Étape 5 — Ton module spécifique

Ajouter un domaine d'expertise à ton inventaire : commandes PowerShell, structure JSON, affichage dédié.

5TTR | Exercice ambitieux

Objectif de l'étape

Chaque élève apporte un **domaine d'expertise** au parc. Ton module :

1. ajoute une **section dans le JSON** envoyé par PowerShell,
2. ajoute une **section dédiée** dans la fiche PC,
3. (bonus) contribue à un indicateur global sur le dashboard.

À la fin, **tous les modules sont visibles dans le même dashboard** — chacun voit le résultat des autres.

La liste des modules

Ton numéro est inscrit au tableau. Voici les 8 modules :

N°	Module	Slug	Question à éclairer
1	Réseau & IP	`network`	Mon PC est-il bien configuré pour le réseau ?
2	Sécurité & patchs	`security`	Mon PC est-il à jour ? Defender est-il actif ?
3	Utilisateurs & groupes	`users`	Qui a les droits sur mon PC ?
4	Services & démarrage	`startup`	Qu'est-ce qui démarre avec Windows ?
5	Périphériques USB	`usb`	Qu'est-ce qui est branché en USB ?
6	Tâches planifiées	`scheduler`	Qu'est-ce qui se déclenche tout seul ?
7	Connexions actives	`connections`	Mon PC parle-t-il à des inconnus ?
8	Santé du disque	`health`	Combien de temps va vivre mon disque ?

La structure imposée

Tous les modules doivent respecter cette enveloppe :

```
{
  "module": {
```

```
      "name": "network",
      "summary": "Une phrase humaine (15 mots max) résumant l'état observé",
      "stats": {
        "primaryMetric": "valeur clé chiffrée",
        "primaryLabel":  "ce que représente la valeur"
      },
      "data": { /* dépend du module */ }
    }
  }
```

Le `summary` et la `primaryMetric` sont affichés **directement dans la carte du dashboard**. Le `data` est affiché en détail sur la fiche PC.

> 💡 Pourquoi cette enveloppe ? Pour que le **PHP DU DASHBOARD** puisse afficher quelque chose **sans connaître les modules à l'avance**. C'est de l'**extensibilité par convention** — un principe qu'on retrouve dans tous les gros logiciels.

Module 1 — Réseau & IP (`network`)

Commandes PowerShell

```powershell
$adaptateurs = Get-NetAdapter |
    Where-Object Status -eq 'Up' |
    Select-Object Name, InterfaceDescription, LinkSpeed, MacAddress

$ips = Get-NetIPAddress -AddressFamily IPv4 |
    Where-Object { $_.IPAddress -notlike '169.*' -and $_.IPAddress -ne '127
    Select-Object InterfaceAlias, IPAddress, PrefixLength

$dns = Get-DnsClientServerAddress -AddressFamily IPv4 |
    Where-Object ServerAddresses |
    Select-Object InterfaceAlias, ServerAddresses
```

Stats à exposer

- `primaryMetric` : nombre d'adaptateurs actifs
- `primaryLabel` : "interfaces actives"
- `summary` : ex. *"3 interfaces actives, IP principale 192.168.1.42"*

Affichage PHP suggéré

Tableau des adaptateurs, badge pour chaque IP, liste des serveurs DNS.

Module 2 — Sécurité & patchs (`security`)

Commandes

```powershell
$hotfixes = Get-HotFix | Select-Object HotFixID, Description, InstalledOn |
    Sort-Object InstalledOn -Descending |
    Select-Object -First 20

$defender = Get-MpComputerStatus |
    Select-Object AntivirusEnabled, RealTimeProtectionEnabled,
                  AntivirusSignatureLastUpdated, AMServiceEnabled
```

Stats

- `primaryMetric` : nombre de jours depuis le dernier patch
- `summary` : *"Defender actif, dernier patch il y a 12 jours"*

Pièges

- `Get-HotFix` ne montre que les patchs `wusa`. Pour Windows 10/11 récents, `Get-WindowsUpdate` (du module `PSWindowsUpdate`) est plus complet, mais demande une installation. **Reste sur** `Get-HotFix` pour le TP.

Module 3 — Utilisateurs & groupes (`users`)

Commandes

```powershell
$users = Get-LocalUser |
    Select-Object Name, Enabled, LastLogon, PasswordRequired
```

```powershell

$admins = Get-LocalGroupMember -Group 'Administrators' -ErrorAction Silentl
    Select-Object Name, PrincipalSource

$groups = Get-LocalGroup | Select-Object Name, Description
```

Stats

- `primaryMetric` : nombre d'administrateurs locaux
- `summary` : *"4 utilisateurs locaux, 2 admins"*

Angle pédagogique

C'est l'occasion de discuter du **principe de moindre privilège**. Un PC avec 5 administrateurs locaux est une mauvaise idée. Tu peux **colorer en rouge** les comptes admin dans la fiche PC.

Module 4 — Services & démarrage (`startup`)

Commandes

```powershell
$servicesActifs = Get-Service |
    Where-Object Status -eq 'Running' |
    Select-Object Name, DisplayName, StartType

$startups = Get-CimInstance Win32_StartupCommand |
    Select-Object Name, Command, Location, User
```

Stats

- `primaryMetric` : nombre de programmes au démarrage
- `summary` : *"42 services actifs, 8 programmes au démarrage"*

Réflexion

Quels programmes au démarrage sont vraiment nécessaires ? Pour ton mockup, mets un bouton « Suspicieux » qui surligne les `Command` pointant ailleurs que `C:\Program Files`.

Module 5 — Périphériques USB (`usb`)

Commandes

```powershell
$usb = Get-PnpDevice -PresentOnly |
    Where-Object Class -in @('USB','HIDClass','Camera','Bluetooth','Image') |
    Select-Object FriendlyName, Class, Manufacturer, Status
```

Stats

- `primaryMetric` : nombre de périphériques USB connectés
- `summary` : *"7 périphériques USB, dont 1 caméra et 1 clé"*

Bonus

Différencier visuellement les classes (icône clavier, souris, stockage…). Beau défi de design.

Module 6 — Tâches planifiées (`scheduler`)

Commandes

```powershell
$taches = Get-ScheduledTask |
    Where-Object State -eq 'Ready' |
    Select-Object TaskName, TaskPath, Author,
        @{N='LastRun'; E={(Get-ScheduledTaskInfo $_).LastRunTime}},
        @{N='NextRun'; E={(Get-ScheduledTaskInfo $_).NextRunTime}}
```

Stats

- `primaryMetric` : nombre de tâches prêtes à se déclencher
- `summary` : *"112 tâches planifiées, prochaine dans 2 heures"*

Piège

Il y a **beaucoup** de tâches Microsoft. Filtre `TaskPath -notlike '*\Microsoft\*'` si tu veux ne voir que les tâches tierces.

Module 7 — Connexions actives (`connections`)

Commandes

```powershell
$connexions = Get-NetTCPConnection -State Established |
    Select-Object LocalAddress, LocalPort, RemoteAddress, RemotePort,
        @{N='Process'; E={(Get-Process -Id $_.OwningProcess -ErrorAction Si
```

Stats

- `primaryMetric` : nombre de connexions sortantes ouvertes
- `summary` : *"23 connexions actives, 8 processus différents"*

Bonus

Regrouper par `Process` et compter. Top 5 des processus qui communiquent le plus.

Module 8 — Santé du disque (`health`)

Commandes

```powershell
$physical = Get-PhysicalDisk |
    Select-Object FriendlyName, MediaType, HealthStatus,
        @{N='SizeGo'; E={[math]::Round($_.Size/1GB,1)}},
        Temperature, Usage

```

```
    $reliability = Get-PhysicalDisk | Get-StorageReliabilityCounter |
        Select-Object Temperature, ReadErrorsTotal, WriteErrorsTotal,
                      PowerOnHours, Wear
```

Stats

- `primaryMetric` : score de santé (0-100 calculé par toi)
- `summary` : *"SSD 512 Go, état Healthy, 2 ans en service"*

Calcul du score

Inspire-toi : `100 - (wear%) - (errors > 100 ? 20 : 0)`. À toi de doser.

Lien éco

C'est **le module le plus directement écoresponsable**. Repérer un disque en fin de vie permet de le remplacer **avant** que la machine devienne inutilisable. C'est concret : tu peux mettre une **alerte rouge** sur le dashboard si un disque a `Wear > 80 %`.

Affichage côté PHP

Dans `public/pc.php`, ajoute juste avant le footer :

```
<?php if (!empty($rapport['module']['name'])):
        $modName = $rapport['module']['name'];
        $modFile = __DIR__ . "/modules/$modName.php";
        if (is_file($modFile)) {
            require $modFile;
        }
?>
    <?php endif; ?>
```

Et tu crées `public/modules/network.php` (et un pour chaque module). Chaque fichier reçoit `$rapport['module']['data']` et l'affiche à sa façon.

> 💡 **POURQUOI UN FICHIER PAR MODULE ?** C'est le pattern « plugin ». Chaque élève maintient **son fichier**, sans toucher au reste. Pas de conflit Git, pas de cassage mutuel.

🤖 Travailler avec l'IA — pour démarrer ton module

Très bon prompt :

> Je travaille sur le module **"USB"** d'un inventaire informatique en PHP/ PowerShell. Voici les commandes PowerShell qui me donnent les périphériques USB connectés : [colle le bloc]. J'aimerais 3 propositions de visualisation HTML (Bootstrap 5, dark mode) pour la fiche PC, qui mettent en évidence le **nombre par classe** et les périphériques **inconnus**. Pas de code, juste les idées + ce qu'elles racontent à l'utilisateur.

Mauvais prompt :

> Fais-moi le module USB en entier.

Comprends pourquoi : avec le mauvais prompt, **ton voisin pourrait livrer exactement le même rendu que toi**. Avec le bon, tu fais un choix de design qui te distingue.

Tronc commun + modules : ce qu'apporte la classe entière

Une fois tout intégré, le dashboard montre **pour chaque PC** :

- les KPI tronc commun (RAM, disque, OS),
- le `summary` du module spécifique,
- une icône qui indique quel module est rattaché.

Et un élève peut **comparer** son PC à celui de toute la classe. C'est ça, un inventaire de parc.

✅ Critères de réussite de l'étape 5

- [ ] Le JSON envoyé contient bien `module.name`, `module.summary`, `module.stats`, `module.data`.
- [ ] La fiche PC affiche **automatiquement** la bonne section pour ton module.
- [ ] Le dashboard affiche le `summary` du module pour chaque PC.
- [ ] Le code de ton module est dans **un seul fichier** PHP indépendant.
- [ ] Tu peux **expliquer en 30 secondes** ce que ton module apporte au parc.

⚠️ Pièges fréquents

Piège	Solution
`Get-ScheduledTask` lent ou bloqué par l'antivirus	Limiter avec `-TaskPath '\'`

Piège	Solution
`Get-LocalGroupMember` plante avec un message bizarre	`-ErrorAction SilentlyContinue`
Tu casses la fiche PC pour les autres élèves	Module **dans un fichier séparé** + `is_file` avant `require`
Données sensibles dans le JSON (mot de passe, token)	**Ne jamais** envoyer ces données — vérifier avant l'envoi

Pour la suite

Dernière étape : **sécuriser l'API**. Pourquoi ? Parce qu'aujourd'hui, n'importe qui sur le réseau de l'école peut envoyer des rapports **au nom de n'importe quel PC**. On verra le token, et **pourquoi** on en a besoin.