Victor CHENG
07/01/2026

# 📘 Documentation Technique : CVE Manager Dashboard

## 1. Architecture du Projet (Version Simplifiée)

Le projet utilise une structure "à plat" pour maximiser la compatibilité entre les scripts et la base de données SQLite.

### 📂 Organisation des fichiers

- **`app.py`** : Serveur principal FastAPI qui expose l'API pour le dashboard.
- **`cve.db`** : Base de données SQLite stockant les équipements et les vulnérabilités.
- **`cve_fetcher.py`** : Script de synchronisation avec l'API du NIST (NVD).
- **`import_inventory.py`** : Script d'intégration initiale depuis le fichier JSON.
- **`inventory.json`** : Source de données brute contenant la liste du parc informatique.
- **`static/`** : Dossier contenant l'interface web (HTML/CSS/JS).
- **`screenshots/`** : Captures d'écran pour la démonstration du projet.

## 2. Intégration et Consommation des API

Le système repose sur un flux de données à deux niveaux pour garantir l'actualité des informations.

### A. L'API Source (NVD NIST)

Le script cve_fetcher.py interroge l'API externe du NIST pour récupérer les vulnérabilités mondiales.

- Authentification : Utilisation d'une clé API transmise via variable d'environnement pour augmenter les quotas de requêtes.
- Filtrage : Les requêtes sont ciblées par CPE (Common Platform Enumeration) pour ne télécharger que ce qui est pertinent pour l'inventaire.

### B. Votre API Locale (FastAPI)

Le fichier app.py crée une interface de programmation locale.

- Rôle : Elle sert de passerelle entre la base de données cve.db et le navigateur.
- Format : Les données sont exposées en JSON, permettant au Frontend de mettre à jour les compteurs "Total CVE" et "Équipements" en temps réel.

# 3. Guide d'Installation et Déploiement

## Prérequis

- Python 3.12.
- Environnement virtuel Python (`venv`) activé.

## Étapes de mise en service

**Installation des dépendances** :
Bash
pip install fastapi uvicorn requests

1. **Initialisation de la base de données** (Si `cve.db` est absente : Bash) :

   “python3 import_inventory.py”

2. Cette commande crée la table `equipments` et importe les 22 éléments de `inventory.json`.

   **Récupération des CVE** (Bash) :
   “python3 cve_fetcher.py”
   => Remplit la table `cves` en interrogeant le NIST pour chaque équipement.

3. **Lancement du Dashboard** (Bash) :
   “uvicorn app:app --reload ou python3 app.py”

# 4. Fonctionnement Technique

- **Backend (FastAPI)** : Gère les routes API pour fournir les statistiques en temps réel (Total CVE, distribution de sévérité).
- **Base de données (SQLite)** : Utilise des relations simples entre les équipements (via leur chaîne CPE) et les vulnérabilités répertoriées.
- **Frontend (Vanilla JS)** : Utilise `fetch()` pour récupérer les données et Chart.js pour la visualisation graphique.

# 5. Gestion des Erreurs et Résolution (Rétrospective)

L'architecture actuelle a été choisie pour corriger et prévenir les erreurs critiques rencontrées lors du développement. En gardant les scripts et les données dans le même

répertoire (`/cve_manager`), le chemin relatif reste constant (`./`), ce qui facilite le déploiement sur différentes machines.

## 🛠️ Récapitulatif des erreurs résolues :

- **Gestion des Chemins (Path Errors)** :
  - **`ModuleNotFoundError`** : Résolu en ramenant les scripts à la racine, permettant à `app.py` d'importer les modules sans manipulation complexe du `sys.path`.
  - **`FileNotFoundError`** : L'unification du répertoire racine garantit que les scripts trouvent systématiquement `inventory.json` et `cve.db`.

- **Intégrité de la Base de Données** :
  - **Bases "Fantômes"** : Correction du problème où des bases de données vides étaient créées dans des sous-dossiers. Désormais, une base unique `cve.db` est partagée par tous les composants.
  - **Reconstruction Post-Suppression** : Mise en place d'un workflow robuste permettant de régénérer intégralement les données en deux commandes (`import_inventory.py` puis `cve_fetcher.py`) en cas de corruption de la base.

- **Interface et Affichage (UI/UX)** :
  - **Conflits CSS** : Nettoyage des doublons de classes dans `style.css` pour assurer l'alignement horizontal des compteurs statistiques.
  - **Synchronisation API** : Ajustement des fonctions `fetch()` en JavaScript pour pointer vers les bons points d'entrée (endpoints) de l'API locale fournie par FastAPI.