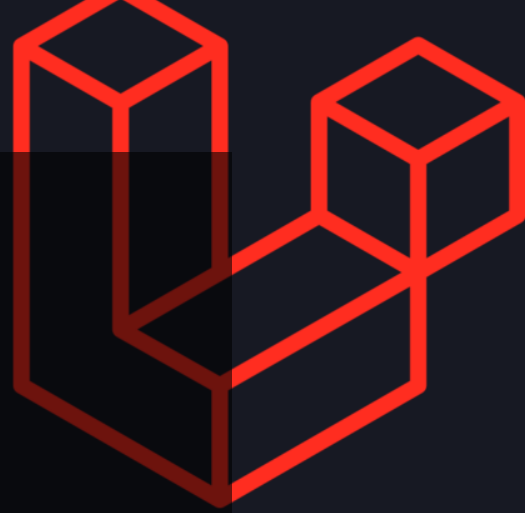


Développer en back-end



Découvrir le Framework PHP Laravel (Suite)

Formatrice : Elidrissi Asmae



Plan du cour

Architecture d'un projet Laravel 01

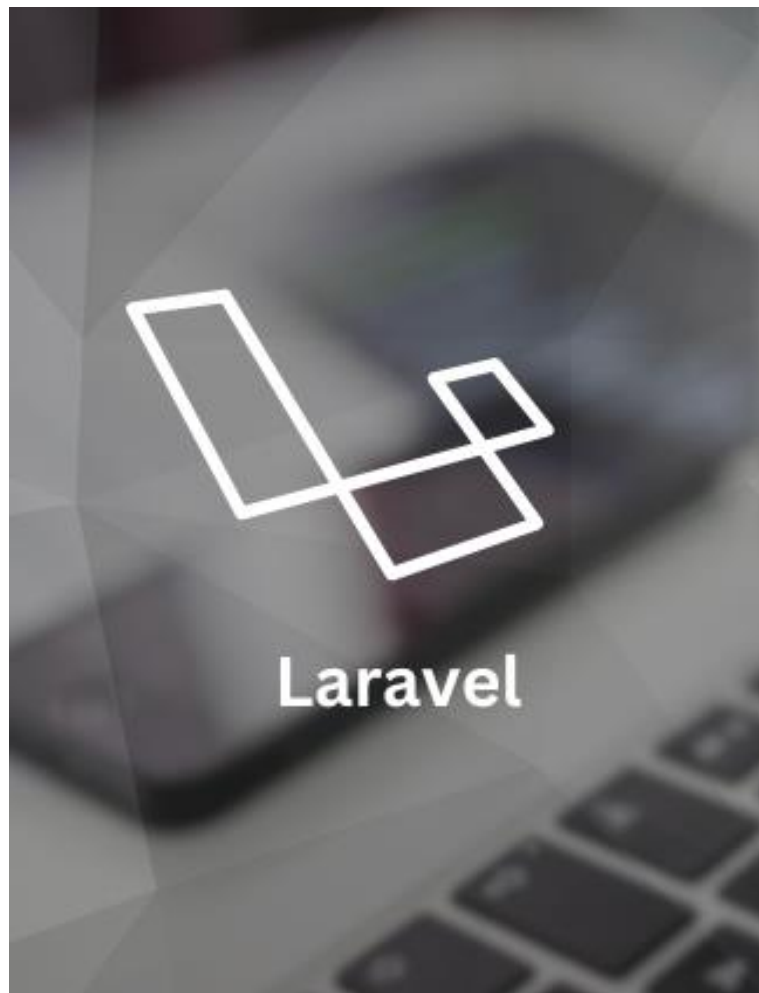
Laravel Artisan 02

Les commandes Laravel Artisan 03

Rappel: Programmation php 04

TP 05





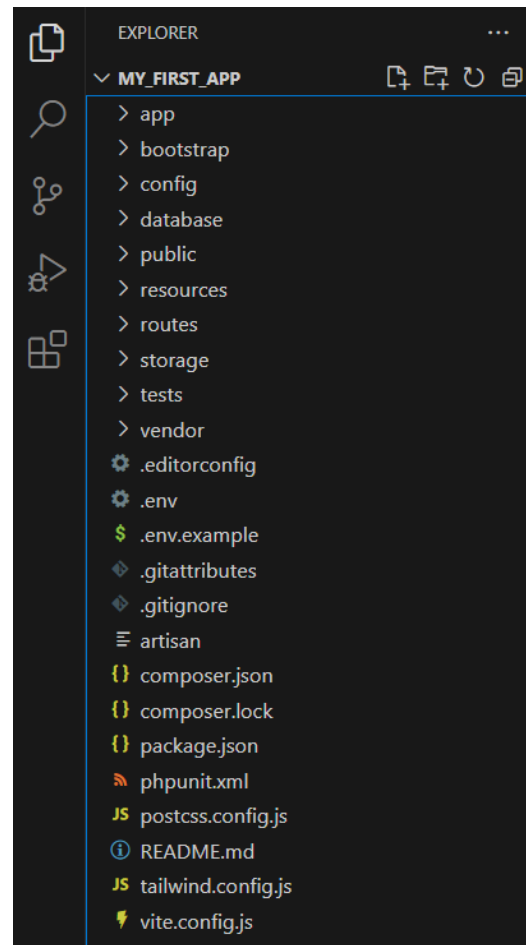
Architecture d'un projet Laravel

Présentation des Frameworks PHP

/app : Le cœur de l'application. Contient le code métier, organisé en sous-dossiers :

- Models : Contient les modèles.
- Http : Contient les contrôleurs et middlewares.
- Providers : Contient les classes pour configurer les services.

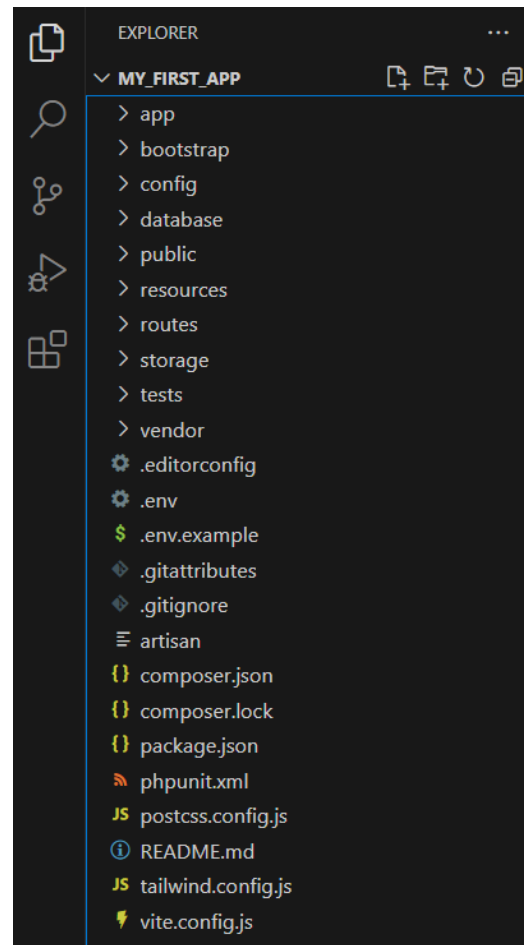
- **/config** : contient tous les fichiers de configuration de votre application, authentification, namespace, mails, base de données etc...
- **/database** : Gère tout ce qui est lié à la base de données les migrations de base de données, les données factices (faux data) de vos modèles.



Présentation des Frameworks PHP

Les répertoires racines d'un projet Laravel

- **/public:** Contient les fichiers accessibles publiquement, comme index.php (point d'entrée), ainsi que les fichiers CSS, JavaScript, et images. (le seul qui doit être accessible par le serveur)
- **/resources:** Ce dossier contient les fichiers liés à l'interface utilisateur.
 - views/ : Contient les fichiers Blade (templates) pour générer les pages HTML dynamiques.
 - js/ et css/ : Gère les fichiers front-end (JavaScript et CSS).
- **/routes** : Les fichiers définissant les routes de l'application.
 - web.php : Pour les routes web (interfaces utilisateur).
 - console.php : Pour les commandes Artisan personnalisées.



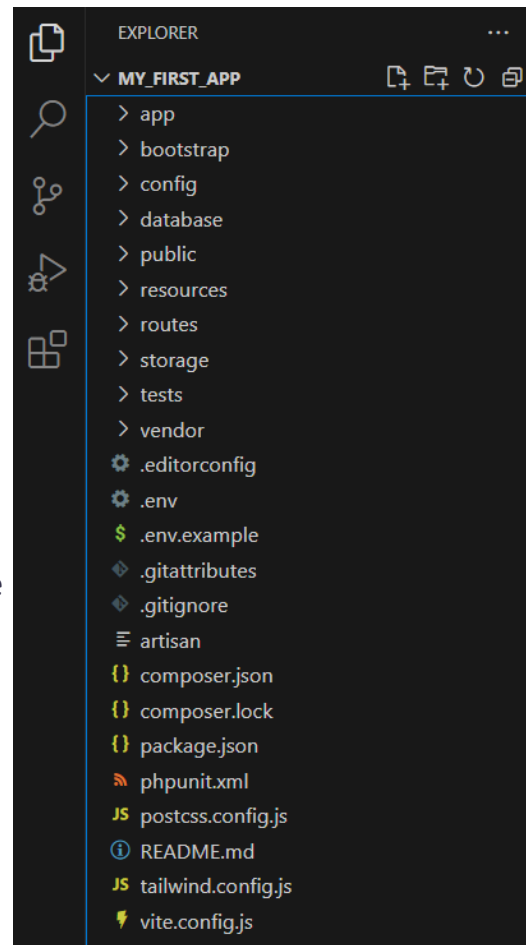
Présentation des Frameworks PHP

Les répertoires racines d'un projet Laravel

- **/storage:** contient vos logs, modèles de vues (blades) compilés, sessions basées sur des fichiers, fichiers caches et autres fichiers générés par le framework.

Ce répertoire est divisé en répertoires **app**, **framework** et **logs**.

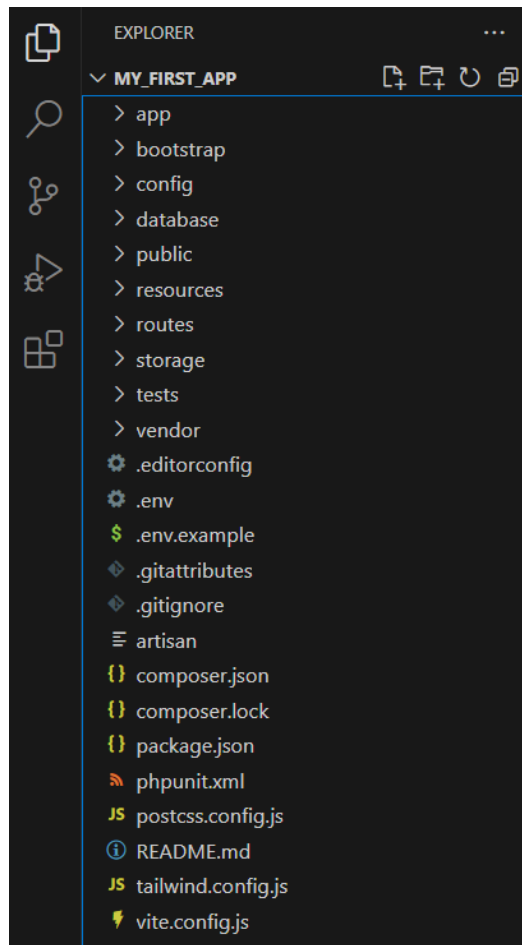
- **storage/app** peut être utilisé pour stocker tous les fichiers générés par votre application.
- **storage/framework** est utilisé pour stocker les fichiers et les caches générés par le framework.
- **storage/logs** contient les fichiers journaux de votre application.
- **storage/app/public:** peut être utilisé pour stocker des fichiers générés par l'utilisateur, tels que des avatars de profil, les images des vos publications, qui doivent être accessibles au public.



Présentation des Frameworks PHP

Les répertoires racines d'un projet Laravel

- **/bootstrap** : démarrage de l'app.
- **/tests**: contient vos tests automatisés. Des exemples de tests unitaires PHPUnit et de tests de fonctionnalités sont fournis prêts à l'emploi.
 - Chaque classe de test doit être suffixée par le mot Test.
 - Vous pouvez exécuter vos tests à l'aide des commandes **php unit** ou **php vendor/bin/phpunit**.
 - Ou, si vous souhaitez une représentation plus détaillée et plus belle de vos résultats de test, vous pouvez exécuter vos tests à l'aide de la commande Artisan **php artisan test**.



Présentation des Frameworks PHP

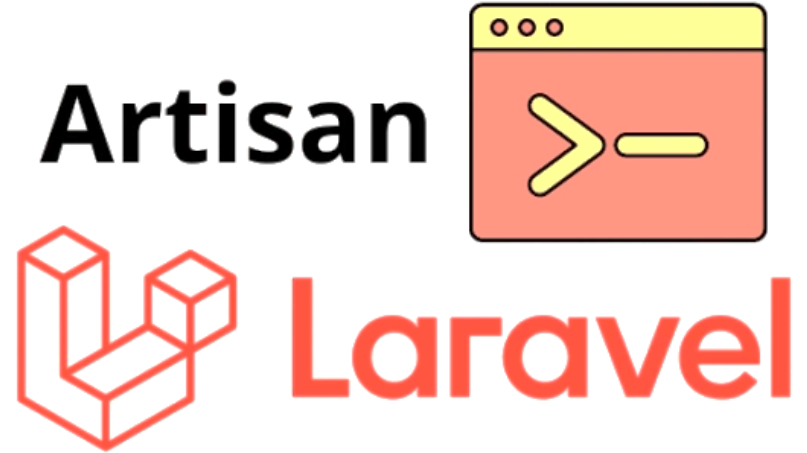
Le fichier **composer.json**

Dans un projet Laravel, le fichier **composer.json** est un fichier clé utilisé par Composer, le gestionnaire de dépendances de PHP. Ce fichier contient des métadonnées sur le projet, telles que les dépendances nécessaires, les scripts personnalisés et d'autres configurations utiles pour le développement.

- Déclarer les dépendances du projet (bibliothèques et packages PHP nécessaires).
- Spécifier les versions des packages et les contraintes de compatibilité.
- Définir des informations sur le projet (nom, description, auteur, etc.).
- Ajouter des scripts personnalisés à exécuter via Composer (par exemple, `composer install`).
- Configurer le chargement automatique de classes (autoloader).

```
{  
  "name": "laravel/laravel",  
  "type": "project",  
  "description": "The Laravel Framework.",  
  "keywords": [ "framework", "laravel" ],  
  "license": "MIT",  
  "require": {  
    "php": "^7.1.3",  
    "fideloper/proxy": "^4.0",  
    "laravel/framework": "5.7.*",  
    "laravel/tinker": "^1.0"  
  },  
  [...]
```

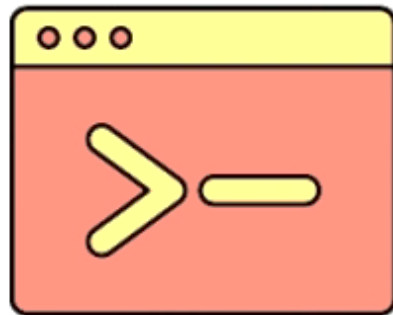

Laravel Artisan



Laravel Artisan

Qu'est ce que Laravel Artisan?

Laravel Artisan est une Interface en Ligne de Commande (CLI) qui va vous permettre de gérer votre application en lançant des commandes via le terminal (effacer le cache de l'application, gérer des modèles, des contrôleurs, des routes...)



Laravel Artisan

❖ Les commandes utiles de Laravel Artisan:

1. Afficher la liste des commandes artisan

```
php artisan list
```

2. Pour afficher un écran d'aide, faites précéder le nom de la commande de help

```
php artisan help migrate
```

3. Pour créer un modèle

```
php artisan make:model NomDuModel
```

4. Vérifier la version de Laravel installée

```
php artisan -version
```

5. lancer le serveur , lancer le projet Laravel

```
php artisan serve
```

6. voir toutes les routes de votre application

```
artisan route:list
```

7. Exécuter toutes les migrations:

```
php artisan migrate
```

8. faire un retour en arrière pour la dernière migration

```
php artisan migrate : rollback
```



Rappel:
Programmation php

Rappel: Programmation php

❖ Instructions conditionnelles

– L'instruction conditionnelle **if**

Structure	Exemple
<pre>if (cond) { ... } elseif (cond) { ... } else { ... }</pre>	<pre><?php \$note = 15; if (\$note<10) echo "redoublant"; elseif (\$note>10 and \$note<15) echo "moyen"; else echo "excellent";</pre>
Affichage :	
excellent	

– L'instruction conditionnelle **switch**

Structure	Exemple
<pre>switch (expr) { case VALEUR1 : ... break ; default : ... break ; }</pre>	<pre><?php \$couleur = 'Vert'; switch (\$couleur) { case 'Vert': echo "Démarrer"; break; case 'Rouge': echo "Stop"; break; case 'Orange': echo "Attention"; break; }</pre>
Affichage :	
Démarrer	

Rappel: Programmation php

❖ Instructions itératives

– La boucle **for**

Structure	Exemple
<pre>for (init ; cond ; modif) { ... }</pre>	<pre><?php for (\$i=1; \$i<5; \$i++) { echo \$i.' ' }</pre>
Affichage :	
1 2 3 4	

– La boucle **while**

Structure	Exemple
<pre>while (cond) { ... }</pre>	<pre><?php \$i=1; while (\$i<5) { echo \$i.' ' \$i++; }</pre>
Affichage :	
1 2 3 4	

Rappel: Programmation php

❖ Les fonctions

```
<?php
function ma_fonc ( $param1 , $param2 , ... ) {
    ...
    return ...;
}
```

```
<?php
function bonjour() {
    echo " Bonjour ";
}
bonjour();
```

Affichage :

Bonjour BENANI HIND

```
<?php
function ma_fonc( ) {
    global $var;
    $var = 54;
}
$var = 0;
echo "Avant ma_fonc $var".PHP_EOL;
ma_fonc( ) ;
echo "Après ma_fonc $var";
```

Affichage :

Avant ma_fonc 0

Après ma_fonc 54

Rappel: Programmation php

❖ Programmation orienté objet :

Fichier classes/utilisateur.class.php :

```
<?php
class Utilisateur {
    public $login;
    public $password;
    public function getLogin(){
        return $this->login;
    }
    public function setLogin($new_login){
        $this->login = $new_login;
    }
    public function setPasseword($new_password){
        $this->password = $new_password;
    }
}
?>
```

Fichier index.php

```
<!DOCTYPE html>
<html lang="en">
<head><meta charset="UTF-8"></head>
<body>
    <?php
        require_once 'classes/Utilisateur.class.php';
        $user1 = new Utilisateur();
        $user1->login = "ALLAOUI";
        $user1->password = "all123";

        $user2 = new Utilisateur();
        $user2->setLogin("BENANI");
        $user2->setPasseword("ben123");

        echo "Utilisateur 1 : ".$user1->getLogin()."<br/>";
        echo "Utilisateur 2 : ".$user2->getLogin()."<br/>";
    ?>
</body>
</html>
```

Affichage :

Utilisateur 1 : ALLAOUI
Utilisateur 2 : BENANI

Rappel: Programmation php

❖ Programmation orienté objet :

Fichier classes/utilisateur.class.php :

```
<?php
class Utilisateur {
    private $login;
    private $password;

    public function __construct($new_login,
$new_password){
        $this->login = $new_login;
        $this->password = $new_password;
    }

    public function getLogin(){
        return $this->login;
    }
}
?>
```

Fichier index.php

```
<!DOCTYPE html>
<html lang="en">
<head><meta charset="UTF-8"></head>
<body>
    <?php
        require_once 'classes/Utilisateur.class.php';
        $user1 = new Utilisateur("ALLAOUI", "all123");
        $user2 = new Utilisateur("BENANI", "ben123");

        echo "Utilisateur 1 : ".$user1->getLogin()."<br/>";
        echo "Utilisateur 2 : ".$user2->getLogin()."<br/>";
    ?>
</body>
</html>
```

Affichage :

Utilisateur 1 : ALLAOUI
Utilisateur 2 : BENANI

Rappel: Programmation php

❖ Programmation orienté objet :

- Le constructeur **__construct()** et le destructeur **__destruct()** sont tous les deux des méthodes magiques qui s'exécutent automatiquement suite à un événement.
- **__get(\$att)** connue par le nom de getter est une méthode qui s'exécute automatiquement quand on appelle un attribut inexistant ou inaccessible.

```
<?php
class Utilisateur {
    private $login;
    public function __get($att){
        return 'L\'attribut login est inaccessible.';
    }
}
$objet=new Utilisateur();
echo $objet->login;
```

Affichage :

L'attribut login est inaccessible.

- **__set(\$att,\$val)** connue sous le nom de setter est appelée automatiquement quand on essaie de modifier un attribut inexistant ou inaccessible.

```
<?php
class Utilisateur {
    private $login;
    public function __get($att){
        return $this->$att;
    }
    public function __set($att,$val){
        $this->$att=$val;
    }
}
$objet=new Utilisateur();
$objet->login="BENANI";
echo $objet->login;
?>
```

Affichage :

BENANI

Rappel: Programmation php

❖ Programmation orienté objet :

- **__isset(\$att)** est appelé automatiquement quand on exécute la fonction `isset()` en lui passant en paramètre un attribut inexistant ou inaccessible.
- **__unset(\$att)** est appelé automatiquement quand on essaie de supprimer un attribut inexistant ou inaccessible à l'aide de la fonction `unset()`.
- **__toString()** est appelé automatiquement quand on tente de traiter un objet en tant que chaîne de caractères (à l'aide de la structure `echo` par exemple).

```
<?php
class Utilisateur {
    private $login;
    public function __construct($param){
        $this->login = $param;
    }
    public function __toString(){
        return "Bonjour ".$this->login;
    }
}
$objet=new Utilisateur("BENANI");
echo $objet;

?>
```

Affichage :

Bonjour BENANI

Rappel: Programmation php

❖ Gestion des exceptions

```
<?php
$a=10;
$b=0;
try{
    $c=$a/$b;
    echo $c;
} catch(Exception $e) {
    echo $e->getMessage();
}
?>
```

Affichage :

PHP Warning: Division by zero

```
<?php
$a=10;
$b=0;
try{
    if($b==0)
        throw new Exception("Le dénominateur ne doit pas
        être null.");
    $c=$a/$b;
    echo $c;
} catch(Exception $e) {
    echo $e->getMessage();
}
?>
```

Affichage :

Le dénominateur ne doit pas être null.

- getMessage(): retourne le message d'erreur passé au constructeur lors du lancement de l'exception.
- getCode(): retourne le code d'erreur passé aussi au constructeur.
- getLine(): retourne le numéro de la ligne où l'exception a été lancée.
- getFile(): retourne le nom du document où les choses se passent.

Rappel: Programmation php

❖ Développer une application avec PHP et MySQL : API PDO, MySQLi

Pour pouvoir manipuler nos bases de données MySQL en PHP (sans passer par phpMyAdmin), nous allons déjà devoir nous connecter à MySQL.

Pour cela, le PHP met à notre disposition deux API (Application Programming Interface) :

- L'extension MySQLi ;
 - L'extension PDO (PHP Data Objects).
-
- Chacune de ces deux API possède des forces différentes et comme vous vous en doutez elles ne sont pas forcément interchangeables.
 - Il existe notamment une différence notable entre ces deux API : l'extension MySQLi ne va fonctionner qu'avec les bases de données MySQL tandis que PDO va fonctionner avec 12 systèmes de bases de données différents.

Rappel: Programmation php

- ❖ Développer une application avec PHP et MySQL : API PDO, MySQLi

Connexion au serveur avec MySQLi

```
<?php
    $servername = 'localhost';
    $username = 'root';
    $password = '';
    //On établit la connexion
    $conn = new mysqli($servername, $username, $password);
    //On vérifie la connexion
    if(!$conn){
        die('Erreur : ' .mysqli_connect_error());
    }
    echo 'Connexion réussie';
    $conn->close();
?>
```

Rappel: Programmation php

- ❖ Développer une application avec PHP et MySQL : API PDO, MySQLi

Connexion au serveur avec PDO

```
<?php
$servername = 'localhost';
$username = 'root';
$password = '';
try{
    $conn = new PDO("mysql:host=$servername;dbname=bddtest",
$username, $password);
    echo 'Connexion réussie';
} catch(PDOException $e) {
    echo "Erreur : " . $e->getMessage();
}
$conn = null;
?>
```

Rappel: Programmation php

- ❖ Développer une application avec PHP et MySQL : API PDO, MySQLi

Exécuter une requête

- **PDO::prepare** : Prépare une requête à l'exécution et retourne un objet
- **PDO::exec** : Exécute une requête SQL et retourne le nombre de lignes affectées

```
$stmt = $conn->prepare("SELECT * FROM utilisateurs WHERE id=?");  
$stmt->bindParam(1,$id);  
$stmt->execute();
```

- Ensuite, on traite les données avec un foreach (exemple avec affichage des colonnes nom :

```
$res = $stmt->fetchAll();  
foreach ( $res as $row ) {  
    echo $row['nom'];  
}
```

- **PDO::query** : Prépare et Exécute une requête SQL sans marque substitutive

```
$sql = 'SELECT nom, prenom FROM utilisateurs ORDER BY name';  
foreach ($conn->query($sql) as $row) {  
    print $row['nom'] . "\t";  
    print $row['prenom'] . "\n";  
}
```



```
if (count($workerProc) == 1) {  
    $test_files = array_merge($test_files, $sequentialTests);  
    $sequentialTests = [];  
}  
$files = [];  
$maxBatchSize = $valgrind ? 1 : ($batchSize < 100 ? $batchSize : 100);  
$averageFilesPerWorker = max(1, ($batchSize / $valgrind));  
$batchSize = min($maxBatchSize, $averageFilesPerWorker);  
while (count($files) <= $batchSize) {  
    foreach ($fileConflictsWith as $file) {  
        if (isset($activeConflicts[$fileConflictsWith[$file]]) &&  
            $waitingTests[$fileConflictsWith[$file]] == $file) {  
            continue 2;  
        }  
        $files[] = $file;  
    }  
    if ($files) {  
        foreach ($files as $file) {  
            foreach ($fileConflictsWith as $fileConflictsWithKey) {  
                if ($fileConflictsWithKey == $file) {  
                    continue;  
                }  
                if (isset($activeConflicts[$fileConflictsWithKey]) &&  
                    $waitingTests[$fileConflictsWithKey] == $fileConflictsWithKey) {  
                    continue;  
                }  
                $files[] = $fileConflictsWithKey;  
            }  
        }  
    }  
}
```

TP

TP

○ Créer le Premier projet

➤ Objectif : Créer un site web avec l'architecture MVC en php natif

On souhaite créer une application web de gestion des voitures en suivant l'architecture MVC,

Pour ce, on utilisera la base de données VoituresDB qui contient la table voiture suivante :

Voiture (immatriculation, marque, kilometrage) :

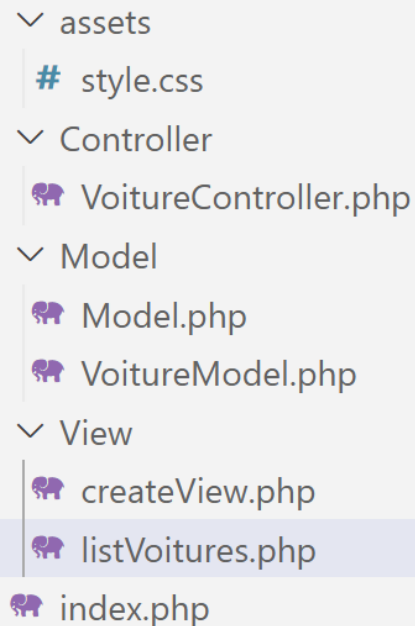
Script de création de la base de données :

```
create database voituresDB;  
use voituresDB;  
create table voiture(  
    immatriculation varchar(20) primary key,  
    marque varchar(50),  
    kilometrage int);
```

```
insert into voiture values  
    ('123-A-45', 'Dacia', 58000),  
    ('5689-B-89', 'Renault', 100000),  
    ('10088-A-547', 'Dacia', 20000),  
    ('2365-A-101', 'Ford', 34000);
```

TP

- Les fichiers de l'application seront organisés de la manière suivante :



A file explorer view showing the application structure. The folders 'assets', 'Controller', 'Model', and 'View' are expanded, indicated by downward arrows. The file 'listVoitures.php' is highlighted with a blue background.

- assets
 - style.css
- Controller
 - VoitureController.php
- Model
 - Model.php
 - VoitureModel.php
- View
 - createView.php
 - listVoitures.php
- index.php

QCM



A decorative header bar with a dark blue background. It features several 3D cubes outlined in a reddish-orange color. Some cubes are faint and in the background, while others are more prominent in the foreground.

À la prochaine !