🎯 OBJECTIF DU PROJET

Créer une application web permettant d'automatiser l'extraction et la saisie de commandes clients à partir de PDFs, en réduisant les erreurs et en simplifiant le processus pour qu'une personne formée en 1-2h puisse l'utiliser.

Gains attendus :

- Temps de traitement : **30 secondes** vs 5-10 minutes en manuel
- Taux d'erreur : **< 1%** vs 5-10% en manuel
- Formation : **1-2 heures** pour être autonome
- Volume : **10-20 commandes/jour** traitées rapidement

🏗️ ARCHITECTURE TECHNIQUE

Stack technologique

Frontend :

- Vue.js 3 (Composition API)
- Tailwind CSS (styling simple et moderne)
- Axios (requêtes HTTP)

Backend :

- Python 3.10+
- FastAPI (API REST)
- SQLite (base de données locale)
- pdfplumber (extraction PDF → JSON)
- Mistral AI via API (extraction intelligente)

Principe de fonctionnement :

```
None
PDF → pdfplumber/Mistral → JSON structuré → Validation humaine
→ CSV export
```

Structure des dossiers

```
order-extractor/
├── backend/
│   ├── main.py                # FastAPI app
```

```
|   ├── database.py             # SQLite setup + models
|   ├── pdf_processor.py        # Extraction PDF
|   ├── ai_extractor.py         # Mistral AI integration
|   ├── backup_manager.py       # Sauvegardes automatiques
|   ├── cleanup_manager.py      # Nettoyage auto des vieilles
données
|   ├── requirements.txt
|   ├── .env                    # Configuration (clé API
Mistral)
|   └── data/
|       ├── database.db
|       ├── backups/            # Sauvegardes hebdomadaires
|       └── uploads/            # PDFs uploadés (nettoyés
après 7 jours)
├── frontend/
|   ├── src/
|   |   ├── components/
|   |   |   ├── CommandeValidation.vue
|   |   |   ├── ProfilClientCreation.vue
|   |   |   ├── ListeCommandes.vue
|   |   |   ├── GestionPrix.vue
|   |   |   └── ...
|   |   ├── views/
|   |   |   ├── Home.vue
|   |   |   ├── NouvelleCommande.vue
|   |   |   ├── Clients.vue
|   |   |   ├── Articles.vue
|   |   |   └── Statistiques.vue
|   |   ├── App.vue
|   |   └── main.js
|   ├── package.json
|   └── index.html
└── README.md
```

📊 STRUCTURE DE LA BASE DE DONNÉES

Principe de stockage

PERMANENT (gardé indéfiniment) :

- Clients et profils d'extraction
- Articles (référence, poids)
- Correspondances (client → produit)
- Prix (simple et dégressif)

TEMPORAIRE (nettoyé automatiquement) :

- Commandes complètes (supprimées après 7 jours)
- Logs (supprimés après 90 jours)

Pourquoi : Le logiciel est un **outil de saisie**, pas un ERP. Odoo/SAP stocke les commandes. On garde uniquement ce qui permet l'apprentissage automatique.

Tables SQL

sql

```sql
-- ============================================
-- TABLES PERMANENTES
-- ============================================

-- Clients et leurs profils d'extraction
CREATE TABLE clients (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    nom TEXT NOT NULL UNIQUE,
    profil_extraction TEXT, -- JSON avec règles d'extraction
    utilise_ai BOOLEAN DEFAULT 0,
    date_creation TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    derniere_commande TIMESTAMP
);

-- Articles de notre catalogue
CREATE TABLE articles (
    notre_reference TEXT PRIMARY KEY, -- ex: "110 007"
    designation TEXT,
    poids REAL, -- en kg
    date_creation TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    date_maj TIMESTAMP
);

-- Correspondances client/produit (CŒUR DU SYSTÈME)
```

```sql
CREATE TABLE correspondances (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    client_id INTEGER NOT NULL,
    ref_client TEXT NOT NULL, -- ex: "512052"
    designation_client TEXT, -- ex: "MARQUAGE CONE 750"
    notre_reference TEXT NOT NULL,
    nb_validations INTEGER DEFAULT 1, -- Compteur de confiance
    statut TEXT DEFAULT 'manuel', -- 'manuel', 'confirmé'
(après 5 utilisations)
    date_creation TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    derniere_utilisation TIMESTAMP,
    FOREIGN KEY (client_id) REFERENCES clients(id) ON DELETE
CASCADE,
    FOREIGN KEY (notre_reference) REFERENCES
articles(notre_reference),
    UNIQUE(client_id, ref_client)
);

-- Prix par client/produit
CREATE TABLE prix (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    client_id INTEGER NOT NULL,
    notre_reference TEXT NOT NULL,
    type_prix TEXT DEFAULT 'simple', -- 'simple' ou 'degressif'
    prix_unitaire REAL, -- Pour type simple
    paliers TEXT, -- JSON pour type dégressif
    date_creation TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    date_maj TIMESTAMP,
    FOREIGN KEY (client_id) REFERENCES clients(id) ON DELETE
CASCADE,
    FOREIGN KEY (notre_reference) REFERENCES
articles(notre_reference),
    UNIQUE(client_id, notre_reference)
);

-- Configuration globale
CREATE TABLE configuration (
    cle TEXT PRIMARY KEY,
    valeur TEXT,
```

```
    type TEXT, -- 'string', 'number', 'boolean'
    description TEXT
);

-- ============================================
-- TABLES TEMPORAIRES (nettoyées auto)
-- ============================================

-- Commandes (gardées 7 jours après export)
CREATE TABLE commandes (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    numero_commande TEXT NOT NULL,
    client_id INTEGER NOT NULL,
    date_commande DATE,
    date_livraison DATE,
    adresse_livraison TEXT,
    contact_livraison TEXT, -- ex: "MR FORIEN 0608009672"
    notes_client TEXT, -- Notes extraites du PDF
    montant_total REAL,
    poids_total REAL,
    statut TEXT DEFAULT 'brouillon', -- 'brouillon', 'validé',
'exporté'
    pdf_path TEXT,
    json_extract TEXT, -- JSON brut extrait du PDF (pour debug)
    date_creation TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    date_validation TIMESTAMP,
    date_export TIMESTAMP,
    FOREIGN KEY (client_id) REFERENCES clients(id)
);

-- Lignes de commandes (gardées 7 jours)
CREATE TABLE lignes_commandes (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    commande_id INTEGER NOT NULL,
    ref_client TEXT, -- Référence du client
    notre_reference TEXT NOT NULL,
    designation TEXT,
    quantite INTEGER NOT NULL,
    prix_unitaire REAL NOT NULL,
```

```
    ligne_total REAL NOT NULL,
    poids_unitaire REAL,
    poids_total REAL,
    FOREIGN KEY (commande_id) REFERENCES commandes(id) ON
DELETE CASCADE,
    FOREIGN KEY (notre_reference) REFERENCES
articles(notre_reference)
);

-- Logs (gardés 90 jours)
CREATE TABLE logs (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    timestamp TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    niveau TEXT, -- 'info', 'warning', 'error'
    action TEXT, -- 'commande_creee', 'prix_modifie', etc.
    entite_type TEXT, -- 'commande', 'client', 'prix'
    entite_id INTEGER,
    details TEXT, -- JSON avec détails
    erreur TEXT -- Message d'erreur si applicable
);

-- ============================================
-- INDEX POUR PERFORMANCE
-- ============================================

CREATE INDEX idx_correspondances_client ON
correspondances(client_id);
CREATE INDEX idx_correspondances_ref ON
correspondances(ref_client);
CREATE INDEX idx_prix_client ON prix(client_id);
CREATE INDEX idx_commandes_client ON commandes(client_id);
CREATE INDEX idx_commandes_statut ON commandes(statut);
CREATE INDEX idx_commandes_date ON commandes(date_creation);
CREATE INDEX idx_logs_timestamp ON logs(timestamp);

-- ============================================
-- CONFIGURATION PAR DÉFAUT
-- ============================================
```

```
INSERT INTO configuration (cle, valeur, type, description)
VALUES
('backup_enabled', 'true', 'boolean', 'Activer les sauvegardes
automatiques'),
('backup_jour', 'lundi', 'string', 'Jour de sauvegarde
hebdomadaire'),
('backup_heure', '02:00', 'string', 'Heure de sauvegarde'),
('backup_retention', '4', 'number', 'Nombre de sauvegardes à
conserver'),
('cleanup_commandes_jours', '7', 'number', 'Supprimer les
commandes après X jours'),
('cleanup_logs_jours', '90', 'number', 'Supprimer les logs
après X jours'),
('mistral_api_key', '', 'string', 'Clé API Mistral'),
('mistral_model', 'mistral-small-latest', 'string', 'Modèle
Mistral à utiliser'),
('extraction_method_default', 'ai', 'string', 'Méthode
extraction par défaut (ai/manual)'),
('alerte_doublon', 'true', 'boolean', 'Alerter en cas de
numéro de commande déjà traité'),
('seuil_variation_prix', '0.20', 'number', 'Seuil alerte
variation prix (0.20 = 20%)');
```

🔄 EXTRACTION PDF → JSON

Approche hybride (AI + règles)

Mode AI (par défaut, recommandé) :

python

```python
def extract_with_ai(pdf_path: str) -> dict:
    """
    Extraction intelligente avec Mistral AI
    """
    # 1. Convertir PDF en texte
    text = extract_text_from_pdf(pdf_path)
```

```
    # 2. Prompt structuré pour Mistral
    prompt = f"""
Analyse ce bon de commande et extrais les informations au
format JSON strict.

IMPORTANT : Réponds UNIQUEMENT avec un objet JSON valide, sans
texte avant ou après.

Format attendu :
{{
  "numero_commande": "...",
  "client": "...",
  "date_commande": "YYYY-MM-DD",
  "date_livraison": "YYYY-MM-DD",
  "adresse_livraison": "...",
  "contact_livraison": "...",
  "notes_client": "...",
  "articles": [
    {{
      "ref_client": "...",
      "designation": "...",
      "quantite": ...,
      "prix_unitaire": ...
    }}
  ]
}}

Règles :
- Dates au format ISO (YYYY-MM-DD)
- Nombres sans espace ni symbole (12.50 pas 12,50€)
- Si info absente, mettre null
- Extraire TOUTES les lignes du tableau articles

Texte du document :
{text}
"""

    # 3. Appel API Mistral
```

```
    response = mistral_client.chat(
        model="mistral-small-latest",
        messages=[{"role": "user", "content": prompt}],
        temperature=0.1  # Basse température = plus
déterministe
    )

    # 4. Parser la réponse
    response_text = response.choices[0].message.content

    # Nettoyer les markdown si présents
    response_text = response_text.strip()
    if response_text.startswith("```json"):
        response_text = response_text[7:]
    if response_text.endswith("```"):
        response_text = response_text[:-3]
    response_text = response_text.strip()

    # 5. Parser JSON
    return json.loads(response_text)
```

Mode Règles (fallback si AI échoue ou profil manuel) :

python

```python
def extract_with_rules(pdf_path: str, profile: dict) -> dict:
    """
    Extraction basée sur les règles du profil client
    """
    import pdfplumber
    import re

    pdf = pdfplumber.open(pdf_path)
    page = pdf.pages[0]
    text = page.extract_text()

    result = {}
```

```
    # Extraire numéro de commande
    if "numero_commande" in profile["fields"]:
        pattern =
profile["fields"]["numero_commande"]["pattern"]
        match = re.search(pattern, text)
        result["numero_commande"] = match.group(1) if match
else None

    # Extraire adresse (zone définie)
    if "adresse_livraison" in profile["fields"]:
        zone = profile["fields"]["adresse_livraison"]["zone"]
        bbox = tuple(zone)  # (x0, y0, x1, y1)
        cropped = page.crop(bbox)
        result["adresse_livraison"] =
cropped.extract_text().strip()

    # Extraire tableau
    if "tableau_articles" in profile["fields"]:
        table = page.extract_table()
        articles = []

        col_mapping =
profile["fields"]["tableau_articles"]["columns"]

        for row in table[1:]:  # Skip header
            if row and any(row):  # Ligne non vide
                article = {
                    "ref_client":
row[col_mapping["ref_client"]],
                    "designation":
row[col_mapping["designation"]],
                    "quantite":
parse_number(row[col_mapping["quantite"]]),
                    "prix_unitaire":
parse_number(row[col_mapping["prix_unitaire"]])
                }
                articles.append(article)

        result["articles"] = articles
```

```
    return result

def parse_number(value: str) -> float:
    """Convertir string en nombre (gérer , et .)"""
    if not value:
        return 0
    value = str(value).replace(" ", "").replace(",", ".")
    return float(re.sub(r'[^\d.]', '', value))
```

Détection automatique du client :

python

```python
def detect_client(text: str) -> Optional[int]:
    """
    Détecte le client en cherchant des patterns connus
    """
    clients = db.query(Client).all()

    for client in clients:
        # Chercher le nom du client dans le texte
        if client.nom.upper() in text.upper():
            return client.id

        # Chercher dans les adresses du profil
        profile = json.loads(client.profil_extraction or "{}")
        if "identifiers" in profile:
            for identifier in profile["identifiers"]:
                if identifier.upper() in text.upper():
                    return client.id

    return None
```

🔄 WORKFLOWS COMPLETS

Workflow 1 : Première utilisation (Setup initial)

Étape 1 : Import des articles (une seule fois)

python

Python

```python
# backend/import_articles.py
import pandas as pd
from database import Session, Article

def import_articles_from_excel(file_path: str):
    """Importe les articles depuis le fichier Excel"""
    df = pd.read_excel(file_path)

    session = Session()

    for _, row in df.iterrows():
        article = Article(
            notre_reference=str(row['Référence']).strip(),
            designation=str(row['Désignation']).strip(),
            poids=float(row['Poids']) if
pd.notna(row['Poids']) else None
        )

        session.merge(article)  # Insert or update

    session.commit()
    print(f"✓ {len(df)} articles importés")

# Utilisation
import_articles_from_excel("Liste___Articles.xlsx")
```

Interface :

```
┌─────────────────────────────────────────────────────┐
│  🚀 BIENVENUE - CONFIGURATION INITIALE              │
├─────────────────────────────────────────────────────┤
│                                                     │
│  Étape 1/2 : Import des articles                    │
│                                                     │
│  Glissez votre fichier Excel ici :                  │
```

```
|  (Liste___Articles.xlsx)                                  |
|                                                           |
|  [📎 Parcourir...]                                        |
|                                                           |
|  Format attendu :                                         |
|  - Colonne "Référence" : 110 007, 110 041, etc.           |
|  - Colonne "Désignation" : Description du produit         |
|  - Colonne "Poids" : Poids en kg                          |
|                                                           |
|  [Suivant →]                                              |
└───────────────────────────────────────────────────────────┘
```

Étape 2 : Configuration Mistral AI
```
┌───────────────────────────────────────────────────────────┐
|  Étape 2/2 : Configuration de l'IA                        |
├───────────────────────────────────────────────────────────┤
|                                                           |
|  Pour activer l'extraction automatique, entrez            |
|  votre clé API Mistral :                                  |
|                                                           |
|  Clé API : [_____________________________________]        |
|                                                           |
|  [Tester la connexion]                                    |
|                                                           |
|  ℹ️ Vous pouvez obtenir une clé sur :                     |
|  https://console.mistral.ai/                              |
|                                                           |
|  [Passer cette étape] [Sauvegarder et continuer]          |
└───────────────────────────────────────────────────────────┘
```

Workflow 2 : Création d'un profil client (1 fois par client)

Interface principale :
```
```

```
┌──────────────────────────────────────────────────────────────┐
│  👥 NOUVEAU CLIENT                                           │
├──────────────────────────────────────────────────────────────┤
│                                                              │
│  Nom du client : [Kelias_________________]                   │
│                                                              │
│  Méthode de configuration :                                  │
│  ⦿ Automatique avec IA (recommandé)                          │
│  ○ Manuelle                                                  │
│                                                              │
│  Uploadez 1-2 exemples de bons de commande :                 │
│                                                              │
│  📄 4500340455_EHS.pdf                  [×]                  │
│  [+ Ajouter un exemple]                                      │
│                                                              │
│  [Annuler] [Analyser →]                                      │
└──────────────────────────────────────────────────────────────┘
```

Analyse AI en cours :

```
┌──────────────────────────────────────────────────────────────┐
│  🤖 ANALYSE EN COURS...                                      │
├──────────────────────────────────────────────────────────────┤
│                                                              │
│  ⏳ Lecture du PDF...                  [████████] 100%       │
│  ⏳ Analyse par Mistral AI...          [██████░░]  75%       │
│  ⏳ Détection de la structure...       [████░░░░]  50%       │
│                                                              │
│  Temps estimé : 15 secondes                                  │
└──────────────────────────────────────────────────────────────┘
```

Résultat de l'analyse :

```
┌──────────────────────────────────────────────────────────────┐
│  ✓ PROFIL DÉTECTÉ AUTOMATIQUEMENT                            │
```

```
├──────────────────────────────────────────────────────────────────────
────┤
│
│
│  Vérifiez les informations détectées :
│
│
│
│  📌 Numéro de commande
│
│     Exemple trouvé : "4500340455"
│
│     Pattern détecté : "Commande N°:?\s*(\d+)"
│
│     ✓ Valide    [✏️ Modifier]
│
│
│
│  📍 Adresse de livraison
│
│     Exemple trouvé :
│
│     "ARD Meaux / Villenoy
│
│      1 rue des Raguins
│
│      77124 VILLENOY"
│
│     Zone détectée : Section "ADRESSE DE LIVRAISON"
│
│     ✓ Valide    [✏️ Modifier]
│
│
│
│  📊 Tableau des articles
│
│     Colonnes détectées :
│
```

- "Article" → Référence client

- "Désignation" → Description

- "Quantité" → Quantité commandée

- "Px unit." → Prix unitaire

✓ 2 articles extraits [✏️ Modifier]

📅 Dates

- Date commande : 23.10.2025

- Date livraison : 17.11.2025

✓ Valide [✏️ Modifier]

📞 Contact livraison

"RDV MR FORIEN 0608009672"

✓ Valide [✏️ Modifier]

📝 Notes

"ATTENTION, cette commande est liée à une commande client soumise à pénalité de retard."

✓ Valide [✏️ Modifier]

```
|
|
|  [← Retour] [Tester sur un autre PDF] [Sauvegarder]
|
└──────────────────────────────────────────────────────────────────────
─────┘
```

Backend - Sauvegarde du profil :

python

```python
# POST /api/clients
@app.post("/api/clients")
async def create_client(
    nom: str,
    pdf_samples: List[UploadFile],
    method: str = "ai"  # 'ai' ou 'manual'
):
    """Crée un nouveau profil client"""

    # 1. Créer le client
    client = Client(nom=nom, utilise_ai=(method == "ai"))
    db.add(client)
    db.commit()

    # 2. Analyser les PDFs exemples
    if method == "ai":
        profile = await analyze_pdfs_with_ai(pdf_samples)
    else:
        profile = {}  # À remplir manuellement

    # 3. Sauvegarder le profil
    client.profil_extraction = json.dumps(profile)
    db.commit()

    return {
        "status": "success",
        "client_id": client.id,
```

```
        "profile": profile
    }

async def analyze_pdfs_with_ai(pdf_files: List[UploadFile]) ->
dict:
    """Analyse les PDFs avec Mistral pour détecter la
structure"""

    # Analyser le premier PDF
    pdf_path = save_upload(pdf_files[0])
    extracted_data = extract_with_ai(pdf_path)

    # Construire le profil basé sur l'extraction
    profile = {
        "method": "ai",
        "fields": {
            "numero_commande": {
                "enabled": True,
                "example":
extracted_data.get("numero_commande")
            },
            "adresse_livraison": {
                "enabled": True,
                "example":
extracted_data.get("adresse_livraison")
            },
            "date_commande": {
                "enabled": True,
                "format": "DD.MM.YYYY"
            },
            "date_livraison": {
                "enabled": True,
                "format": "DD.MM.YYYY"
            },
            "contact_livraison": {
                "enabled": True,
                "example":
extracted_data.get("contact_livraison")
            },
```

```
            "notes_client": {
                "enabled": True
            },
            "articles": {
                "enabled": True,
                "columns": ["ref_client", "designation",
"quantite", "prix_unitaire"]
            }
        },
        "identifiers": [extracted_data.get("client", "")]  #
Pour détecter le client
    }

    return profile
```

Workflow 3 : Traitement d'une commande (quotidien)

Étape 1 : Upload du PDF
```
┌──────────────────────────────────────────────────────────────┐
│  📥 NOUVELLE COMMANDE                                        │
├──────────────────────────────────────────────────────────────┤
│                                                              │
│  Glissez un PDF ici ou cliquez pour parcourir               │
│                                                              │
│  ┌──────────────────────────────────────────────────────┐   │
│  │                                                      │   │
│  │                    📄                                │   │
│  │              Déposez votre                           │   │
│  │           bon de commande ici                        │   │
│  │                                                      │   │
│  │              [Parcourir...]                          │   │
│  │                                                      │   │
│  └──────────────────────────────────────────────────────┘   │
│                                                              │
│  📎 Commandes récentes :                                    │
```

```
|  - kelias_cmd_20251029.pdf (il y a 2h)                    |
|  - aximum_cmd_20251029.pdf (il y a 4h)                    |
|                                                            |
└────────────────────────────────────────────────────────────┘
```

Backend - Upload et extraction :

python

```python
# POST /api/commandes/upload
@app.post("/api/commandes/upload")
async def upload_commande(file: UploadFile):
    """Upload et extraction d'un PDF de commande"""

    try:
        # 1. Sauvegarder le PDF
        pdf_path = save_upload(file)

        # 2. Extraire le texte brut
        text = extract_text_from_pdf(pdf_path)

        # 3. Détecter le client
        client_id = detect_client(text)

        if not client_id:
            return {
                "status": "client_unknown",
                "message": "Client non détecté. Veuillez le
sélectionner manuellement.",
                "pdf_path": pdf_path
            }

        client = db.query(Client).get(client_id)

        # 4. Extraire les données selon le profil
        if client.utilise_ai:
            json_data = extract_with_ai(pdf_path)
        else:
```

```
            profile = json.loads(client.profil_extraction)
            json_data = extract_with_rules(pdf_path, profile)

        # 5. Vérifier si commande déjà traitée (alerte doublon)
        existing =
check_duplicate(json_data["numero_commande"], client_id)

        if existing:
            return {
                "status": "duplicate_warning",
                "existing_commande": existing,
                "new_data": json_data,
                "pdf_path": pdf_path
            }

        # 6. Créer la commande en base (statut brouillon)
        commande = Commande(
            numero_commande=json_data["numero_commande"],
            client_id=client_id,

date_commande=parse_date(json_data.get("date_commande")),

date_livraison=parse_date(json_data.get("date_livraison")),

adresse_livraison=json_data.get("adresse_livraison"),

contact_livraison=json_data.get("contact_livraison"),
            notes_client=json_data.get("notes_client"),
            statut="brouillon",
            pdf_path=pdf_path,
            json_extract=json.dumps(json_data)
        )
        db.add(commande)
        db.commit()

        # 7. Créer les lignes
        lignes = []
        for article_data in json_data.get("articles", []):
            ligne = LigneCommande(
```

```
                commande_id=commande.id,
                ref_client=article_data["ref_client"],
                designation=article_data["designation"],
                quantite=article_data["quantite"],
                prix_unitaire=article_data["prix_unitaire"]
            )
            db.add(ligne)
            lignes.append(ligne)

        db.commit()

        # 8. Pour chaque ligne, vérifier les correspondances
        for ligne in lignes:
            correspondance = check_correspondance(client_id,
ligne.ref_client)

            if correspondance:
                # Correspondance trouvée
                ligne.notre_reference =
correspondance.notre_reference

                # Incrémenter le compteur de validation
                correspondance.nb_validations += 1
                correspondance.derniere_utilisation =
datetime.now()

                # Passer en statut "confirmé" après 5
validations
                if correspondance.nb_validations >= 5:
                    correspondance.statut = "confirmé"

                # Charger infos article (poids)
                article =
db.query(Article).get(ligne.notre_reference)
                if article:
                    ligne.poids_unitaire = article.poids

                # Charger le prix
```

```
                prix = get_prix(client_id,
ligne.notre_reference, ligne.quantite)
                if prix:
                    ligne.prix_unitaire = prix

            # Calculer totaux
            ligne.ligne_total = ligne.quantite *
ligne.prix_unitaire
            ligne.poids_total = ligne.quantite *
(ligne.poids_unitaire or 0)

        db.commit()

        # 9. Calculer totaux commande
        commande.montant_total = sum(l.ligne_total for l in
lignes)
        commande.poids_total = sum(l.poids_total for l in
lignes)
        db.commit()

        # 10. Logger
        log_action("info", "commande_creee", "commande",
commande.id,
                   f"Commande {commande.numero_commande}
créée")

        return {
            "status": "success",
            "commande_id": commande.id,
            "client": client.nom,
            "redirect": f"/commandes/{commande.id}/validation"
        }

    except Exception as e:
        log_action("error", "upload_failed", None, None,
str(e))
        raise HTTPException(500, f"Erreur lors de l'extraction
: {str(e)}")
```

```python
def check_correspondance(client_id: int, ref_client: str):
    """Vérifie si une correspondance existe"""
    return db.query(Correspondance).filter(
        Correspondance.client_id == client_id,
        Correspondance.ref_client == ref_client
    ).first()

def get_prix(client_id: int, notre_ref: str, quantite: int) ->
Optional[float]:
    """Récupère le prix applicable selon la quantité"""
    prix = db.query(Prix).filter(
        Prix.client_id == client_id,
        Prix.notre_reference == notre_ref
    ).first()

    if not prix:
        return None

    if prix.type_prix == "simple":
        return prix.prix_unitaire

    # Prix dégressif
    paliers = json.loads(prix.paliers)
    for palier in paliers:
        min_qty = palier["min"]
        max_qty = palier["max"]

        if max_qty is None:  # Dernier palier
            if quantite >= min_qty:
                return palier["prix"]
        else:
            if min_qty <= quantite <= max_qty:
                return palier["prix"]

    return None

def check_duplicate(numero_commande: str, client_id: int):
    """Vérifie si la commande a déjà été traitée récemment"""
    today = datetime.now().date()
```

```
    existing = db.query(Commande).filter(
        Commande.numero_commande == numero_commande,
        Commande.client_id == client_id,
        Commande.date_creation >= today
    ).first()

    return existing
```

Étape 2 : Page de validation

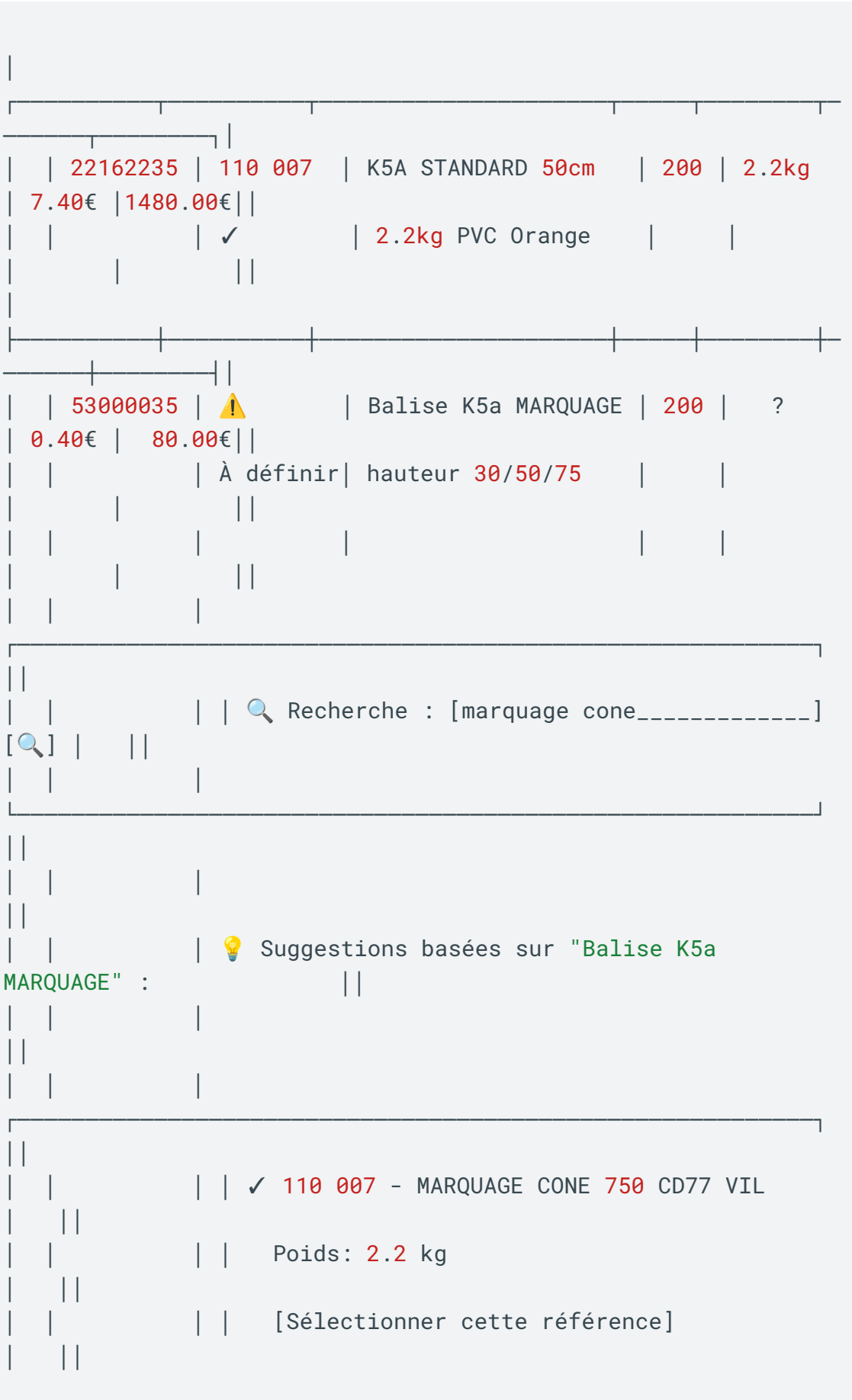
CAS A : Toutes les correspondances connues (100% auto)
```
┌──────────────────────────────────────────────────────────────────────
──────────────────────┐
│  ✓ COMMANDE PRÊTE À VALIDER
[Aide ?] [✕]     │
├──────────────────────────────────────────────────────────────────────
──────────────────────┤
│
│
│  📋 Commande n° 4500340455
│
│  👤 Client : Kelias
│
│  📅 Date commande : 23/10/2025      📅 Livraison :
17/11/2025 (dans 19 jours) │
│
│
│  📍 Adresse de livraison :                              [✏️]
│
│  ARD Meaux / Villenoy
│
│  1 rue des Raguins
│
│  77124 VILLENOY
│
│
│
```

| 📞 Contact : MR FORIEN 0608009672
|
|
|
|
┌──┐
| ⚠️ NOTES CLIENT
| |
| ATTENTION, cette commande est liée à une commande client soumise à |
| pénalité de retard. RDV obligatoire.
| |
|
└──┘
|
|
| 📦 ARTICLES
|
|

Réf. Client	Notre Réf.	Désignation	Qté	Poids Unit.	P.U.	Total
512052	110 007	MARQUAGE CONE 750 ✓ CD77 VIL	200	2.2kg	0.40€	80.00€
561151	110 041	CONE 750 K5A LESTE	200	5.2kg	11.51€	2302.00€

```
|  |          | ✓        | MONOB CLT2 4,2KG    |     |
|       |        ||
|
└──────────┴──────────┴─────────────────────┴────┴────────┴─
──────┴────────┘|
|
|
|  💰 MONTANT TOTAL : 2 382.00 €
|
|  ⚖️  POIDS TOTAL : 1 480 kg
|
|
|
|  [❌ Annuler]  [💾 Enregistrer en brouillon]  [✓ Valider la
commande]        |
└──────────────────────────────────────────────────────────────
──────────────────┘
```

CAS B : Nouvelle référence à mapper
```
┌──────────────────────────────────────────────────────────────
──────────────────┐
|  ⚠️ COMMANDE NÉCESSITE VALIDATION
[Aide ?] [✕]     |
├──────────────────────────────────────────────────────────────
──────────────────┤
|
|
|  📋 Commande n° CF25/03187
|
|  👤 Client : Aximum
|
|
|
|  📦 ARTICLES
|
```


22162235 | 110 007 | K5A STANDARD 50cm | 200 | 2.2kg | 7.40€ | 1480.00€
✓ | 2.2kg PVC Orange
53000035 | ⚠️ | Balise K5a MARQUAGE | 200 | ? | 0.40€ | 80.00€
À définir | hauteur 30/50/75
🔍 Recherche : [marquage cone_____] [🔍]
💡 Suggestions basées sur "Balise K5a MARQUAGE" :
✓ 110 007 - MARQUAGE CONE 750 CD77 VIL
Poids: 2.2 kg
[Sélectionner cette référence]

Backend - Recherche de suggestions :

python

```python
# GET /api/articles/search?q=marquage+cone
```

```python
@app.get("/api/articles/search")
async def search_articles(q: str, limit: int = 5):
    """Recherche d'articles par similarité de texte"""
    from difflib import SequenceMatcher

    query_lower = q.lower()
    articles = db.query(Article).all()

    # Calculer score de similarité pour chaque article
    scores = []
    for article in articles:
        designation_lower = article.designation.lower()

        # Score basé sur SequenceMatcher
        ratio = SequenceMatcher(None, query_lower,
designation_lower).ratio()

        # Bonus si contient les mots clés
        words = query_lower.split()
        word_matches = sum(1 for word in words if word in
designation_lower)
        bonus = word_matches * 0.1

        final_score = ratio + bonus
        scores.append((article, final_score))

    # Trier par score décroissant
    scores.sort(key=lambda x: x[1], reverse=True)

    # Retourner top N
    suggestions = []
    for article, score in scores[:limit]:
        if score > 0.3:  # Seuil minimum de pertinence
            suggestions.append({
                "notre_reference": article.notre_reference,
                "designation": article.designation,
                "poids": article.poids,
                "score": round(score, 2)
            })
```

```python
    return {"suggestions": suggestions}

# POST /api/correspondances
@app.post("/api/correspondances")
async def create_correspondance(
    client_id: int,
    ref_client: str,
    designation_client: str,
    notre_reference: str
):
    """Crée une nouvelle correspondance client/produit"""

    # Vérifier que l'article existe
    article = db.query(Article).get(notre_reference)
    if not article:
        raise HTTPException(404, "Article non trouvé")

    # Créer ou mettre à jour la correspondance
    corresp = db.query(Correspondance).filter(
        Correspondance.client_id == client_id,
        Correspondance.ref_client == ref_client
    ).first()

    if corresp:
        # Mise à jour
        corresp.notre_reference = notre_reference
        corresp.designation_client = designation_client
        corresp.nb_validations += 1
    else:
        # Création
        corresp = Correspondance(
            client_id=client_id,
            ref_client=ref_client,
            designation_client=designation_client,
            notre_reference=notre_reference,
            nb_validations=1,
            statut="manuel"
        )
```

```
        db.add(corresp)

    corresp.derniere_utilisation = datetime.now()
    db.commit()

    log_action("info", "correspondance_creee",
"correspondance", corresp.id,
               f"Nouvelle correspondance : {ref_client} →
{notre_reference}")

    return {
        "status": "success",
        "correspondance_id": corresp.id,
        "article": {
            "notre_reference": article.notre_reference,
            "designation": article.designation,
            "poids": article.poids
        }
    }
```

CAS C : Prix modifié
```
┌──────────────────────────────────────────────────────────────────
──────────────────────┐
|  | 512052   | 110 007  | MARQUAGE CONE 750   | 200 | 2.2kg
| 0.45€ |  90.00€||
|  |          | ✓        |                     |     |
| ⚠️ NEW|        ||
|  |          |          |                     |     |
|       |        ||
|  |          |
┌──────────────────────────────────────────────────────────────────┐
||
|  |          | | ⚠️ Le prix a changé
|   ||
|  |          | |
|   ||
```

```
|  |          | | Prix sauvegardé : 0.40€
|   ||
|  |          | | Prix dans le PDF : 0.45€ (+12.5%)
|   ||
|  |          | |
|   ||
|  |          | | Que voulez-vous faire ?
|   ||
|  |          | |
|   ||
|  |          | | ○ Utiliser 0.45€ pour cette commande
uniquement          |   ||
|  |          | | ⦿ Mettre à jour le prix définitivement à
0.45€           |   ||
|  |          | |   (toutes les futures commandes)
|   ||
|  |          | |
|   ||
|  |          | | [Valider mon choix]
|   ||
|  |          |
└──────────────────────────────────────────────────────────┘
||
└──────────────────────────────────────────────────────────────
_____________________┘
```

Backend - Mise à jour de prix :

python

Python

```python
# POST /api/prix
@app.post("/api/prix")
async def create_or_update_prix(
    client_id: int,
    notre_reference: str,
    prix_unitaire: float = None,
    type_prix: str = "simple",
    paliers: str = None  # JSON pour dégressif
```

```
):
    """Crée ou met à jour un prix"""

    prix = db.query(Prix).filter(
        Prix.client_id == client_id,
        Prix.notre_reference == notre_reference
    ).first()

    if prix:
        # Mise à jour
        if type_prix == "simple":
            ancien_prix = prix.prix_unitaire
            prix.prix_unitaire = prix_unitaire
            log_action("info", "prix_modifie", "prix",
prix.id,
                       f"Prix modifié : {ancien_prix}€ →
{prix_unitaire}€")
        else:
            prix.type_prix = type_prix
            prix.paliers = paliers
            log_action("info", "prix_modifie", "prix",
prix.id,
                       "Prix dégressif mis à jour")

        prix.date_maj = datetime.now()
    else:
        # Création
        prix = Prix(
            client_id=client_id,
            notre_reference=notre_reference,
            type_prix=type_prix,
            prix_unitaire=prix_unitaire,
            paliers=paliers
        )
        db.add(prix)
        log_action("info", "prix_cree", "prix", None,
                   f"Nouveau prix créé : {prix_unitaire}€")

    db.commit()
```

```
    return {"status": "success", "prix_id": prix.id}
```

Étape 3 : Validation finale

python

```python
# PUT /api/commandes/{id}/valider
@app.put("/api/commandes/{commande_id}/valider")
async def valider_commande(commande_id: int):
    """Valide définitivement la commande"""

    commande = db.query(Commande).get(commande_id)
    if not commande:
        raise HTTPException(404, "Commande non trouvée")

    # Vérifier que toutes les lignes ont une référence
    lignes = db.query(LigneCommande).filter(
        LigneCommande.commande_id == commande_id
    ).all()

    for ligne in lignes:
        if not ligne.notre_reference:
            raise HTTPException(400, "Toutes les lignes
doivent avoir une référence")

    # Passer en statut validé
    commande.statut = "validé"
    commande.date_validation = datetime.now()
    db.commit()

    log_action("info", "commande_validee", "commande",
commande_id,
               f"Commande {commande.numero_commande} validée")

    return {
        "status": "success",
        "message": "Commande validée avec succès",
```

```
        "commande_id": commande_id
    }
```

Confirmation :
```
┌──────────────────────────────────────────────────────────────┐
│  ✓ COMMANDE VALIDÉE AVEC SUCCÈS                              │
├──────────────────────────────────────────────────────────────┤
│                                                              │
│  Commande n° 4500340455                                      │
│  Client : Kelias                                             │
│  Montant : 2 382.00 €                                        │
│                                                              │
│  📊 Statistiques :                                           │
│  - 2 articles traités                                        │
│  - 2 correspondances validées                                │
│  - Temps de traitement : 35 secondes                         │
│                                                              │
│  Que voulez-vous faire ?                                     │
│                                                              │
│  [📥 Exporter en CSV pour Odoo]                              │
│  [👁️ Voir la commande]                                       │
│  [📨 Traiter une nouvelle commande]                          │
│  [🏠 Retour à l'accueil]                                     │
│                                                              │
└──────────────────────────────────────────────────────────────┘
```

Workflow 4 : Export CSV

python

Python
```python
# GET /api/commandes/{id}/export-csv
@app.get("/api/commandes/{commande_id}/export-csv")
async def export_commande_csv(commande_id: int):
```

```
    """Exporte la commande en CSV pour Odoo"""

    commande = db.query(Commande).get(commande_id)
    if not commande:
        raise HTTPException(404, "Commande non trouvée")

    lignes = db.query(LigneCommande).filter(
        LigneCommande.commande_id == commande_id
    ).all()

    # Générer CSV
    output = StringIO()
    writer = csv.writer(output)

    # Header
    writer.writerow([
        'numero_commande', 'client', 'date_commande',
'date_livraison',
        'adresse_livraison', 'contact_livraison',
'notes_client',
        'reference', 'designation', 'quantite',
'prix_unitaire',
        'poids_unitaire', 'total_ligne', 'poids_total'
    ])

    # Lignes
    client = commande.client
    for ligne in lignes:
        writer.writerow([
            commande.numero_commande,
            client.nom,
            commande.date_commande.isoformat() if
commande.date_commande else "",
            commande.date_livraison.isoformat() if
commande.date_livraison else "",
            commande.adresse_livraison or "",
            commande.contact_livraison or "",
            commande.notes_client or "",
            ligne.notre_reference,
```

```
            ligne.designation,
            ligne.quantite,
            ligne.prix_unitaire,
            ligne.poids_unitaire or 0,
            ligne.ligne_total,
            ligne.poids_total or 0
        ])

    # Marquer comme exporté
    commande.statut = "exporté"
    commande.date_export = datetime.now()
    db.commit()

    log_action("info", "commande_exportee", "commande",
commande_id,
              f"Commande {commande.numero_commande} exportée
en CSV")

    output.seek(0)

    return Response(
        content=output.getvalue(),
        media_type="text/csv",
        headers={
            "Content-Disposition": f"attachment;
filename=commande_{commande.numero_commande}.csv"
        }
    )
```

🔧 GESTION DES PRIX DÉGRESSIFS

Interface de création/modification

vue

```
None
<template>
  <div class="prix-degressif-editor">
```

```
    <h3>Prix pour {{ client.nom }} - {{ article.designation
}}</h3>

    <div class="type-selector mb-4">
      <label class="mr-4">
        <input type="radio" v-model="typePrix" value="simple">
Prix simple
      </label>
      <label>
        <input type="radio" v-model="typePrix"
value="degressif"> Prix dégressif
      </label>
    </div>

    <!-- Prix simple -->
    <div v-if="typePrix === 'simple'" class="prix-simple">
      <label>Prix unitaire :</label>
      <input type="number" step="0.01"
v-model.number="prixSimple" class="ml-2">
      <span class="ml-1">€</span>
    </div>

    <!-- Prix dégressif -->
    <div v-else class="prix-degressif">
      <table class="w-full border">
        <thead>
          <tr class="bg-gray-100">
            <th>De (quantité)</th>
            <th>À (quantité)</th>
            <th>Prix unitaire</th>
            <th></th>
          </tr>
        </thead>
        <tbody>
          <tr v-for="(palier, index) in paliers" :key="index">
            <td>
              <input type="number" v-model.number="palier.min"
readonly class="w-20">
            </td>
```

```
            <td>
              <input
                type="number"
                v-model.number="palier.max"
                :placeholder="index === paliers.length - 1 ?
'∞' : ''"
                class="w-20"
              >
            </td>
            <td>
              <input type="number" step="0.01"
v-model.number="palier.prix" class="w-24">
              <span class="ml-1">€</span>
            </td>
            <td>
              <button
                v-if="paliers.length > 1"
                @click="supprimerPalier(index)"
                class="btn-danger-small"
              >
                ×
              </button>
            </td>
          </tr>
        </tbody>
      </table>

      <button @click="ajouterPalier" class="btn-secondary
mt-2">
        + Ajouter un palier
      </button>

      <!-- Aperçu -->
      <div class="apercu mt-4 p-4 bg-blue-50 rounded">
        <h4>💡 Aperçu des prix :</h4>
        <ul>
          <li v-for="(exemple, idx) in exemplesCalcules"
:key="idx">
```

```
            {{ exemple.quantite }} unités → {{
exemple.prix_unitaire }}€/u
            ({{ exemple.total }}€ total)
          </li>
        </ul>
      </div>
    </div>

    <div class="actions mt-4">
      <button @click="annuler"
class="btn-secondary">Annuler</button>
      <button @click="sauvegarder" class="btn-primary
ml-2">Sauvegarder</button>
    </div>
  </div>
</template>

<script setup>
import { ref, computed, watch } from 'vue'
import axios from 'axios'

const props = defineProps(['clientId', 'article',
'prixExistant'])

const typePrix = ref(props.prixExistant?.type_prix ||
'simple')
const prixSimple = ref(props.prixExistant?.prix_unitaire || 0)
const paliers = ref(props.prixExistant?.paliers ?
JSON.parse(props.prixExistant.paliers) : [
  { min: 1, max: 10, prix: 0 }
])

function ajouterPalier() {
  const dernierPalier = paliers.value[paliers.value.length -
1]
  const nouveauMin = (dernierPalier.max || 0) + 1

  paliers.value.push({
    min: nouveauMin,
```

```
    max: nouveauMin + 50,
    prix: 0
  })
}

function supprimerPalier(index) {
  paliers.value.splice(index, 1)

  // Recalculer les min des paliers suivants
  for (let i = index; i < paliers.value.length; i++) {
    if (i > 0) {
      paliers.value[i].min = (paliers.value[i-1].max || 0) + 1
    }
  }
}

const exemplesCalcules = computed(() => {
  if (typePrix.value === 'simple') {
    return [
      { quantite: 10, prix_unitaire: prixSimple.value, total:
(10 * prixSimple.value).toFixed(2) },
      { quantite: 50, prix_unitaire: prixSimple.value, total:
(50 * prixSimple.value).toFixed(2) },
      { quantite: 200, prix_unitaire: prixSimple.value, total:
(200 * prixSimple.value).toFixed(2) }
    ]
  } else {
    const exemples = [10, 25, 100, 300]
    return exemples.map(qty => {
      const prix = calculerPrixPourQuantite(qty)
      return {
        quantite: qty,
        prix_unitaire: prix.toFixed(2),
        total: (qty * prix).toFixed(2)
      }
    })
  }
})
```

```
function calculerPrixPourQuantite(quantite) {
  for (const palier of paliers.value) {
    if (palier.max === null && quantite >= palier.min) {
      return palier.prix
    }
    if (quantite >= palier.min && quantite <= palier.max) {
      return palier.prix
    }
  }
  return 0
}

async function sauvegarder() {
  const data = {
    client_id: props.clientId,
    notre_reference: props.article.notre_reference,
    type_prix: typePrix.value
  }

  if (typePrix.value === 'simple') {
    data.prix_unitaire = prixSimple.value
  } else {
    data.paliers = JSON.stringify(paliers.value)
  }

  await axios.post('/api/prix', data)
  alert('Prix sauvegardé avec succès !')
}

function annuler() {
  // Retour
}
</script>
```

🔄 TÂCHES AUTOMATIQUES (Cron Jobs)

python

Python

```python
# backend/tasks.py
import schedule
import time
from threading import Thread
from datetime import datetime, timedelta
import shutil
import os

def backup_database():
    """Sauvegarde hebdomadaire de la base de données"""
    config = get_config()

    if config.get('backup_enabled') != 'true':
        return

    timestamp = datetime.now().strftime("%Y%m%d_%H%M%S")
    backup_dir = "data/backups"

    os.makedirs(backup_dir, exist_ok=True)

    # Copier database.db
    shutil.copy(
        "data/database.db",
        f"{backup_dir}/database_{timestamp}.db"
    )

    # Nettoyer les vieux backups (garder seulement les N
derniers)
    retention = int(config.get('backup_retention', 4))
    cleanup_old_backups(backup_dir, retention)

    log_action("info", "backup_created", None, None,
               f"Backup créé : database_{timestamp}.db")

def cleanup_old_backups(backup_dir: str, keep: int):
    """Supprime les vieux backups"""
    backups = sorted([f for f in os.listdir(backup_dir) if
f.startswith("database_")])
```

```
    if len(backups) > keep:
        to_delete = backups[:-keep]
        for backup in to_delete:
            os.remove(os.path.join(backup_dir, backup))

def cleanup_old_commandes():
    """Supprime les commandes exportées depuis > 7 jours"""
    config = get_config()
    jours = int(config.get('cleanup_commandes_jours', 7))

    cutoff_date = datetime.now() - timedelta(days=jours)

    # Supprimer les commandes exportées anciennes
    deleted = db.query(Commande).filter(
        Commande.statut == "exporté",
        Commande.date_export < cutoff_date
    ).delete()

    db.commit()

    if deleted > 0:
        log_action("info", "cleanup_commandes", None, None,
                   f"{deleted} commandes supprimées")

def cleanup_old_logs():
    """Supprime les logs > 90 jours"""
    config = get_config()
    jours = int(config.get('cleanup_logs_jours', 90))

    cutoff_date = datetime.now() - timedelta(days=jours)

    deleted = db.query(Log).filter(
        Log.timestamp < cutoff_date
    ).delete()

    db.commit()

    if deleted > 0:
        log_action("info", "cleanup_logs", None, None,
```

```
                f"{deleted} logs supprimés")

def cleanup_old_uploads():
    """Supprime les PDFs uploadés > 7 jours"""
    upload_dir = "data/uploads"
    cutoff_date = datetime.now() - timedelta(days=7)

    deleted = 0
    for filename in os.listdir(upload_dir):
        filepath = os.path.join(upload_dir, filename)
        file_time =
datetime.fromtimestamp(os.path.getmtime(filepath))

        if file_time < cutoff_date:
            os.remove(filepath)
            deleted += 1

    if deleted > 0:
        log_action("info", "cleanup_uploads", None, None,
                f"{deleted} PDFs supprimés")

def run_scheduled_tasks():
    """Lance les tâches planifiées en arrière-plan"""
    config = get_config()

    # Backup hebdomadaire
    jour = config.get('backup_jour', 'lundi')
    heure = config.get('backup_heure', '02:00')

    jour_mapping = {
        'lundi': schedule.every().monday,
        'mardi': schedule.every().tuesday,
        'mercredi': schedule.every().wednesday,
        'jeudi': schedule.every().thursday,
        'vendredi': schedule.every().friday,
        'samedi': schedule.every().saturday,
        'dimanche': schedule.every().sunday
    }
```

```python
    jour_mapping[jour].at(heure).do(backup_database)

    # Nettoyages quotidiens à 3h du matin
    schedule.every().day.at("03:00").do(cleanup_old_commandes)
    schedule.every().day.at("03:00").do(cleanup_old_logs)
    schedule.every().day.at("03:00").do(cleanup_old_uploads)

    while True:
        schedule.run_pending()
        time.sleep(60)  # Vérifier chaque minute

# Lancer dans un thread séparé
def start_background_tasks():
    thread = Thread(target=run_scheduled_tasks, daemon=True)
    thread.start()

# Dans main.py
@app.on_event("startup")
async def startup_event():
    start_background_tasks()
```

📊 STATISTIQUES

python

```python
# GET /api/statistiques
@app.get("/api/statistiques")
async def get_statistiques(
    periode: str = "mois",  # 'jour', 'semaine', 'mois',
'annee'
    client_id: Optional[int] = None
):
    """Génère des statistiques sur l'activité"""

    # Définir la période
    now = datetime.now()
    if periode == "jour":
```

```
        start_date = now.replace(hour=0, minute=0, second=0)
    elif periode == "semaine":
        start_date = now - timedelta(days=now.weekday())
    elif periode == "mois":
        start_date = now.replace(day=1, hour=0, minute=0,
second=0)
    else:  # annee
        start_date = now.replace(month=1, day=1, hour=0,
minute=0, second=0)

    # Query de base
    query = db.query(Commande).filter(
        Commande.date_creation >= start_date,
        Commande.statut.in_(["validé", "exporté"])
    )

    if client_id:
        query = query.filter(Commande.client_id == client_id)

    commandes = query.all()

    # Calculs
    nb_commandes = len(commandes)
    ca_total = sum(c.montant_total or 0 for c in commandes)
    poids_total = sum(c.poids_total or 0 for c in commandes)

    # Temps moyen de traitement
    temps_traitement = []
    for c in commandes:
        if c.date_creation and c.date_validation:
            delta = (c.date_validation -
c.date_creation).total_seconds()
            temps_traitement.append(delta)

    temps_moyen = sum(temps_traitement) /
len(temps_traitement) if temps_traitement else 0

    # Par client
    par_client = {}
```

```
    for c in commandes:
        client_nom = c.client.nom
        if client_nom not in par_client:
            par_client[client_nom] = {
                "nb_commandes": 0,
                "ca": 0,
                "auto_validations": 0
            }

        par_client[client_nom]["nb_commandes"] += 1
        par_client[client_nom]["ca"] += c.montant_total or 0

        # Calculer taux d'auto-validation
        lignes = db.query(LigneCommande).filter(
            LigneCommande.commande_id == c.id
        ).all()

        nb_lignes = len(lignes)
        nb_auto = sum(1 for l in lignes if
has_correspondance(c.client_id, l.ref_client))

        if nb_lignes > 0:
            taux = (nb_auto / nb_lignes) * 100
            par_client[client_nom]["auto_validations"] += taux

    # Moyenner les taux
    for client in par_client.values():
        if client["nb_commandes"] > 0:
            client["taux_auto"] = client["auto_validations"] /
client["nb_commandes"]
        else:
            client["taux_auto"] = 0

    # Produits les plus commandés
    produits = {}
    for c in commandes:
        lignes = db.query(LigneCommande).filter(
            LigneCommande.commande_id == c.id
        ).all()
```

```
        for ligne in lignes:
            ref = ligne.notre_reference
            if ref not in produits:
                article = db.query(Article).get(ref)
                produits[ref] = {
                    "reference": ref,
                    "designation": article.designation if
article else "",
                    "quantite_totale": 0
                }

            produits[ref]["quantite_totale"] += ligne.quantite

    top_produits = sorted(
        produits.values(),
        key=lambda x: x["quantite_totale"],
        reverse=True
    )[:10]

    return {
        "periode": periode,
        "start_date": start_date.isoformat(),
        "nb_commandes": nb_commandes,
        "ca_total": round(ca_total, 2),
        "poids_total": round(poids_total, 2),
        "temps_moyen_traitement": round(temps_moyen, 0),  # en
secondes
        "par_client": par_client,
        "top_produits": top_produits
    }

def has_correspondance(client_id: int, ref_client: str) ->
bool:
    """Vérifie si une correspondance existe pour cette réf
client"""
    return db.query(Correspondance).filter(
        Correspondance.client_id == client_id,
        Correspondance.ref_client == ref_client
```

```
    ).first() is not None
```

🚀 INSTALLATION ET DÉMARRAGE

Backend

bash

```shell
# 1. Créer l'environnement virtuel
cd backend
python -m venv venv
source venv/bin/activate  # Windows: venv\Scripts\activate

# 2. Installer les dépendances
pip install -r requirements.txt

# 3. Créer le fichier .env
cat > .env << EOF
MISTRAL_API_KEY=votre_cle_api_ici
MISTRAL_MODEL=mistral-small-latest
DATABASE_URL=sqlite:///data/database.db
EOF

# 4. Initialiser la base de données
python -c "from database import init_db; init_db()"

# 5. Importer les articles
python import_articles.py ../Liste___Articles.xlsx

# 6. Lancer le serveur
uvicorn main:app --reload --port 8000
```

Frontend

bash

```shell
# 1. Installer les dépendances
```

```
cd frontend
npm install

# 2. Lancer le serveur de développement
npm run dev

# 3. Ouvrir dans le navigateur
# http://localhost:5173
```

requirements.txt
```
fastapi==0.104.1
uvicorn[standard]==0.24.0
pdfplumber==0.10.3
python-multipart==0.0.6
sqlalchemy==2.0.23
mistralai==0.0.11
python-dotenv==1.0.0
pandas==2.1.3
openpyxl==3.1.2
schedule==1.2.0
python-dateutil==2.8.2
```

package.json

json

JSON
```
{
  "name": "order-extractor-frontend",
  "version": "1.0.0",
  "scripts": {
    "dev": "vite",
    "build": "vite build",
    "preview": "vite preview"
  },
  "dependencies": {
    "vue": "^3.3.8",
```

```
    "vue-router": "^4.2.5",
    "axios": "^1.6.2"
  },
  "devDependencies": {
    "@vitejs/plugin-vue": "^4.5.0",
    "autoprefixer": "^10.4.16",
    "postcss": "^8.4.32",
    "tailwindcss": "^3.3.6",
    "vite": "^5.0.4"
  }
}
```

📅 PLAN DE DÉVELOPPEMENT (4 sprints)

Sprint 1 (Semaine 1) - MVP Core

Backend :

- Setup FastAPI + SQLite
- Créer toutes les tables
- Script d'import articles Excel
- Extraction PDF basique (pdfplumber)
- API endpoints CRUD de base

Frontend :

- Setup Vue 3 + Tailwind + Router
- Page upload PDF
- Page validation commande (version simple)
- Gestion correspondances manuelles

Livrable : Upload PDF → Extraction manuelle → Validation → CSV

Sprint 2 (Semaine 2) - Profils & AI

Backend :

- Intégration Mistral AI
- Système de profils clients
- Détection automatique du client

- Extraction AI + règles

Frontend :

- Page création profil client (AI)
- Page gestion clients
- Liste des commandes
- Amélioration interface validation

Livrable : Profils clients fonctionnels + Extraction AI

Sprint 3 (Semaine 3) - Prix & Features avancées

Backend :

- Gestion prix simples
- Gestion prix dégressifs
- Suggestions de correspondances
- Système de confiance (nb_validations)
- Alerte changement de prix
- Alerte doublon

Frontend :

- Interface prix dégressifs
- Page gestion articles & prix
- Recherche/filtres avancés
- Alertes et notifications

Livrable : Système complet avec prix et intelligence

Sprint 4 (Semaine 4) - Finitions & Production

Backend :

- Tâches automatiques (backup, cleanup)
- Statistiques
- Logs
- Optimisations performance
- Tests

Frontend :

- Page statistiques
- Page paramètres
- Interface logs

- Polish UX/UI
- Tests

Livrable : Application production-ready

✅ CHECKLIST PRÉ-LANCEMENT

Technique

- Base de données créée et indexée
- Articles importés depuis Excel
- 3 profils clients de test créés
- Extraction PDF testée sur 10 documents variés
- Correspondances testées (création, réutilisation, confiance)
- Prix simples testés
- Prix dégressifs testés
- Export CSV validé et compatible Odoo
- Sauvegarde automatique configurée
- Nettoyage automatique configuré
- Clé API Mistral configurée

Fonctionnel

- Workflow complet testé de bout en bout
- Gestion des erreurs testée
- Alertes testées (doublon, prix)
- Interface responsive testée
- Performance testée (10-20 commandes simultanées)

Documentation

- README.md créé
- Guide d'installation rédigé
- Guide utilisateur créé (avec captures)
- Formation prévue (1-2h)

🎯 CRITÈRES DE SUCCÈS

1. **Temps de traitement :** < 1 minute par commande (objectif : 30 secondes)
2. **Taux d'erreur :** < 1% sur les correspondances validées
3. **Taux d'auto-validation :** > 90% après 1 mois d'utilisation
4. **Formation :** Nouvelle personne autonome en < 2 heures
5. **Stabilité :** Aucun crash sur 100 commandes traitées

🆘 TROUBLESHOOTING

Erreur : PDF illisible

Cause : PDF scanné ou corrompu **Solution :** Mode saisie manuelle ou re-scanner le document

Erreur : Client non détecté

Cause : Nouveau client ou format inhabituel **Solution :** Sélection manuelle du client puis création de profil

Erreur : Extraction AI échoue

Cause : Clé API invalide, quota dépassé, ou format trop complexe **Solution :** Fallback sur extraction par règles ou saisie manuelle

Erreur : Base de données corrompue

Cause : Crash pendant écriture **Solution :** Restaurer dernière sauvegarde (backups/)

📝 NOTES IMPORTANTES

Limitations connues

- PDFs scannés (images) non supportés → saisie manuelle
- Maximum 50 articles par commande (limite technique pdfplumber)
- Extraction AI nécessite connexion internet

Évolutions futures possibles

- Support multi-utilisateurs avec gestion des droits
- Application mobile pour scan de PDFs
- Intégration directe avec Odoo (API)
- Reconnaissance OCR pour PDFs scannés
- Tableau de bord temps réel
- Notifications email/SMS
- Export autres formats (Excel, JSON)

🎓 FORMATION UTILISATEUR

Programme (1-2 heures)

Partie 1 : Découverte (15 min)

- Présentation de l'interface
- Principe des profils clients
- Système de correspondances

Partie 2 : Commande simple (30 min)

- Upload d'un PDF Kelias
- Vérification des données
- Validation et export CSV

Partie 3 : Commande complexe (30 min)

- Upload PDF avec nouvelle référence
- Recherche et sélection de correspondance
- Gestion d'un changement de prix

Partie 4 : Profil client (15 min)

- Création d'un profil avec AI
- Test avec PDF exemple

Partie 5 : Questions/Réponses (15 min)