SAE 1.05
Traiter les données

MULLER Yoann et HARTMANN Quentin
RT132

Sommaire

01

Présentation du besoin

La gestion et l’analyse des fichiers volumineux sont essentielles pour optimiser l’espace disque et améliorer la gestion des ressources informatiques. Cet outil a été développé pour répondre à ce besoin en permettant de :

- Identifier rapidement les fichiers de grande taille dans un répertoire.
- Visualiser leur répartition à l’aide d’un graphique interactif.
- Offrir des fonctionnalités de suppression de fichiers de manière sécurisée.

L’hypothèse de départ est que l’utilisateur ne connaît pas en détail la répartition des fichiers stockés sur son disque et a besoin d’un moyen efficace pour analyser et agir en conséquence.

02

Problématiques et solutions proposées

Problématique 1 : Identification des gros fichiers

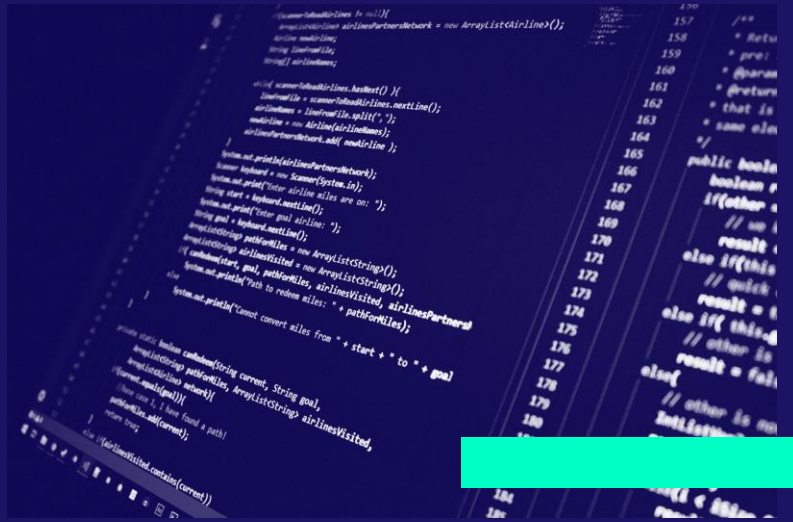
Défi : Scanner un répertoire et lister uniquement les fichiers volumineux.

Solution : Un script (`script_analyse_fichiers.py`) parcourt l'arborescence d'un répertoire, trie les fichiers par taille et sauvegarde les résultats dans un fichier JSON.

Problématique 2 : Visualisation des données

Défi : Offrir une vue claire et interactive des fichiers les plus volumineux.

Solution : Un graphique circulaire ("camembert") est généré (`Creation_Camembert.py`) à partir des données du JSON, affichant la répartition des fichiers par taille.



Problématique 3 : Interaction et actions utilisateur

Défi : Permettre une interaction fluide avec les résultats et offrir des fonctionnalités avancées.

Solution : Une interface graphique en PyQt5 a été développée (`script_affichage_graphique.py`) avec des onglets pour :

- Naviguer entre les différentes visualisations (`Creation_Onglets.py`).
- Consulter les fichiers via des légendes interactives (`Creation_Legendes.py`).
- Lancer des actions comme la suppression de fichiers (`Creation_Boutons.py`).

03

Explication du code

L’outil repose sur plusieurs scripts, chacun ayant un rôle précis

1. Analyse et extraction des fichiers volumineux

1.1 Sélection du répertoire (`script_selection_repertoire.py`)

Ce script permet à l'utilisateur de choisir un répertoire à analyser via une boîte de dialogue.

Analyse et extraction des fichiers volumineux

- `script_selection_repertoire.py` : Ouvre une boîte de dialogue pour sélectionner le répertoire à analyser.
- `script_analyse_fichiers.py` : Parcourt le répertoire, trie et filtre les fichiers selon leur taille.

Explication du code

```
from pathlib import Path

from PyQt5.QtWidgets import QApplication, QFileDialog
```

- `pathlib.Path` est utilisé pour gérer les chemins de fichiers.
- `QFileDialog` est une boîte de dialogue native de PyQt5 permettant de sélectionner un dossier.

```
def selectionner_repertoire():
    app = QApplication([])
    dialog = QFileDialog()
    dialog.setFileMode(QFileDialog.Directory)
    dialog.setOption(QFileDialog.ShowDirsOnly, True)

    if dialog.exec_():
        repertoire_selectionne = dialog.selectedFiles()[0]
        return str(Path(repertoire_selectionne))

    return None
```

- `QFileDialog()` ouvre une boîte de dialogue de sélection.
- `dialog.setFileMode(QFileDialog.Directory)` limite la sélection aux dossiers.
- Si l'utilisateur valide `dialog.exec_()`, on retourne le chemin du dossier.

1.2 Analyse des fichiers (`script_analyse_fichiers.py`)

Ce script récupère tous les fichiers d'un répertoire donné, filtre ceux dépassant une certaine taille et les sauvegarde dans un fichier JSON.

Explication du code

```
from pathlib import Path
import json
import sys
```

- `pathlib.Path` est utilisé pour parcourir le système de fichiers.
- `json` permet de sauvegarder les résultats sous format JSON.

```
def construire_liste_fichiers(repertoire_de_base):
    liste_fichiers = []
    repertoire = Path(repertoire_de_base)

    for fichier in repertoire.rglob('*'):
        if fichier.is_file():
            taille = fichier.stat().st_size
            liste_fichiers.append([str(fichier), taille])

    return liste_fichiers
```

- `repertoire.rglob('*')` parcourt récursivement le dossier.
- `fichier.stat().st_size` récupère la taille du fichier en octets.
- On stocke chaque fichier sous la forme `[chemin, taille]`.

```
def trier_liste_fichiers(liste_fichiers):
    return sorted(liste_fichiers, key=lambda x: x[1], reverse=True)
```

➢ Trie les fichiers du plus gros au plus petit.

```
def filtrer_gros_fichiers(liste_fichiers, taille_mini_mo, nb_maxi_fichiers):
    taille_mini_octets = taille_mini_mo * 1048576
    return [f for f in liste_fichiers if f[1] >= taille_mini_octets][:nb_maxi_fichiers]
```

➢ Filtre uniquement les fichiers dont la taille dépasse un seuil donné

```
def sauvegarder_json(liste_fichiers, nom_fichier):
    for liste_fichier in liste_fichiers:
        liste_fichier[0] = liste_fichier[0].replace('\\', '\\\\')
    with open(nom_fichier, 'w', encoding='utf-8') as f:
        json.dump(liste_fichiers, f, ensure_ascii=False, indent=4)
```

➢ Sauvegarde la liste des fichiers sous forme de fichier JSON.

2. Affichage et interface graphique

2.1 Structure des onglets (`Creation_Onglets.py`)

Ce module permet de structurer l'interface avec des onglets pour afficher les différentes informations.

Affichage et interface graphique

- `script_affichage_graphique.py` : Interface principale avec des onglets et un graphique.
- `Creation_Camembert.py` : Génère un camembert interactif représentant la répartition des fichiers.
- `Creation_Legendes.py` : Affiche les légendes associées au graphique.
- `Creation_Boutons.py` : Crée un bouton pour générer un script de suppression.

Explication du code

```
from PyQt5.QtWidgets import QWidget, QTabWidget, QVBoxLayout, QMainWindow
```

- `QTabWidget` permet de créer un système d'onglets.

```
class Onglets(QMainWindow):
    def __init__(self):
        super().__init__()
        self.setWindowTitle("Résultat de recherche des 'gros' fichiers...")
        self.setGeometry(200, 100, 1000, 700)
        self.onglets = QTabWidget()
        self.setCentralWidget(self.onglets)
```

- Permet de définir une fenêtre avec des onglets.

```
def add_onglet(self, titre, layout_a_ajouter):
    onglet = QWidget()
    layout_onglet = QVBoxLayout()
    layout_onglet.addWidget(layout_a_ajouter)
    onglet.setLayout(layout_onglet)
    self.onglets.addTab(onglet, titre)
```

- Permet d’ajouter de façon dynamique un nouvel onglet.

2.2 Affichage du camembert (`Creation_Camembert.py`)

Ce module génère un graphique circulaire pour visualiser la répartition des fichiers.

Explication du code

```
from PyQt5.QtChart import QChart, QChartView, QPieSeries
from PyQt5.QtGui import QFont
```

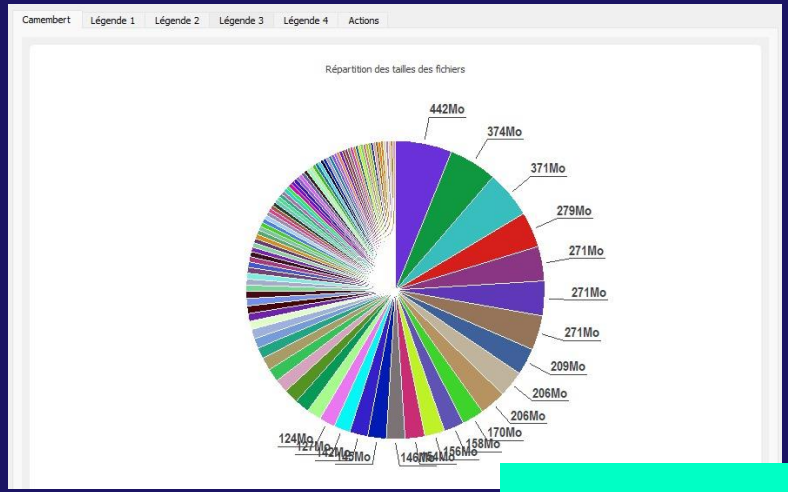
- `QPieSeries` permet de représenter un camembert.

```
def dessine_camembert(self):
    series = QPieSeries()
    series.setLabelsVisible(True)
    taille_totale = sum(t[1] for t in self.liste_fichiers)

    for path_fichier, taille_fichier in self.liste_fichiers:
        etiquette = f"{taille_fichier // 1048576}Mo"
        pourcentage = taille_fichier / taille_totale * 100
        slice_ = series.append(etiquette, pourcentage)
        slice_.setBrush(self.liste_couleurs[len(series)-1])
```

- Crée une série de données pour le graphique.
- Chaque tranche représente un fichier et affiche sa taille en Mo.

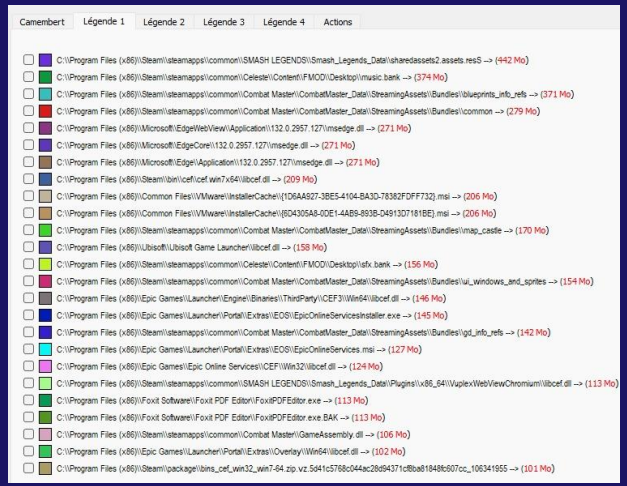
```
fromage = QChart()

fromage.addSeries(series)

fromage.setTitle("Répartition des tailles des fichiers")

fromage.legend().hide()

layout_fromage = QChartView(fromage)
```

- `QChartView` permet d'afficher le graphique.



2.3 Affichage des légendes (`Creation_Legendes.py`)

Ce module affiche une liste des fichiers analysés avec des cases à cocher.



Explication du code

```
for num_slice in range(self.num_legende_start, self.num_legende_stop):
    case_a_cocher = QCheckBox()
    case_a_cocher.setChecked(False)
```

- Création de cases à cocher pour chaque fichier.

```
rectangle_colore = QWidget()
rectangle_colore.setFixedSize(16, 16)
couleur = self.liste_couleurs[num_slice].name()
rectangle_colore.setStyleSheet(f"background-color: {couleur}; border: 1px solid black;")
```

- Création d’un carré coloré représentant chaque fichier dans le camembert.

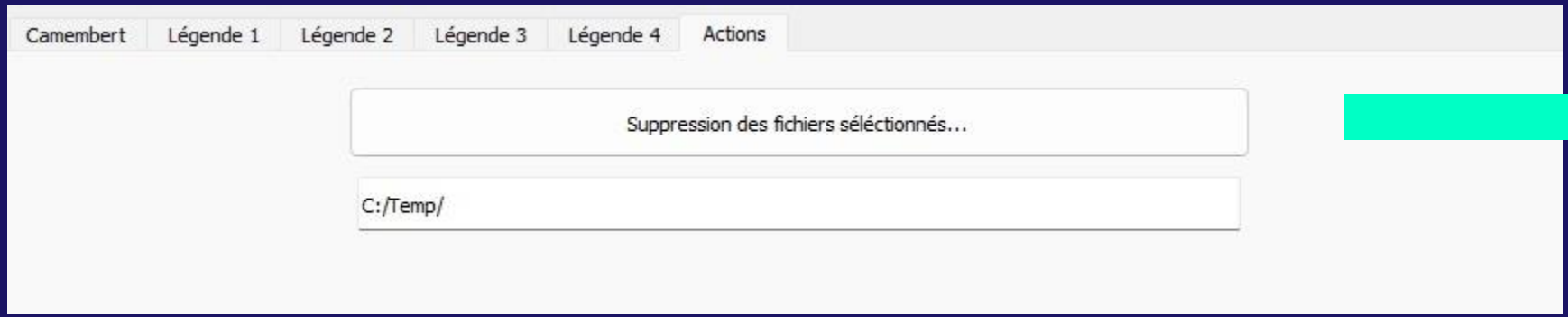
2.4 Ajout de boutons d'action (`Creation_Boutons.py`)

Ce module ajoute un bouton pour générer un script de suppression.

Explication du code

```
bouton_genere_delete_script = QPushButton(« Suppression des fichiers sélectionnés…")
bouton_genere_delete_script.clicked.connect(self.callback)
```

- Ce bouton permet de supprimer les fichiers qui ont été sélectionner dans la légende



3. Intégration et exécution du programme (`script_affichage_graphique.py`)

Ce script assemble tous les éléments précédents.

Interaction avec l'utilisateur

- `Creation_Onglets.py` : Gère les onglets pour naviguer entre les différentes interfaces

Explication du code

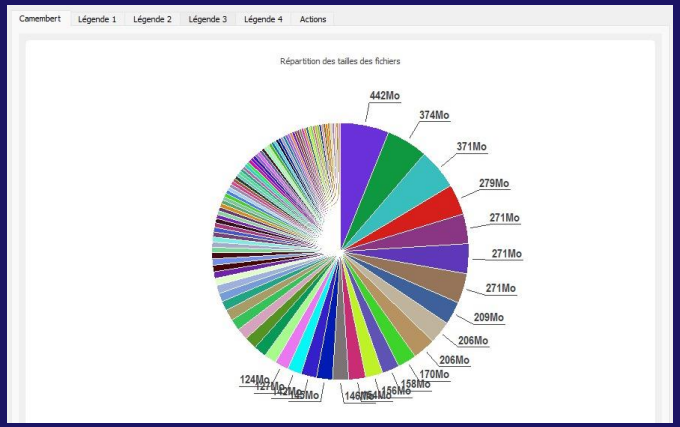
```
app = QApplication(sys.argv)
fenetre = QMainWindow()
onglets = Onglets()
donnees = charger_donnees('gros_fichiers.json')
```

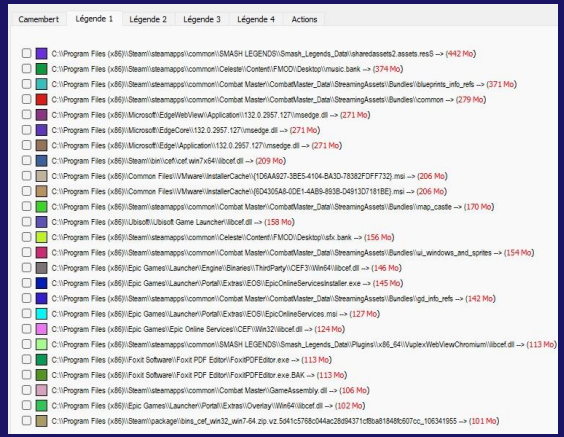
- Charge les données et initialise l'interface.

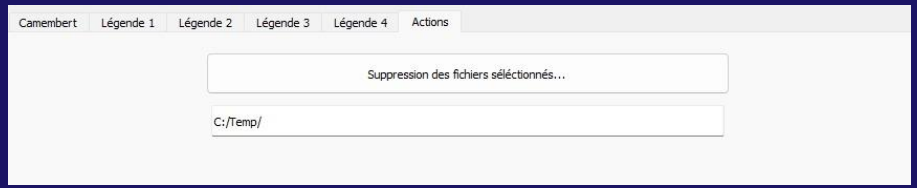
```
camembert = Camembert(donnees, generer_couleurs(len(donnees)))

layout_fromage = camembert.dessine_camembert()

onglets.add_onglet("Camembert", layout_fromage)
```

- Génère le graphique et l'ajoute à l'interface.

Voici l'ensemble des interfaces que nous obtenons en exécutant notre programme








```
PS C:\Users\install\Documents\SAE103> .\launch
Script d'analyse des gros fichiers
Analyse du répertoire : C:\Program Files (x86)
100 gros fichiers trouvés et sauvegardés dans gros_fichiers.json
Erreur : Le fichier de résultats n'a pas été créé.

Appuyez sur une touche pour terminer...
```

04

Atteinte des objectifs et conclusion

L’outil répond efficacement aux objectifs initiaux :

- ✓ Identification et tri des fichiers volumineux.
- ✓ Visualisation claire et intuitive via un graphique et des légendes.
- ✓ Possibilité d'interagir avec les résultats et d'agir en conséquence.

L’outil pourrait être amélioré en ajoutant :

- ➢ Un filtre interactif pour ajuster les seuils de taille des fichiers.
- ➢ Une gestion plus poussée des actions (suppression avec confirmation (archivage, etc.).
- ➢ Une meilleure ergonomie pour l'affichage des résultats.

Ce projet démontre l’importance des outils d’analyse de fichiers et comment une approche basée sur l’interface graphique peut améliorer l’expérience utilisateur.

Merci pour votre attention

Avez-vous des questions ?