

Océrisation et extraction de données de factures

Titre professionnel "Développeur en Intelligence Artificielle"

Enregistré au TNCP sous le n° 34757



Pierre-Vincent FERRAT (06/2021)

Sommaire

Contexte du projet	4
Fonctionnement en entreprise	4
Problématique et travail de l'expert-comptable	4
Qu'est-ce qu'un logiciel d'Océrisation ?	5
Expression du besoin	6
Gestion du projet	6
Pilotage du projet	6
Méthodologie du projet	7
Rappel du cycle Scrum	7
Outils utilisés pour le projet	8
Le Scrum board (tableau scrum), un des piliers de la transparence	8
Les sprints	10
Lancement et imprégnation	10
Choix des technologies d'Intelligence Artificielle	11
Océrisation	11
Choix de l'API	11
Fonctionnement Amazon Textract et preprocessing des factures	12
Natural Language Processing	12
Données d'entraînement du modèle	14
Création de données d'entraînement pour SpaCy	14
Utilisation de données existantes	18
Entraînement du modèle	20
Base de données	22
Structure de la base de données	23
Les requêtes SQL via SQLAlchemy	24
Le modèle de base de données	24
Back-up de la base de données	25
Intégration et utilisation du modèle dans l'application	25
Maquettage et parcours Utilisateur	26
Maquettage de la solution proposée	27
Important :	27
Back-end	28

Monitoring de l'application	28
Stratégie de tests	30
Tests unitaires	30
Tests fonctionnels	31
Déploiement de l'application	32
Conclusion	33
Annexe 1 : Projet E3 – État de l'art	34
Annexe 2 : API d'OCR (quelques exemples)	41
Annexe 3 : Amazon Textract et processing	47
Annexe 4 : Création BDD	51
Annexe 5 : BDD modèle	52
Annexe 6 : Fonction d'utilisation du modèle SpaCy	53
Annexe 7 : Arbre du dossier du projet	54
Annexe 8 : Exemple de notification Sentry reçue par mail	55
Annexe 9 : Mockups et captures de l'application	56
Annexe 10 : Dockerfile	61
Annexe 11 : Scores	62
Annexe 12 : Exemple avec UBIAI : (https://ubiai.tools/)	65

I. Contexte du projet

J'ai réalisé mon alternance, et donc ce projet, au sein de l'entreprise Exco de Bayonne. Exco est un cabinet d'Expertise comptable, d'audit, de conseil, d'activités juridiques, fiscales et sociales qui comprend 140 cabinets, 2400 collaborateurs et qui est implanté dans 17 pays dans le Monde. La partie française est scindée en 5 grandes régions. Mon bureau de rattachement est celui d'Exco Fiduciaire Sud Ouest, au sein de l'agence de Bayonne (64). J'étais directement rattaché à Magali VILLENAVE, Associée et Directrice Innovation et Développement pour une action transverse au sein du groupe. Mon tuteur de stage était David GOGUET, Associé et Directeur des Systèmes d'Information, basé à Mérignac (33).

1. Fonctionnement en entreprise

Les 5 grandes régions d'Exco France fonctionnent comme des holdings indépendantes : une communication unique, mais des gouvernances et des services propres à chaque région. Au sein de ces régions, les différents cabinets Exco ont des latitudes d'organisation, et de fonctionnement, mais font partie de la même entité juridique, centralisée à Toulouse.

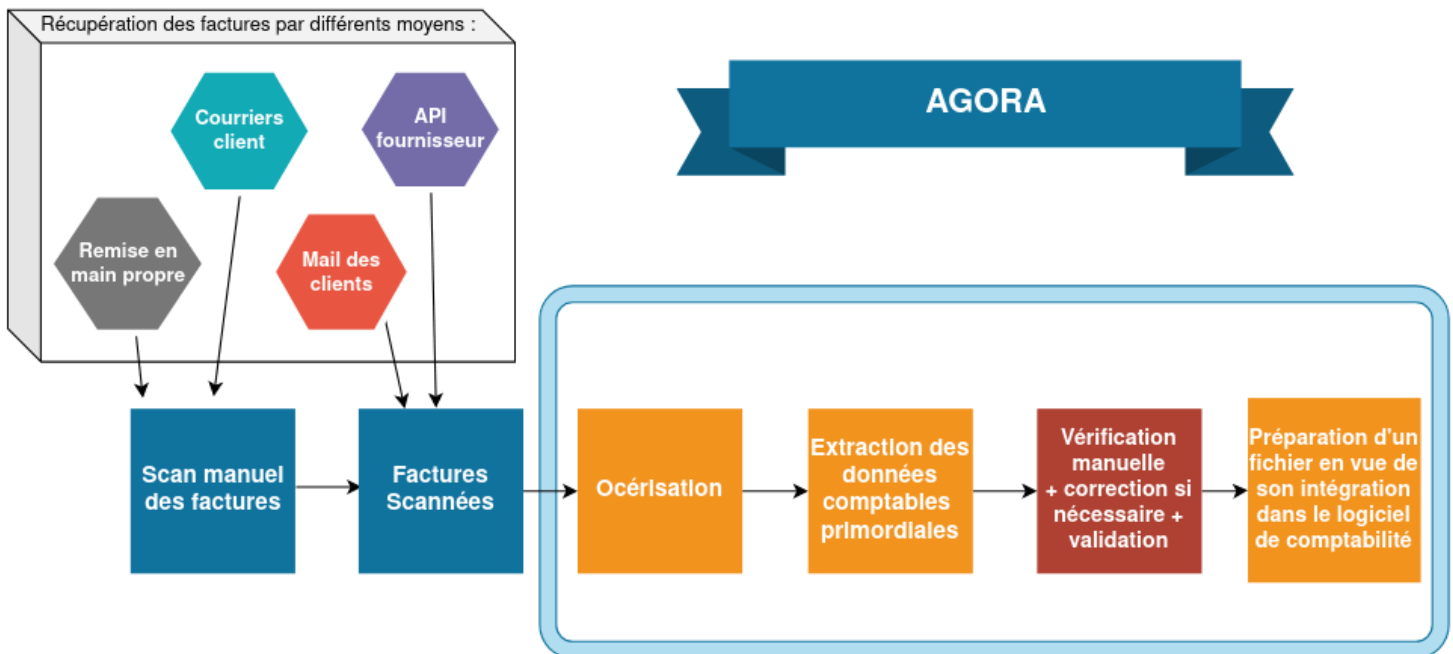
J'étais le seul développeur du cabinet de Bayonne et de l'entreprise Exco Fiduciaire Sud Ouest. Grâce à la confiance de mon tuteur, j'ai pu évoluer en grande autonomie dans mon travail, que ce soit d'un point de vue organisationnel mais également dans les reportings et suivis. Concernant le choix des technologies y compris pour la réalisation complète de ce projet, j'ai eu une très grande latitude d'actions.

2. Problématique et travail de l'expert-comptable

Une partie des missions des experts-comptables consiste à récupérer les factures d'achat ou de vente des clients dont ils gèrent la comptabilité. Ces factures sont envoyées au comptable soit en version papier, soit en version numérique. Ensuite, le comptable doit reprendre manuellement les informations de ces factures, en les lisant, afin de les saisir à la main dans son livre de comptabilité.

Heureusement, des outils existent pour faciliter et automatiser ce traitement et cette ressaisie fastidieuse des données, dans un nouvel outil informatique propre à l'entreprise. Le logiciel en production actuellement chez Exco est efficace et s'appelle AGORA.

Voici les grandes étapes de fonctionnement du logiciel AGORA :



3. Qu'est-ce qu'un logiciel d'Océrisation ?

Définition : un logiciel **OCR** est un logiciel de Reconnaissance Optique de Caractères. Concrètement, ce type de logiciel permet de transformer n'importe quel document imprimé ou scanné en un fichier bureautique qu'il sera possible d'exploiter et de modifier par la suite. Ces logiciels utilisent des technologies poussées de reconnaissance de caractères pour pouvoir convertir, le plus justement possible, un fichier a priori inexploitable, qu'il faudrait entièrement retaper à la main, en un fichier exploitable, et cela en quelques instants. C'est ce que l'on appelle "l'**océrisation**"

Pour plus de détails sur l'océrisation : **Cf Annexe 1 : Projet E3 – État de l'art**
Reconnaissance automatique de caractères, appliquée aux factures

4. Expression du besoin

Aujourd'hui, la question du coût d'un logiciel tel qu'AGORA est étudiée. Mais pas seulement. La DSI d'Exco aimerait connaître la **possibilité d'internaliser** le workflow du traitement des factures pour en maîtriser la technologie, garder la **maîtrise de ses coûts**, mais aussi **conserver le contrôle sur les données** qui ont de plus en plus tendance à être externalisées, notamment avec les solutions cloud.

Sujet : Étude de la possibilité de développer un outil interne, permettant de remplacer, en tout ou partie, le logiciel AGORA déjà en place.

Voici la liste des besoins exprimés par ma hiérarchie concernant les fonctionnalités d'un potentiel outil interne :

- Océrisation des factures
- Reconnaissance et extraction des données essentielles à la comptabilité
- Vérification des données extraites et mise à jour de celles-ci par le comptable
- Préparation d'un fichier de données permettant de réaliser les "écritures bancaires*" (*enregistrement d'un flux bancaire ou financier dans le grand livre de comptabilité)
- Contrainte imposée : Ne pas développer d'outils d'océrisation à proprement dit. Je dois utiliser une API permettant de réaliser l'opération d'océrisation

II. Gestion du projet

1. Pilotage du projet

Le chef de projet était David GOGUET. J'ai été le seul développeur à travailler sur ce projet. D'ailleurs, j'ai réalisé la maîtrise d'ouvrage du projet de manière complète. En voici les points clés :

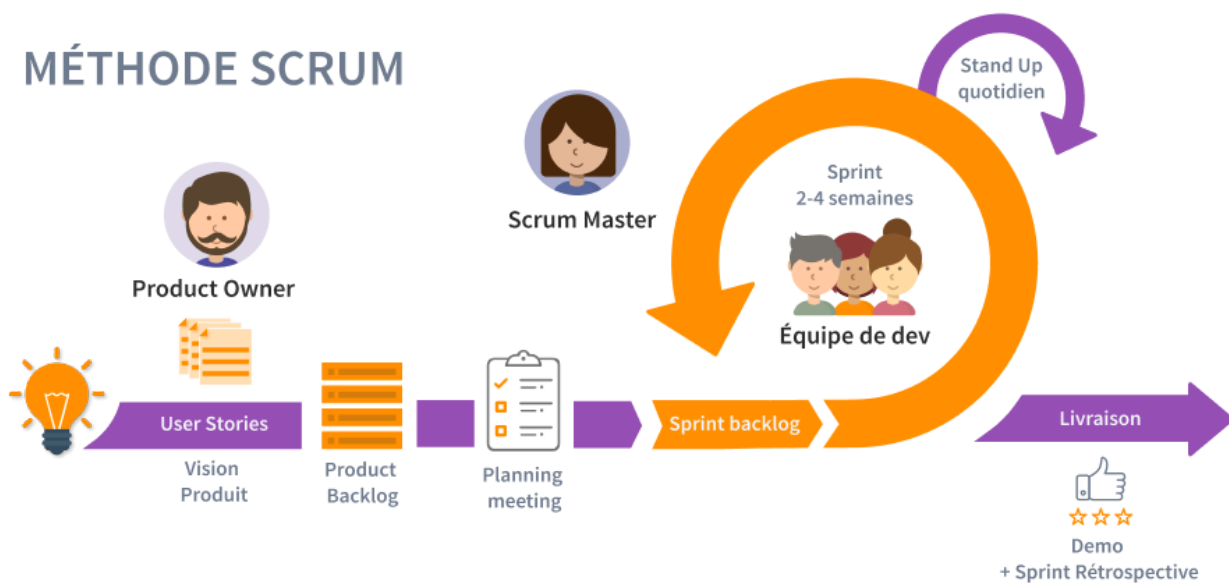
- Méthodologie de réalisation du projet
- Choix de l'outil de gestion de projet
- Choix de l'outil de versioning de code
- Choix des technologies utilisées (langage de programmation, type d'Intelligence Artificielle)
- Benchmark des solutions et api d'océrisation existantes
- Création d'un modèle d'IA et son entraînement
- Développement d'un prototype embarqué dans une application "web"
- " Dockerisation " (conteneurisation)
- Déploiement
- Tests

2. Méthodologie du projet

Pour réaliser ce projet, il m'a semblé pertinent de choisir d'**utiliser la méthode Scrum**, qui est l'une des méthodes agiles éprouvée. Son grand principe fonctionne sur le mode de livraison par blocs de fonctionnalité, permettant une **optimisation du projet** grâce aux feed-backs réguliers des participants. Ainsi, une adaptation permanente des objectifs est permise conformément aux principes de la méthode. J'ai décidé de l'adopter pour deux raisons principales : parce que je la connais bien grâce à la certification que j'ai obtenue en 2020, mais aussi parce que j'ai parfois été amené à l'utiliser lors des mes précédentes expériences professionnelles.

Étant en autonomie complète sur ce projet, j'ai opté pour l'application des grands principes tels que l'**utilisation d'un tableau de suivi** de type Kanban, la **livraison de fonctionnalités par blocs**, mais également l'**utilisation de sprints** pour formaliser les grands temps forts du projet.

a. Rappel du cycle Scrum



Source www.tuleap.org

J'ai, toutefois, dû adapter cette méthode à une **utilisation unipersonnelle**, de la façon suivante :

- Le product Owner, en tant que porteur de la vision produit : David Goguet
- Le scrum Master, garant de l'application de l'approche : moi-même
- L'équipe de développement : moi-même

Nous avons conservé un rituel hebdomadaire de revue de sprint. Chaque vendredi matin, j'animais la réunion de présentation des avancements auprès de David Goguet. Cette réunion hebdomadaire a d'ailleurs été utilisée comme un daily, rétrospective, planning de sprint et réunion de planification de sprint.

b. Outils utilisés pour le projet

Afin de mettre en place ce projet, j'ai eu recours aux outils suivants :

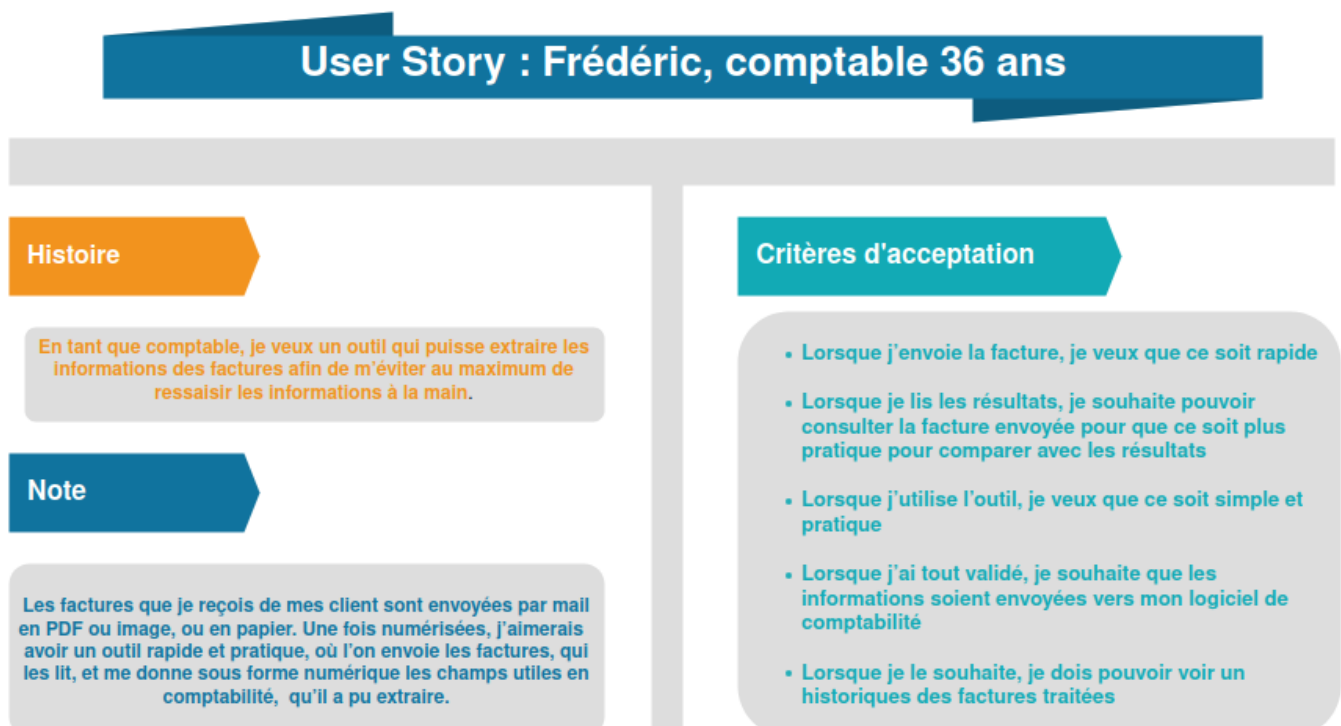
- GitLab dashboard : pour réaliser le scrum board numérique / Kanban + product backlog
- GitLab : pour le versionning du code
- VSCodium : l'IDE de développement
- Machine Linux sous Debian

c. Le Scrum board (tableau scrum), un des piliers de la transparence

Les 3 éléments suivants constituent mon scrum board :

Users story

Ci-dessous, la principale user story, créée à partir du **personae d'un comptable représentatif**. Celle-ci m'a servi de ligne directrice pour que le développement soit au plus près des attentes des experts-comptables.



Le product backlog et le Kanban :

Le **product backlog** est l'outil de travail principal du Product Owner et n'est autre qu'une liste ordonnée de tâches à effectuer par l'équipe. Il recueille toutes les caractéristiques du produit et évolue avec celui-ci.

Kanban est une méthode de gestion du workflow créée afin d' aider à visualiser le travail avec des tâches ayant un des statuts suivants :



Capture d'écran de mon **Scrum board** du GitLab : **Kanban et product backlog**, utilisé comme Trello :

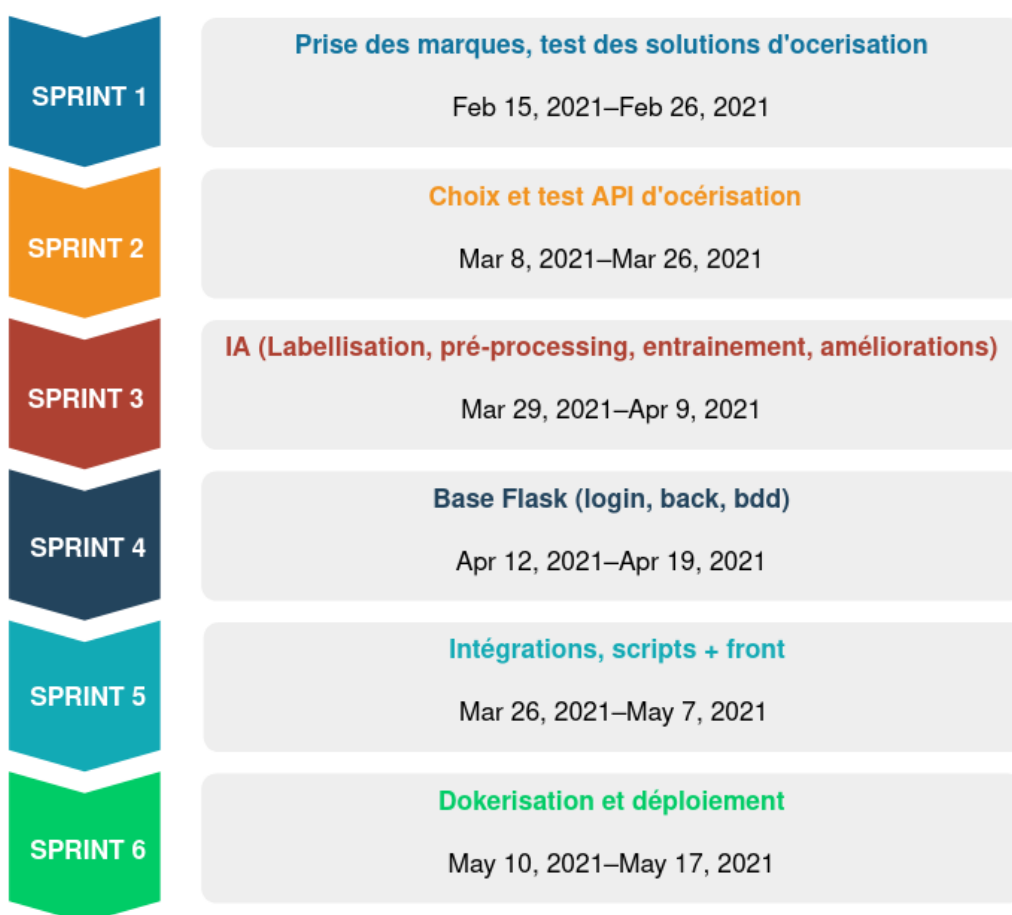
La capture d'écran montre un tableau Kanban GitLab avec cinq colonnes correspondant aux statuts du workflow :

- backlog** (10 tâches) : Générer un fichier d'écritures bancaires (#41), Enregistrer les résultats dans une base de données (#40), Créer un historique (#39), Affichage du résultat sous forme de tableau (#34), Limiter l'inscription aux seuls utilisateurs EXCO (#37), L'utilisateur doit pouvoir corriger ou compléter le résultat d'OCR (#35), Utiliser une API pour océriser (#31), Envoyer le mot de passe par email (#38), Extraire les données principales (#33).
- To Do** (9 tâches) : Refaire la labellisation Doccano (#30), Mise en forme des résultats ocr (#29), Mise en forme historique (#28), Message Flash à rétablir (#27), script d'écriture des écritures bancaires (#23), Déploiement sur aws beanstalk (#16), Dockerisation du projet (#15), Stockage des données d'agora dans une base de données (#12), back office de gestion des users (#9).
- Doing** (4 tâches) : labellisation des champs obligatoires dans une facture (#20), Script spacy, avec apprentissage (#21), récupération des données de la base agora (#10), Récup le numéro de dossier agora pour la requete permettant de récupérer les informations de niveau 1 de base de données (#22).
- waiting validation** (4 tâches) : nettoyages des données provenant d'agora (#11), login flask (#7), Gestion des différents formats de factures déposés par l'utilisateur (#14), Protéger l'accès avec une fonction login (#36).
- Closed** (14 tâches) : Reprendre la sélection de factures qui sont blanches suite au mauvais enregistrement (#24), Enregistrement du retour d'océrisation sous forme de txt (#19), Recueil des besoins auprès des utilisateurs comptables (#18), Intégration dans flask (#6), Test Mindee api (#2), test api google document ia (#5), Test api AWS Retour texte (#1), test api AWS retour tables (#3), aws api retour key/values (#4).

d. Les sprints

Les sprints sont les **grands temps forts** du projet. J'ai également utilisé GitLab pour gérer cette partie.

Il y a eu 6 grands sprints, de durée allant de 1 semaine à 3 semaines pour s'adapter au mieux à la prévision.



3. Lancement et imprégnation

Pour bien comprendre un nouveau sujet, et **monter en compétences**, il est primordial de **s'imprégner du sujet** et de la matière avec laquelle on va devoir travailler. Pour cela, j'ai procédé de la façon suivante :

- Immersion au sein du service comptabilité,
- Apprentissages des termes spécifiques au domaine de la comptabilité,
- Observation et tests des outils existants,
- Recueil des actions réalisées par le collaborateur comptable,
- Questionnement sur ce qui plaît dans le logiciel existant,
- Questionnement sur les besoins du métier au quotidien.

Le **besoin de l'utilisateur** final doit toujours rester au **centre de la préoccupation** du développeur, car aucune autre personne n'est mieux placée que l'utilisateur final pour savoir ce dont il a besoin. Cependant, le développeur peut et doit être force de proposition, d'amélioration, mais toujours en tenant compte des retours qu'il peut avoir.

Cette "mise en route" représente le sprint 1 et a duré environ 2 semaines.

III. Choix des technologies d'Intelligence Artificielle

Plusieurs technologies d'Intelligence Artificielle (IA) ont été utilisées pour mener à bien ce projet, et ainsi, répondre à l'expression du besoin exprimé.

1. Océrisation

a. Choix de l'API

Comme je l'ai indiqué préalablement, une des seules contraintes techniques qui m'a été imposée était de "ne pas réinventer ce qui existe déjà et fonctionne très bien dans l'océrisation d'un document". L'utilisation d'**une API était donc fortement suggérée** pour cette partie.

Pour répondre à la "brique" océrisation, j'ai réalisé **un large benchmark**, et de nombreux tests de différentes API disponibles sur le marché. In fine, j'ai opté pour le service **Amazon Textract**. En effet, celui-ci m'a semblé correspondre le mieux aux besoins et aux attentes liés au projet.

Amazon Textract est un service de machine learning qui extrait automatiquement du texte, de l'écriture manuscrite et des données à partir de documents numérisés. Ses avantages sont les suivants :

- Un des coûts les plus bas du marché :

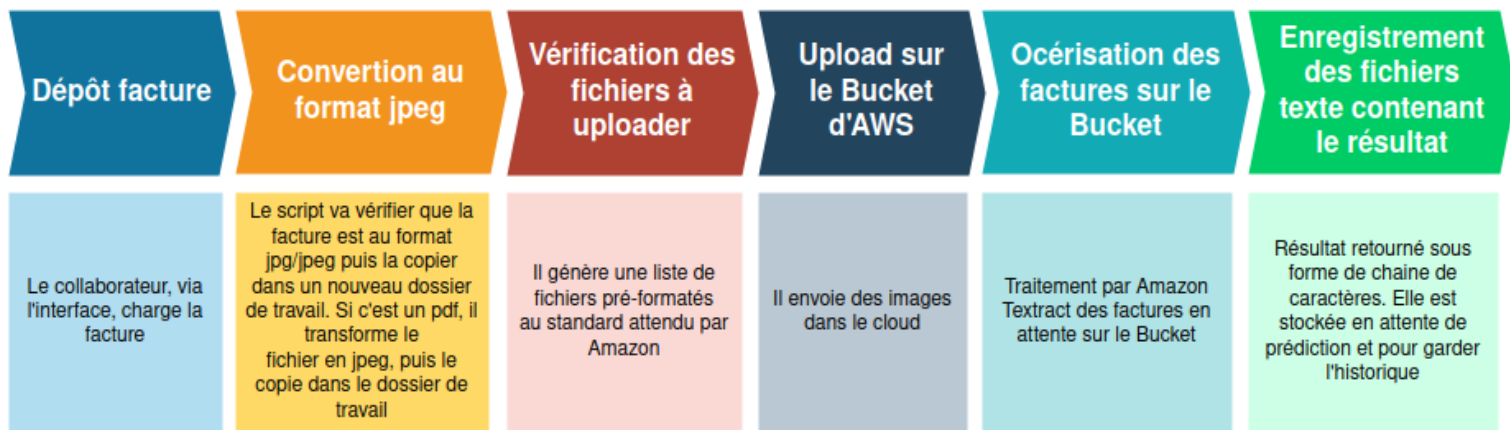
Mensuellement	Tarif par page	Prix pour 1 000 pages
Premier million de pages	0,001755 USD	1,755 USD
Plus d'un million de pages	0,000702 USD	0,702 USD

- Une accuracy excellente,
- Un script relativement court et rapide à mettre en place,
- Un retour des résultats sous forme de TXT exploitables plus facilement.

Vous pouvez retrouver en **Annexe 2** le détail des différentes API testées, les sources, ainsi que le code utilisé et ayant permis la réalisation des différents tests.

b. Fonctionnement Amazon Textract et preprocessing des factures

Afin de réaliser une océrisation simple et qualitative, et conformément aux besoins des utilisateurs, je commence par procéder de la façon suivante :



Retrouvez le code Amazon Textract et preprocessing des factures en **Annexe 3**

2. Natural Language Processing

Les technologies permettant d'arriver au but de notre projet sont multiples. Il existe **différentes méthodes, différentes librairies**. Mon choix s'est arrêté sur **SpaCy**, qui est une librairie de Natural Language Processing (NLP) très puissante permettant d'effectuer des opérations d'analyse de textes dans plus de 50 langues. Cette technologie permet aux machines de comprendre le langage humain grâce à l'**Intelligence Artificielle**.

SpaCy est une librairie contenant du **Deep Learning** et est connue pour sa vitesse d'exécution, ce qui la rend idéale pour une utilisation en production.

Actions réalisées par **SpaCy** :

- Tokenization (action de découpage d'un texte en série de "tokens" individuels)
- NER, Reconnaissance d'entités nommées (cela consiste à rechercher des mots (labels), ou groupes de mots catégorisables dans des classes telles que le numéro de facture, la date, le montant hors taxe, etc.)

Dans le projet, c'est la fonction **NER** qui nous intéresse tout particulièrement. C'est un processus d'**identification automatique des entités dans un texte**, tel que «personne», «organisation», «emplacement», dans la version livrée de base avec les modèles pré-entraînés. La bibliothèque SpaCy nous permet de former des modèles NER spécifiques en mettant à jour un modèle SpaCy existant pour l'adapter à un nouveau contexte de documents et de textes, mais aussi de **former un nouveau modèle NER de A à Z**. Ce dernier, est la solution que j'ai choisie car préconisée par SpaCy dans notre cas.

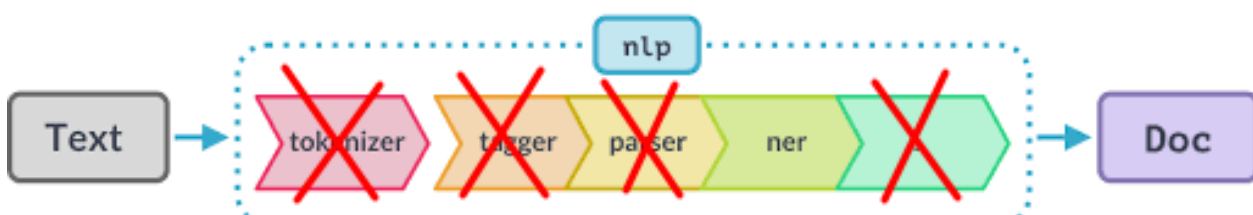
Dans SpaCy, la **reconnaissance d'entité nommée** est implémentée par le pipeline composant NER. La plupart des modèles l'ont dans leur pipeline de traitement par défaut. Le NER réalise des tâches de NLP.

Exemple de chargement d'un modèle existant et des différents éléments du pipeline :

```
# Load a spacy model
import spacy
nlp=spacy.load('fr_core_news_sm')
```

```
nlp.pipe_names
['tok2vec', 'morphologizer', 'parser', 'ner', 'attribute_ruler', 'lemmatizer']
```

Pipeline SpaCy:



Ici, je n'ai besoin que du NER, et en l'occurrence, d'un NER personnalisé, je crée donc un modèle vide, puis je l'ajoute à mon pipeline :

```
import spacy
# intitialisation from a blank pipeline
nlp = spacy.blank('fr')
```

```
# create ner if not exist
if "ner" not in nlp.pipe_names:
    ner = nlp.add_pipe('ner')
else:
    ner = nlp.get_pipe("ner")
```

Ensuite, il faut ajouter au NER, les données permettant d'entraîner le modèle (ici TRAINING_DATA):

```
#add all entities from training data in the ner
for m,n in TRAINING_DATA:
    for ent in n.get("entities"):
        ner.add_label(ent[2])
```

IV. Données d'entraînement du modèle

SpaCy, est un outil performant et puissant. Mais il faut lui donner **de la matière pour qu'il puisse apprendre**, comprendre, et ensuite retrouver les informations que l'on considère comme essentielles à notre tâche, dans un document. L'entraînement d'un modèle est donc utile quand on a des exemples et que l'on souhaite que notre système puisse **généraliser** sur la base d'exemples.

1. Création de données d'entraînement pour SpaCy

Labellisation :

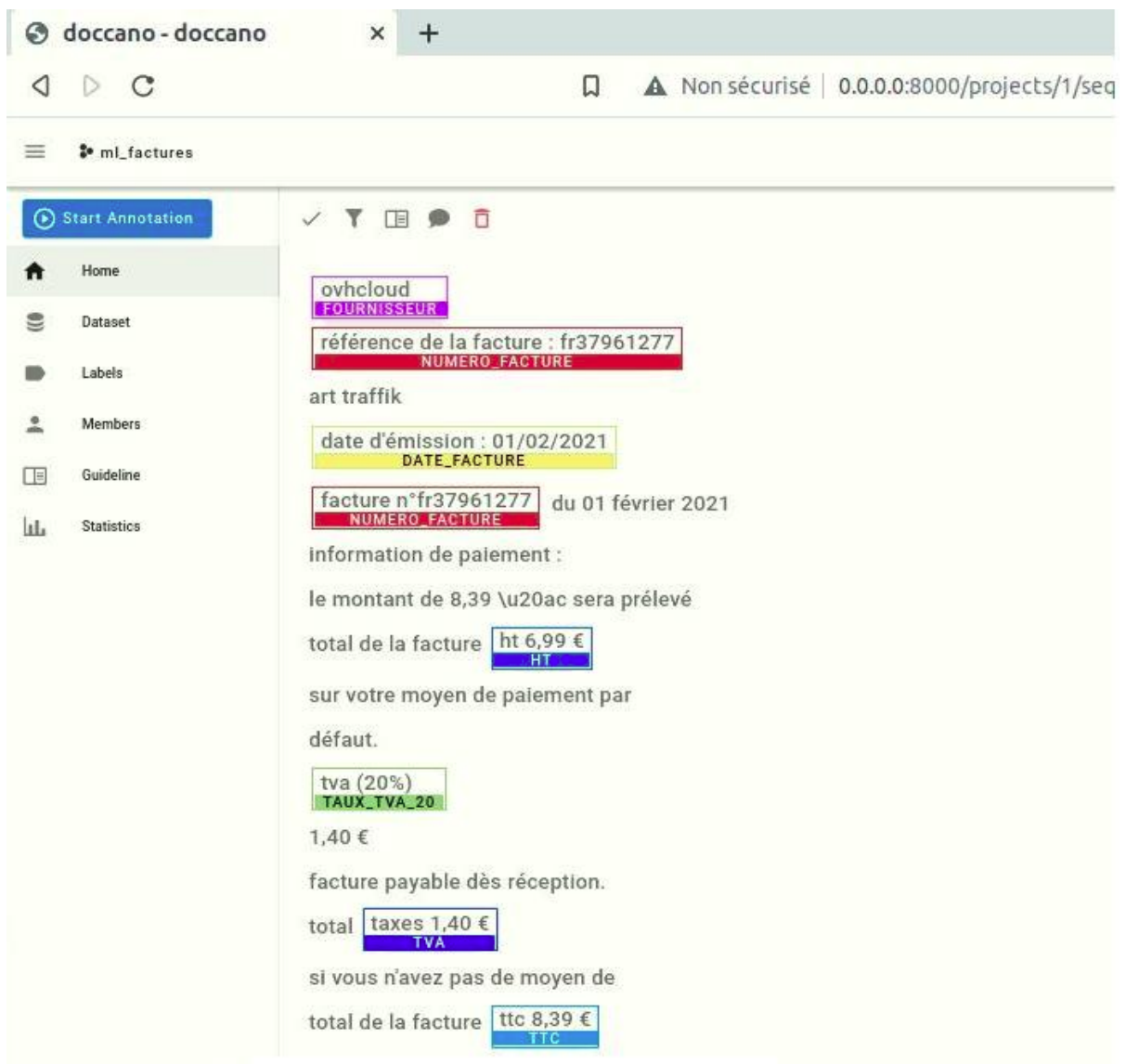
Le but de la labellisation est de **sélectionner les entités** qui sont "intéressantes" pour notre projet. Dans notre cas : la labellisation des champs obligatoires dans une facture. Pour cela, j'ai eu recours à un outil très connu pour cette fonction : **Doccano**. Avant d'arriver à ce choix, j'ai également procédé à des tests sur UBIAI (gratuit pendant une période d'essai de 14 jours).

Doccano cumule plusieurs avantages. Il est puissant, gratuit, et il permet **un export presque formaté** idéalement pour l'utiliser avec SpaCy. De plus, il s'installe en local depuis un dépôt GitHub. C'est un Web service local en mode SaaS.

Cf **Annexe 11** pour exemple UBIAI

Passage à Doccano, car version plus simple (locale), gratuite, et pérenne.

Exemple de labellisation avec DOCCANO :




Cette labellisation est très chronophage, mais elle est **déterminante pour la qualité de l'apprentissage de SpaCy** et donc des résultats de prédiction.

Difficultés rencontrées :


L'un des inconvénients est la lecture en ligne, de gauche à droite. Il devient parfois difficile de labelliser le document. En effet, cette labellisation ne peut se faire que par sélection d'éléments à la suite, alors qu'ils sont souvent décalés (exemple, dans un tableau on lit les informations également de haut en bas) :

Facture à gauche / résultat à droite :



Société Civile à Capital Variable - 775 875 739 RCS Nanterre
N°SIRET 775 875 739 03 131 - N° de TVA intracommunautaire : FR 42 775 875 739
329 av. Chateaux de Gaulle - 92028 NEUILLY SUR SEINE CEDEX - FRANCE

www.sacem.fr



Note de Débit numéro : 0120010347391

Date d'émission le : 06/06/2020

Vos références à rappeler

LE 35
SARLE LE 35
35 RUE MAZAGRAN
64200 BIARRITZ

Numéro de client : 27813765 N° de compte : 30001395972

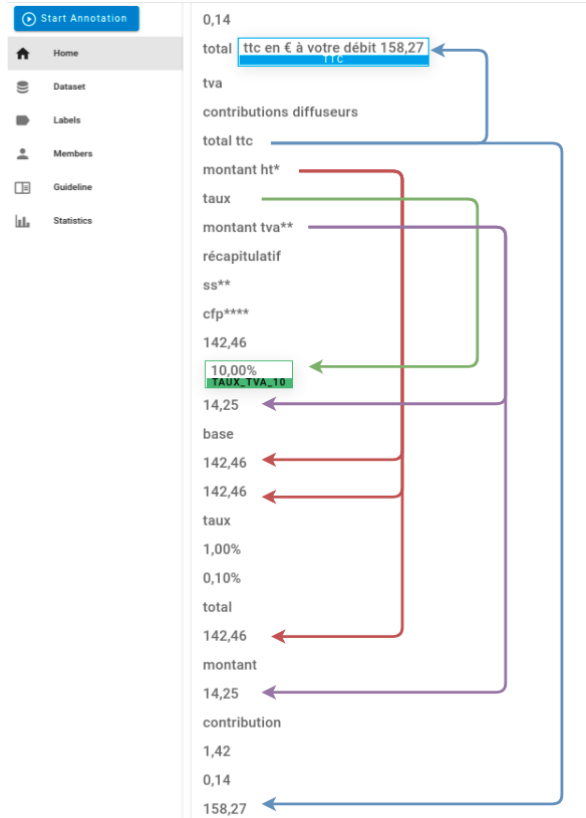
Téléphone : 06 87 34 82 39
Courriel : d.barrat@sacem.fr
Résidence Rapraut, 3 bis avenue de la Merne
B P 194 64004 BIARRITZ CEDEX

Paiement comptant avant le 30/06/2020.
Pas d'escompte pour paiement anticipé.

Désignation	Montant Hors Taxes*	TVA**	SS***	CFP****
Commence de détail	142,46	10,00%	1,00%	0,10%
Credite d'impôt du contrat 02-1000001605-10 Le montant de droit d'auteur hors taxes est déduit en fonction des éléments déduits dans l'annexe remontée avec la présente note pour la période du 01/04/2020 au 31/03/2021				
Total en €				
	142,46	14,25	1,42	0,14
Total TTC en € à votre Débit				
				158,27

Récapitulatif	Montant HT*	TVA Taux	Montant TVA**	SS***	CFP****	TOTAL TTC
	142,46	10,00%	14,25	Base	142,46	
				Taux	1,00%	
TOTAL	142,46		14,25	Montant contribution	1,42	0,14
						158,27

PAGE 1 / 1



Une **autre difficulté**, et pas des moindres, est l'alignement des labels. C'est un impératif. Il est primordial que les labels soient **parfaitement alignés**, sans quoi, après le preprocessing, il peut y avoir des décalages entre le label et l'emplacement annoncé pour SpaCy. Cela entraîne des **"erreurs W030"**. Malheureusement, il n'y a, à ce jour, que peu de solutions pour remédier à ce problème de façon automatique. En théorie, il ne devrait pas y avoir d'erreur de décalage du fait de l'utilisation d'un logiciel spécifique à cette tâche.

Exemple de décalage :

Fichier :

```
{'text': 'test nom fournisseur facture'}
{'entities': [(5, 20, 'FOURNISSEUR')]}
```

Ici, nous sommes bien alignés :

```
FOURNISSEUR = nom_fournisseur
```

Exemple de label non aligné :

```
{'entities': [(7, 22, 'FOURNISSEUR')]}
```

Ce qui donne un mauvais résultat :

```
FOURNISSEUR = m_fournisseur f
```


Exemple d'erreur W030 :

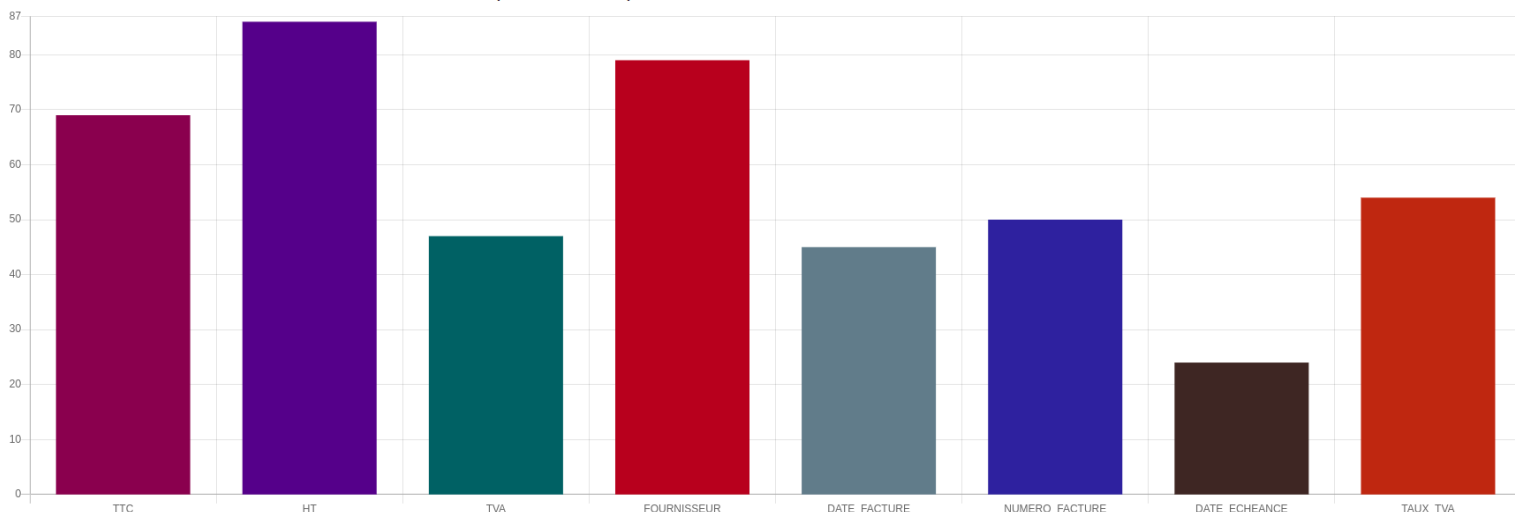
```
Entrée [15]: # add text and annotation for prepare the training of model
examples = []
for text, annots in TRAINING_DATA:
    examples.append(Example.from_dict(nlp.make_doc(text), annots))

/home/ubuntu/.local/lib/python3.8/site-packages/spacy/training/iob_utils.py:139: UserWarning: [W030] Some entities could not be aligned in the text "bergouey ind act vd carrieres lieu-dit camy la..." with entities "[ (22, 31, 'FOURNISSEUR'), (48, 55, 'FOURNISSEUR'), ... ]". Use `spacy.training.offsets_to_biluo_tags(nlp.make_doc(text), entities)` to check the alignment. Misaligned entities ('-') will be ignored during training.
warnings.warn(
/home/ubuntu/.local/lib/python3.8/site-packages/spacy/training/iob_utils.py:139: UserWarning: [W030] Some entities could not be aligned in the text "frans bonhomme 001203 (000789) - 0001/0001 8 rue..." with entities "[ (0, 15, 'FOURNISSEUR'), (261, 282, 'DATE FACTURE'), ... ]". Use `spacy.training.offsets_to_biluo_tags(nlp.make_doc(text), entities)` to check the alignment. Misaligned entities ('-') will be ignored during training.
```

Voici la **liste des labels**, donc des **informations essentielles**, que l'IA doit réussir à extraire:

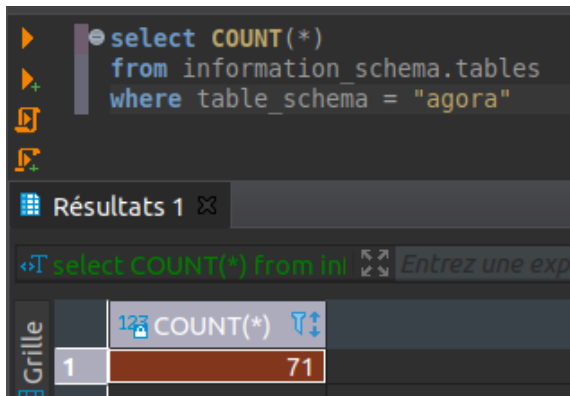
<input type="checkbox"/>	Name	Shortcut	Color
<input type="checkbox"/>	TTC	0	#2196F3
<input type="checkbox"/>	HT	1	#2148F3
<input type="checkbox"/>	TVA	2	#3A21F3
<input type="checkbox"/>	FOURNISSEUR	3	#A821F3
<input type="checkbox"/>	DATE_FACTURE	4	#FFEE58
<input type="checkbox"/>	NUMERO_FACTURE	5	#D32F2F
<input type="checkbox"/>	DATE_ECHEANCE	6	#263238
<input type="checkbox"/>	TAUX_TVA_10	7	#66BB6A
<input type="checkbox"/>	TAUX_TVA_20	8	#9CCC65
<input type="checkbox"/>	TAUX_TVA_55	9	#AED581
<input type="checkbox"/>	TAUX_TVA_0	a	#C5E1A5

Répartition des labels dans la labellisation :



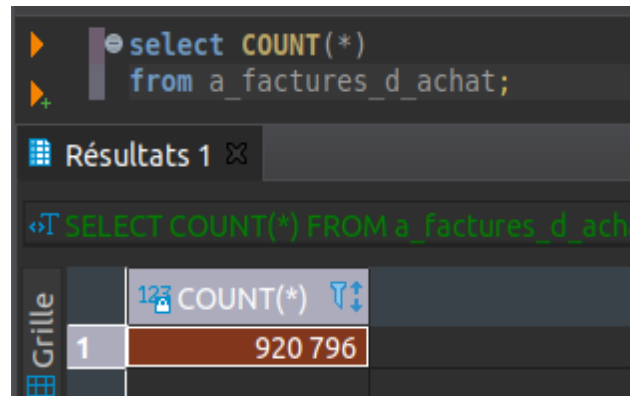
2. Utilisation de données existantes

Pour cela, j'ai eu accès à la base de données du logiciel AGORA. Malheureusement, cette base de données n'a pas pu être autant exploitée que ce que nous avons pu l'imaginer au début du projet. En effet, celle-ci contient **71 tables**, et, pour certaines, plus de **920000 lignes**.



```
select COUNT(*)
from information_schema.tables
where table_schema = "agora"
```

Grille	1
COUNT(*)	71



```
select COUNT(*)
from a_factures_d_achat;
```

Grille	1
COUNT(*)	920 796

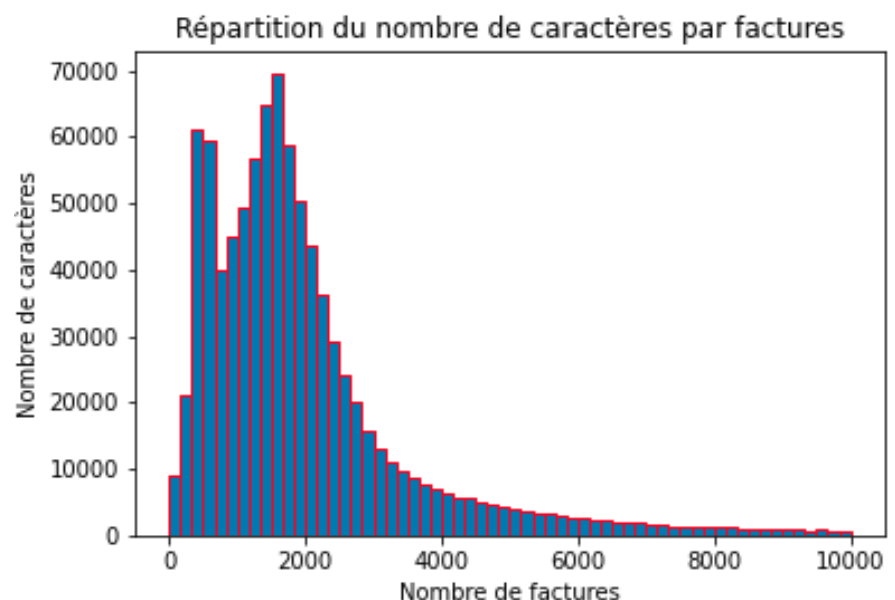
Cependant, cette base de données est **une sorte d'historique**, où l'on retrouve une multitude d'informations, mais pas celles dont nous avons besoin :

- Le contexte, c'est-à-dire les chaînes de caractères avant et après celles recherchées pour faciliter la recherche (pour TTC, HT, HT, etc.)
- L'emplacement des labels dans le document

Quelques chiffres sur les données existantes dans la base de données AGORA :

En moyenne, sur les **923047** factures scannées dans la base (depuis sa création), en moyenne chaque facture compte **2569** caractères.

Voici, ci-contre, une répartition du nombre de caractères par factures:



herr herrard
mariotte rd
1 xl
bergkamen lieferanschrift
florian mas christoph spalek
de mariotte
avenue privee
spalex großhandel
versand versichert
nr auftragsnummer
präsidentenstr bergkamen
spalexgrosshandel de
frankreich spalex
info spalex
sells herr
rechnung
rechnung nr
rd lattes
mas de
versandart
telefon telefax
lattes frankreich
bergkamen deutschland
de mariotte
avenue privee
spalex großhandel
präsidentenstr bergkamen
spalex
herrard
florian
versand
de avenue
privée sixième

V. Entraînement du modèle

L'**entraînement du modèle** est une étape assez rapide contrairement à la préparation du dataset d'entraînement. Il s'effectue en **quelques lignes de code**. Il faut compter de quelques minutes à quelques heures, en fonction du **nombre d'itérations**. La majeure partie du travail est bien évidemment **en amont de cet entraînement**, pour le preprocessing global.

Pour lancer l'entraînement du modèle, il faut préalablement charger le fichier JSON de labellisation, issu de Doccano, puis, il faut **construire le dataset d'entraînement** ainsi :

```
TRAINING_DATA = []
for entry in labeled_data:
    entities = []
    for e in entry['labels']:
        entities.append((e[0], e[1], e[2]))
    text_text = texte_processing(entry['text'])
    spacy_entry = (text_text, {"entities": entities})
    TRAINING_DATA.append(spacy_entry)
```

On applique un premier niveau de pré processing :

```
def texte_processing(x):
    x= x.replace("\n", " ").replace("ndeg", "n°").replace("é", "e")
    x= x.replace("è", "e").replace("ê", "e").replace("à", "a").lower()
    return x
```

Initialisation du modèle et création du NER :

```
from spacy.scorer import Scorer
import spacy
# intitialisation from a blank pipeline
nlp = spacy.blank('en')
scorer = Scorer(nlp)
# create ner if not exist
if "ner" not in nlp.pipe_names:
    ner = nlp.add_pipe('ner')
else:
    ner = nlp.get_pipe("ner")
```

Ensuite, on ajoute des labels :

```
#add all entities from training data in the ner
for m,n in TRAINING_DATA:
```

```
for ent in n.get("entities"):
    print(ent[2])
    ner.add_label(ent[2])
```

Entités fournies à SpaCy:

```
Entrée [17]: # list of entities of my ner
entities
```

```
Out[17]: [(0, 26, 'NUMERO FACTURE'),
          (28, 50, 'DATE FACTURE'),
          (51, 81, 'NUMERO FACTURE'),
          (265, 274, 'TAUX TVA 20'),
          (248, 264, 'FOURNISSEUR'),
          (284, 303, 'TVA'),
          (305, 319, 'TTC')]
```

Ensuite ajout des **données d'exemples** (texte + position des labels), **soit 40** factures labellisées **pour la totalité du dataset** (train = 35 et test = 5)

```
# add text and anotation for prepare the training of model
examples = []
for text, annots in TRAINING_DATA:
    examples.append(Example.from_dict(nlp.make_doc(text), annots))
```

Lancement de l'entraînement avec **90 itérations** :

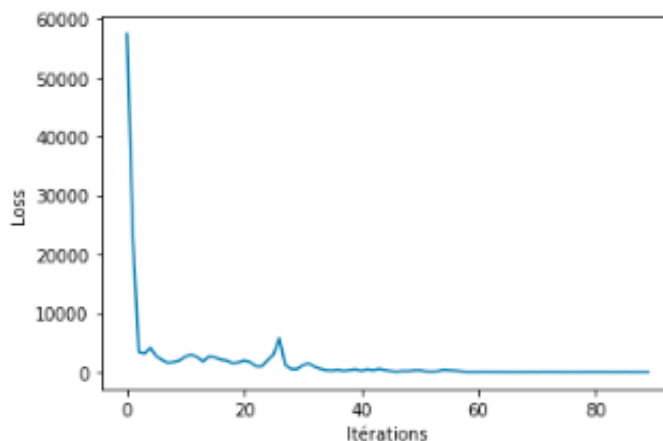
```
# training model
nlp.initialize(lambda: examples)
# Training for 90 iterations
loss=[]
for i in range(90):
    losses = {}
    random.shuffle(examples)
    for batch in minibatch(examples, size=8):
        #print(batch)
        nlp.update(examples, losses=losses)
    loss.append(losses['ner'])
    print("Losses", losses, 'i:', i)
```

```
Losses {'ner': 57455.06472015381} i: 0
Losses {'ner': 23410.005881786346} i: 1
Losses {'ner': 3429.2894279119355} i: 2
Losses {'ner': 3151.9781434510805} i: 3
Losses {'ner': 4119.302923798561} i: 4
Losses {'ner': 2687.7147589176893} i: 5
...
```

[...]

```
Losses {'ner': 36.69899589622722} i: 85
Losses {'ner': 14.583877049984748} i: 86
Losses {'ner': 11.56781003036312} i: 87
Losses {'ner': 17.043454009072185} i: 88
Losses {'ner': 13.81963490885396} i: 89
```

Grâce la **fonction loss**, on constate que le nombre d'itération est suffisant, ce qui démontre que le modèle apprend correctement, car on se rapproche de 0 :



Pour des documents comme des factures, le modèle a du mal à “s'accrocher” aux tokens et à faire de bonnes prédictions. Par conséquent, la perte est élevée au début des itérations puis chute assez rapidement avec l'apprentissage.

Scores :

Le code permettant d'arriver à ce résultat est disponible en **Annexe 11**

```
model_score(nlp, TEST_DATA)
```

	Precision	Recall	F1
0	0.811	0.997	0.883

Le **rappel** (Recall) est la proportion de résultats positifs réels qui ont été correctement identifiés.

La **précision** est la proportion d'identifications positives qui était effectivement correcte.

Le **score F1** est la moyenne pondérée de la précision et du rappel.

“Precision”, “Recall” et “F1” vont de 0 à 1, 1 étant le meilleur résultat (les scores peuvent varier d'un entraînement à l'autre et en fonction du découpage train / test / split).

Une fois l'entraînement terminé, je **sauvegarde le modèle**, pour son intégration dans Flask et sa **mise en production**, en utilisant la commande suivante :

```
1 #save model
2 nlp.to_disk('./dernier_model_spacy')
3 print('Modèle sauvegardé avec succès !')
```

VI. Base de données

Pour stocker les données fonctionnelles de l'application, j'ai décidé d'utiliser une base de données relationnelle. Ce système de BDD est un **ensemble de données organisées** sous la forme de tables et pouvant avoir des relations les unes avec les autres. Le système de gestion de base de données relationnelle (SGBDR) que j'ai choisi est SQLite. Il permet de **stocker les informations** liées au fonctionnement de mon prototype d'application.

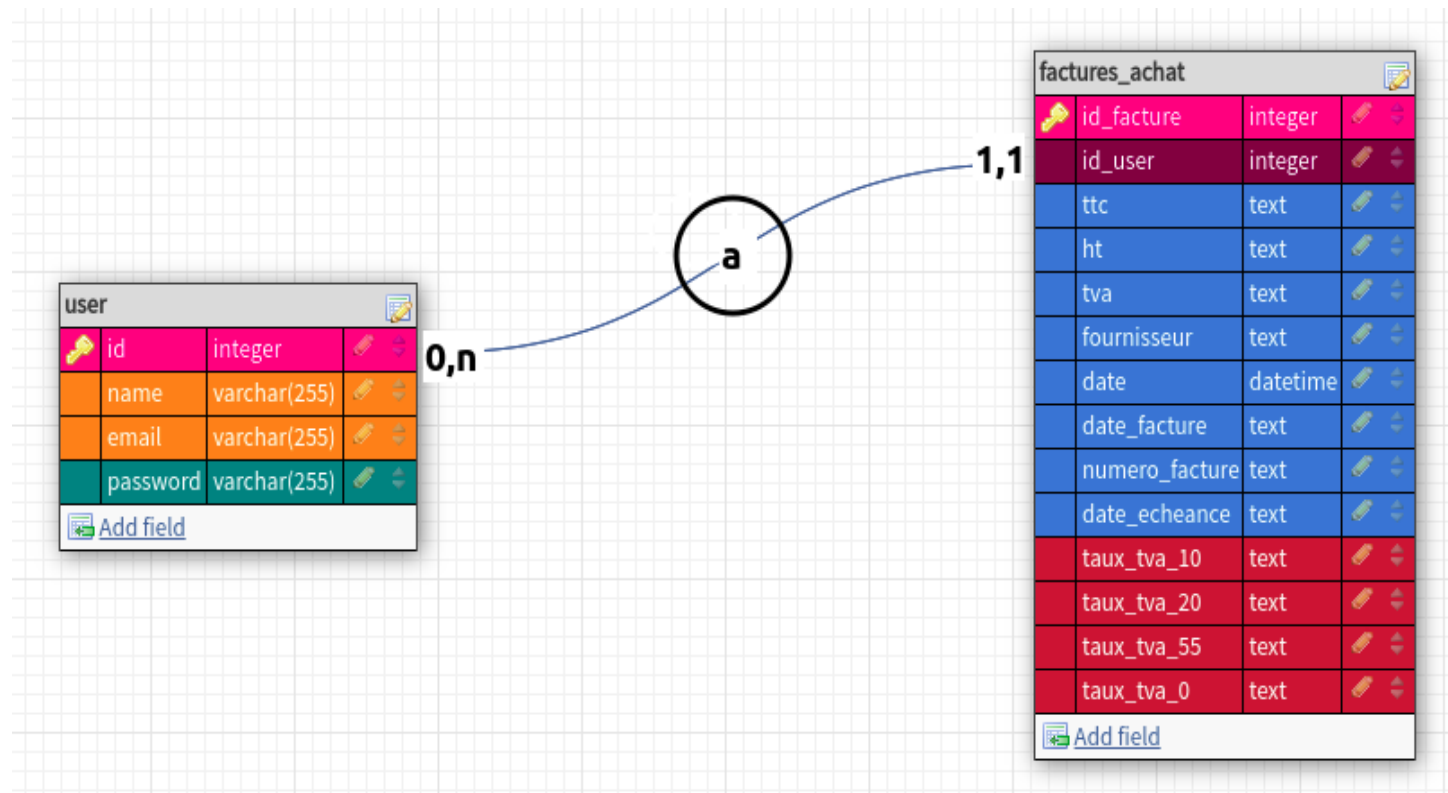
1. Structure de la base de données

La base de données du projet est constituée de 2 tables :

- Une table USER, qui permet de stocker les informations liées à l'utilisateur (**Nom, Mail, Mot de passe**)
- Une table factures_achat qui a vocation à stocker les informations d'océrisation des factures, de production, et d'historiques

Le code inhérent à la création des tables est en **Annexe 4**

Voici la représentation MCD (Modèle Conceptuel des Données) :



Pour plus de sécurité, les mots de passe ne sont pas stockés en clair mais **“hashés”** (sha256).

La cardinalité entre les 2 tables est formalisée par la présence de la **clé étrangère** “id_user” dans la table “factures_achat”, rattachée à la **clé primaire** “id” de la table “user”. Cela nous permet ainsi de stocker l'id de l'utilisateur logué, afin de connaître l'historique de ses traitements de facture.

2. Les requêtes SQL via SQLAlchemy

SQLAlchemy est un ORM (Object Relational Mapping), c'est-à-dire un type de programme informatique qui se place en interface **entre un programme applicatif et une base de données relationnelle**. Il s'agit donc d'un moteur de base de données permettant de prendre en charge des bases de données MySQL, ou encore SQLite. J'ai utilisé cette librairie car elle est **idéale avec Flask**. Pour ce qui est de la base de données, pour un prototype comme celui-ci, SQLite est suffisant et ne nécessite pas de serveur. Lors d'un déploiement vers une production, il est possible de migrer facilement vers une base de données MySQL, plus robuste, sans changer l'application.

Voici quelques exemples de quelques correspondances entre requête SQL et transposition via SQLAlchemy :

Définition	Requête SQL	SQLAlchemy
Inscription d'un nouvel utilisateur	INSERT INTO User VALUES (name, password , email);	user = User(email=email, name=name, password=password)* db.session.add(user) db.session.commit()
Récupération de l'adresse mail pour vérifier si un utilisateur est inscrit	SELECT email FROM User WHERE email LIKE email;	User.query.filter_by(email=e mail).first()
Récupérer l'id d'un utilisateur	SELECT id FROM User WHERE email LIKE email;	User.query.get(int(user_id))

3. Le modèle de base de données

Les classes **User** et **Factures_achat** créées héritent de **db.Model**. C'est la base pour tous les modèles de Flask-SQLAlchemy. Ces classes définissent **plusieurs champs comme variables** de classe. Les champs sont créés en tant qu'instances du db.Column pour communiquer avec la BDD.

Le script **model.py** contenant la classe **User** et la classe **Factures_achat** est disponible en **Annexe 5**

4. Back-up de la base de données

Il est impératif de faire des sauvegardes (Back-up) des BDD. Les sauvegardes fréquentes préviennent **la perte de données** et garantissent une **sécurité des informations à long terme**.

En matière de back-up la **règle des 3 2 1** est de rigueur :

- 3 copies de sauvegarde,
- 2 supports différents,
- 1 copie hors site

Pour une sauvegarde efficace, j'ai utilisé le planificateur de tâches **crontab** afin d'exécuter un script.sh de façon régulière. Ce script date automatiquement les sauvegardes (dump) :

```
#!/bin/sh
/home/ubuntu/anaconda3/bin/sqlite3
/home/ubuntu/EXCO/plateforme_flask_old_soldes-cfonb/my_app/db.sqlite .dump >
/home/ubuntu/EXCO/plateforme_flask_old_soldes-cfonb/my_app/backup_db_user/dbsqlite_backup-bdd-$(date +"%Y%m%d%H%M%S").sql_.dump.sql
```

VII. Intégration et utilisation du modèle dans l'application

Pour pouvoir utiliser notre modèle entraîné, je l'ai intégré au **script contenant les principales fonctions et outils : utils.py** de mon Flask. On y retrouve les outils que j'ai implémentés, permettant le fonctionnement de l'application :

```
import spacy
# Load model at startup to make it faster for the rest
nlp = spacy.load('./dernier_model_spacy')
print('Modèle chargé avec succès dans utils!')
```

Une fois le modèle chargé, il peut être utilisé pour faire des prédictions sur les nouvelles factures à traiter.

La fonction permettant le traitement est en **Annexe 6**.

Cette fonction de traitement de la facture renvoie un dictionnaire avec les entités retrouvées, ayant la plus forte probabilité de correspondre aux labels recherchés. Voici le format de ce dictionnaire :

```
list_of_results= {"TTC":"","  
                  'HT'":"","  
                  'TVA'":"","  
                  'FOURNISSEUR'":"","  
                  'DATE_FACTURE'":"","  
                  'NUMERO_FACTURE'":"","  
                  'DATE_ECHEANCE'":"","  
                  'TAUX_TVA_10'":"","  
                  'TAUX_TVA_20'":"","  
                  'TAUX_TVA_55'":"","  
                  'TAUX_TVA_0'":""}
```

Ce dictionnaire permettra de **pré remplir** le tableau des résultats.

VIII. Maquettage et parcours Utilisateur

Un parcours très simple, a été prévu de la sorte, pour simplifier le parcours de l'utilisateur (parcours UX):



Schéma fonctionnel des grandes étapes de l'outil développé :



4. Maquettage de la solution proposée

Les **mockups** et les **captures d'écran du front end** de l'application sont disponibles en **Annexe 9**.

Important :

Dans le menu de l'application, il y a 2 onglets et donc 2 fonctionnalités supplémentaires qui ne sont pas expliquées dans ce rapport. Ce sont des modules que j'ai développés pour Exco et qui ne rentrent pas dans le cadre du projet. Je les ai fusionnés au projet pour pouvoir

centraliser les services proposés. Ces modules sont fonctionnels mais ne seront pas détaillés ici.

[Accueil](#) [Extraction de fichiers CFONB](#) [Transformation de fichiers OFX en CFONB](#) [Ml-Factures\(Agora\)](#) [Se déconnecter](#)

5. Back-end

Comme cela a déjà été évoqué, j'ai choisi d'utiliser le **framework Flask** en tant que **back end pour porter l'application**. C'est un outil fiable et flexible pour le développement d'interface web et du back end allant avec.

Flask fonctionne avec des **"routes"** (décorateurs) qui appellent des fonctions Python et des pages en html, pouvant être couplées à des framework comme BULMA (c'est mon cas) qui gèrent le **CSS**.

Vous trouverez en **Annexe 7 l'arbre du dossier du projet**.

L'implémentation de la structure fonctionnelle du prototype sous Flask a été réalisée grâce aux **principales librairies** suivantes :

- Blueprint pour gérer les pages accessibles aux utilisateurs logués ou non logués
- Flask login pour la gestion des logins utilisateurs
- Flask mail pour l'envoi du mot de passe
- SQLAlchemy pour la base de données (cf partie VI. sur la BDD)
- Sentry pour le monitoring (cf partie IX. sur le monitoring)

IX. Monitoring de l'application

Pour monitorer le projet, j'ai eu recours à Sentry, qui a l'intérêt d'être un outil puissant, et assez rapide à mettre en place.

```
sentry_sdk.init(  
    dsn="https://4fa34371cdb148df9eb7274622531eae@o571402.ingest.sentry.io/5721403",
```

```
integrations=[FlaskIntegration()],
# Set traces_sample_rate to 1.0 to capture 100%
# of transactions for performance monitoring.
# We recommend adjusting this value in production.
traces_sample_rate=1.0)
```

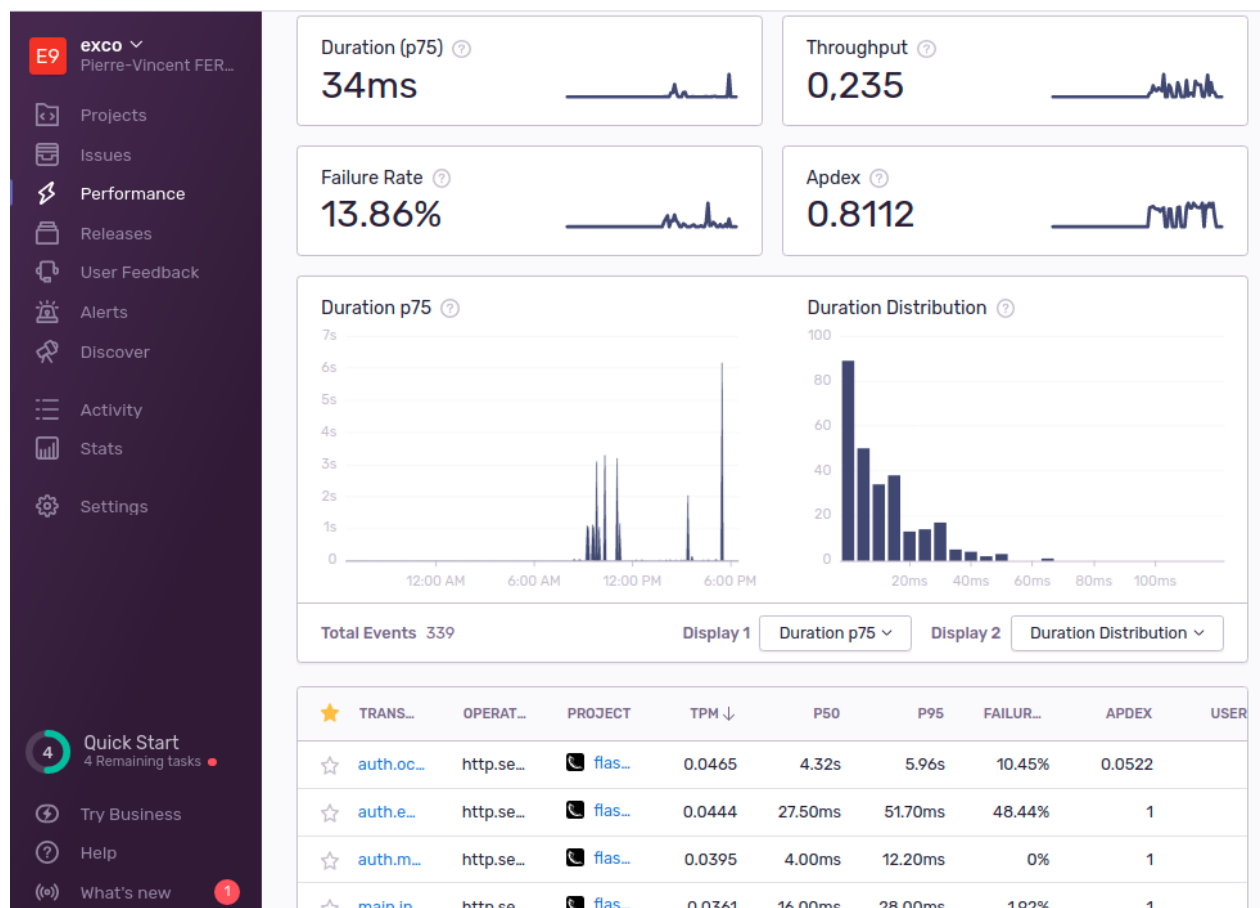
Ces informations sont fournies par la solution, et à intégrer à Flask, dans les paramètres de configuration.

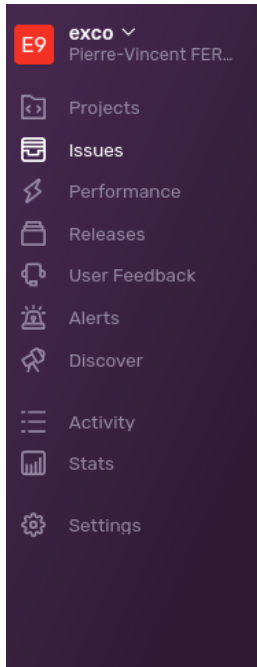
Les alertes sont paramétrables, et peuvent être reçues sous différents formats : par mail, SMS, appel, etc.

Cf **Annexe 8** pour voir un exemple de notification d'erreur envoyée par mail.

Le dashboard permet de suivre les performances de l'application :

Et de suivre les erreurs en temps réel :





		EVENTS	USERS	ASSIGNEE
<input type="checkbox"/>	<input checked="" type="checkbox"/> Resolve <input type="checkbox"/> Ignore <input type="checkbox"/> Mark Reviewed ...			
<input type="checkbox"/>	FileNotFoundError auth.ocr_texttract [Errno 2] No such file or directory: './retour...' FLASK-FB-22 Unhandled 16hr ago 16hr ol	1	0	v
<input type="checkbox"/>	FileNotFoundError auth.ocr_texttract [Errno 2] No such file or directory: 'factures' FLASK-FB-21 Unhandled 16hr ago 16hr ol	1	0	v
<input type="checkbox"/>	TypeError auth.history_invoices load_user() missing 1 required positional a... FLASK-FB-20 Unhandled 16hr ago 16hr ol	1	0	v
<input type="checkbox"/>	NameError auth.history_invoices name 'user_id' is not defined FLASK-FB-1Z Unhandled 16hr ago 16hr ol	1	0	v
<input type="checkbox"/>	TemplateSyntaxError auth.history_in... Encountered unknown tag 'endblock'. You ... FLASK-FB-TY Unhandled 17hr ago 17hr ol	1	0	v
<input type="checkbox"/>	TypeError auth.ocr_texttract The view function did not return a valid res... FLASK-FB-R Unhandled 18hr ago 5d old	5	0	v

X. Stratégie de tests

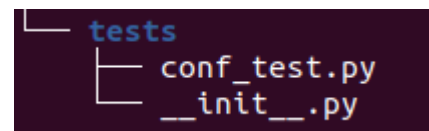
Les tests sont des procédures permettant de **vérifier le bon fonctionnement** d'une partie précise d'un logiciel ou d'une portion de script.

1. Tests unitaires

Dans mon cas, j'ai utilisé Pytest, qui est une **librairie consacrée aux tests**. Son objectif est de rendre l'écriture de code plus intuitive et est **préconisée** pour les développeurs travaillant sur Flask.

J'ai choisi de tester la disponibilité du service chez AWS, ainsi qu'une des fonctions de pre processing des factures.

Par convention, les scripts Python de test sont placés dans un dossier nommé test, à la racine du projet, comme illustré ci-contre.



En voici le script :

```
import pytest
from src.utils import *

@pytest.fixture
```

```

def app_context():
    with app.app_context():
        yield

def test_processing():
    result = texte_processing("Coucou \n Léon ndeg, , éndeg, jkhkj
l'été l'été")
    assert result == "coucou leon n°, , en°, jkhkj l'ete l'ete"

def test_aws(ACCESS_KEY='AKIAQ2VPDMHFJAWZDLKZ',
             SECRET_KEY='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',
             bucket_name='mlfacture'):
    session = boto3.Session(ACCESS_KEY, SECRET_KEY)
    assert session.region_name == 'eu-central-1'

```

Pour lancer les tests, il suffit de lancer la commande Pytest dans le terminal, pour obtenir les résultats. Ici les 2 tests sont passés :

```

(env_projet) ubuntu@ubuntu-Precision-Tower-3430:~/EXCO/ml-sur-factures/my_app_plateforme_flask$ pytest tests/
===== test session starts =====
platform linux -- Python 3.8.5, pytest-6.2.4, py-1.10.0, pluggy-0.13.1
rootdir: /home/ubuntu/EXCO/ml-sur-factures/my_app_plateforme_flask
plugins: flask-1.2.0
collected 2 items

tests/conf_test.py .. [100%]

===== 2 passed in 10.60s =====

```

2. Tests fonctionnels

Voici la liste des principaux tests fonctionnels de l'application :

L'utilisateur reçoit dans sa boîte mail un mot de passe généré automatiquement. Cela lui est signifié par un message Flash.
L'utilisateur reçoit un message d'erreur de type Flash quand il essaie de s'enregistrer avec une adresse mail autre que contenant @exco.fr
Le nom, l'adresse mail et le mot de passe hashé sont bien stockés dans la BDD.
Un utilisateur non logué n'a accès qu'aux onglets "se connecter" et "s'inscrire".
L'entraînement du modèle arrive bien à son terme.
Le modèle est bien chargé sur la plateforme Flask.

Le chargement de la photo se passe bien, ainsi que sont preprocessing et envoi vers AWS.

En cliquant sur le bouton “extraire les données” le chargement de la photo se passe bien, ainsi que sont preprocessing, l’envoi vers AWS, le retour sous forme de texte océrisé, l’analyse par le modèle et l’affichage des résultats sous forme de tableau pré rempli avec les informations ayant la plus haute probabilité.

L’enregistrement des données dans la base de données est bien effectué quand l’utilisateur clique sur “Enregistrer les données”.

L’historique de traitement des factures, pour l'utilisateur logué, s’affiche correctement quand l’utilisateur clique sur “ voir l’historique de traitement des factures”.

XI. Déploiement de l’application

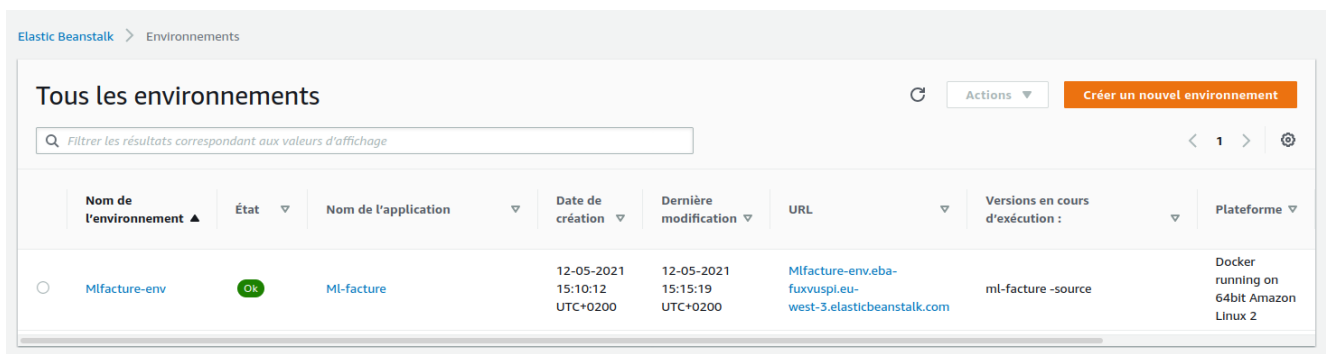
Afin de **déployer** l’application conformément aux standards des Dev-ops, j’ai décidé d’utiliser le logiciel Docker qui est une technologie permettant l’**utilisation de conteneurs Linux**.

Ces conteneurs sont comme des **machines virtuelles très légères** et modulaires que l’on peut créer, déployer, copier et déplacer d’un environnement à un autre, tout en gardant une configuration précise (versions de librairie). Ainsi, ces conteneurs sont des **environnements fonctionnels et stables** qui restent hermétiques aux interférences.

Vous trouverez en **Annexe 10** le dockerfile permettant la conteneurisation.

Une fois le conteneur éprouvé en local, il est temps de le déployer en ligne. Pour cela j’ai choisi d’utiliser le service **AWS Elastic Beanstalk** qui remplit parfaitement cette fonction.

Pour cela, il faut faire un **zip de l’application à la racine** du projet et d’y **inclure le dockerfile** ainsi que tous les fichiers et dossiers ayant permis la dockerisation en local, puis de simplement l’uploader au moment de la création la création de l’environnement



Nom de l'environnement ▲	État ▼	Nom de l'application ▼	Date de création ▼	Dernière modification ▼	URL ▼	Versions en cours d'exécution : ▼	Plateforme ▼
Ml-facture-env	OK	Ml-facture	12-05-2021 15:10:12 UTC+0200	12-05-2021 15:15:19 UTC+0200	Ml-facture-env-ebafuxvuspl.eu-west-3.elasticbeanstalk.com	ml-facture -source	Docker running on 64bit Amazon Linux 2

XII. Conclusion

La réalisation de ce projet a été pour moi la consécration de plus d'**un an et demi d'investissement personnel** dans ma reconversion professionnelle en tant que développeur en Intelligence Artificielle. Il m'a permis d'**apprendre** et de **mettre en application** beaucoup de principes informatiques et de développement vus au cours de la formation, mais aussi appris dans le contexte professionnel.

Le prototype, en tant que tel, est fonctionnel. J'espère qu'il apportera à Exco **le niveau d'information et de faisabilité nécessaire** afin de **prendre la décision de le pérenniser** en développant une version de production ; ou, au contraire, de choisir de continuer à utiliser la solution propriétaire déjà en place.

Comme toute intelligence artificielle, il est possible d'**améliorer indéfiniment la solution par des approches et des technologies différentes**. Il est alors nécessaire de procéder à un arbitrage entre le résultat attendu, et l'investissement (financier et humain) que l'on souhaite y consacrer.

Annexe 1 : Projet E3 – État de l'art

Reconnaissance automatique de caractères, appliquée aux factures

Pierre-Vincent FERRAT (12/2020)



Partie 1 - Sources

- <https://medium.com/@CharlesCrouspeyre/comment-les-r%C3%A9seaux-de-neurones-%C3%A0-convolution-fonctionnent-b288519dbcf8>

Medium est un site plutôt fiable, de manière globale. L'auteur de cet article, Brandon Rohrer est un spécialiste du Machine learning, de la Data science et de la robotique. Il est suivi par près de 13K personnes sur Twitter, ce qui lui donne de la crédibilité.

- <https://towardsdatascience.com/how-to-extract-text-from-images-with-python-db9b87fe432b?qi=86de0f4313d0#:~:text=The%20Python%20Library&text=Python%20Tesseract%20is%20an%20optical,for%20Google's%20Tesseract%20OCR%20Engine.>

Costas Andreou, l'auteur de cet article, est reconnu dans le monde de la data science. Il écrit régulièrement des articles sur towardsdatascience.com (branche de medium.com).

- <https://towardsdatascience.com/covolutional-neural-network-cb0883dd6529>

Dshahid380, l'auteur, a été remercié plus de 1300 fois sur towardsdatascience.com pour cet article.

- www.azopio.com
- <https://ipaidthat.io/>

Ces deux sites sont fiables. Ils proposent la vente de solutions d'océrisation.

- https://fr.wikipedia.org/wiki/Reconnaissance_optique_de_caract%C3%A8res

Wikipédia permet de bien comprendre des thématiques à la fois généralistes et spécialistes. Comme ce site est participatif et ouvert à la modification (par quiconque), il va de soi qu'il est nécessaire de s'en servir de base, puis de croiser les informations avec des sources que l'on considère plus fiables et plus spécifiques au secteur d'activité.

Partie 2 - OCR

Reconnaissance automatique de caractères, appliquée aux factures

La comptabilité n'échappe pas aux **nouvelles transformations numériques**. Et cela n'est sûrement pas une mauvaise chose pour ce secteur.

Cette évolution, touche principalement la tâche de saisie des écritures bancaires, issues des factures numérisées. Cette démarche, qui est longue et fastidieuse, est de plus, considérée comme sans valeur ajoutée. Heureusement, des **méthodes de reconnaissance automatique** permettent de numériser les documents administratifs et comptables, et d'exploiter les données qu'ils contiennent. Cela permet d'avoir **une comptabilité dématérialisée**, plus rapide et plus fluide.

Cette méthode d'automatisation permettant d'extraire les informations des documents numérisés est l'**océrisation**.

L'océrisation qu'est-ce que c'est ?

Concrètement, le terme océrisation signifie «Optical Character Recognition (OCR)». C'est donc un système qui fournit une **reconnaissance alphanumérique complète des caractères imprimés ou manuscrits, à partir d'images**. Elle est utilisée dans beaucoup de domaines comme la banque (pour la lecture des chèques), dans les centres de tri postaux (pour la lecture des adresses et le tri), mais aussi, et c'est le sujet qui nous concerne, pour la lecture des informations contenues dans les factures numérisées .

Pourquoi l'océrisation des factures ?

La raison est évidente, pour dématérialiser la chaîne de traitement, automatiser les tâches, et gagner du temps sur le traitement pour gagner en compétitivité.

Avantages et inconvénients de la reconnaissance automatique de factures

Avantage : Gain de temps, plus de papier à stocker, archivage numérique, contrôle d'informations en doublons ou contradictoires,...

Inconvénient : Il y a forcément quelques limites à ce genre de technologies. Les outils peuvent avoir du mal à lire certains supports numérisés, du fait de l'écriture dans plusieurs sens, peu lisible ou en superposition.

L'intervention humaine est donc nécessaire pour une partie du processus, que ce soit pour l'apprentissage initial, pour intervenir quand la machine n'arrive pas à «comprendre» des champs présents dans le document, ou plus tard, dans la phase succédant l'océrisation, pour les écritures comptables. Il faudra alors que la machine apprenne à ranger les informations au bon endroit pour que les écritures comptables soient enregistrées sur les bons comptes.

Mais, comme le dit l'expression pour les relations humaines, la confiance n'évite pas le contrôle. C'est également applicable pour le processus d'océrisation, puis lors du traitement de l'information par les diverses solutions. Le collaborateur devra contrôler régulièrement que le travail de la machine est correct pour ne pas engendrer des problèmes en chaîne par la suite.

Les services cloud clé en main

Des services cloud existent et proposent de réaliser ces automatisations. Il y a deux possibilités :

- uniquement lors de la phase d'océrisation, permettant de reconnaître les «blocs» de texte et de les sortir sous forme de chaîne de caractères à stocker dans une base de données pour utilisation extérieure,
- ou pour une chaîne complète du scan à l'écriture comptable.

Voici quelques exemples de plateformes proposant ces services complets :

- www.klippa.com
- www.azopio.com

«Azopio extrait automatiquement les données pertinentes des factures et reçus (Nom du fournisseur, Date, Numéro de facture, Devise, Montants TTC, TVA, HT) permettant une recherche puissante par mots clés ou valeurs et la génération automatique des écritures comptables. »



Il existe une multitude d'autres services en ligne qui gèrent toute la phase de

numérisation de la facture, jusqu'au résultat final : **l'écriture bancaire pour la comptabilité**.

Nous allons nous focaliser sur la partie océrisation pour la suite de cet état de l'art.

Tout en restant sur la partie cloud et si l'on souhaite mieux maîtriser la chaîne de travail, il existe les API des plateformes comme Azure notamment via ses «**Cognitive Services**». Elle propose un service d'extraction des données de ces documents grâce à l'API Vision. Le document numérique y est envoyé et en retour une chaîne de caractères en JSON est produite contenant les textes du document.

Les API les plus connues sont Google Vision, Amazon Rekognition et Microsoft Cognitive Services. Pour un développeur, elles sont assez simples à mettre en place, mais ces solutions sont plus techniques que la solution clé en main. On s'éloigne, de plus en plus, de la méthode en «clic bouton» avec les API, car celles-ci ne sont qu'une partie de la solution complète.

Avantages et inconvénients des services cloud

Rapide à mettre en place mais avec un **coût certain** lors de l'utilisation. Tout dépend de la sollicitation du service et si la ressource existe en interne.

La partie API sera la **plus adaptée**, à ce niveau de l'état de l'art, pour pouvoir ensuite traiter les chaînes de caractères qui auront été extraites des documents.

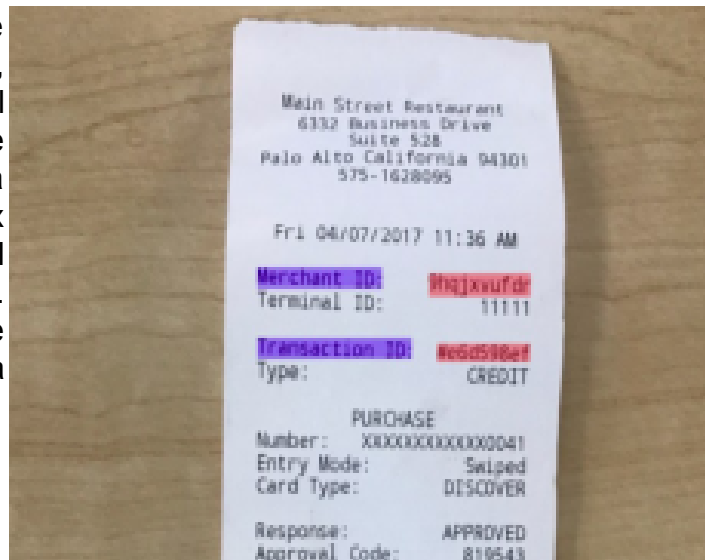
Pourquoi développer sa propre solution via des librairies ?

Cela permet de maîtriser l'intégralité de la chaîne, tout en s'appuyant sur les librairies performantes nous permettant d'arriver à notre finalité, sans être dépendant d'un service en ligne (coût, disponibilité, contraintes du service).

Il existe donc plusieurs outils pour « océriser » :

- Tesseract, qui est « un moteur de [reconnaissance optique de caractères](#) [...] Il permet déjà d'obtenir une reconnaissance optique de qualité sur un certain nombre de documents». C'est un outil complet, qui peut être utilisé seul en ligne de commandes, ou dans des scripts Python par exemple.
- OpenCV (pour Open [Computer Vision](#)) qui est une [bibliothèque graphique libre](#), initialement développée par [Intel](#), spécialisée dans le [traitement d'images](#) en temps réel. On l'utilise principalement en pré-processing des images.

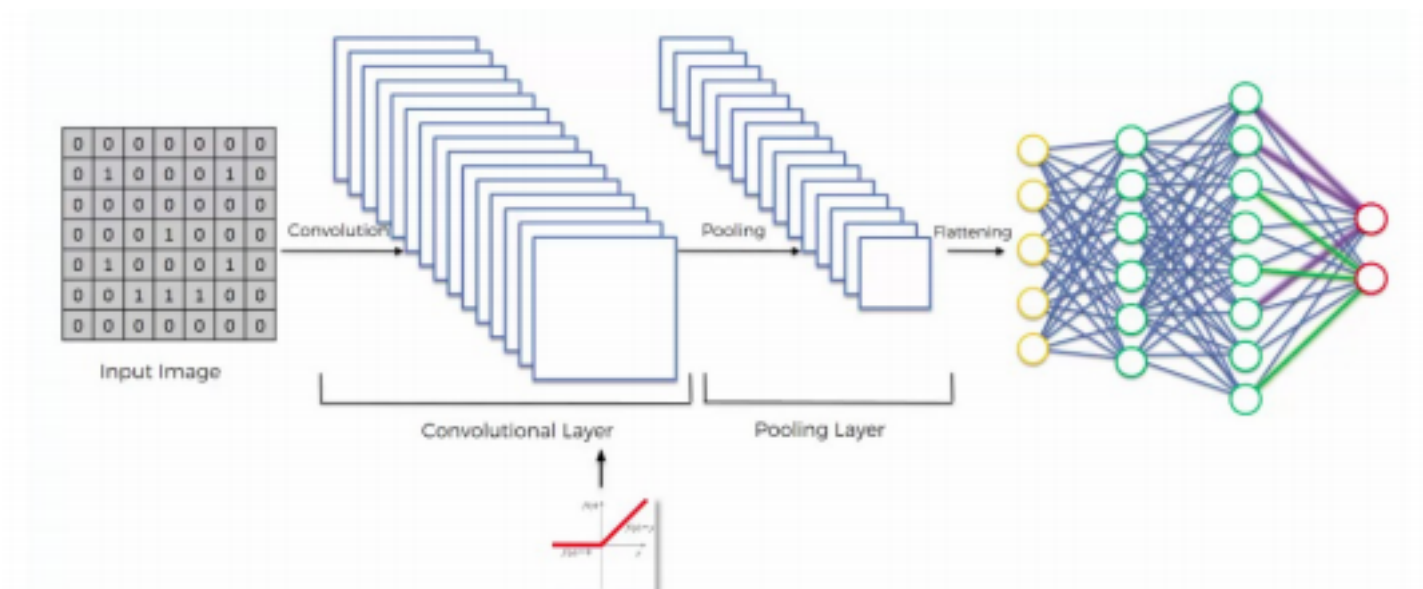
OpenCV permet également de créer les « boîtes de contour », autour des informations. Lorsque il est combiné à Keras, une bibliothèque permettant la constitution rapide de réseaux neuronaux, on obtient un outil puissant de lecture de documents. Cet outil permet, notamment, de créer un réseau de neurones à convolution (CNN).



Focus sur les CNN

Regardons comment **fonctionnent ces CNN** et le computer Vision, pour créer notre propre solution.

Schéma d'un CNN type :



Pour un ordinateur, une image n'est rien d'autre qu'un tableau de pixels en 2D (une sorte d'échiquier géant) avec chaque case contenant un numéro spécifique.

Le CNN prend ces «cases», et les compare, fragment par fragment. Les fragments qu'il recherche sont appelés les **caractéristiques**.

Chaque caractéristique est comme une mini-image, un petit tableau de valeurs en 2 Dimensions. Les caractéristiques rassemblent les aspects les plus communs des images, dans le but de **trouver les similitudes** avec les lettres apprises par le modèle.

Quand on lui présente une nouvelle image, le CNN ne sait pas exactement si les caractéristiques seront présentes dans l'image, ni où elles pourraient être. Il cherche donc à les trouver dans toute l'image et dans n'importe quelle position.

En calculant dans toute l'image si une caractéristique est présente, le CNN réalise un filtrage. Les mathématiques utilisées pour réaliser cette opération sont appelées une **convolution**, de laquelle les réseaux de neurones à convolution tiennent leur nom. L'étape de **pooling** (max pooling), permet d'alléger les «features» (caractéristiques), en ne conservant que les dominantes. On élimine le bruit qui ne contient pas d'information déterminante.

L'étape suivante est le **Flattening**. Il permet de mettre à plat les données pour les injecter dans le réseau de neurones à proprement parlé, dans la couche «fully connected». Cette dernière va réaliser les calculs de poids et de valeurs, pour donner, en résultat, une **liste de probabilités** pour le caractère lu.

Il est tout à fait possible de coder soi-même l'équivalent de ces bibliothèques en utilisant les CNN, mais il serait dommage, et coûteux en temps, de se priver de ces libraires puissantes où tout est déjà implémenté. La bibliothèque **Numpy** permet un affranchissement de toute autre technologie ou mise à jour. On pourrait ainsi avoir la main sur absolument tout. Toutefois, cette solution est fortement déconseillée.

Conclusion

L'océrisation est donc un procédé très puissant permettant de **recupérer le texte des images et documents scannés**. Que l'on souhaite une solution clé en main, ou une solution intermédiaire via des API ou encore un codage complet, le choix se fera en fonction des besoins, du coût, des ressources que l'on a à mettre sur le projet, et le temps que l'on souhaite y attribuer.

Annexe 2 : API d'OCR (quelques exemples)

1.1. AWS Textract

AWS Text detect with boto3:

```
import boto3

# Document
documentName =
"/home/ubuntu/EXCO/ml-sur-factures/env/my_app/factures-jpg/1.pdf6.jpg"

# Read document content
with open(documentName, 'rb') as document:
    imageBytes = bytearray(document.read())

# Amazon Textract client
textract = boto3.client('textract')

# Call Amazon Textract
response = textract.detect_document_text(Document={'Bytes': imageBytes})

#print(response)

# Print detected text
for item in response["Blocks"]:
    if item["BlockType"] == "LINE":
        print (item["Text"])
```

AWS key: values forms with boto3 :

```
import boto3
from trp import Document

# Document
s3BucketName = "ki-textract-demo-docs"
documentName =
"/home/ubuntu/EXCO/ml-sur-factures/env/my_app/factures-jpg/1.pdf6.jpg"
# Amazon Textract client
textract = boto3.client('textract')

# Call Amazon Textract
```

```

response = textract.analyze_document(
    Document={
        'S3Object': {
            'Bucket': s3BucketName,
            'Name': documentName
        }
    },
    FeatureTypes=["FORMS"])

#print(response)
doc = Document(response)
for page in doc.pages:
    # Print fields
    print("Fields:")
    for field in page.form.fields:
        print("Key: {}, Value: {}".format(field.key, field.value))

    # Get field by key
    print("\nGet Field by Key:")
    key = "Phone Number:"
    field = page.form.getFieldByKey(key)
    if(field):
        print("Key: {}, Value: {}".format(field.key, field.value))

    # Search fields by key
    print("\nSearch Fields:")
    key = "address"
    fields = page.form.searchFieldsByKey(key)
    for field in fields:
        print("Key: {}, Value: {}".format(field.key, field.value))

```

1.2. Azure:

Non testée faute de pouvoir y ajouter ma CB.

1.3. Rossum :

Pas vraiment d'API, ne retourne pas les informations. Il y a une interface graphique pour repérer les champs. Belle interface user friendly, mais pas adaptée au projet.

```
import requests

username = "pierre-vincent.ferrat@exco.fr"
password = "ud6f6prKBVim7Du"
queue_id = 91172
path = "/home/ubuntu/EXCO/ml-sur-factures/env/my_app/factures-jpg/5.jpg"

##### upload #####
url = f"https://api.elis.rossum.ai/v1/queues/{queue_id}/upload"
with open(path, "rb") as f:
    response = requests.post(
        url,
        files={"content": f},
        auth=(username, password)
    )
annotation_url = response.json()["results"][0]["annotation"]
print("The file is reachable at the following URL:", annotation_url)

##### get results #####
endpoint = 'https://api.elis.rossum.ai/v1'

import datetime
response = requests.post(
    f'{endpoint}/auth/login',
    json={'username': username, 'password': password}
)
if not response.ok:
    raise ValueError(f'Failed to authorize: {response.status_code}')
auth_token = response.json()["key"]

date_today = datetime.date.today()
date_end = date_today
date_start = date_today - datetime.timedelta(days=1)
response = requests.get(
    f'{endpoint}/queues/{queue_id}/export?format=xml &'
    f'exported_at_after={date_start.isoformat()}&'
    f'exported_at_before={date_end.isoformat()}&'
    f'ordering=exported_at&'
)
```

```

    f'page_size=100&page=1',
    headers={'Authorization': f'Token {auth_token}'}
)

if not response.ok:
    raise ValueError(f'Failed to export: {response.status_code}')

print(response.content)

##### written file #####
with open('response.xml', 'wb') as file:
    file.write(response.content)

```

1.4. Nanonets

Fonctionnement identique à celui de Rossum. Non adapté au projet.

1.5. Mindee

Résultats intéressants, mais ne correspondent pas exactement à l'idéal que nous recherchons, c'est-à-dire une simple océrisation structurée.

```

import requests
import json
path_to_file =
"/home/ubuntu/EXCO/ml-sur-factures/env/my_app/tmp/SIMPBAYKON2200903181401
e_0006.pdf"

def extract_datas_invoces(path_to_file):
    url = "https://api.mindee.net/products/invoices/v2/predict"

    with open(path_to_file, "rb") as myfile:
        files = {"file": myfile}
        headers = {"X-Inferuser-Token":
"7a7e831622c669e7b9e8aaf99a265c6d"}
        response = requests.post(url, files=files, headers=headers)
        #print(response.text)
    return response.text

def extract_all_datas(path_to_file):

```

```

url = "https://api.mindee.net/products/mindee_vision/v1/predict"

with open(path_to_file, "rb") as myfile:
    files = {"file": myfile}
    headers = {"X-Inferuser-Token":
"996b12efda2744193cc7509788cd2981"}
    response = requests.post(url, files=files, headers=headers)
    #print(response.text)
    return response.text

def liste_all_text_of_doc(retour_mindee_extract_all_datas):
    import pprint
    #pprint.pprint(retour_mindee_extract_all_datas)

    #longueur liste
    longueur_liste_extract_all_datas =
len(retour_mindee_extract_all_datas['predictions'][0]['all_words'])
    text_extract = []
    for i in range(0, longueur_liste_extract_all_datas):
        #print(a['predictions'][0]['all_words'][i]['text'])

text_extract.append(retour_mindee_extract_all_datas['predictions'][0]['all
words'][i]['text'])

    return text_extract # tout le texte de la facture sous forme de liste

#extraction des données principales de la facture
retour_mindee_extract_datas_invoces = extract_datas_invoces(path_to_file)
#extraction de toutes les informations de la facture
retour_mindee_extract_all_datas =
(json.loads(extract_all_datas(path_to_file))) # json to dict
#extraction de tout le texte, et juste le texte, issu de l'extraction de
toutes les informations de la facture
retour_liste_all_text_of_doc =
liste_all_text_of_doc(retour_mindee_extract_all_datas)

with open("sample.json", "w") as outfile:
    json.dump(retour_mindee_extract_all_datas, outfile)

```

1.6. ABBYY

Les résultats sont très bons. L'un des avantages est que ABBYY océrise en gardant une structure aux données. Comme la plupart des outils d'OCR, il lit en ligne, mais il ne va pas garder uniquement le texte. Il va insérer un espace à chaque zone blanche qu'il va trouver. Il en résulte un fichier texte lisible par un utilisateur. Cependant, dans notre projet, cette fonctionnalité n'est pas utile. Par ailleurs, le prix est plus élevé que AWS.

Tableau des prix :

\$29.99 per month	\$99.99 per month	\$199.99 per month	\$299.99 per month	\$839.99 per month	Contact sales
500 pages	2 000 pages	5 000 pages	10 000 pages	30 000 pages	Need over 30 000 pages per month?
———— \$0.06 per additional page	———— \$0.05 per additional page	———— \$0.04 per additional page	———— \$0.03 per additional page	———— \$0.028 per additional page	

Source <https://cloud.ocrsdk.com>

Annexe 3 : Amazon Textract et processing

Code Amazon Textract et processing des factures

```
ACCESS_KEY = 'AKIAQ2VPDMHFJAWZDLKZ'
SECRET_KEY = '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'
bucket_name = 'mlfacture'
region_name='eu-west-3'

import boto3
import logging
from botocore.exceptions import ClientError
import os
import glob

##### Function to list of files on S3 #####
def files_on_s3(ACCESS_KEY,SECRET_KEY,bucket_name):
    session = boto3.Session(ACCESS_KEY, SECRET_KEY)
    s3 = session.resource('s3')
    bucket = s3.Bucket(bucket_name)

    print("files on S3 : ")
    for obj in bucket.objects.all():
        print(obj.key)

files_on_s3(ACCESS_KEY,SECRET_KEY,bucket_name)

##### Bucket cleaning #####
def clean_aws_s3():
    s3_client = boto3.client('s3')

    PREFIX = ''
    #PREFIX = '/'

    response = s3_client.list_objects_v2(Bucket=bucket_name, Prefix=PREFIX)
    try:
        for object in response['Contents']:
            print('Deleting', object['Key'])
            s3_client.delete_object(Bucket=bucket_name, Key=object['Key'])
    except:
        pass

clean_aws_s3()
```

```

#### Processing des factures pour les transformer en png ou jpeg, et copier dans
le dossier de traitement ###
#change directory:
os.chdir("./factures")
# find spaces and rename
for f in os.listdir("."):
    r = f.replace(" ", "")
    if( r != f):
        print(f)
        print(r)
        os.rename(f,r)
os.chdir("../")

#liste de facture dans le dossier à uploader
liste_factures = glob.glob("./factures/*")

# verify if folder is empty or not
try :
    name_of_files = os.listdir('./factures')
except:
    print("Le dossier factures est vide")
    pass

name_of_files

#vérification de l'extension de la facture, si pdf, création d'un copie en jpg
dans le dossier factures jpg:
for i,k in zip(liste_factures,name_of_files):
    if i[-3:] == "pdf":
        print(f"Fichier {i}, EN PDF---> à convertir")

        #conversion avec pdf 2 image

        from pdf2image import convert_from_path, convert_from_bytes
        from pdf2image.exceptions import ( PDFInfoNotInstalledError,
                                           PDFPageCountError,
                                           PDFSyntaxError
                                           )

        images = convert_from_path(i)
        for j in range(len(images)):
            #print(j)
            # Save pages as images in the pdf
            images[j].save('factures-jpg/'+k+str(j+1)+'.jpg', 'JPEG') # j+1 pour
numérotation à partir de 1
            print('factures-jpg/'+k+str(j+1)+'.jpg ---> créé dans le dossier

```



```

factures_jpg! ')

    else:
        pass
        #print(f"fichier {i}, n'est pas un pdf")

#copie des factures en jpeg ou en png (autres que pdf) dans le dossier
facture_jpeg
import os
import shutil

for file in os.listdir("./factures"):
    if file.endswith(".jpg"):
        os.system(f"cp './factures/{file}' './factures-jpg/{file}'")
        print(f'{file} ---> copié dans le dossier factures-jpg')

    elif file.endswith(".png"):
        os.system(f"cp './factures/{file}' './factures-jpg/{file}'")
        print(f'{file} ---> copié dans le dossier factures-jpg')

# get list ok invokes converted in jpg
liste_factures_en_jpg = os.listdir("./factures-jpg/")

##### upload file on s3 #####

def upload_file(file_name, bucket, object_name=None):
    """Upload a file to an S3 bucket

    :param file_name: File to upload
    :param bucket: Bucket to upload to
    :param object_name: S3 object name. If not specified then file_name is used
    :return: True if file was uploaded, else False
    """

    # If S3 object_name was not specified, use file_name
    if object_name is None:
        object_name = file_name

    # Upload the file
    s3_client = boto3.client('s3')
    try:
        response = s3_client.upload_file(file_name, bucket, object_name)
    except ClientError as e:
        logging.error(e)

```

```

        return False
    return True

#upload des fichiers listés précédement:
os.chdir("./factures-jpg/") #changement de dossier pour l'upload, afin d'être
bien à la racine
for document in liste_factures_en_jpg:
    upload_file(file_name=document,
                bucket=bucket_name,
                object_name=None)
    print(f"{document}, has been up on s3")
    #main(document)
    #retour_textract = process_text_analysis(bucket_name,document)
#enregistrement des résultats d'extraction

#Ocerisation by amazon textract:
def ocerisation_by_amazon(ACCESS_KEY, SECRET_KEY, bucket_name, region_name):
    session = boto3.Session(ACCESS_KEY, SECRET_KEY)

    s3 = session.resource('s3')
    bucket = s3.Bucket(bucket_name)

    # Amazon Textract client
    textract = boto3.client('textract', region_name="eu-west-3")

    print("files on S3 : ")
    for obj in bucket.objects.all():
        print(obj.key)

    # Amazon Textract
    response = textract.detect_document_text(
        Document={
            'S3Object': {
                'Bucket': bucket_name,
                'Name': obj.key })})

    # Print detected text
    for item in response["Blocks"]:
        if item["BlockType"] == "LINE":
            #print(item["Text"])
            with open(f"./retour_txt_aws/{obj.key}.txt", "a") as myfile:
                myfile.write((item["Text"]+" \n").lower())

    print(f"{obj.key}.txt --> créé ")

    print('traitement aws terminé')

ocerisation_by_amazon(ACCESS_KEY, SECRET_KEY, bucket_name, region_name)

```

Annexe 4 : Création BDD

```
CREATE TABLE user (  
    id integer PRIMARY KEY AUTOINCREMENT,  
    name varchar,  
    email varchar,  
    password varchar  
);  
  
CREATE TABLE factures_achat (  
    id_facture integer PRIMARY KEY AUTOINCREMENT,  
    id_user integer,  
    ocr_result text,  
    ttc text,  
    ht text,  
    tva text,  
    fournisseur text,  
    date datetime,  
    date_facture text,  
    numero_facture text,  
    date_echeance text,  
    taux_tva_10 text,  
    taux_tva_20 text,  
    taux_tva_55 text,  
    taux_tva_0 text,  
    FOREIGN KEY(id_user) REFERENCES user(id)  
);
```

Annexe 5 : BDD modèle

```
# models.py

from flask_login import UserMixin
from __init__ import db
from datetime import datetime

class User(UserMixin, db.Model):
    __tablename__ = "user"
    id = db.Column(db.Integer, primary_key=True) # primary keys are required by
SQLAlchemy
    email = db.Column(db.String(100), unique=True)
    password = db.Column(db.String(100))
    name = db.Column(db.String(1000))

class Factures_achat(db.Model):
    id_facture = db.Column(db.Integer, primary_key=True)
    id_user = db.Column(db.Integer, db.ForeignKey('user.id'))
    ocr_result = db.Column(db.Text)
    ttc = db.Column(db.Text)
    ht = db.Column(db.Text)
    tva = db.Column(db.Text)
    fournisseur = db.Column(db.Text, index=True)
    date = db.Column(db.DateTime, index=True, default=datetime.utcnow)
    date_facture = db.Column(db.Text, index=True)
    numero_facture = db.Column(db.Text, index=True)
    date_echeance = db.Column(db.Text)
    taux_tva_10 = db.Column(db.Text)
    taux_tva_20 = db.Column(db.Text)
    taux_tva_55 = db.Column(db.Text)
    taux_tva_0 = db.Column(db.Text)
```

Annexe 6 : Fonction d'utilisation du modèle SpaCy

```
def use_spacy_model(text_from_ocr_for_spacy):

    doc = nlp(text_from_ocr_for_spacy)

    for ent in doc.ents:
        print(ent.text, ent.label_)

    list_of_dict =[]
    list_of_results= {"TTC":"","
                      'HT':" ",
                      'TVA':" ",

                      'FOURNISSEUR':" ",
                      'DATE_FACTURE':" ",
                      'NUMERO_FACTURE':" ",
                      'DATE_ECHEANCE':" ",
                      'TAUX_TVA_10':" ",
                      'TAUX_TVA_20':" ",
                      'TAUX_TVA_55':" ",
                      'TAUX_TVA_0':" "}

    for ent in reversed(doc.ents): # reversed to get first element find ?


        list_of_results[ent.label_.upper()]=ent.text.upper()
        #print(list_of_results)
        #list_of_results.update(p)
        #list_of_dict.append(list_of_results)

    #return list(list_of_results.items())
    return list_of_results
```

Annexe 7 : Arbre du dossier du projet

```
├── auth.py
├── backup_db_user
│   ├── dbsqlite_backup-bdd-20210416161101.sql_.dump.sql
│   └── script_backup_bdd.sh
├── datas
│   ├── fichier_converti.cfonb
│   └── resultat_extraire_cfonb.csv
├── db.sqlite
├── dernier_model_spacy
│   ├── config.cfg
│   ├── entity_ruler
│   │   ├── cfg
│   │   └── patterns.jsonl
│   ├── meta.json
│   ├── ner
│   │   ├── cfg
│   │   ├── model
│   │   └── moves
│   ├── tokenizer
│   └── vocab
│       ├── key2row
│       ├── lookups.bin
│       ├── strings.json
│       └── vectors
├── doccano.db
├── Dockerfile
├── factures
├── factures-jpg
├── __init__.py
├── main.py
├── models.py
├── requirements.txt
├── retour_txt_aws
├── script_ofx_to_cfonb.py
├── src
│   ├── __init__.py
│   └── utils.py
├── static
│   ├── img
│   │   └── logo.png
│   └── styles
├── templates
│   ├── base.html
│   ├── cfonb.html
│   ├── enregistrement_valeurs.html
│   ├── history_invoices.html
│   ├── index.html
│   ├── login.html
│   ├── ml_factures.html
│   ├── ofx.html
│   ├── result_ocr.html
│   ├── signup.html
│   └── success4.html
├── tests
│   ├── conf_test.py
│   └── __init__.py
├── tree.jpg
└── 15 directories, 42 files
```

Annexe 8 : Exemple de notification Sentry reçue par mail

[View on Sentry](#)

New alert from flask-fb in production

ISSUE

Error

NameError `auth.history_invoices`
name 'boummmm' is not defined

May 4, 2021, 9:56:07 a.m. UTC ID: 2aa1318e4b064f69acd579b333aa0eaa

Exception

```
NameError: name 'boummmm' is not defined
(3 additional frame(s) were not displayed)
...
  File "flask/_compat.py", line 39, in reraise
    raise value
  File "flask/app.py", line 1950, in full_dispatch_request
    rv = self.dispatch_request()
  File "flask/app.py", line 1936, in dispatch_request
    return self.view_functions[rule.endpoint](**req.view_args)
  File "flask_login/utils.py", line 272, in decorated_view
    return func(*args, **kwargs)
  File "my_app_plateforme_flask/auth.py", line 273, in
history_invoices
    print (boummmm)
```

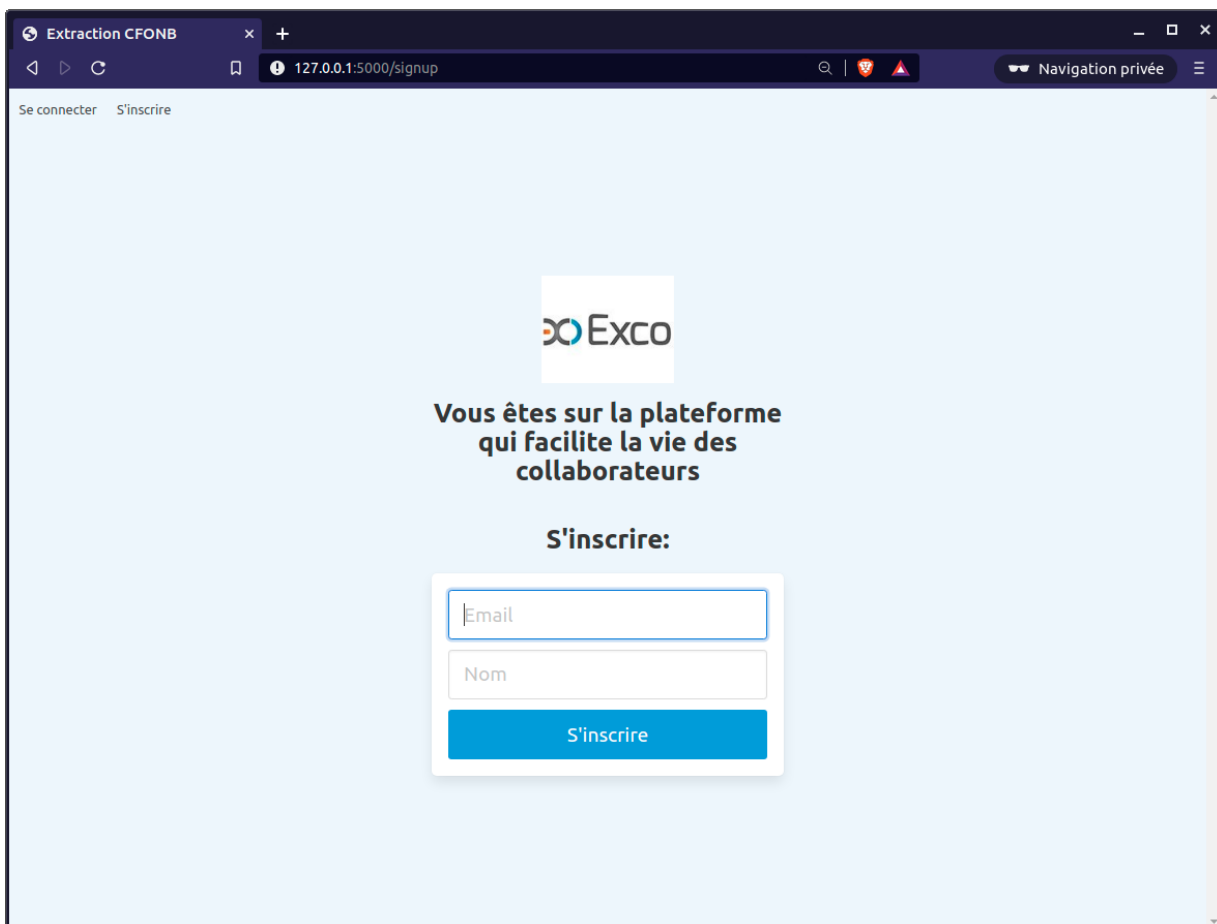
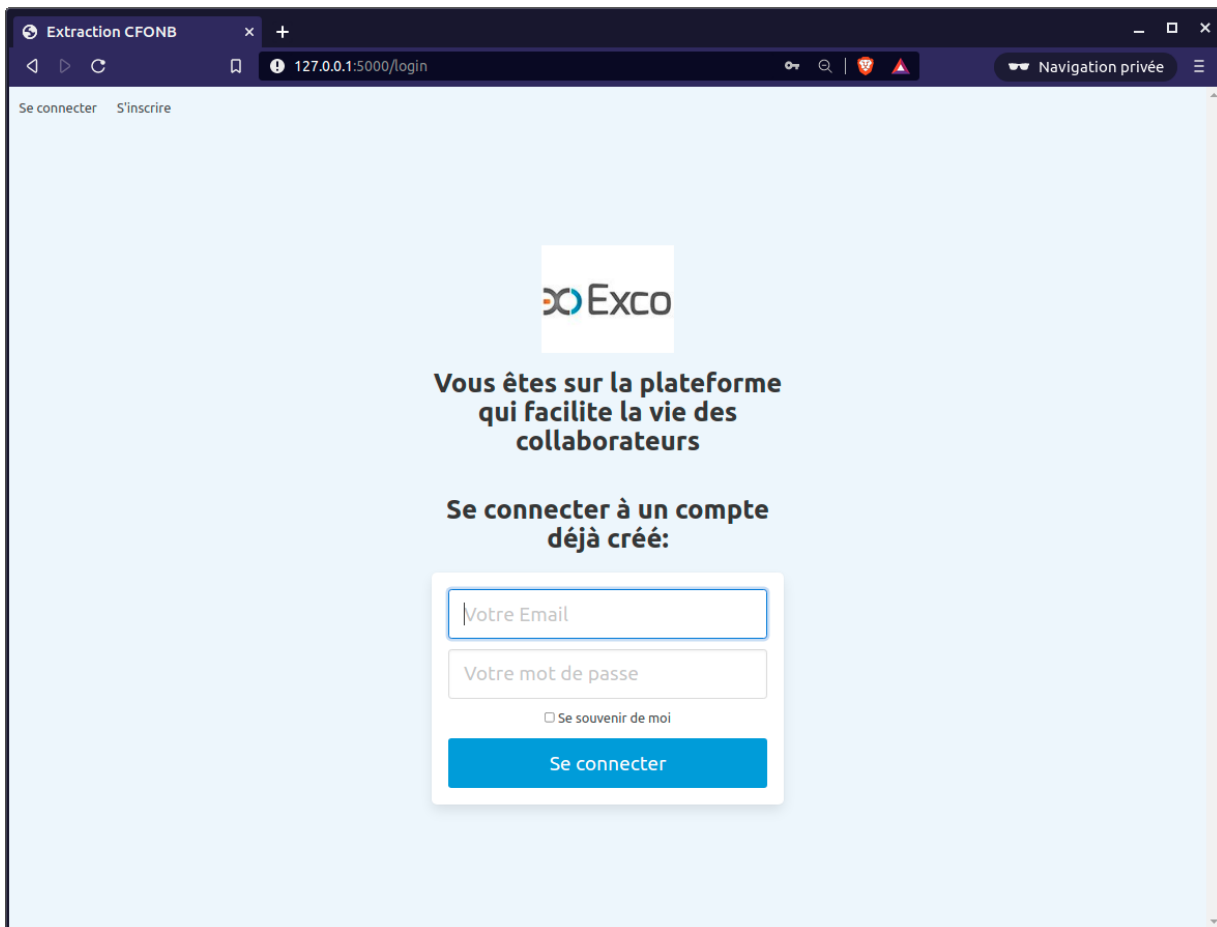
Request

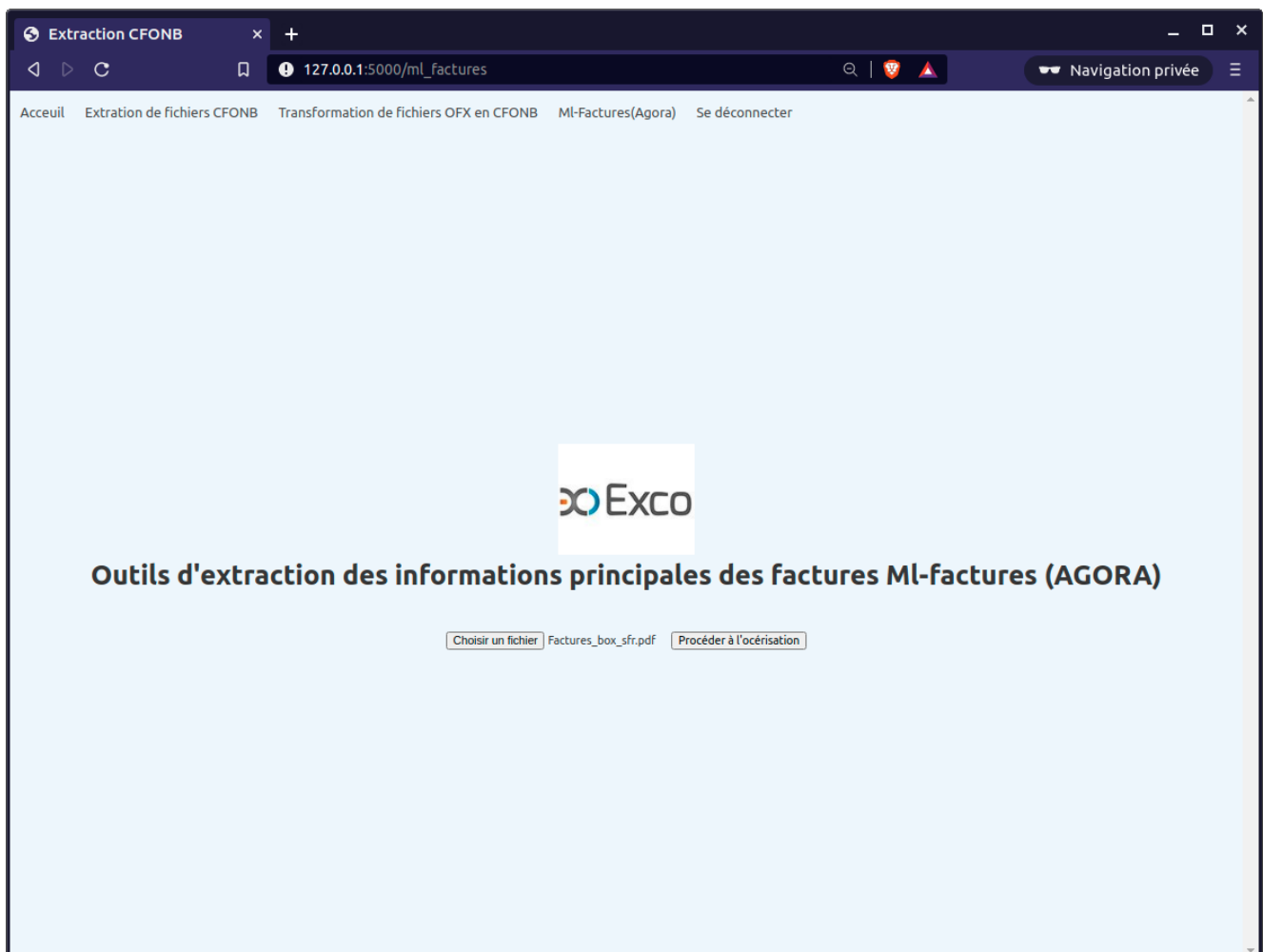
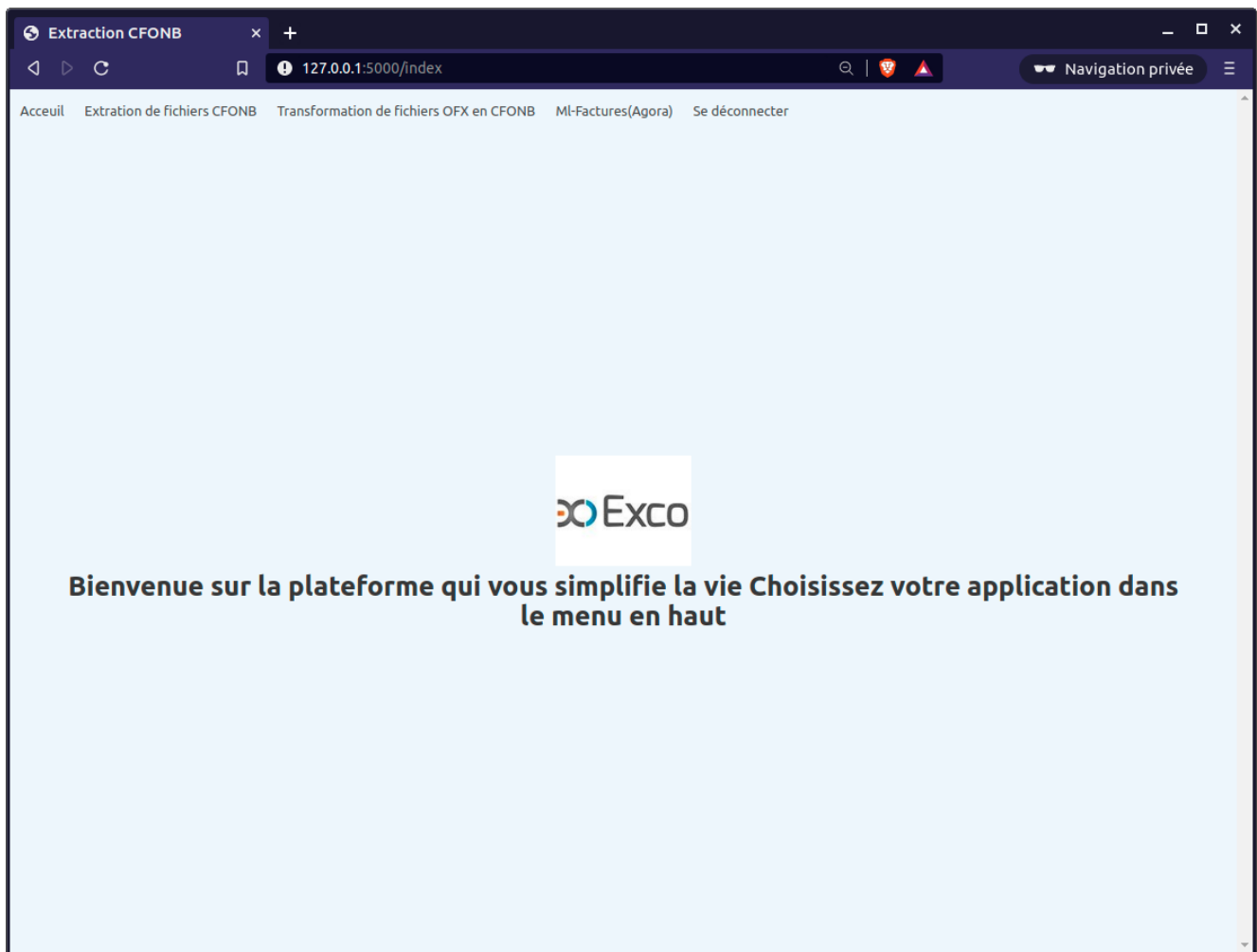
URL	http://127.0.0.1:5000/history_invoices
Method	GET

Tags

browser = Chrome 90.0.4430	browser.name = Chrome	
client_os.name = Linux	environment = production	handled = no
level = error	mechanism = flask	runtime = CPython 3.8.5
runtime.name = CPython		
release = 4ef76ed607995b8721e5a230e4091c720ff880c5		
server_name = ubuntu-Precision-Tower-3430		
transaction = auth.history_invoices		
url = http://127.0.0.1:5000/history_invoices		

Annexe 9 : Mockups et captures de l'application






e.leclerc
orthes distribution s.a.s.
route de bayonne bp 419
64304 orthes cedex
sas au capital de 39 479 euros
tel:05 59 69 42 22 fax:05 59 69 92 05
dom bancscredit agricole angleat
16906-00042-01016088756-36
rcs pau b 319 469 698
no ii : fr 24 319 469 698
bltp
19 chemin lassauque
64270 belloccq
date d'impression : 15/01/21
client en compte n° 006726
b.l.t.p
facture n° 210000216
folio 1/1
désignation
quantité
prix unitaire ht
tva
montant ttc
carte 9250004090001988830 - iveco daily 7200 zg 64
ticket 04/01/21 1 n5t2 02h00 18:41
3502290000038 gas oil
45.200
1.051
20%00
57.00
total carte 9250004090001988830
45.200
57.00
carte 9250004090002877701 - fiat doppio db-605-wr
ticket 04/01/21 1 n4t2 03100 17:32
3502290000038 gas oil
46.580
1.051
20%00
58.74
ticket 13/01/21 1 n1t2 01800 17:20
3502290000038 gas oil
35.260
1.069
20%00
45.24
total carte 9250004090002877701

Extraction CFONB

127.0.0.1:5000/enregistrement_valeurs

Navigation privée

AccueilExtraction de fichiers CFONBTransformation de fichiers OFX en CFONBML-Factures(Agora)Se déconnecter



Enregistrement des valeurs

Les valeurs que vous avez validées ont été enregistrées dans la base de données.


Afficher mon historique de traitement des factures: [cliquez ici](#)
Une autre facture à traiter, [cliquez ici](#)

Extraction CFONB

127.0.0.1:5000/history_invoices

Navigation privée

AccueilExtraction de fichiers CFONBTransformation de fichiers OFX en CFONBML-Factures(Agora)Se déconnecter



Historique de traitement des factures

Vous avez effectué 8 traitements de facture.

ID facture	ID utilisateur	Résultat OCR	TTC	HT	TVA	Fournisseur	Date traitement	Date facture	Numéro de facture	Date d'échéance	Taux de tva 10%	Taux de tva 20%	Taux de tva 5,5%	Taux de tva 0%
40	2	industrie- bedarf rechnung los art traffik verkauft von 10, allée de samadet vos industriebedarf gmbh & co.kg 64600 anglet ust-idnr. fr82832987879 frankreich rechnungsdatum: 18.02.2021 rechnungs-nr.: 1103102 rechnungsbetrag: 47,61 € rechnungsadresse lieferadresse verkauft von art traffik bureau de poste bayonne allees marines bpvos industriebedarf gmbh & co.kg 10, allée de samadet rue de la nouvelle poste empeler str. 55 64600 anglet 64100 bayonne 46459 rees frankreich frankreich ust-idnr. fr82832987879 registergericht: ag kleve hr-nr.: hra 3462 wilhelm vos bestellinformation bestelldatum: 18.02.2021 amazon-bestell-nr.: 405-3672312-5289945 beschreibung menge stückpreis ust. % stückpreis gesamtprice (ohne ust.) (inkl. ust.) (inkl. ust.) fripa essuie-mains en papier, pliage en v, 2 plis, blanc 1 39,67 € 20,00 % 47,61 € 47,61 € sku: fbfri4012103 gesamtprice euro eur 47,61 € ust. % zwischensumme ust. (ohne ust.) eur 20,00 % 39,67 € 7,94 € alle umrechnungskurse vom 17.02.2021 gem. ezb. die lieferung erfolgte per standard am 18.02.2021 13:19:36 durch lp_collect mit desedungssummer 6a9043733512 the				RECHNUNG	2021-05-04 09:59:21.226051							

Mockup de la page de résultats de prédictions / contrôle des données / validation

Menu app / s'enregistrer / s'inscrire

Image de la facture uploadée

Tableau pré-rempli par SpaCy	
1	Value 1
2	Value 2
3	Value 3

Bouton de validation

Tout le texte océrisé

Item 1

Item 2

Item 3

Annexe 10 : Dockerfile

```
FROM python:3.8-slim

WORKDIR /code

ENV FLASK_APP=.

ENV FLASK_RUN_HOST=0.0.0.0

RUN apt update

RUN pip install --no-cache-dir spacy &&\
    python -m spacy download en

RUN pip install --upgrade pip setuptools wheel

COPY requirements.txt requirements.txt

RUN pip install -r requirements.txt

EXPOSE 5000

COPY . /code

RUN chmod 777 /code

CMD ["flask", "run"]
```

Annexe 11 : Scores

```
import spacy
from spacy.training import *
from spacy.scorer import Scorer

from itertools import chain
import numpy as np
import pandas as pd
# Calcul des scores
def calc_precision(pred, true):
    precision = len([x for x in pred if x in true]) / (len(pred) + 1e-100) # true
    positives / total pred
    return precision

def calc_recall(pred, true):
    recall = len([x for x in true if x in pred]) / (len(true) + 1e-100) # true
    positives / total test
    return recall

def calc_f1(precision, recall):
    f1 = 2 * ((precision * recall) / (precision + recall + 1e-100))
    return f1
# Evaluation des entités
def score_per_ent(model, TEST_DATA):
    # run the predictions on each sentence in the test dataset, and return the spacy
    object
    test_text = [model(x[0]) for x in TEST_DATA]
    # Création de listes vides pour le dataframe
    tr, tp = [], []
    entities, preds, trues, precs, recs, fs = [], [], [], [], [], []

    for row in range(len(TEST_DATA)):
        doc = model(TEST_DATA[row][0])
        for ent in doc.ents :
            entity = ent.text

    test = []
    for true in TEST_DATA :
        test.append(true[1]['entities'])

    # # iterate over predictions and test data and calculate precision, recall, and
    F1-score
    for pred_val, start, end, ent, pred, true in zip(test_text, TEST_DATA, test_text,
test_text, test_text, TEST_DATA):

        true = [x[2] for x in list(chain.from_iterable(true[1].values()))] # x[2] =
        annotation, true[1] = (start, end, annot)
```

```

pred = [i.label_ for i in pred.ents]
ent = [entity.text for entity in ent.ents]
pred_val = [(i.start_char, i.end_char, f'{i.label_}')] for i in pred_val.ents]

tp.append(pred_val)

if len(pred) > len(true) :
    data_length = len(true)
else:
    data_length = len(pred)

for i in range(data_length) :

    print(f'{ent[i]} >> {pred[i]} >> {true[i]}')
    precision = calc_precision(true[i], pred[i])
    recall = calc_recall(true[i], pred[i])
    f1s = calc_f1(precision, recall)

    entities.append(ent[i])
    preds.append(pred[i])
    trues.append(true[i])
    precs.append(np.around(precision, 3))
    recs.append(np.around(recall, 3))
    fs.append(np.around(f1s, 3))

print(tp[0])
print(test[0])
# print(tr[0])
score_per_ent = pd.DataFrame({
    'Entity': entities,
    'Pred': preds,
    'True': trues,
    'P': precs,
    'R': recs,
    'F1': fs })

return score_per_ent
def model_score(nlp, TEST_DATA):
    # run the predictions on each sentence in the test dataset, and return the spacy
    object
    preds = [nlp(x[0]) for x in TEST_DATA]

    precisions, recalls, f1s = [], [], []

    # iterate over predictions and test data and calculate precision, recall, and
    F1-score
    for pred, true in zip(preds, TEST_DATA):
        true = [x[2] for x in list(chain.from_iterable(true[1].values()))] # x[2] =
        pred = [i.label_ for i in pred.ents]          precision = calc_precision(true,
pred)
        precisions.append(precision)
        recall = calc_recall(true, pred)

```

```

        recalls.append(recall)
        f1s.append(calc_f1(precision, recall))

    model_score = [[np.around(np.mean(precisions), 3)], [np.around(np.mean(recalls),
3)], [np.around(np.mean(f1s), 3)]]
    model_score = pd.DataFrame({'Precision' : [np.around(np.mean(precisions), 3)],
        'Recall' : [np.around(np.mean(recalls), 3)],
        'F1' : [np.around(np.mean(f1s), 3)]    })
    return model_score



model_score(nlp, TEST_DATA)

```

Source : Meidi Kadri pour le scorer

Annexe 12 : Exemple avec UBI AI : (<https://ubiai.tools/>)

UBIAI

PROJECTS ▾ MODELS ▾ COLLABORATION ▾ ANALYSIS ▾ DOCUMENTATION ▾ PACKAGES Free Trial Expiring in 8 days  

Annotation Details Documents

Project Details

Project Name : ml_factures

Project Type : Text Annotation

☒ Auto save ☒ Auto detection


Progress


Finished Documents : 12


Total Documents : 33


36.36 %


Documents


 Factures_crea.pdf1.jpg_2021-03-23_1... ✓


 Factures_decath.pdf1.jpg_2021-03-2... ✓


 Factures_ducasse.pdf1.jpg_2021-03-2... ✕


 Factures_ducasse2.pdf1.jpg_2021-03-... ✕


 Factures_france_bonh.pdf1.jpg_2021... ✓

 Factures_gaia.pdf1.jpg_2021-03-23_1... ✕

 Factures_garonne.pdf1.jpg_2021-03-... ✕

 Factures_lagelouze.pdf1.jpg_2021-03-... ✕

 Factures_lagune.pdf1.jpg_2021-03-23... ✕

 Factures_lagune_2.pdf1.jpg_2021-03-... ✕

Previous

1

2

3

4







5

Next

« Entities Relations »

TTC 2 HT 3 TVA 4 FOURNISSEUR 5 DATE_FACTURE 6 NUMERO_FACTURE 7 DATE_ECHEANCE 8 TAUX_TVA 9

ATTENTE AVOIR AVOIR OK **GATA FOURNISSEUR** SITE D4AURIGNAC FACTURE N ° **14400RX21006253 NUMERO_FACTURE** RTE DE BOUSSENS Gaig 31420 AURIGNAC Date **28/02/2021 DATE_FACTURE** SIRET : N / Ref : FC21001336F Valorisons nos ressources Tél : Fax : Act . , 10 N ° compte 16310819 LASSERRE BASTIEN 19 CHEMIN DE LASSAUQUE 64270 BELLOCC TVAIC Client : FR77514560804 Coordonnées bancaires IBAN : FR05 3000 2038 2200 0006 0204 E40 SIRET Client 51456080400023 BIC : CRLYFRPP NON ! Agence : LOL LE CREDIT LYONNAIS + mawals RIB ! TGAP 2021 établie à 0.21 € / To suite publication au BOFP Taux Article Dénomination Qté P.U. Mont . H.T.T.V.A. Site : Site d' AURIGNAC Dépôt : AURIGNAC 51456080400023 BIC : CRLYFRPP NON ! Agence : LOL LE CREDIT LYONNAIS + mawals RIB ! TGAP 2021 établie à 0.21 € / To suite publication au BOFP Taux Article Dénomination Qté P.U. Mont . H.T.T.V.A. Site : Site d' AURIGNAC Dépôt : AURIGNAC CARRIERE Chantier : RAMOUS RAMOUS * AU1016C CONCASSE 10/16 CALCAIRE AURIGNAC **20 TAUX_TVA** % 30,160 T 19,50 € 588,12 € TGAP Taxe Générale sur les Activités Polluantes **20 TAUX_TVA** % 30,160 T 0,21 € 6,33 € S / s Total Chantier : RAMOUS RAMOUS **594.45 HT** € S / s Total Dépôt : AURIGNAC CARRIERE **594.45 HT** € S / s Total Site : Site d' AURIGNAC **594.45 HT** 0 Taux T.V.A. H.T.T.V.A. Montant **20 TAUX_TVA** % **594.45 HT** **118.89 TVA** H.T. **594.45 HT** € Conditions de règlement Mode derèglement Date de règlement T.V.A. **118.89 TVA** E 30 jours date de facture Traite **30/03/2021 DATE_ECHEANCE** T.T.C. **713.34 TTC** € Pénalités en cas de retard de paiement : taux BCE + 10 % article L441 - 6 code de commerce . L' indemnité forfaitaire pour frais de recouvrement due en cas de retard de paiement est égale à 40 E. GAÏA . Chez COLAS SUD-OUEST Avenue Charles LINDBERG 33700 MERIGNAC - SARL AU CAPITAL DE 6 165 993.50 € BORDEAUX 494 024 409 APE 0812Z - TVA : FR54494024409 - Tél:05 58 71 59 60 - Fax:05 58 71 69 03 Page 1/1 SGS SES

Factures_gaia.pdf1.jpg_2021-03-23_13:41:46.txt

Ti 0

65