Rapport Projet python Elias Baroudi & Nassim Ghlami

Table des matières

[1.Spécifications 3](#_Toc186902588)

[2. L’analyse 3](#_Toc186902589)

[2.1#L’environnement de travail et justification 3](#_Toc186902590)

[2.2 Outils et Bibliothèques 3](#_Toc186902591)

[2.3 les données dans les spécifications 4](#_Toc186902592)

[2.3 Diagramme des classes 5](#_Toc186902593)

[2.3.1 Classe CVE 5](#_Toc186902594)

[2.3.2 Classe KevinCVE 5](#_Toc186902595)

[2.3.3 Classe NSTCVE 6](#_Toc186902596)

[2.3.4 La classe Corpus 6](#_Toc186902597)

[2.3.5 La classe SearchingEngine 7](#_Toc186902598)

[2.3.6 Relations d’héritage 7](#_Toc186902599)

[2.3.7 Relations d’utilisations 7](#_Toc186902600)

[3. La conception 8](#_Toc186902601)

[3.1 partages des tâches 8](#_Toc186902602)

[3.2 Détail Algorithme 8](#_Toc186902603)

[3.3 Problèmes rencontrés 8](#_Toc186902604)

[3.4 Exemple commenté d’utilisation du programme 8](#_Toc186902605)

[4. La validation 8](#_Toc186902606)

[4.1 Test unitaires 9](#_Toc186902607)

[4.2 Test Globaux 9](#_Toc186902608)

[5. La maintenance 9](#_Toc186902609)

# 1.Spécifications

<https://github.com/EliasBaroudi/search-engine-python>

Le programme a pour objectif principal de fournir un moteur de recherche liée à la cybersécurité pour recherche des vulnérabilités connues, les CVE (Common Vulnerabilities and Exposures) et montrer des articles scientifiques associés.

Les fonctionnalités principales, telles que définies par les besoins identifiés sont la recherche des CVE à partir de mots-clés, les CVE retournées doivent inclure des informations détaillées telles que leur ID, leur description, les notes associées et leur source d’origine. Ensuite l’affichage des articles scientifiques, lorsque les articles scientifiques liés aux CVE sont disponibles on doit les affichés en parallèle des résultats de recherche et chaque article doit inclure un titre et un lien permettant de pouvoir accéder à l’article. De plus, les résultats doivent être classés selon leur pertinence par rapport à la demande de l’utilisateur et l’algorithme TF-IDF est utilisé pour cela. Puis l’utilisateur doit pouvoir filtrer les résultats selon les sources des CVE qui sont soit Kevin ou NST et choisir s’il veut uniquement les CVE avec articles scientifiques ou pas. Enfin un bon interface utilisateur, l’application propose une interface web simple et claire permettant d’effectuer une recherche et afficher le résultat de façon lisible et choisir le nombre de résultat.

# 2. L’analyse

## 2.1#L’environnement de travail et justification

## 2.2 Outils et Bibliothèques

Pour ce projet nous avons utilisé plusieurs Outils et bibliothèques telles que :

* Le Framework Dash car elle nous fournit une interface web interactive
* Les bibliothèques NumPy et Pandas car elles nous permettent la manipulation des matrices et l’analyse des données
* Pytest pour effectuer les tests unitaires
* Flask pour avoir une base solide pour l’exécution de l’application Dash
* API Kevin pour avoir les CVE les plus récentes
* API NST pour compléter

## 2.3 les données dans les spécifications

Tout d’abord les données au cœur de l’application et qui alimentent le moteur de recherche sont les données des CVE qui sont :

* Le CVE ID
* La date d’ajout
* Source
* Description
* Notes associées
* Produits impactés

Ensuite, nous avons les données sur les articles scientifiques qui sont :

* Titre
* L’URL permettant d’accéder à l’article
* Résumé
* Source des articles

De plus, nous avons le corpus qui est la structure qui organise toutes les données pour faire une bonne recherche. Il joue un rôle clé dans les calculs TF-IDF. Il est composé de :

* La liste des CVE
* Des Mots-clés
* Description complète
* Nombre de documents

Enfin, nous avons les requêtes utilisateur qui sont ces données qui sont saisies ou sélectionnés par l’utilisateur pour faciliter sa recherche :

* Les mots-clés
* Les filtres (Source, nombre de résultats articles associés)

## 2.3 Diagramme des classes

Une image contenant texte, cercle, diagramme, Police

Description générée automatiquement

### 2.3.1 Classe CVE

Cette classe à plusieurs attributs principaux tels que :

* cveID
* description
* dateAdded
* notes
* source

Et ça méthode principale est getType() qui retourne le type de CVE.

### 2.3.2 Classe KevinCVE

C’est une sous classe de CVE, elle représente les CVE qui viennent de l’API Kevin.

Attributs :

* nvdData
* product
* vulnerabilityName

En méthode principale elle utilise getType() qui retourne « KevinCVE ».

### 2.3.3 Classe NSTCVE

C’est une sous classe de CVE aussi, elle représente les CVE issues de L’API NST.

Elle a comme Attribut :

* shortDescription()

Et pour méthode principale un getType() qui retourne « NSTCVE ».

### 2.3.4 La classe Corpus

Cette classe gère un ensemble de CVE et les organise comme il faut pour permettre des recherches efficaces. Elle a plusieurs attributs principaux :

* nom
* cve
* ndoc
* desc
* vulnerabilities

En méthodes principales il y a add(cve) qui ajoute un CVE au corpus, getNDoc() qui retourne le nombre de documents (CVE) dans le corpus et getCVE() qui retourne la liste complète des CVE.

### 2.3.5 La classe SearchingEngine

Cette classe cherche à implémenter le moteur de recherche basé sur le corpus.

Elle a plusieurs attributs principaux qui sont :

* corpus
* vocabulaire
* mat\_TF
* mat\_TFIDF

Elle a plusieurs méthodes principales telles que :

* getTf() qui calcule la matrice TF pour le corpus.
* getTFIDF() qui calcule la matrice TF-IDF pour le corpus
* search(query,num\_results,filters,has\_articles) qui effectue une recherche basée sur une requêtes et qui va retourner les résultats les plus pertinents.

### 2.3.6 Relations d’héritage

Les classes KevinCVE et NSTCVE sont des classes héritées de la classe CVE car la classe CVE a des attributs commun comme cveID, description et il y a forcément les méthodes principales partagées comme getTYpe(). KevincCVE et NSTCVE développe CVE en ajoutant leurs données spécifiques avec leur API.

### 2.3.7 Relations d’utilisations

La classe corpus gère les collections d’objets CVE, elle agit comme une base structurée pour stocker et organiser les données.

La classe SerchingEngine elle utilise les données du Corpus pour créer des matrices TF et TF-IDF qui permettent d’avoir des recherches pertinentes et elle est chargée aussi d’effectuer les recherches en fonction des requêtes utilisateur.

Enfin, l’interface utilisateur qui est définie dans le fichier interface.py interagit avec SearchingEngine. Elle envoie les requêtes de recherche, récupère les résultats et les affiche.

# 3. La conception

## 3.1 partages des tâches

Le sujet nous donnait déjà une indication du partage des tâches avec les v1, v2 et v3.

Elias a travaillé sur les versions 1 et 2 qui correspond aux TDs 3 à 7 qui sont le socle de base de l’application et le moteur de recherche. Il a eu ce rôle un peu de lead developer.

Nassim à travaillé sur la v3 qui correspond à l’interface et l’extension et a fait le rapport.

Il y a eu beaucoup d’échange WhatsApp sur l’avancement des tâches et des commit GitHub.

## 3.2 Détail Algorithme

############

## 3.3 Problèmes rencontrés

Le premier problème rencontré à était de choisir vers quoi on allait s’orienter, bien qu’on ait les TD travaillé en cours cela n’était pas suffisant et fallait pas juste recopier. On l’a résolu car on s’est dit qu’il fallait faire un projet qui nous sera utile et comme tous les deux on se dirige vers la cybersécurité autant faire un projet lié à ce domaine.

Le second problème était de trouver les API car certaines étaient payantes et d’autres trop compliqués à utiliser donc on en a essayé plusieurs avant d’arriver à Kevin et NST.

## 3.4 Exemple commenté d’utilisation du programme

#######

# 4. La validation

## 4.1 Test unitaires

## 4.2 Test Globaux

# 5. La maintenance

On pourrait en faire un support multilingue qui permettrait de rechercher des CVE ou des articles dans différentes langues. Pour cela, on utiliserait des bibliothèques comme Google Translate API pour traduire les requêtes. De plus, il faudrait augmenter le corpus avec les descriptions traduites des CVE.

Cela serait de difficulté moyenne car les bibliothèques de traduction sont bien intégrées à Python mais faut gérer les erreurs de traduction surtout sur des termes techniques qui là serait bien plus compliqué.

On pourrait aussi permettre aux utilisateurs de télécharger les résultats sous forme de fichiers CSV ou PDF avec la bibliothèque pandas pour exporter les données en CSV et FPDF pour créer des fichiers PDF. Cela serait facile car les bibliothèques nécessaires sont bien connues et utilisées.