Rapport Maya Sprint 4 Mission 1, 2 & 3

“Pagination et recherche multi-critères”

Léandro VEYRENC-CORDERO - 19/11/2025

Sommaire :

Rapport Maya Sprint 4 Mission 1, 2 & 3

“Pagination et recherche multi-critères”

Léandro VEYRENC-CORDERO - 19/11/2025

Mission 1 : Gestion des produits

Pour la mission 1 nous devions simplement gérer les produits comme tous les autres objets de l’application.

Pour ce faire, puisque nous avions déjà l’entité produit et son contrôleur nous n'avions pas eu à les créer, mais on a ajouté de nouveaux champs comme cru, cuit, bio, debutDisponibilite, finDisponibilite à l’aide la commande ***php bin/console make:entity***. Pour mettre à jour la base de donnée on a fait une migration mais que des différences avec la commande ***php bin/console doctrine:migrations:diff*** et ***php bin/console doctrine:migrations:migrate***. On a continué par créer le form / type à l’aide de la commande

Rapport Maya Sprint 4 Mission 1, 2 & 3

“Pagination et recherche multi-critères”

Léandro VEYRENC-CORDERO - 19/11/2025

php bin/console make:form

Fichier Maya/src/Form/ProduitType.php

```php
<?php

namespace App\Form;

use App\Entity\Categorie;
use App\Entity\Produit;
use Symfony\Bridge\Doctrine\Form\Type\EntityType;
use Symfony\Component\Form\AbstractType;
use Symfony\Component\Form\FormBuilderInterface;
use Symfony\Component\OptionsResolver\OptionsResolver;
use Symfony\Component\Form\Extension\Core\Type\TextType;
use Symfony\Component\Form\Extension\Core\Type\DateType;
use Symfony\Component\Form\Extension\Core\Type\TextareaType;
use Symfony\Component\Form\Extension\Core\Type\CheckboxType;
use Symfony\Component\Form\Extension\Core\Type\MoneyType;

class ProduitType extends AbstractType
{
    public function buildForm(FormBuilderInterface $builder, array $options): void
    {
        $builder
            ->add('libelle', TextType::class, [
                'label' => 'Libellé'
            ])
            ->add('prix', MoneyType::class, [
                'label' => 'Prix',
                'invalid_message' => 'Nombre attendu'
            ])
//            ->add('dateCreation')      // date de création non saisie car
positionnée à date du jour dans le constructeur Produit
            ->add('description', TextareaType::class, [
                'label' => 'Description',
            ])
            ->add('cru')
            ->add('cuit')
            ->add('bio')
            ->add('debutDisponibilite', DateType::class,  [
                'label' => 'Début disponibilité',
                'widget' => 'single_text',
            ])
            ->add('finDisponibilite',DateType::class, [
                'label' => 'Fin disponibilité',
                'widget' => 'single_text',
            ])
            ->add('categorie', EntityType::class, [
                'label' => 'Catégorie',
                'class' => Categorie::class,
```

```
            ->add('bio')
            ->add('debutDisponibilite', DateType::class,  [
                'label' => 'Début disponibilité',
                'widget' => 'single_text',
            ])
            ->add('finDisponibilite',DateType::class, [
                'label' => 'Fin disponibilité',
                'widget' => 'single_text',
            ])
            ->add('categorie', EntityType::class, [
                'label' => 'Catégorie',
                'class' => Categorie::class,
                'choice_label' => 'libelle',
                'multiple' => false,
                'expanded' => false
            ])
//            ->add('recettes')    // on ne gère pas les recettes dans la gestion
des produits
        ;
    }
```

Ensuite on a adapté le contrôleur comme celui des utilisateurs par exemple et sa vue pour qu’elle ressemble à toutes les pages des gestions.

Nous avons donc une vue globale puis pour modifier, ça en est une pour un seul objet.

Rapport Maya Sprint 4 Mission 1, 2 & 3

“Pagination et recherche multi-critères”

Léandro VEYRENC-CORDERO - 19/11/2025

Rapport Maya Sprint 4 Mission 1, 2 & 3

“Pagination et recherche multi-critères”

Léandro VEYRENC-CORDERO - 19/11/2025

Mission 2 : Recherche multi-critère et session

Recherche Multi-critère

Pour cette deuxième mission, nous avons mis en place une recherche multi-critère.

Pour ce faire, on a créé une nouvelle entité ProduitRecherche afin d’enregistrer les recherches.
Mais cette entité n’a pas été créée avec la commande mais juste en ajoutant un fichier dans le dossier entity.

```php
amespace App\Entity;


lass ProduitRecherche

    /**
     * @var string|null
     */
    private $libelle;
    /**
     * @var float|null
     */
    private $prixMini;
    /**
     * @var float|null
     */
    private $prixMaxi;
    /**
     * @var Categorie|null
     */
    private $categorie;
    /**
     * @var int|null
     */
    private $categorieId;

    /**
     * @return string|null
     */
    public function getLibelle(): ?string
    {
        return $this->libelle;
    }

    /**
     * @param string|null $libelle
     */
    public function setLibelle(?string $libelle): void
    {
```

Ensuite, on a créé le formulaire de cette entité pour inclure un formulaire de recherche dans la vue. Ce form / type a également été rédigé et non créé par la commande.

```php
use App\Entity\ProduitRecherche;
use App\Entity\Categorie;
use Symfony\Component\Form\AbstractType;
use Symfony\Bridge\Doctrine\Form\Type\EntityType;
use Symfony\Component\Form\FormBuilderInterface;
use Symfony\Component\OptionsResolver\OptionsResolver;
use Symfony\Component\Form\Extension\Core\Type\TextType;
use Symfony\Component\Form\Extension\Core\Type\MoneyType;

class ProduitRechercheType extends AbstractType
{
    public function buildForm(FormBuilderInterface $builder, array $options): void
    {
        $builder
            ->add('libelle', TextType::class, [
                'label' => 'Libellé',
                'required' => false,
            ])
            ->add('categorie', EntityType::class, [
                'label' => 'Catégorie',
                'class' => Categorie::class,
                'choice_label' => 'libelle',
                'multiple' => false,
                'expanded' => false,
                'mapped' => true
            ])
            ->add('prixMini', MoneyType::class, [
                'label' => 'Prix minimum',
                'required' => false,
                'invalid_message' => 'Nombre attendu'
            ])
            ->add('prixMaxi', MoneyType::class, [
                'label' => 'Prix maximum',
                'required' => false,
                'invalid_message' => 'Nombre attendu'
            ]);

    }

    public function configureOptions(OptionsResolver $resolver): void
    {
        $resolver->setDefaults([
            'data_class' => ProduitRecherche::class,
        ]);
    }
}
```

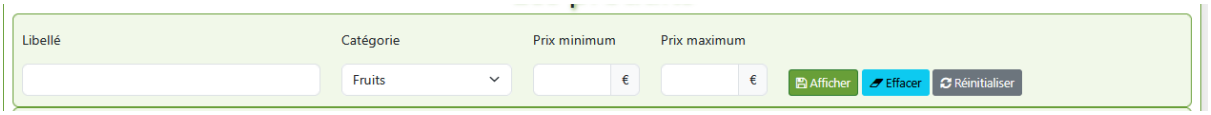
Ensuite, afin d’inclure ce formulaire dans notre vue, on ajoute à la vue des produits une div contenant les labels et zone d’entrée des différents champs ou l’on peut faire la recherche.

Rapport Maya Sprint 4 Mission 1, 2 & 3

“Pagination et recherche multi-critères”

Léandro VEYRENC-CORDERO - 19/11/2025

```
{# formulaire de recherche des produits #}
{{ form_start(formRecherche, {'method': 'get', 'action': path('app_produit')}) }}
    <div class="container-fluid contenu"> <div class="row">
        <div class="col-md-8 row font-weight-bold">
            <div class="col-md-5">{{ form_label(formRecherche.libelle) }}</div>
            <div class="col-md-3">{{ form_label(formRecherche.categorie) }}</div>
            <div class="col-md-2">{{ form_label(formRecherche.prixMini) }}</div>
            <div class="col-md-2">{{ form_label(formRecherche.prixMaxi) }}</div>
        </div>
    </div>

    <div class="row">

        <div class="col-md-8 row">
            <div class="col-md-5">{{ form_widget(formRecherche.libelle) }}</div>
            <div class="col-md-3">{{ form_widget(formRecherche.categorie) }}</div>
            <div class="col-md-2">{{ form_widget(formRecherche.prixMini) }}</div>
            <div class="col-md-2">{{ form_widget(formRecherche.prixMaxi) }}</div>
        </div>

        <div
            class="col-md-4">
            <button class="btn btn-primary btn-sm" type="submit" title="Afficher les produits">
                <i class="fa fa-save"></i>
                Afficher</button>
            <button class="btn btn-info btn-sm" type="reset" title="Effacer la saisie">
                <i class="fa fa-eraser"></i>
                Effacer</button>
            {# lien qui réinitialise la recherche : renvoie à la route app_produit sans query string #}
            <a href="{{ path('app_produit_reinitialiser') }}"  class="btn btn-secondary btn-sm" title="Réinitialis
                <i class="fa fa-refresh"></i>
                Réinitialiser</a>
        </div>
    </div>
</div>
{{ form_end(formRecherche) }}
```



Rapport Maya Sprint 4 Mission 1, 2 & 3

“Pagination et recherche multi-critères”

Léandro VEYRENC-CORDERO - 19/11/2025

Pour que le formulaire fonctionne correctement, on vient adapter le contrôleur pour qu’il agissent avec le repository avec des fonctions de recherche qu’on va créer juste après.

```php
use App\Entity\ProduitRecherche;
use App\Form\ProduitRechercheType;


class ProduitController extends AbstractController
{
    #[Route('/produit', name: 'app_produit')]
    public function index(Request $request, ProduitRepository $repository):
Response
    {
        // créer l'objet et le formulaire de recherche
        $produitRecherche = new ProduitRecherche();
        // form en GET : Symfony lira les paramètres depuis $request->query
        $formRecherche = $this->createForm(ProduitRechercheType::class,
$produitRecherche, [
            'method' => 'GET',
            // optionnel : désactiver CSRF pour formulaires GET
            // 'csrf_protection' => false,
        ]);
        $formRecherche->handleRequest($request);
        if ($formRecherche->isSubmitted() && $formRecherche->isValid()) {
            $produitRecherche = $formRecherche->getData();
            // cherche les produits correspondant aux critères, triés par libellé
            // requête construite dynamiquement alors il est plus simple
d'utiliser le querybuilder
            $lesProduits = $repository->findAllByCriteria($produitRecherche);

        } else {
            $lesProduits = $repository->findAllOrderByLibelle();
        }

        return $this->render('produit/index.html.twig', [
            'formRecherche' => $formRecherche,
            'lesProduits' => $lesProduits,
        ]);
    }
```

Ensuite, dans le fichier ProduitRepository.php, on y ajoute deux fonctions, findAllbyCriteria et findAllOrderByLibelle qui nous permettent d’affiner les résultats présent dans la base de donnée en fonction des différents critères de recherche que nous saisissons.

```php
/**
 * @return Produit[]
 */
public function findAllByCriteria(ProduitRecherche $produitRecherche): Array
{
    // le "p" est un alias utilisé dans la requête
    $qb = $this->createQueryBuilder('p')
        ->orderBy('p.libelle', 'ASC');

    if ($produitRecherche->getLibelle()) {
        $qb->andWhere('p.libelle LIKE :libelle')
            ->setParameter('libelle', $produitRecherche->getLibelle().'%');
    }

    if ($produitRecherche->getPrixMini()) {
        $qb->andWhere('p.prix >= :prixMini')
            ->setParameter('prixMini', $produitRecherche->getPrixMini());
    }

    if ($produitRecherche->getPrixMaxi()) {
        $qb->andWhere('p.prix < :prixMaxi')
            ->setParameter('prixMaxi', $produitRecherche->getPrixMaxi());
    }

    $query = $qb->getQuery();
    return $query->execute();
}
```

```php
/**
 * @return Produit[]
 */
public function findAllOrderByLibelle(): array
{
    $entityManager = $this->getEntityManager();
    $query = $entityManager->createQuery(
        'SELECT p
        FROM App\Entity\Produit p
        ORDER BY p.libelle ASC'
    );

    // retourne un tableau d'objets de type Produit
    return $query->getResult();
}
```

Rapport Maya Sprint 4 Mission 1, 2 & 3

“Pagination et recherche multi-critères”

Léandro VEYRENC-CORDERO - 19/11/2025

Session

Afin de garder en mémoire la précédente recherche, on va utiliser les sessions.
Pour ce faire dans le contrôleur des produits, on inclut un package de Symfony nommé SessionInterface.
use Symfony\Component\HttpFoundation\Session\SessionInterface;

Et ainsi pour sauvegarder la recherche dans la session, on modifie le contrôleur pour soit lire les informations ou soit les écrire.

```
        $lesProduits = $repository->findAllByCriteria($produitRecherche);
        // mémoriser les critères de sélection dans une variable de session
        $session->set('ProduitCriteres', $produitRecherche);
    } else {
        // lire les produits
        if ($session->has("ProduitCriteres")) {
            // récupérer les critères en session
            $produitRecherche = $session->get("ProduitCriteres");
            $lesProduits = $repository->findAllByCriteria($produitRecherche);
            $formRecherche = $this->createForm(ProduitRechercheType::class,
oduitRecherche);
            // injecter les critères en session dans le formulaire de recherche
            $formRecherche->setData($produitRecherche);
        } else {
            $lesProduits = $repository->findAllOrderByLibelle();
        }
    }
```

Ensuite, pour que le bouton "Réinitialiser" fonctionne et aboutisse à l'affichage de la liste complète, il faut supprimer la variable de session lors de la réinitialisation :

```
    #[Route('/produit/reinitialiser', name: 'app_produit_reinitialiser', methods:
['GET'])]
    public function reinitialiser(Request $request): Response
    {
        // supprimer les critères de recherche en session
        $session = $request->getSession();
        if ($session->has('ProduitCriteres')) {
            $session->remove('ProduitCriteres');
        }

        return $this->redirectToRoute('app_produit');
    }
```

Et on ajoute le chemin de la route dans le bouton réinitialiser :

```
{# lien qui réinitialise la recherche : renvoie à la route app_produit sans query string #}
<a href="{{ path('app_produit_reinitialiser') }}"  class="btn btn-secondary btn-sm" title="Réinitialiser la liste">
```

Rapport Maya Sprint 4 Mission 1, 2 & 3

“Pagination et recherche multi-critères”

Léandro VEYRENC-CORDERO - 19/11/2025

Mission 3 : Pagination

Premièrement, pour mettre en place une pagination, nous allons installer un bundle de knplabs nommé knp-paginator-bundle à l’aide de composer, notre gestionnaire de paquet.
composer require knplabs/knp-paginator-bundle

Ensuite, on vient l’inclure dans notre contrôleur afin qu’il soit utilisable dans la fonction index.

```
use Knp\Component\Pager\PaginatorInterface;
```

```
    public function index(Request $request, ProduitRepository $repository,
SessionInterface $session, PaginatorInterface $paginator): Response
    {
```

Pour continuer, au départ nous envoyons simplement un tableau contenant toute les valeurs des produits de la base de donnée mais à l’aide de la fonction ***paginate*** du paginator, on lui donne en paramètre toute nos entrées, le numéro de page ou si précisé aucun alors la page 1 et le nombre d’entré par page.

```
$lesProduits = $paginator->paginate(
    $repository->findAllByCriteria($produitRecherche),
    $request->query->getint('page', 1),
    5
);
```

```
$prodRech = new ProduitRecherche();
$lesProduits = $paginator->paginate(
    $repository->findAllOrderByLibelle($prodRech),
    $request->query->getint('page', 1),
    5
);
```

Ensuite on modifie le Repository pour qu’il retourne un objet de classe Query car la fonction ***paginate*** demande en premier argument un Query et non un tableau.

```php
use Doctrine\ORM\Query;
use Doctrine\ORM\QueryBuilder;


. . .

    /**
     * @return Query
     */
    public function findAllByCriteria(ProduitRecherche $produitRecherche): Query
    {
        // le "p" est un alias utilisé dans la requête
        $qb = $this->createQueryBuilder('p')
            ->orderBy('p.libelle', 'ASC');

        if ($produitRecherche->getLibelle()) {
            $qb->andWhere('p.libelle LIKE :libelle')
                ->setParameter('libelle', $produitRecherche->getLibelle().'%');
        }

        if ($produitRecherche->getPrixMini()) {
            $qb->andWhere('p.prix >= :prixMini')
                ->setParameter('prixMini', $produitRecherche->getPrixMini());
        }

        if ($produitRecherche->getPrixMaxi()) {
            $qb->andWhere('p.prix < :prixMaxi')
                ->setParameter('prixMaxi', $produitRecherche->getPrixMaxi());
        }

        return $qb->getQuery();
//         $query = $qb->getQuery();
//         return $query->execute();
```

```php
    /**
     * @return Query
     */
    public function findAllOrderByLibelle(): Query
    {
        $entityManager = $this->getEntityManager();
        $query = $entityManager->createQuery(
            'SELECT p
            FROM App\Entity\Produit p
            ORDER BY p.libelle ASC'
        );

        return $query;
    }

. . .
```

Rapport Maya Sprint 4 Mission 1, 2 & 3

“Pagination et recherche multi-critères”

Léandro VEYRENC-CORDERO - 19/11/2025

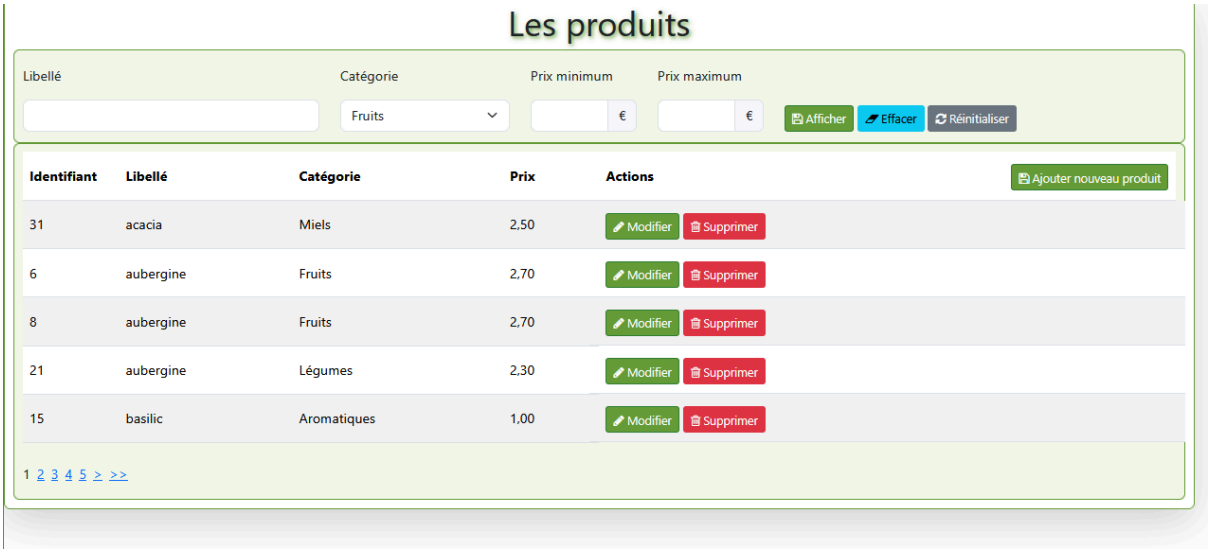
Puis pour afficher la pagination, on ajoute un bout de code à la fin du tableau.

```
{# display navigation #}
<div class="navigation">
    {{ knp_pagination_render(lesProduits) }}
</div>
```

Pour que cette pagination soit un peu plus esthétique, on ajoute du CSS dans notre app.css

```
/* espacer les numéros de page (pagination)
   - on utilise display:flex / gap pour être robuste face aux styles bootstrap
   - on cible .pagination (généré par knp_pagination) et les liens .page-link */
.pagination {
  display: inline-flex; /* place les éléments sur une ligne et permet gap */
  gap: 8px;             /* espace horizontal entre éléments */
  align-items: center;
  padding-left: 0;      /* éviter le padding de list-style par défaut */
  list-style: none;
}
.pagination .page-item {
  display: inline-block; /* backup si besoin */
  margin: 0;             /* gap gère l'espacement, annule margin éventuel */
}
.pagination .page-item .page-link {
  /* petit padding pour garder un toucher confortable */
  padding: .375rem .6rem;
}
```

Voici le résultat à la fin de ces 3 missions :



Rapport Maya Sprint 4 Mission 1, 2 & 3

“Pagination et recherche multi-critères”

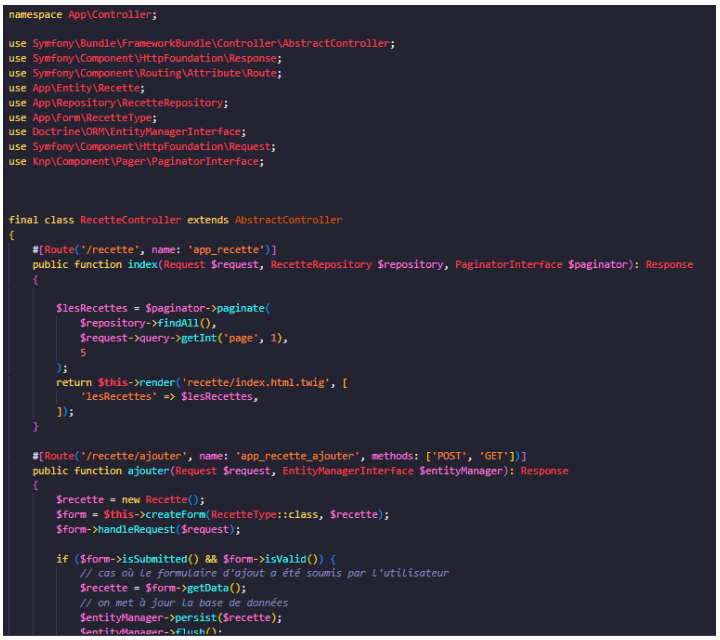
Léandro VEYRENC-CORDERO - 19/11/2025

Mission 4 : Développez les autres fonctionnalités du sprint 4

Gestion des recettes

Pour gérer les recettes, on a d’abord ajouté de nouveaux champs comme la description, le temps de cuisson, le temps de préparation et les ingrédients.
php bin/console make:entity

Ensuite j’ai adapté le contrôleur pour agir sur les recettes avec les fonctions CRUD



```php
namespace App\Controller;

use Symfony\Bundle\FrameworkBundle\Controller\AbstractController;
use Symfony\Component\HttpFoundation\Response;
use Symfony\Component\Routing\Attribute\Route;
use App\Entity\Recette;
use App\Repository\RecetteRepository;
use App\Form\RecetteType;
use Doctrine\ORM\EntityManagerInterface;
use Symfony\Component\HttpFoundation\Request;
use Knp\Component\Pager\PaginatorInterface;


final class RecetteController extends AbstractController
{
    #[Route('/recette', name: 'app_recette')]
    public function index(Request $request, RecetteRepository $repository, PaginatorInterface $paginator): Response
    {

        $lesRecettes = $paginator->paginate(
            $repository->findAll(),
            $request->query->getInt('page', 1),
            5
        );
        return $this->render('recette/index.html.twig', [
            'lesRecettes' => $lesRecettes,
        ]);
    }

    #[Route('/recette/ajouter', name: 'app_recette_ajouter', methods: ['POST', 'GET'])]
    public function ajouter(Request $request, EntityManagerInterface $entityManager): Response
    {
        $recette = new Recette();
        $form = $this->createForm(RecetteType::class, $recette);
        $form->handleRequest($request);

        if ($form->isSubmitted() && $form->isValid()) {
            // cas où le formulaire d'ajout a été soumis par l'utilisateur
            $recette = $form->getData();
            // on met à jour la base de données
            $entityManager->persist($recette);
            $entityManager->flush();
```

```php
#[Route('/recette/modifier/{id<\d+>}', name: 'app_recette_modifier', methods: ['POST', 'GET'])]
public function modifier(Recette $recette, Request $request, EntityManagerInterface $entityManager): Response
{
    $form = $this->createForm(RecetteType::class, $recette);
    $form->handleRequest($request);

    if ($form->isSubmitted() && $form->isValid()) {
        // cas où le formulaire  a été soumis par l'utilisateur et est valide
        //pas besoin de "persister" l'entité : en effet, l'objet a déjà été retrouvé à partir de Doctrine ORM.
        $entityManager->flush();
        $this->addFlash(
            'success',
            'La recette ' . $recette->getNom() . ' a été modifié.'
        );

        return $this->redirectToRoute('app_recette');
    }
    // cas où l'utilisateur a demandé la modification, on affiche le formulaire pour la modification
    return $this->render('recette/modifier.html.twig', [
        'form' => $form,
    ]);
}

#[Route('/recette/supprimer/{id<\d+>}', name: 'app_recette_supprimer', methods: ['GET'])]
public function supprimer(Recette $recette, Request $request, EntityManagerInterface $entityManager): Response
{
    if ($this->isCsrfTokenValid('action-item' . $recette->getId(), $request->get('_token'))) {
        $entityManager->remove($recette);
        $entityManager->flush();
        $this->addFlash(
            'success',
            'La recette ' . $recette->getNom() . ' a été supprimé.'
        );
    }
    return $this->redirectToRoute('app_recette');
}
```

J’ai ensuite mis à jour le formulaire type des recettes avec les nouveaux champs en faisant
php bin/console make:form

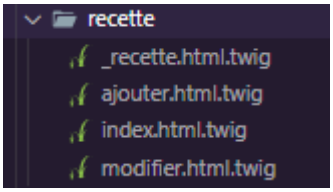
Rapport Maya Sprint 4 Mission 1, 2 & 3

“Pagination et recherche multi-critères”

Léandro VEYRENC-CORDERO - 19/11/2025

```php
<?php

namespace App\Form;

use App\Entity\Produit;
use App\Entity\Recette;
use Symfony\Bridge\Doctrine\Form\Type\EntityType;
use Symfony\Component\Form\AbstractType;
use Symfony\Component\Form\FormBuilderInterface;
use Symfony\Component\OptionsResolver\OptionsResolver;

class RecetteType extends AbstractType
{
    public function buildForm(FormBuilderInterface $builder, array $options): void
    {
        $builder
            ->add('nom')
            ->add('description')
            ->add('tempsPreparation')
            ->add('tempsCuisson')
            ->add('ingredient')
            ->add('produits', EntityType::class, [
                'class' => Produit::class,
                'choice_label' => 'libelle',
                'multiple' => true,
            ])
        ;
    }

    public function configureOptions(OptionsResolver $resolver): void
    {
        $resolver->setDefaults([
            'data_class' => Recette::class,
        ]);
    }
}
```

J’ai ensuite créé une vue comme les produits avec une vue globale pour toutes les recettes en liste et lors de la modification, seule la recette concernée est affichée.



Rapport Maya Sprint 4 Mission 1, 2 & 3

“Pagination et recherche multi-critères”

Léandro VEYRENC-CORDERO - 19/11/2025

Affichage des catégories sous forme de card

Nous souhaitons que les utilisateurs voient les catégories sous forme de cartes cliquable redirigeant vers les produits de cette catégories et que les administrateurs gardent leur vue en liste avec les fonctions crud.

On va donc agir sur la vue à l’accueil en affichant pour chaque entrée un carte avec son libellé, une image si elle en possèdent une, et de plus la carte est cliquable car elle est du point de vue logicielle un lien qui redirige vers app_produit où la catégorie de recherche est la catégorie cliqué sous la forme http://localhost:8000/produit?categorie=1.

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

{% block title %}Accueil
{% endblock %}

{% block body %}
    <div class="col-md-10 contenu-blanc">

        <h1>
            Accueil
        </h1>
        {{ include('messages.html.twig') }}
        <div class="contenu">
            <div class="row gap-1 row-gap-5 justify-content-center ">
                {% for key, categorie in lesCategories %}
                    <div class="col-md-3">
                        <a href="{{ path('app_produit', { categorie: categorie.id }) }}" class="text-decoration-none text-dark">
                            <div class="card">
                                {% if categorie.imgFilename == null %}
                                    <div class="card-header"></div>
                                {% else %}
                                    <img src="{{ asset('uploads/categorieIMG/' ~ categorie.imgFilename) }}" class="card-img-top" alt="Card image cap">
                                {% endif %}
                                <div class="card-body">
                                    <h5 class="card-title">{{ categorie.libelle }}</h5>
                                </div>
                            </div>
                        </a>
                    </div>
                {% endfor %}
            </div>
        </div>
    </div>
    <!--fin div col-md-6-->
{% endblock %}
```

Puisque nous utilisons la vue de accueil avec des valeurs non fixe, on doit mettre à jour le contrôleur.

Rapport Maya Sprint 4 Mission 1, 2 & 3

“Pagination et recherche multi-critères”

Léandro VEYRENC-CORDERO - 19/11/2025

```php
<?php

namespace App\Controller;

use App\Repository\CategorieRepository;
use Symfony\Bundle\FrameworkBundle\Controller\AbstractController;
use Symfony\Component\HttpFoundation\Response;
use Symfony\Component\Routing\Attribute\Route;

final class AccueilController extends AbstractController
{
    #[Route('/accueil', name: 'app_accueil')]
    public function index(CategorieRepository $repository): Response
    {
        // lire les catégories
        $lesCategories = $repository->findAll();
        return $this->render('accueil/index.html.twig', [
            'lesCategories' => $lesCategories,
        ]);
    }
}
```

Pour récupérer l’Id de la catégorie présent dans l’URL, utilise request et pour que la recherche soit prise en compte on modifie la session en injectant tout mais en changeant la catégorieId.

```php
        if ($request->query->get('categorie')) {

            // Charger l'entité catégorie
            $categorie = $entityManager->getRepository(Categorie::class)
                ->find($request->query->get('categorie'));

            if ($categorie) {
                $produitRecherche->setCategorie($categorie);
                $formRecherche->get('categorie')->setData($categorie);
            }
            $data = $session->get("ProduitCriteres");
            $session->set('ProduitCriteres', [
                'libelle' => $data['libelle'],
                'prixMini' => $data['prixMini'],
                'prixMaxi' => $data['prixMaxi'],
                'categorieId' => $request->query->get('categorie')
            ]);
        }
```