🔴 Projet : Inventaire de parc informatique (v1)

Classe : 5TQ Informatique
Durée : 2 heures en classe
Travail : Individuel
Prérequis : Syntaxe PHP de base, `include`, Bootstrap 5, notions PowerShell

---

🎯 Objectifs du projet

#	Objectif
1	Comprendre ce qu'est un **inventaire de parc informatique** et pourquoi c'est indispensable
2	**Collecter des informations système** avec PowerShell et les envoyer vers un serveur
3	**Créer une API REST minimaliste** en PHP pour recevoir des données JSON
4	**Afficher un dashboard** et une fiche détaillée par machine

---

📚 Les 5 notions-clés

Avant de coder, lis attentivement ces 5 notions. Elles sont le cœur du projet.

1. 📦 L'inventaire de parc informatique

Dans toute entreprise ou école, il y a des dizaines (voire des centaines) d'ordinateurs. Un **inventaire de parc informatique** permet de savoir :

- Quelles machines existent et où elles se trouvent
- Leur configuration matérielle (RAM, CPU…)
- Quel système d'exploitation est installé
- Qui les utilise

Sans inventaire, impossible de planifier les mises à jour, de détecter une machine en panne ou de gérer les licences logicielles. C'est un outil de base pour tout technicien informatique.

---

2. 🔌 Une API REST

Une **API REST** est un service web qui reçoit des requêtes HTTP et répond en JSON.
Dans ce projet, le serveur PHP joue le rôle d'une API : il **écoute** les données envoyées par les postes.

```
[Poste A] ──POST JSON──▶ http://serveur/inventaire/api/enregistrer.php
[Poste B] ──POST JSON──▶ http://serveur/inventaire/api/enregistrer.php
[Poste C] ──POST JSON──▶ http://serveur/inventaire/api/enregistrer.php
```

Le verbe HTTP utilisé est **POST** car on **envoie** des données (comme remplir un formulaire, mais depuis un script).

---

3. 📄 JSON comme format d'échange

Le **JSON** (JavaScript Object Notation) est le format universel pour échanger des données entre systèmes. PowerShell peut produire du JSON avec `ConvertTo-Json`. PHP peut le lire avec `json_decode()`.

```json
{
  "hostname": "PC-LABO-12",
  "host": {
    "ram_go": 8,
    "constructeur": "Dell Inc."
  }
}
```

---

4. 💾 Stocker sans base de données

En version 1, on stocke les données dans des **fichiers JSON sur le serveur**.
Chaque machine a son propre fichier, nommé d'après son hostname :

```
data/
├── PC-LABO-01.json
├── PC-LABO-02.json
└── PC-LABO-12.json
```

👉 Le **hostname** est l'identifiant unique de la machine. Si la même machine envoie ses infos deux fois, le fichier est **écrasé** (mis à jour).

---

5. 🔁 `php://input` pour lire un POST en JSON

Quand PHP reçoit un formulaire HTML classique, on lit les données avec `$_POST`.
Mais quand on reçoit du **JSON brut** (depuis PowerShell ou Postman), les données ne passent pas par `$_POST`.
Il faut lire le **corps brut de la requête** avec :

```php
$contenu = file_get_contents('php://input');
$donnees = json_decode($contenu, true); // true = tableau associatif
```

C'est la particularité des API REST : les données arrivent dans le "body" de la requête, pas dans les paramètres habituels.

🏗️ Structure du projet

Crée cette arborescence dans ton dossier `htdocs` de Laragon :

```
inventaire/
├── index.php              ← Dashboard : liste de tous les postes
├── poste.php              ← Fiche détaillée d'un poste
├── includes/
│   ├── header.php         ← En-tête HTML commune
│   └── footer.php         ← Fermeture HTML commune
├── api/
│   └── enregistrer.php    ← Endpoint REST (reçoit le JSON)
└── data/                  ← Créé automatiquement (un .json par poste)
```

⚠️ Le dossier `data/` sera créé **automatiquement** par PHP. Tu n'as pas besoin de le créer manuellement.

📋 Format JSON envoyé par PowerShell

Voici exactement le JSON que le script PowerShell envoie au serveur.
Garde cette structure sous les yeux pendant tout le projet : elle détermine comment lire les données en PHP.

```json
{
  "hostname": "PC-LABO-12",
  "date_scan": "2026-04-22 10:35:00",
  "host": {
    "nom_machine": "PC-LABO-12",
    "constructeur": "Dell Inc.",
    "modele": "OptiPlex 7080",
    "ram_go": 16,
    "hyperviseur": false,
    "nom_dns": "PC-LABO-12.cepes.local",
    "utilisateur": "CEPES\\eleve05"
  },
  "cpu": [
    {
      "modele": "Intel(R) Core(TM) i5-10400 CPU @ 2.90GHz",
      "coeurs": 6,
      "threads": 12,
      "identifiant": "BFEBFBFF000A0653",
      "designation_courte": "Intel64 Family 6 Model 165 Stepping 3",
      "fabricant": "GenuineIntel"
    }
  ],
  "os": {
    "nom": "Microsoft Windows 11 Éducation",
    "version": "10.0.22621",
    "build": "22621",
    "langues_mui": "fr-BE",
    "organisation": "CEPES Jodoigne",
    "architecture": "64 bits"
  }
}
```

💡 **Remarque :** La clé `cpu` est un **tableau** (entre `[ ]`) car un ordinateur peut avoir plusieurs processeurs. En PHP, on y accède avec `$poste['cpu'][0]` pour le premier.

🛠️ Étape 1 – Le script PowerShell

Durée estimée : 30 min

Le script PowerShell tourne sur **chaque poste client**. Il collecte les infos et les envoie au serveur.

Enregistre ce fichier sous `inventaire_poste.ps1` :

```powershell
# ================================================
# INVENTAIRE POSTE - Script PowerShell
# Collecte les infos système et les envoie au serveur
```

```powershell
# ====================================================

# ---- Informations générales du poste (host) ----
$sys = Get-WmiObject Win32_ComputerSystem

$donnees_host = @{
    nom_machine  = $env:COMPUTERNAME
    constructeur = $sys.Manufacturer
    modele       = $sys.Model
    ram_go       = [math]::Round($sys.TotalPhysicalMemory / 1GB, 2)
    hyperviseur  = [bool]$sys.HypervisorPresent
    nom_dns      = $sys.DNSHostName
    utilisateur  = $sys.UserName
}

# ---- Processeur(s) (cpu) ----
# Get-WmiObject peut retourner plusieurs CPU → on boucle avec ForEach-Object
$donnees_cpu = @(
    Get-WmiObject Win32_Processor | ForEach-Object {
        @{
            modele             = $_.Name.Trim()
            coeurs             = $_.NumberOfCores
            threads            = $_.NumberOfLogicalProcessors
            identifiant        = $_.ProcessorId.Trim()
            designation_courte = $_.Caption
            fabricant          = $_.Manufacturer
        }
    }
)

# ---- Système d'exploitation (os) ----
$sys_os = Get-WmiObject Win32_OperatingSystem

$donnees_os = @{
    nom          = $sys_os.Caption
    version      = $sys_os.Version
    build        = $sys_os.BuildNumber
    langues_mui  = ($sys_os.MUILanguages -join ", ")
    organisation = $sys_os.Organization
    architecture = $sys_os.OSArchitecture
}

# ---- Assemblage du JSON final ----
$payload = @{
    hostname = $env:COMPUTERNAME
    host     = $donnees_host
    cpu      = $donnees_cpu
    os       = $donnees_os
}

# ConvertTo-Json -Depth 5 car on a des objets imbriqués (cpu est un tableau d'objets)
$json = $payload | ConvertTo-Json -Depth 5

# ---- Envoi vers le serveur PHP ----
# ⚠️ Remplace XX.XX par l'adresse IP du serveur (le PC du prof / le serveur Laragon)
$url = "http://192.168.1.XX/inventaire/api/enregistrer.php"

```

```powershell
Write-Host "📡 Envoi des données vers $url ..."

try {
    $reponse = Invoke-RestMethod -Uri $url -Method POST -Body $json -ContentType "application/
    Write-Host "✅ Succès ! Fichier enregistré : $($reponse.fichier)"
} catch {
    Write-Host "❌ Erreur lors de l'envoi : $_"
}
```

Comment exécuter le script

1. Ouvre **PowerShell en tant qu'administrateur**
2. Si nécessaire, autorise l'exécution : `Set-ExecutionPolicy RemoteSigned`
3. Lance le script : `.\inventaire_poste.ps1`

🛠️ Étape 2 – L'API PHP (`api/enregistrer.php`)

Durée estimée : 20 min

Ce fichier est le **récepteur**. Il tourne sur le serveur et attend les données des postes.

```php
<?php
// ================================================
// api/enregistrer.php
// Reçoit un JSON depuis un poste et le sauvegarde
// ================================================

// Indique que notre réponse sera du JSON
header('Content-Type: application/json');

// Étape 1 : lire le corps brut de la requête POST
// (pas $_POST car les données arrivent en JSON, pas en formulaire)
$contenu = file_get_contents('php://input');

// Étape 2 : décoder le JSON en tableau PHP
// Le "true" transforme les objets JSON en tableaux associatifs PHP
$donnees = json_decode($contenu, true);

// Étape 3 : vérification minimale
// On vérifie que le JSON est valide ET qu'il contient un hostname
if ($donnees === null || empty($donnees['hostname'])) {
    http_response_code(400); // 400 = Bad Request
    echo json_encode(['erreur' => 'Données invalides ou hostname manquant']);
    exit;
}

// Étape 4 : ajouter la date du scan (côté serveur, plus fiable)
$donnees['date_scan'] = date('Y-m-d H:i:s');

// Étape 5 : créer le dossier data/ s'il n'existe pas encore
```

```php
if (!is_dir('data')) {
    mkdir('data', 0777, true);
}

// Étape 6 : nettoyer le hostname pour en faire un nom de fichier valide
// preg_replace remplace tout caractère NON alphanumérique (sauf - et _) par _
$hostname_propre = preg_replace('/[^a-zA-Z0-9\-_]/', '_', $donnees['hostname']);
$fichier = 'data/' . $hostname_propre . '.json';

// Étape 7 : sauvegarder le JSON dans le fichier
// JSON_PRETTY_PRINT = lisible par un humain
// JSON_UNESCAPED_UNICODE = garde les accents (é, à, ê...)
file_put_contents($fichier, json_encode($donnees, JSON_PRETTY_PRINT | JSON_UNESCAPED_UNICODE));

// Étape 8 : répondre au client PowerShell
http_response_code(200); // 200 = OK
echo json_encode([
    'succes'  => true,
    'fichier' => $fichier,
    'message' => 'Poste ' . $donnees['hostname'] . ' enregistré avec succès'
]);
```

✅ Comment tester sans PowerShell

Tu peux tester l'API avec **Postman** ou **Insomnia** :

- Méthode : `POST`
- URL : `http://localhost/inventaire/api/enregistrer.php`
- Body : `raw` → `JSON` → colle l'exemple JSON de la section précédente

Si tout fonctionne, tu verras apparaître un fichier dans `data/`.

🛠️ Étape 3 – Les includes (`includes/`)

```php
<?php // includes/header.php ?>
<!DOCTYPE html>
<html lang="fr">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <title>Inventaire Parc Informatique</title>
</head>
<body>

<nav>
    <a href="index.php">Dashboard</a>
</nav>

<main>
```

```php
<?php // includes/footer.php ?>
</main>

<footer>
    <p>Inventaire Parc Informatique – 5TQ Informatique</p>
</footer>

</body>
</html>
```

💡 La mise en forme (CSS, Bootstrap…) est **à ta charge**. Ces includes fournissent uniquement la structure HTML minimale.

🛠️ Étape 4 – Le dashboard (`index.php`)

Durée estimée : 25 min

La page principale liste tous les postes inventoriés dans un tableau. Chaque ligne pointe vers la fiche détaillée.

```php
<?php
// ============================================
// index.php – Dashboard de l'inventaire
// ============================================

include 'includes/header.php';

// Étape 1 : lire tous les fichiers .json dans data/
// glob() retourne un tableau avec les chemins de tous les fichiers correspondants
$fichiers = glob('data/*.json');

// Étape 2 : charger les données de chaque fichier
$postes = [];

foreach ($fichiers as $chemin) {
    $contenu = file_get_contents($chemin);
    $poste   = json_decode($contenu, true);

    if ($poste !== null) {
        $postes[] = $poste;
    }
}

// Étape 3 : quelques chiffres résumés
$nb_postes = count($postes);

$total_ram = 0;
foreach ($postes as $p) {
    $total_ram += $p['host']['ram_go'] ?? 0;
}
$ram_moyenne = ($nb_postes > 0) ? round($total_ram / $nb_postes, 1) : 0;
?>

```

```php
<h1>Dashboard – Parc Informatique</h1>

<p>Postes inventoriés : <strong><?php echo $nb_postes; ?></strong></p>
<p>RAM moyenne : <strong><?php echo $ram_moyenne; ?> Go</strong></p>

<hr>

<?php if ($nb_postes === 0): ?>

    <p>Aucun poste inventorié. Lance le script PowerShell sur un poste !</p>

<?php else: ?>

    <table border="1">
        <thead>
            <tr>
                <th>Hostname</th>
                <th>Constructeur</th>
                <th>Modèle</th>
                <th>RAM (Go)</th>
                <th>OS</th>
                <th>Dernier scan</th>
                <th>Détail</th>
            </tr>
        </thead>
        <tbody>
            <?php foreach ($postes as $poste): ?>
                <tr>
                    <td><?php echo htmlspecialchars($poste['hostname']                ?? ''); ?></td>
                    <td><?php echo htmlspecialchars($poste['host']['constructeur']    ?? ''); ?></td>
                    <td><?php echo htmlspecialchars($poste['host']['modele']          ?? ''); ?></td>
                    <td><?php echo htmlspecialchars($poste['host']['ram_go']          ?? ''); ?></td>
                    <td><?php echo htmlspecialchars($poste['os']['nom']               ?? ''); ?></td>
                    <td><?php echo htmlspecialchars($poste['date_scan']               ?? ''); ?></td>
                    <td>
                        <a href="poste.php?hostname=<?php echo urlencode($poste['hostname']);
                            Voir
                        </a>
                    </td>
                </tr>
            <?php endforeach; ?>
        </tbody>
    </table>

<?php endif; ?>

<?php include 'includes/footer.php'; ?>
```

🛠️ Étape 5 – La fiche poste (`poste.php`)

Durée estimée : 25 min

Cette page reçoit un `hostname` en paramètre GET, charge le fichier JSON correspondant et affiche toutes ses informations.

```php
<?php
// ============================================
// poste.php – Fiche détaillée d'un poste
// ============================================

include 'includes/header.php';

// Étape 1 : récupérer le hostname depuis l'URL  ex: poste.php?hostname=PC-LABO-12
$hostname = $_GET['hostname'] ?? '';

// Étape 2 : sécuriser le nom et construire le chemin du fichier
$hostname_propre = preg_replace('/[^a-zA-Z0-9\-_]/', '_', $hostname);
$fichier = 'data/' . $hostname_propre . '.json';

// Étape 3 : vérifier que le fichier existe
if (!file_exists($fichier)) {
    echo '<p>Poste introuvable.</p>';
    include 'includes/footer.php';
    exit;
}

// Étape 4 : charger les données
$poste = json_decode(file_get_contents($fichier), true);
$host  = $poste['host'] ?? [];
$cpu   = $poste['cpu']  ?? [];
$os    = $poste['os']   ?? [];
?>

<h1>Fiche poste : <?php echo htmlspecialchars($poste['hostname'] ?? ''); ?></h1>
<p>Dernier scan : <?php echo htmlspecialchars($poste['date_scan'] ?? ''); ?></p>

<a href="index.php">← Retour au dashboard</a>

<hr>

<!-- Informations générales -->
<h2>Informations générales</h2>
<table border="1">
    <tr><th>Nom de la machine</th>    <td><?php echo htmlspecialchars($host['nom_machine']  ??
    <tr><th>Constructeur</th>         <td><?php echo htmlspecialchars($host['constructeur'] ??
    <tr><th>Modèle</th>               <td><?php echo htmlspecialchars($host['modele']       ??
    <tr><th>RAM totale</th>           <td><?php echo htmlspecialchars($host['ram_go']       ??
    <tr><th>Hyperviseur</th>          <td><?php echo $host['hyperviseur'] ? 'Oui' : 'Non'; ?><
    <tr><th>Nom DNS</th>              <td><?php echo htmlspecialchars($host['nom_dns']      ??
    <tr><th>Utilisateur connecté</th> <td><?php echo htmlspecialchars($host['utilisateur']  ??
</table>

<hr>

<!-- Processeur(s) -->
<h2>Processeur(s)</h2>

<?php if (empty($cpu)): ?>
```

```
52      <p>Aucune information CPU disponible.</p>
53  <?php else: ?>
54      <?php foreach ($cpu as $index => $proc): ?>
55          <h3>CPU #<?php echo $index + 1; ?></h3>
56          <table border="1">
57              <tr><th>Modèle</th>             <td><?php echo htmlspecialchars($proc['modele']
58              <tr><th>Fabricant</th>          <td><?php echo htmlspecialchars($proc['fabricant']
59              <tr><th>Cœurs</th>              <td><?php echo htmlspecialchars($proc['coeurs']
60              <tr><th>Threads</th>            <td><?php echo htmlspecialchars($proc['threads']
61              <tr><th>Identifiant</th>        <td><?php echo htmlspecialchars($proc['identifiant
62              <tr><th>Désignation courte</th> <td><?php echo htmlspecialchars($proc['designatior
63          </table>
64      <?php endforeach; ?>
65  <?php endif; ?>
66
67  <hr>
68
69  <!-- Système d'exploitation -->
70  <h2>Système d'exploitation</h2>
71  <table border="1">
72      <tr><th>Nom</th>          <td><?php echo htmlspecialchars($os['nom']          ?? ''); ?></
73      <tr><th>Version</th>      <td><?php echo htmlspecialchars($os['version']      ?? ''); ?></
74      <tr><th>Build</th>        <td><?php echo htmlspecialchars($os['build']        ?? ''); ?></
75      <tr><th>Langues MUI</th>  <td><?php echo htmlspecialchars($os['langues_mui']  ?? ''); ?></
76      <tr><th>Organisation</th> <td><?php echo htmlspecialchars($os['organisation'] ?? ''); ?></
77      <tr><th>Architecture</th> <td><?php echo htmlspecialchars($os['architecture'] ?? ''); ?></
78  </table>
79
80  <?php include 'includes/footer.php'; ?>
```

✅ Résumé des fichiers à créer

Fichier	Ce qu'il fait	Technologie
`inventaire_poste.ps1`	Collecte les infos et envoie le JSON	PowerShell
`api/enregistrer.php`	Reçoit le JSON et le sauvegarde	PHP
`includes/header.php`	En-tête HTML commune	PHP/HTML
`includes/footer.php`	Pied de page HTML commun	PHP/HTML
`index.php`	Dashboard : tableau de tous les postes	PHP/HTML
`poste.php`	Fiche détaillée d'un poste	PHP/HTML

🔎 Questions de compréhension

Réponds à ces questions dans un commentaire en haut de ton fichier `index.php` :

1. **Pourquoi utilise-t-on `php://input` et pas `$_POST` pour lire le JSON ?**
2. **Pourquoi le hostname est-il utilisé comme nom de fichier ?**
3. **Que se passe-t-il si le même poste envoie ses infos deux fois ?**
4. **Pourquoi la clé `cpu` est-elle un tableau et pas un objet simple ?**

5. **Quelle est la différence entre `json_encode()` et `json_decode()` ?**

Grille d'évaluation (V1)

Critère	Points
Script PowerShell : collecte correcte de `host`, `cpu`, `os`	/4
API PHP : réception et sauvegarde du JSON avec validation	/4
Dashboard (`index.php`) : tableau fonctionnel + lien vers fiche	/3
Fiche poste (`poste.php`) : affichage complet des 3 sections + lien retour	/3
Utilisation correcte de `include` (header/footer)	/2
Qualité du code : commentaires, lisibilité	/2
Questions de compréhension	/2
Total	**/20**