Projet professionnel réalisé en entreprise :

Développement d'un utilitaire de bureau pour la validation de conformité des fichiers FEC — « FEC validator »

avec PYTHON et les librairies Tkinter et Pandas

Sommaire

I. Introduction

Le Fichier des Écritures Comptables (FEC) est un document normé par la Direction Générale des Finances Publiques (DGFiP). Toutes les entreprises soumises à l'impôt sur les sociétés ou à l'impôt sur le revenu dans la catégorie BIC/BNC doivent être en mesure de le fournir lors d'un contrôle fiscal.

Au cabinet IVC AUDIT, les clients transmettent ces fichiers FEC qui peuvent contenir plusieurs milliers de lignes comptables. Si une colonne obligatoire manque, si une valeur est absente ou si l'équilibre débit/crédit est rompu, les logiciels d'audit peuvent planter — bloquant ainsi tout le flux de travail des collaborateurs.

Objectif :

Développer un utilitaire Python permettant de valider automatiquement la conformité réglementaire d'un fichier FEC avant son intégration dans les outils d'audit, en offrant un rapport d'analyse clair aux collaborateurs non-techniciens du cabinet.

II. Analyse du besoin

2.1 Qu’est-ce qu'un fichier FEC ?

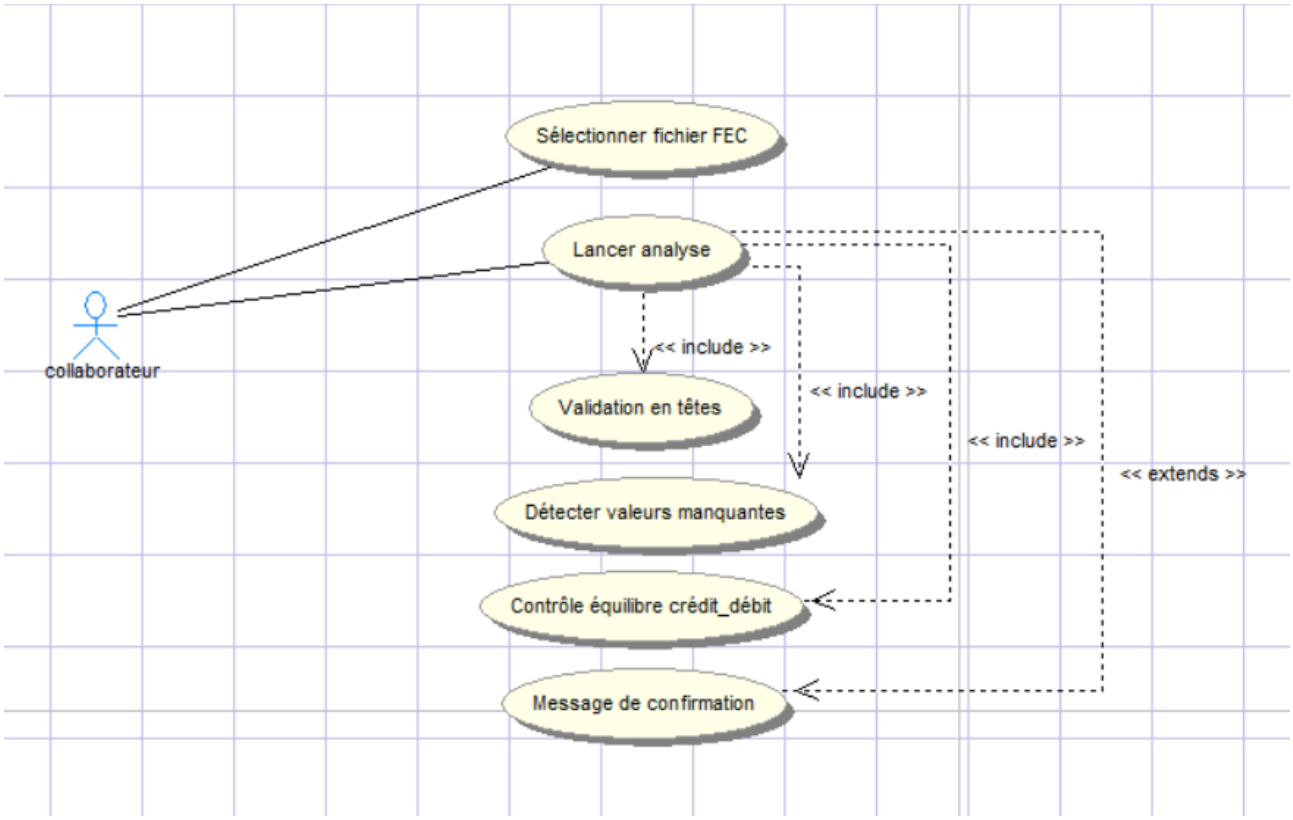
Le Fichier des Écritures Comptables (FEC) est l’export informatique normalisé de toutes les écritures comptables d’un exercice, que toute entreprise tenant une comptabilité informatisée doit obligatoirement remettre à l’administration fiscale en cas de contrôle, prévue dans le code des procédures fiscales. Il contient l’intégralité des journaux, avec chaque écriture détaillée ligne par ligne. Le FEC peut être transmis sous format .txt ou .csv, avec comme séparateurs « ; », « | », ou « tabulation ».

Colonne obligatoire	Description
JournalCode	Code du journal comptable
JournalLib	Libellé du journal comptable
EcritureNum	Numéro de l'écriture
EcritureDate	Date de l'écriture
CompteNum	Numéro du compte général
CompteLib	Libellé du compte général
CompAuxNum	Numéro de compte auxiliaire (si applicable)
CompAuxLib	Libellé de compte auxiliaire
PieceRef	Référence de la pièce justificative
PieceDate	Date de la pièce justificative
EcritureLib	Libellé de l'écriture
Debit	Montants débit
Credit	Montants crédit
EcritureLet	Lettrage
DateLet	Date de lettrage
ValidDate	Date de validation de l’écriture
Montantdevise	Montant devise (si applicable)
Idevise	Code devise

2.2 Fonctionnalités attendues

- Sélectionner un fichier FEC (format .txt ou .csv) via un explorateur de fichiers graphique
- Détecter automatiquement l'encodage du fichier et le séparateur utilisé
- Valider la présence et la conformité des 18 colonnes officielles
- Détecter les valeurs manquantes colonne par colonne
- Contrôler l'équilibre comptable : vérifier que la somme des débits est égale à la somme des crédits
- Générer un rapport d'analyse coloré avec codes visuels succès/erreur, affiché dans l'interface
- Fonctionner sans installation de Python sur les postes utilisateurs
- Être accessible depuis SharePoint (Microsoft 365) à l'ensemble des collaborateurs

2.3 Diagramme de cas d'utilisation (DCU avec Windesign)



III. Développement

3.1 Choix techniques

Technologie	Usage	Justification
Python 3	Langage principal	Appris en cours de formation, ecosystème data riche
Pandas	Traitement des données tabulaires	Lecture CSV robuste, gestion encodages, calculs vectorisés sur milliers de lignes
Tkinter	Interface graphique (GUI)	Bibliothèque native Python, aucune dépendance externe
PyInstaller	Compilation .exe	Déploiement sans environnement Python
SharePoint (M365)	Distribution	Infrastructure déjà en place, accessible à tous

3.2 Architecture de l'application

L'application est structurée en modèle orienté objet autour d'une classe principale **FecValidatorApp**, cette approche facilite la maintenance et l'extension du code :

- La méthode **__init__()** initialise la fenêtre principale et lance la construction de l'interface.

```python
import pandas as pd
import os
import tkinter as tk
from tkinter import filedialog, messagebox

class FecValidatorApp:
    def __init__(self, root):
        self.root = root
        self.root.title("IVC AUDIT - FEC validator")
        self.root.geometry("750x600")
        self.root.minsize(650, 500)

        self.chemin_fichier = tk.StringVar()
        self.creer_interface()
```

- La méthode **creer_interface()** construit tous les widgets Tkinter (header, champ fichier, boutons, zone rapport).

```python
def creer_interface(self):
    header_frame = tk.Frame(self.root, bg="#1E293B", pady=15)
    header_frame.pack(fill="x")

    titre_label = tk.Label(header_frame, text="IVC AUDIT- FEC validator",
                           font=("Arial", 16, "bold"), fg="white", bg="#1E293B")
    titre_label.pack()

    corps_frame = tk.Frame(self.root, padx=20, pady=10)
    corps_frame.pack(fill="both", expand=True)

    select_frame = tk.LabelFrame(corps_frame, text=" Sélectionner le fichier FEC ", font=("Arial", 10, "bold"), padx=10, pady=10)
    select_frame.pack(fill="x", pady=10)

    entry_fichier = tk.Entry(select_frame, textvariable=self.chemin_fichier, width=50, font=("Arial", 10))
    entry_fichier.pack(side="left", padx=5, expand=True, fill="x")

    btn_parcourir = tk.Button(select_frame, text="Parcourir...", command=self.parcourir_fichier,
                              bg="#3B82F6", fg="white", font=("Arial", 10, "bold"), relief="flat", padx=10)
    btn_parcourir.pack(side="right", padx=5)

    self.btn_analyser = tk.Button(corps_frame, text="⚡ Lancer l'analyse de conformité", command=self.analyser_fec,
                                  bg="#10B981", fg="white", font=("Arial", 11, "bold"), relief="flat", pady=8)
    self.btn_analyser.pack(fill="x", pady=10)

    rapport_frame = tk.LabelFrame(corps_frame, text=" Rapport d'analyse de données ", font=("Arial", 10, "bold"), padx=10, pady=10)
    rapport_frame.pack(fill="both", expand=True)

    scrollbar = tk.Scrollbar(rapport_frame)
    scrollbar.pack(side="right", fill="y")

    self.txt_rapport = tk.Text(rapport_frame, yscrollcommand=scrollbar.set, font=("Consolas", 10), bg="#F8FAFC", fg="#334155")
    self.txt_rapport.pack(fill="both", expand=True)
    scrollbar.config(command=self.txt_rapport.yview)

    self.txt_rapport.tag_config("titre_etape", foreground="#1E293B", font=("Arial", 10, "bold"))
    self.txt_rapport.tag_config("erreur", foreground="#EF4444", font=("Consolas", 10, "bold"))
    self.txt_rapport.tag_config("succes", foreground="#10B981")

    self.ecrire_rapport("Veuillez sélectionner un fichier FEC (.txt ou .csv) puis lancez l'analyse.")
```

- La méthode **parcourir_fichier()** ouvre l'explorateur système et charge le chemin du fichier sélectionné.

```python
def parcourir_fichier(self):
    fichier_selectionne = filedialog.askopenfilename(
        title="Sélectionner un fichier FEC",
        filetypes=[("Fichiers FEC (*.txt, *.csv)", "*.txt *.csv"), ("Tous les fichiers", "*.*")]
    )
    if fichier_selectionne:
        self.chemin_fichier.set(fichier_selectionne)
        self.txt_rapport.delete("1.0", tk.END)
        self.ecrire_rapport(f"Fichier chargé : {os.path.basename(fichier_selectionne)}", tag="titre_etape")
        self.ecrire_rapport("Prêt pour l'analyse règlementaire.", tag="succes")
```

- La méthode **ecrire_rapport()** insère une ligne dans la zone texte avec le tag de couleur correspondant (succès / erreur).

```python
def ecrire_rapport(self, texte, tag=None):
    if tag:
        self.txt_rapport.insert(tk.END, texte + "\n", tag)
    else:
        self.txt_rapport.insert(tk.END, texte + "\n")
        self.txt_rapport.see(tk.END)
```

- La méthode **detecter_separateur_et_encodage()** teste les encodages UTF-8-BOM et Latin-1 ; détecte le séparateur optimal par comptage.

```python
    def detecter_separateur_et_encodage(self, chemin):
        encodages_a_tester = ['utf-8-sig', 'latin-1']

        for enc in encodages_a_tester:
            try:
                with open(chemin, 'r', encoding=enc) as f:
                    premiere_ligne = f.readline()

                if not premiere_ligne:
                    continue

                separateurs = [';', '|', '→', ',', '\t']
                compteur = {sep: premiere_ligne.count(sep) for sep in separateurs}
                meilleur_sep = max(compteur, key=compteur.get)

                if compteur[meilleur_sep] > 0:
                    return meilleur_sep, enc
            except (UnicodeDecodeError, LookupError):
                continue

        return None, None
```

- La méthode **analyser_fec()** orchestre les 3 étapes de validation et alimente le rapport en temps réel.

```python
    def analyser_fec(self):
        chemin = self.chemin_fichier.get()
        if not chemin:
            messagebox.showwarning("Attention", "Veuillez d'abord sélectionner un fichier !")
            return

        self.txt_rapport.delete("1.0", tk.END)
        self.ecrire_rapport(f"")
        self.ecrire_rapport(f"ANALYSE DU FICHIER : {os.path.basename(chemin)}", tag="titre_etape")

        if os.path.exists(chemin) and os.path.getsize(chemin) == 0:
            self.ecrire_rapport("Le fichier sélectionné est complètement VIDE !", tag="erreur")
            messagebox.showwarning("Fichier Vide", "Le fichier sélectionné ne contient aucune donnée.")
            return

        # On récupère le séparateur ET l'encodage détecté
        sep_detecte, encodage_detecte = self.detecter_separateur_et_encodage(chemin)

        if not sep_detecte:
            self.ecrire_rapport("Impossible d'analyser le fichier (problème de format ou d'encodage) !", tag="erreur")
            messagebox.showerror("Erreur", "Le format ou l'encodage du fichier n'est pas supporté.")
            return

        colonnes_officielles = [
            "JournalCode", "JournalLib", "EcritureNum", "EcritureDate",
            "CompteNum", "CompteLib", "CompAuxNum", "CompAuxLib",
            "PieceRef", "PieceDate", "EcritureLib", "Debit",
            "Credit", "EcritureLet", "DateLet", "ValidDate",
            "Montantdevise", "Idevise"
        ]
```

```python
        try:
            # Utilisation de l'encodage qui a fonctionné à la détection
            df = pd.read_csv(chemin, sep=sep_detecte, engine='python', encoding=encodage_detecte, dtype=str)

            if df.empty:
                self.ecrire_rapport("Le fichier ne contient aucune ligne comptable !", tag="erreur")
                messagebox.showwarning("Fichier Vide", "Le fichier contient des en-têtes mais aucune donnée.")
                return

            self.ecrire_rapport(f"Lignes comptables chargées : {len(df)}")
            self.ecrire_rapport(f"Encodage détecté : {encodage_detecte.upper()}")
            nom_sep = "Tabulation" if sep_detecte == "\t" else f"'{sep_detecte}'"
            self.ecrire_rapport(f"Séparateur détecté : {nom_sep}\n")
```

```
            #etape 1 en têtes
            self.ecrire_rapport("VALIDATION DES EN TETES", tag="titre_etape")
            colonnes_trouvees = list(df.columns)
            manquantes = [c for c in colonnes_officielles if c not in colonnes_trouvees]
            inattendues = [c for c in colonnes_trouvees if c not in colonnes_officielles]

            if len(manquantes) == 0:
                self.ecrire_rapport("Structure conforme : les 18 colonnes officielles sont présentes", tag="succes")
            else:
                self.ecrire_rapport(f"Structure non-conforme ({len(colonnes_trouvees)}/18 colonnes détectées) !", tag="erreur")
                if manquantes: self.ecrire_rapport(f"  -> Manquantes : {manquantes}")
                if inattendues: self.ecrire_rapport(f"  -> Non-standard : {inattendues}")

            # étape 2 valeurs manquante
            self.ecrire_rapport("\nANALYSE CONFORMITE (Champs obligatoires)", tag="titre_etape")
            colonnes_a_tester = [c for c in ["JournalCode", "JournalLib", "EcritureNum", "EcritureDate", "CompteNum", "CompteLib", "CompAuxNum", "CompAuxLib",
                                             "PieceRef", "PieceDate", "EcritureLib", "Debit", "Credit", "EcritureLet", "DateLet", "ValidDate", "Montantdevise", "Idevise"
                                             ]
                                 if c in df.columns]
            for col in colonnes_a_tester:
                vides = df[col].isnull().sum() + (df[col].str.strip() == "").sum()
                if vides > 0:
                    self.ecrire_rapport(f"Colonne '{col}' contient {vides} valeur(s) vide(s) !", tag="erreur")
                else:
                    self.ecrire_rapport(f"Colonne '{col}' saine (aucune valeur manquante)", tag="succes")
```

```
                # étape 3 ctrl debit/credit
                self.ecrire_rapport("\nCONTROLE EQUILIBRE DEBIT/CREDIT", tag="titre_etape")
                if "Debit" in df.columns and "Credit" in df.columns:
                    debits_clean = df["Debit"].str.replace(',', '.', regex=False).fillna('0')
                    credits_clean = df["Credit"].str.replace(',', '.', regex=False).fillna('0')

                    total_debit = pd.to_numeric(debits_clean, errors='coerce').sum()
                    total_credit = pd.to_numeric(credits_clean, errors='coerce').sum()

                    self.ecrire_rapport(f"Somme totale des Débits  : {total_debit:.2f} €")
                    self.ecrire_rapport(f"Somme totale des Crédits : {total_credit:.2f} €")

                    ecart = total_debit - total_credit
                    if abs(ecart) < 0.01:
                        self.ecrire_rapport("Le fichier FEC est parfaitement équilibré (Débit = Crédit)", tag="succes")
                    else:
                        self.ecrire_rapport(f"Erreur comptable, le fichier n'est pas équilibré ! Écart de : {ecart:.2f} €", tag="erreur")
                else:
                    self.ecrire_rapport("Impossible de vérifier l'équilibre : colonnes Débit/Crédit introuvables !", tag="erreur")

                self.ecrire_rapport("\n")
                self.ecrire_rapport("FIN DE L'ANALYSE", tag="titre_etape")
                self.ecrire_rapport("")
                messagebox.showinfo("Succès", "L'analyse du fichier FEC est terminée !")

            except Exception as e:
                self.ecrire_rapport(f"\nÉchec de l'analyse : {e}", tag="erreur")
                messagebox.showerror("Erreur", f"Erreur lors du traitement : {e}")

if __name__ == "__main__":
    root = tk.Tk()
    app = FecValidatorApp(root)
    root.mainloop()
```

3.3 Moteur d'analyse de conformité

Le cœur de l'application est la méthode **analyser_fec()**, qui exécute trois étapes de validation successives et indépendantes :

Étape 1 — Validation des en-têtes

La liste des colonnes du DataFrame est comparée aux 18 colonnes officielles du livre des procédures fiscales. L'outil identifie les colonnes manquantes (bloquant) et les colonnes non standard (avertissement).

```
#etape 1 en têtes
self.ecrire_rapport("VALIDATION DES EN TETES", tag="titre_etape")
colonnes_trouvees = list(df.columns)
manquantes = [c for c in colonnes_officielles if c not in colonnes_trouvees]
inattendues = [c for c in colonnes_trouvees if c not in colonnes_officielles]

if len(manquantes) == 0:
    self.ecrire_rapport("Structure conforme : les 18 colonnes officielles sont présentes", tag="succes")
else:
    self.ecrire_rapport(f"Structure non-conforme ({len(colonnes_trouvees)}/18 colonnes détectées) !", tag="erreur")
    if manquantes: self.ecrire_rapport(f"  -> Manquantes : {manquantes}")
    if inattendues: self.ecrire_rapport(f"  -> Non-standard : {inattendues}")
```

Étape 2 — Détection des valeurs manquantes

Pour chaque colonne présente, l'outil cumule les valeurs NULL (isnull) et les chaînes vides après normalisation (str.strip). Ce double contrôle est indispensable car les FEC contiennent fréquemment des espaces insécables à la place de valeurs vides.

```
# étape 2 valeurs manquante
self.ecrire_rapport("\nANALYSE CONFORMITE (Champs obligatoires)", tag="titre_etape")
colonnes_a_tester = [c for c in ["JournalCode", "JournalLib", "EcritureNum", "EcritureDate", "CompteNum", "CompteLib", "CompAuxNum", "CompAuxLib",
                                 "PieceRef", "PieceDate", "EcritureLib", "Debit", "Credit", "EcritureLet", "DateLet", "ValidDate", "Montantdevise", "Idevise"
                                ]
                     if c in df.columns]
for col in colonnes_a_tester:
    vides = df[col].isnull().sum() + (df[col].str.strip() == "").sum()
    if vides > 0:
        self.ecrire_rapport(f"Colonne '{col}' contient {vides} valeur(s) vide(s) !", tag="erreur")
    else:
        self.ecrire_rapport(f"Colonne '{col}' saine (aucune valeur manquante)", tag="succes")
```

Étape 3 — Contrôle de l'équilibre débit/crédit

La règle comptable fondamentale impose que la somme des débits soit strictement égale à la somme des crédits. L'outil normalise d'abord les séparateurs décimaux (virgule vers point), convertit en numérique et calcule l'écart. Une tolérance de 0,01 € est appliquée pour les arrondis de conversion.

```
# étape 3 ctrl debit/credit
self.ecrire_rapport("\nCONTROLE EQUILIBRE DEBIT/CREDIT", tag="titre_etape")
if "Debit" in df.columns and "Credit" in df.columns:
    debits_clean = df["Debit"].str.replace(',', '.', regex=False).fillna('0')
    credits_clean = df["Credit"].str.replace(',', '.', regex=False).fillna('0')

    total_debit = pd.to_numeric(debits_clean, errors='coerce').sum()
    total_credit = pd.to_numeric(credits_clean, errors='coerce').sum()

    self.ecrire_rapport(f"Somme totale des Débits  : {total_debit:.2f} €")
    self.ecrire_rapport(f"Somme totale des Crédits : {total_credit:.2f} €")

    ecart = total_debit - total_credit
    if abs(ecart) < 0.01:
        self.ecrire_rapport("Le fichier FEC est parfaitement équilibré (Débit = Crédit)", tag="succes")
    else:
        self.ecrire_rapport(f"Erreur comptable, le fichier n'est pas équilibré ! Écart de : {ecart:.2f} €", tag="erreur")
else:
    self.ecrire_rapport("Impossible de vérifier l'équilibre : colonnes Débit/Crédit introuvables !", tag="erreur")
```

3.4 Interface graphique

L'interface Tkinter est organisée en zones fonctionnelles imbriquées (LabelFrame) pour une lisibilité maximale, même pour des utilisateurs non-techniciens, elle adopte une charte graphique épurée :

Composant UI	Rôle et comportement
Header sombre	Bandeau titre de l'application, identitaire IVC AUDIT
Champ et bouton parcourir	Sélection du fichier FEC via filedialog avec le chemin affiché dans le champ Entry
Bouton lancer analyse	Déclenche **analyser_fec()**
Zone rapport	Affiche le rapport ligne par ligne avec tags colorés
Tags de couleur Tkinter	tag_config permet à l'utilisateur de voir immédiatement les anomalies

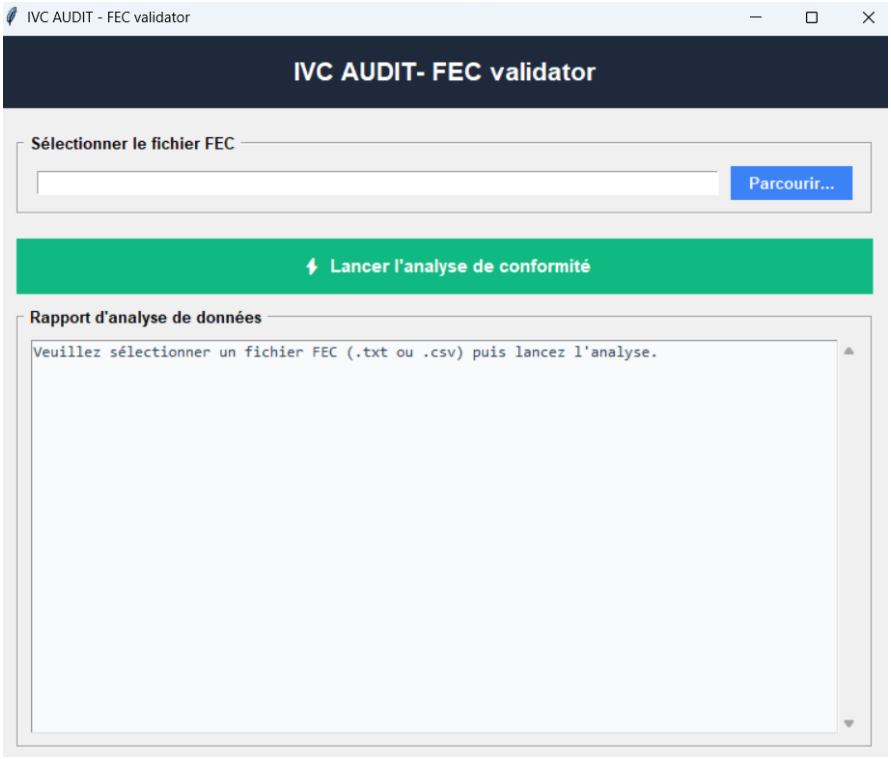
3.5 Déploiement

- Compilation en exécutable Windows autonome (.exe) via PyInstaller, incluant Python et toutes les dépendances (Pandas, NumPy) : aucune installation requise sur les postes.
- Dépôt sur SharePoint (Microsoft 365) dans un espace partagé accessible à l'ensemble des collaborateurs du cabinet.
- Mise à jour transparente : une nouvelle version compilée remplace simplement le fichier SharePoint, sans intervention sur les postes utilisateur.

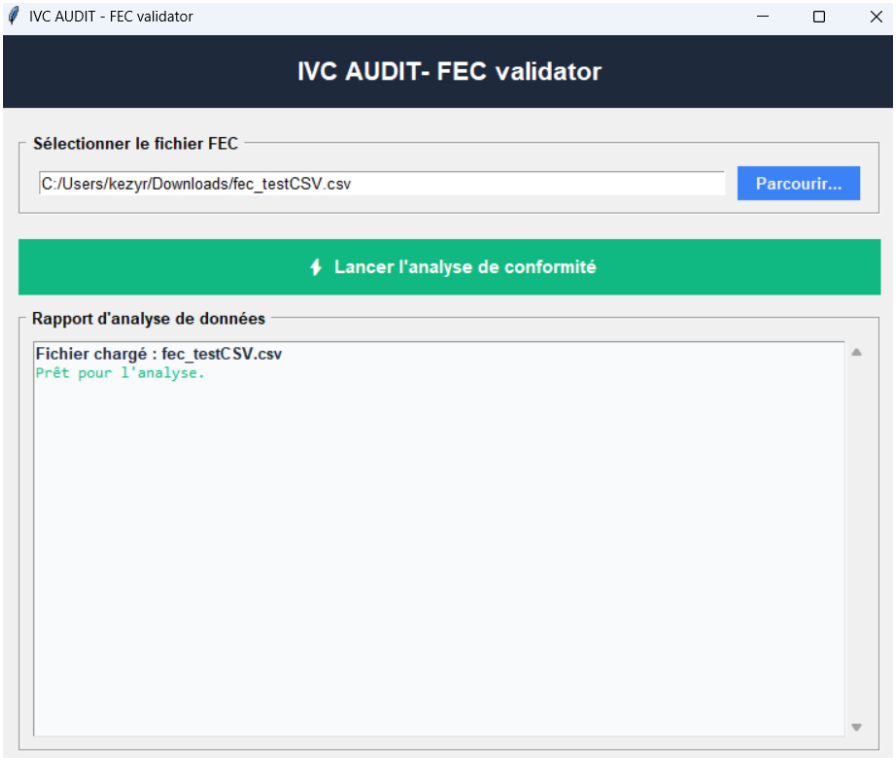
IV. Résultat et bénéfices

4.1 Fonctionnement de l'outil

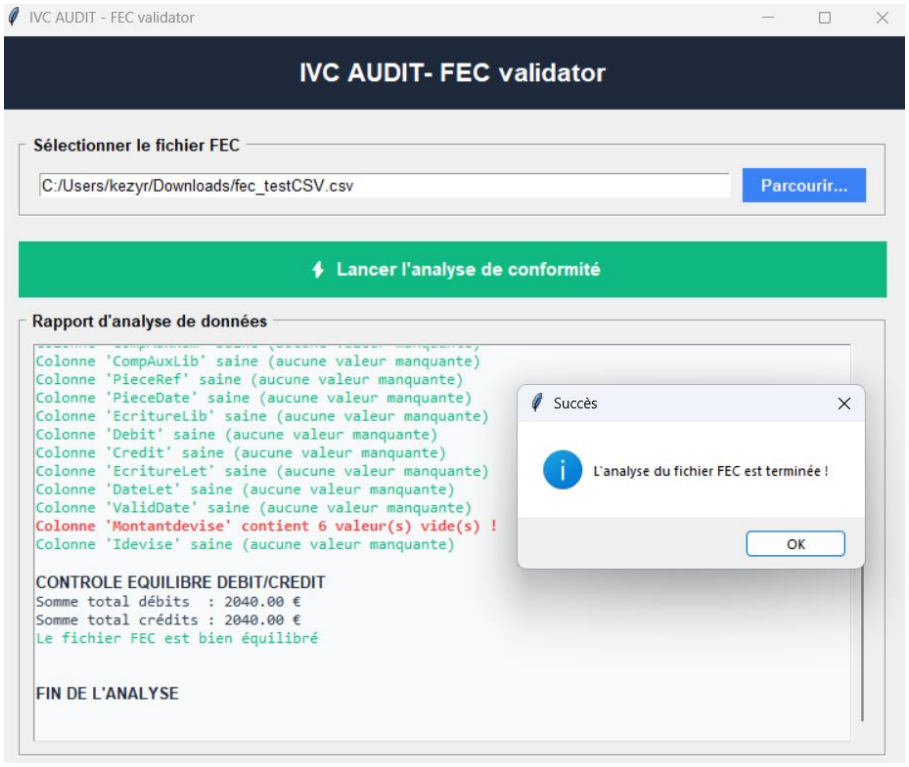
Une petite fenêtre s'affiche à l'exécution de l'utilitaire, l'utilisateur est invité à sélectionner son fichier via l'explorateur avec le bouton « parcourir » :

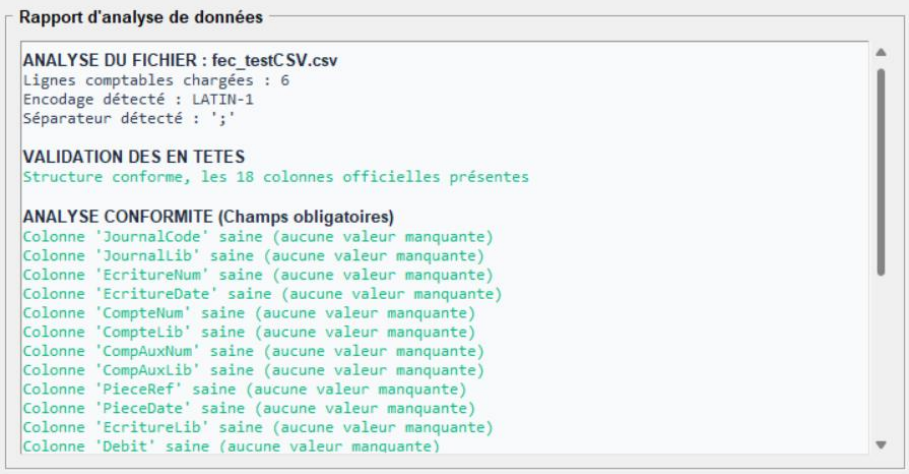


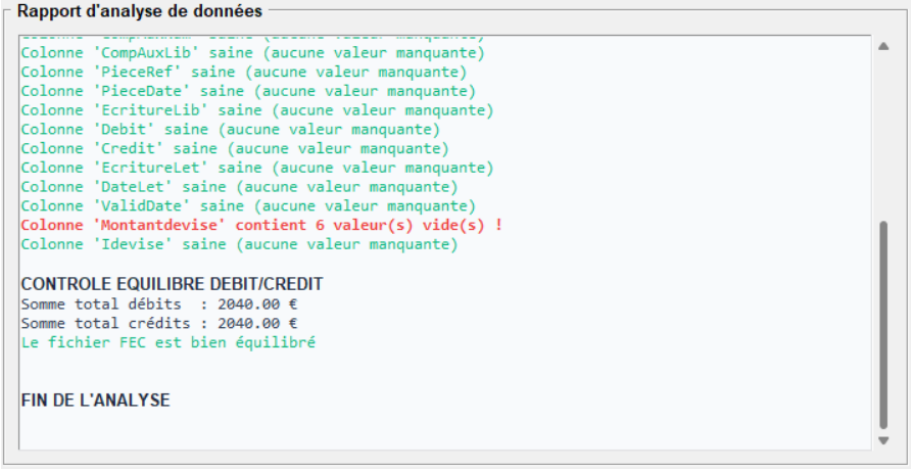
L'utilisateur peut maintenant lancer l'analyse de son fichier FEC :



Enfin le rapport est généré avec toutes les indications nécessaires sur la conformité du fichier traité, une fenêtre contextuelle l'avertissement de la fin de l'analyse apparaît et en cliquant sur « ok » l'utilisateur accède au rapport :







4.2 Bénéfices pour l’entreprise

Avant la solution	Après la solution
Vérification manuelle impossible sur des fichiers de plusieurs milliers de lignes	Analyse complète en quelques secondes, quelle que soit la taille du fichier
Plantage des logiciels d'audit	Anomalies identifiées en amont, avant intégration dans les outils
Diagnostic flou	Rapport précis : colonne incriminée et nombre exact de valeurs manquantes
Dépendance à un technicien pour lire les fichiers bruts	Outil autonome accessible à tous les collaborateurs via SharePoint

VI. Conclusion et perspectives

Ce projet illustre une démarche complète de développement orienté vers la valeur métier : identifier une contrainte réglementaire précise, comprendre comment elle impacte le travail quotidien des collaborateurs, et livrer une solution technique qui optimise les processus métiers.

Perspectives d'évolution

- Ajout d'un contrôle sur le format des dates (EcritureDate, PieceDate) : vérification du format AAAA-MM-JJ imposé par la DGFiP.
- Détection des doublons de numéros d'écriture (EcritureNum) qui invalideraient le fichier lors d'un contrôle fiscal.
- Export du rapport d'analyse au format PDF pour transmission au client avec les corrections à effectuer.
- Ajout d'un contrôle de la séquence chronologique des écritures par journal.
- Mode batch : analyser plusieurs fichiers FEC d'un seul dossier en une seule opération.

En résumé, cet outil transforme une tâche de vérification réglementaire autrefois compliqué à réaliser manuellement en un processus automatisé, fiable et accessible à l'ensemble des collaborateurs du cabinet