Table des matières

I- Mission 1

a) Contexte et Objectifs

Doctrine, un ORM qui simplifie la gestion des bases de données en utilisant des objets PHP. En l'intégrant à Symfony, j'ai pu créer, modifier et vérifier des produits grâce à des commandes intuitives et des requêtes SQL. Cela a facilité la manipulation des données dans mon application.

Cette mission concerne la deuxième itération du projet Maya, où l'objectif est d'apprendre à utiliser l'ORM Doctrine avec Symfony en se concentrant sur les produits et leurs catégories.

b) Produit/Categorie

1) Produit : oppération base de donnée

Lors de la suppression d'un produit, j'ai d'abord ajouté plusieurs éléments à la base pour les tester. Ensuite, j'ai mis en place une méthode dans le contrôleur pour supprimer un produit spécifique. En vérifiant à nouveau la base, j'ai constaté avec satisfaction que le produit avait été correctement supprimé, démontrant l'efficacité de Doctrine dans ces opérations.

Dans la section des requêtes avancées, j'ai appris à créer des méthodes spécifiques pour répondre à des besoins particuliers. J'ai ajouté une méthode pour récupérer les produits dont le prix est supérieur à un paramètre donné, optimisant ainsi ma recherche d'informations spécifiques dans la base de données.

#	Nom	Type	Interclassement	Attributs	Null	Valeur par défaut	Commentaires	Extra	Action
1	id	int(11)			Non	*Aucun(e)*		AUTO_INCREMENT	Modifier Supprimer Plus
2	categorie_id	int(11)			Non	*Aucun(e)*			Modifier Supprimer Plus
3	libelle	varchar(40)	utf8mb4_unicode_ci		Non	*Aucun(e)*			Modifier Supprimer Plus
4	prix	decimal(7,2)			Non	*Aucun(e)*			Modifier Supprimer Plus
5	date_creation	datetime			Non	*Aucun(e)*			Modifier Supprimer Plus

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

Voici le libellé du produit : mirabelle créer le 2023-10-09 12:17:55

2) Catégorie : Rélation

En poursuivant, j'ai découvert l'importance des associations entre entités. Grâce à Doctrine, j'ai créé une entité "Categorie" et établi une relation ManyToOne, cette rélation permet de réccupèrer des infom entre la catégorie et le produit. En générant et exécutant des migrations, j'ai mis à jour la base de données pour refléter ces nouvelles associations, renforçant ainsi la structure de mon application.

#	Nom	Type	Interclassement	Attributs	Null	Valeur par défaut	Commentaires	Extra	Action
1	id	int(11)			Non	*Aucun(e)*		AUTO_INCREMENT	Modifier Supprimer Plus
2	libelle	varchar(50)	utf8mb4_unicode_ci		Non	*Aucun(e)*			Modifier Supprimer Plus

c) Palette de Commande

- php bin/console doctrine:query:sql [Requête SQL]
  Exécute une requête SQL directement dans la base de données.

- php bin/console doctrine:query:sql "SELECT * FROM produit"
  Exécute une requête SQL pour sélectionner tous les produits de la table "produit".

- Php bin/console make:form
  Créer un formulaire

d) Conclusion

En conclusion, cette expérience avec Doctrine et Symfony m'a permis de mieux comprendre la gestion des bases de données en PHP. J'ai acquis des compétences précieuses dans la création, la modification et la suppression d'entités, ainsi que dans la mise en place de relations entre elles. Ces connaissances seront essentielles pour développer des applications web plus complexes à l'avenir.

II- Mission 2

a) Opérations de base avec deux entités associées

- Création de l'entité Recette avec une relation ManyToMany vers Produit.
- Configuration des attributs mappedBy et inversedBy pour établir la relation bidirectionnelle.
- association ManyToMany

Création d'un contrôleur RecetteController pour gérer la création de recettes et leur persistance en base de données.

b) Créer des données de tests (fixtures)

Utilisation de DoctrineFixturesBundle pour créer des données de test. Création de catégories, de produits associés à ces catégories, et utilisation aléatoire de produits dans les recettes.

c) Conclusion

Le tutoriel nous guide à travers la création d'entités, l'établissement de relations, et l'ajout de données de test. Il utilise Doctrine et Symfony pour simplifier la manipulation des données.

III- Mission 3

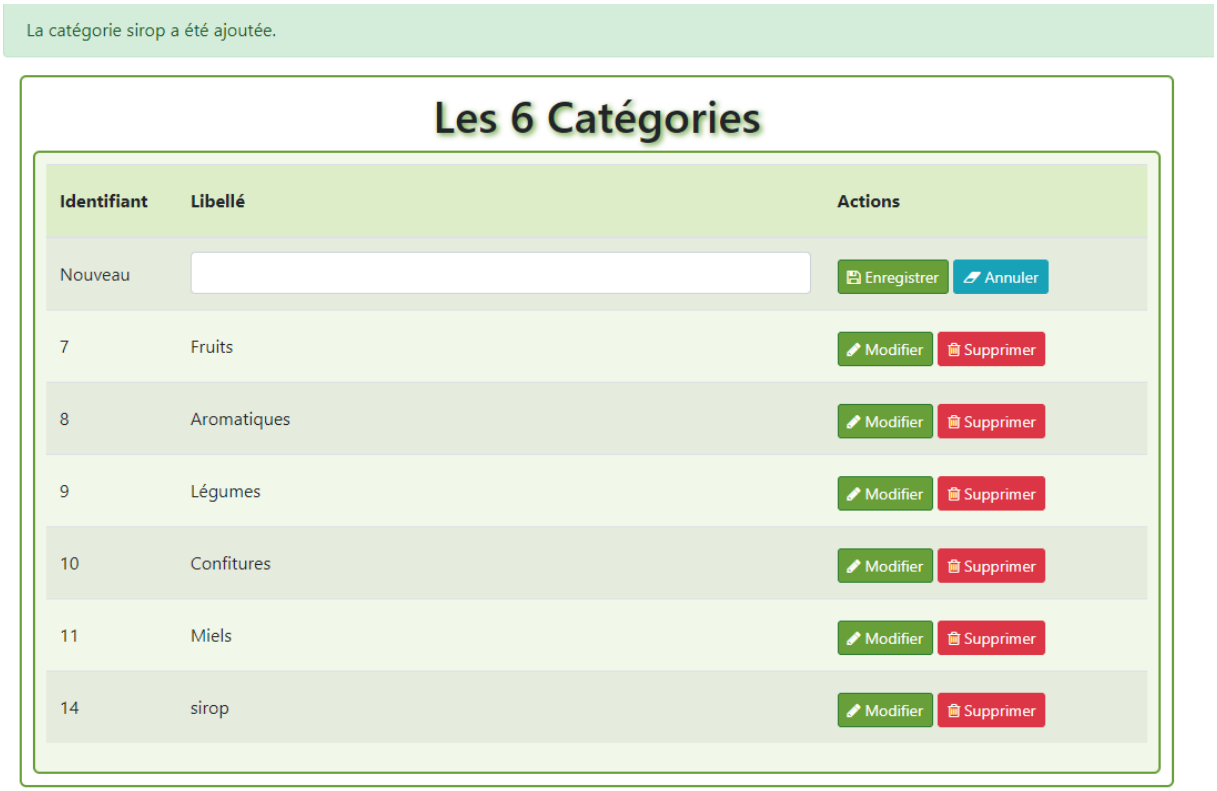
a) Introduction

Symfony est un framework PHP qui simplifie le processus de développement web en fournissant une structure organisée et des composants réutilisables. Dans le contexte de la refonte du site internet de la Ferme Maya, l'entreprise Logma a choisi d'utiliser Symfony pour développer le back office, c'est-à-dire l'application web d'administration des données. L'utilisation de Symfony permettra de faciliter la prise en charge des évolutions futures envisagées par la Ferme Maya.

b) Préparation

Dans cette première partie, nous abordons les étapes nécessaires pour préparer l'environnement de travail. Nous avons déjà créé une application Symfony nommée "Maya" lors des missions précédentes. Pour cette mission, nous devons faire une copie de cette application et la renommer en "Maya_sprint2-M1-M2". Ensuite, nous devons ouvrir l'application "Maya" dans notre environnement de développement intégré (EDI) et effectuer des modifications dans les classes "ProduitController" et "RecetteController" en supprimant certaines méthodes.

b) Catégories



1) Gérer

Dans cette partie, nous allons nous concentrer sur la gestion des catégories de produits en utilisant les formulaires Symfony.

2) Afficher

Nous commençons par afficher la liste des catégories, ce qui est le comportement par défaut pour la gestion des catégories. Nous utilisons la route "/categorie" et la méthode index du contrôleur CategorieController. Nous adaptons le code de CategorieController en injectant le repository nécessaire pour récupérer les catégories. En parallèle, nous modifions le template "index.html.twig" pour afficher les catégories dans un tableau HTML.

3) Nouvelle catégorie

Nous ajoutons un formulaire au-dessus de la liste des catégories pour permettre à l'utilisateur de saisir les informations d'une nouvelle catégorie. Nous utilisons les outils de Symfony pour créer un formulaire type. Nous créons un formulaire de type "CategorieType" à partir de l'objet "Categorie". Nous observons le code généré et comprenons comment définir les champs du formulaire. Nous expliquons les différents types de champs disponibles.

4) Modifier une catégorie

Nous adaptons la vue pour ajouter un formulaire en cas de modification d'une catégorie. Nous expliquons comment les formulaires de modification fonctionnent par rapport aux formulaires de création.

c) Formulaire

1) Afficher

Dans le contrôleur, nous créons un formulaire à partir du formulaire type et le passons à la vue. Dans le template Twig, nous ajoutons le code nécessaire pour afficher le formulaire de création d'une nouvelle catégorie.

2) Soumettre le formulaire

Nous expliquons comment la soumission du formulaire fonctionne et comment cela déclenche la route correspondante pour traiter les données du formulaire.

3) Valider le formulaire

Nous abordons les différentes étapes de validation du formulaire, en mettant l'accent sur la validation côté serveur. Nous expliquons comment ajouter des contraintes de validation à l'entité Categorie à l'aide d'annotations.

d) Conclusion

Dans cette mission, nous avons appris à gérer les catégories de produits en utilisant les formulaires Symfony. Nous avons abordé la création de formulaires, la validation des données, et la protection contre les attaques CSRF. Ces compétences seront essentielles pour le développement du back office de la Ferme Maya avec Symfony.