Projet de fin d’année

HUGO MEURIEL &
BRUEL MATHIS

Données non structurées

Pour cette partie, nous avons pris 3 fichiers de logs. Le premier vient d’un serveur apache2, le second d’un projet web pour l’association Endorah, et le dernier d’un serveur Minecraft.

Pour le premier script, nous avons utilisé les logs de notre projet web Endorah. Nous avons essayé de traiter toutes les données utiles disponibles dans ce fichier de logs (Tout ce qui est initialisé, Toutes les pages charger ainsi que tous les contenus qui sont chargés (Fichier de style, fichiers de script, images, …) Pour se faire nous avons fait juste du traitement de données grâces aux chaînes de caractères.

```
---------- Statistiques des données ----------

Initialisations:
Nombre total d'initialisations: 15
Composants initialisés:
- Administrator
- Administrators
- Images
- Reseaux
- Member
- Members
- Partners
- Project
- Tag
- Category
- Projects
- Services
- Pages
- Navbars
- Portfolios

Pages:
Nombre total de pages/requêtes: 7

Codes de statut HTTP:
Code "GET: 6 requêtes
```

```
Code "POST: 1 requêtes

Types de ressources chargées:
png: 5 fichiers
webp: 1 fichiers
jpg: 1 fichiers
```

Pour le second script, nous avons utilisé les logs du serveur Minecraft plus complexe à traiter car beaucoup plus d'informations. Nous avons récupéré toutes les données utiles/possibles (erreurs, plugins load, map load, les joueurs qui rejoint/quitte, les commandes exécuter par qui, quand, …) Pour ce faire, nous avons essayé d'utiliser le plus possible des expressions régulières. Nous avons donc utilisé le REGEX.

```
=== Statistiques des erreurs ===
Nombre total d'erreurs: 1

Erreurs par plugin:
- plugins/MetaBrushesAxiom-1.1-SNAPSHOT.jar: 1 erreurs

=== Statistiques des plugins ===
Nombre de plugins chargés: 29

=== Statistiques des mondes ===
Nombre de mondes chargés: 42

Liste des mondes:
- [04:00:51] - world
- [04:00:52] - world_nether
- [04:00:52] - world_the_end
- [04:01:18] - grrrrrr
- [04:01:21] - STG_CivitasAugusta_6
- [04:01:21] - Schematics
- [04:01:21] - STG_CivitasAugusta_12
- [04:01:22] - STG_CivitasAugusta_8
- [04:01:22] - AssetsPlants
- [04:01:22] - Citypack
- [04:01:22] - AssetsBundle
- [04:01:22] - Elfique_pv
- [04:01:22] - tree2
- [04:01:22] - lothlorien
- [04:01:23] - Pantheon
- [04:01:23] - housepack
- [04:01:23] - elora
- [04:01:23] - arbres
- [04:01:23] - All_Builds
```

```
- [04:01:23] - STG_LostTemple_4
- [04:01:23] - Elven_Treehouse
- [04:01:23] - projetxz-v5
- [04:01:24] - projetxz-v4
- [04:01:24] - spoil
- [04:01:24] - mushroom
- [04:01:24] - flowers
- [04:01:24] - Arene
- [04:01:24] - Fantasy_Castle
- [04:01:25] - stg
- [04:01:25] - orcs
- [04:01:25] - tree
- [04:01:26] - STG_LostTemple_6
- [04:01:26] - mine1-terra
- [04:01:26] - STG_LostTemple_8
- [04:01:26] - STG_LostTemple_12
- [04:01:27] - Elven_Citadel
- [04:01:27] - ElvenCity-Lemuria
- [04:01:27] - spikes
- [04:01:28] - USW
- [04:01:28] - temple
- [04:01:28] - STG_CivitasAugusta_4
- [04:01:28] - TestFlatWorld

=== Statistiques des joueurs ===
Nombre de connexions/déconnexions: 5
Nombre de commandes exécutées: 15
```

Données semi structurées

Les formats semi-structurés sont des formats de données qui, contrairement aux données structurées (comme les bases de données relationnelles), ne suivent pas un schéma strict et prédéfini. Ils offrent plus de flexibilité tout en maintenant une certaine organisation.

Formats étudiés

1. JSON (JavaScript Object Notation)

- **Origine** : Créé en 2001 par Douglas Crockford
- **Historique** : Initialement développé pour le langage JavaScript, il est devenu un standard indépendant
- **Caractéristiques** :
    - Format texte lisible par l'homme
    - Structure hiérarchique basée sur des paires clé-valeur
    - Support natif dans la plupart des langages de programmation
    - Très populaire pour les API REST
- **Avantages** :
    - Léger et compact
    - Facile à parser
    - Excellent support dans les langages modernes
- **Inconvénients** :
    - Pas de support des commentaires
    - Pas de support des références cycliques

2. CSV (Comma-Separated Values)

- **Origine** : Format très ancien, utilisé depuis les débuts de l'informatique
- **Historique** : Standardisé en 2005 (RFC 4180)
- **Caractéristiques** :
    - Format tabulaire simple
    - Séparation par délimiteurs (virgule, point-virgule, tabulation)
    - Première ligne souvent utilisée pour les en-têtes
- **Avantages** :
    - Très simple à lire et à écrire
    - Parfait pour les données tabulaires
    - Compatible avec Excel et autres outils bureautiques
- **Inconvénients** :
    - Pas de support des structures complexes
    - Problèmes avec les caractères spéciaux et les délimiteurs dans les données

3. YAML (YAML Ain't Markup Language)

- **Origine** : Créé en 2001 par Clark Evans, Ingy döt Net, et Oren Ben-Kiki
- **Historique** : Développé pour être plus lisible que XML
- **Caractéristiques** :
    - Format basé sur l'indentation
    - Support des commentaires

    - Support des références
- **Avantages** :
    - Très lisible par l'homme
    - Support des structures complexes
    - Idéal pour les fichiers de configuration
- **Inconvénients** :
    - Sensible à l'indentation
    - Peut être ambigu dans certains cas
    - Parsing plus complexe que JSON

4. XML (EXTENSIBLE MARKUP LANGUAGE)

- **Origine** : Créé en 1996 par le W3C
- **Historique** : Successeur du SGML, standardisé en 1998
- **Caractéristiques** :
    - Format basé sur des balises
    - Support des schémas (DTD, XSD)
    - Structure hiérarchique stricte
- **Avantages** :
    - Très robuste et mature
    - Excellent support des métadonnées
    - Validation possible via schémas
- **Inconvénients** :
    - Verbose et lourd
    - Complexe à parser
    - Moins populaire pour les API modernes

COMPARAISON GÉNÉRALE

UTILISATION TYPIQUE

- **JSON** : APIs web, échange de données entre services
- **CSV** : Données tabulaires, import/export de tableurs
- **YAML** : Fichiers de configuration, documentation
- **XML** : Documents structurés, échange de données entre systèmes d'entreprise

PERFORMANCE

- **CSV** > **JSON** > **YAML** > **XML** (en termes de taille et vitesse de parsing)

FLEXIBILITÉ

- **YAML** > **JSON** > **XML** > **CSV** (en termes de structures de données supportées)

CONCLUSION

Chaque format a ses points forts et ses cas d'utilisation spécifiques. Le choix dépendra des besoins :

- Pour des données simples et tabulaires : CSV
- Pour des APIs web et échanges de données : JSON
- Pour des configurations et documentations : YAML
- Pour des documents complexes et structurés : XML

Exemple sortie code python:

```
Liste des étudiants :
       nom  prenom  age  note_math  note_donnees
0  Meuriel    Hugo   18       20.0          20.0
1    Bruel  Mathis   18        0.0           0.0
2     Remi  Durand   19       -6.0          -6.0

Statistiques :
Nombre total d'étudiants : 3
Moyenne des notes de mathématiques : 4.67
Moyenne des notes de base de données : 4.67
```

Données structurées

Afin de faire cette partie, nous avons repris une base de données de notre projet web Endorah. Nous avons pris une base de données de configuration de Navbar, qui permet grâce à la page administrateur de configurer la navbar.

Nous avons donc repris notre gestionnaire qui avait déjà été fait en Python qui permet de se connecter à la base de données et de gérer celle-ci. Le tout stocker dans une liste Python où nous faisons que des requêtes lors du lancement et lors de modifications.

```
=== Statistiques de Navigation ===
Nombre total d'éléments: 22
Éléments à gauche: 3
Éléments à droite: 19
Éléments parents: 3
Éléments enfants: 16
Éléments avec redirection: 5
```

Dans cette étude, les données de base sont généralement structurées en tables, ce qui rend leur conservation et leur administration plus aisée. Cependant, les structures de données en Python telles que les listes, dictionnaires et objets offrent une flexibilité accrue et autorisent une manipulation des données plus dynamique. L'association entre les deux se trouve dans la transformation des données : lorsqu'elles sont extraites de la base, elles sont fréquemment converties en objets ou en structures telles que des dictionnaires pour faciliter leur manipulation. Cette conversion est cruciale pour assurer que les données demeurent cohérentes et utilisables durant tout le processus de traitement.

Analyse finale:

Pour conclure, l'examen des données structurées, semi-structurées et non structurées met en évidence les avantages et les inconvénients de chaque catégorie de données selon leur format et leur traitement. Les techniques de gestion et de traitement des données diffèrent, mais chacune présente des atouts en fonction du contexte. Ainsi, cette recherche offre une meilleure compréhension de l'interaction entre les formats de données et les structures de données utilisées dans Python.

Info complémentaire :

Nous avons utilisé l'IA pour générer les données en xml, json, csv et yaml, cependant nous avons quand même changé les données pour les personnaliser.
Nous avons aussi utilisé l'ia pour faire les docs sur les formats de données.
Sinon nous avons aussi beaucoup utilisé la doc de python sur comment utiliser les différents formats.
Utilisation d'un générateur de REGEX pour les simples tel que cet exemple (https://regex-generator.olafneumann.org/?sampleText=%5B04%3A00%3A38%5D%20%5BServer%20thread%2FINFO%5D%3A%20%5BVault%5D%20Loading%20server%20plugin%20Vault%20v1.7.3-b131&flags=Pi&selection=1%7CTime,0%7CSquare%20brackets) pour générer un regex pour l'heure dans les logs Minecraft.
Mais utilisation de l'ia dans les REGEX complexes car la compréhension des expressions régulières n'est pas très intuitif.