Sprint 3 - PHP Mastery: From Fundamentals to Frameworks

Stage PFE – Entreprise **VOID**

Stagiaire : Hibat Allah Rguiti

22 février 2026

Table des matières

Chapitre 1

PHP Syntax, Setup & Tooling

1.1. Introduction to PHP

PHP (*Hypertext Preprocessor*) est un langage de programmation côté serveur, principalement utilisé pour construire des applications web dynamiques. Contrairement au JavaScript exécuté dans le navigateur, PHP s'exécute sur le serveur et génère du contenu HTML envoyé ensuite au client.

PHP est couramment utilisé pour :

— gérer des formulaires et des requêtes HTTP,
— interagir avec des bases de données,
— construire des APIs REST,
— développer des applications web complètes.

L'écosystème moderne s'est structuré autour de bonnes pratiques :

— **PHP 7** a apporté un gain de performance considérable,
— **Composer** a transformé la gestion des dépendances,
— les frameworks comme Laravel et Symfony ont modernisé l'image du langage,
— **PHP 8+** introduit des fonctionnalités comparables aux langages modernes.

1.2. Hands-on Coding and Tooling Setup

Après avoir étudié les bases et les concepts modernes de PHP, la deuxième étape consiste à appliquer ces connaissances à travers un exercice pratique inspiré d'un projet réel open-source : *The PHP League Config*.

L'objectif principal n'est pas de recopier simplement du code, mais de comprendre comment un projet PHP moderne est structuré et de se familiariser avec les outils professionnels utilisés dans l'écosystème.

1.2.1. Configure Tooling

Dans le développement PHP moderne, l'écriture du code ne suffit pas : il est essentiel d'utiliser des outils automatiques de qualité.

Installing PHPCS (Code Style Checker)

PHPCS (*PHP CodeSniffer*) est un outil qui permet de vérifier que le code respecte un style standardisé.

Il est utilisé pour :

— appliquer des conventions de formatage,

— garantir un code lisible et homogène,

— suivre les recommandations PSR (PHP Standards Recommendations).

Cela permet de travailler efficacement en équipe sur de grands projets.

Installing PHPStan (Static Analysis)

PHPStan est un outil d'analyse statique du code.

Contrairement à l'exécution classique, l'analyse statique détecte les erreurs sans lancer le programme :

— types incorrects,

— méthodes inexistantes,

— variables non définies,

— incohérences dans les classes.

PHPStan améliore fortement la fiabilité du code, surtout dans les projets complexes.

{} composer.json > { } autoload

```json
{
    "name": "hibatallah/dummy-config",
    "description": "Dummy package for testing purposes only",
    "type": "library",
    "license": "MIT",
    "autoload": {
        "psr-4": {
            "Hibatallah\\Config\\": "src/"
        }
    },
    "require": {},
    "require-dev": {
        "squizlabs/php_codesniffer": "^4.0",
        "phpstan/phpstan": "^2.1"
    },
    "scripts": {
        "lint": "phpcs",
        "fix": "phpcbf",
        "stan": "phpstan analyse"
    }
}
```

1.2.2. Step 4 : Publish a Dummy Package

La dernière étape consiste à publier le projet sous forme de package PHP.

Un **package** est une bibliothèque réutilisable distribuée via Composer.

Publishing to Packagist

Packagist est le dépôt officiel des packages PHP utilisés par Composer.

Pour valider que toutes les étapes ont été correctement réalisées, une capture d'écran est ajoutée montrant :

— l'exécution correcte de PHPCS,

— l'analyse réussie avec PHPStan,

— la publication effective sur Packagist.

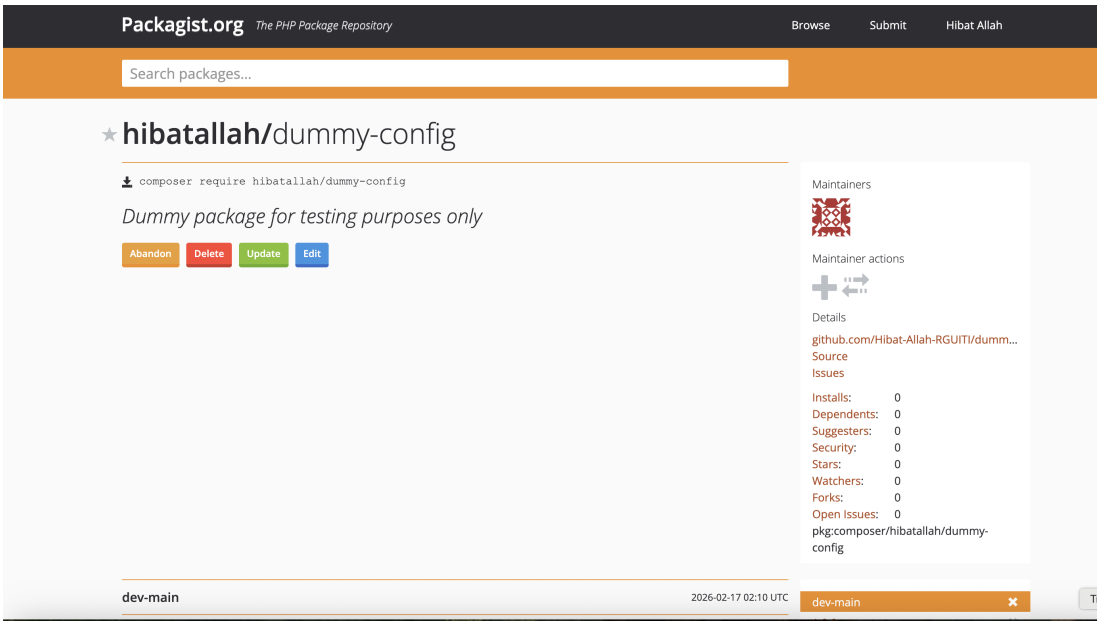


FIGURE 1.1 – Successful validation and publication of the dummy PHP package

Chapitre 2

Create Your Own Framework

2.1. Build a Simple PHP MVC Framework

Le pattern MVC (Modèle-Vue-Contrôleur) permet de séparer la logique métier de l'affichage.

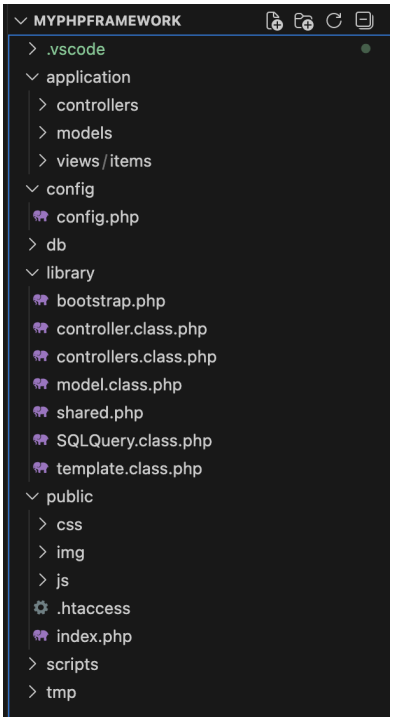


FIGURE 2.1 – Structure du framework

2.2. Fichier .htaccess et URL Rewriting

Le fichier `.htaccess` redirige toutes les requêtes vers `index.php`.

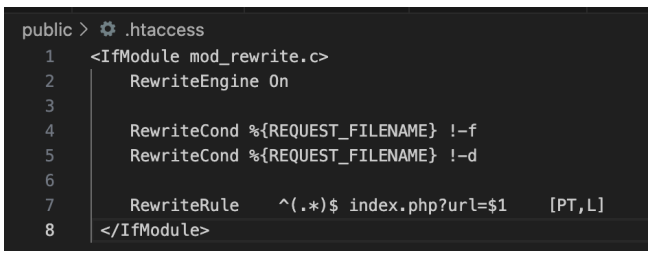


FIGURE 2.2 – Structure du framework

Explications :

— `RewriteEngine On` : active le moteur.

— `REQUEST_FILENAME` : vérifie si le fichier existe physiquement.

— `index.php?url=$1` : redirige le chemin vers le paramètre `url`.

2.3. Fichiers index.php et bootstrap.php

public > index.php

```php
<?php

define('DS', DIRECTORY_SEPARATOR);
define('ROOT', dirname(dirname(__FILE__)));

$url=$_GET['url'] ?? '';
if ($url === '') {
    $requestPath = $_SERVER['REQUEST_URI'] ?? '';
    $parsedPath = parse_url($requestPath, PHP_URL_PATH);
    $url = is_string($parsedPath) ? trim($parsedPath, '/') : '';
    // If someone hits /index.php/... keep only the routed part.
    if (str_starts_with($url, 'index.php/')) {
        $url = substr($url, strlen('index.php/'));
    } elseif ($url === 'index.php') {
        $url = '';
    }
}

require_once(ROOT.DS.'library'.DS.'bootstrap.php');
```

library > bootstrap.php

```php
<?php

require_once(ROOT.DS.'config'.DS.'config.php');
require_once(ROOT.DS.'library'.DS.'shared.php');

```

— **index.php** : point d'entrée unique.

— **bootstrap.php** : charge la configuration et `shared.php`.

2.4. Gestion des dépendances : composer.json et composer.lock

— `composer.json` : définit les besoins.

— `composer.lock` : fige les versions exactes pour la production.

2.5. Accès à la base de données avec PDO

L ;utilisation de (`PDO`) pour interagir avec les bases de données. Elle offre plusieurs avantages :

— Abstraction des différents types de bases (MySQL, PostgreSQL, SQLite, etc.),

— Sécurité renforcée grâce aux requêtes préparées, évitant les injections SQL,

— Gestion uniforme des erreurs via des exceptions.

Remarques :

— L'utilisation de `prepare()` et `execute()` permet d'éviter toute injection SQL,

— PDO facilite la portabilité et la maintenabilité du code base de données.

2.6. Configuration des environnements

config > config.php

```php
<?php

define ('DEVELOPMENT_ENVIRONMENT',true);

define('DB_NAME', 'testdb');
define('DB_USER', 'root');
define('DB_PASSWORD', '');
define('DB_HOST', 'localhost');

```

2.7. Résultat

Chapitre 3

Working with Databases & CLI

3.1. Importation d'un fichier CSV avec Symfony 8 et MySQL

3.1.1. Objectif

L'objectif de cet exercice est de développer une commande CLI avec Symfony 8 permettant d'importer un fichier CSV dans une base de données MySQL en utilisant PDO et la bibliothèque `league/csv`.

L'import doit :

— Lire un fichier CSV avec en-têtes,
— Insérer les données dans la table `customers`,
— Utiliser une transaction pour garantir l'intégrité,
— Optimiser les performances pour les gros volumes via insertion par batch,
— Fournir une trace détaillée (nombre inséré / nombre ignoré),
— Afficher le temps de début, de fin et d'exécution.

3.1.2. Architecture modulaire et rôle des fichiers

Pour respecter les bonnes pratiques, l'import CSV est structuré en classes, chacune ayant un rôle spécifique :

— `DatabaseConnection.php` : encapsule la connexion PDO, la gestion des transactions et la configuration de la base de données.
— `CsvReader.php` : lit le fichier CSV, gère l'encodage, les en-têtes et fournit un itérateur sur les lignes.
— `CsvImporter.php` : insère les données en base, applique la validation, gère les batchs et trace les statistiques (insérées / ignorées).
— `ImportCsvCommand.php` : commande Symfony CLI qui orchestre l'import, initialise les classes précédentes et affiche les informations de traçabilité.

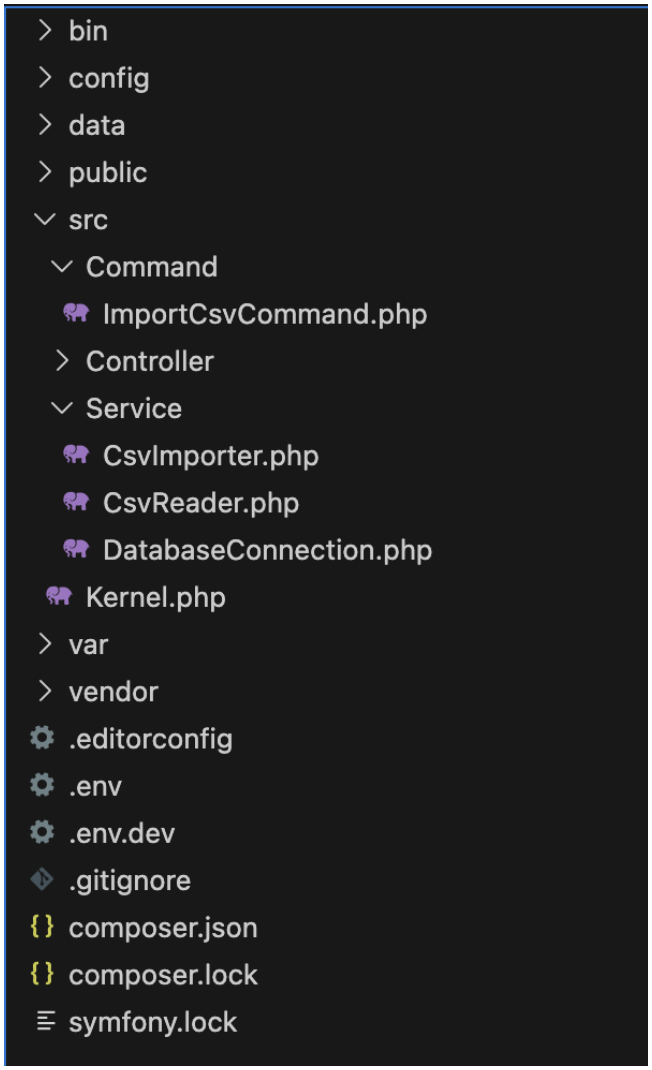


FIGURE 3.1 – Structure modulaire de l'import CSV

3.1.3. Création de la base de données et de l'utilisateur

Avant l'import, il est nécessaire de créer la base et l'utilisateur :

```
mysql> CREATE DATABASE csv_import_db;
Query OK, 1 row affected (0.01 sec)

mysql> CREATE USER 'hiba'@'localhost' IDENTIFIED BY '123';
Query OK, 0 rows affected (0.00 sec)

mysql> GRANT ALL PRIVILEGES ON csv_import_db.* TO 'hiba'@'localhost';
Query OK, 0 rows affected (0.01 sec)

mysql> FLUSH PRIVILEGES;
Query OK, 0 rows affected (0.00 sec)
```

3.1.4. Stratégies d'insertion en base

Insertion ligne par ligne

```
foreach ($csv as $row) {
    $stmt->execute([...]);
}
```

Simple mais inefficace pour les gros volumes.

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

void@192 csv_import_cli % php bin/console app:import-csv data/customers.csv
Import finished in 0.6 seconds
void@192 csv_import_cli %
```

Insertion par batch Pour améliorer les performances, les insertions sont regroupées par paquets (ex. 500 lignes) :

```php
public function __construct(PDO $pdo, int $batchSize = 500)
{
    $this->pdo = $pdo;
    $this->batchSize = $batchSize;
}

public function import(array $rows): array
{
    $inserted = 0;
    $skipped = 0;
    $batch = [];

    $this->pdo->beginTransaction();

    foreach ($rows as $row) {
        // Validation simple
        if (empty($row['Customer Id']) || empty($row['Email'])) {
            $skipped++;
            continue;
        }

        $batch[] = $row;

        if (count($batch) >= $this->batchSize) {
            [$i, $s] = $this->insertBatch($batch);
            $inserted += $i;
            $skipped += $s;
            $batch = [];
        }
    }
```

Avantages :

— Meilleure performance pour les gros fichiers,

— Transactions gérées de manière efficace,

— Statistiques de traçabilité par batch.

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

void@192 csv_import_cli % php bin/console app:import-csv data/customers.csv
Import completed in 0.22 seconds: 10000 inserted, 0 skipped.
void@192 csv_import_cli %
```

Chapitre 4

REST API Using Laravel

4.1. Création d'une API REST avec Laravel

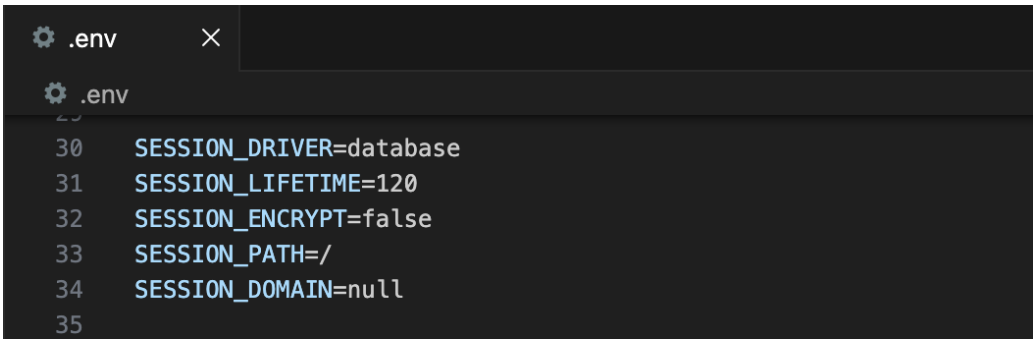
4.1.1. Objectif

L'objectif de cet exercice est de développer une API REST en utilisant **Laravel**, permettant de gérer les informations des clients stockées dans une base de données MySQL. L'API doit respecter les bonnes pratiques suivantes :

- Pagination des résultats
- Filtrage et tri des données via des paramètres de requête
- Ajout d'un header personnalisé `x-api-version`
- Sécurisation des endpoints avec une authentification Basic Auth

4.1.2. Configuration de l'environnement

L'application a été développée sur un serveur local Apache avec MySQL. Laravel a été installé via Composer et la base de données configurée dans le fichier `.env` :



4.1.3. Structure des fichiers importants

- **routes/api.php** : contient les routes de l'API, par exemple la route `/api/customers` pour récupérer les clients.
- **app/Models/Customer.php** : modèle Eloquent correspondant à la table `customers`, configuré avec `customer_id` comme clé primaire.
- **app/Http/Controllers/CustomerController.php** : contrôleur qui gère la logique des endpoints, incluant la pagination, le filtrage, le tri et l'ajout du header `x-api-version`.

— **app/Http/Middleware/BasicAuthMiddleware.php** : middleware personnalisé permettant de sécuriser l'API avec un nom d'utilisateur et un mot de passe fixes.

4.1.4. Fonctionnalités de l'API

— **Pagination** : les résultats sont paginés pour limiter le nombre de clients retournés par requête.

```
    $pagesize = $request->get('pagesize', 10);
    $cutomers = $query->paginate($pagesize);

    return response()->json($cutomers)->header('x-api-version', 'v1');
}
```

— **Filtrage et tri** : les clients peuvent être filtrés et triés par plusieurs champs grâce aux paramètres de la requête.

```
if ($request->has('filter')) {
    foreach ($request->filter as $field => $value) {
        $query->where($field, 'like', "%$value%");
    }
}

if ($request->has('sort')) {
    foreach ($request->sort as $field => $direction) {
        $query->orderBy($field, $direction);
    }
}
```

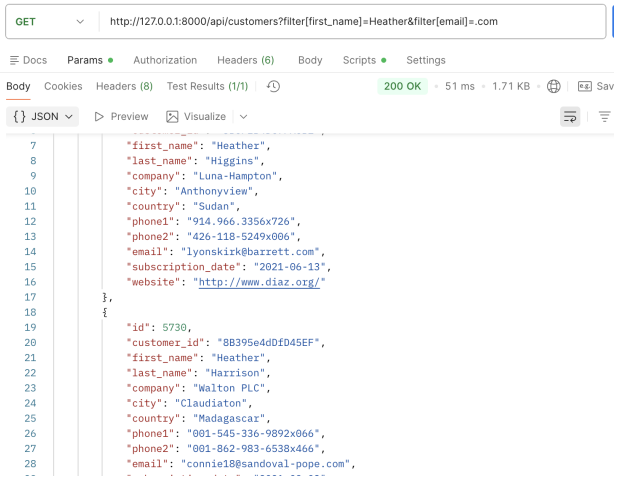
— **Header personnalisé** : chaque réponse contient le header `x-api-version: v1`.

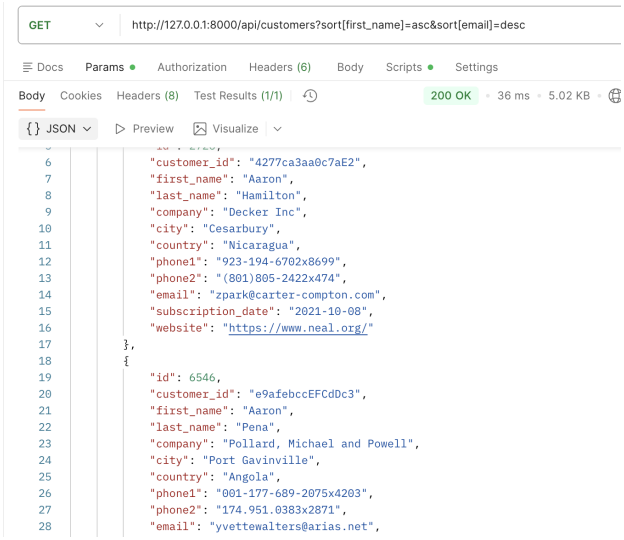
```
return response()->json($cutomers)->header('x-api-version', 'v1');
```

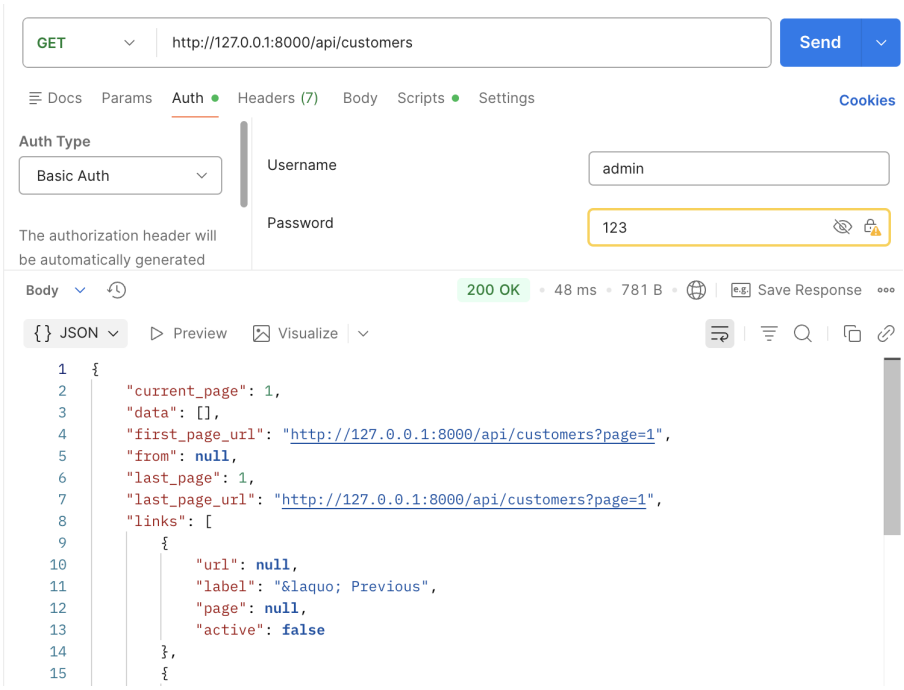
— **Authentification** : l'accès est protégé par Basic Auth. Seuls les utilisateurs connaissant le couple `username/password` peuvent accéder aux endpoints.

```
class BasicAuthMiddleware
{
    /**
     * Handle an incoming request.
     *
     * @param  \Closure(\Illuminate\Http\Request): (\Symfony\Component\HttpFoundation\Response)  $next
     */
    public function handle(Request $request, Closure $next): Response
    {
        $USERNAME = 'admin';
        $PASSWORD = '123';

        $has_auth = $request->getUser() === $USERNAME && $request->getPassword() === $PASSWORD;

        if (!$has_auth) {
            return response()->json(['message' => 'Unauthorized'], 401, ['WWW-Authenticate' => 'Basic']);
        }
        return $next($request);
    }
}
```

4.1.5. Resultat







Dépôts GitHub

Les exercices développés dans le cadre de ce sprint sont disponibles sur GitHub.

- **Dummy Package** : https://github.com/Hibat-Allah-RGUITI/dummy-config.git
- **Simple PHP MVC Framework** : https://github.com/Hibat-Allah-RGUITI/PhpMVCFramework.git
- **Databases & CLI)** : https://github.com/Hibat-Allah-RGUITI/symfony-csv-import-cli.git
- **REST API Using Laravel** : https://github.com/Hibat-Allah-RGUITI/REST-API.git