Récupération de votre énoncé

Veuillez récupérer le dossier `eval_oop` dans le dossier `evaluation` sur Github. Mettez votre environnement en place, le site doit être fonctionnel.

Veuillez créer une base de données nommée `eval_oop` et exécutez la requête SQL ci-dessous :

```
CREATE TABLE `logement` (
  `id` int NOT NULL PRIMARY KEY AUTO_INCREMENT,
  `nom` varchar(255) CHARACTER SET utf8 COLLATE utf8_general_ci NOT NULL,
  `prix` decimal(15,2) NOT NULL,
  `image` varchar(255) CHARACTER SET utf8 COLLATE utf8_general_ci NOT NULL,
  `type` varchar(255) CHARACTER SET utf8 COLLATE utf8_general_ci NOT NULL,
  `description` varchar(255) CHARACTER SET utf8 COLLATE utf8_general_ci NOT NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb3 ROW_FORMAT=COMPACT;
```

Créer une classe abstraite Entity qui hérite de SimpleOrm

Q1 : Créer une classe abtraite `Entity` qui hérite de `SimpleOrm`, appelée `Entity.php`

Faire hériter les modeles de la Entity

Q2 : Pour chaque modèle, faire hériter le modele de la classe `Entity` de tel sorte de penser intellectuellement que le modèle est une entité. Indirectement, comme `Entity` hérite de `SimpleOrm` alors le modèle héritera toujours de `SimpleOrm`. Donc ne plus faire hériter directement les entités de `SimpleOrm`.

Mettre les propriétés en private et créer le getter et le setter pour chaque propriété

Pour éviter toute manipulation externe des propriétés des entités, nous allons créer des getters et des setters. Par exemple, pour la propriété `id` :

- créer une méthode publique `getId(): string` qui renvoie la propriété interne
- créer une méthode publique `setId(string $value): void` qui permet de changer la propriété interne

Utilisez ces getters partout où l'on lit la valeur des propriétés et les setters partout où l'on modifie la valeur des propriétés.

Créer une classe AbstractController vide

Q4 : Nous voulons classer les controllers également. Nous allons classer l'ensemble des controllers dans une classe abtraite nommée `AbstractController`.
Veuillez créer une classe abstraite `AbstractController` vide

Créer une interface CrudController

Q5 : Nous allons exiger des controllers CRUD d'implémenter une interface `CrudController` avec les méthodes adéquates à déclarer. Créer une interface `CrudController.php` avec les méthodes adéquates

Créer les controllers et les faire hériter de AbstractController et les faire implémenter CrudController

Q6 : Pour chaque controller, transofrmer les fonctions en méthodes (les mettre dans une classe) et faire hériter la classe de `AbstractController` et les faire implémenter l'interface `CrudController`. Par ailleurs, je vous rappelle que tout appel interne à une méthode doit être précédé par `$this->`.

BONUS : Renommer les fichiers et dossiers

Renommer les fichiers et les dossiers suivants:

- renommer le dossier `controllers` en `Controller` et verifier que cela fonctionne toujours (dans le cas contraire, faites les évolutions nécessaires)
- renommer le dossier `models` en `Entity` et verifier que cela fonctionne toujours (dans le cas contraire, faites les évolutions nécessaires)
- renommer le dossier `views` en `templates` et verifier que cela fonctionne toujours (dans le cas contraire, faites les évolutions nécessaires)
- renommer les fichiers des controllers et verifier que cela fonctionne toujours (dans le cas contraire, faites les évolutions nécessaires)

- mettre les fichiers `SimpleOrm.class.php` et `functions.php` dans un dossier `vendor` et verifier que cela fonctionne toujours (dans le cas contraire, faites les évolutions nécessaires)