Opérations CRUD dans Symfony 1/3

1. ## Objectifs

2. ## Présentation

   - Doctrine est un outil essentiel dans Symfony, considéré comme une pièce maîtresse du framework.
   - Cet ORM fonctionne avec des objets PHP pour représenter et manipuler les données de la base de données. Chaque entité dans Symfony est représentée par un modèle de classe qui est mappé à une table dans la base de données.
   - Les modèles de classe contiennent des méthodes pour effectuer des opérations CRUD (Créer, Lire, Mettre à jour, Supprimer) sur les données de la base de données.
   - Doctrine peut être configuré pour travailler avec différentes bases de données, simplifiant ainsi les tâches liées à la persistance des données dans Symfony.
   - Les opérations CRUD effectuées avec Doctrine permettent aux développeurs Symfony de manipuler facilement les données dans la base de données.
   - Doctrine rend simple l'exécution des opérations CRUD (Créer, Lire, Mettre à jour, Supprimer) sur les entités Symfony.
   - Il gère les opérations de création, de récupération, de mise à jour et de suppression (CRUD), ainsi que la correspondance entre les objets de modèle et les tables de base de données. Doctrine s'occupe de toutes les interactions nécessaires avec la base de données pour les opérations CRUD.

3. ## Installation des Packages Requis

   - Avant de commencer avec les opérations **CRUD**, installez les packages nécessaires en exécutant la commande suivante:`composer require form validator security-csrf`
   - Cette étape garantit que toutes les dépendances nécessaires sont installées.
   - Les deux packages que vous avez mentionnés, à savoir "**form-validator**" et "**security-csrf**", servent à renforcer la sécurité des applications web en assurant la validation des données entrantes et en protégeant contre les **attaques CSRF (Cross-Site Request Forgery)**.

     1. **Form Validator:**

- Ce package aide à valider les données entrantes dans les formulaires web. Il offre généralement des fonctionnalités pour vérifier que les données saisies par l'utilisateur correspondent aux critères définis, tels que les types de données attendus, les longueurs minimales ou maximales, les formats spécifiques, etc. Cela permet de garantir que seules les données valides sont acceptées par l'application, réduisant ainsi les risques d'injection de code malveillant ou d'erreurs de traitement.

2. **Security CSRF (Cross-Site Request Forgery):**

- Ce package fournit des mécanismes pour prévenir les attaques CSRF, qui sont des attaques où un attaquant force un utilisateur à exécuter des actions non désirées sur un site web auquel l'utilisateur est authentifié. Pour prévenir ces attaques, le package génère et vérifie des jetons CSRF (Cross-Site Request Forgery tokens) pour chaque formulaire soumis. Ces jetons sont des chaînes aléatoires uniques qui doivent être présentes dans les requêtes HTTP et qui sont vérifiées par le serveur pour garantir que la demande provient bien de l'utilisateur autorisé.

- En intégrant ces deux packages dans votre application, vous améliorez sa sécurité globale en garantissant que les données sont correctement validées et en protégeant contre les attaques CSRF, ce qui contribue à prévenir les vulnérabilités et à renforcer la confiance des utilisateurs dans votre application web.

4. EntityManager et Repository

1. **L'EntityManager?**

- L'**EntityManager** est une composante clé dans le contexte de l'**ORM (Object-Relational Mapping)**, utilisée dans des frameworks et des bibliothèques PHP telles que **Doctrine ORM**.
- L'**EntityManager** est responsable de la gestion des entités, qui sont des représentations d'objets dans votre application, et de leur interaction avec la base de données relationnelle.
- L'**EntityManager** dans Doctrine est en effet un composant central qui facilite la gestion des entités et leur synchronisation avec la base de données.
- Voici une explication détaillée des tâches pour lesquelles l'EntityManager est responsable, telles que présentées dans votre exemple :

1. **`Persist() :`**

    - La **méthode persist()** de l'**EntityManager** prend un objet et le marque comme prêt à être stocké dans la base de données.
    - Cela signifie que l'**EntityManager** commence à suivre cet objet et prépare son insertion dans la base de données lors de l'appel ultérieur à `flush()`. Par exemple :**`$entityManager->persist($product);`**

2. **`Flush() :`**

    - Après avoir persisté un ou plusieurs objets, l'EntityManager attend que vous confirmiez ces changements en appelant la méthode `flush()`.
    - Cette méthode effectue réellement les opérations de stockage dans la base de données pour tous les objets marqués avec persist(). Par exemple :**`$entityManager->flush();`**

3. **`Remove() :`**

    - Pour supprimer un enregistrement de la base de données, vous pouvez utiliser la méthode `remove()` de l'EntityManager.
    - Elle marque l'objet comme destiné à être supprimé lors de l'appel ultérieur à flush(). Par exemple :

    ```
    $entityManager->remove($product);
    $entityManager->flush();
    ```

4. **`Find() et Query() :`**

    - L'**EntityManager** peut également être utilisé pour récupérer des objets depuis la base de données.
    - Vous pouvez utiliser la méthode `getRepository()` pour obtenir un référentiel (repository) correspondant à une entité spécifique. Ensuite, vous pouvez utiliser des méthodes comme `find()` ou des requêtes (Query) pour récupérer des objets.
    - Par exemple :

    ```
    $repository = $entityManager->getRepository(Product::class);
    $product = $repository->find($productId);
    ```

apcpedagogie.com

5. En résumé, l'EntityManager joue un rôle crucial dans la gestion des entités et leur interaction avec la base de données dans Doctrine. Il simplifie les opérations de persistance, de récupération et de suppression des données, offrant ainsi une abstraction puissante pour travailler avec les données persistantes dans une application Symfony.

2. **Repository**

- Un Repository, dans le contexte de Doctrine et d'autres ORM (Object-Relational Mapping), est une classe qui agit comme un intermédiaire entre votre application et la couche de persistance, souvent une base de données relationnelle.
- Dans le contexte de Doctrine et des Repositories, voici comment vous pouvez trouver un objet par son identifiant et un ensemble d'objets en fonction de critères spécifiques, ainsi que comment écrire des requêtes personnalisées :

1. **Trouver un objet par son identifiant :**

   - Pour trouver un objet par son identifiant, vous pouvez utiliser la méthode find() du Repository en passant simplement l'identifiant de l'objet que vous recherchez. Par exemple :`$product = $productRepository->find($id);`

2. **Trouver un ensemble d'objets en fonction de critères spécifiques :**

   - Si vous souhaitez récupérer un ensemble d'objets en fonction de critères spécifiques, vous pouvez utiliser la méthode findBy() du Repository en passant un tableau associatif de critères de recherche. Par exemple, pour récupérer tous les produits ayant le nom "Keyboard" :`$products = $productRepository->findBy(['name' => 'Keyboard']);`

3. **Requêtes personnalisées :**

   - Vous pouvez écrire des méthodes de requête personnalisées dans vos Repositories pour effectuer des requêtes plus complexes. Voici un exemple de méthode personnalisée dans un Repository pour récupérer des produits par catégorie :

```
// Méthode de requête personnalisée pour récupérer des produits
par catégorie
public function findByCategory($category)
```

```
{
    return $this->createQueryBuilder('p')
        ->andWhere('p.category = :category')
        ->setParameter('category', $category)
        ->getQuery()
        ->getResult();
}
```

- Vous pouvez ensuite appeler cette méthode dans votre code pour récupérer les produits correspondant à une catégorie spécifique : `$productsInCategory = $productRepository->findByCategory('Electronics');`
- Ces exemples illustrent comment utiliser les méthodes standard fournies par les Repositories ainsi que la manière d'écrire des requêtes personnalisées pour répondre à des besoins spécifiques dans votre application.

5. Les FormTypes dans Symfony

- Un **FormType**, dans le contexte de Symfony, est une classe PHP utilisée pour définir et configurer des formulaires Web. Ces formulaires sont utilisés pour collecter des données de l'utilisateur, les valider et les soumettre pour traitement, souvent dans le cadre d'opérations CRUD (Create, Read, Update, Delete) sur des entités.
- Voici quelques points clés à retenir sur les FormTypes dans Symfony :

1. **Définition des champs de formulaire :**
   - Un FormType définit les champs qui composent le formulaire, ainsi que les options et les contraintes de validation associées à chaque champ.
   - Vous pouvez définir des champs simples tels que des champs de texte, des cases à cocher et des listes déroulantes, ainsi que des champs plus complexes tels que des collections imbriquées ou des champs personnalisés.

2. **Configuration du formulaire :**
   - Vous pouvez configurer divers aspects du formulaire à l'aide de méthodes spécifiques du FormType. Cela inclut la définition des options globales du formulaire, la gestion des événements de formulaire, la personnalisation du rendu des champs et l'ajout de fonctionnalités supplémentaires telles que la protection CSRF (Cross-Site Request Forgery).

apcpedagogie.com

3. **Liaison avec des entités :**
    - Les FormTypes sont souvent liés à des entités Symfony. Cela signifie qu'un FormType est responsable de la création de formulaires pour créer et mettre à jour des instances de ces entités.
    - Les champs du formulaire sont généralement liés aux propriétés des entités, facilitant ainsi la validation et la persistance des données.

4. **Utilisation dans les contrôleurs :**
    - Une fois défini, un FormType peut être utilisé dans les méthodes de contrôleur Symfony pour générer des instances de formulaires dans les templates Twig, pour valider les données soumises par les utilisateurs, et pour traiter les soumissions de formulaire.

> 1. *En résumé, un **FormType** dans Symfony est une composante essentielle pour la création et la manipulation de formulaires Web. Il offre une abstraction puissante pour la définition et la configuration de formulaires, facilitant ainsi la collecte, la validation et le traitement des données utilisateur dans les applications Symfony.*

Cours Symfony

Source : https://apcpedagogie.com/operations-crud-dans-symfony-1-3 · téléchargé le 07/11/2026 · © APC Pédagogie · apcpedagogie.com

apcpedagogie.com