

Inventaire — Étape 2 : l'API PHP de réception

Étape 2 du projet inventaire : construire l'API PHP qui reçoit, valide et stocke le JSON envoyé par PowerShell.

5TQ

📶 Découverte

Tu vas construire le **côté serveur**. À la fin, ton script PowerShell pourra envoyer son rapport à `http://localhost/inventaire/api/report` au lieu de `webhook.site`, et le rapport sera **sauvegardé en JSON sur le disque**.

Pas de framework, pas de Composer, pas de base de données. Du PHP brut.

🎯 Objectif de l'étape

Construire un endpoint `POST /inventaire/api/report` qui :

1. n'accepte que la méthode **POST**,
2. décode le JSON reçu,
3. **valide** la structure,
4. **stocke** le rapport en deux fichiers : `latest.json` + un snapshot horodaté.

Partie 1 — Structure du projet

Crée dans `C:\laragon\www\inventaire` (ou ton équivalent) cette arborescence :

```
1 | inventaire/
2 |   └─ public/           # racine web (à pointer dans Laragon)
3 |     └─ index.php      # routeur très simple
4 |     └─ api/
5 |       └─ report.php   # endpoint POST de réception
6 |     └─ css/
7 |     └─ js/
8 |   └─ data/
9 |     └─ pcs/           # créé automatiquement, un sous-dossier par PC
10 |  └─ src/
11 |    └─ storage.php     # fonctions de lecture/écriture JSON
12 |    └─ validation.php  # fonctions de validation du payload
```

```
13 | | └─ auth.php           # vide pour l'instant – utilisé à l'étape 6
14 | | └─ README.md
```

💡 **POURQUOI PUBLIC/ ?** Pour qu'on puisse plus tard servir uniquement ce dossier au web et garder `data/` et `src/` inaccessibles depuis l'extérieur. C'est une bonne habitude dès le départ.

Partie 2 – Le routage minimal

Dans `public/index.php` :

```
<?php
1  declare(strict_types=1);
2
3  require __DIR__ . '/../src/storage.php';
4  require __DIR__ . '/../src/validation.php';
5
6  // On découpe l'URL pour savoir ce qu'on doit afficher
7  $path = parse_url($_SERVER['REQUEST_URI'], PHP_URL_PATH);
8  $path = rtrim($path, '/');
9
10 // Très simple : on regarde le début du chemin
11 if (str_starts_with($path, '/inventaire/api/report')) {
12     require __DIR__ . '/api/report.php';
13     exit;
14 }
15
16 if ($path === '/inventaire' || $path === '/inventaire/') {
17     require __DIR__ . '/dashboard.php'; // (sera créé à l'étape 4)
18     exit;
19 }
20
21 if (str_starts_with($path, '/inventaire/pc/')) {
22     require __DIR__ . '/pc.php'; // (sera créé à l'étape 3)
23     exit;
24 }
25
26 http_response_code(404);
27 echo "404 – Page non trouvée";
```

💡 Si tu trouves ça compliqué, sache que **C'EST PLUS SIMPLE QU'UN FRAMEWORK**. Tu fais en 15 lignes ce que Symfony fait en 50 fichiers.

Travailler avec l'IA – routage

Tu peux demander à l'IA :

Explique-moi `str_starts_with` en PHP, depuis quelle version c'est disponible, et donne-moi un équivalent compatible PHP 7 au cas où.

Pourquoi cette question ? Parce que **Laragon a peut-être PHP 8**, mais **l'IA va parfois te proposer du code PHP 5**. Tu dois savoir reconnaître le niveau de modernité du code qu'on te donne.

Partie 3 – L'endpoint de réception (version minimale)

Dans `public/api/report.php` :

```
<?php
1 declare(strict_types=1);
2
3 // Toujours en JSON
4 header('Content-Type: application/json; charset=utf-8');
5
6 // On n'accepte que les POST
7 if ($_SERVER['REQUEST_METHOD'] !== 'POST') {
8     http_response_code(405); // Method Not Allowed
9     echo json_encode(['error' => 'Méthode non autorisée. Utilise POST.']);
10    exit;
11 }
12
13 // Lecture du corps brut (PowerShell envoie du JSON, pas du form-url-encoded)
14 $raw = file_get_contents('php://input');
15
16 if ($raw === '' || $raw === false) {
17     http_response_code(400);
18     echo json_encode(['error' => 'Corps de requête vide.']);
19     exit;
20 }
21
22 // Décodage
23 $data = json_decode($raw, true);
24 if (json_last_error() !== JSON_ERROR_NONE) {
25     http_response_code(400);
26     echo json_encode([
27         'error' => 'JSON invalide.',
28         'detail' => json_last_error_msg()
29     ]);
30     exit;
31 }
32
```

```

33 // À ce stade, $data est un tableau PHP. On va valider puis stocker.
34 // (Les fonctions sont dans src/)
35
36 $erreurs = valider_rapport($data);
37 if (!empty($erreurs)) {
38     http_response_code(422); // Unprocessable Entity
39     echo json_encode(['error' => 'Données invalides.', 'detail' => $erreurs]);
40     exit;
41 }
42
43 try {
44     enregistrer_rapport($data);
45 } catch (Throwable $e) {
46     http_response_code(500);
47     echo json_encode(['error' => 'Erreur serveur.', 'detail' => $e->getMessage()]);
48     exit;
49 }
50
51 http_response_code(201); // Created
52 echo json_encode([
53     'ok'         => true,
54     'hostname'  => $data['hostname'],
55     'savedAt'   => date('c')
56 ]);

```

⚠ **POURQUOI FILE_GET_CONTENTS('PHP://INPUT') ET PAS \$_POST ?** Parce que `$_POST` ne contient que les données envoyées en `application/x-www-form-urlencoded` (un formulaire HTML classique). Quand tu envoies du JSON, PHP ne le parse pas tout seul. Tu lis le flux brut, tu le décodes toi-même.

Partie 4 – La validation

Dans `src/validation.php`. **Vraiment simple**, on vérifie juste que les champs essentiels existent :

```

<?php
1
2 function valider_rapport(array $data): array
3 {
4     $erreurs = [];
5
6     if (empty($data['hostname']) || !is_string($data['hostname'])) {
7         $erreurs[] = "Champ 'hostname' manquant ou invalide.";
8     } elseif (!preg_match('/^[a-zA-Z0-9_\-]{1,64}$/', $data['hostname'])) {
9         // Sécurité : on n'autorise QUE lettres, chiffres, _, -
10        $erreurs[] = "Hostname contient des caractères interdits.";
11    }
12

```

```

13     if (empty($data['timestamp'])) {
14         $erreurs[] = "Champ 'timestamp' manquant.";
15     }
16
17     foreach (['system', 'memory'] as $clef) {
18         if (!isset($data[$clef]) || !is_array($data[$clef])) {
19             $erreurs[] = "Section '$clef' manquante.";
20         }
21     }
22
23     return $erreurs;
24 }

```

⚠ **LE PIÈGE DE SÉCURITÉ MAJEUR** : le hostname va devenir un **nom de dossier**. Si un attaquant envoie `hostname = "../../../etc/passwd"`, il peut écrire **n'importe où** sur ton disque. La regex ci-dessus empêche ça. **Ne supprime jamais cette ligne.**

Partie 5 – Le stockage

Dans `src/storage.php` :

```

<?php
1
2     const DATA_ROOT = __DIR__ . '/../data/pcs';
3
4     function enregistrer_rapport(array $data): void
5     {
6         $hostname = $data['hostname'];
7         $dossier = DATA_ROOT . '/' . $hostname;
8
9         // Crée le dossier (et l'historique) si nécessaire
10        if (!is_dir($dossier . '/history')) {
11            mkdir($dossier . '/history', 0775, true);
12        }
13
14        // 1) Snapshot horodaté dans l'historique
15        $timestamp = date('Ymd-His'); // ex : 20260513-103015
16        $fichierHistorique = $dossier . "/history/$timestamp.json";
17        file_put_contents(
18            $fichierHistorique,
19            json_encode($data, JSON_PRETTY_PRINT | JSON_UNESCAPED_UNICODE)
20        );
21
22        // 2) Latest = dernière version connue (écrasée à chaque fois)
23        $fichierLatest = $dossier . '/latest.json';
24        file_put_contents(
25            $fichierLatest,

```

```
26 |         json_encode($data, JSON_PRETTY_PRINT | JSON_UNESCAPED_UNICODE)
27 |     );
28 | }
```

💡 **POURQUOI DEUX FICHIERS ?** Le `latest.json` est lu **très souvent** (dashboard, fiche PC). On veut un accès ultra-rapide à la dernière valeur. L'historique sert au graphe d'évolution. C'est un compromis classique entre **rapidité de lecture** et **conservation des données**.

Travailler avec l'IA – discussions de design

Tu peux demander :

Pourquoi stocker à la fois un `latest.json` et un dossier `history/` ? Quel est le compromis ? Quand est-ce que ce serait une mauvaise idée ?

C'est une **bonne** question à poser à l'IA, parce qu'elle t'explique des **alternatives** (BDD, log unique append-only, fichier par jour...) sans te livrer du code. Tu apprends à raisonner en architecte.

Mauvaise question :

Refais mon stockage en utilisant SQLite à la place.

Trop tôt, et hors-sujet pédagogique. Reste sur le fichier.

Partie 6 – Tester l'API

Tu as **trois manières** de tester ; utilise les trois, dans cet ordre :

a) En ligne de commande, avec PowerShell

```
1 | $body = @{ hostname = "TEST"; timestamp = (Get-Date).ToString("o");
2 |           system = @{}; memory = @{} } | ConvertTo-Json
3 |
4 | Invoke-RestMethod -Uri "http://localhost/inventaire/api/report" `
5 |                 -Method Post `
6 |                 -ContentType "application/json" `
7 |                 -Body $body
```

Tu dois recevoir `{ ok = True; hostname = TEST }`.

b) Avec Thunder Client (extension VS Code)

Encore plus pratique : tu peux enregistrer la requête, la rejouer, et voir le code de statut.

c) En modifiant volontairement le payload

Envoie successivement :

1. Un POST vide → tu dois obtenir un **400**.
2. Un POST avec un JSON cassé → **400** avec `detail`.
3. Un POST sans `hostname` → **422**.
4. Un POST avec `hostname = "../etc"` → **422** (validation refuse).

Si une de ces 4 attaques aboutit à un **200**, tu as un bug à corriger.

Partie 7 – Brancher PowerShell sur ton API

Modifie le `-ApiUrl` de ton script de l'étape 1 :

```
1 | .\inventaire.ps1 -ApiUrl "http://localhost/inventaire/api/report"
```

Le serveur PHP doit te répondre `{ ok: true, hostname: "...", savedAt: "..." }`, et tu dois trouver un fichier dans `C:\laragon\www\inventaire\data\pcs\TON-PC\latest.json`.

Ouvre ce fichier et vérifie qu'il est lisible, bien indenté, et que les accents passent bien.

✓ Critères de réussite

- [] L'arborescence est conforme.
- [] Un POST vide / mal formé renvoie un **400** ou **422**, jamais 500.
- [] Un POST correct renvoie un **201** et crée bien les fichiers.
- [] Le fichier `latest.json` est écrit en UTF-8, lisible.
- [] L'historique se remplit à chaque envoi.
- [] Tu peux expliquer **pourquoi** on lit `php://input` au lieu de `$_POST`.

⚠ Récap des pièges

Piège	Solution
Encodage cassé côté PowerShell	<code>-Encoding utf8</code> en sortie, <code>charset=utf-8</code> en header
<code>\$_POST</code> vide alors qu'on envoie du JSON	Lire <code>php://input</code>
<code>ConvertTo-Json</code> perd les sous-objets	<code>-Depth 6</code> minimum
Hostname <code>..\..\..\.</code> qui écrit n'importe où	Regex de validation stricte
Dossier <code>data/</code> accessible depuis le web	Mettre <code>data/</code> hors de <code>public/</code>

Suite

À l'[étape 3](#), on lit ce JSON et on l'affiche dans une **belle fiche PC**.