Part 06 - Vos papiers svp

L'objectif de cette partie sera d'améliorer notre routeur pour avoir des routes protégés. Il sera donc nécessaire d'être authentifié pour y accéder.

Authentification

1 - Stocker nos papier

1.1 Pour mettre en place tout cela, il nous faudra stocker nos utilisateurs. C'est pour cela que nous allons ajouter un modèle à notre BD.

On restera très simple et fonctionnel.

USERS			
varchar	id	PK	
varchar	username		NOT NULL
varchar	hash_pwd		NOT NULL

1.2 Comme le but n'est pas de créer toute l'administration de nos utilisateurs, je vous invite à utiliser SQL pour créer au moins un ou 2 utilisateurs.

Pour générer les hash et uid, j'ai utilisé un PHP sandbox en ligne avec ce code :

```
var_dump(password_hash("test", PASSWORD_DEFAULT)) ;

var_dump(password_hash("admin", PASSWORD_DEFAULT)) ;

var_dump(uniqid()) ;

var_dump(uniqid()) ;
```

```
Vous remarquerez que j'ai utilisé PASSWORD_DEFAULT. C'est pour rester
compatible partout mais vous êtes libre d'utiliser un autre algo
```

1.3 Sans avoir besoin de fioritures, je vous invite à créer le model `USERS` et le DAO `USERSDAO`. Le DAO peut se contenter d'avoir une fonction `getByUsername(string $username)`

Nous avons maintenant de quoi récupérer notre modèle. Je vous invite à le tester rapidement sur une page vierge si besoin.

2 - Scanner nos papier

2.1 Maintenant nous allons attaquer la page permettant de se connecter. Créez une vue `login.php` dans le dossier `views`.

Celle-ci devras surtout avoir un formulaire avec un champs pour l'username ainsi que pour le password. L'action visera la route `login` en méthode post.

L'obejctif étant d'avoir une page protégé, créez une 2ème vue (peut importe le nom donc ici j'utiliserai `protected.php`). Son contenu n'as pas d'importance, elle sera juste ici pour prouver qu'on peut avoir une page qui ne s'affiche qu'une fois connecté un plus loin dans le sujet.

Vous pouvez aussi créer votre `AuthController` pour avoir des fonctions qui afficherons vos templates.

2.2 Il est temps de créer notre AuthService. Celui-ci aura au moins 1 fonction. Vous pourrez en ajouter d'autres plus tard

```php
public static function login($username, $password): void
{
    // Récupère le user avec son username
    // Vérifie que le password fourni est correct
    // Crée des variable session userUID et timeout
    // A vous de déterminer le temps que la connexion est valide
}
```

Cette fonction sera donc appelé dans le AuthController dans une fonction qui fera le login et redirigera sur l'index ou bien sur le formulaire de connexion (réussite / echec).

2.3 Il est temps de créer 2 nouvelles routes ! Login et logout ! Pour la plus facile, logout n'as rien de précis à faire d'autre qu'appeler une fonction du `AuthController`.

Celle-ci aura pour but de supprimer les variable de `$_SESSION` allouées à la connexion, puis d'afficher l'index avec un message de confirmation.

Pour la route login, nous avons donc le `get` qui affiche la page de login à l'aide du `AuthController`.

Puis le `post`, qui récupère les données du formulaire puis fais la passe au `AuthController`.

La procédure reste donc la même que pour les autres routes liées à un formulaire. Il nous reste donc à pouvoir accéder facilement à ses pages. Direction la NAV !

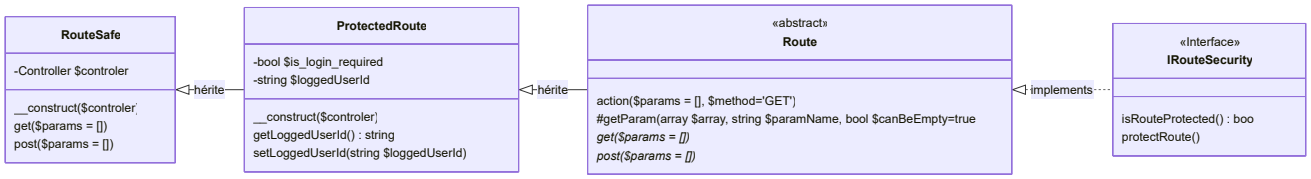
Dans le fichier view qui gère votre barre de navigation, ajoutez un lien vers la page de login. Cependant, si nous sommes déjà loggué, il devrait alors afficher un lien vers logout.

Dans le but de tester plus tard notre route protégé, ajoutez aussi un lien vers celle-ci (Qui ne devrait pas fonctionner si vous n'ajoutez pas une route au routeur).

A ce stade, nous avons nos armes pour passer à l'évolution de notre système de routeur o/

3 - Une orientation protégé

3.1 Il est temps de mettre à jour notre routeur et surtout d'ajouter un nouveau type de route. Dans le diagramme suivant, il faudra créer l'interface `RouteSecurity`, et la classe `ProtectedRoute`. Ensuite, toutes nos routes dont on voudra ajouter la protection, devra hériter de `ProtectedRoute` au lieu de Route.



Pour ce qui est de l'implementation des fonctions de l'interface, `isRouteProtected()` retourne vrai (ProtectedRoute) ou faux (Route).

Puis, `ProtectedRoute()` ne fait rien pour Route, alors que pour ProtectedRoute, vérifie que la session possède bien un UID valide et le cas échéant, lève les exceptions utiles.

3.2 Nous avons maintenant des routes protégées, mais nous n'appelons jamais cette protection dans notre process pour empêcher l'accès aux routes.

Il vous faudra donc ajouter une étape dans la fonction `routing()` de votre Routeur.

Celle-ci devra donc appeler la méthode `protectRoute()` de la route selectionné puis, si exception se produit, gérer l'erreur pour empêcher l'accès. Si tout va bien, alors le process ce déroule normalement.

```
En cas de non connexion de l'utilisateur, je vous invite à redirgier vers la page de login avec un petit message !
```

4 - Application ! Collection !

Maintenant que nous avons des pages qui demande une connexion et donc une identification. Il est possible d'ajouter la fonction de collection de nos personnages !

Rien ne devrais être compliquer à réaliser mais voici les différentes actions à entreprendre :

- Ajouter un bouton +/- qui permet d'ajouter/supprimer un perso de la collection une fois connecté.
- Avoir une page utilisateur qui fais un recap de tous les persos possédé par l'utilisateur (Avec le bouton -)

Et avec cette fonctionnalité, cela peut conclure ce projet de site `Collection de personnage du Hoyoverse` !

Rien ne vous empêche d'aller plus loin pour vous entrainer.

```
Félicitation d'être arrivé jusqu'ici ! Vous avez bien travaillez
```