E1

Sofian SAADI

Bloc de compétences 1 : Réaliser la collecte, le stockage et la mise à disposition des données

d'un projet en intelligence artificielle

Compétence(s) visée(s)

C1. Automatiser l'extraction de données depuis un service web, une page web (scraping*), un fichier

de données, une base de données et un système big data* en programmant le script* adapté afin

de pérenniser la collecte des données nécessaires au projet.

C2. Développer des requêtes de type SQL d'extraction des données depuis un système de gestion

de base de données et un système big data en appliquant le langage de requête propre au système afin

de préparer la collecte des données nécessaires au projet.

C3. Développer des règles d'agrégation de données issues de différentes sources en programmant,

sous forme de script, la suppression des entrées corrompues et en programmant l'homogénéisation des

formats des données afin de préparer le stockage du jeu de données final.

C4. Créer une base de données dans le respect du RGPD en élaborant les modèles conceptuels et

physiques des données à partir des données préparées et en programmant leur import afin de stocker le

jeu de données du projet.

C5. Développer une API mettant à disposition le jeu de données en utilisant l'architecture REST afin

de permettre l’exploitation du jeu de données par les autres composants du

projet.

C1 – Automatiser l’extraction de données depuis un fichier PDF

Ce qu’il fallait faire

Le projet consistait à **automatiser l’extraction d’informations importantes** (adresse, numéro de commande, produits,quantité, prix , etc.) depuis des bons de commande reçus sous forme de fichiers PDF. Chaque bailleur (comme Val d’Oise ou Versailles Habitat) avait un format différent. Il fallait donc **détecter le type de document, extraire le texte s’il était scanné, puis parser les informations**, afin de pouvoir **effectuer des saisies de masse**, **réduire la charge de travail des secrétaires** et **gagner du temps** sur le traitement des documents.

Quelle solution technique j’ai choisie

Mise en place une architecture modulaire avec une fonction de dispatch (`detect_and_dispatch`) qui reconnaît automatiquement le type de document selon son contenu textuel. Ensuite, chaque format est traité par un parseur spécifique (par exemple, `extract_data_from_pdf_ocr` pour Val d’Oise). Pour les documents scannés, j’ai intégré un OCR technologie informatique permettant de **reconnaître et convertir automatiquement du texte présent dans des images ou des documents scannés en texte éditable**.(EasyOCR ou PaddleOCR) pour récupérer le texte.

EasyOCR : bibliothèque Python d'OCR qui utilise l'**intelligence artificielle** pour extraire du texte à partir d'images dans plusieurs langues, facilement intégrable dans des projets.

PaddleOCR : bibliothèque OCR développée par **Baidu**, basée sur l'IA, permettant de **reconnaître et extraire du texte** dans des images avec de bonnes performances pour différentes langues et formats.

On peut utiliser EasyOCR et PaddleOCR de manière complémentaire car ils peuvent avoir des performances différentes selon la langue, la qualité ou le format des images, permettant ainsi de choisir le meilleur outil pour chaque cas.

Implémentation

Détection automatique du type de PDF client :

```python
def detect_and_dispatch(pdf_path: str) -> dict:
    text = get_text_from_pdf(pdf_path).lower()
    if "val d'oise" in text:
        return extract_data_from_pdf_ocr(pdf_path)
    elif "versailles" in text:
        return extract_data_from_pdf_versail(pdf_path)
```

Grâce à cette logique, je peux diriger chaque document vers un parseur spécifique selon son contenu, ce qui me permet de traiter différents formats clients.

OCR avec EasyOCR pour les PDF scannés :

```python
def extract_text_from_ocr(pdf_path):
    images = convert_from_path(pdf_path)
    for img in images:
        img_cv = np.array(img)
        img_cv = cv2.cvtColor(img_cv, cv2.COLOR_RGB2GRAY)
        img_cv = cv2.adaptiveThreshold(img_cv, 255, cv2.ADAPTIVE_THRESH_GAU
        text += pytesseract.image_to_string(img_cv, lang="fra")
```

Chaque fonction `extract_data_from_pdf_<client>` renvoie un dictionnaire structuré contenant tous les champs nécessaires.

Détection automatique du type de PDF client :

```python
def is_pdf_scanned(pdf_path):
    doc = fitz.open(pdf_path)
    for page in doc:
        if page.get_text("text").strip():
            return False
    return True
```

Cette fonction me permet de détecter automatiquement si un PDF est scanné (image) ou structuré (texte), pour choisir entre OCR ou parsing direct.

Cette partie utilise pytesseract bibliothèque Python qui sert d'**interface pour Tesseract OCR**, permettant de **lire et extraire du texte à partir d'images directement dans des programmes Python**. pour convertir des PDF scannés en texte. Le traitement inclut un seuillage adaptatif pour améliorer la précision OCR.

Détection automatique du type de PDF client

Vérification du fonctionnement

J'ai testé plusieurs fichiers PDF de différents formats. À chaque fois, j'ai vérifié que la bonne fonction de parsing était déclenchée, que le texte était bien extrait (même pour les PDF scannés) et que les données finales étaient dans le bon format. J'ai aussi renvoyé le résultat en JSON pour pouvoir l'afficher et le valider dans une interface front.
Contenu du PDF pour un bailleur (Paris Habitat), sachant que chaque bailleur possède un document distinct et que les informations peuvent varier.

BON DE COMMANDE
n°S20027 du 07/01/2025

Edité le 07/01/2025

Paris Habitat
vivre ensemble la ville

Paris Habitat - OPH
21 bis, rue Claude Bernard
75253 Paris Cedex 05 - BP 137
N° de TVA : FR 433 448 18825
N° de Siret : 344 810 825 00366
Dépôt factures sur le portail CHORUS Pro

Marché Ref : 2022/M0150 - IKOS : 220150

SAS RENOV OUEST DG
302 AVENUE JEAN JAURES
95100 ARGENTEUIL

Contact : OUEDRAOGO Yessoufou
Tél : 0640708439 mail :

DTSO Ag. A2 Gérance P4
87 boulevard LEFEBVRE
75015 PARIS

Travaux à terminer pour le 07/01/2025

Commande suivie par : M OUEDRAOGO Yessoufou
tél: 0640708439 portable: 0640708439 mail : yessoufou.ouedraogo@parishabitat.fr

Prestations parties privatives - Lieu des travaux				
HAMEAU 16/26 BâtA Esc04 (1-15HA-A-04 **24 rue du Hameau** **75015 PARIS** **Logement n°153584, porte n°0151, Rez-de-chaussée, Type 3**	**Gardien :** M OUEDRAOGO Yessoufou (yessoufou.ouedraogo@parishabitat.fr) **16/26 RUE DU HAMEAU - 75015 PARIS** **tél :0640708439 portable : 0640708439** **Occupant actuel : MME WILLIAMS Gloria (520148/34)**			
	P.U.HT (€)	**Quantité**	**Montant HT (€)**	**Montant TTC (€)**
REVIS° OCCULTAT° DE FENE (ECME32) Révision d'une occultation de fenêtre par volet roulant y compris mécanisme toute dimension Le volet roulant de la fenêtre de la chambre ne fonctionne pas. merci de contacter mon collaborateur au 0640708439 TVA appliquée : 10,00 % Code : T10 Budget : 61517N (PPHC-NR) *DTSO*	**125,93 €**	**1,00 U**	**125,93 €**	**138,52 €**

Descriptif : Le volet roulant de la chambre ne fonctionne pas

C2 – Développer des requêtes de type SQL d’extraction des données depuis un système de gestion

Lors de mon stage, je n’ai pas eu l’occasion de répondre directement à la compétence C2. C’est pourquoi j’ai choisi d’ajouter un exercice réalisé au cours de l’année, qui illustre ma capacité à développer des requêtes SQL d’extraction de données depuis un système de gestion.

Ce qu’il fallait faire

Le projet consistait à modéliser un système de gestion généalogique permettant de stocker et d’extraire des informations sur les individus et leurs relations familiales (parents, enfants, conjoints, etc.). L’objectif était de pouvoir interroger la base pour retrouver des liens familiaux complexes (fratrie, ascendance, descendance, etc.) tout en respectant les contraintes métier (cohérence des dates, types de relations, etc.).

Quelle solution technique j’ai choisie

J’ai conçu une base relationnelle autour de deux tables principales : `Personnes` et `Relations`. La table `Personnes` contient les données personnelles (nom, prénoms, dates), tandis que la table `Relations` permet de représenter dynamiquement tous les types de liens familiaux entre individus (parent biologique, adoptif, enfant, conjoint…).

Cette structure permet une grande flexibilité dans les requêtes SQL, notamment grâce à des jointures sur la table `Relations` et des auto-références sur `Personnes`.

Comment je l’ai implémentée

Structure SQL de base

```sql
CREATE TABLE Personnes (
    id INT PRIMARY KEY,
    nom VARCHAR(100),
    prenoms TEXT[],
    date_naissance DATE,
    date_deces DATE
);

CREATE TABLE Relations (
    id INT PRIMARY KEY,
    id_source INT REFERENCES Personnes(id),
    id_cible INT REFERENCES Personnes(id),
    type_relation VARCHAR(50) CHECK (type_relation IN (
        'parent_biologique', 'parent_adoptif', 'enfant', 'conjoint'
    ))
);
```

Requêtes d’extraction développées

1. Récupérer tous les enfants d'un individu

```sql
SELECT p.*
FROM Relations r
JOIN Personnes p ON p.id = r.id_cible
WHERE r.id_source = :id_parent
AND r.type_relation IN ('parent_biologique', 'parent_adoptif');
```

2. Récupérer tous les parents d'un individu

```sql
SELECT p.*
FROM Relations r
JOIN Personnes p ON p.id = r.id_source
WHERE r.id_cible = :id_enfant
AND r.type_relation IN ('parent_biologique', 'parent_adoptif');
```

Récupérer les frères et sœurs d'un individu

```sql
SELECT DISTINCT fr.*
FROM Relations r1
JOIN Relations r2 ON r1.id_source = r2.id_source
JOIN Personnes fr ON fr.id = r2.id_cible
WHERE r1.id_cible = :id_individu
AND r1.type_relation IN ('parent_biologique', 'parent_adoptif')
AND r2.type_relation IN ('parent_biologique', 'parent_adoptif')
AND r2.id_cible != :id_individu;
```

Ici, on fait plusieurs jointures pour retrouver les individus qui partagent les mêmes parents.

Une jointure en SQL permet de relier deux tables (ou une table à elle-même) pour combiner des informations. Dans ton cas, on relie les individus entre eux pour reconstruire les liens familiaux.

Récupérer les conjoints d'un individu

```sql
SELECT p.*
FROM Relations r
JOIN Personnes p ON p.id = r.id_cible
WHERE r.id_source = :id_individu
AND r.type_relation = 'conjoint';
```

Récupérer les ancêtres jusqu'à N générations PostgreSQL

```sql
WITH RECURSIVE Ancetres AS (
    SELECT r.id_source AS id_ancetre, r.id_cible AS id_descendant, 1 AS generation
    FROM Relations r
    WHERE r.id_cible = :id_individu AND r.type_relation LIKE 'parent_%'

    UNION ALL

    SELECT r.id_source, a.id_ancetre, a.generation + 1
    FROM Relations r
    JOIN Ancetres a ON r.id_cible = a.id_ancetre
    WHERE r.type_relation LIKE 'parent_%'
)
SELECT p.*, a.generation
FROM Ancetres a
JOIN Personnes p ON p.id = a.id_ancetre;
```

Comment j’ai vérifié que ça fonctionnait

J’ai inséré un jeu de données de test représentant plusieurs familles avec des relations variées (parents biologiques, enfants adoptés, conjoints). J’ai ensuite exécuté chaque requête pour m’assurer que les résultats étaient cohérents avec les liens définis dans la base.

Par exemple :

La requête des enfants renvoie bien tous les individus liés à un parent donné.

La requête des frères et sœurs exclut l’individu lui-même et ne retourne que ceux partageant au moins un parent.

La requête récursive des ancêtres permet de remonter plusieurs générations sans limite fixe.

J’ai également vérifié les performances et la lisibilité des résultats en les formatant en JSON pour une intégration future dans une API REST.

C3 – Agrégation & homogénéisation

Ce qu’il fallait faire

Les données extraites via OCR contiennent souvent des erreurs ou des formats incohérents. Par exemple, la quantité est parfois extraite sous la forme "12,00 U". Il fallait nettoyer ces données pour qu’elles soient exploitables (int, float, etc.) et supprimer les lignes corrompues.

Quelle solution technique j’ai choisie

J’ai mis en place un nettoyage des chaînes de caractères et des conversions de type (string → float/int) pour les quantités et les prix. J’ai aussi filtré les lignes où des champs essentiels comme le code produit ou la quantité étaient manquants.

Extraction et normalisation des prix

```
price_ht_match = re.search(r"(\d{1,3}[.,]\d{2})\s*€", lines[price_ht_index])
if price_ht_match:
    price_ht_value = float(price_ht_match.group(1).replace(',', '.'))
```

Ici, j'extrais les prix HT en euros, que je normalise en `float` pour stockage.

Une expression régulière (regex) est une séquence de caractères qui définit un motif de recherche, utilisée pour identifier, extraire ou remplacer des chaînes de texte selon des règles précises de correspondance.

Explication de la REGEX

`\d{1,3}` : correspond à 1 à 3 chiffres (ex. : 9, 99, 999).

`[,.]` : accepte soit une virgule ou un point comme séparateur décimal.

`\d{2}` : exactement 2 chiffres après le séparateur (ex. : 99).

`(\d{1,3}[,.]\d{2})` : capture le nombre entier + décimal (ex. : `12,50` ou `999.99`).

`\s*` : accepte zéro ou plusieurs espaces après le nombre.

`€` : attend le symbole euro juste après.

En résumé, cette regex détecte un prix au format européen ou international suivi du symbole €, comme `12,99 €` ou `999.99€`.

Calcul du montant TTC et de la TVA

```
tva_ratio = tvaeuroGlobal / total_ht_global
price_ttc_matches = [ht * (1 + tva_ratio) for ht in price_ht_matches]
```

Je calcule la TVA à partir du montant HT et du taux global, et je génère les prix TTC automatiquement.

Comment j'ai vérifié que ça fonctionnait

Après chaque extraction, j'affiche le dictionnaire JSON final. J'ai aussi connecté le front Angular à l'API pour vérifier que les champs du formulaire se remplissaient bien. Si un champ était vide ou incohérent, j'allais vérifier dans le PDF pour ajuster le parseur.

C4 – Modélisation & stockage RGPD

Une structure claire en classes `Produit`, `Intervention`, `Adresse`

Une sortie JSON finale prête à être stockée pour

Modularité : Chaque classe (Produit, Intervention, Adresse) représente un concept métier distinct, ce qui rend le code plus lisible et maintenable.

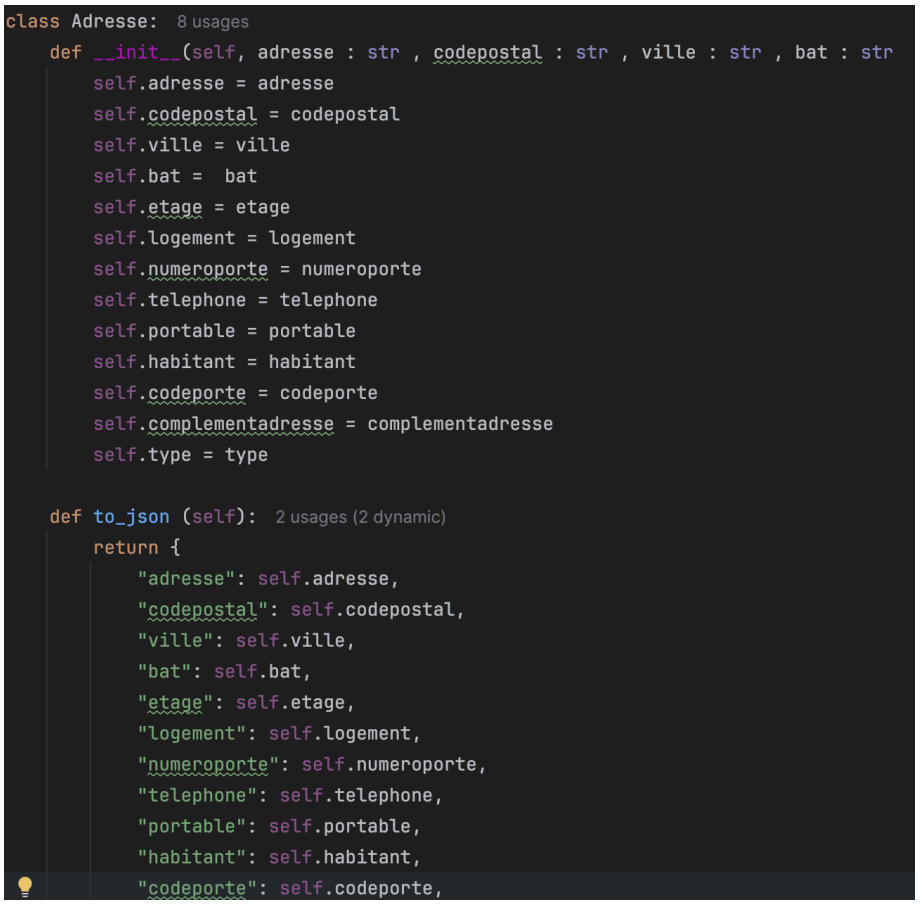
Réutilisabilité : Tu peux manipuler ces objets facilement dans d'autres parties du programme (API, traitement, affichage...).

Validation des données : Tu peux intégrer des règles de validation directement dans les classes (ex. : un prix ne peut pas être négatif).

Mappage d'un libellé texte vers un identifiant d'une base de données Dans cette partie, j'ai conçu une logique permettant de traduire une information textuelle ("parties communes") extraite du PDF en un identifiant numérique (`id`) correspondant à une valeur d'un dictionnaire présent en base.

```
if re.search(r"\bparties communes\b", text, re.IGNORECASE):
    type = 1
else:
    type = 2
```

Ce `type` est ensuite injecté dans mon objet `Intervention` :



```python
class Adresse:  8 usages
    def __init__(self, adresse : str , codepostal : str , ville : str , bat : str
        self.adresse = adresse
        self.codepostal = codepostal
        self.ville = ville
        self.bat =  bat
        self.etage = etage
        self.logement = logement
        self.numeroporte = numeroporte
        self.telephone = telephone
        self.portable = portable
        self.habitant = habitant
        self.codeporte = codeporte
        self.complementadresse = complementadresse
        self.type = type

    def to_json (self):  2 usages (2 dynamic)
        return {
            "adresse": self.adresse,
            "codepostal": self.codepostal,
            "ville": self.ville,
            "bat": self.bat,
            "etage": self.etage,
            "logement": self.logement,
            "numeroporte": self.numeroporte,
            "telephone": self.telephone,
            "portable": self.portable,
            "habitant": self.habitant,
            "codeporte": self.codeporte,
```

Lors de l'analyse des documents, j'ai automatisé la correspondance entre des informations textuelles issues du document PDF et des identifiants métiers présents en base de données. Par exemple, la mention "parties communes" est automatiquement associée à l'`id` numérique 1, utilisé dans la table `types_intervention`. Cela garantit l'uniformité des données stockées, facilite les jointures SQL, et permet de sécuriser l'injection en base (évite d'insérer du texte libre potentiellement erroné). Cette correspondance permet également une compatibilité avec des applications tierces ou une interface d'administration.

C5 – API REST

Une fois les données extraites et nettoyées, il fallait les mettre à disposition d'une interface front ou d'autres services. L'objectif était de proposer un

point d’entrée REST permettant de recevoir les fichiers à traiter et de renvoyer les données extraites.

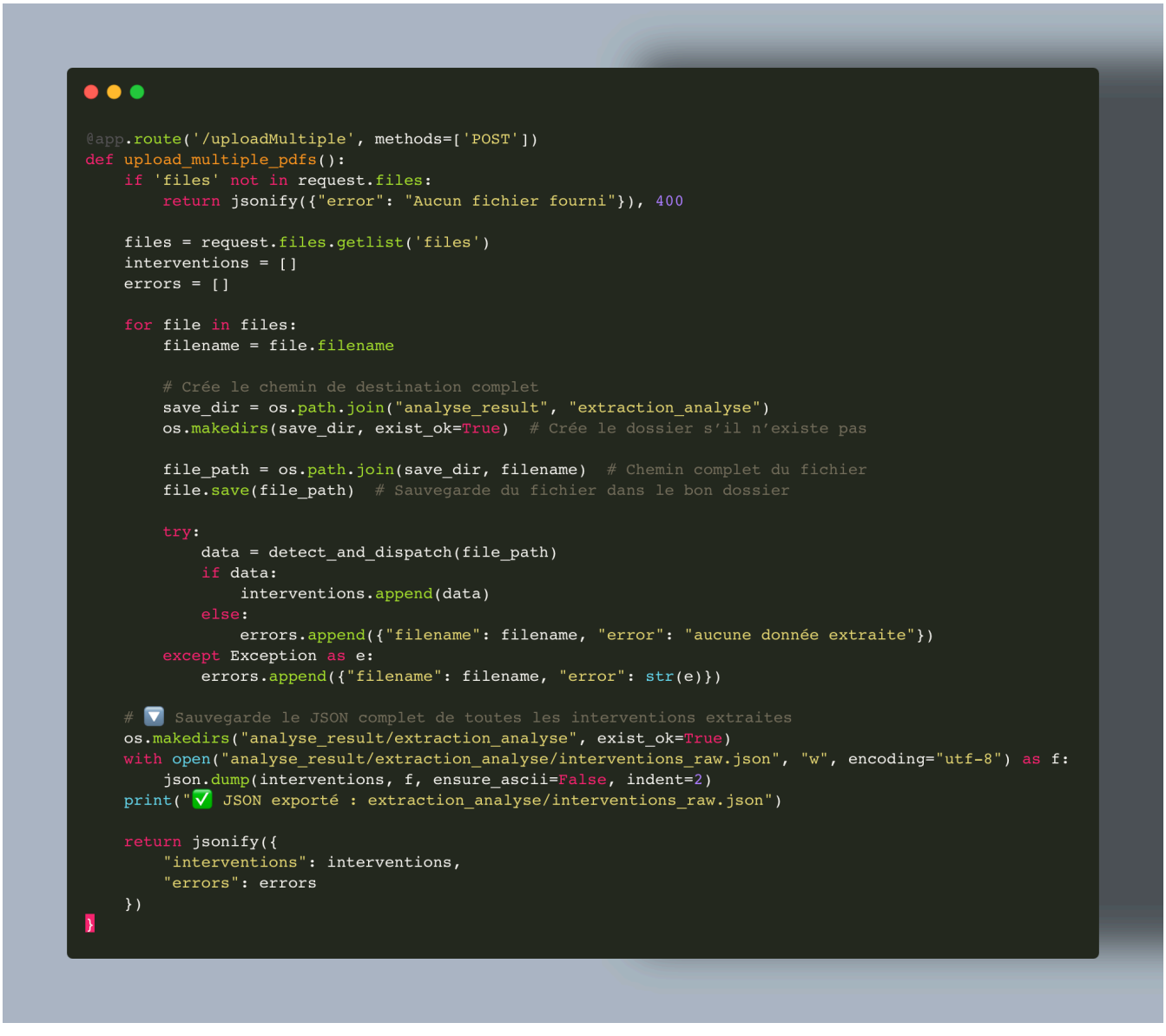
Quelle solution technique j’ai choisie

J’ai créé une API REST avec Flask. Elle expose un endpoint `/uploadMultiple`, qui accepte plusieurs fichiers PDF en POST, les traite et renvoie un JSON avec les données ou les erreurs.

Comment je l’ai implémentée

Création d’un endpoint Flask pour recevoir des fichiers

```python
@app.route('/uploadMultiple', methods=['POST'])
def upload_multiple_pdfs():
    files = request.files.getlist('files')
    for file in files:
        data = detect_and_dispatch(file_path)
        interventions.append(data)
```



```python
@app.route('/uploadMultiple', methods=['POST'])
def upload_multiple_pdfs():
    if 'files' not in request.files:
        return jsonify({"error": "Aucun fichier fourni"}), 400

    files = request.files.getlist('files')
    interventions = []
    errors = []

    for file in files:
        filename = file.filename

        # Crée le chemin de destination complet
        save_dir = os.path.join("analyse_result", "extraction_analyse")
        os.makedirs(save_dir, exist_ok=True)  # Crée le dossier s'il n'existe pas

        file_path = os.path.join(save_dir, filename)  # Chemin complet du fichier
        file.save(file_path)  # Sauvegarde du fichier dans le bon dossier

        try:
            data = detect_and_dispatch(file_path)
            if data:
                interventions.append(data)
            else:
                errors.append({"filename": filename, "error": "aucune donnée extraite"})
        except Exception as e:
            errors.append({"filename": filename, "error": str(e)})

    # 🔽 Sauvegarde le JSON complet de toutes les interventions extraites
    os.makedirs("analyse_result/extraction_analyse", exist_ok=True)
    with open("analyse_result/extraction_analyse/interventions_raw.json", "w", encoding="utf-8") as f:
        json.dump(interventions, f, ensure_ascii=False, indent=2)
    print("✅ JSON exporté : extraction_analyse/interventions_raw.json")

    return jsonify({
        "interventions": interventions,
        "errors": errors
    })
}
```

Renvoi des `interventions` au format JSON structuré :

```python
class Intervention:  9 usages
    def __init__(self , infoClient : str , engagement_no : str  , contactName : str , adresses
        self.infoClient = infoClient
        self.engagement_no = engagement_no
        self.contactName = contactName
        self.adresses = adresses
        self.produits = produits
        self.commentaire = commentaire
        self.contactPortable = contactPortable
        self.contactTelephone = contactTelephone
        self.totalTtcGlobal = totalTtcGlobal
        self.tvaeuroGlobal = tvaeuroGlobal


    def to_json (self):  6 usages (2 dynamic)
        result = {
            "infoClient" : self.infoClient,
            "engagement_no": self.engagement_no,
            "contactName":  self.contactName,
            "adresses": [],
            "produits": [],
            "commentaire": self.commentaire,
            "contactPortable": self.contactPortable,
            "contactTelephone": self.contactTelephone,
            "totalTtcGlobal" : self.totalTtcGlobal,
            "tvaeuroGlobal": self.tvaeuroGlobal
        }
        for adresse in self.adresses :
            result["adresses"].append(adresse.to_json())
```

```python
class Produit:  8 usages
    def __init__(self, code : str , designation : str ,commentaire : str ,  qte : float , prix : float
        self.code = code
        self.designation = designation
        self.commentaire = commentaire
        self.qte = qte
        self.prix = prix
        self.totalHt = totalHt
        self.totalTtc = totalTtc
        self.tva = tva
        self.tvaeuro = tvaeuro

    def to_json(self):  2 usages (2 dynamic)
        return {
            "code": self.code,
            "designation": self.designation,
            "commentaire": self.commentaire,
            "qte": self.qte,
            "prix": self.prix,
            "totalHt": self.totalHt,
            "totalTtc": self.totalTtc,
            "tva": self.tva,
            "tvaeuro" : self.tvaeuro,

        }
```

```python
class Adresse:  8 usages
    def __init__(self, adresse : str , codepostal : str , ville : str , bat : str
        self.adresse = adresse
        self.codepostal = codepostal
        self.ville = ville
        self.bat =  bat
        self.etage = etage
        self.logement = logement
        self.numeroporte = numeroporte
        self.telephone = telephone
        self.portable = portable
        self.habitant = habitant
        self.codeporte = codeporte
        self.complementadresse = complementadresse
        self.type = type

    def to_json (self):  2 usages (2 dynamic)
        return {
            "adresse": self.adresse,
            "codepostal": self.codepostal,
            "ville": self.ville,
            "bat": self.bat,
            "etage": self.etage,
            "logement": self.logement,
            "numeroporte": self.numeroporte,
            "telephone": self.telephone,
            "portable": self.portable,
            "habitant": self.habitant,
            "codeporte": self.codeporte,
```


```
  "adresses": [
    {
      "adresse": "22 rue du Hameau",
      "bat": "",
      "codeporte": "",
      "codepostal": "75015",
      "complementadresse": "",
      "etage": "3",
      "habitant": "MME LAMHAMAH SAMIRA",
      "logement": "153623",
      "numeroporte": "0007",
      "portable": "0640708439",
      "telephone": "0640708439",
      "type": 2,
      "ville": "PARIS"
    }
  ],
  "commentaire": [
    "La porte-fenêtre ne ferme pas. Contact du locataire : 06.25.16.
  ],
  "contactName": "OUEDRAOGO Yessoufou",
  "contactPortable": "0640708439",
  "contactTelephone": "0640708439",
  "engagement_no": "S19957",
  "infoClient": "Paris habitat",
  "produits": [
    {
      "code": "ECME01",
      "commentaire": "La porte-fenêtre ne ferme pas. Contact du loca
      "designation": "PETITE INTERVENTION",
      "prix": 85.38,
      "qte": 1,
      "totalHt": 85.38,
      "totalTtc": 93.92,
      "tva": 10,
      "tvaeuro": 8.54
```


Cette API permet à l’ utilisateur de téléverser un ou plusieurs fichiers PDF, automatiquement analysés et renvoyés en JSON prêt à

stocker ou exploiter.

Json Avec fausse donnée teste d’ un bon de commande teste


```
[
  {
    "adresses": [
      {
        "adresse": "22 rue du Hameau",
        "bat": "",
        "codeporte": "",
        "codepostal": "75015",
        "complementadresse": "",
        "etage": "3",
        "habitant": "MME LAMHAMAH SAMIRA",
        "logement": "153623",
        "numeroporte": "0007",
        "portable": "0640708439",
        "telephone": "0640708439",
        "type": 2,
        "ville": "PARIS"
      }
    ],
    "commentaire": [
      "La porte-fenêtre ne ferme pas. Contact du locataire : 06.25.16.
    ],
    "contactName": "OUEDRAOGO Yessoufou",
    "contactPortable": "0640708439",
    "contactTelephone": "0640708439",
    "engagement_no": "S19957",
    "infoClient": "Paris habitat",
    "produits": [
      {
        "code": "ECME01",
        "commentaire": "La porte-fenêtre ne ferme pas. Contact du loca
        "designation": "PETITE INTERVENTION",
        "prix": 85.38,
        "qte": 1,
        "totalHt": 85.38,
        "totalTtc": 93.92,
        "tva": 10,
        "tvaeuro": 8.54
      }
    ],
    "totalTtcGlobal": 93.92,
    "tvaeuroGlobal": 8.54
  }
```