associées

Rappel du schéma relationnel :



- Création d'une association ManyToOne / OneToMany

Un produit appartient à une seule catégorie et une catégorie comprend plusieurs produits.

Pour Doctrine ce cas correspond au type d'association : **ManyToOne / OneToMany**
Ce type d'association est courant et mappé avec une clé étrangère. Elle est vue des 2 côtés :

ManyToOne : vue du produit
OneToMany : vue de la catégorie

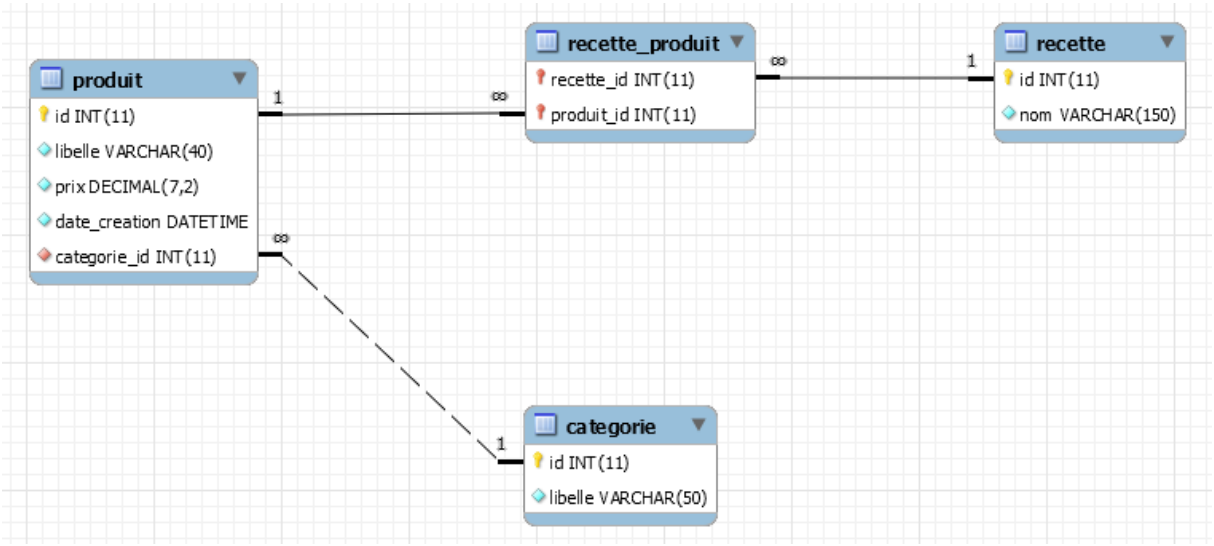
- Sauvegarde association ManyToOne / OneTo Many

```
Nouveau produit enregistré avec l'id : 10 et nouvelle catégorie enregistrée avec id: 1
```

- Création d'une association ManyToMany

La Ferme Maya propose également des recettes dont les ingrédients sont majoritairement issus de sa production.
Dans un premier temps on se contente de gérer le nom de la recette, son identifiant et la liste des produits de la ferme qui sont utilisés dans la recette.
Une recette utilise de 1 à plusieurs produits et un produit est utilisé dans 0 à n recettes.
Pour Doctrine ce cas correspond au type d'association : **ManyToMany**

Schéma relationnel :




```
Relation type? [ManyToOne, OneToMany, ManyToMany, OneToOne]:
> ManyToMany
```

- Sauvegarde association ManyToMany

```
Nouvelle recette enregistrée avec 2 produits, son id est : 1
```

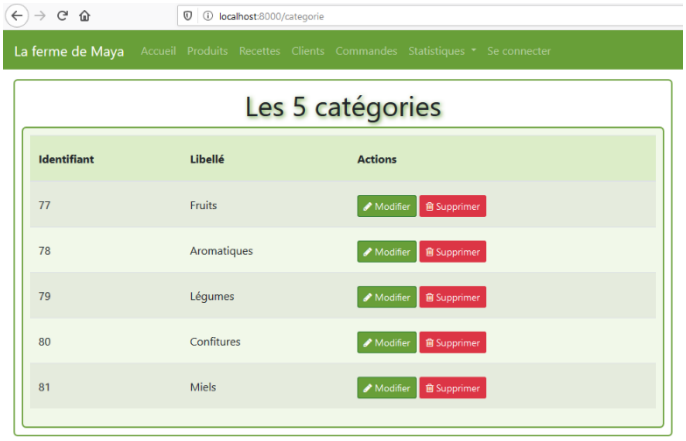
- Création de donnée de test (fixtures)

  Utilisation de **DoctrineFixturesBundle** qui facilite la mise en place des données de tests ou fixtures.

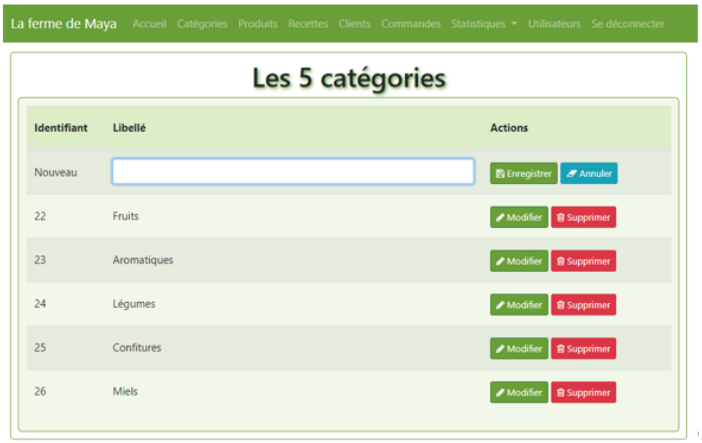
Gestion des catégories de produits avec les formulaires Symfony

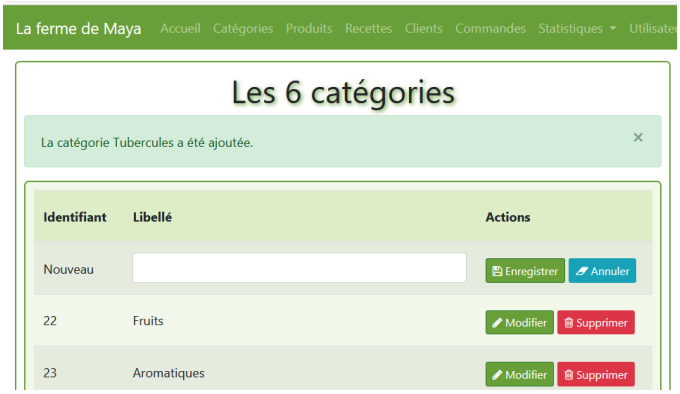
L'entité Categorie et le controleur CategorieController ont déjà été créés.

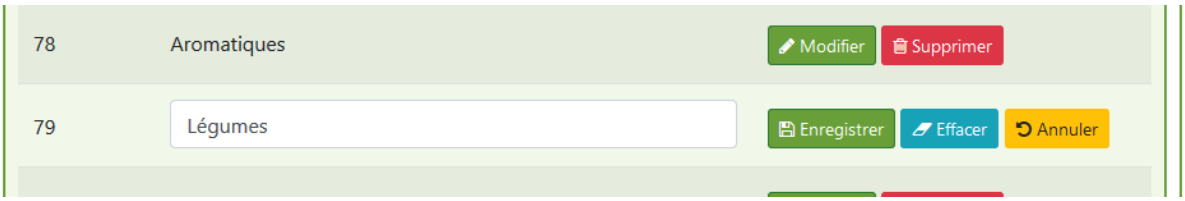
- Affichage des catégories



- Création d'un formulaire pour ajouter/supprimer/modifier une catégorie







- Protection contre les attaques **CSRF**

Les attaques **CSRF** (*Cross-Site Request Forgery*) sont un vecteur d'attaque puissant et souvent ignoré. Symfony est livré avec les outils pour s'en protéger, mais une approche active est nécessaire pour être totalement protégé.

Symfony propose depuis le départ un outil permettant de protéger les formulaires, les fameux CsrfToken. Concrètement, il s'agit d'ajouter à la fin de chaque formulaire un token aléatoire, aussi stocké dans la session utilisateur, qui sera validé lorsque l'utilisateur soumet le formulaire. Ce token n'est ni connu ni devinable par un attaquant, il sera donc incapable de fabriquer une fausse requête.
Sous conditions que le mécanisme soit bien implémenté, cryptographiquement, c'est une mesure efficace, notre formulaire est protégé. Il est activé par défaut sur tous les formulaires Symfony."
Le token est ajouté automatiquement en champ hidden au formulaire.

Affichage du token en mode inspection

```
<tr>
  <form name="categorie" method="post"></form>
  <td class="col-md-1">Nouveau</td>
  <td class="col-md-6">…</td>
  <td class="col-md-3">…</td>
  <input id="categorie__token" type="hidden" name="categorie[_token]"
  value="e494e591e3c3faa3c98495ca5b16.G4YT0Jd_Ty2knMFN21_war6LXgPxbU2…Y.X_xBnME0DVWS8ocs6AupXtm_a2uiBCfmu212TZMm-rJD4UXh3TR7ftatmA">
</tr>
<tr>…</tr>
```

- Validation du formulaire

SPRINT 3

Formulaire Produit, Recherche Multicritères, Pagination

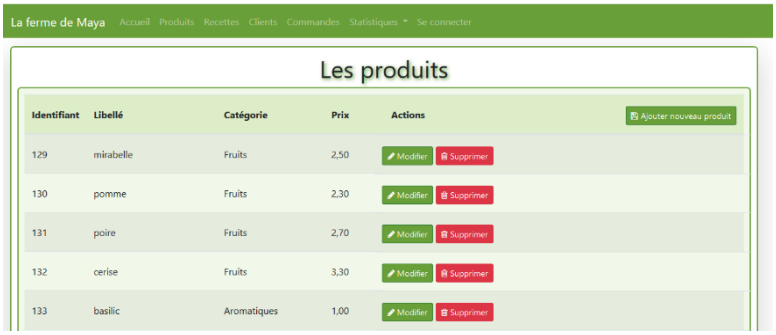
Dans cette mission il s'agit de compléter la gestion des produits en mettant en œuvre les formulaires Symfony. Ce sera l'occasion d'utiliser différents types de champs.

Nous allons mettre en place le filtrage des données via une recherche multicritères sur les produits et ajouter la pagination.

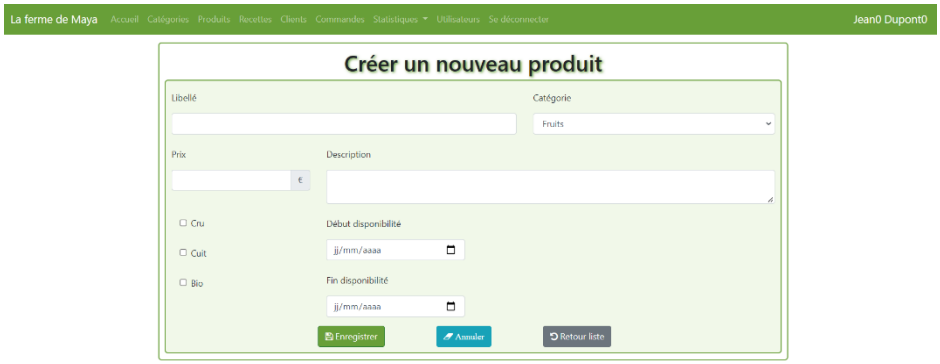
Tâches :

Gestion des produits (opérations CRUD)

- Compléter les informations Produit
- Affichage de la liste des produits



- Ajouter un nouveau produit



- Modifier un produit

- Supprimer un produit

Recherche multi-critères sur les produits à afficher par pages

- Création d'un formulaire de recherche

- Conservation des critères de recherche après opération. Nous allons conserver les critères de sélection dans une variable de session.

Gérer les recettes (opérations CRUD)

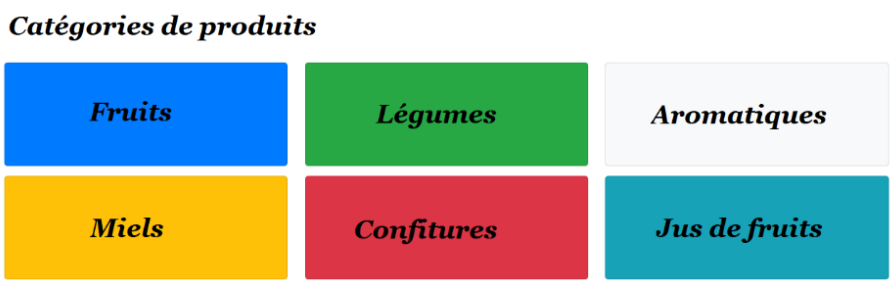
Ajouter la pagination et la recherche multi-critères à la gestion des clients

Ajouter la pagination et la recherche multi-critères à la gestion des fournisseurs

Ajoutez la pagination aux objets de gestion de l'application

Afficher la liste des catégories sous forme de card

- Utilisation des cards de Bootstrap



SPRINT 4

Consommer une API REST , AJAX dans Maya

a) Le but est d'afficher le temps (météo) sur la page d'accueil en consommant une API REST, dans notre cas l'API OpenWeathermap.

Une API (Application Programming Interface) ou interface de programmation est un ensemble normalisé de classes, de méthodes ou de fonctions qui sert de façade par laquelle un logiciel offre des services à d'autres logiciels.

REST est un style d'architecture logicielle pour les systèmes distribués, basé sur HTTP, défini dans la thèse de Roy Fielding dans les années 2000 (l'un des principaux auteurs de la spécification HTTP).

Lorsqu'une application appelle une API REST, on dit qu'elle consomme un service de l'API REST, ou tout simplement ***qu'elle consomme*** *l'API.*

OpenWeatherMap est une API qui fournit régulièrement un énorme volume de données météorologiques. Pour des usages de base, le service est gratuit avec un accès limité.

b) Nous utilisons aussi une requête AJAX pour l'affichage d'une modale qui informe le client des recettes associées aux produits choisis.

L'AJAX permet d'envoyer et récupérer des données d'un serveur de manière asynchrone (en arrière-plan) sans interférer avec l'affichage et le comportement de la page existante.
L'AJAX nous permet de modifier de manière dynamique le contenu d'une page, c'est-à-dire sans qu'il soit nécessaire de recharger l'intégralité de la page.

Tâches :

Consommer l'API REST OpenWeatherMap dans l'application Maya

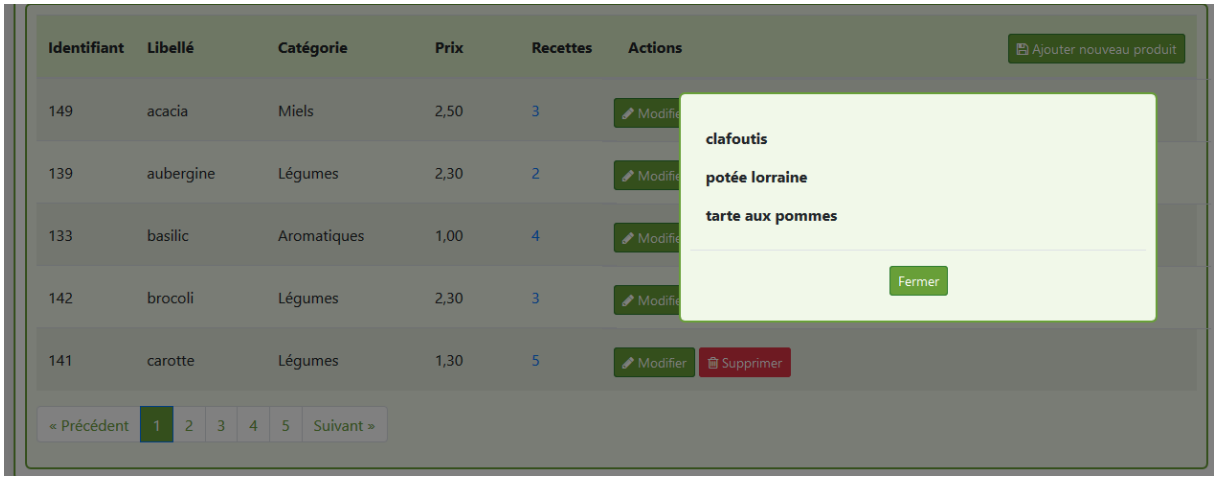
- Intégrer l'affichage du temps dans la page d'accueil de l'application Maya
- Compléter le style
- Adapter le contrôleur de la page d'accueil

Résultat obtenu

Mise en oeuvre d'Ajax dans Maya

- Créer la fenêtre modale
- Mettre en place la requête Ajax

Résultat obtenu



SPRINT 5

Finalisation Version 1.0 de l'application Maya

Opérations attendues :

Ajouter une photo aux catégories, aux produits et aux animaux et gérer les opérations CRUD.

Sur la page d'accueil, afficher les 3 derniers événements passés et les 3 prochains événements afin de les mettre en valeur.

Ajouter une page statistiques qui affiche par catégorie, le nombre de produits, le prix minimum, le prix maximum et le prix moyen des produits.

Ajouter une page qui permet de sélectionner un produit dans une liste déroulante et d'afficher le détail de toutes les recettes utilisant ce produit en les présentant par durée croissante. La liste des recettes sera obtenue via une requête Ajax.

Ajouter la trace des connexions réussies et en échec en utilisant le logger Symfony.

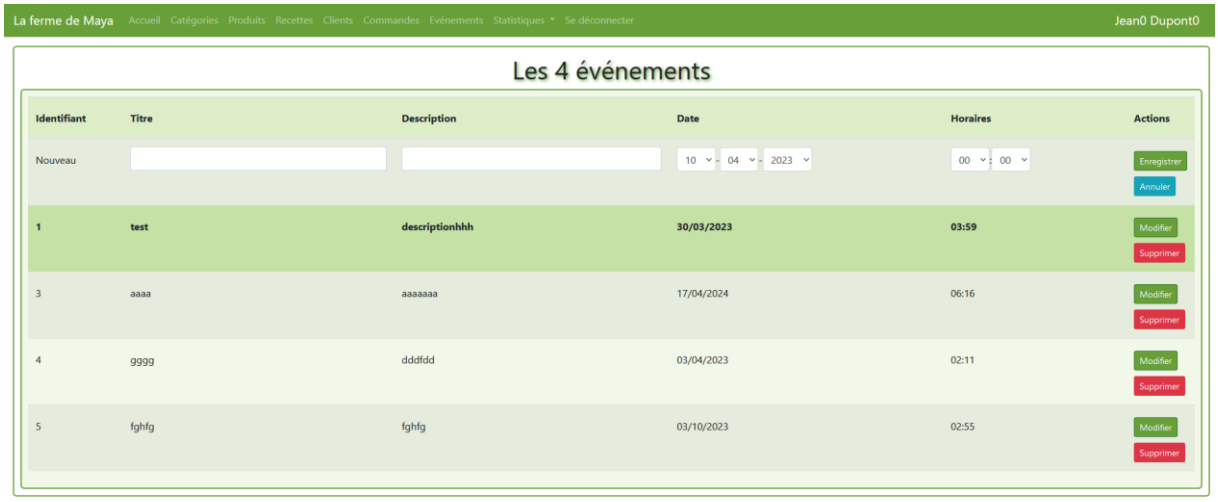
Réalisation personnelle : Gestion des évènements

Durant le sprint 4 du projet Maya, j'ai créé la page concernant la gestion des événements.

J'ai créé, en utilisant l'invite de commandes, l'entité Evenement.php contenant les différentes fonctions de gestion du contenu, le controlleur EvenementController.php permettant d'afficher, de créer, modifier et supprimer des événements.

Pour accéder à la page, j'ai ajouté le lien de redirection en Twig dans le fichier base.html.twig.

Copies d'écran :



templates > evenement > index.html.twig

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

{% block title %}Evénements
{% endblock %}

{% block body %}
    <div class="col-md-12 col-lg-12 contenu-blanc">
        <h1>Les
            <span>
                {{  lesEvenements | length }}
            </span>événements</h1>
        {{ include('messages.html.twig') }}
        <div class="contenu">
            <table class="table table-striped table-advance table-hover">
                <thead>
                    <tr class="bg-entete">
                        <th>Identifiant</th>
                        <th>Titre</th>
                        <th>Description</th>
                        <th>Date</th>
                        <th>Horaires</th>
                        <th>Actions</th>
                    </tr>
                </thead>
                <tbody>

                    <!-- formulaire pour ajouter un nouveau événement-->
                    <tr>
                        {{ form_start(formCreation) }}
                        <td class="col-md-1">Nouveau</td>
                        <td class="col-xl-3 col-md-6 col-lg-3">{{ form_widget(formCreation.titre) }}
                            {{ form_errors(formCreation.titre) }}</td>
                        <td class="col-xl-3 col-md-6 col-lg-3">{{ form_widget(formCreation.description) }}
                            {{ form_errors(formCreation.description) }}</td>
                        <td class="col-xl-3 col-md-6 col-lg-3">{{ form_widget(formCreation.date)  }}
                            {{ form_errors(formCreation.date) }}</td>
                        <td class="col-xl-3 col-md-6 col-lg-3">{{ form_widget(formCreation.horaires) }}
                            {{ form_errors(formCreation.horaires) }}</td>
```

templates > evenement > index.html.twig

```twig
                        <td class="col-xl-3 col-md-3">
                            <button class="btn btn-primary btn-sm" type="submit" formaction="{{ path('evenement_ajouter') }}" title="Enregistrer nouveau événement">
                                <i class="fa fa-save"></i>
                                Enregistrer</button>
                            <button class="btn btn-info btn-sm" type="reset" title="Effacer la saisie">
                                <i class="fa fa-eraser"></i>
                                Annuler
                            </button>
                        </td>
                        {# Ceci va générer le champ CSRF #}
                        {{ form_rest(formCreation) }}
                        {{ form_end(formCreation) }}
                    </tr>

                    {% for key, evenement in lesEvenements %}
                        <tr>
                            {% if evenement.id != idEvenementModif %}
                                <td>{{ evenement.id }}</td>
                                <td>{{ evenement.titre }}</td>
                                <td>{{ evenement.description }}</td>
                                <td>{{ evenement.date | date("d/m/Y")}}</td>
                                <td>{{ evenement.horaires | date("H:i") }}</td>
                                <!-- formulaire pour demander la modification ou la suppression -->
                                <td>
                                    <form>
                                        <button type="submit" class="btn btn-primary btn-sm" formaction="{{ path('evenement_demandermodification', {'id' : evenement.id }) }}" title="Modifier">
                                            <i class="fa fa-pencil"></i>
                                            Modifier</button>
                                        <button class="btn btn-danger btn-sm" type="submit" formaction="{{ path('evenement_supprimer', {'id' : evenement.id }) }}" title="Supprimer" onclick="return confirm('Voulez-vous vraiment supprimer cet événement ?');">
                                            <i class="fa fa-trash-o "></i>
                                            Supprimer</button>
                                        <input
                                        type="hidden" name="_token" value="{{ csrf_token('action-item' ~ evenement.id) }}">
                                    {#  ~     pour concaténer des chaines en twig #}
```

templates > evenement > index.html.twig

```
                                {#  ~     pour concaténer des chaines en twig #}
                                </form>
                            </td>

                        {% else %}
                            <!-- formulaire pour modifier un événement -->
                            {{ form_start(formModification) }}
                            <td>{{ evenement.id }}</td>
                            <td class="col-xl-3 col-md-6 col-lg-3">{{ form_widget(formModification.titre) }}
                                {{ form_errors(formModification.titre) }}</td>
                            <td class="col-xl-3 col-md-6 col-lg-3">{{ form_widget(formModification.description) }}
                                {{ form_errors(formModification.description) }}</td>
                            <td class="col-xl-3 col-md-6 col-lg-3">{{ form_widget(formModification.date)}}
                                {{ form_errors(formModification.date) }}</td>
                            <td class="col-xl-3 col-md-6 col-lg-3">{{ form_widget(formModification.horaires) }}
                                {{ form_errors(formModification.horaires) }}</td>
                            <td class="col-xl-3 col-md-4 col-lg-3">
                                <button class="btn btn-primary btn-sm" type="submit" formaction="{{ path('evenement_modifier', {'id' : evenement.
                                id }) }}" title="Enregistrer">
                                    <i class="fa fa-save"></i>
                                    Enregistrer</button>
                                <button class="btn btn-info btn-sm" type="reset" title="Effacer la saisie">
                                    <i class="fa fa-eraser"></i>
                                    Effacer</button>
                                <button class="btn btn-warning btn-sm" type="submit" formaction="{{ path('evenement') }}" title="Annuler">
                                    <i class="fa fa-undo"></i>
                                    Annuler</button>
                            </td>
                            {# Ceci va générer le champ CSRF #}
                            {{ form_rest(formModification) }}
                            {{ form_end(formModification) }}
                        {% endif %}

                    </tr>
                {% endfor %}
            </tbody>
        </table>
    </div>
    <!--fin div contenu-->
</div>
<!--fin div col-md-12 col-lg-12-->
{% endblock %}
```

Par la suite, j'ai créé une page statistique comptabilisant le nombre total d'événements.

templates > statistiques > index.html.twig

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

{% block title %}Statistiques{% endblock %}

{% block body %}
<div class="conteneur">
    <h2>Statistiques</h2>
    <div>Nombre d'événements : {{nbEvenements}}</div>
</div>
{% endblock %}
```