Part 02 - Stocker et afficher

1 - Pull base de données

1.1 : Vous devriez avoir accès à une base de données MySQL (via grp ou bien XAMPP). Regardez la procédure pour accéder à votre outil PhPMyAdmin. Cela nous servira à administrer la base de données. (PhPMyAdmin n'est pas obligatoire, utiliser un autre moyen comme mysql-cli, Datagrip ou bien MySQLWorkbench peut très bien fonctionner).
Connectez vous à votre SGBD et sélectionnez la bonne base de données. Nous sommes prêt à commencer !

1.2 : Nous allons, pour le moment, nous contenter d'une seule entité pour représenter nos personnages. Nous allons donc créer une table qui suit ce schéma :

PERSONNAGE			
varchar	id	PK	
varchar	name		NOT NULL
varchar	element		NOT NULL
varchar	unitclass		NOT NULL
varchar	origin		
int	rarity		NOT NULL
varchar	url_img		NOT NULL

```
Contrairement à d'habitude, l'ID est un varchar. Je vous invite à utiliser la fonction uniqid() de php pour les générer.
```

Je vous invite à bien utiliser UTF-8 (*utf8_general_ci* par exemple) pour éviter les soucis d'accents.
De plus, veuillez à utiliser *InnoDB* comme moteur pour votre table. Nous pourrons en avoir besoin plus tard.

Essayez d'insérer un personnage avec des données cohérentes que nous pourrons afficher plus tard sur notre page web.

Dans un soucis de compatibilitée entre les jeux voici l'équivalence (les noms sont basé sur les wiki des jeux):

	Genshin	**HSR**	**ZZZ**
element	Element	CombatType	Attribute

	Genshin	HSR	ZZZ
unitclass	Weapon	Path	Speciality
origin	Region	NULL	Faction
rarity	Quality	Rarity	Rank(S=5,A=4)

```
Si vous avez choisis HSR, vous pouvez mettre le lieux phare de l'histoire du personnage si vous le connaissez
```

2 - Pull code

2.0 Il est temps de repasser sur notre projet PHP. Dans un soucis de sécurité, il est temps de mettre en place le système qui stockera nos infos de connexion.

Je vous propose d'essayer de remédier à ce problème en externalisant ces infos dans un autre fichier qui pourrait être une classe Config par exemple.

Celle-ci pourrait charger les informations à l'aide d'un fichier de configuration `.ini`.

Pour vous aiguiller, regarder la doc de la fonction `parse_ini_file`.

Voici un exemple de classe `Config` à mettre dans le dossier 📁`Config`. Le fichier peut être modifier si besoin

```php
namespace Config;
use Exception;

class Config {
    private static $param;

    // Renvoie la valeur d'un paramètre de configuration
    public static function get($nom, $valeurParDefaut = null) {
        if (isset(self::getParameter()[$nom])) {
            $valeur = self::getParameter()[$nom];
        }
        else {
            $valeur = $valeurParDefaut;
        }
        return $valeur;
    }

    // Renvoie le tableau des paramètres en le chargeant au besoin
    private static function getParameter() {
```

```
        if (self::$param == null) {
            $cheminFichier = "Config/prod.ini";
            if (!file_exists($cheminFichier)) {
                $cheminFichier = "Config/dev.ini";
            }
            if (!file_exists($cheminFichier)) {
                throw new Exception("Aucun fichier de configuration
trouvé");
            }
            else {
                self::$param = parse_ini_file($cheminFichier);
            }
        }
        return self::$param;
    }
}
```

Voici un exemple de fichier `dev.ini`

```
;config dev
[DB]
dsn = 'mysql:host=localhost;dbname=YOURDBNAME;charset=utf8';
user = 'YOUR_USERNAME';
pass = 'YOUR_PASSWORD';
```

```
N'oubliez pas de changer les placeholders dans les 3 variables !!
```

Ainsi, vous n'aurez qu'à ajouter dans le fichier `.gitignore` votre `dev.ini` et mettre un `dev_sample.ini` avec des informations standard.

L'utilisateur voulant utiliser votre projet n'aura qu'à mettre ses infos ici et renommer le fichier (très utile pour le partage ou le déploiement).

Comme les fonctions de Config sont static, vous pourrez récupérer une info avec le simple code suivant :

```
Config::get('NOM_VARIABLE_DANS_INI_FILE')
```

2.1 : Créez le fichier *models/BasePDODAO.php* suivant ce schéma :

BasePDODAC
-PDO db
#execRequest(string $sql, array $params = null) : PDOStatement|fals -getDB() : PDC

Il vous faudra coder la fonction `getDB()` : cette fonction à pour but d'instancier un objet PDO avec les infos de connexion dans l'attribut `$db` si ce dernier est null.

Puis, elle retournera simplement l'attribut `$db`. N'hésitez à vous référer à votre cours et à la doc pour l'instance de PDO. C'est ici que vous utiliserez `Config`.

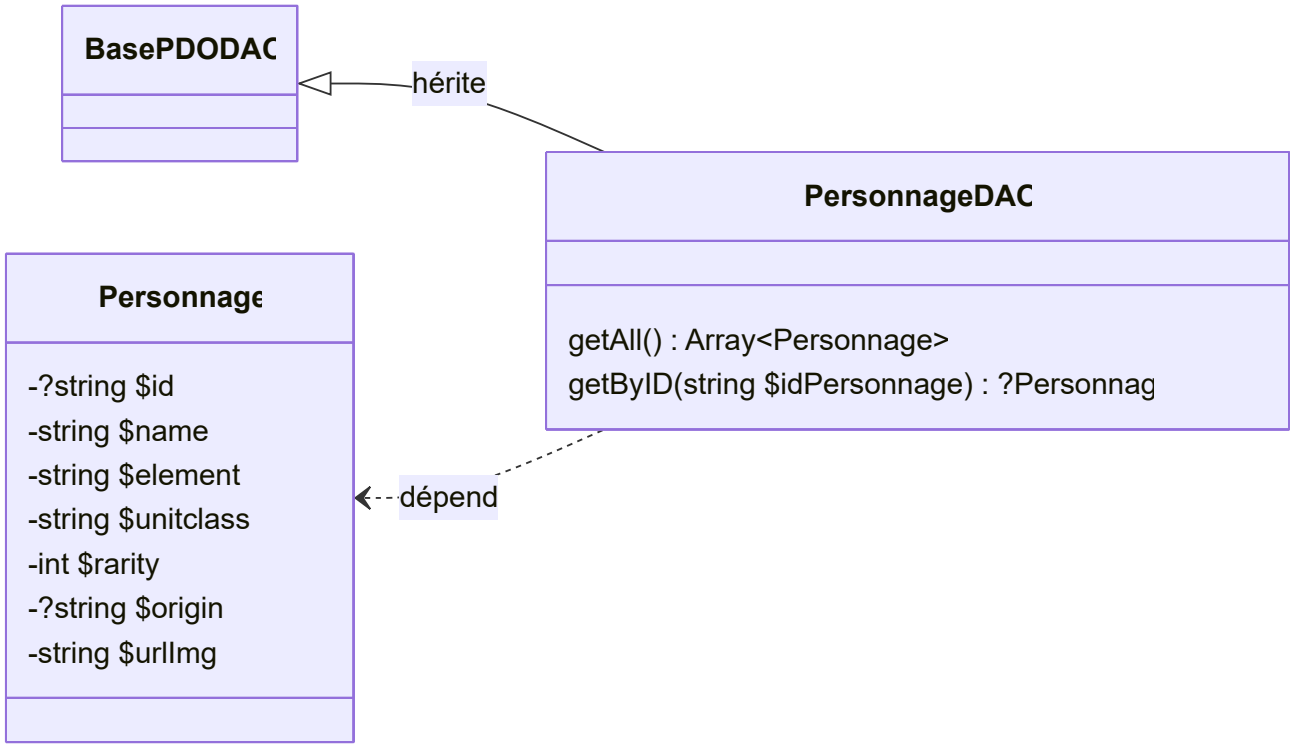
Pour la fonction `execRequest`, celle-ci a pour objectif d'exécuter la requête `$sql` passée en paramètre. Elle pourra être préparée et exécutée avec les `$params` s'ils existent (👀 `$params` à une valeur par défaut).

Notre fonction retournera le résultat de la fonction `execute` de **PDO** (qui est un *PDOStatement*).

Un peu d'aide => Voici un exemple de paramètre que notre fonction pourrait recevoir :

```
$sql = 'select * from T_COMMENTAIRE where BIL_ID=?';
$commentaires = $this->executerRequete($sql, array($idBillet));
```

2.2 : Il est temps de créer notre entité avec son DAO ! Voici le diagramme de nos classes *models/Personnage.php* et *models/PersonnageDAO.php*



Comme les attributs de la classe `Personnage` sont privés. Vous ajouterez les `Getter` & `Setter` associés.

```
  Si vous voulez implémenter l'hydratation dès maintenant, ne vous gênez pas
;)
  Cela sera demandé plus tard dans tous les cas.
```

Petite information qui pourrait vous éviter du soucis.

```
Votre hydratation ne sait pas gérer les _ dans les paramètres. Si vous lui
donné en clé url_img, il cherchera setUrl_img. Hors si votre fonction
s'appelle setUrlImg, il ne la trouvera pas. Donc soit vous améliorez votre
hydratation, soit vous ne passez pas des clé au format snake_case
```

Il vous faudra implémenter les méthodes `getAll` et `getByID` de la classe `PersonnageDAO`.

Elles ont pour vocation d'utiliser la méthode `execRequest` pour récupérer les données de la BD en retournant les données brutes.

2.3 : Maintenant que nous avons toutes nos armes pour récupérer la donnée,

il faut que le contrôleur les récupère, puis les transformes en `Personnage` ou Array de `Personnage` pour les envoyer à la vue et enfin les afficher o/

```
Vous avez surement remarqué un dossier service dans l'architecture ! C'est
ici qu'il peut servir.
```

```
Avoir un PersonnageService qui gère la transformation des données en
`Personnage`
```

Pour tester que tout fonctionne, faite une instance du DAO dans la fonction `Index`.

Sauvegarder dans 3 variables différentes le retour des fonctions `getAll()`, `getByID(idQuiExiste)` et `getByID(idQuiNexistePas)`.

Et pour finir, passez les à votre vue `Index` générée et réalisez un `var_dump` de ces variables dans le fichier `home.php`.

Vous devriez avoir une liste de `Personnage`, une `Personnage`, et `null` si tout s'est déroulé correctement (dans un format d'affichage moche au possible ;)).

```
Pour que la vue accède à une $var, n'oubliez pas de lui passer dans la
fonction render(["var" => "$var", ...])
```

Code dans `home.php`

```php
<?php var_dump($listPersonnage); ?>
<?php var_dump($first); ?>
<?php var_dump($other); ?>
```

Affichage Moche (votre navigateur peut l'afficher encore plus moche et les données affichées ne sont pas forcement à jour) :

```
C:\Dev\nyan-cat-project\src\Views\profile.php:7:
array (size=1)
  0 =>
    array (size=10)
      'id' => string '65c0ab1ad02d9' (length=13)
      0 => string '65c0ab1ad02d9' (length=13)
      'name' => string 'Kennen' (length=6)
      1 => string 'Kennen' (length=6)
      'cost' => int 1
      2 => int 1
      'origin' => string 'True Damage' (length=11)
      3 => string 'True Damage' (length=11)
      'url_img' => string 'https://cdn.mobalytics.gg/assets/tft/images/champions/thumbnail/set10/kennen.jpg?v=52' (length=85)
      4 => string 'https://cdn.mobalytics.gg/assets/tft/images/champions/thumbnail/set10/kennen.jpg?v=52' (length=85)

C:\Dev\nyan-cat-project\src\Views\profile.php:9:
array (size=10)
  'id' => string '65c0ab1ad02d9' (length=13)
  0 => string '65c0ab1ad02d9' (length=13)
  'name' => string 'Kennen' (length=6)
  1 => string 'Kennen' (length=6)
  'cost' => int 1
  2 => int 1
  'origin' => string 'True Damage' (length=11)
  3 => string 'True Damage' (length=11)
  'url_img' => string 'https://cdn.mobalytics.gg/assets/tft/images/champions/thumbnail/set10/kennen.jpg?v=52' (length=85)
  4 => string 'https://cdn.mobalytics.gg/assets/tft/images/champions/thumbnail/set10/kennen.jpg?v=52' (length=85)

C:\Dev\nyan-cat-project\src\Views\profile.php:11:
array (size=0)
  empty
```

3 - Pull design

3.1 : Il est grand temps de rendre cet affichage un peu plus classe. Sur notre page *Index*, faite afficher les données de nos personnages !

```
Vous êtes libre d'utiliser une librairie pour le CSS ou de le coder vous-même. La structure en tableau n'est pas obligatoire, si vous vous sentez de faire quelques chose de plus graphique, vous êtes libre (attention au temps tout de même)
```

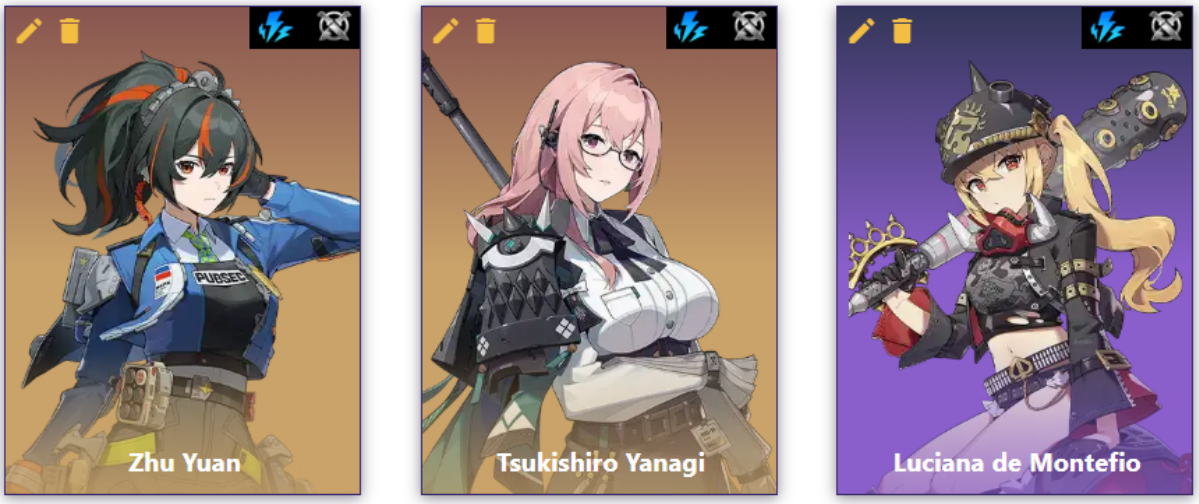
3.2 : Nous allons préparer l'avenir. Pour cela, il faudra ajouter un endroit (div/colonne) qui accueillera des options

Nous pourrons alors pour chaque personnage, ajouter un lien représenté soit par un texte, soit par une icône, soit par un bouton, etc... .

Ces derniers permettront de modifier ou supprimer une personnage !

```
Quand nous cliquons sur les liens/icons, cela doit nous envoyer sur une page vide (ou une erreur)
```

Exemple css design



Exemple avec un tableau Materialize (Données non contractuelles)

Bienvenue chez NyanCat

ID	Pokemon	Description	Types	Image	Options
1	Dracofeu	grougrou Dragon	Feu Vol		
2	Celebi	Il possède le pouvoir de voyager dans le temps. Il a été considéré comme une divinité sylvestre à différentes époques.	Psy Plante		

Bien joué si vous êtes toujours en vie jusqu'ici :D