Test technique - Stage de développeur backend

Objectif :

Développement d'une API REST nommée **SubNauticApp** qui permet la gestion de bases nautiques.

Spécifications techniques :

Entité à implémenter :

Routes de l'API :

Voici la liste des routes API qu'il faut développer :

- Créer une base nautique
- Liste des bases nautiques : chaque élément de la liste ne doit contenir que son nom et sa description
- Récupération des détails d'une base nautique
- Mettre à jour une base nautique
- Supprimer une base nautique

Appréciation :

Une fois le test terminé, envoie-nous à l'adresse dev@guidap.co le lien du repository Git (GitHub, Gitlab, etc.) sur lequel tu auras déposé ta solution.

Impératifs :

- Développer l'API REST avec **Symfony 3.4** sous **PHP 7.2**
- Respecter les spécifications techniques
- Utiliser Doctrine pour stocker les bases nautiques dans une base de données MySQL ou MariaDB (disponible sous XAMP/MAMP/WAMP ou gratuitement en ligne)
- Respecter les conventions PSR-1 et PSR-2
- Respecter les règles pour implémenter une API REST (cf en bas de ce document)
- Documenter dans le fichier *README.md* comment nous devons installer et tester ton projet
- Utilise Git pour versionner ta solution

Aucune limite de temps n'est imposée.

Bonus optionnels :

- Paginer la liste des bases nautiques
- Développer une interface simple qui dialogue avec l’API (ça c’est du bonus++, seulement si t’as envie de montrer ton niveau de dev fullstack 😉)

Règles à suivre pour la réalisation d'une API REST

Règle n°1 : l'URI comme identifiant des ressources

REST se base sur les **URI (Uniform Resource Identifier)** afin d'identifier une ressource. Ainsi une application se doit de construire ses URI (et donc ses URL) de manière précise, en tenant compte des contraintes REST. Il est nécessaire de prendre en compte la hiérarchie des ressources et la sémantique des URL pour les éditer :

Quelques exemples de construction d'URL avec RESTful :

Liste des livres

NOK : http://mywebsite.com/book

OK : http://mywebsite.com/books

Filtre et tri sur les livres

NOK : http://mywebsite.com/books/filtre/policier/tri/asc

OK : http://mywebsite.com/books?filtre=policier&tri=asc

Affichage d'un livre

NOK : http://mywebsite.com/book/display/87

OK : http://mywebsite.com/books/87

Tous les commentaires sur un livre

NOK : http://mywebsite.com/books/comments/87

OK : http://mywebsite.com/books/87/comments

Affichage d'un commentaire sur un livre

NOK : http://mywebsite.com/books/comments/87/1568

OK : http://mywebsite.com/books/87/comments/1568

En construisant correctement les URI, il est possible de les trier, de les hiérarchiser et donc d'améliorer la compréhension du système.

L'URL suivante peut alors être décomposée logiquement :

*http://mywebsite.com/books/87/comments/1568 => **un commentaire pour un livre***

*http://mywebsite.com/books/87/comments => **tous les commentaires pour un livre***

*http://mywebsite.com/books/87 => **un livre***

*http://mywebsite.com/books => **tous les livres***

Règle n°2 : les verbes HTTP comme identifiant des opérations

La seconde règle d'une architecture REST est d'utiliser les verbes HTTP existants plutôt que d'inclure l'opération dans l'URI de la ressource. Ainsi, généralement pour une ressource, il y a 4 opérations possibles (CRUD) :

- Créer (create)
- Afficher (read)
- Mettre à jour (update)
- Supprimer (delete)

HTTP propose les verbes correspondant :

- Créer (create) => **POST**
- Afficher (read) => **GET**
- Mettre à jour (update) => **PUT**
- Supprimer (delete) => **DELETE**

Exemple d'URL pour une ressource donnée (un livre par exemple) :

Créer un livre

NOK : GET http://mywebsite.com/books/create

OK : POST http://mywebsite.com/books

Afficher

NOK : GET http://mywebsite.com/books/display/87

OK : GET http://mywebsite.com/books/87

Mettre à jour

NOK : POST http://mywebsite.com/books/editer/87

OK : PUT http://mywebsite.com/books/87

Supprimer

NOK : GET http://mywebsite.com/books/87/delete

OK : DELETE http://mywebsite.com/books/87