**Pour créer l'entité Produit**

CTRL CTRL Make:entity // ça va créer Produit dans dossier Entity et dossier Repository

CTRL CTRL make:controller // creer le fichier ProduitController dans le dossier Controllers et ça crée un dossier produit dans templates

Pour afficher tous nos produits go ProduitController et pour recuperer nos produits, mettre dans les params de la funciton index ProduitRepository $repo, grace a ça on a les methodes findall appelé:::

#[Route('/produit', name: 'produit')]  
public function index(ProduitRepository $repo): Response  
{  
 $products = $repo->findAll();  
  
 return $this->render('produit/index.html.twig', [  
 'produits' => $products,  
 ]);  
}composer updat

-> tableau d'objet (de produits) pour la vue on lui donne un nom choisi et on lui assigne la variable $products qui a toutes les infos

après ça go index.html.twig du dossier Produit ou j’ai reçu la variable produits et je boucle avec for pour produit in produits. Pour la mise en forme on le met dans un tableau

AJOUTER UN PRODUIT

Go ProduitController et créer une une fonction add qui renverra sur add.html.twig avec un formulaire d’ajout de produit

#[Route('produit/add', name: 'produit\_add')]  
public function add()  
{  
 return $this->render('produit/add.html.twig');  
}

Créer un bouton sur la page index pour créer un nouveau produit

#[Route('produit/add', name: 'produit\_add')]  
public function add()  
{  
 return $this->render('produit/add.html.twig');  
}

On va créer un nouvel objet Produit qui va contenir les infos du nouveau produit (qui est encore vide pour le moment)

$produit = new Produit();

Ce que me retourne le new Produit avec un dump sur le twig (vue), un objet vide, qui existe mais encore vide

[App\Entity\Produit](http://localhost:8000/_profiler/open?file=src\Entity\Produit.php&line=9#line9) {#508 ▼

-id: **null**

-reference: **null**

-nom: **null**

-description: **null**

-prix: **null**

-stock: **null**

-deleted: **null**

}

Créer un formulaire(ProduitType) et récupérer la requête

$form = $this->createForm(ProduitType::class, $produit);  
$form->handleRequest($request);

Pour créer l’affichage de notre formulaire avec nos champs, on utilise la méthode createView qui a tous mes champs étant donné que $form est un ProduitType

$formView = $form->createView();

C’est la variable $formView que je me passe à ma vue (add.html.twig)

return $this->render('produit/add.html.twig', [  
 'p' =>$produit,  
 'form' => $formView  
]);

Donc après ça, je passe sur add.html.twig pour afficher le formulaire :

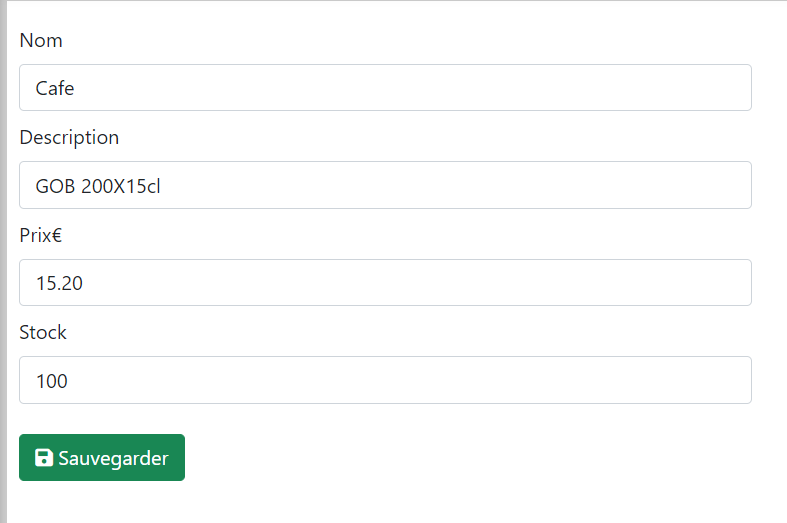
<form action="" method="post" enctype="multipart/form-data">  
 {{ form\_widget(form.\_token) }}  
 <div class="row">  
 <div class="form-group mt-2 col-6" readonly="true" >  
 {{ form\_row(form.nom) }}  
 </div>  
 </div>  
 <div class="row">  
 <div class="form-group mt-2 col-6">  
 {{ form\_row(form.description) }}  
 </div>  
 </div>  
 <div class="row">  
 <div class="form-group mt-2 col-6">  
 {{ form\_row(form.prix) }}  
 </div>  
 </div>  
 <div class="row">  
 <div class="form-group mt-2 col-6">  
 {{ form\_row(form.stock) }}  
 </div>  
 </div>  
 <div hidden>  
 {{ form\_rest(form) }}  
 </div>  
  
  
 <div class="form-group mt-4">  
 <button type="submit" class="btn btn-success">  
 <i class="fa-solid fa-floppy-disk"></i>  
 Sauvegarder  
 </button>  
 </div>  
</form>

Le formulaire s’affiche mais il ne se passe rien, go ProduitController pour gérer le comportement

Pour voir l’état de mon formulaire je soumet le formulaire et je verifie avec isSubmitted

if($form->isSubmitted()){

Pour voir les données de mon formulaire



dd($form->getData());

if($form->isSubmitted()){  
   
 dd($form->getData());  
 }

Avec le résultat

[ProduitController.php](http://localhost:8000/_profiler/open?file=src\Controller\ProduitController.php&line=42#line42) on line 42:

[App\Entity\Produit](http://localhost:8000/_profiler/open?file=src\Entity\Produit.php&line=9#line9) {#430 ▼

-id: **null**

-reference: **null**

-nom: "**Cafe**"

-description: "**GOB 200X15cl**"

-prix: **15.2**

-stock: **100**

-deleted: **null**

}

La référence ne peut pas être modifier via le formulaire, elle est générée grâce au nom et au nombre de produits avec le même nom

La variable $produit me permet de modifier cet attribut avec la fonction setReference() : dans l’entité(Produit)

public function setReference($reference)  
{  
 $this->reference = $reference;  
 return $this;  
}

Le code ci-dessous est pour recuperer les 4 premieres lettre de mon nom, en majuscule, et compté les doublons pour rajouter +1 si existe déjà (voir countByRef dans ProduitRepository)

$ref = strtoupper(substr($produit->getNom(), 0, 4));  
$count = $repo->countByRef($ref) + 1;  
$ref .= str\_pad($count, 4, '0', *STR\_PAD\_LEFT*);  
$produit->setReference($ref);

Meme chose pour deleted avec setDeleted() qu’on passe a false

$produit->setDeleted(false);

if($form->isSubmitted()){  
 $ref = strtoupper(substr($produit->getNom(), 0, 4));  
 $count = $repo->countByRef($ref) + 1;  
 $ref .= str\_pad($count, 4, '0', *STR\_PAD\_LEFT*);  
 $produit->setReference($ref);  
  
 $produit->setDeleted(false);  
 dd($form->getData());  
   
}

Maintenant que je soumets le formulaire les attributs référence et delete ne sont plus nuls

[ProduitController.php](http://localhost:8000/_profiler/open?file=src\Controller\ProduitController.php&line=42#line42) on line 42:

[App\Entity\Produit](http://localhost:8000/_profiler/open?file=src\Entity\Produit.php&line=9#line9) {#595 ▼

-id: **null**

-reference: "**CAFE0001**" ////////ici ça remplit la reference

-nom: "**Cafe**"

-description: "**GOB 200X15cl**"

-prix: **15.2**

-stock: **100**

-deleted: **false /////////et ça set le deleted a false**

}

Ensuite on a persisté (qui garde les données en local) et flush (qui push dans la db) avant de redirigé vers la route app\_produit qui affiche tous mes produits

$em->persist($produit);  
$em->flush();  
$this->addFlash('success', 'Produit crée');  
return $this->redirectToRoute('app\_produit');

FORMULAIRE D’EDITION

Création de la route et de la fonction édit dans ProduitController

#[Route('/produit/edit/{id}', name: 'produit\_edit')]  
public function edit($id)  
{  
  
}

Générer le formulaire pour l’entité (tous les éléments en champ)

php bin/console make:form

personalisation du formulaire avec les attributs

public function buildForm(FormBuilderInterface $builder, array $options): void  
{  
 $builder  
 ->add('reference')  
 ->add('nom')  
 ->add('description')  
 ->add('prix')  
 ->add('stock')  
 ->add('deleted')  
 ;  
}

Dans ProduitController on s’occupe de recupere 1 produit par son id en utilisant le repository

#[Route('/produit/edit/{id}', name: 'produit\_edit')]  
public function edit($id, ProduitRepository $repo)  
{  
 $product = $repo->find($id);  
  
 return $this->render('produit/edit.html.twig', [  
 'product' => $product // on l’envoi a la vue en lui donnant un nom pour twig  
 ]);  
}

Dans Produit/edit.html.twig on recupere la variable products