Construction de l'API

Routes et résultats

L'API doit permettre de répondre par exemple aux demandes suivantes, suivant les verbes HTTP (pour les tests en local) :

- GET : http://localhost/rest_chocolatein/api/produit : retourne tous les enregistrements de la table produit.
- GET : http://localhost/rest_chocolatein/api/produit/5 : retourne le produit d'id 5 (s'il existe).
- DELETE : http://localhost/rest_chocolatein/api/produit/5 : supprime le produit d'id 5 (s'il existe).
- POST : http://localhost/rest_chocolatein/api/produit/{"nom":"mars glacé", "description":"barre chocolatée et caramel","packaging":"boite","gamme":"glace"} : ajoute le produit dont les caractéristiques sont données au format JSON.
- PUT : http://localhost/rest_chocolatein/api/produit/21/{"nom":"mars original", "description":"barre chocolatée et caramel","gamme":"friandise"} : modifie le produit d'id 21 avec les nouvelles valeurs données au format JSON (si l'id 21 existe).

D'une manière générale, il y a 5 possibilités d'url (à rediriger) :

- http://localhost/rest_chocolatein/api
- http://localhost/rest_chocolatein/api/nomtable
- http://localhost/rest_chocolatein/api/nomtable/num
- http://localhost/rest_chocolatein/api/nomtable/{...}
- http://localhost/rest_chocolatein/api/nomtable/num/{...}

Fonctionnement de l'application

L'entrée de l'api va se faire sur le fichier "chocolatein.php". Celui-ci va solliciter différentes méthodes de "Controle.php", suivant le verbe HTTP reçu.
"Controle.php" va solliciter "AccessBDD.php" pour demander de réaliser des traitements sur la BDD.
"AccessBDD.php" va préparer les requêtes et demander à "ConnexionPDO.php" de les exécuter.
"AccessBDD.php" va récupérer le résultat de l'exécution des requêtes et va le renvoyer vers "Controle.php" qui va retourner l'information au client (en l'affichant au format JSON).

Les différents fichiers

Tous les fichiers doivent être placés dans le dossier "api".

ConnexionPDO.php

(fichier réutilisable à récupérer et à étudier)
Le constructeur permet de se connecter à une BDD.
La methode "execute" permet d'exécuter une requête paramétrée de modification (update, insert, delete).
La méthode "queryAll" permet de retourner plusieurs lignes d'une table.
La méthode "query" permet de retourner une ligne d'une table.
Remarquez, dans le constructeur, le "catch" qui contient "throw $e" pour faire remonter l'erreur à la méthode qui sollicitera le constructeur.

AccessBDD.php

Créez le fichier AccessBDD.php qui contient la classe de même nom et qui inclut le fichier précédent. Cette classe doit préparer les requêtes à faire exécuter par ConnexionPDO.

Déclarez en privé et remplir toutes les propriétés d'accès à la BDD ($login, $bd...) ainsi que la propriété $conn initialisée à null.

Créez le constructeur qui valorise $conn avec une instance de ConnexionPDO. Cette ligne de code doit être dans un try/catch, dans le cas où la connexion échouerait, en faisant aussi un "throw $e" pour faire remonter l'erreur.

Créez la méthode "selectAll" qui reçoit en paramètre un nom de table et retourne l'appel de la bonne méthode dans ConnexionPDO en lui envoyant une simple requête pour récupérer toutes les lignes de la table concernée.

Créez la méthode "selectOne" qui reçoit en paramètre un nom de table et une valeur d'id, ajoute l'id dans un tableau associatif et retourne l'appel de la bonne méthode dans ConnexionPDO en lui envoyant la requête (pour récupérer la ligne concernée de la table) ainsi que le tableau contenant l'id.

Créez la méthode "deleteOne" qui reçoit en paramètre un nom de table et une valeur d'id, ajoute l'id dans un tableau associatif et retourne l'appel de la bonne méthode dans ConnexionPDO en lui envoyant la requête (pour supprimer la ligne concernée de la table) ainsi que le tableau contenant l'id.

Créez la méthode "insertOne" qui reçoit en paramètre un nom de table et $champs, tableau associatif contenant les noms des champs et valeurs à insérer. Cette méthode construit la chaîne de la requête d'insertion en utilisant les paramètres puis retourne l'appel de la bonne méthode dans ConnexionPDO.

Créez la méthode "updateOne" qui reçoit en paramètre un nom de table, une valeur d'id et $champs, tableau associatif contenant les noms des champs et valeurs à insérer. Cette méthode construit la chaîne de la requête de modification en utilisant les paramètres puis retourne l'appel de la bonne méthode dans ConnexionPDO.

En dehors du constructeur, toutes les méthodes doivent contrôler que l'objet de connexion n'est pas null avant de réaliser les traitements, dans le cas contraire il faut retourner null.
Pour éviter aussi les erreurs dus à des tableaux de champs vides, il faut aussi tester si ce paramètre n'est pas null dans les méthodes concernées (insertOne et updateOne).

Controle.php

Créez le fichier Controle.php qui contient la classe de même nom et qui inclut le fichier précédent. Cette classe reçoit les sollicitations du point d'entrée et exploite la classe précédente pour accéder à la BDD.

Déclarez la propriété privée $accessBDD.

Créez le constructeur qui valorise cette propriété en lui affectant une instance de la classe AccessBDD. Cette ligne de code doit être mise dans un try/catch dans le cas où la connexion échouerait et, cette fois, dans le catch, il faut appeler la méthode "reponse" en retournant un code 500, "erreur serveur" puis il faut stopper l'application avec "`die()`".

Créez la méthode privée "reponse" sur le même modèle que dans le guide.

Créez la méthode publique "get" qui sera sollicitée si des informations arrivent avec le verbe GET. Cette méthode reçoit en paramètre un nom de table et une valeur d'id (à initialiser à null dans les paramètres, dans le cas où le but est de tout récupérer). Suivant si l'id est null ou non, il faut mémoriser dans une variable locale, le résultat de l'appel de la bonne méthode (selectAll ou selectOne) dans AccessBDD, en envoyant les bons paramètres. Après cela, si la variable locale contient null ou false, il faut appeler la méthode "reponse" en envoyant juste 2 paramètres pour signaler "requete invalide". Dans le cas contraire, il faut appeler la méthode réponse, cette fois en signalant "OK" et en envoyant la variable locale (qui contient le résultat) en troisième paramètre.

Créez la méthode publique "delete" qui reçoit les mêmes paramètres, excepté que l'id ne peut pas être null. Cette méthode récupère dans une variable locale, le résultat de l'appel de la bonne méthode dans AccessBDD, en envoyant les bons paramètres. Ensuite, même logique de tests sur la variable locale excepté que, si l'opération est "OK", inutile de mettre un troisième paramètre car il n'y a rien à retourner.

Créez la méthode publique "post" sur la même logique que la précédente.

Créez la méthode publique "put" sur la même logique que la précédente.

chocolatein.php

Créez le fichier "chocolatein.php" qui va être le point d'entrée de l'API. Il ne contient pas de classe ni de méthode. Il doit être construit sur la même logique que dans le guide à quelques détails près :

Il faut inclure le fichier "Controle.php" et créer une variable $controle en lui affectant une instance de la classe correspondante.

Ensuite, il faut récupérer les paramètres reçus éventuellement dans l'URL : le nom de la table, la valeur de l'id et le contenu des champs. Faites une récupération propre pour limiter les tentatives d'injection sql. En ce qui concerne le dernier paramètre qui est au format JSON, il faut le décoder pour le transformer en tableau, de la façon suivante :

```
$contenu = json_decode($contenu, true);
```

Enfin, il faut inclure la succession de tests sur le verbe HTTP utilisé, comme vu dans le guide, mais cette fois, pour chaque verbe, il faut appeler la méthode correspondante dans la classe Contrôle, en envoyant les bons paramètres.

.htaccess

Ce fichier doit redéfinir les 5 routes citées au début de ce document, en redirigeant à chaque fois vers "chocolatein.php", en ajoutant dans l'URL les bons paramètres, lorsque cela est nécessaire.

Tests

A réaliser dans Postman.

	url	résultat

GET	Il manque la table :
	http://localhost/rest_chocolatein/api/
	Nom de table incorrect :
	http://localhost/rest_chocolatein/api/table
	id inexistant :
	http://localhost/rest_chocolatein/api/produit/28
		<pre>{
		"code": 400,
		"message": "requete invalide",
		"result": ""
		}</pre>

GET	http://localhost/rest_chocolatein/api/produit/3	<pre>{
		"code": 200,
		"message": "OK",
		"result": {
		"id": "3",
		"nom": "Boite de luxe",
		"description": "Boite de luxe",
		"packaging": "de 4 à 49 chocolats",
		"urlimg": "./vues/images/produits/chocolats/luxe",
		"gamme": "chocolats"
		}
		}</pre>

GET	http://localhost/rest_chocolatein/api/produit	{
		"code": 200,
		"message": "OK",
		"result": [
		... // 26 résultats entre accolades
		]
		}

POST	http://localhost/rest_chocolatein/api/produit/{"nom":"mars glacé", "description":"barre chocolatée et caramel","packaging":"boite","gamme":"glace"}	{
		"code": 200,
		"message": "OK",
		"result": ""
		}
		Vérifier l'ajout en 27 dans la BDD (avec le champs "urlimg" à null) ou en faisant une requête GET sur 27 dans Postman (qui doit retourner la ligne).

PUT	http://localhost/rest_chocolatein/api/produit/27/{"nom":"mars original","gamme":"friandise"}	{
		"code": 200,
		"message": "OK",
		"result": ""
		}
		Vérifier les 2 modifications en 27 dans la BDD ou en faisant une requête GET sur 27 dans Postman (qui doit retourner la ligne).

DELETE	http://localhost/rest_chocolatein/api/produit/27	{
		"code": 200,
		"message": "OK",
		"result": ""
		}
		Vérifier la ligne a été supprimée dans la BDD ou en faisant une requête GET sur 27 dans Postman (qui doit retourner une erreur).

PUT	Modification impossible car id inexistant :
	http://localhost/rest_chocolatein/api/produit/27/{"nom":"mars original","gamme":"friandise"}
		{
		"code": 400,
		"message": "requete invalide",
		"result": ""
		}

DELETE	Suppression impossible car id inexistant :
	http://localhost/rest_chocolatein/api/produit/27
		{
		"code": 400,
		"message": "requete invalide",
		"result": ""
		}

Essayez aussi de mettre un nom de BDD incorrect dans la propriété correspondante dans AccessBDD et faites un GET qui est censé marcher. Vous devriez obtenir :

```
{
    "code": 500,
    "message": "erreur serveur",
    "result": ""
}
```