Site de rencontre L😍veLink partie2

I. 📎 Contexte

L'équipe de développement de Lovelink a reçu un audit de code par l'équipe chargée de la qualité. Celui-ci nous préconise de faire du refactoring pour utiliser la programmation orientée objet (POO), afin de le rendre plus **modulaire**, **réutilisable** et **maintenable**. Cette approche facilitera les futures évolutions et corrections du projet.

II. ⚙️ Installation du projet

Si votre siteRencontre du précédent TP est abouti et sans erreur, repartez de votre TP, sinon récupérez les sources du professeur ici :

1. Lancer gitBash
2. Se positionner dans votre répertoire web : `cd "c:\wamp64\www\"`

`(Remarquez l'utilisation des guillemets dans le chemin, car GitBash fonctionne sous Linux, où les chemins utilisent des slashs / au lieu des anti-slashs \.`
`Nous pouvons donc pouvoir accéder au répertoire en tapant la commande : cd /c/wamp64/www)`

3. Lancer la commande `git clone` qui permet de récupérer les sources en local depuis un repository distant :
   `git clone https://github.com/StephanieANDRES/siteRencontre_part2.git`
4. Lancer VScode
5. Dans VSCode, faites `File>>Open Folder>>` et aller chercher le repository `c:\wamp64\www\siteRencontre_part2`
6. Lancer le site : http://localhost/siteRencontre_part2/

III. 🔍 Création des classes

Le premier conseil de l'expert est de commencer par observer notre **base de données**. En effet, chaque table de la base de données peut correspondre à une **classe** dans notre code.

1. **Lister toutes les tables** présentes dans notre base de données.

Étape 1 : La classe Utilisateur

Vous allez être guidé pour réaliser la classe Utilisateur. Vous devrez ensuite faire de même pour les autres classes.

2. **Créer** la classe Utilisateur.

```php
<?php
class Utilisateur {
	// Classe vide pour l'instant
}
?>
```

Pour rappel, par convention une classe est nommée en **PascalCase** :

Nom de la case	Variable	Constante	Fonction	Classe	Fichier	Slug d'URL	Attribut HTML
`camelCase`	✅	❌	✅	❌	❌	❌	❌
`PascalCase`	❌	❌	❌	✅	❌	❌	❌
`snake_case`	✅	❌	✅	❌	✅	❌	❌
`kebab-case`	❌	❌	❌	❌	✅	✅	✅
`UPPERCASE`	❌	✅	❌	❌	❌	❌	❌

✅ Idéal ❌ Déconseillé

Étape 2 : Définition des propriétés

Nous allons transformer nos *attributs* (colonnes) de table en *propriétés* de classe.

3. Les *attributs* (colonnes) de la table Utilisateur sont : *id, nom, prenom, pseudo, email, mot_de_passe, age, genre, preference, bio, localisation, date_inscription*
   - Les propriétés de nos classes porteront le même nom que les attributs des tables
   - Les propriétés sont privées à la classe
   - Le constructeur permet de construire l'objet

4. Finir de créer les *propriétés* et le *constructeur* de la classe Utilisateur

```php
<?php
class Utilisateur {
	private $id;
	private $nom;
```

```
        ...

        public function __construct(...) {
        $this->id = $id;
        $this->nom = $nom;
        ...
        }
}
?>
```

Étape 3 : Identification et implémentation des méthodes

Les ***méthodes*** permettent d'agir sur les données de la classe. Il faut se poser la question “Quelles sont les actions d’un utilisateurs ?”.

Pour lister toutes les méthodes nécessaires, il faut connaître les fonctionnalités de notre site de rencontre. Ou bien lire le code.

Concentrons-nous d’abord sur la page `connexion.php`. Un utilisateur peut se connecter au site.

5. Donc commençons par créer la méthode `seConnecter()`.

```
<?php
class Utilisateur {
        private $id;
        private $nom;
        ...

        public function __construct(...) {
        ...
        }

        public function seConnecter() {
        // Code pour se Connecter sur le site
        }

}
?>
```

6. Le code de la page de *connexion.php* à adapter dans notre classe serait donc celui-ci :

```php
$stmt = $pdo->prepare("SELECT * FROM utilisateurs WHERE email = :email");
$stmt->bindParam(':email', $email, PDO::PARAM_STR);
$stmt->execute();
$user = $stmt->fetch();

//Si l'email existe et que le mot de passe correspond, on est loggué et
redirigé avec la page index
if ($user && password_verify($mot_de_passe, $user['mot_de_passe'])) {
   $_SESSION['user_id'] = $user['id'];
   header('Location: index.php');
} else {
   //Sinon on affiche une erreur
   $erreur = "Identifiants incorrects";
}
```

Récupérons ce bout de code et collons-le dans la classe. Constatons les erreurs.

```php
public function seConnecter() {
    $stmt = $pdo->prepare("SELECT * FROM utilisateurs WHERE email = :email");
    $stmt->bindParam(':email', $email, PDO::PARAM_STR);
    $stmt->execute();
    $user = $stmt->fetch();

    //Si l'email existe et que le mot de passe correspond, on est loggué et rec
    if ($user && password_verify($mot_de_passe, $user['mot_de_passe'])) {
        $_SESSION['user_id'] = $user['id'];
        header('Location: index.php');
    } else {
        //Sinon on affiche une erreur
        $erreur = "Identifiants incorrects";
    }
}
```

- les variables $pdo, $email, $mot_de_passe n'existent pas : c'est normal.
    - Ajoutons-les en paramètre de la méthode `seConnecter($pdo, $email, $mot_de_passe)`

A ce stade, nous n'avons plus d'erreur mais maintenant il nous faut changer le comportement de la connexion.
En effet, il ne devrait pas être de la responsabilité de la classe Utilisateur de faire des redirections sur les pages du site, ni de mettre à jour la variable de session .

La méthode `seConnecter()` devrait plutôt renvoyer un l'objet de type ***Utilisateur*** si le login et mot de passe sont corrects ou ***null*** dans le cas contraire.

Voici l'implémentation de la méthode :

```php
public function seConnecter($pdo, $email, $mot_de_passe) {
   $stmt = $pdo->prepare("SELECT * FROM utilisateurs WHERE email = :email");
```

```
    $stmt->bindParam(':email', $email, PDO::PARAM_STR);
    $stmt->execute();
    $user = $stmt->fetch();

    //Si l'email existe et que le mot de passe correspond, on est loggué et
redirigé avec la page index
    if ($user && password_verify($mot_de_passe, $user['mot_de_passe'])) {
              return $user;
         }
         return null;
     }
```

7. L'objet PDO

Nous pourrions encore améliorer un peu notre classe. Telle que nous l'avons construite, nous allons devoir passer l'objet `$pdo` en paramètre de chaque méthode de la classe Utilisateur. Pas très pratique…
Nous devrions plutôt créer une propriété `$pdo` que l'on initialiserait lors de la création d'un objet. Ainsi pour utiliser un objet de type Utilisateur, nous aurions besoin de passer le `$pdo` qu'une seule fois, au moment de l'instanciation.

Pour cela nous allons rajouter la propriété `$pdo` à la classe Utilisateur. Nous devons également ajouter le `$pdo` dans notre constructeur.

- Ajoutez la propriété pdo à la classe Utilisateur

```
private $pdo;
```

Le problème est que notre constructeur, pour être appelé, nécessite qu'on lui renseigne tous les paramètres (toutes les propriétés de la classe). Lorsque nous allons appeler la méthode `seConnecter()`, notre objet de type Utilisateur sera vide, seule la propriété $pdo sera connue.

Pour gérer cette problématique, normalement nous devrions créer un second constructeur à notre classe, avec une *signature* différente (paramètres attendus). Mais ceci n'est pas possible en PHP.

Une possibilité pour résoudre ce problème est de spécifier que chaque paramètre de notre constructeur peut être *null*.

- Modifiez le constructeur :

```
public function __construct($pdo, $id = null, $nom = null, $prenom =
null, $pseudo = null, $email = null, $motDePasse = null, $age = null,
$genre = null, $preference = null, $bio = null, $localisation = null,
$dateInscription = null) {
 ...
```

```
}
```

- Modifiez la méthode `seConnecter()` pour qu'elle fasse appel à la propriété `$this->pdo`.
- Retirez le paramètre `$pdo` de notre méthode `seConnecter()`

```
$stmt = $this->pdo->prepare("SELECT * FROM utilisateurs WHERE email =
:email");
...
$user = $stmt->fetch();

if ($user && password_verify($mot_de_passe, $user['mot_de_passe'])) {
   return $user;
}
 return null;
```

8. Maintenant que nous avons implémenté notre méthode `seConnecter()`, modifions le code de la page connexion.php pour appeler notre méthode.

- Inclure la classe Utilisateur

```
include 'includes/db.php';
include 'class/Utilisateur.php'; // Inclure la classe Utilisateur
```

- Créer un objet Utilisateur

```
$utilisateur = new Utilisateur($pdo);
```

- Appeler la méthode `seConnecter()` avec les paramètres appropriés

```
$user = $utilisateur->seConnecter($email, $mot_de_passe);
if($user!=null){
   $_SESSION['user_id'] = $user['id'];
   header('Location: index.php');
   exit(); // Utilisez exit() après header() pour arrêter l'exécution du
script
}else{
   $erreur = "Identifiants incorrects";
}
```

- Tester la connexion 👍

9. Sur le modèle de la méthode `seConnecter()`; créer la méthode `sInscrire()` et modifier la page inscription en conséquence.

IV. 👨‍💻 La classe Message

10. Sur le modèle de la classe ***Utilisateur***, créer la classe ***Message***.
Adapter le code de la page messagerie.php en fonction de votre nouvelle classe.

⚠ Si vous êtes reparti des sources git siteRencontre_part2, la date de naissance et le flag Lu/Non lu n'existent pas dans cette version. Mettez vos attributs non-obligatoires dans votre BDD pour éviter les problèmes.