

Projet SymRecipe*

P2 : CRUD des ingrédients

12 mai 2024

1 Liste des ingrédients

1.1 Contrôleur

1.1.1 A votre tour



Exécutez la commande :

```
php bin/console make:controller IngredientController
```

```
Windows PowerShell
PS C:\Users\alexa\symfony-projects\symrecipe> symfony serve -d

[WARNING] run "symfony.exe server:ca:install" first if you want to run the web server with TLS support, or use "--p12" or "--no-tls" to avoid this warning

[WARNING] The local web server is optimized for local development and MUST never be used in a production setup.

[OK] Web server listening
The Web server is using PHP CGI 8.2.11
http://127.0.0.1:8000

Stream the logs via symfony.exe server:log
PS C:\Users\alexa\symfony-projects\symrecipe> php bin/console make:controller IngredientController
created: src/Controller/IngredientController.php
created: templates/ingredient/index.html.twig

Success!

Next: Open your new controller class and add some pages!
PS C:\Users\alexa\symfony-projects\symrecipe> |
```

Puis vérifiez que les deux nouveaux fichiers suivants ont bien été créés :

- `src/Controller/IngredientController.php` :

*Emilien Gantois - <https://www.youtube.com/@developpeur.muscle>

```

1 <?php
2
3 namespace App\Controller;
4
5 use
6
7
8
9 class IngredientController extends AbstractController
10 {
11     new *
12     #[Route('/ingredient', name: 'app_ingredient')]
13     public function index(): Response
14     {
15         return $this->render(view: 'ingredient/index.html.twig', [
16             'controller_name' => 'IngredientController',
17         ]);
18     }
19 }

```

- templates/ingredient/index.html.twig :

```

1 {% extends "base.html.twig" %}
2
3 {% block title %}Hello IngredientController!{% endblock %}
4
5 {% block body %}
6
7     <style>
8         .example-wrapper { margin: 1em auto; max-width: 800px; width: 95%; font: 18px/1.5 sans-serif; }
9         .example-wrapper code { background: #F5F5F5; padding: 2px 6px; }
10     </style>
11
12     <div class="example-wrapper">
13         <h1>Hello {{ controller_name }}! </h1>
14
15         This friendly message is coming from:
16
17         <ul>
18             <li>Your controller at <code>C:/Users/alexa/symfony-projects/symrecipe/src/Controller/IngredientController
19             <li>Your template at <code>C:/Users/alexa/symfony-projects/symrecipe/templates/ingredient/index.html.twig<
20         </ul>
21     </div>
22 {% endblock %}

```

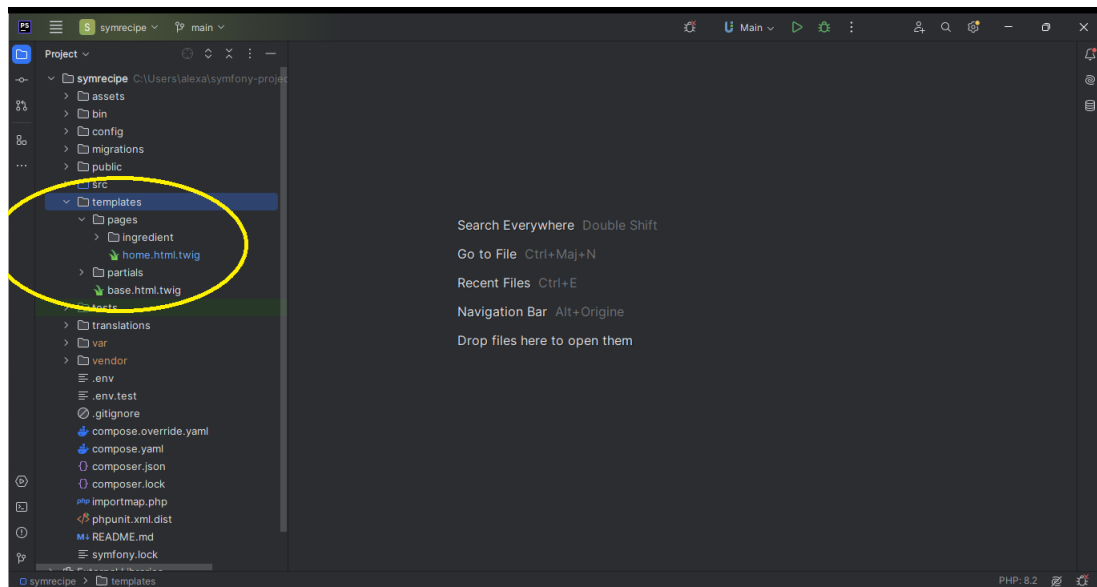
1.2 Refactorisation

1.2.1 A votre tour



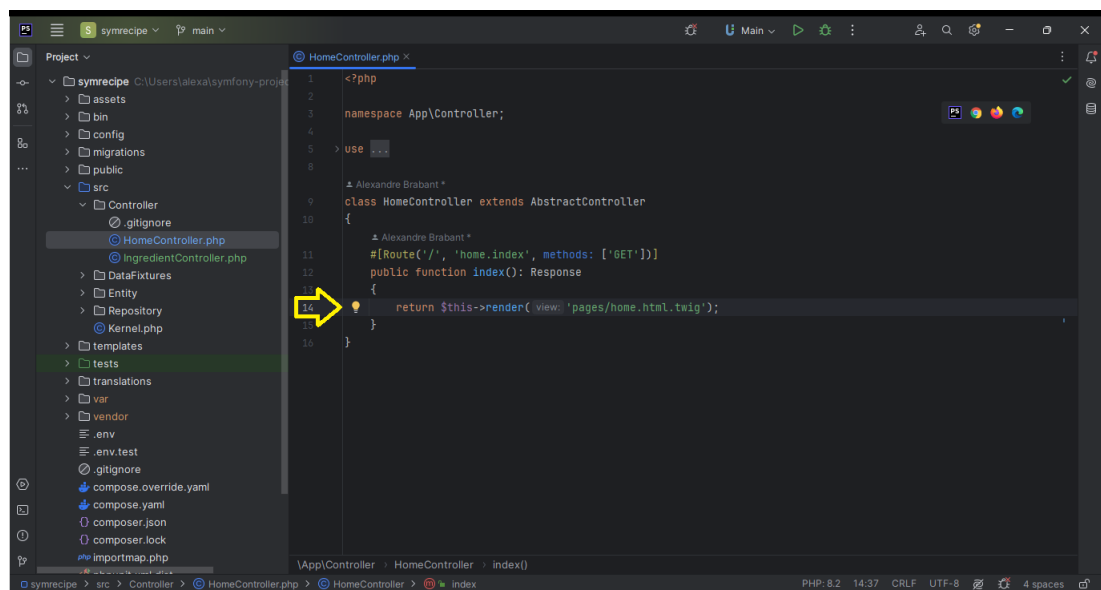
Placez vous dans le répertoire templates de votre projet symrecipe et :

1. commencez par créer un dossier pages ;
2. puis déplacez le répertoire ingredient et le fichier home.html.twig dans ce nouveau dossier pages.

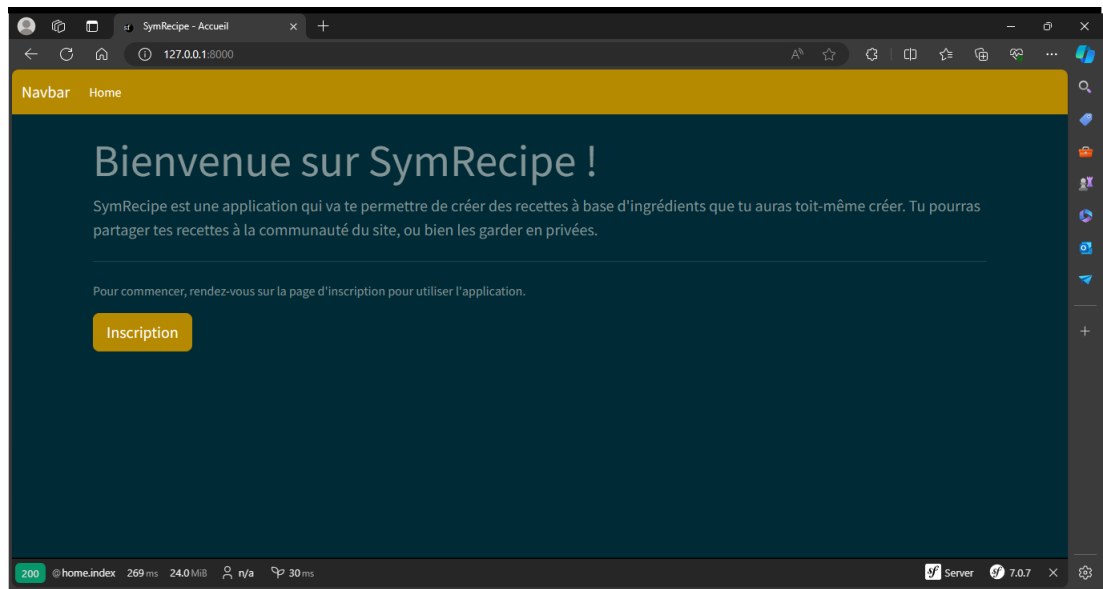


3. Enfin, refactorisez le code :

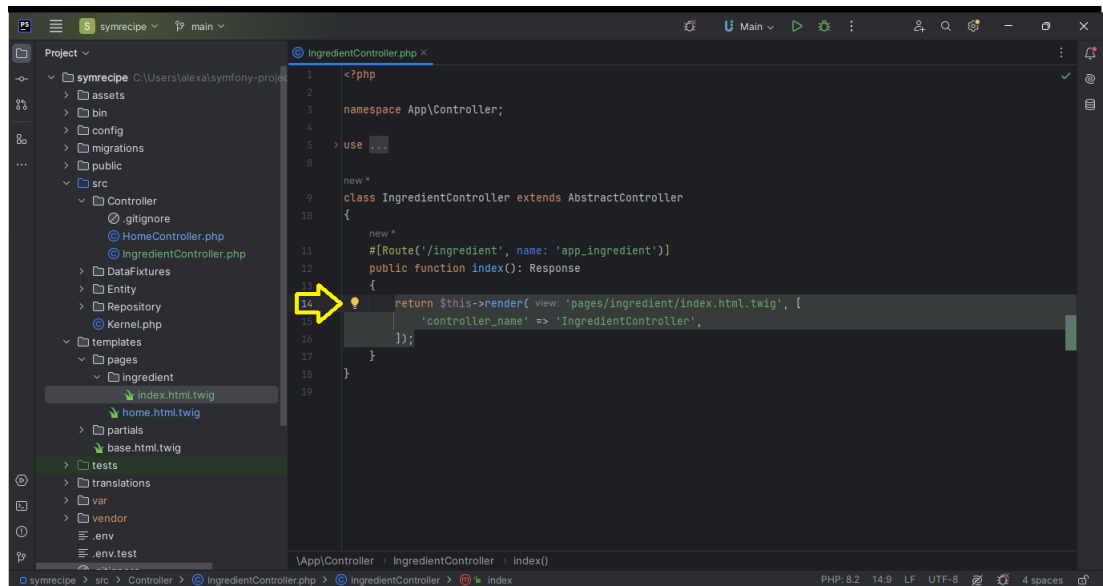
- dans le fichier `src/Controller/HomeController.php` :



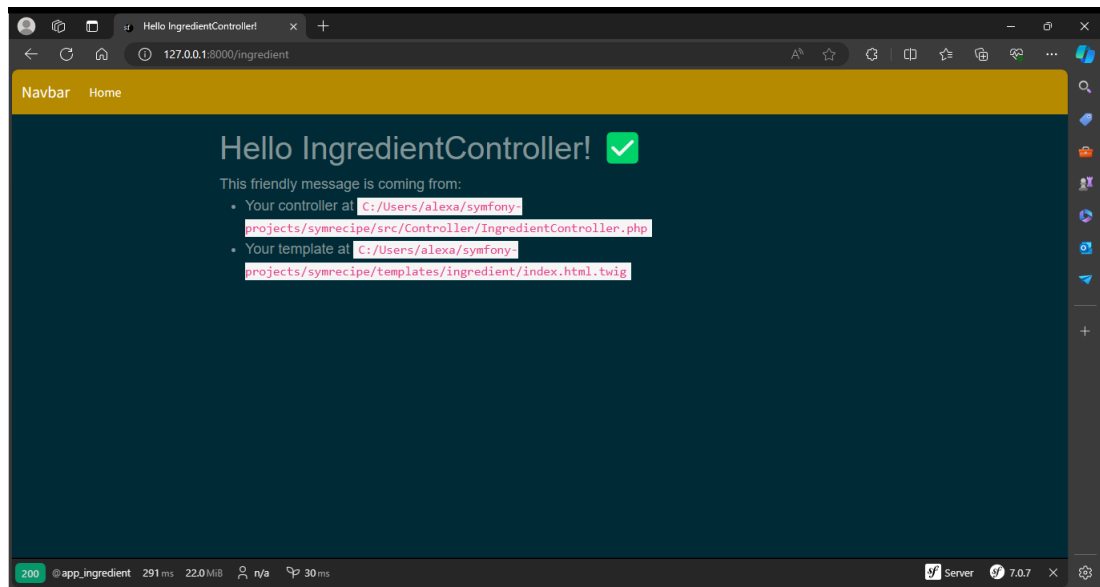
Vérifiez en rechargeant la page d'accueil / :



- dans le fichier `src/Controller/IngredientController.php` :



Vérifiez en rechargeant la page `/ingredient` :

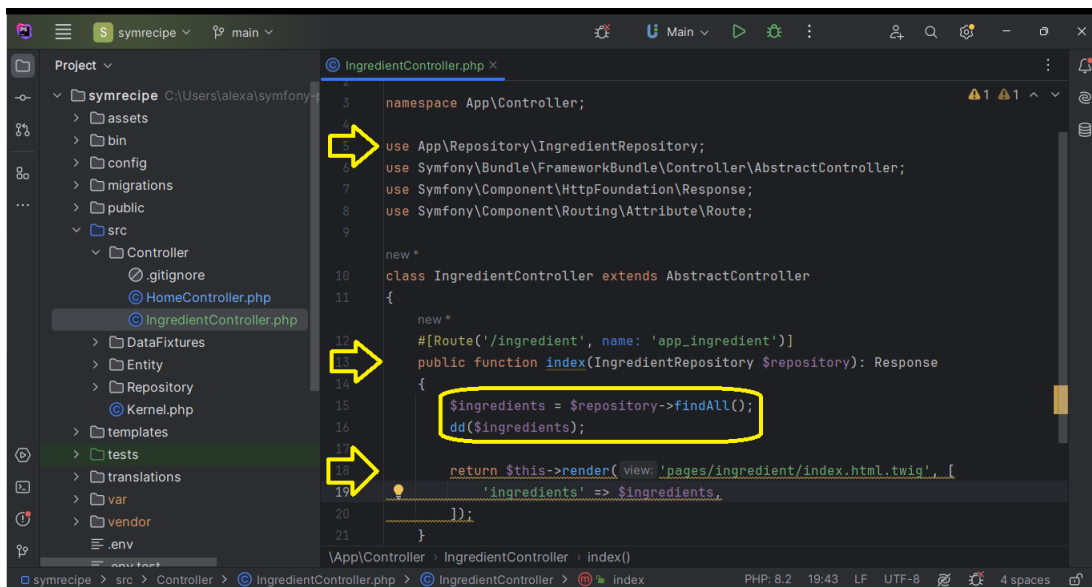


1.3 Repository

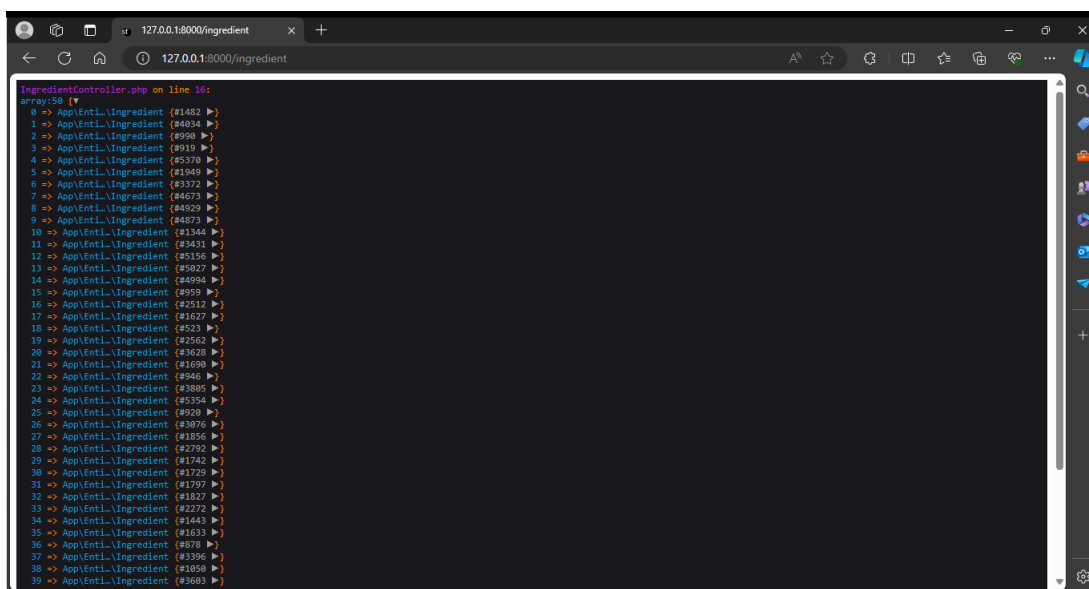
1.3.1 A votre tour



Modifiez le contenu du fichier `src/Controller/IngredientController.php` :



Puis rechargez la page `/ingredient` :

A screenshot of a web browser window displaying a list of ingredients. The browser's address bar shows the URL '127.0.0.1:8000/ingredient'. The page content is a dark-themed interface with a list of ingredients, each preceded by a number and an arrow pointing right. The ingredients are listed in a single column, with their names and IDs visible. The list starts with 'App\Entil\Ingredient (#1482)' and continues down to 'App\Entil\Ingredient (#3603)'. The browser's developer tools are open, showing the 'IngredientController.php on line 16' and a list of ingredients in a dark-themed interface.

Enfin, dans le fichier `src/Controller/IngredientController.php`, supprimez l'instruction `dd($ingredients);`

Injection de dépendances

Dans la signature de la méthode `index` du contrôleur `IngredientController`, en insérant le paramètre `IngredientRepository $repository`, on a effectué une injection de dépendances.

L'injection de dépendances (*dependency injection* en anglais) est un mécanisme qui permet d'implémenter le principe de l'inversion de contrôle.

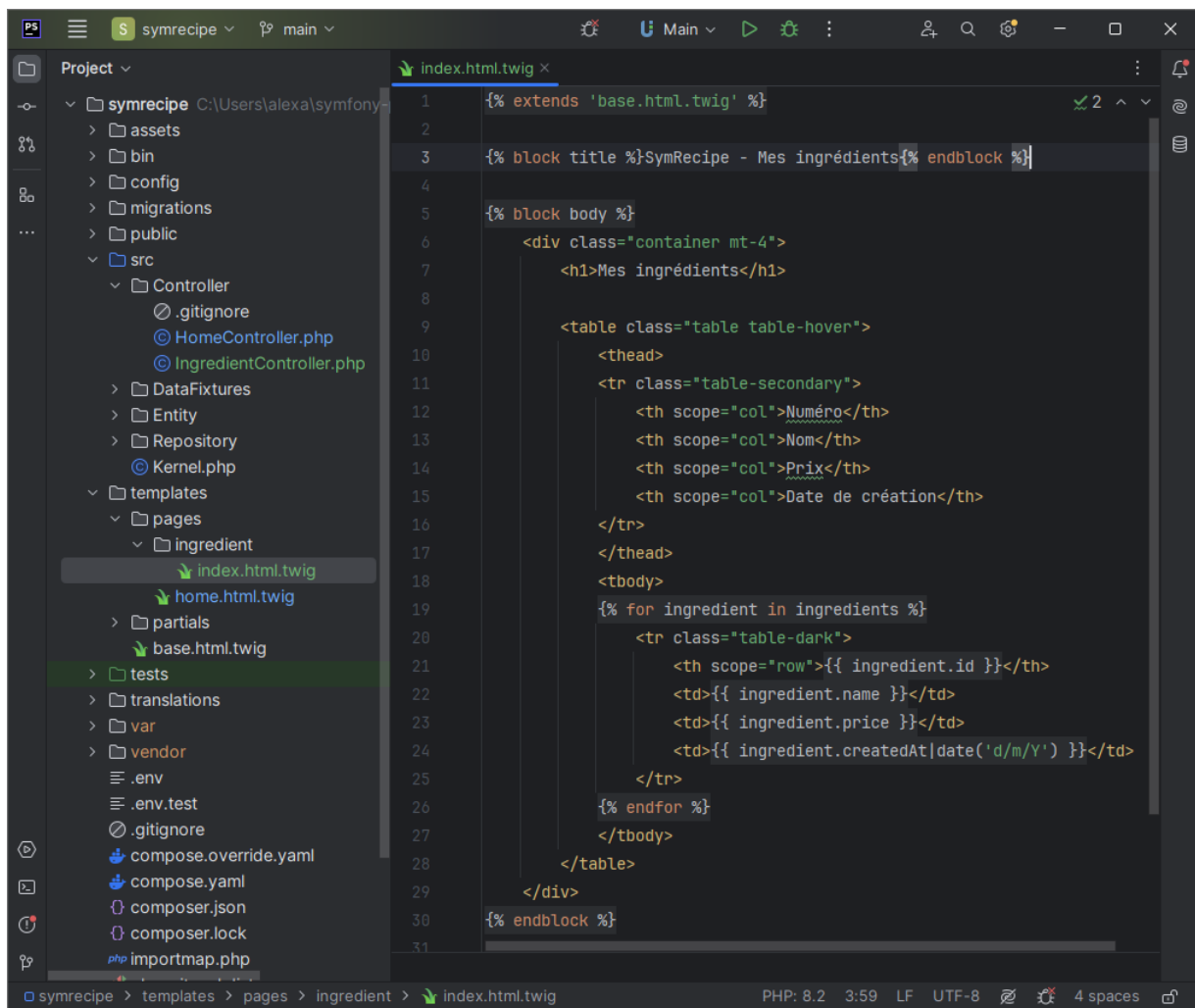
Il consiste à créer dynamiquement (injecter) les dépendances entre les différents objets en s'appuyant sur une description (fichier de configuration ou métadonnées) ou de manière programmatique. Ainsi les dépendances entre composants logiciels ne sont plus exprimées dans le code de manière statique mais déterminées dynamiquement à l'exécution. Source [Wikipedia](#)

1.4 Twig et Bootswatch

1.4.1 A votre tour

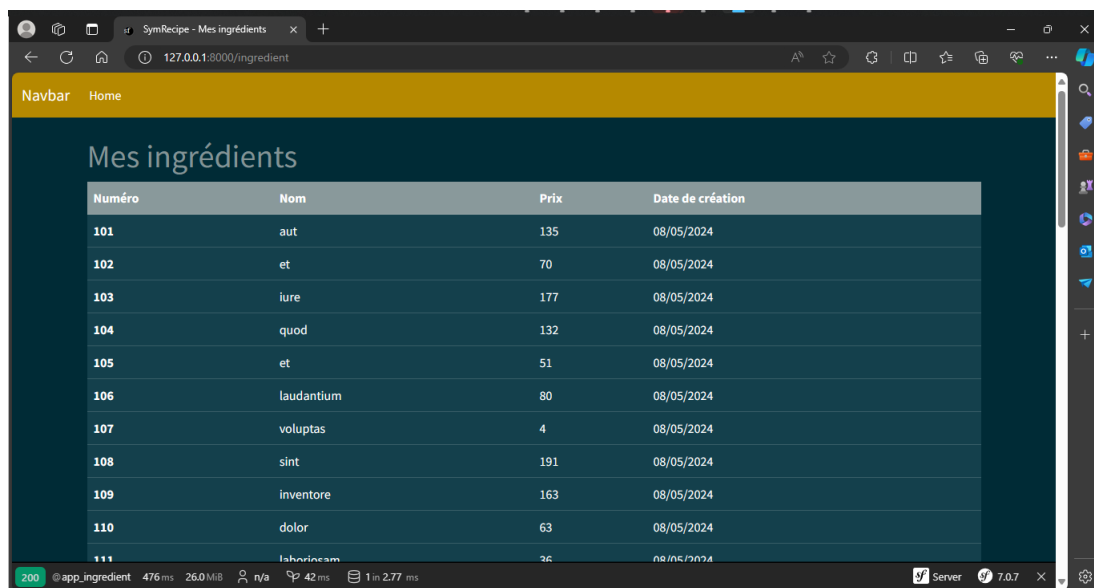


Modifiez le contenu du fichier `templates/ingredient/index.html.twig` :



```
1 {% extends 'base.html.twig' %}
2
3 {% block title %}SymRecipe - Mes ingrédients{% endblock %}
4
5 {% block body %}
6     <div class="container mt-4">
7         <h1>Mes ingrédients</h1>
8
9         <table class="table table-hover">
10             <thead>
11                 <tr class="table-secondary">
12                     <th scope="col">Numéro</th>
13                     <th scope="col">Nom</th>
14                     <th scope="col">Prix</th>
15                     <th scope="col">Date de création</th>
16                 </tr>
17             </thead>
18             <tbody>
19                 {% for ingredient in ingredients %}
20                     <tr class="table-dark">
21                         <th scope="row">{{ ingredient.id }}</th>
22                         <td>{{ ingredient.name }}</td>
23                         <td>{{ ingredient.price }}</td>
24                         <td>{{ ingredient.createdAt|date('d/m/Y') }}</td>
25                     </tr>
26                 {% endfor %}
27             </tbody>
28         </table>
29     </div>
30 {% endblock %}
31
```

Puis rechargez la page /ingredient :



Numéro	Nom	Prix	Date de création
101	aut	135	08/05/2024
102	et	70	08/05/2024
103	iure	177	08/05/2024
104	quod	132	08/05/2024
105	et	51	08/05/2024
106	laudantium	80	08/05/2024
107	voluptas	4	08/05/2024
108	sint	191	08/05/2024
109	inventore	163	08/05/2024
110	dolor	63	08/05/2024

1.5 KnpPaginatorBundle

Nous allons utiliser [KnpPaginatorBundle](#). C'est un bundle Symfony pour faire de la pagination.

1.5.1 A votre tour



1. Commencez par installer le paquet en exécutant la commande :

```
composer require knplabs/knp-paginator-bundle
```



```
Windows PowerShell
PS C:\Users\alexa\symfony-projects\symrecipe> composer require knplabs/knp-paginator-bundle
./composer.json has been updated
Running composer update knplabs/knp-paginator-bundle
Loading composer repositories with package information
Updating dependencies
Lock file operations: 2 installs, 0 updates, 0 removals
- Locking knplabs/knp-components (v4.4.0)
- Locking knplabs/knp-paginator-bundle (v6.3.0)
Writing lock file
Installing dependencies from lock file (including require-dev)
Package operations: 2 installs, 0 updates, 0 removals
- Downloading knplabs/knp-components (v4.4.0)
- Downloading knplabs/knp-paginator-bundle (v6.3.0)
- Installing knplabs/knp-components (v4.4.0): Extracting archive
- Installing knplabs/knp-paginator-bundle (v6.3.0): Extracting archive
Generating autoload files
114 packages you are using are looking for funding.
Use the 'composer fund' command to find out more!

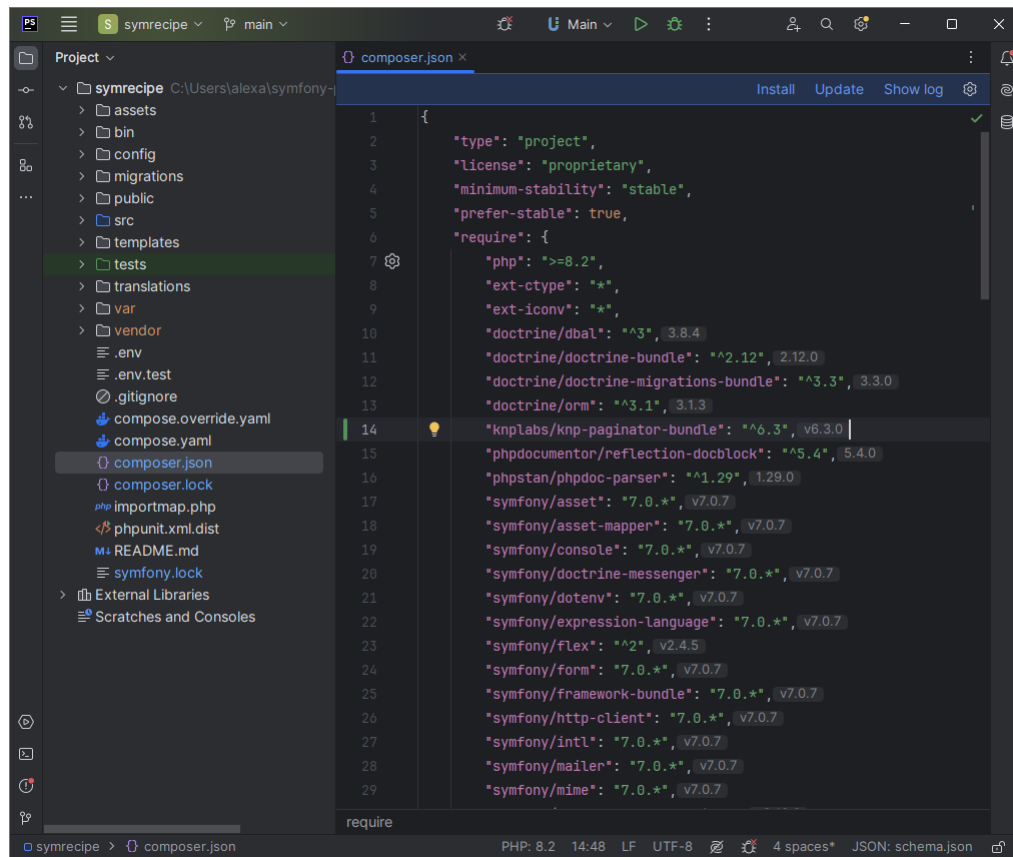
Symfony operations: 1 recipe (9c3e1985ec8316ec6883bae0cb4662a1)
- Configuring knplabs/knp-paginator-bundle (>=v6.3.0): From auto-generated recipe
Executing script cache:clear [OK]
Executing script assets:install public [OK]
Executing script importmap:install [OK]

What's next?

Some files have been created and/or updated to configure your new packages.
Please review, edit and commit them: these files are yours.

No security vulnerability advisories found.
Using version ^6.3 for knplabs/knp-paginator-bundle
PS C:\Users\alexa\symfony-projects\symrecipe> |
```

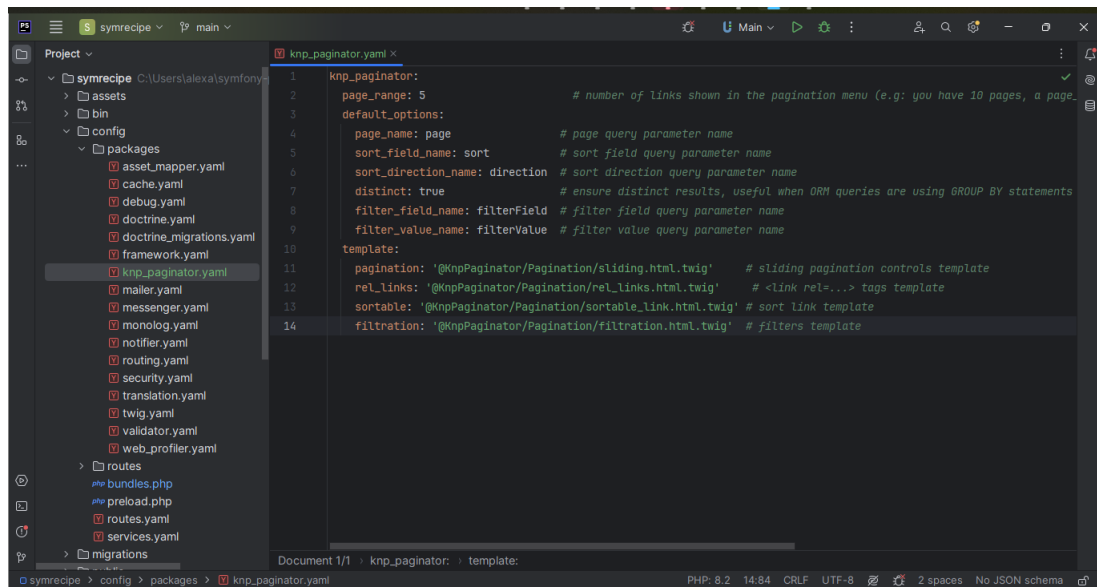
Vérifiez la présence de `knplabs/knp-paginator-bundle` dans `composer.json` :



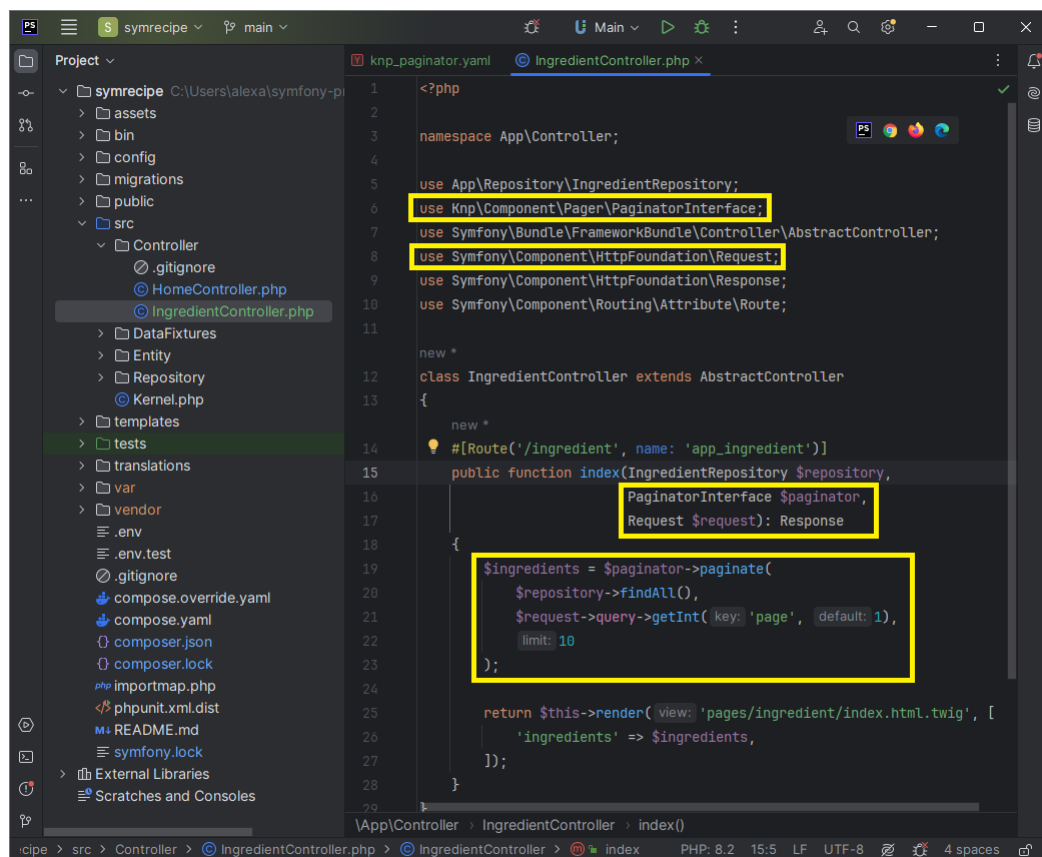
2. Dans config/packages créez un fichier knp_paginator.yaml avec le contenu suivant :

```
knp_paginator:
  page_range: 5
  default_options:
    page_name: page
    sort_field_name: sort
    sort_direction_name: direction
    distinct: true
    filter_field_name: filterField
    filter_value_name: filterValue
  template:
    pagination: '@KnpPaginator/Pagination/bootstrap_v5_pagination.html.twig'
    sortable: '@KnpPaginator/Pagination/sortable_link.html.twig'
    filtration: '@KnpPaginator/Pagination/filtration.html.twig'
```





3. Modifiez le contenu du fichier src/Controller/IngredientController.php :



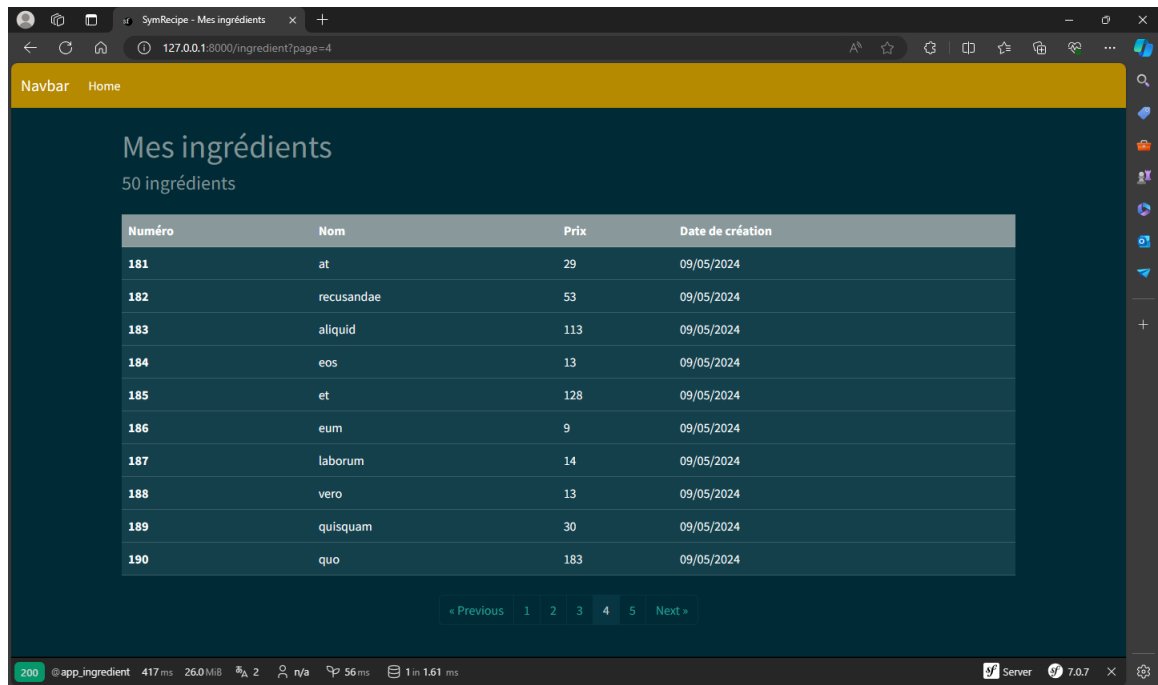
4. Modifiez le contenu du fichier templates/ingredient/index.html.twig :

```

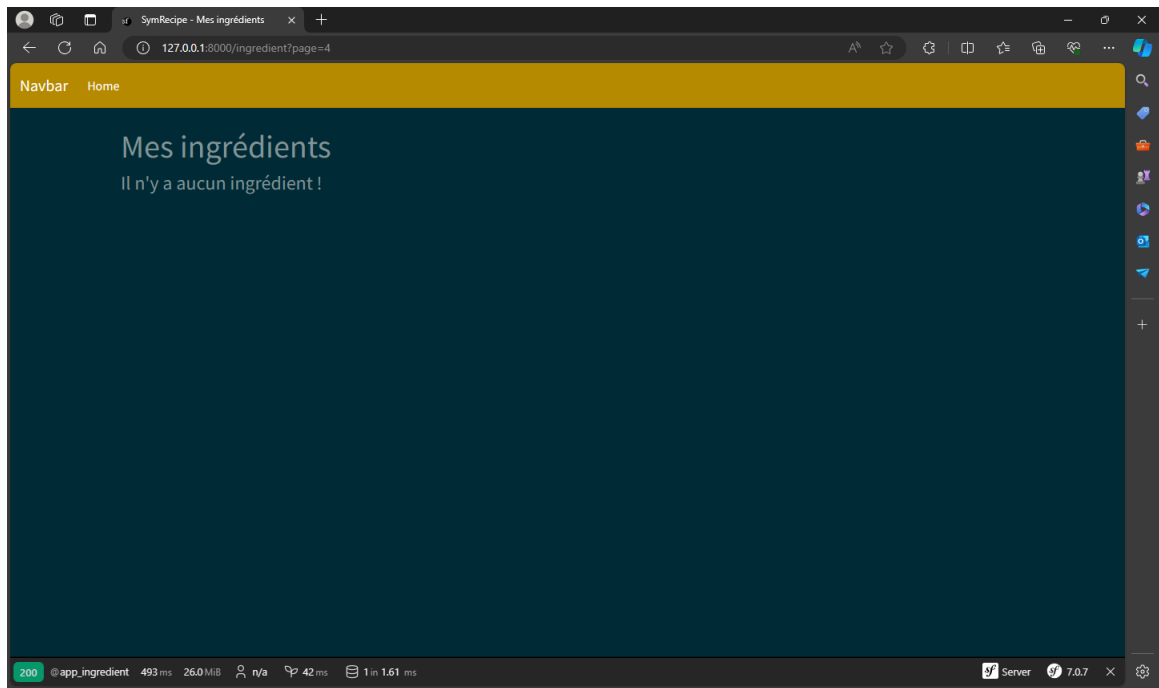
1 {% extends 'base.html.twig' %}
2
3 {% block title %}SymRecipe - Mes ingrédients{% endblock %}
4
5 {% block body %}
6     <div class="container mt-4">
7         <h1>Mes ingrédients</h1>
8         {% if not ingredients.items is same as ([]) %}
9             <h4>{{ ingredients.getTotalItemCount }} ingrédients</h4>
10            <table class="table table-hover mt-4">
11                <thead>
12                    <tr>
13                        <th>Numéro</th>
14                        <th>Nom</th>
15                        <th>Prix</th>
16                        <th>Date de création</th>
17                    </tr>
18                </thead>
19                <tbody>
20                    {{ knp_pagination_render(ingredients) }}
21                </tbody>
22            </table>
23        {% else %}
24            <h4>Il n'y a aucun ingrédient !</h4>
25        {% endif %}
26    </div>
27 {% endblock %}

```

5. Rechargez la page /ingredient :



Enfin, videz la table ingredient et rechargez :



Message de validation : « Partie 2 : CRUD des ingrédients - Liste des ingrédients »

2 Formulaire pour créer un ingrédient

Le workflow recommandé lorsque vous travaillez avec des [formulaires Symfony](#) est le suivant :

1. Construisez le formulaire dans un contrôleur Symfony ou en utilisant une classe de formulaire dédiée ;
2. Rendre le formulaire dans un modèle afin que l'utilisateur puisse le modifier et le soumettre ;
3. Traitez le formulaire pour valider les données soumises, transformez-les en données PHP et faites-en quelque chose (par exemple, conservez-les dans une base de données).

Chacune de ces étapes est expliquée en détail dans les sections suivantes.



Form Types

Avant de créer votre premier formulaire Symfony, il est important de comprendre la notion de « type de formulaire » (*Form Types* en anglais). Dans d'autres frameworks, il est courant de faire la différence entre les « formulaires » et les « champs de formulaire ». Dans Symfony, ce sont tous des « types de formulaires » :

- un seul champ de formulaire `<input type="text">` est un « type de formulaire » (par exemple `TextType`) ;
- un groupe de plusieurs champs HTML utilisé pour saisir une adresse postale est un « type de formulaire » (par exemple `PostalAddressType`) ;
- un `<form>` entier avec plusieurs champs pour modifier un profil utilisateur est un « type de formulaire » (par exemple `UserProfileType`).

Cela peut paraître déroutant au début, mais cela vous semblera bientôt naturel. En outre, cela simplifie le code et rend les champs de formulaire plus faciles à composer et intégrer dans une application.

2.1 Construire un formulaire

2.1.1 A votre tour



1. Créez votre premier formulaire avec la commande suivante :

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

et répondez aux deux questions comme indiqué dans la capture d'écran ci-après :

```
Windows PowerShell
PS C:\Users\alexa\symfony-projects\symrecipe> php bin/console make:form

The name of the form class (e.g. OrangeElephantType):
> IngredientType

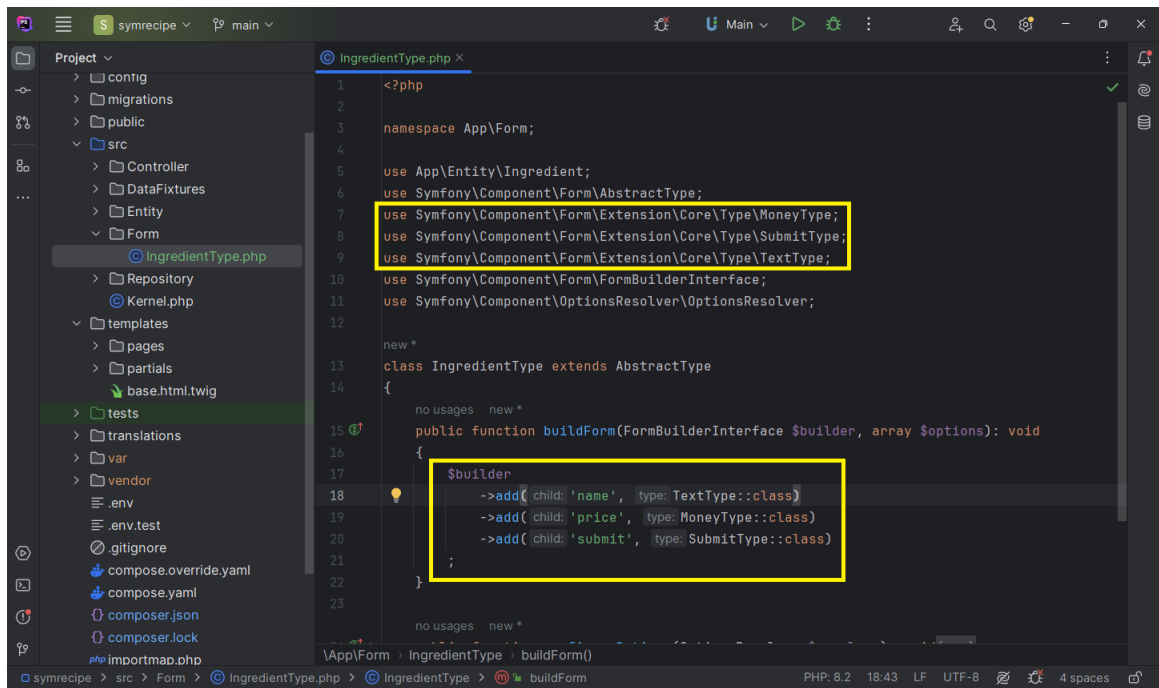
The name of Entity or fully qualified model class name that the new form will be bound to (empty for none):
> Ingredient

created: src/Form/IngredientType.php

Success!

Next: Add fields to your form and start using it.
Find the documentation at https://symfony.com/doc/current/forms.html
PS C:\Users\alexa\symfony-projects\symrecipe> |
```

2. Puis, dans le nouveau fichier `src/Form/IngredientType.php` créé à l'instant par la dernière commande, **supprimez** la ligne qui ajoute la date de création `createdAt` car elle est créée dans le constructeur de la classe `Ingredient`.
Ensuite, ajoutez les *Form Types* et le bouton de validation du formulaire `submit` :



2.2 Rendre le formulaire

2.2.1 A votre tour



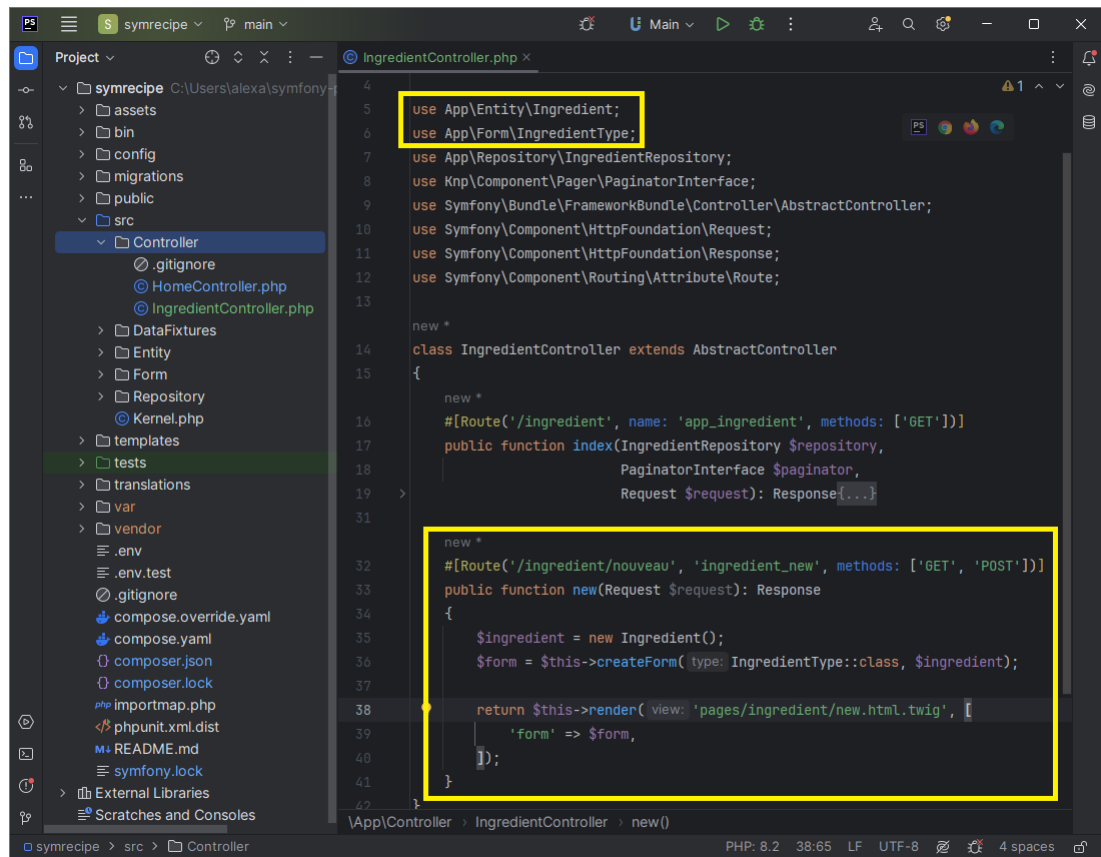
1. Ajoutez au contrôleur (classe IngredientController) la méthode suivante :

```

<?php
//src/Controller/IngredientController.php
// ...
use App\Entity\Ingredient;
use App\Form\IngredientType;
// ...
#[Route('/ingredient/nouveau', 'ingredient_new', methods: ['GET', 'POST'])]
public function new(Request $request): Response
{
    $ingredient = new Ingredient();
    $form = $this->createForm(IngredientType::class, $ingredient);

    return $this->render('pages/ingredient/new.html.twig');
}
//...
  
```





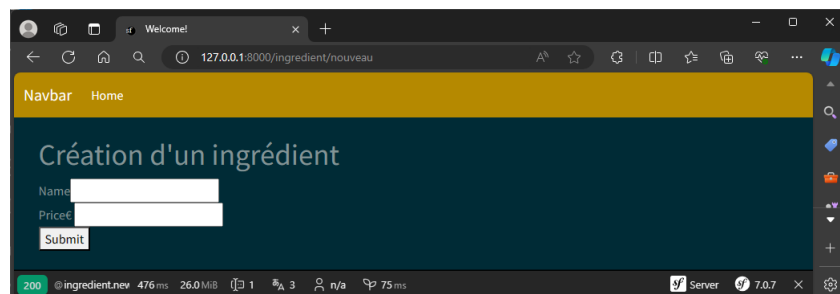
En interne, la méthode `render()` appellera `$form->createView()` pour transformer le formulaire en instance de vue de formulaire.

2. Puis, créez un fichier `new.html.twig` avec le contenu suivant :

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

{% block body %}
    <div class="container" >
        <h1 class="mt-4">Création d'un ingrédient</h1>
        {{ form(form) }}
    </div>
{% endblock %}
```

3. Chargez la page `/ingredient/nouveau` :



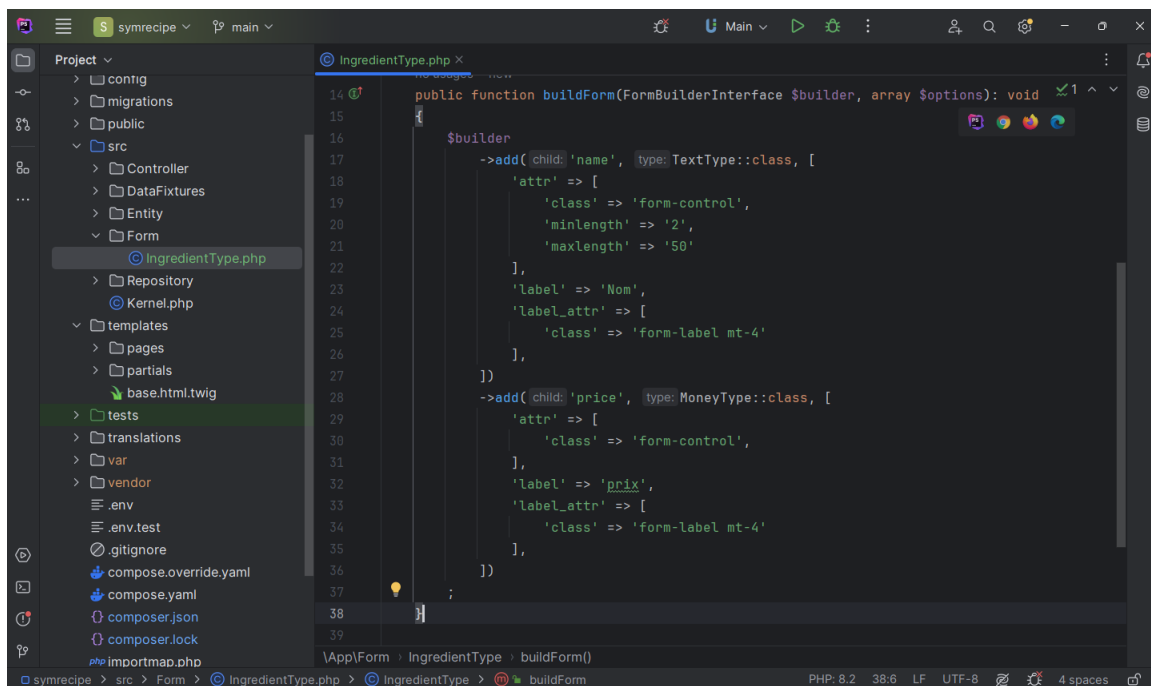
Et voilà ! La fonction `form()` a restitué tous les champs du formulaire (y compris les messages d'erreur, vérifiez le) ainsi les balises de début et de fin `<form>`. Par défaut, la méthode du formulaire est POST et l'URL cible est la même que celle qui affiche le formulaire, mais les deux sont modifiables.

2.2.2 A votre tour

Nous allons maintenant améliorer le rendu de nos deux champs de formulaire `MoneyType` et `MoneyType` en leur ajoutant deux *options* (`attr` et `label`) ainsi que des contraintes :

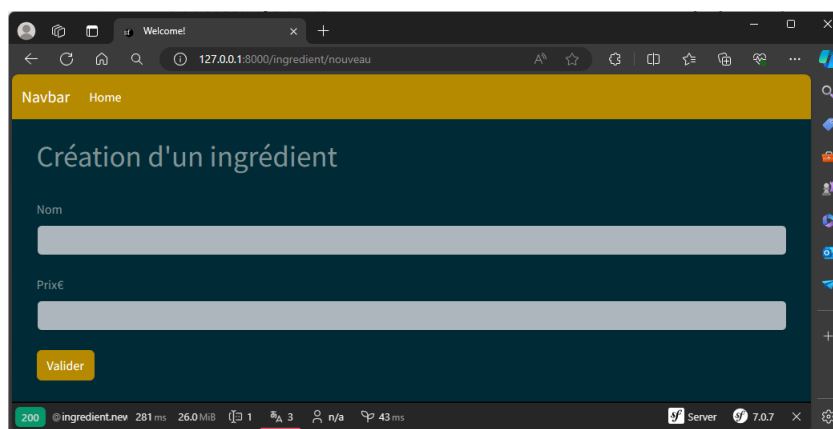


Modifiez la méthode `buildForm` de la classe `IngredientType` :



```
14 public function buildForm(FormBuilderInterface $builder, array $options): void
15 {
16     $builder
17         ->add('name', type: TextType::class, [
18             'attr' => [
19                 'class' => 'form-control',
20                 'minlength' => '2',
21                 'maxlength' => '50'
22             ],
23             'label' => 'Nom',
24             'label_attr' => [
25                 'class' => 'form-label mt-4'
26             ],
27         ])
28         ->add('price', type: MoneyType::class, [
29             'attr' => [
30                 'class' => 'form-control',
31             ],
32             'label' => 'prix',
33             'label_attr' => [
34                 'class' => 'form-label mt-4'
35             ],
36         ])
37     ;
38 }
```

Rechargez la page :

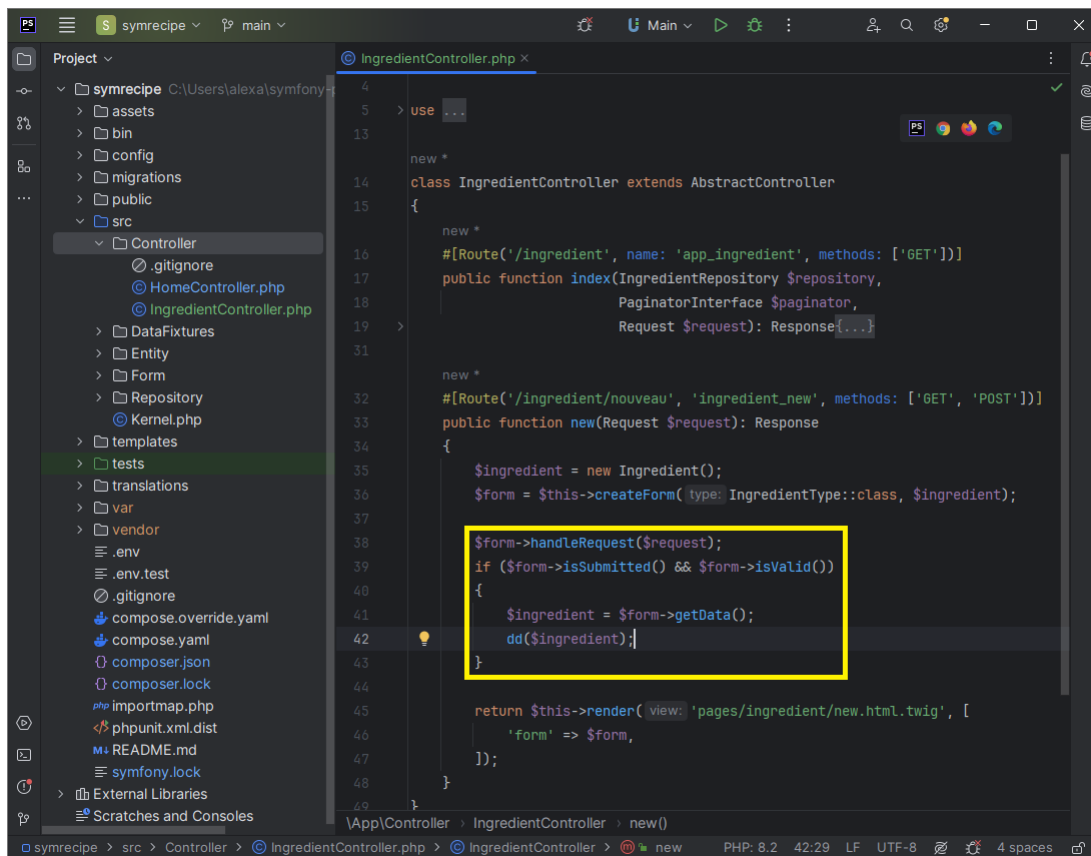


2.3 Traiter le formulaire

2.3.1 A votre tour

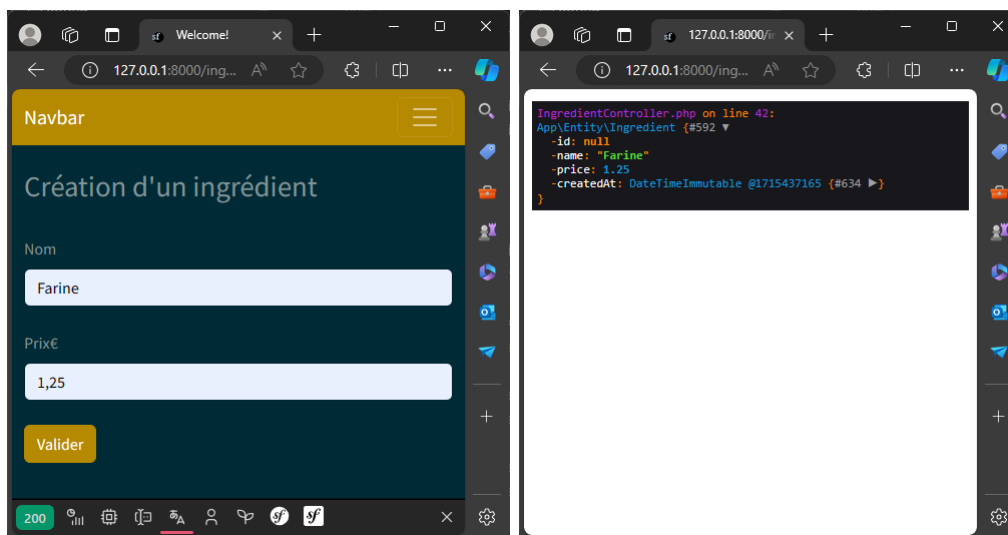


Modifiez le contrôleur :



```
4
5 > use ...
13
14 new *
15 class IngredientController extends AbstractController
16 {
17     new *
18     #[Route('/ingredient', name: 'app_ingredient', methods: ['GET'])]
19     public function index(IngredientRepository $repository,
20         PaginatorInterface $paginator,
21         Request $request): Response{...}
22
23     new *
24     #[Route('/ingredient/nouveau', 'ingredient_new', methods: ['GET', 'POST'])]
25     public function new(Request $request): Response
26     {
27         $ingredient = new Ingredient();
28         $form = $this->createForm( type: IngredientType::class, $ingredient);
29
30         $form->handleRequest($request);
31         if ($form->isSubmitted() && $form->isValid())
32         {
33             $ingredient = $form->getData();
34             dd($ingredient);
35         }
36
37         return $this->render( view: 'pages/ingredient/new.html.twig', [
38             'form' => $form,
39         ]);
40     }
41
42 }
```

Testez :



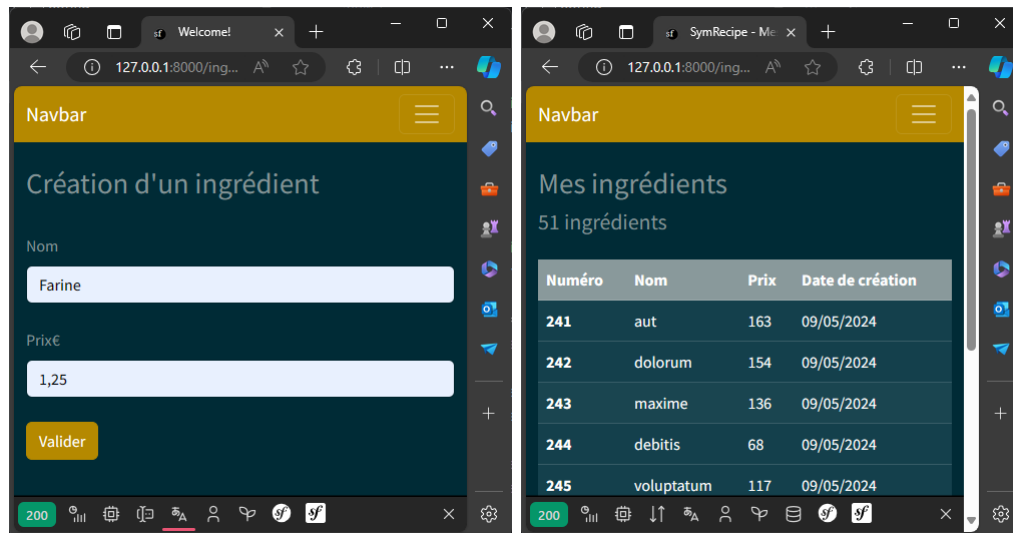
Enfin, dans le contrôleur, remplacez l'instruction php ligne 42 :

```
dd($ingredient);
```

par la suivante :

```
return $this->redirectToRoute('app_ingredient');
```

Refaites un test :

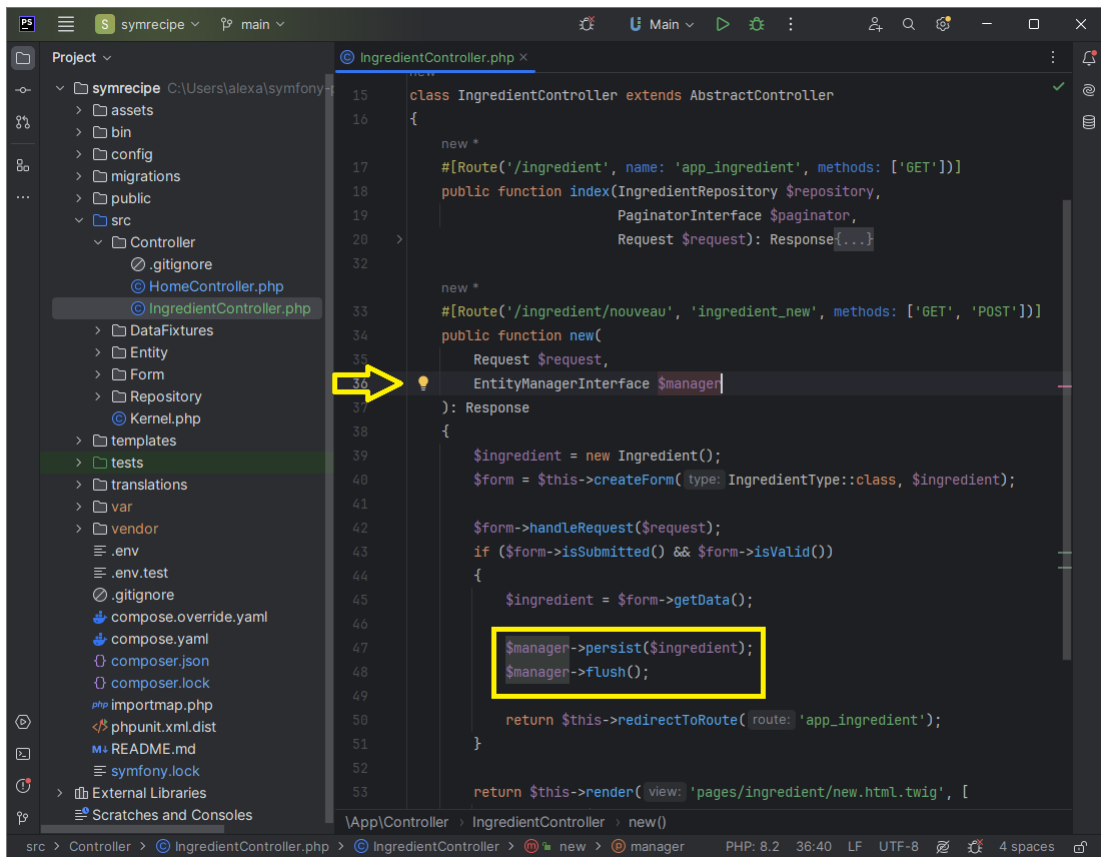


2.3.2 A votre tour

Maintenant, nous allons sauvegarder le nouvel ingrédient en base de données :



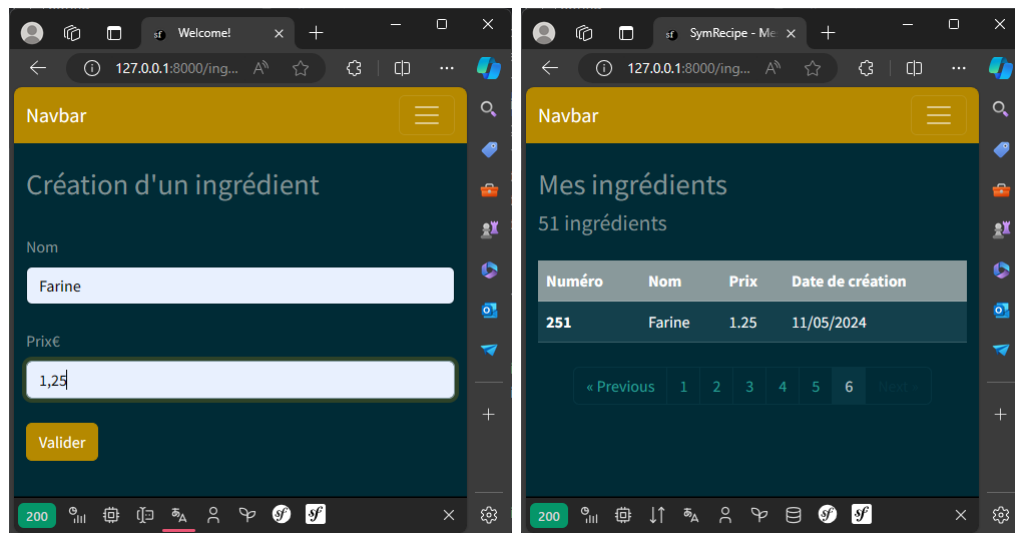
Pour ce faire, modifiez le contrôleur :



N'oubliez, en début de fichier (invisible sur la capture d'écran ci-avant), d'insérer la ligne :

```
use Doctrine\ORM\EntityManagerInterface;
```

Testez et vérifiez que le nouvel ingrédient est enregistré dans la base de données :





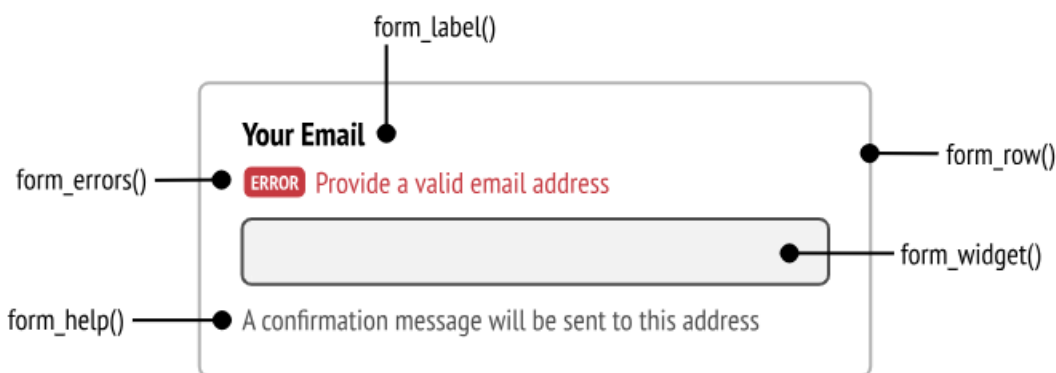
Message de validation : « Partie 2 : CRUD des ingrédients - Formulaire pour créer un ingrédient (construction, rendu et traitement) »

2.4 Personnalisation du rendu du formulaire

Symfony propose plusieurs façons de [personnaliser le rendu d'un formulaire](#).

Précédemment, un seul appel à la fonction *Twig* `form()` nous a suffi pour restituer le formulaire entier, y compris tous ses champs et messages d'erreur.

Maintenant, nous allons utiliser les fonctions *Twig* `form_start()`, `form_end()`, `form_errors()` et `form_row()` pour restituer les différentes parties du formulaire afin que vous puissiez les personnaliser en ajoutant des éléments et des attributs HTML :



2.4.1 A votre tour

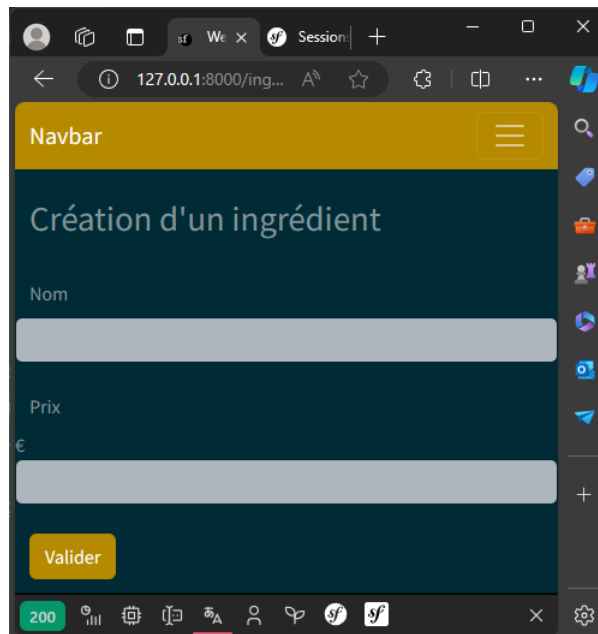


Modifiez votre fichier `templates/pages/ingredient/new.html.twig` :

The screenshot shows a code editor with a project sidebar on the left. The sidebar lists the project structure: symrecipe > templates > pages > ingredient. The selected file is 'new.html.twig'. The main editor displays the following Twig code:

```
1 {% extends "base.html.twig" %}
2
3 {% block body %}
4     <div class="container" >
5         <h1 class="mt-4">Création d'un ingrédient</h1>
6         {{ form_start(form) }}
7         <div class="row">
8             <div class="form-error">
9                 {{ form_errors(form.name) }}
10            </div>
11            {{ form_label(form.name) }}
12            {{ form_widget(form.name) }}
13        </div>
14        <div class="row">
15            <div class="form-error">
16                {{ form_errors(form.price) }}
17            </div>
18            {{ form_label(form.price) }}
19            {{ form_widget(form.price) }}
20        </div>
21        <div class="row">
22            {{ form_row(form.submit) }}
23        </div>
24        {{ form_end(form) }}
25    </div>
26 {% endblock %}
```

Puis, rechargez la page /ingredient/nouveau :



2.5 Message flash

Vous pouvez stocker des messages spéciaux, appelés [Flash Messages](#), sur la session de l'utilisateur. De par leur conception, les messages flashs sont destinés à être utilisés une seule fois : ils disparaissent automatiquement de la session dès que vous les récupérez. Cette fonctionnalité rend les messages flashs particulièrement utiles pour stocker les notifications des utilisateurs.

Par exemple, imaginez que vous traitez l'envoi d'un formulaire...

2.5.1 A votre tour



1. Dans votre contrôleur, ajouter l'instruction suivante :

```
33 new *
34 #[Route('/ingredient/nouveau', 'ingredient_new', methods: ['GET', 'POST'])]
35 public function new(
36     Request $request,
37     EntityManagerInterface $manager
38 ): Response
39 {
40     $ingredient = new Ingredient();
41     $form = $this->createForm(type: IngredientType::class, $ingredient);
42
43     $form->handleRequest($request);
44     if ($form->isSubmitted() && $form->isValid())
45     {
46         $ingredient = $form->getData();
47
48         $manager->persist($ingredient);
49         $manager->flush();
50
51         $this->addFlash(
52             type: 'success',
53             message: 'Vos changements ont été enregistrés !'
54         );
55
56         return $this->redirectToRoute(route: 'app_ingredient');
57     }
58
59     return $this->render(view: 'pages/ingredient/new.html.twig', [
60         'form' => $form,
61     ]);
62 }
```

Après avoir traité la requête du formulaire, le contrôleur dépose donc un message flash dans la session et exécute l'instruction de redirection.

La clé du message (`success` dans notre exemple) va être utilisée pour récupérer le message :

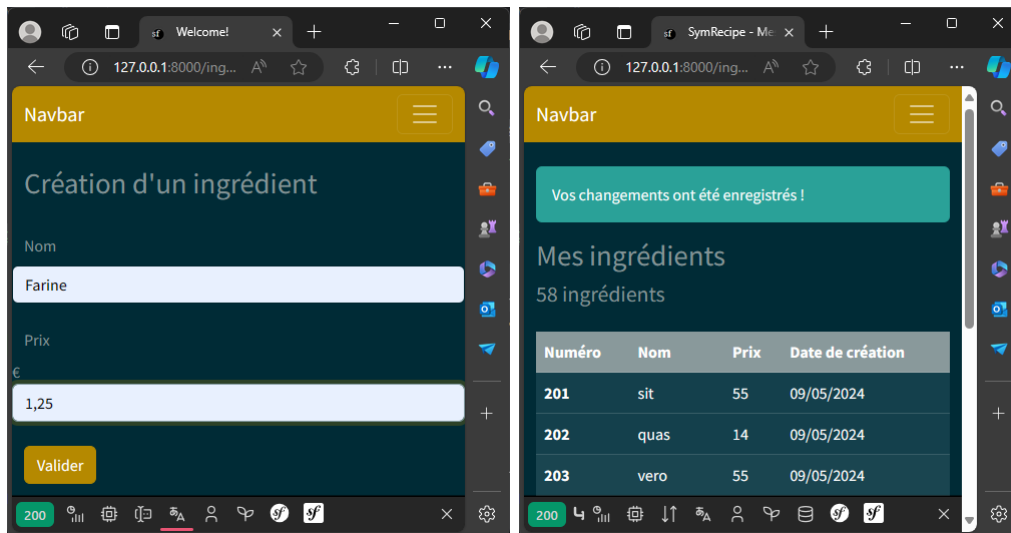
2. Dans le fichier `templates/pages/ingredient/index.html.twig`, ajoutez le code suivant :

```

1  {% block title %}SymRecipe - Mes ingrédients{% endblock %}
2
3  {% block body %}
4
5  <div class="container mt-4">
6
7      {% for message in app.flashes('success') %}
8          <div class="alert alert-dismissible alert-success mt-4">
9              {{ message }}
10             </div>
11         {% endfor %}
12     <h1>Mes ingrédients</h1>
13
14     {% if not ingredients.items is same as ([]) %}
15         <h4>{{ ingredients.getTotalItemCount }} Ingrédients</h4>
16         <table class="table table-hover mt-4">
17             <thead>
18                 <tr class="table-secondary">
19                     <th scope="col">Numéro</th>
20                     <th scope="col">Nom</th>
21                     <th scope="col">Prix</th>
22                     <th scope="col">Date de création</th>
23                 </tr>
24             </thead>
25             <tbody>
26                 {% for ingredient in ingredients %}
27                     <tr class="table-dark">
28                         <th scope="row">{{ ingredient.id }}</th>
29                         <td>{{ ingredient.name }}</td>
30                         <td>{{ ingredient.price }}</td>
31                         <td>{{ ingredient.createdAt|date('d/m/Y') }}</td>
32                     </tr>
33                 {% endfor %}
34             </tbody>
35         </table>
36     {% else %}
37         <h4>Aucun ingrédient trouvé</h4>
38     {% endif %}
39 </div>

```

3. Testez :



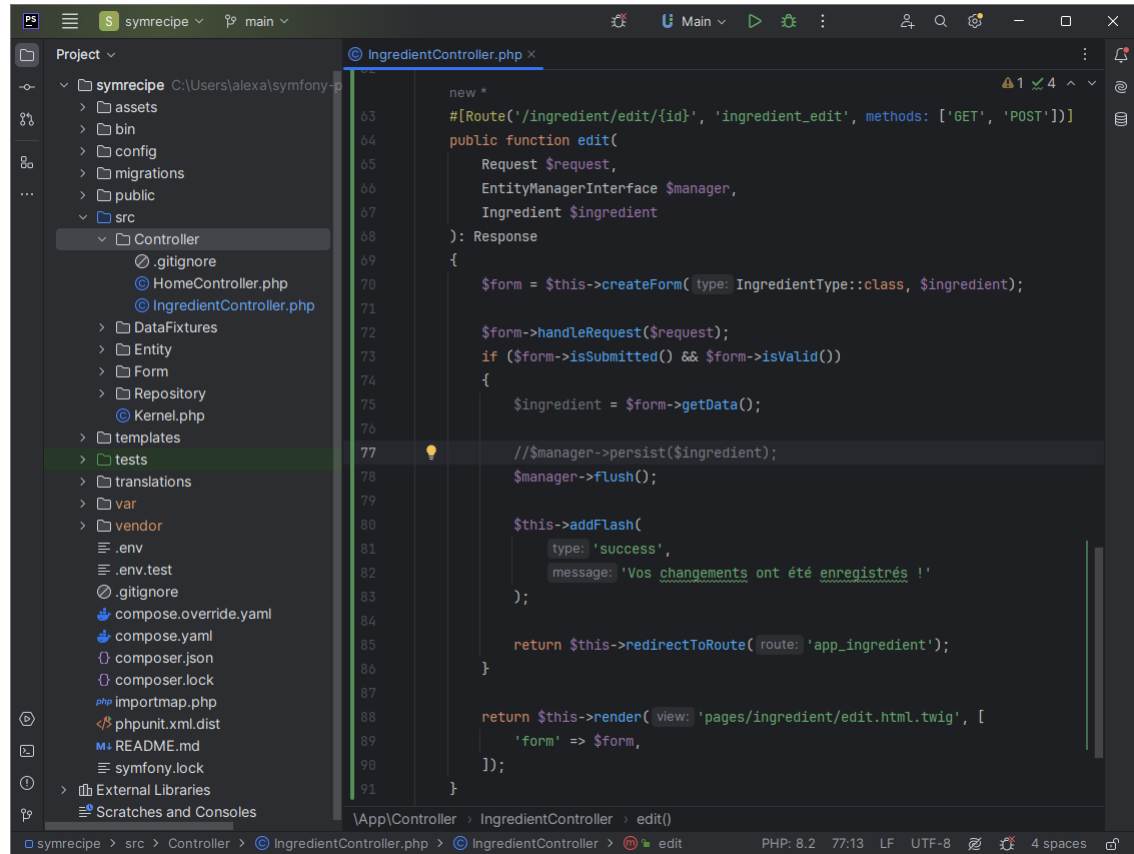
Message de validation : « Partie 2 : CRUD des ingrédients - Formulaire pour créer un ingrédient (Personnalisation du rendu et message flash) »

3 Modifier un ingrédient

3.1 A votre tour



1. Ajoutez à votre IngredientController la méthode suivante :



```
new *
#[Route('/ingredient/edit/{id}', 'ingredient_edit', methods: ['GET', 'POST'])]
public function edit(
    Request $request,
    EntityManagerInterface $manager,
    Ingredient $ingredient
): Response
{
    $form = $this->createForm( type: IngredientType::class, $ingredient);

    $form->handleRequest($request);
    if ($form->isSubmitted() && $form->isValid())
    {
        $ingredient = $form->getData();

        // $manager->persist($ingredient);
        $manager->flush();

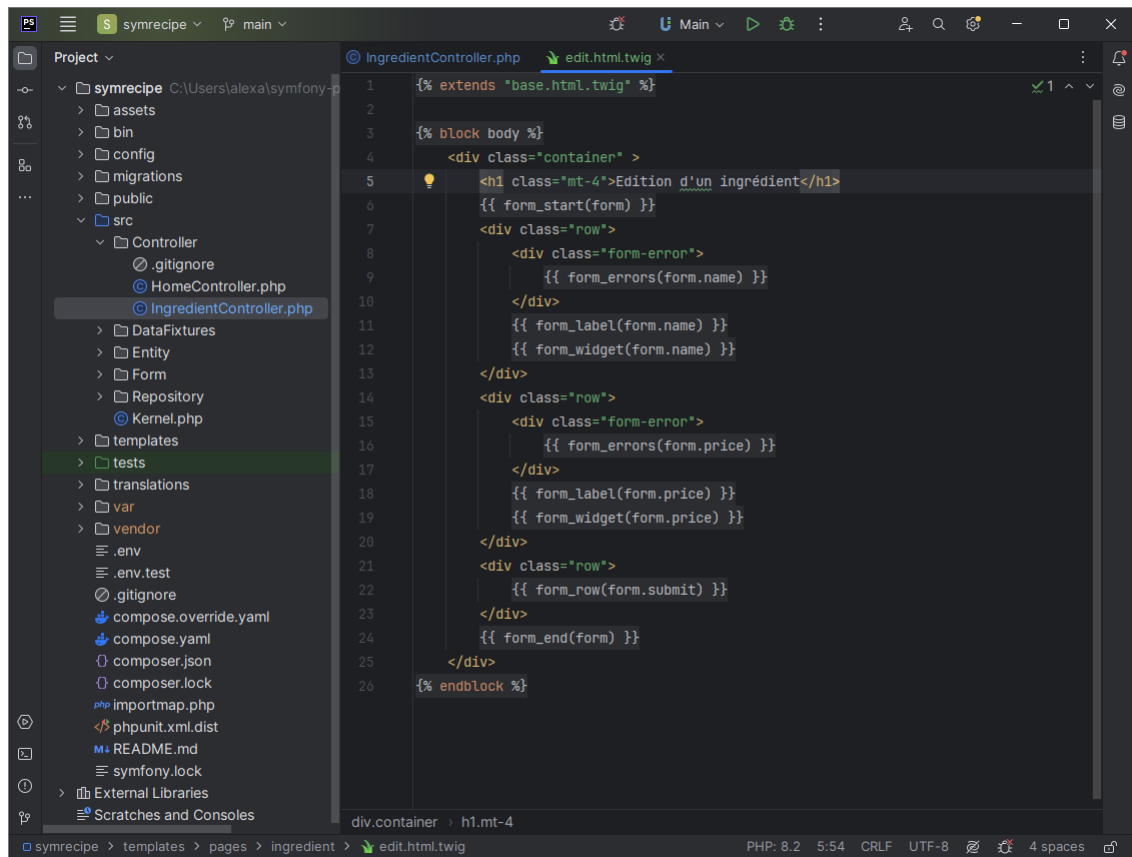
        $this->addFlash(
            type: 'success',
            message: 'Vos changements ont été enregistrés !'
        );

        return $this->redirectToRoute( route: 'app_ingredient');
    }

    return $this->render( view: 'pages/ingredient/edit.html.twig', [
        'form' => $form,
    ]);
}
```

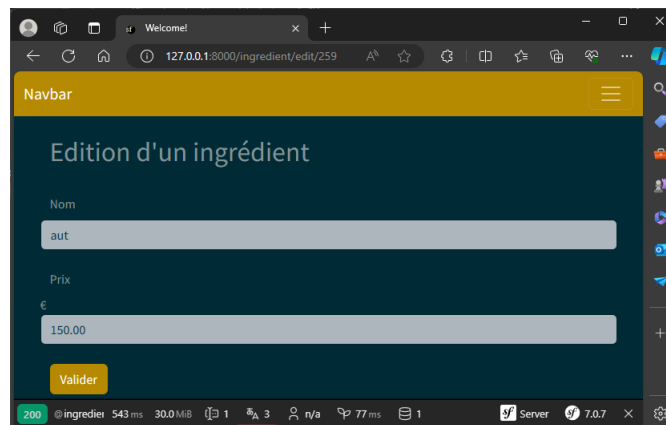
Vous pouvez appeler `$entityManager->persist($ingredient)`, mais ce n'est pas nécessaire : Doctrine « surveille » déjà les modifications apportées à votre entité.

2. Puis, créez un modèle `edit.html.twig` avec le contenu suivant :



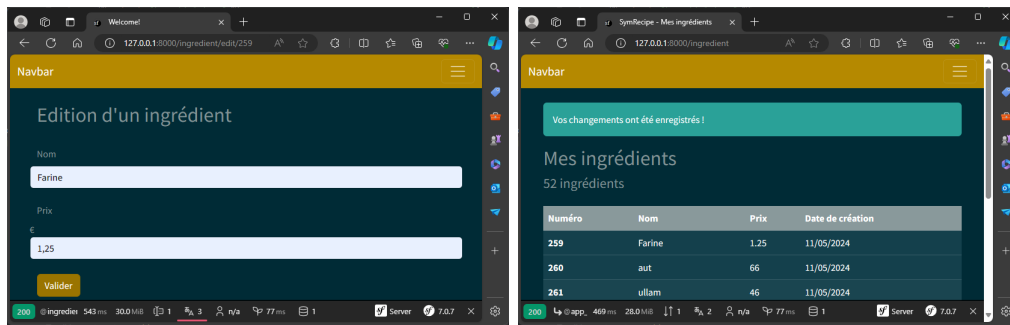
```
1 {% extends "base.html.twig" %}
2
3 {% block body %}
4     <div class="container" >
5         <h1 class="mt-4">Edition d'un ingrédient</h1>
6         {{ form_start(form) }}
7         <div class="row">
8             <div class="form-error">
9                 {{ form_errors(form.name) }}
10            </div>
11            {{ form_label(form.name) }}
12            {{ form_widget(form.name) }}
13        </div>
14        <div class="row">
15            <div class="form-error">
16                {{ form_errors(form.price) }}
17            </div>
18            {{ form_label(form.price) }}
19            {{ form_widget(form.price) }}
20        </div>
21        <div class="row">
22            {{ form_row(form.submit) }}
23        </div>
24        {{ form_end(form) }}
25    </div>
26 {% endblock %}
```

3. Enfin, chargez la page `/ingredient/edit/id` en remplaçant `id` par un nombre identifiant un ingrédient :



Modifiez, et validez...





Message de validation : « Partie 2 : CRUD des ingrédients - Modifier un ingrédient »

4 Supprimer un ingrédient

La suppression d'une entité est très similaire, mais nécessite un appel à la méthode `remove()` du gestionnaire d'entités :

4.1 A votre tour



1. Commencez par ajouter à votre `IngredientController` la méthode suivante :

```

Project
├── symrecipe
│   ├── assets
│   ├── bin
│   ├── config
│   ├── migrations
│   ├── public
│   └── src
│       ├── Controller
│       │   ├── IngredientController.php
│       │   ├── HomeController.php
│       │   ├── Kernel.php
│       │   ├── DataFixtures
│       │   ├── Entity
│       │   ├── Form
│       │   ├── Repository
│       │   ├── templates
│       │   ├── tests
│       │   ├── translations
│       │   ├── var
│       │   ├── vendor
│       │   ├── .env
│       │   ├── .env.test
│       │   ├── .gitignore
│       │   ├── compose.override.yaml
│       │   └── compose.yaml
│       └── tests
│           ├── tests
│           ├── translations
│           ├── var
│           ├── vendor
│           ├── .env
│           ├── .env.test
│           ├── .gitignore
│           ├── compose.override.yaml
│           └── compose.yaml
└── tests
    ├── tests
    ├── translations
    ├── var
    ├── vendor
    ├── .env
    ├── .env.test
    ├── .gitignore
    ├── compose.override.yaml
    └── compose.yaml

```

```

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

new *
#[Route('/ingredient/remove/{id}', 'ingredient_remove', methods: ['GET'])]
public function remove(
    Request $request,
    EntityManagerInterface $manager,
    Ingredient $ingredient
): Response
{
    $manager->remove($ingredient);
    $manager->flush();

    $this->addFlash(
        type: 'success',
        message: "L'ingrédient a été supprimé !"
    );

    return $this->redirectToRoute('route:app_ingredient');
}

```

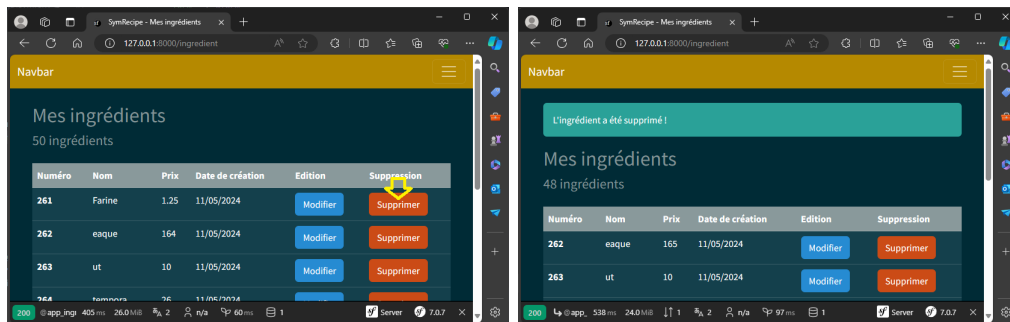
Comme vous vous en doutez, la méthode `remove()` informe Doctrine que vous souhaitez supprimer l'entité donnée de la base de données. La requête `DELETE` n'est réellement exécutée que lorsque la méthode `flush()` est appelée.

2. Ajoutez un « bouton supprimer » et un « bouton modifier » tant qu'on y est !



```
20 <th scope="col">Prix</th>
21 <th scope="col">Date de création</th>
22 <th scope="col">Edition</th>
23 <th scope="col">Suppression</th>
24 </tr>
25 </thead>
26 <tbody>
27 {% for ingredient in ingredients %}
28 <tr class="table-dark">
29 <th scope="row">{{ ingredient.id }}</th>
30 <td>{{ ingredient.name }}</td>
31 <td>{{ ingredient.price }}</td>
32 <td>{{ ingredient.createdAt|date('d/m/Y') }}</td>
33 <td>
34 <a
35 href="{{ path('ingredient_edit', { id: ingredient.id }) }}"
36 class="btn btn-info">Modifier
37 </a>
38 <td>
39 <a
40 href="{{ path('ingredient_remove', { id: ingredient.id }) }}"
41 class="btn btn-warning">Supprimer
42 </a>
43 </td>
44 </tr>
45 </tbody>
```

3. Vérifiez que la modification fonctionne et testez la suppression :



Message de validation : « Partie 2 : CRUD des ingrédients - Supprimer un ingrédient »