Étape 5 — Ton module spécifique

5TQ

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

Voici les 16 modules disponibles — ton prof t'en attribue 2.

Modules de base (tronc commun de la classe)

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 ?

Modules complémentaires

N°	Module	Slug	Question à éclairer
9	Performance temps réel	`performance`	Mon PC est-il sous-utilisé ou en goulot ?
10	Pare-feu Windows	`firewall`	Quelles règles laissent passer le trafic ?
11	Partages SMB	`shares`	Mon PC partage-t-il quelque chose ?
12	Batterie & alim.	`power`	En quelle santé est ma batterie ?
13	GPU & écrans	`display`	Quel matériel d'affichage est branché ?
14	Logs d'événements	`events`	Qu'est-ce qui plante sur mon PC ?
15	Chiffrement BitLocker	`encryption`	Mes données sont-elles protégées ?
16	Réseaux Wi-Fi	`wifi`	Où mon portable a-t-il déjà été connecté ?

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

```
$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

$admins = Get-LocalGroupMember -Group 'Administrators' -ErrorAction SilentlI
    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

```
$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.

Modules complémentaires

Les modules suivants (9 à 16) suivent **exactement la même structure** que les huit premiers : enveloppe `module` identique, fichier PHP séparé, affichage automatique sur la fiche PC. Ils explorent simplement d'autres facettes du système Windows.

Module 9 — Performance & charge système (`performance`)

Le tronc commun te donne déjà les **processus** les plus gourmands. Ce module regarde la **machine dans son ensemble** : combien tourne-t-elle à vide, combien depuis quand, et a-t-elle assez de mémoire pour bien vivre ?

Commandes

```powershell
$cpu = Get-CimInstance Win32_PerfFormattedData_PerfOS_Processor |
    Where-Object Name -eq '_Total' |
    Select-Object @{N='CpuPct'; E={[int]$_.PercentProcessorTime}}

$os  = Get-CimInstance Win32_OperatingSystem
$ram = @{
    UsedGo = [math]::Round(($os.TotalVisibleMemorySize - $os.FreePhysicalMe
    FreeGo = [math]::Round($os.FreePhysicalMemory/1MB, 2)
    UsedPct = [int](($os.TotalVisibleMemorySize - $os.FreePhysicalMemory)
}

$uptime = (Get-Date) - $os.LastBootUpTime  # objet TimeSpan
$pageFile = Get-CimInstance Win32_PageFileUsage |
    Select-Object Name, AllocatedBaseSize, CurrentUsage
```

Stats

- `primaryMetric` : `cpuPct` (entier 0-100)
- `primaryLabel` : "% CPU instantané"
- `summary` : *"Charge CPU 23 %, RAM 58 %, uptime 4 j 8 h"*

Affichage PHP suggéré

Trois jauges côte à côte (CPU, RAM, page file), un compteur d'uptime formaté humainement, et un drapeau "sous-utilisé / équilibré / saturé" calculé sur la moyenne des dernières 24 h.

Pièges

- La valeur de `PercentProcessorTime` est une **photo instantanée**. Pour un vrai diagnostic, il faudrait moyenner sur plusieurs minutes. Mentionne cette limite dans le summary.
- `Win32_PageFileUsage` peut renvoyer **vide** si Windows gère le fichier d'échange automatiquement. Prévois ce cas.

Module 10 — Pare-feu Windows (`firewall`)

Différent du module 2 (Defender = antivirus) et du module 7 (connexions actuelles) : ici, on regarde **les règles** qui filtrent le trafic, pas ce qui passe en ce moment.

Commandes

```powershell
$profiles = Get-NetFirewallProfile |
    Select-Object Name, Enabled, DefaultInboundAction, DefaultOutboundActio

# IMPORTANT : Get-NetFirewallRule sans filtre est TRÈS lent (~30 s).
# On filtre dès l'appel.
$rulesIn = Get-NetFirewallRule -Direction Inbound -Action Allow -Enabled Tr
    Select-Object -First 30 DisplayName, Profile, Owner

$counts = @{
    InAllow  = (Get-NetFirewallRule -Direction Inbound  -Action Allow -Enab
    InBlock  = (Get-NetFirewallRule -Direction Inbound  -Action Block -Enab
    OutAllow = (Get-NetFirewallRule -Direction Outbound -Action Allow -Enab
}
```

Stats

- `primaryMetric` : nombre de règles entrantes "Allow" actives
- `primaryLabel` : "règles entrantes Allow"
- `summary` : *"3 profils actifs, 47 règles entrantes en allow, 1 050 en block"*

Angle pédagogique

Pourquoi le profil **Public** bloque-t-il plus que **Domain** ? Quelles règles ont été ajoutées **manuellement** (filtrer sur `Owner -eq $null` ou DisplayName personnalisé) ? Un PC avec 300+ règles entrantes en allow sur le profil Public est potentiellement exposé.

Pièges

- `Get-NetFirewallRule` **sans filtre** peut prendre 30 secondes. Toujours filtrer dès l'appel (`-Direction`, `-Enabled True`, `-First N`).
- Sur certaines versions de Windows, le module nécessite l'élévation pour voir toutes les règles. Capter l'erreur, remonter "partiel" dans le summary si applicable.

Module 11 — Partages SMB & mappings (`shares`)

```powershell
# Partages publiés par mon PC (en excluant les partages système C$, ADMIN$...)
$shares = Get-SmbShare |
    Where-Object Special -eq $false |
    Select-Object Name, Path, Description, EncryptData

# Lecteurs réseau que MON PC a montés
$mappings = Get-SmbMapping |
    Select-Object LocalPath, RemotePath, Status

# Connexions SMB sortantes actives (vers quels serveurs je parle ?)
$connections = Get-SmbConnection |
    Select-Object ServerName, ShareName, UserName, Dialect, Encrypted
```

Stats

- `primaryMetric` : nombre de partages locaux publics (hors système)
- `primaryLabel` : "partages publiés"
- `summary` : *"2 partages locaux, 3 lecteurs mappés vers serveur-prof"*

Angle pédagogique

Un partage local **non-protégé** est une fuite potentielle. Quels lecteurs sont mappés vers où ? Bon sujet de discussion classe : pourquoi Windows expose-t-il par défaut des partages cachés `C$`, `ADMIN$` ? Comment les désactiver proprement ?

Affichage PHP suggéré

Deux listes côte à côte : "Mon PC partage" (sortant) et "Mon PC se connecte à" (entrant). Mettre un drapeau orange sur les connexions **non chiffrées** (`Encrypted -eq $false`).

Module 12 — Batterie & alimentation (`power`)

⚠️ **Module applicable uniquement aux ordinateurs portables.** Sur un PC fixe, ton script doit détecter l'absence de batterie et remonter un `module.data.notApplicable = true`. C'est une bonne occasion d'apprendre à gérer les cas conditionnels.

Commandes

```powershell
$bat = Get-CimInstance Win32_Battery -ErrorAction SilentlyContinue

if ($bat) {
    $health = @{
        Charge       = $bat.EstimatedChargeRemaining        # %
        Status       = $bat.BatteryStatus                   # 1=Discharging
        DesignCap    = $bat.DesignCapacity
        FullChargeCap= $bat.FullChargeCapacity
        HealthPct    = if ($bat.DesignCapacity -gt 0) {
            [math]::Round(($bat.FullChargeCapacity / $bat.DesignCapacity) *
        } else { $null }
    }
}

$plan = (powercfg /getactivescheme) -match 'Power Scheme GUID' |
    ForEach-Object { ($_ -split ':')[1].Trim() }
```

Stats

- `primaryMetric` : pourcentage de santé batterie (FullChargeCapacity / DesignCapacity)
- `primaryLabel` : "% santé batterie"
- `summary` : *"Batterie 78 % de santé, charge 64 %, mode 'Équilibré'"*

Affichage PHP suggéré

Une jauge ronde de la **santé** (différente de la charge !) avec code couleur : vert > 80 %, orange 60-80 %, rouge < 60 %. Une seconde jauge plus petite pour la charge instantanée. Badge du plan d'alimentation actif.

Pièges

- Sur un fixe, `Get-CimInstance Win32_Battery` retourne `$null`. **Tester avec** `if ($bat)` avant de lire les propriétés.
- `BatteryStatus` est un **code numérique** (1-11). Documenter une table de correspondance dans le code PowerShell ou côté PHP.

Module 13 — GPU & écrans (`display`)

```powershell
$gpu = Get-CimInstance Win32_VideoController |
    Select-Object Name, AdapterRAM, DriverVersion, VideoModeDescription,
```

```
                    CurrentRefreshRate,
                    CurrentHorizontalResolution, CurrentVerticalResolution

# Les vrais détails écran sont dans une classe WMI à part
$monitors = Get-CimInstance WmiMonitorID -Namespace root\wmi -ErrorAction S
    ForEach-Object {
        [PSCustomObject]@{
            Manufacturer = -join ($_.ManufacturerName | Where-Object {$_ -n
            Model        = -join ($_.UserFriendlyName | Where-Object {$_ -n
            YearOfMfg    = $_.YearOfManufacture
        }
    }
```

Stats

- `primaryMetric` : nombre d'écrans branchés
- `primaryLabel` : "écrans détectés"
- `summary` : *"GPU NVIDIA T1000, 2 écrans (Dell 2021, BenQ 2019)"*

Pièges

- `WmiMonitorID` peut échouer sur certaines machines (surtout les écrans connectés via USB ou DisplayLink). Toujours `-ErrorAction SilentlyContinue`.
- Les noms d'écran sont stockés en **tableaux d'octets** (ASCII). La conversion ci-dessus filtre les zéros (terminateurs) avant le `-join`.

Module 14 — Logs d'événements système (`events`)

```
# Derniers événements critiques / erreurs du journal Système (7 derniers jo
$errors = Get-WinEvent -FilterHashtable @{
    LogName   = 'System'
    Level     = 1,2   # 1 = Critical, 2 = Error
    StartTime = (Get-Date).AddDays(-7)
} -MaxEvents 30 -ErrorAction SilentlyContinue |
    Select-Object TimeCreated, ProviderName, Id, LevelDisplayName,
        @{N='Message'; E={
            $_.Message.Substring(0, [math]::Min(120, $_.Message.Length))
```

```
        }}

# Compter les démarrages des 30 derniers jours (ID 6005 = "Event log starte
$boots = (Get-WinEvent -FilterHashtable @{
    LogName = 'System'; Id = 6005; StartTime = (Get-Date).AddDays(-30)
} -ErrorAction SilentlyContinue | Measure-Object).Count
```

Stats

- `primaryMetric` : nombre d'erreurs critiques sur 7 jours
- `primaryLabel` : "erreurs (7j)"
- `summary` : *"4 erreurs critiques cette semaine, 12 redémarrages ce mois"*

Angle pédagogique

Un PC qui redémarre **30 fois en un mois** a un problème (matériel ? mise à jour ratée ?). Bon sujet pour parler des **niveaux de log** (Information / Warning / Error / Critical) et de la différence entre les journaux **System**, **Application** et **Security**.

Affichage PHP suggéré

Tableau chronologique des 10 derniers événements critiques, avec une icône par niveau, l'ID, et un extrait du message (tronqué). Un compteur d'erreurs par jour sur les 7 derniers jours en mini-graphe.

Pièges

- Le journal **Security** demande des droits administrateur. **Reste sur System** pour rester accessible à un utilisateur standard.
- Les messages d'erreur peuvent être **très longs**. Toujours tronquer avant l'envoi pour ne pas alourdir le JSON.
- `-FilterHashtable` est **beaucoup plus rapide** que `Where-Object` après-coup. Toujours filtrer à la source.

Module 15 — Chiffrement BitLocker (`encryption`)

```
# Get-BitLockerVolume peut nécessiter l'élévation selon les volumes
$bl = Get-BitLockerVolume -ErrorAction SilentlyContinue |
    Select-Object MountPoint, VolumeStatus, ProtectionStatus,
                  EncryptionMethod, EncryptionPercentage,
```

```
                    @{N='HasKeyProtector'; E={ $_.KeyProtector.Count -gt 0 }}

# Si pas de BitLocker disponible, le tableau est vide
$summary = if ($bl) {
    $chiffres = ($bl | Where-Object ProtectionStatus -eq 'On').Count
    $total = $bl.Count
    "$chiffres/$total volumes chiffrés"
} else {
    "BitLocker non disponible ou droits insuffisants"
}
```

Stats

- `primaryMetric` : pourcentage de volumes chiffrés
- `primaryLabel` : "% volumes chiffrés"
- `summary` : *"C: chiffré (XtsAes128), D: non chiffré"*

Affichage PHP suggéré

Pour chaque volume, un cadenas vert (chiffré) ou rouge (en clair), la méthode de chiffrement, et la barre de progression du chiffrement en cours (`EncryptionPercentage`).

Pièges

- Sur **Windows Home**, BitLocker n'est pas disponible — le cmdlet n'existe pas. **Toujours** `-ErrorAction SilentlyContinue` et gérer le cas "non applicable" comme dans le module 12.
- Sur les éditions Pro, certains volumes ne sont visibles qu'avec une élévation. Documenter dans le summary si la lecture est partielle.

Module 16 — Réseaux Wi-Fi enregistrés (`wifi`)

⚠️ **Important — RGPD / vie privée** : on collecte la **liste** des réseaux mémorisés, **pas les mots de passe**. C'est une excellente occasion de discuter en classe de la différence entre **données techniques** et **données personnelles**, et de **ce qu'on collecte parce qu'on peut** vs **ce qu'on a le droit de collecter**.

Commandes

```powershell
# Profils Wi-Fi mémorisés (sans clés)
$rawProfiles = netsh wlan show profiles
$profiles = $rawProfiles | Select-String 'Profil' | ForEach-Object {
    ($_ -split ':')[1].Trim()
} | Where-Object { $_ }

# Réseau actuellement connecté
$current = Get-NetConnectionProfile |
    Where-Object IPv4Connectivity -eq 'Internet' |
    Select-Object Name, InterfaceAlias, NetworkCategory

# Pour chaque profil mémorisé, type d'authentification (sans la clé)
$details = $profiles | ForEach-Object {
    $p = $_
    $info = netsh wlan show profile name="$p" key=clear  # 'clear' demande
    # ...mais on ne la lit volontairement PAS — on prend juste l'authentif
    [PSCustomObject]@{
        Name = $p
        Auth = ($info | Select-String 'Authentification' | Select-Object -F
    }
}
```

Stats

- `primaryMetric` : nombre de profils Wi-Fi enregistrés
- `primaryLabel` : "réseaux mémorisés"
- `summary` : *"12 réseaux mémorisés, connecté à 'CEPES-Eleves' (WPA2)"*

Angle pédagogique

Un portable avec **50 réseaux mémorisés** (cafés, gares, hôtels, écoles…) raconte **où il est passé**. Discussion sécurité : les profils ouverts ou en **WEP** représentent des risques (man-in-the-middle). Discussion RGPD : qui a le droit de voir cette liste ? Le service IT de l'école ? Un employeur ?

Affichage PHP suggéré

Liste des réseaux mémorisés avec un badge couleur par type d'authentification (vert = WPA3/WPA2, orange = WEP, rouge = ouvert). Mise en évidence du réseau actuellement connecté.

Pièges

- Le parsing de `netsh wlan show profiles` est **localisé** : le mot "Profil" en français devient "Profile" en anglais. À détecter ou à documenter pour la classe.

- **Ne jamais lire les clés**, même si l'option `key=clear` existe. À expliquer aux élèves : *"on peut, donc on doit se demander si on doit"*.

Affichage côté PHP

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

```php
<?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 '\'`
`Get-LocalGroupMember` plante avec un message bizarre	`-ErrorAction SilentlyContinue`
`Get-NetFirewallRule` sans filtre prend 30 s	Toujours filtrer dès l'appel (`-Direction`, `-Enabled True`, `-First N`)
`Win32_Battery` est null sur un PC fixe	Tester `if ($bat)` avant de lire ses propriétés ; remonter `notApplicable`
`Get-BitLockerVolume` n'existe pas sur Windows Home	`-ErrorAction SilentlyContinue` et gérer l'absence
Le parsing de `netsh` casse selon la langue de Windows	Documenter pour la classe, ou détecter la locale avant
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, clé Wi-Fi, token)	**Ne jamais** envoyer — vérifier avant l'envoi ; principe RGPD

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.

© 2025 Mr. Lambrechts https://v2.ttrinfo.be 18 | 18