

FORMATION PHP:

PHP OBJET ET DESIGN PATTERNS

COURS 3



1. Introduction et QCM
2. Révision exercice cours précédent
3. Programmation Orientée Objet (POO) en PHP
 1. Définitions et concepts de la POO (attributs, méthodes, visibilité)
 2. Héritage, classes abstraites, interfaces, et traits
 3. Utilisation de namespaces pour l'organisation du code
 4. Gestion des Exceptions
 1. Syntaxe try-catch-finally et gestion des erreurs
 5. Accesseurs, Mutateurs, et Méthodes Magiques
 6. Getters, setters, constructeurs (__construct) et destructeurs (__destruct)
 7. Propriétés et méthodes statiques, constantes de classe
4. Introduction aux Design Patterns
5. Patterns Singleton, Factory, et MVC
 1. Structure et application du pattern MVC
6. Exercice Pratique – Mini-projet avec Design Pattern
7. QCM
8. Présentation de projet/groupes

EXERCICE

1. Utilisez phpMyAdmin pour créer une table « utilisateurs » avec les colonnes id, nom, password .
2. Créez également une table commandes via un script PHP pour renforcer vos compétences.
3. Structure des fichiers :

```
auth-service/|
|— config/
|   |— database.php|
|— functions/
|   |— user_functions.php
|   |— auth_functions.php|
|— services/
|   |— auth_service.php
|   |— product_service.php (simulé)
|   |— order_service.php (simulé)|
|— api_gateway.php|
|— index.php
```

4. Configuration de la base de données
 - i. Complétez le fichier config/database.php avec vos informations de connexion
5. Implémentation des fonctions d'authentification
 - I. Complétez les fonctions dans functions/auth_functions.php et services/auth_service.php
 - II. Ajoutez une vérification pour voir si l'utilisateur existe déjà avant l'enregistrement
 - III. Implémentez la génération d'un token JWT simple pour la fonction de connexion
6. Test de l'API
 - I. Utilisez un outil comme Postman ou cURL pour tester les endpoints /auth avec les actions 'login' et 'register'
 1. Pour s'enregistrer :POST <http://votre-serveur/auth>
 1. action=register&username=test&password=password123
 2. Pour se connecter :POST <http://votre-serveur/auth>
 1. action=login&username=test&password=password123

INTRODUCTION À LA PROGRAMMATION ORIENTÉE OBJET EN PHP

Définition de la POO :

Paradigme de programmation basé sur la modélisation d'entités sous forme d'objets qui regroupent données (attributs) et comportements (méthodes).

Pourquoi la POO ?

- Organisation et lisibilité : Code structuré autour d'objets représentant des entités réelles.
- Réutilisabilité : Possibilité de réutiliser des classes et d'étendre leurs fonctionnalités.
- Maintenabilité : Facilite la modification et la mise à jour du code.

INTRODUCTION À LA PROGRAMMATION ORIENTÉE OBJET EN PHP



```
class Voiture {  
    // Propriétés de la classe  
    public $couleur;  
    public $marque;  
  
    // Constructeur : méthode spéciale appelée automatiquement à la création d'un objet  
    public function __construct($couleur, $marque) {  
        // Initialisation des propriétés avec les valeurs passées en argument  
        $this->couleur = $couleur;  
        $this->marque = $marque;  
    }  
  
    // Méthode pour démarrer la voiture  
    public function demarrer() {  
        echo "La voiture $this->marque de couleur $this->couleur démarre !";  
    }  
}  
  
// Instanciation de l'objet avec les valeurs des propriétés passées au constructeur  
$maVoiture = new Voiture("rouge", "Toyota");  
  
// Appel de la méthode "demarrer" de l'objet "maVoiture"  
$maVoiture->demarrer(); // Affiche : La voiture Toyota de couleur rouge démarre !
```

INTRODUCTION À LA PROGRAMMATION ORIENTÉE OBJET EN PHP

Récapitulatif des visibilitées en PHP :

- **public** : Accessible de n'importe où, à l'intérieur et à l'extérieur de la classe.
- **private** : Accessible uniquement à l'intérieur de la classe elle-même.
- **protected** : Accessible à l'intérieur de la classe et des classes qui en héritent (sous-classes).

Remarque sur la classe elle-même

Pour la classe en PHP, on n'utilise donc pas **public** ni **private**.

Une classe sera accessible partout par défaut, sauf si elle est contenue dans un namespace ou utilise un mécanisme d'autoloading, mais cela relève d'une gestion plus avancée des classes et des dépendances.

INTRODUCTION À LA PROGRAMMATION ORIENTÉE OBJET EN PHP



```
class Voiture {  
    public $marque;    // Propriété publique, accessible depuis l'extérieur de la classe  
    private $couleur; // Propriété privée, uniquement accessible au sein de la classe  
  
    public function __construct($marque, $couleur) {  
        $this->marque = $marque;  
        $this->couleur = $couleur;  
    }  
  
    public function demarrer() {  
        echo "La voiture $this->marque démarre !";  
    }  
  
    private function afficherCouleur() {  
        echo "Couleur : $this->couleur";  
    }  
}
```

INTRODUCTION À LA PROGRAMMATION ORIENTÉE OBJET EN PHP



```
class Animal {  
    public $nom;           // Accessible depuis l'extérieur  
    protected $espece;    // Accessible dans la classe et ses sous-classes  
    private $age;         // Accessible uniquement dans cette classe  
  
    public function __construct($nom, $espece, $age) {  
        $this->nom = $nom;  
        $this->espece = $espece;  
        $this->age = $age;  
    }  
  
    public function getAge() {  
        return $this->age; // Méthode publique pour accéder à une propriété privée  
    }  
}  
  
$animal = new Animal("Max", "Chien", 5);  
echo $animal->nom;           // OK, $nom est public  
// echo $animal->espece;    // Erreur, $espece est protected  
// echo $animal->age;       // Erreur, $age est private  
echo $animal->getAge();      // OK, retourne 5 via une méthode publique
```


INTRODUCTION AUX NAMESPACES EN PHP

Les « namespaces » en PHP sont un mécanisme qui permet d'organiser le code et d'éviter les conflits de noms entre les classes.

Pourquoi utiliser des namespaces ?

- Éviter les collisions de noms : Dans une grande application, il est courant d'avoir plusieurs classes ou fonctions avec le même nom. Les namespaces permettent de les différencier en les regroupant sous un "espace de noms" unique.
- Organiser le code : Les namespaces aident à structurer le code en le regroupant logiquement. Cela facilite la compréhension et la maintenance de l'application.
- Interopérabilité : Lorsqu'on utilise des bibliothèques tierces, il est possible que les noms des classes ou des fonctions entrent en conflit. Les namespaces aident à éviter ces conflits en isolant le code.



```
namespace MonProjet\Module;

class MaClasse {
    public function maMethode() {
        echo "Bonjour du namespace MonProjet\Module !";
    }
}
```

UTILISATION DES NAMESPACES

Pour utiliser une classe d'un namespace dans un autre fichier, on peut faire appel à son nom complet ou utiliser l'instruction « **use** » :

- Permet d'importer une classe, une fonction ou une constante d'un namespace afin de pouvoir l'utiliser sans avoir à spécifier le namespace complet à chaque fois.



```
use MonProjet\Module\MaClasse;  
  
$objet = new MaClasse(); // Utilisation simplifiée
```

- **include** : Est utilisé pour inclure des fichiers PHP, mais ne gère pas les namespaces. Cela peut être utilisé avec des fichiers contenant des classes, mais n'affecte pas l'importation des namespaces.

CONCEPTS AVANCÉS DE LA POO

Héritage :

- Permet à une classe (sous-classe) d'hériter des propriétés et méthodes d'une autre classe (super-classe).
- Cela favorise la réutilisation du code.

```
class Animal {  
    public function parler() {  
        echo "L'animal fait du bruit !";  
    }  
    // Méthode pouvant être utilisée par les classes héritantes  
    public function manger() {  
        echo "L'animal mange !";  
    }  
}
```


```
class Chien extends Animal {  
    public function parler() {  
        echo "Le chien aboie !";  
    }  
}  
  
class Chat extends Animal {  
    public function parler() {  
        echo "Le chat miaule !";  
    }  
}
```

```
// Instanciation des sous-classes  
$monChien = new Chien();  
$monChat = new Chat();  
  
$monChien->parler(); // Affiche : Le chien aboie !  
$monChien->manger(); // Affiche : L'animal mange !  
  
$monChat->parler(); // Affiche : Le chat miaule !  
$monChat->manger(); // Affiche : L'animal mange !
```

CONCEPTS AVANCÉS DE LA POO

Interfaces :

- Définit un contrat que les classes doivent suivre.
- Une classe peut implémenter plusieurs interfaces, ce qui permet d'assurer une certaine structure.



```
// Déclaration de l'interface
interface IAnimal {
    public function parler();
}

// Classe Chien qui implémente l'interface
class Chien implements IAnimal {
    public function parler() {
        echo "Le chien aboie !";
    }
}

// Classe Chat qui implémente l'interface
class Chat implements IAnimal {
    public function parler() {
        echo "Le chat miaule !";
    }
}
```

CONCEPTS AVANCÉS DE LA POO

Classe Abstraite:


- Une classe abstraite est une classe conçue pour servir de modèle pour d'autres classes et ne peut pas être instanciée directement.
- Elle peut contenir des méthodes concrètes (qui ont une implémentation) et des méthodes abstraites (sans implémentation).
- Les méthodes abstraites sont obligatoirement définies dans chaque classe qui hérite de la classe abstraite.

Utilité des Classes Abstraites :

- Elles définissent une structure de base pour les classes qui vont en hériter, imposant des méthodes communes.
- Permettent de centraliser certaines fonctionnalités tout en laissant les classes enfants implémenter des comportements spécifiques.

CONCEPTS AVANCÉS DE LA POO

Exemple Classe Abstraite:



```
abstract class Animal {
    abstract public function crier(); // Méthode abstraite, sans implémentation

    public function seDeplacer() {
        echo "L'animal se déplace";
    }
}

class Chien extends Animal {
    public function crier() {
        echo "Aboyer !";
    }
}

$chien = new Chien();
$chien->crier(); // Affiche : Aboyer !
$chien->seDeplacer(); // Affiche : L'animal se déplace
```

CONCEPTS AVANCÉS DE LA POO

Trait:

- Les traits en PHP permettent de réutiliser des méthodes dans plusieurs classes, sans utiliser l'héritage.
- Ils sont utiles quand plusieurs classes ont besoin des mêmes fonctionnalités, mais ne partagent pas de relation parent-enfant.

```
trait UtiliseGPS {  
    public function obtenirPosition() {  
        echo "Position obtenue via GPS.";  
    }  
}  
  
class Voiture {  
    use UtiliseGPS;  
}  
  
class Bateau {  
    use UtiliseGPS;  
}  
  
// Instanciation  
$voiture = new Voiture();  
$bateau = new Bateau();  
  
$voiture->obtenirPosition(); // Affiche : Position obtenue via GPS.  
$bateau->obtenirPosition(); // Affiche : Position obtenue via GPS.
```

GESTION DES EXCEPTIONS EN PHP

Introduction aux exceptions :

- Une exception est une façon de gérer les erreurs dans le code, en permettant d'interrompre le flux normal de l'application lorsqu'une erreur se produit et d'y répondre de manière contrôlée.
- Syntaxe de base :
 - try : Bloc de code où l'on teste des opérations susceptibles de générer une exception.
 - catch : Bloc de code exécuté si une exception est levée dans le bloc try.
 - finally (optionnel) : Code exécuté après try et catch, que l'exception ait été levée ou non.
- Lancer une exception : Utilisation de throw pour émettre une exception personnalisée.

```
class Calculatrice {  
    public function division($a, $b) {  
        if ($b == 0) {  
            throw new Exception("Division par zéro interdite !");  
        }  
        return $a / $b;  
    }  
}
```

```
try {  
    $calculatrice = new Calculatrice();  
    echo $calculatrice->division(10, 0);  
} catch (Exception $e) {  
    echo "Erreur : " . $e->getMessage();  
} finally {  
    echo "\nFin du calcul.";  
}
```


ACCESSEURS, MUTATEURS ET MÉTHODES MAGIQUES EN PHP

Méthodes Magiques en PHP:

- PHP fournit des méthodes dites "magiques" pour effectuer des actions spéciales lors de l'accès aux propriétés ou de l'appel de méthodes. Ces méthodes commencent par un double underscore « __ ».
- Principales méthodes magiques :
 - `__construct` : Appelée lors de l'instanciation de la classe.
 - `__destruct` : Appelée à la fin du script ou lors de la suppression de l'objet.
 - `__get` et `__set` : Permettent l'accès et l'attribution des valeurs pour des propriétés dynamiques ou inaccessibles.
 - `__toString` : Définit le comportement de l'objet lorsqu'il est utilisé comme une chaîne.

```
class Animal {  
    private $nom;  
  
    public function __construct($nom) {  
        $this->nom = $nom;  
    }  
  
    public function __toString() {  
        return "Animal : " . $this->nom;  
    }  
}  
  
$animal = new Animal("Tigre");  
echo $animal; // Affiche : Animal : Tigre
```

ACCESSEURS, MUTATEURS ET MÉTHODES MAGIQUES EN PHP

Accesseurs (Getters) et Mutateurs (Setters) :

- Les Accesseurs (Getters) sont des méthodes pour lire la valeur d'une propriété privée ou protégée.
- Les Mutateurs (Setters) sont des méthodes pour définir ou modifier la valeur d'une propriété.
- Ils permettent de contrôler et valider l'accès et la modification des propriétés privées, renforçant ainsi l'encapsulation.

```
class Personne {  
    private $nom;  
  
    public function getNom() {  
        return $this->nom;  
    }  
  
    public function setNom($nom) {  
        if (strlen($nom) > 0) {  
            $this->nom = $nom;  
        }  
    }  
}  
  
$personne = new Personne();  
$personne->setNom("Alice");  
echo $personne->getNom(); // Affiche : Alice
```

ACCESSEURS, MUTATEURS ET MÉTHODES MAGIQUES EN PHP

Constructeur (__construct) :

- Le constructeur est une méthode spéciale appelée automatiquement lors de la création d'une instance de la classe.
- Utilisé pour initialiser les propriétés d'un objet avec des valeurs par défaut ou fournies en paramètres.

Destructeur (__destruct) :


- Le destructeur est une méthode spéciale appelée automatiquement quand un objet est détruit ou que le script termine.
- Utilisé pour libérer des ressources ou exécuter des actions de nettoyage.

```
class Fichier {  
    private $nomFichier;  
  
    public function __construct($nomFichier) {  
        $this->nomFichier = fopen($nomFichier, "w");  
    }  
  
    public function __destruct() {  
        fclose($this->nomFichier);  
    }  
}  
  
$fichier = new Fichier("exemple.txt");  
// À la fin du script, __destruct sera appelé automatiquement pour fermer le fichier
```

ACCESSEURS, MUTATEURS ET MÉTHODES MAGIQUES EN PHP

Méthodes Statiques :

- Une méthode statique appartient à la classe elle-même et non à une instance spécifique.
- Appelée directement via la classe en utilisant `NomDeClasse::methode()`.
- Utile pour des fonctions utilitaires ou des constantes partagées.



```
class Utilitaires {  
    public static function addition($a, $b) {  
        return $a + $b;  
    }  
}  
  
echo Utilitaires::addition(3, 5); // Affiche : 8
```

ACCESSEURS, MUTATEURS ET MÉTHODES MAGIQUES EN PHP

Propriétés Statiques:

- Une propriété statique appartient à la classe elle-même et non aux instances de celle-ci. Elle est partagée entre toutes les instances et accessible avec `self::` ou `NomDeClasse::`

Constantes de Classe:

- Une constante de classe est une valeur qui ne change jamais pendant l'exécution du script et qui est partagée entre toutes les instances de la classe.
- Déclarée avec le mot-clé `const`, elle est accessible sans instancier la classe via `NomDeClasse::NOM_CONSTANTE`.



```
class Configuration {  
    const VERSION = "1.0.0";  
}  
  
echo Configuration::VERSION; // Affiche : 1.0.0
```



```
class Compteur {  
    public static $total = 0;  
  
    public function __construct() {  
        self::$total++;  
    }  
}  
  
$obj1 = new Compteur();  
$obj2 = new Compteur();  
echo Compteur::$total; // Affiche : 2
```

INTRODUCTION AUX DESIGN PATTERNS (SINGLETON, FACTORY, MVC)

Qu'est-ce qu'un Design Pattern ?

- Un design pattern (ou patron de conception) est une solution réutilisable à un problème de conception courant.
- Les design patterns aident à structurer le code pour le rendre plus maintenable, modulable, et évolutif.

Pattern Singleton :

- Assure qu'une classe n'a qu'une seule instance et fournit un point d'accès global à cette instance.
- Utilisé pour les classes où une unique instance est suffisante, comme les connexions à la base de données.

INTRODUCTION AUX DESIGN PATTERNS (SINGLETON, FACTORY, MVC)

Pattern Singleton :



```
class Singleton {  
    private static $instance = null;  
  
    private function __construct() {} // Constructeur privé  
  
    public static function getInstance() {  
        if (self::$instance === null) {  
            self::$instance = new Singleton();  
        }  
        return self::$instance;  
    }  
}  
  
$singleton = Singleton::getInstance();
```

INTRODUCTION AUX DESIGN PATTERNS (SINGLETON, FACTORY, MVC)

Pattern Factory :

- Utilisé pour créer des objets sans spécifier la classe exacte de l'objet à créer.
- Fournit une méthode pour créer des objets en fonction d'un type ou d'un paramètre.



```
class AnimalFactory {  
    public static function create($type) {  
        if ($type === 'chien') {  
            return new Chien();  
        } elseif ($type === 'chat') {  
            return new Chat();  
        }  
        return null;  
    }  
}
```

```
$animal = AnimalFactory::create('chien');
```


INTRODUCTION AUX DESIGN PATTERNS (SINGLETON, FACTORY, MVC)

Pattern Factory :

```
class AnimalFactory {
    public static function createAnimal($type) {
        if ($type === 'chien') {
            return new Chien();
        } elseif ($type === 'chat') {
            return new Chat();
        } else {
            throw new Exception("Type d'animal
inconnu.");
        }
    }
}

class Chien {
    public function parler() {
        return "Woof!";
    }
}

class Chat {
    public function parler() {
        return "Meow!";
    }
}

// Utilisation
$animal = AnimalFactory::createAnimal('chien');
echo $animal->parler(); // Affiche : Woof!
```

INTRODUCTION AUX DESIGN PATTERNS (SINGLETON, FACTORY, MVC)

Pattern MVC (Model-View-Controller) :

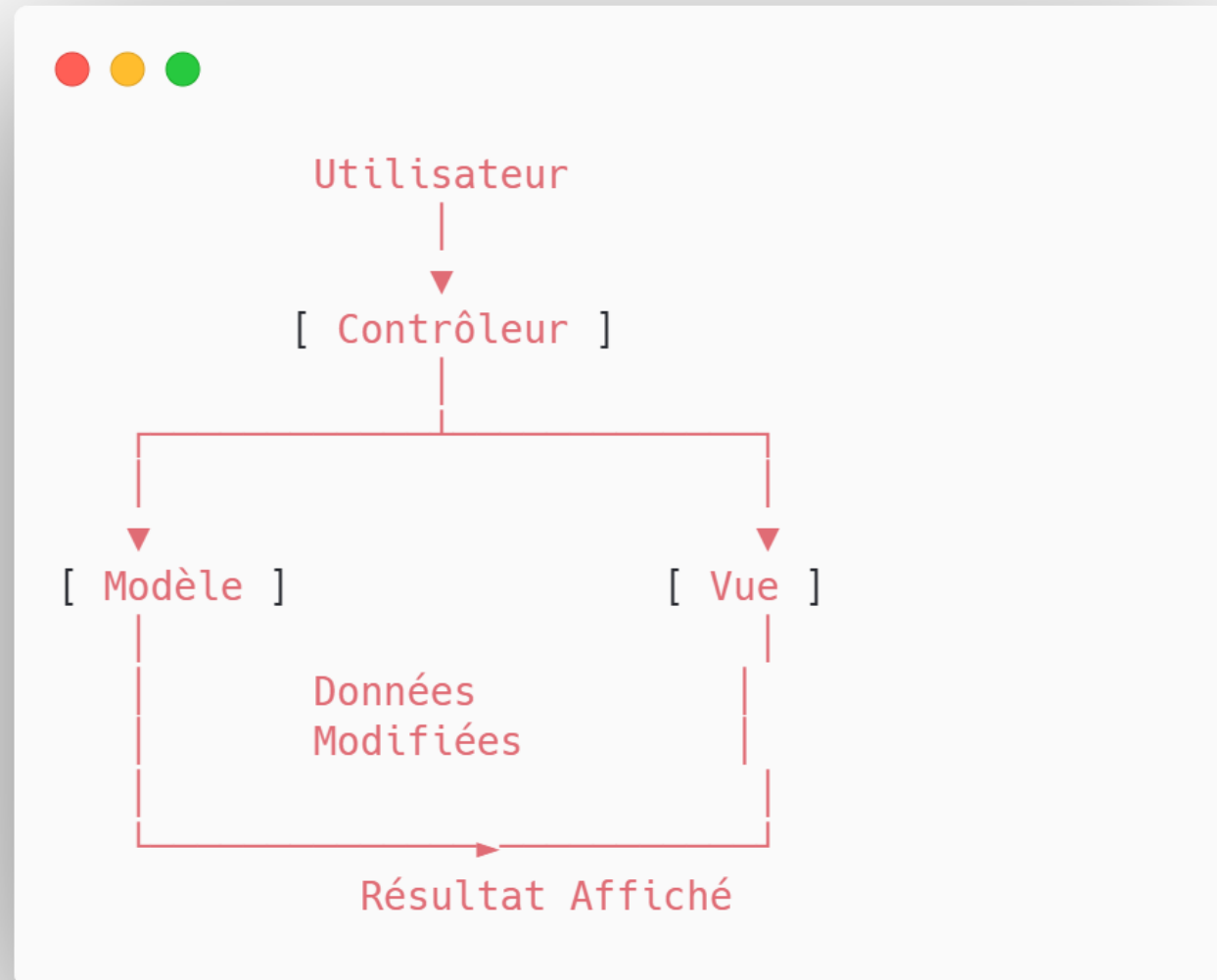
- Structure une application en séparant la logique métier (Modèle), la présentation (Vue), et la logique de contrôle (Contrôleur).
- Permet de structurer les applications en couches distinctes, facilitant la maintenance et les mises à jour.

Schéma MVC :

- Modèle : Gère les données et la logique métier.
- Vue : Affiche les données et l'interface utilisateur.
- Contrôleur : Reçoit les requêtes et coordonne les actions entre le modèle et la vue.

INTRODUCTION AUX DESIGN PATTERNS (Singleton, Factory, MVC)

Pattern MVC :



EXERCICE PRATIQUE – MINI-PROJET EN POO AVEC DESIGN PATTERN

1. Créer une mini-application en PHP orienté objet, intégrant un design pattern pour gérer un ensemble d'objets de manière flexible.

Description de l'application :

Vous allez développer une application simple pour gérer un catalogue de produits. L'utilisateur pourra choisir le type de produit à créer (par exemple, livre, électronique) via une Factory. Chaque produit aura un nom, un prix et des caractéristiques spécifiques selon son type.

Instructions de l'exercice :

1. Création de la classe abstraite Produit :
2. Définissez une classe abstraite Produit avec les propriétés communes suivantes :
 1. nom (string)
 2. prix (float)
3. Ajoutez un constructeur pour initialiser ces propriétés.
4. Déclarez une méthode abstraite getDescription().
5. Création des classes concrètes :
 1. Classe Livre :
 1. Hérite de Produit.
 2. Ajoutez une propriété auteur (string).

6. Implémentez la méthode getDescription() pour retourner une description du livre.
7. Classe Electronique :
 1. Hérite de Produit.
 2. Ajoutez une propriété marque (string).
8. Implémentez la méthode getDescription() pour retourner une description de l'électronique.
9. Création de la classe ProduitFactory :
 6. Créez une classe ProduitFactory avec une méthode statique createProduit(\$type, \$nom, \$prix, \$extra).
 7. Utilisez un switch pour instancier le bon type de produit selon le choix de l'utilisateur.
10. Compléter la classe Database

```
/htdocs/  
└─ catalogue_produits/  
    └─ index.php  
        └─ classes/  
            └─ Produit.php  
            └─ Livre.php  
            └─ Electronique.php  
            └─ ProduitFactory.php  
            └─ Database.php
```

QCM

PROJET DE SOUTENANCE : APPLICATION DE GESTION D'ÉCHANGES DE SERVICES ENTRE ÉTUDIANTS

Dans le cadre de votre cours, vous serez amenés à réaliser un projet de soutenance.

Vous serez répartis en équipes de deux personnes, à l'exception d'une équipe qui pourra être composée de trois membres.

Chaque équipe disposera de 20 minutes pour présenter son projet.

Présentation :

- La soutenance comprendra :
- Un diaporama (PPT) expliquant votre projet.
- Un parcours utilisateur démontrant l'application.
- La livraison du code source.

Objectif du Projet :

- Développer une application web permettant aux étudiants de proposer ou de demander des services entre eux (aide pour les cours, prêt de matériel, etc.) en utilisant un système d'échange basé sur des points.

PROJET DE SOUTENANCE : APPLICATION DE GESTION D'ÉCHANGES DE SERVICES ENTRE ÉTUDIANTS

Technologies à Utiliser :

- Backend : Développement de trois microservices en PHP :
 - Authentification
 - Annonces
 - Gestion des points
- Base de Données (BDD) : Utilisation d'une base de données pour stocker les informations des utilisateurs, des services proposés et des transactions.
- Frontend : Choix libre d'une technologie pour créer une interface moderne et dynamique.
- API REST : Pour la communication entre le frontend et les microservices.
- Gestion de Session : Utilisation de sessions pour la gestion de la connexion des utilisateurs.
- Tokens : Utilisation de tokens pour chaque appel de service afin d'assurer la sécurité et l'authentification des requêtes.
- Cookies : Utilisation de cookies pour sauvegarder des informations.
- Patrons de Conception :
 - Inclure le patron Singleton pour la gestion de la connexion à la base de données.
 - Inclure le patron Factory pour la création d'objets.
- API Gateway : Mise en place d'une API Gateway pour centraliser les appels aux microservices.
- Remarque :
 - Pour le frontend, utilisez la technologie qui vous intéresse pour créer l'interface avec l'application back-office.

FIN

