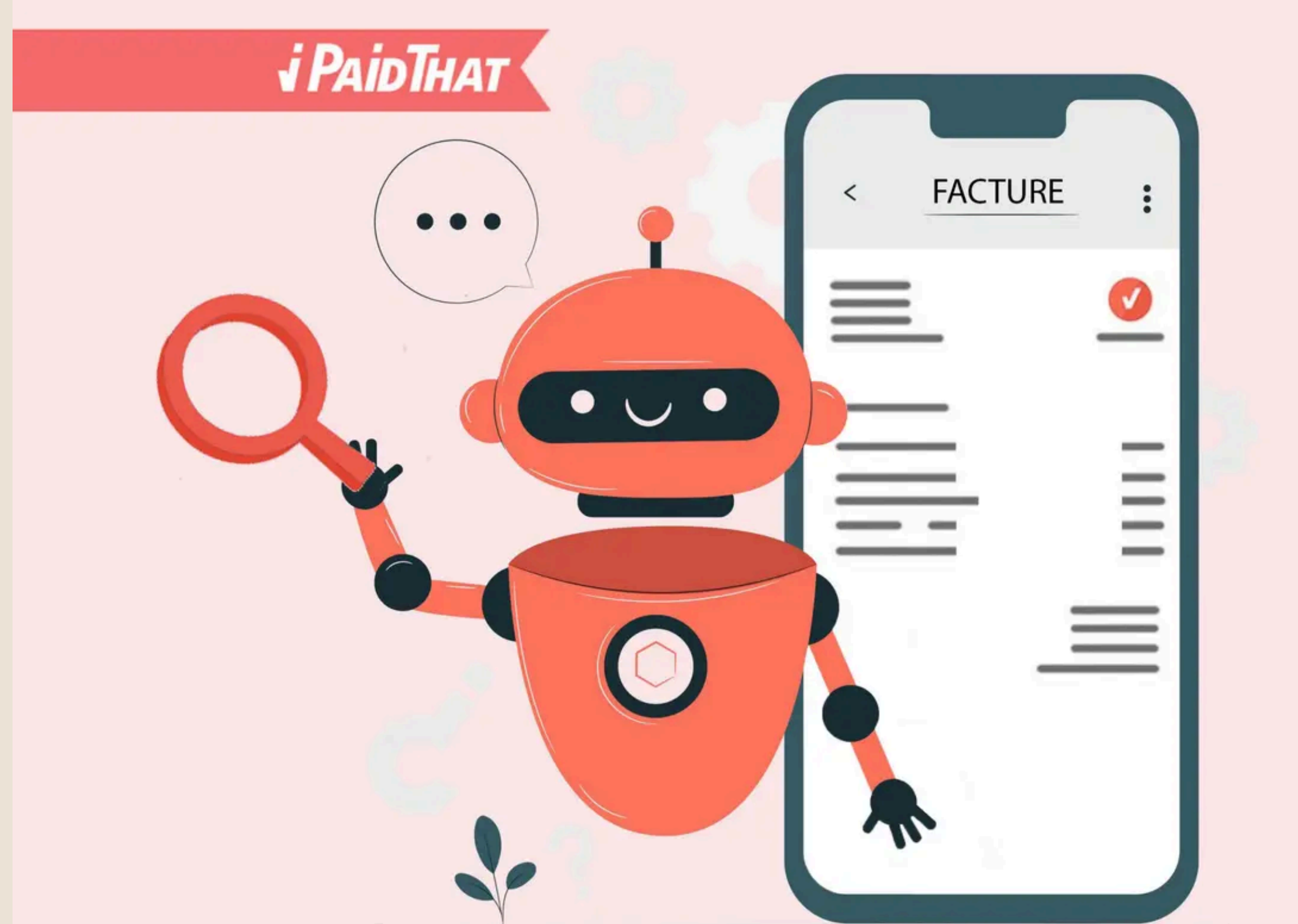


Développer une interface OCR

Par Promise John



Introduction

LE QUOTIDIEN D'UN COMMERÇANT

Un commerçant vend, encaisse, gère sa boutique.

- Mais à la fin de la semaine : la paperasse.
 - ➔ Saisie manuelle des factures
 - ➔ Long, répétitif, source d'erreurs
 - ➔ Du temps perdu

Pourtant, chaque facture contient des données clés :

- Ce qui se vend le mieux
- Ce qui se vend moins
- Des opportunités cachées

Mon application automatise la saisie, transforme les factures en données utiles, et aide à mieux vendre.



Organisation & Analyse

🕒 DEADLINE : 1 MOIS → UN VRAI DÉFI !

Étape 1 : Analyse des besoins

- Lecture attentive du brief
- Identification des fonctionnalités clés

Priorisation en 3 catégories :

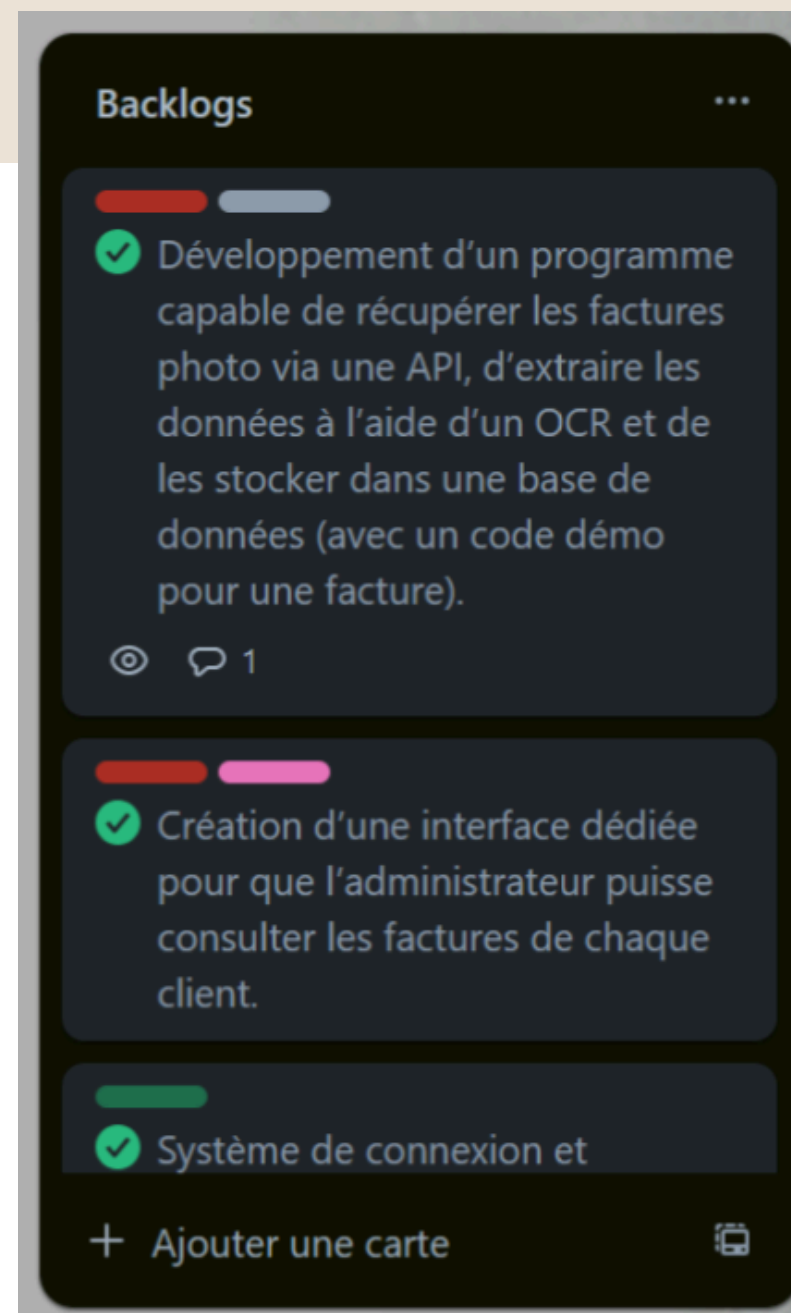
- Must Have (Indispensables)
- Nice to Have (Souhaitables)
- Dream (Optionnelles)

Must Have	Nice To Have	Dream
Programme capable de récupérer les factures photo via une API, d'extraire les données à l'aide d'un OCR et de les stocker dans une base de données	Système de connexion et d'inscription	interface permettant à l'utilisateur de consulter l'historique de ses factures.
interface dédiée pour que l'administrateur puisse consulter les factures de chaque client.	Système de monitoring	Dashboard de monitoring
		système de clustering de client
		Système de recommandation et propositions de produits

Suivi & gestion du projet avec Trello

UNE GESTION FLUIDE, RÉACTIVE, ET ORIENTÉE EFFICACITÉ

- Organisation en backlog
- Étiquettes de priorité
- Suivi itératif et adaptatif



- Ajout progressif de tâches précises
- Documentation des décisions, découvertes, ressources
- Suivi clair de l'évolution du projet

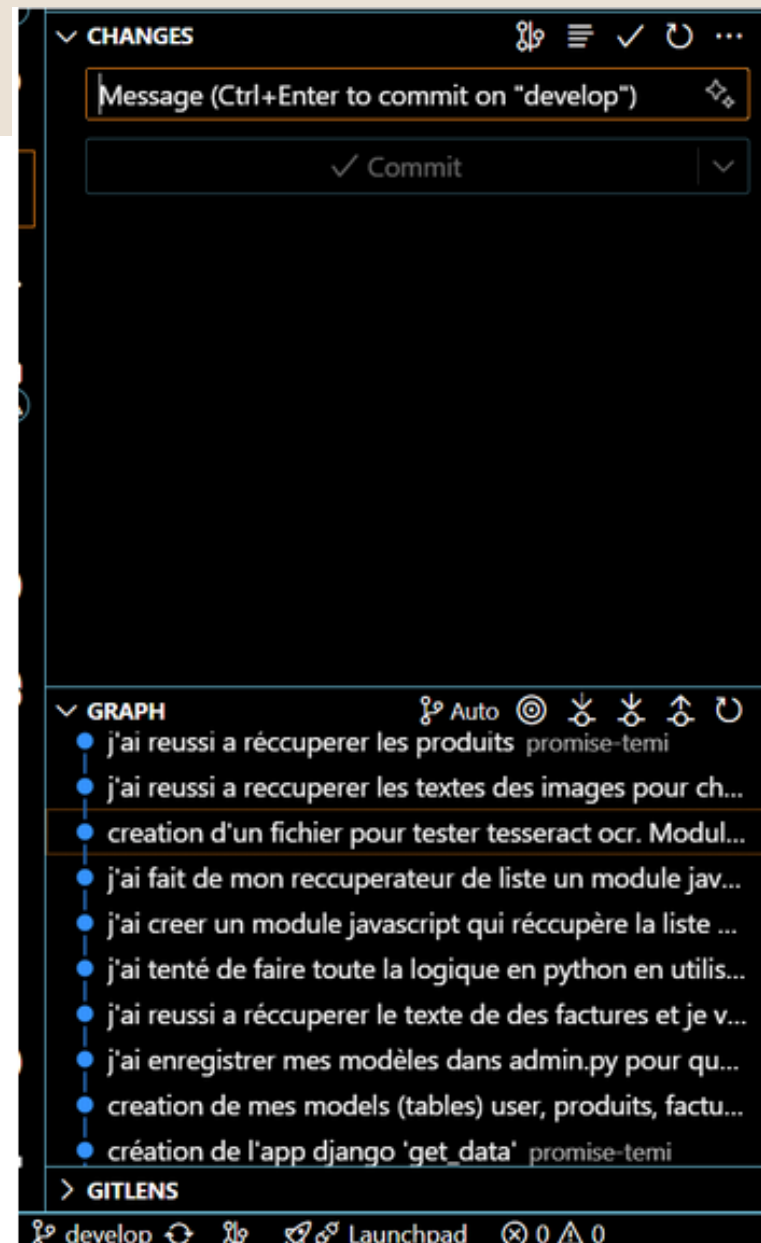


Gestion du code avec Git

UN CODE PROPRE, STRUCTURÉ ET FACILE À MAINTENIR

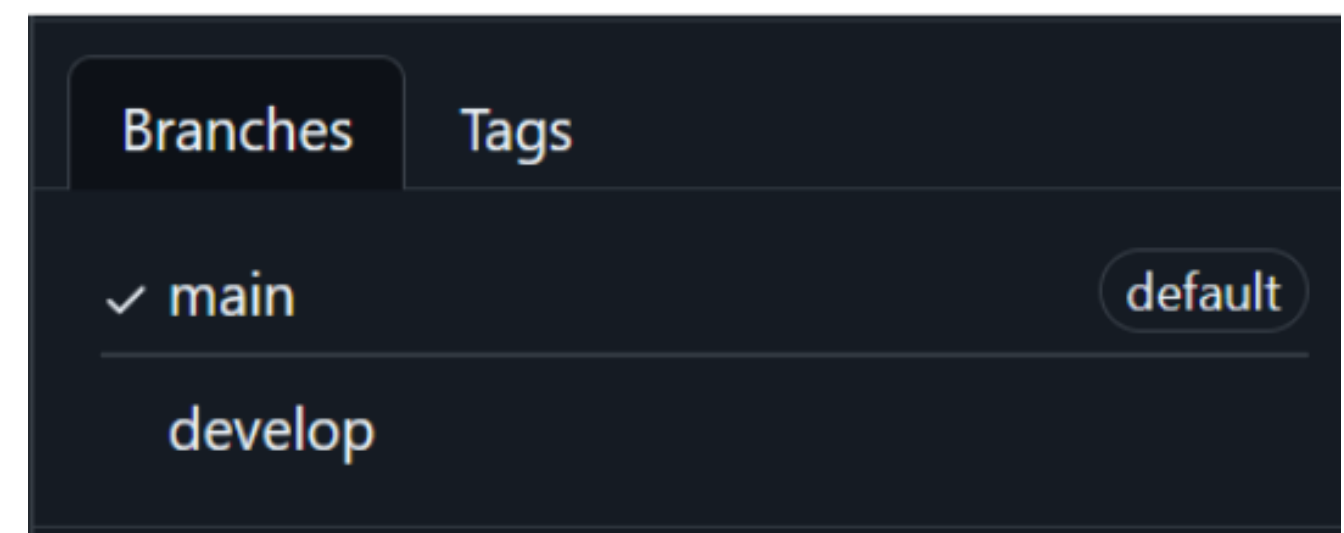
Suivi rigoureux des versions :

- Commits précis et documentés
- Historique clair des évolutions
- Meilleure traçabilité tout au long du projet



Organisation claire en branches :

- develop → pour les développements réguliers
- main → pour les versions stables et livrables



Conception d'interface

Figma : maquettes pour guider le développement

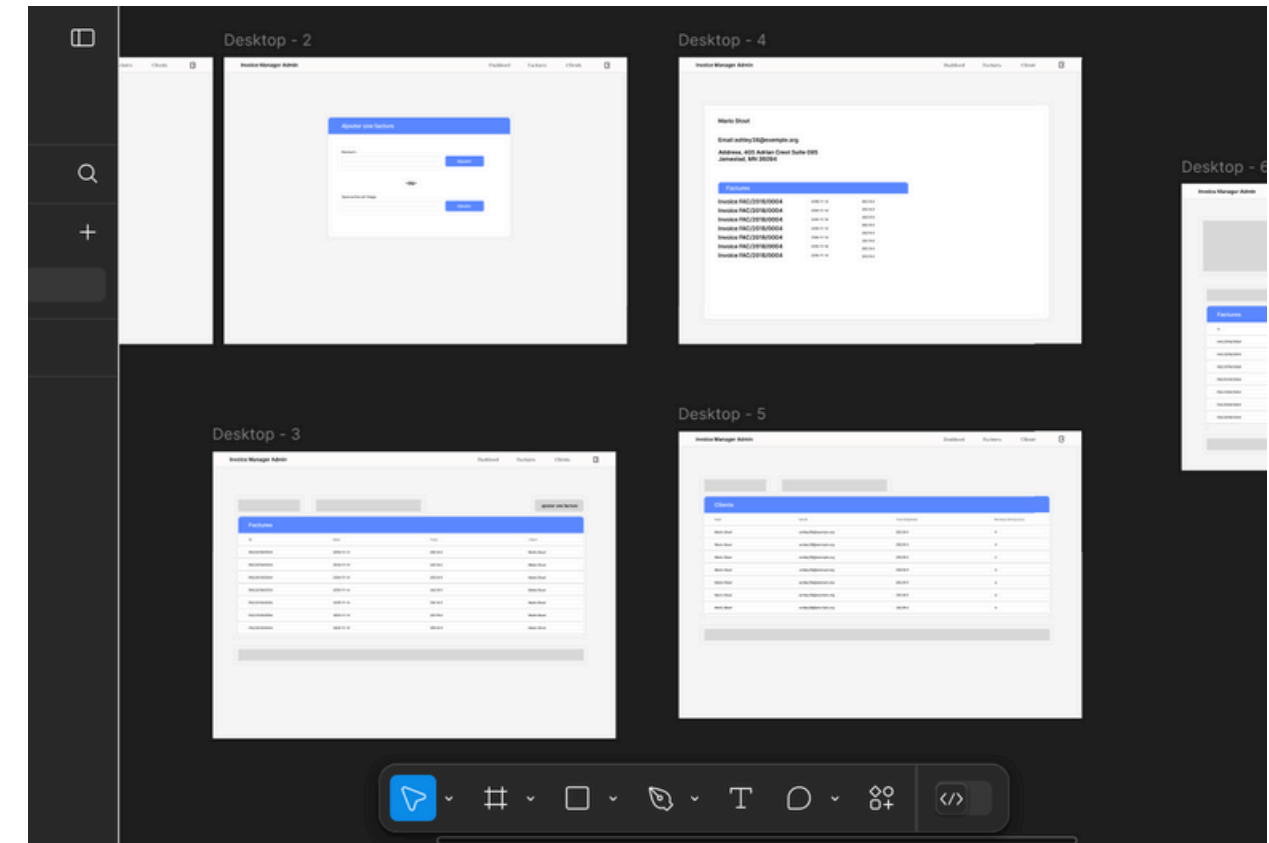
- Réalisées de manière itérative
- Maquettage fonctionnalité par fonctionnalité
- But : savoir quoi développer sans perdre de temps
- Parfois simplifiées en zonings rapides

Gain de temps : Bootstrap

- Utilisation de composants pré-stylés
- Alignement avec les maquettes
- Moins de temps passé sur le design → plus de focus sur la logique

Résultat :

Une vision claire des interfaces, une exécution rapide et structurée.



Nos données de base

- Images accessibles via une URL contenant un ID
- Un fichier XML listait tous les identifiants disponibles

Capture d'un bout du xml :

```
-<EnumerationResults ServiceEndpoint="https://projetocrstorageacc.blob.core.windows.net/" ContainerName="invoices-2018">
  <Blobs>
    <Blob>
      <Name>FAC_2018_0001-654.png</Name>
      <Properties>
        <Creation-Time>Wed, 05 Mar 2025 17:47:11 GMT</Creation-Time>
        <Last-Modified>Wed, 05 Mar 2025 17:47:11 GMT</Last-Modified>
        <Etag>0x8DD5C0DC1AFC838</Etag>
        <Content-Length>77748</Content-Length>
        <Content-Type>image/png</Content-Type>
        <Content-Encoding/>
        <Content-Language/>
        <Content-CRC64/>
        <Content-MD5>3hGVR9yLMRMDdDaHE7dAEA==</Content-MD5>
        <Cache-Control/>
        <Content-Disposition/>
        <BlobType>BlockBlob</BlobType>
        <AccessTier>Hot</AccessTier>
        <AccessTierInferred>true</AccessTierInferred>
        <LeaseStatus>unlocked</LeaseStatus>
        <LeaseState>available</LeaseState>
        <ServerEncrypted>true</ServerEncrypted>
      </Properties>
      <OrMetadata/>
    </Blob>
```

Capture d'un bout de l'image de facture :

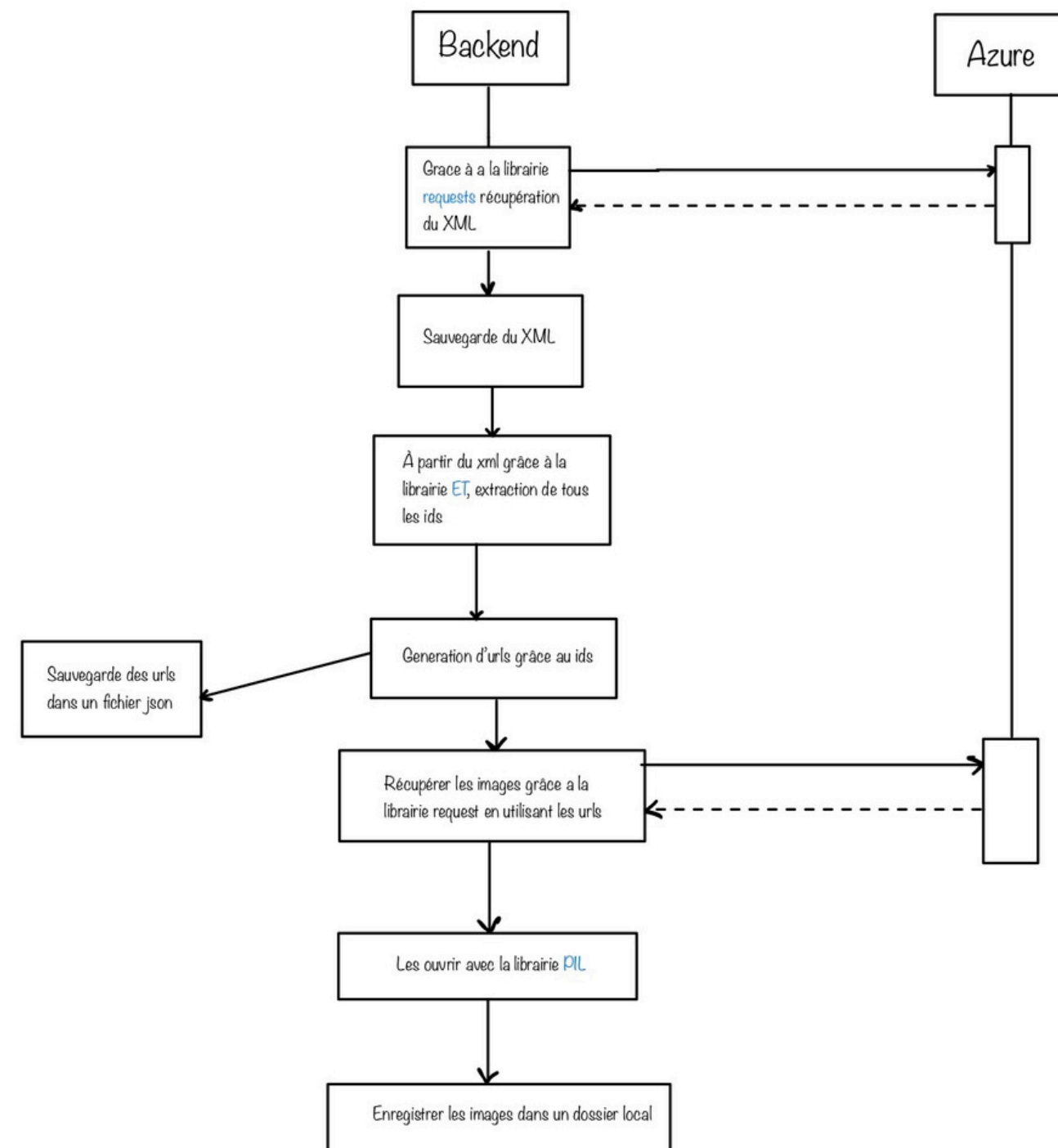


Automatisation du téléchargement

ÉLÉCHARGEMENT AUTOMATIQUE DES FACTURES

Script Python développé pour :

- ☒ Lire le fichier XML
- ☒ Extraire les IDs
- ☒ Générer dynamiquement les URLs
- ☒ Télécharger les factures en local



Choix de l'OCR

SOLUTIONS TESTÉES

Tesseract.js :

- ✗ Difficile à configurer
- ✗ Asynchrone en JavaScript → fastidieux

EasyOCR :

- ✓ Simple à utiliser
- ✗ Résultats peu précis

Choix de l'OCR

SOLUTIONS CHOISIE

Pytesseract (choix final)

- ✓ Basé sur Python
- ✓ Bon niveau de précision
- ✓ Fonctionne très bien même sans prétraitement
- ⚠ Installation fastidieuse, nécessite :
 - Installation de pytesseract (pip)
 - Installation de Tesseract-OCR (binaire système)
 - Configuration du PATH



Un tutoriel complet sur la reconnaissance optique de caractères...

Maîtrisez les fondamentaux de la reconnaissance optique de caractères (OCR) avec PyTesseract et...

 DataCamp / Jan 16

🔧 Découverte utile

Installé en dehors du venv → utilisable partout

➡ Dockerisation obligatoire!

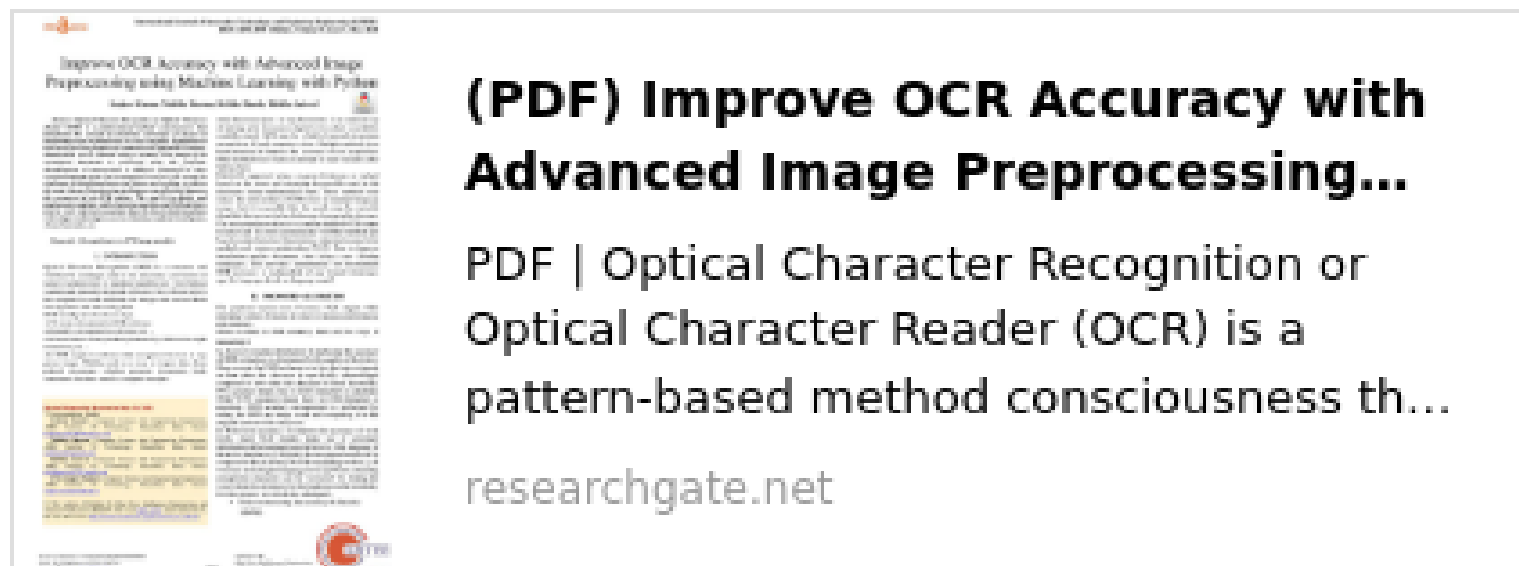
```
#image to text grace à pytesseract
text = pytesseract.image_to_string(image_final, lang='eng', config="--psm 6")
```

```
image_final, lang='eng', config="--psm 6")
```

```
#petit nettoyage du texte
text = text.lower().replace("]", "1")
```

Pipeline de Prétraitement (OpenCV)

VEILLE SUR COMMENT AMÉLIORER LA QUALITÉ DES RÉSULTATS OCR



(PDF) Improve OCR Accuracy with Advanced Image Preprocessing...

PDF | Optical Character Recognition or Optical Character Reader (OCR) is a pattern-based method consciousness th...
researchgate.net

 Medium

<https://medium.com/@Hiadore/preprocessing-images-for-ocr-a-step-by-step-guide-to-quality-recovery-923b6b8f926b>

Pipeline de Prétraitement (OpenCV)

AMÉLIORER LA QUALITÉ DES RÉSULTATS OCR

Masquage du QR code

```
# Dessiner un rectangle blanc pour cacher le QR code et l'image à côté  
cv2.rectangle(image_cv, top_left, bottom_right, (255, 255, 255), -1)
```

Conversion en niveaux de gris

```
# Convertir en niveaux de gris  
image = ImageOps.grayscale(image)
```

Seuillage adaptatif

```
#Appliquer un seuillage adaptatif pour améliorer le contraste  
image_cv = cv2.adaptiveThreshold(cv2.cvtColor(image_cv, cv2.COLOR_BGR2GRAY),  
                                255,  
                                cv2.ADAPTIVE_THRESH_GAUSSIAN_C,  
                                cv2.THRESH_BINARY,  
                                11, 2)
```

Renforcement de la netteté

```
# Augmenter la netteté  
enhancer = ImageEnhance.Sharpness(image)  
image = enhancer.enhance(4.0)
```


Pipeline de Prétraitement (OpenCV)

AMÉLIORER LA QUALITÉ DES RÉSULTATS OCR

➡ Résultat : une image optimisée pour être lue avec précision par l'OCR

➡ Intégré juste avant l'envoi à pytesseract

INVOICE FAC/2020/0204

Issue date 2020-04-11

Bill to Christina Cortez

Email teresalee@example.org

Address 09583 Travis River
West Brett, NC 58822

Plan allow buy turn.

4 x 127.34 Euro

TOTAL

509.36 Euro

Pipeline de Prétraitement (OpenCV)

RÉCCUPÉRATION DES DONNÉES DE QR CODE

```
# Initialiser le détecteur de QR codes
detector = cv2.QRCodeDetector()

# Détecter et décoder le QR code
data, bbox, _ = detector.detectAndDecode(image)
if data:
    print("Données du QR code :", data)
    datas = data.lower().split('\n')
    arranged_datas = {
        'invoice_time': get_time(datas),
        'genre' : get_genre(datas),
        'birth' : get_birth(datas)
```

Intégration avec Django

"LE FRAMEWORK DES PERFECTIONNISTES AVEC UNE DEADLINE

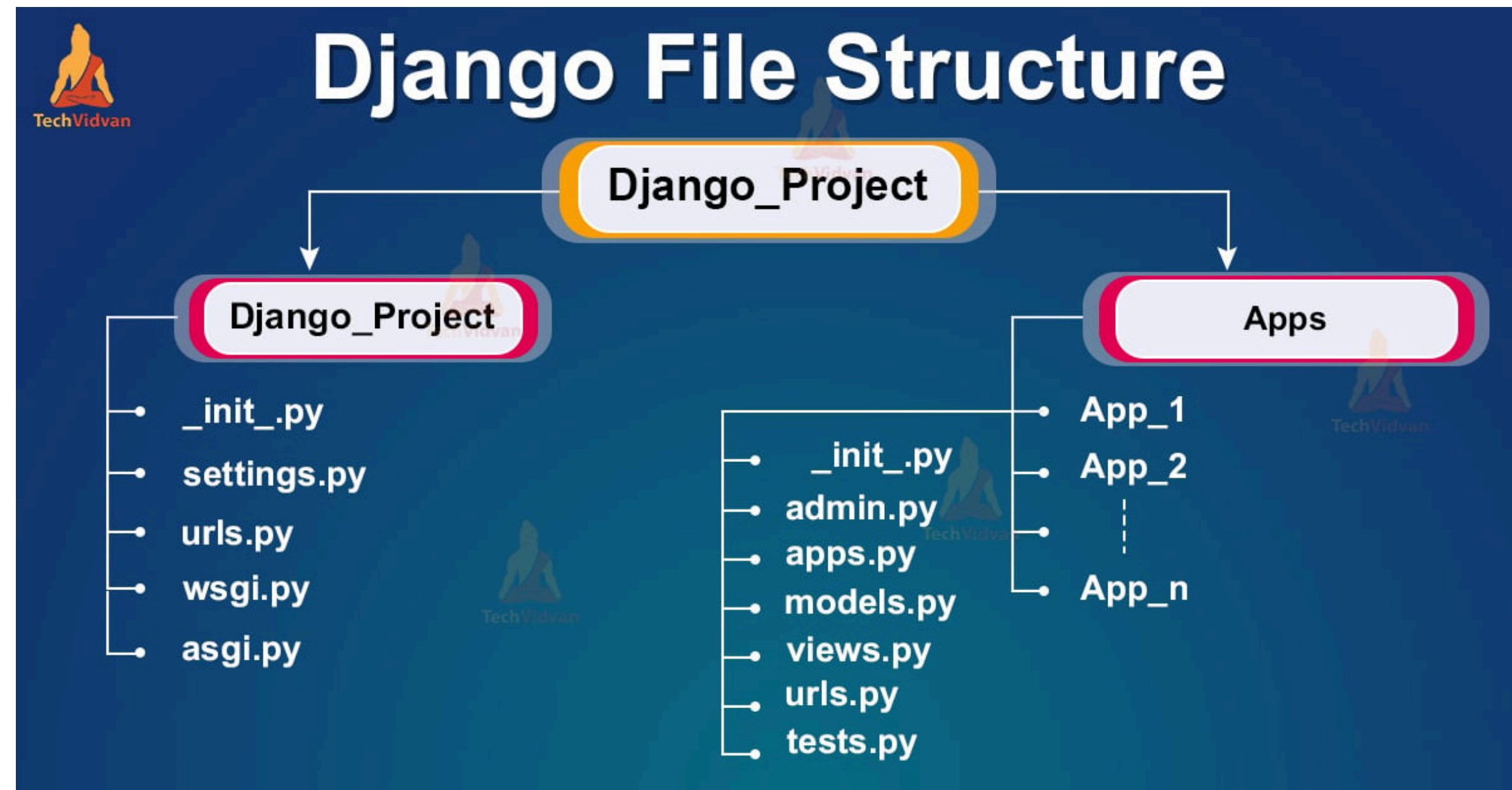
Pourquoi Django ?

- ✓ Robuste, complet, rapide
- ✓ Routage, ORM, templates dynamiques
- ✓ Auth intégrée, interface admin puissante

Sources pour m'aider à apprendre Django :

→ <https://www.w3schools.com/django/>

→ <https://www.djangoproject.com/start/>



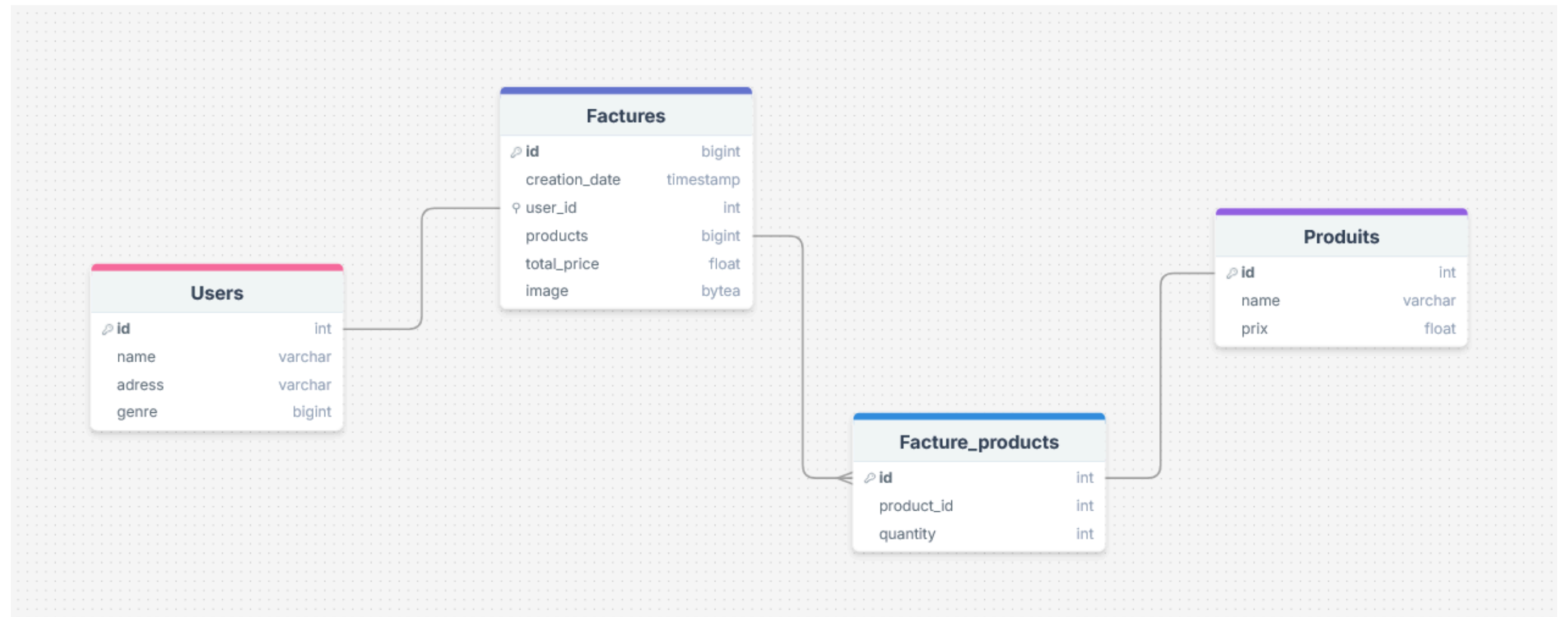
Mise en place de l'application

get_data

Gère toute la logique d'extraction et de traitement des factures

Modèles définis dans models.py :

- Produit
- User
- Facture
- FactureProduct



Mise en place de l'application

get_data

Exemple d'un modèle dans models.py :

```
class User(models.Model):
    name = models.CharField(max_length=250)
    email = models.CharField(max_length=300, null=True, blank=False, unique=True)
    adress = models.CharField(max_length=500)
    city = models.CharField(max_length=100, null=True)
    state = models.CharField(max_length=100, null=True)
    postal_code = models.CharField(max_length=10, null=True)
    type = models.CharField(max_length=10, default="user")
    genre = models.CharField(max_length=10, null=True)
    birth_day = models.DateField(null=True)
```

Ajout des models à admin.py pour y avoir accès dans l'interface admin

```
from django.contrib import admin
from .models import Produit, Facture, User, FactureProduct,

# Register your models here.
admin.site.register(Produit)
admin.site.register(User)
admin.site.register(Facture)
admin.site.register(FactureProduct)
admin.site.register(UserClusterFeatures)
```

Création d'un super admin

Django administration

WELCOME, **SUPER_USER**. [VIEW SITE](#) / [CHANGE PASSWORD](#) / [LOG OUT](#)

Site administration

AUTHENTICATION AND AUTHORIZATION

Groups

+ Add

Change

Users

+ Add

Change

GET_DATA

Facture products

+ Add

Change

Factures

+ Add

Change

Log entrys

+ Add

Change

Produits

+ Add

Change

User cluster featuress

+ Add

Change

Users

+ Add

Change

Recent actions

My actions

✗ User object (46)

User

✗ User object (45)

User

✗ User object (44)

User

✗ User object (43)

User

✗ User object (42)

User

✗ User object (41)

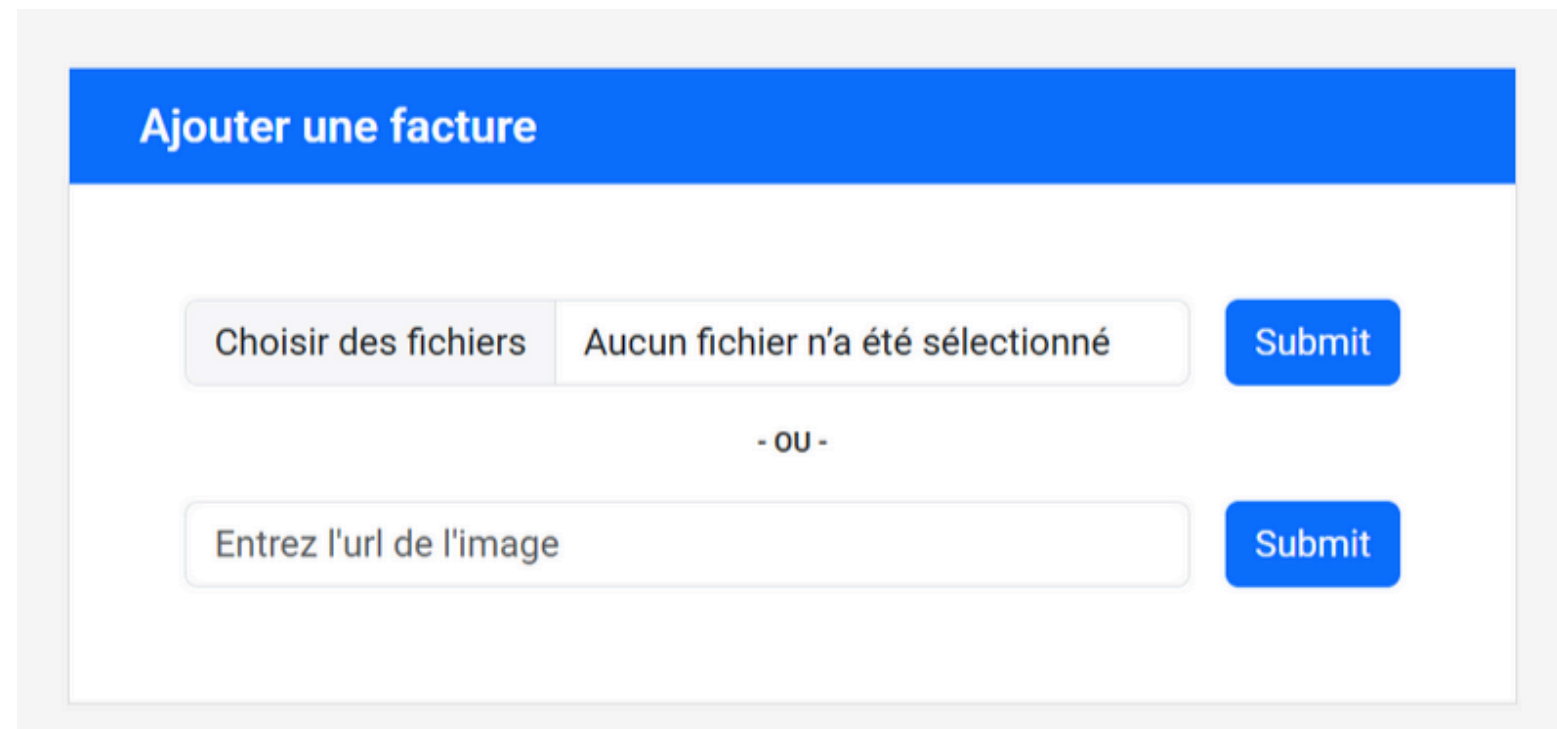
User

Automatisation du Flux de Facturation

Page HTML : create-invoice.html

Contient 2 formulaires :

1. 📁 Téléversement d'image locale (add-invoice2)
2. 🔗 Téléversement via URL (add-invoice)

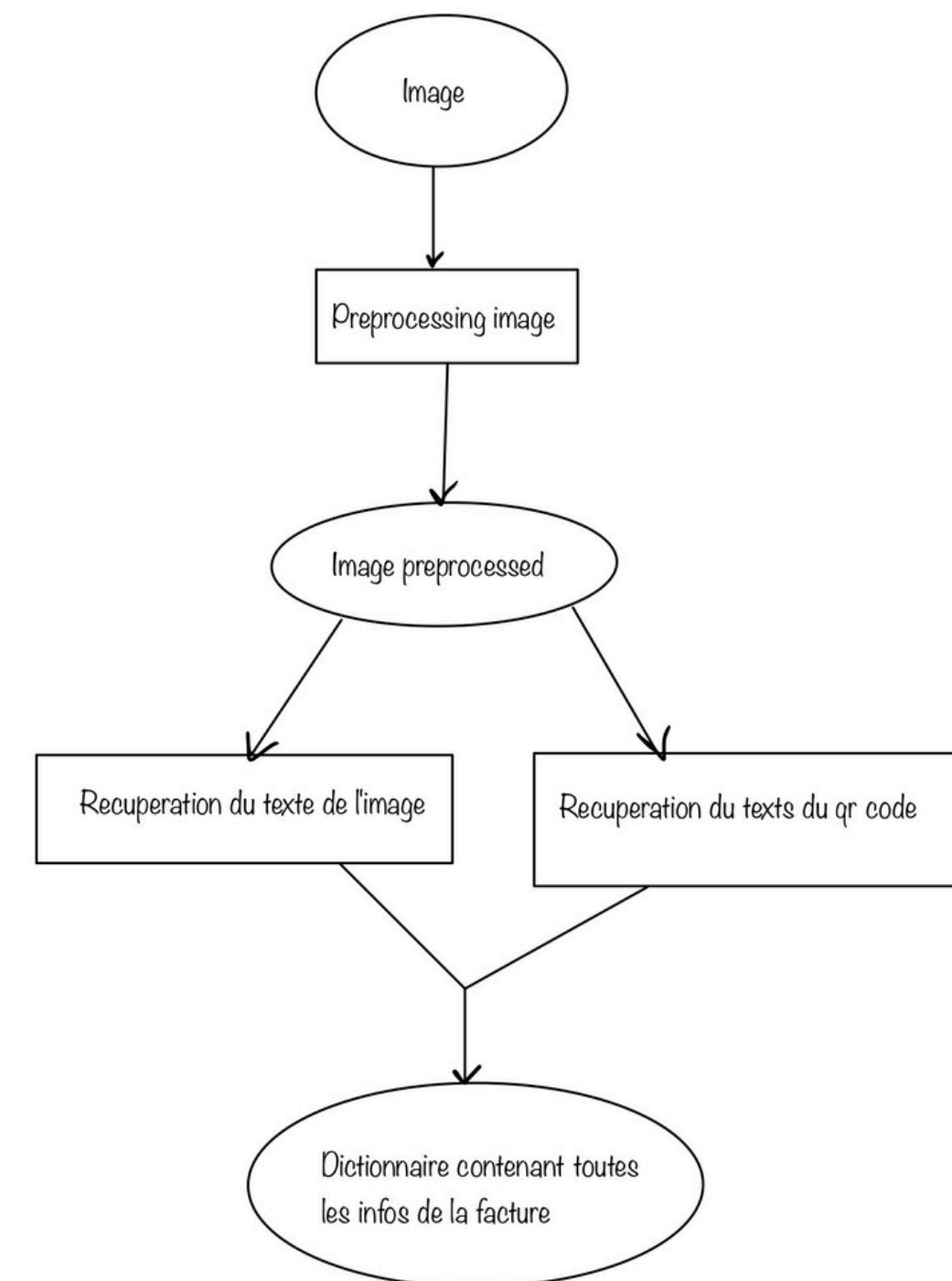
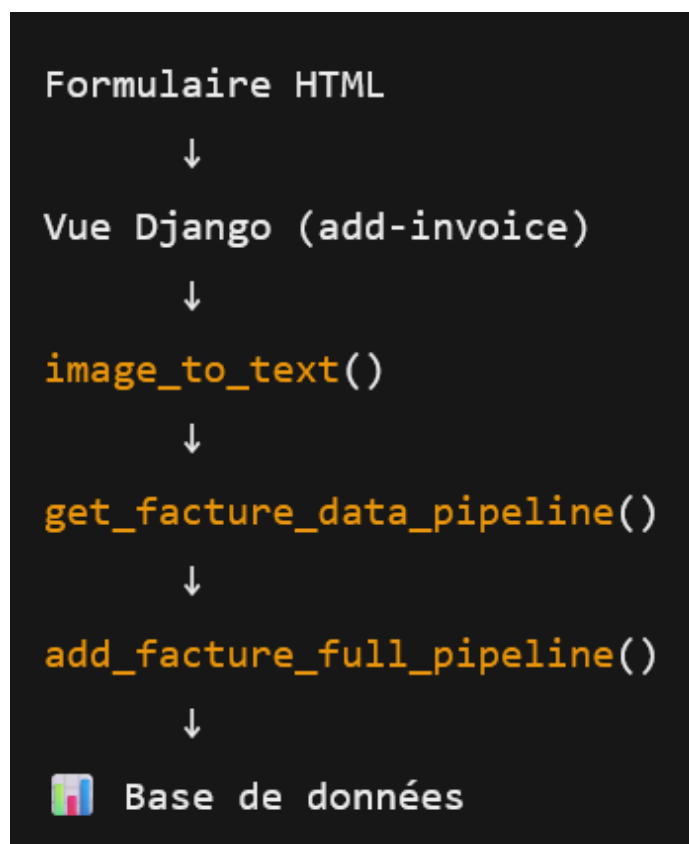


The screenshot shows a web form titled "Ajouter une facture" (Add an invoice) with a blue header. Below the header, there are two distinct sections for adding an invoice. The first section, for local file upload, includes a button labeled "Choisir des fichiers" (Choose files), a text input field containing "Aucun fichier n'a été sélectionné" (No file has been selected), and a blue "Submit" button. The second section, for URL upload, is separated by a "- ou -" (or) separator and includes a text input field labeled "Entrez l'url de l'image" (Enter the image URL) and another blue "Submit" button.

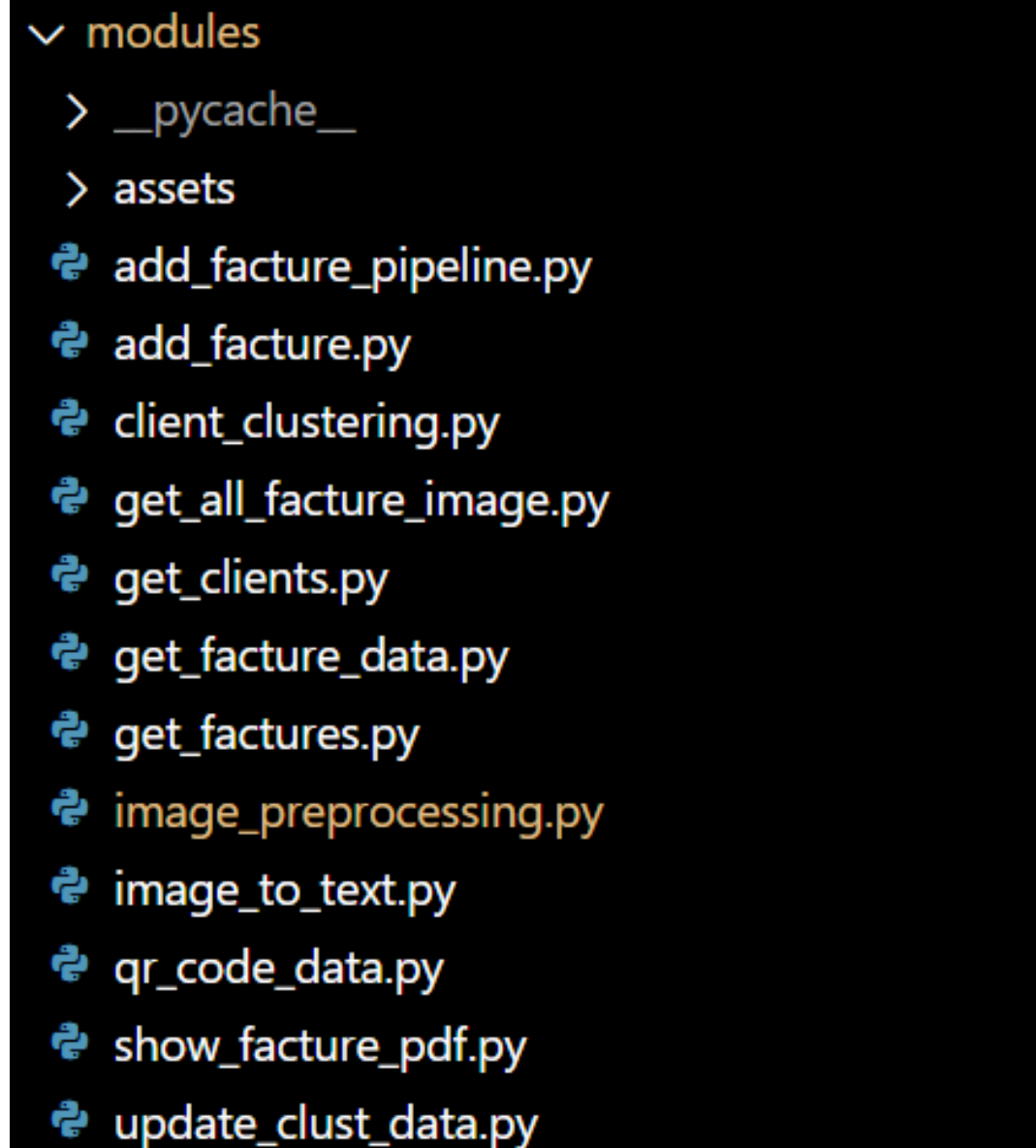
Automatisation du Flux de Facturation

Fonctionnement du pipeline :

1. Envoi du formulaire
2. Appel de `image_to_text()`
3. Extraction via pytesseract (OCR) + OpenCV (QR code)
4. Récupération et structuration des données avec `get_facture_data_pipeline()`
5. Insertion dans la BDD via `add_facture_full_pipeline()`



Organisation du code – Travail modulaire



A screenshot of a file explorer window showing a directory structure for a project. The 'modules' directory is expanded, revealing a list of Python files. The files are: __pycache__, assets, add_facture_pipeline.py, add_facture.py, client_clustering.py, get_all_facture_image.py, get_clients.py, get_facture_data.py, get_factures.py, image_preprocessing.py, image_to_text.py, qr_code_data.py, show_facture_pdf.py, and update_clust_data.py. Each file is preceded by a small icon representing a document.

```
✓ modules
  > __pycache__
  > assets
  📄 add_facture_pipeline.py
  📄 add_facture.py
  📄 client_clustering.py
  📄 get_all_facture_image.py
  📄 get_clients.py
  📄 get_facture_data.py
  📄 get_factures.py
  📄 image_preprocessing.py
  📄 image_to_text.py
  📄 qr_code_data.py
  📄 show_facture_pdf.py
  📄 update_clust_data.py
```

Pour ne pas me perdre dans la complexité du projet, j'ai choisi de travailler de manière modulaire.

Chaque fonctionnalité (OCR, parsing, base de données, clustering...) a été développée dans un fichier ou module dédié.

Résultat :

- Un code plus clair
- Des fonctions réutilisables
- Une architecture facile à maintenir et à faire évoluer

Tests et Validation

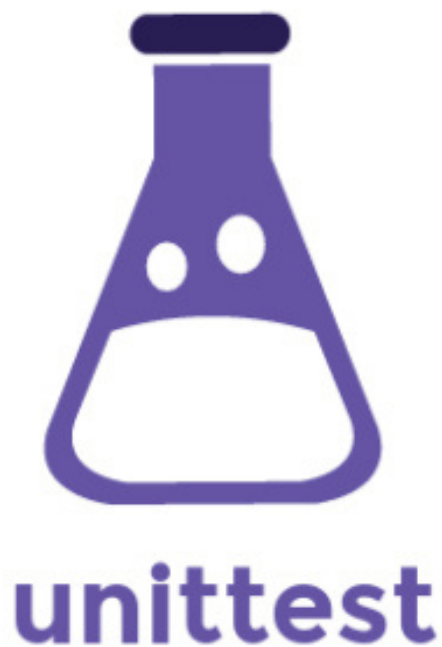
"S'IL TE PLAÎT MARCHE !"

Tests avec `django.test.TestCase` :

- `test_image_to_text_file()` → images locales
- `test_image_to_text_link()` → images Azure

Objectif :

- Vérifier la bonne extraction des données OCR & QR code
- Assurer la stabilité du pipeline



```
#TEST
text_data, qr_code_data = image_to_text(image, "file")
#doit me retourner un dictionnaire
self.assertIsInstance(text_data, dict)
#ce dictionnaire doit contenir la clé text
self.assertIn('text', text_data)
#les données de clé text du dict retourné doit contenir du text
self.assertIsInstance(text_data['text'], str)

# dans ce texte je doit retrouver obligatoirement ces données

#id doit etre present
self.assertRegex(text_data['text'], r"invoice\s+fac/\d{4}/\d+")
#date doit etre presente
self.assertRegex(text_data['text'], r'issue\s+date\s+\d{4}-\d{2}-\d{2}')
```

Interface d'Administration Personnalisée

Nouvelle app : admin_interface

Pages créées :

- factures.html → liste des factures
- info_facture.html → détail facture
- clients.html & info_client.html

Client #165

nom : kristy fitzgerald

Adresse complete :
965 thomas drive apt. 884, micheal pa 45193

email :
[woodbrandon@example.net]

sexe : m

Date de naissance: March 17, 1994

Produits susceptible de l'interesser

- produit 1
- produit 2
- produit 3

Factures

ID	Date	Total
fac/2018/0037	Dec. 24, 2018, 1:28 p.m.	63.78 €
fac/2019/0015	Jan. 7, 2019, 5:27 a.m.	79.0 €
fac/2019/0294	May 9, 2019, 10:40 a.m.	None €

Invoice Manager Admin

Dashbord

Factures

Clients

Ajouter une facture

Clients

ID	Nom	Ville	Nombre d'achats	Total dépensé
136	carol potter	jamesstad	1	1146.84 €
137	sarmuel coleman	turnermouth	1	182.8 €
138	richard dunn	daniel	1	152.98 €
139	mario stout	walshshire	2	292.16 €
140	rachel ramirez	kennethborough	1	74.84 €
141	richard green	amandashire	1	183.93 €
142	bruce pace	craigfort	1	21.9 €
143	david madonald	mayfort	1	0.00 €

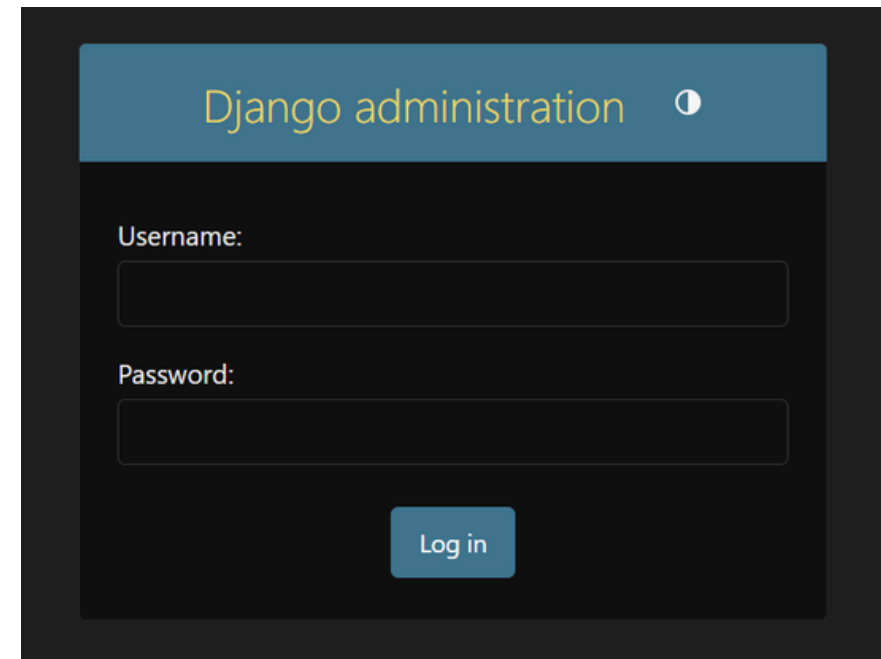
Interface d'Administration Personnalisée

🔒 Sécurisées avec le décorateur @login_required

→ Création d'un superuser au début du projet

```
from django.contrib.auth.decorators import login_required

@login_required
def factures(request):
    data = getfacture()
    template = loader.get_template('factures.html')
    context = {
        'mes_factures': data,
        'You, last week • j'airive'
    }
    return HttpResponse(template.render(context, request))
```

A screenshot of the Django administration login interface. It features a dark blue header with the text "Django administration" and a small circular icon. Below the header, there are two input fields: "Username:" and "Password:". At the bottom, there is a blue button labeled "Log in".

Monitoring Applicatif

Modèle : LogEntry

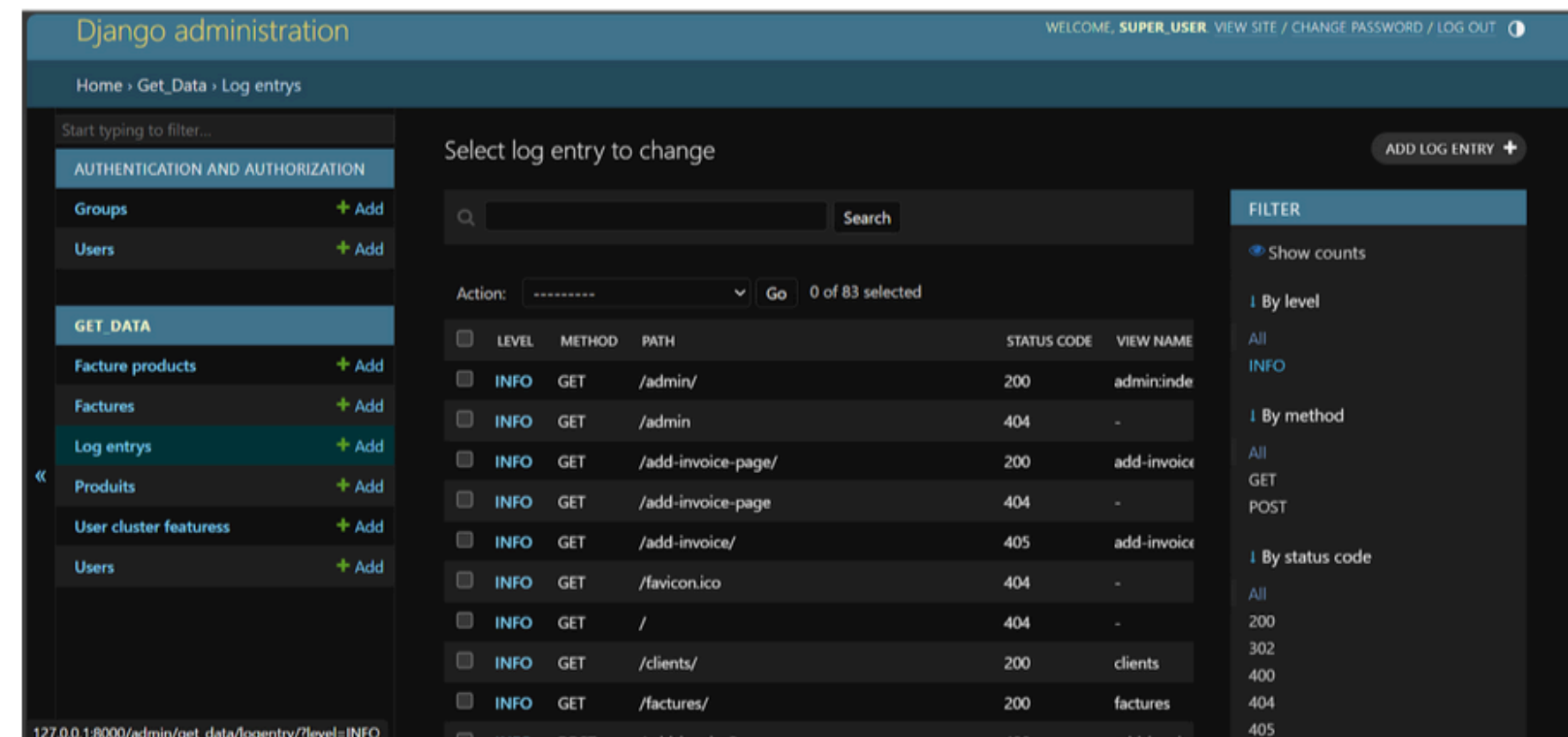
→ Sauvegarde les requêtes HTTP critiques

⚙️ Middleware perso : RequestLoggingMiddleware

→ Capture : méthode, URL, statut, durée

Visualisation via LogEntryAdmin dans l'interface Django

```
class LogEntry(models.Model):
    level = models.CharField(max_length=20)
    message = models.TextField()
    path = models.CharField(max_length=255)
    method = models.CharField(max_length=10)
    status_code = models.PositiveSmallIntegerField()
    view_name = models.CharField(max_length=255, null=True, blank=True)
    ip_address = models.GenericIPAddressField(null=True, blank=True)
    duration = models.FloatField(null=True, blank=True)
    extra_data = models.JSONField(null=True, blank=True)
    created_at = models.DateTimeField(auto_now_add=True)
```



Monitoring Aplikatif

```
MIDDLEWARE = [  
    'django.middleware.security.SecurityMiddleware',  
    'django.contrib.sessions.middleware.SessionMiddleware',  
    'django.middleware.common.CommonMiddleware',  
    'django.middleware.csrf.CsrfViewMiddleware',  
    'django.contrib.auth.middleware.AuthenticationMiddleware',  
    'django.contrib.messages.middleware.MessageMiddleware',  
    'django.middleware.clickjacking.XFrameOptionsMiddleware',  
    # MES MIDDLEWARES  
    'get_data.middleware.RequestLoggingMiddleware',  
]
```



Merci !