# Projet OCR



# I. - Présentation du projet

## 1. - Contexte

Le projet, en tant que Développeur en Intelligence Artificielle pour le compte d'une ESN, était de créer un système d'extraction automatique du texte et des données des factures, à l'aide de fonctionnalités d'OCR, dans le but de l'implémenter dans la procédure de pré-traitement des factures. Le but est donc d'obtenir un système de reporting automatisé de la comptabilité fournisseur de l'ESN.

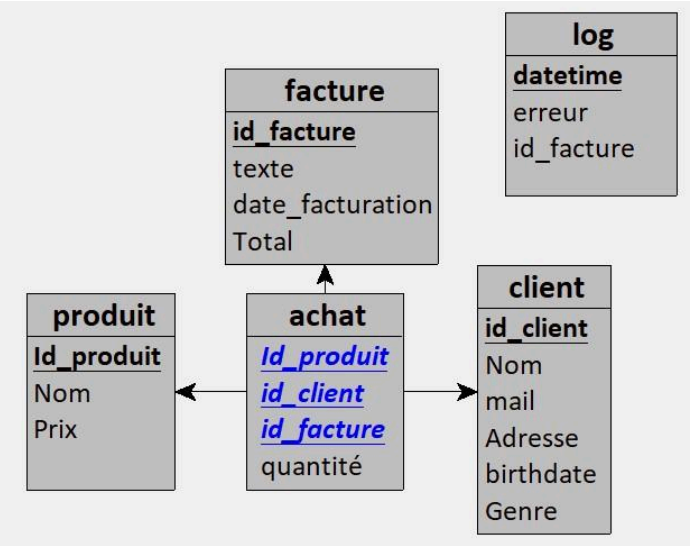
Ce projet est composé de plusieurs parties et composants.
La première partie consistait à créer un système d'OCR pour extraire les données des factures, mises à disposition par l'ESN via une API, et stocker ces données dans une base de données.
Puis dans une deuxième partie, le but était d'intégrer ce service d'OCR dans une application pour mettre à disposition une démonstration de ce service , ainsi que pour présenter les données que l'on avait extraites des factures à l'aide de ce service.
La troisième partie consistait à analyser les données des factures, dans le but de présenter un clustering ou une segmentation des clients et des produits, ainsi qu'un système de recommandation de produits.
Enfin, l'application devait inclure des tests, intégrer un processus de CI/CD, et être conteneurisé via docker.

Pour gérer le projet, j'ai utilisé plusieurs systèmes de gestion de projets.
Tout d'abord, j'ai commencé par créer un schéma de fonctionnement sommaire afin de bien comprendre le contexte du projet et ses composants, ainsi que l'ensemble des éléments obligatoires pour avoir un projet fonctionnel (produit minimum viable) .
Puis, j'ai utilisé un tableau kanban afin de planifier les étapes du projet et extraire les différentes tâches à compléter pour avoir l'ensemble des fonctionnalités essentielles.
J'ai régulièrement mis à jour le schéma de fonctionnement et le tableau kanban au fur et à mesure de mon avancée dans le projet et des changements de structures du projet.
Par la suite, après avoir construit mon système d'OCR fonctionnel, j'ai réalisé une maquette figma simple avant de commencer à créer mon application.

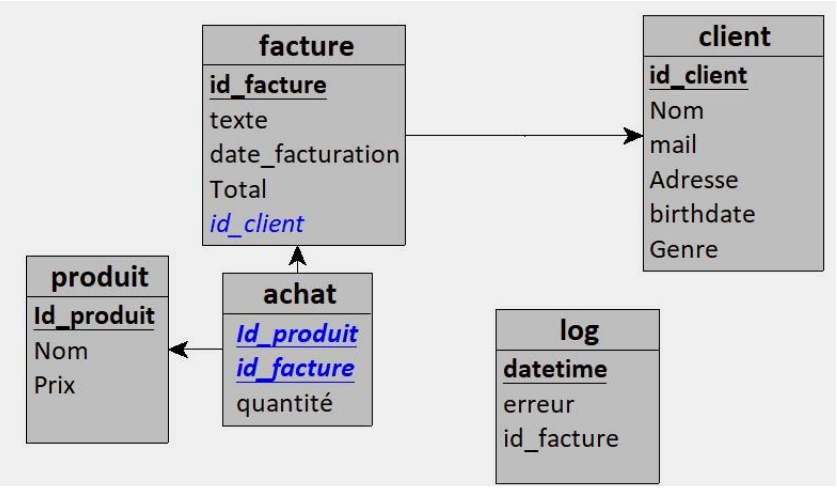
## 2. - Base de données

Après des tests d'extraction de texte des factures, j'ai pu identifier les données que je pouvais extraire des ces factures, et j'ai réfléchi à la structure de ma base de données pour les stocker de façon optimale.

J'ai choisi de réaliser 5 tables différentes, pour stocker séparément les clients, les factures, les produits, ainsi qu'une table pour relier ses 3 tables sous forme d'achat, et une table de monitoring pour stocker les erreurs de l'OCR.
Cela donnait donc un modèle de données comme celui ci:




Après réflexion, et après avoir avancé sur le projet jusqu'à l'analyse des données, il m'est apparu qu'il aurait été plus logique de lier la table client directement à la table facture, plutôt que de le passer via la table achat, puisque 1 facture est lié à 1 client. De plus, cela aurait évité une répétition inutile des clients dans la table achat pour chaque facture.
Cela aurait donc donné un modèle de données comme celui ci:*




Malheureusement, mon avancée sur le projet ne me permettait pas de reprendre ceci. Cela aurait nécessité que je modifie ma base de données, puis que je refasse l'extraction de l'ensemble des données des factures. La durée de l'extraction étant très longue, cela m'aurait fait perdre un temps considérable.

# II. - Présentation de l'OCR

## 1. - Principe

Le sigle OCR signifie Optical Character Recognition. Le principe est donc logiquement de pouvoir reconnaître de numériquement les caractères d'écritures afin de les extraire, et donc de transformer un texte (ou des données textuelles) d'un format image ou manuscrit à un format numérique, ce qui permet de les traiter de façon automatisée.
Il existe de nombreux outils et services qui permettent cette reconnaissance. Après plusieurs recherches, je me suis arrêté sur certains en particulier.
J'ai testé sur quelques factures les outils Tesseract, EasyOCR, DocTR, ainsi donut-base-finetunned-invoices de hugging face.
EasyOCR fonctionnait plutôt bien mais était très lent. Le modèle récupéré de hugging face fonctionnait plutôt bien aussi, mais était très lent aussi et ne récupérait que certaines données spécifiques des factures, et donc pas l'ensemble des informations qui nous intéressent.
Tesseract et DocTR fonctionnaient bien, et étaient plutôt rapides.
J'ai choisi finalement d'utiliser Tesseract qui était un peu plus simple d'utilisation, en particulier pour le preprocessing des images dans le but d'améliorer les performances du service.

## 2. - Méthode et fonctionnement

Le moteur OCR, dans ce projet Tesseract, fonctionne en plusieurs étapes pour extraire les caractères d'une image et les convertir en texte exploitable. Tout d'abord, il commence par analyser l'image pour repérer les zones contenant du texte. Ces zones sont segmentées en blocs, puis en lignes, en mots et enfin en caractères individuels. Une fois les caractères

isolés, Tesseract compare leur forme à une base de données contenant des modèles de lettres, chiffres et symboles. Cette comparaison repose sur des algorithmes qui identifient la structure des caractères et les transforment en texte numérique. Une fois tous les caractères reconnus, le moteur reconstitue le texte complet tel qu'il apparaît dans le document. Il utilise aussi des dictionnaires intégrés pour corriger certaines erreurs courantes et améliorer la précision de la reconnaissance.

De mon côté, avant de passer une facture dans le moteur OCR, je réalise un prétraitement de l'image pour améliorer ses chances d'être correctement analysée. Je vais ainsi ajuster le contraste et la luminosité pour rendre le texte plus lisible, convertir l'image en noir et blanc (binarisation) pour mieux distinguer le texte de l'arrière-plan, et supprimer tout élément parasite comme des taches ou des lignes inutiles.
Pour réaliser ce traitement, j'utilise la librairie openCV qui dispose de nombreuses méthodes de traitement d'image. Elle dispose entre autres d'un lecteur de QR code qui me servira à extraire les données des QR code présent dans les factures fournies.
Je divise ensuite l'image en différentes petites images correspondants aux zones où sont normalement présentes certaines données spécifiques que je souhaite récupérer. Par exemple, le nom de la facture est toujours présent en haut à gauche de la facture, et le QR code en haut à droite.
J'envoie chacune de ces petites images à Tesseract (sauf le QR code que je traite avec openCV).

Une fois que Tesseract m'a renvoyé le texte extrait, je prends le temps de nettoyer et structurer ces données. Je récupère séparément selon l'image traité chaque type de données, vérifie la cohérence (comme le fait que le total de la facture correspond bien à la somme des achats), enlève les espaces blancs, puis structure les données dans des data frames pour pouvoir ensuite les envoyer en base de données.

## 3. - Limites

Malgré le prétraitement que je réalise pour améliorer la qualité des images avant leur passage dans l'OCR, certaines limites restent présentes et peuvent affecter les résultats.
La première limite concerne la qualité des factures fournies. Si une facture est floue, mal scannée ou contient des éléments parasites comme des ombres, des pliures ou des tâches importantes, même après traitement avec OpenCV, Tesseract peut avoir du mal à reconnaître correctement les caractères. Cela peut entraîner des erreurs dans les données extraites.

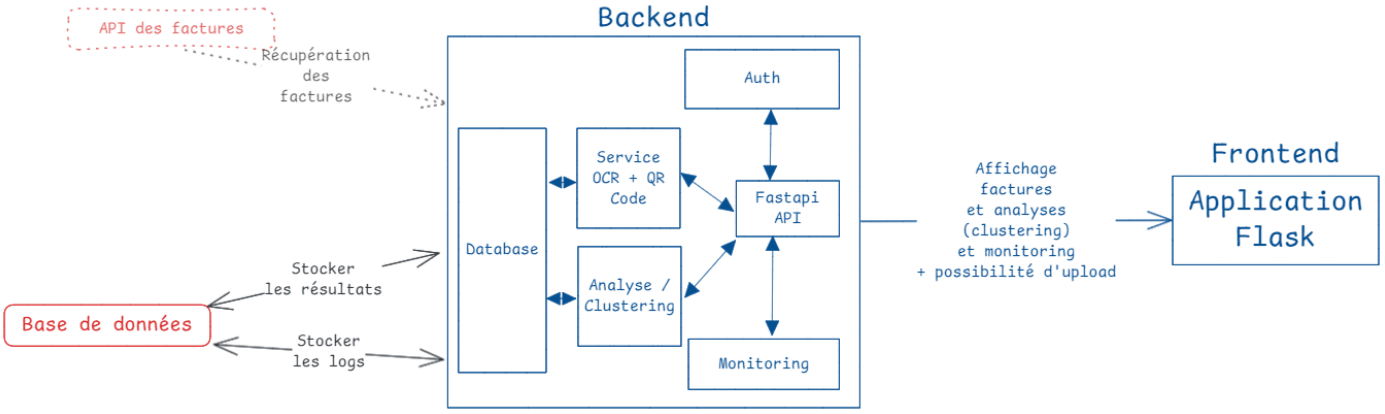
Ensuite, bien que je divise les factures en zones spécifiques pour cibler les données à récupérer (par exemple, le nom en haut à gauche ou le QR code en haut à droite), il pourrait arriver que certaines factures ne respectent pas cette structure. Cela peut compliquer l'extraction et nécessiter un ajustement manuel.
Une autre limite est liée aux caractères similaires. Tesseract peut confondre certains caractères proches visuellement, comme "0" et "O" ou "1" et "I". Ceci est tout de même limité par le fait que, pour les données numériques, j'ai limité les caractères possibles aux chiffres.

Enfin, bien que je vérifie la cohérence des données extraites, certaines incohérences peuvent passer inaperçues si elles ne sont pas flagrantes.

# III. - Fonctionnement de l'application

## 1. - Architecture de l'application

J'ai choisi de séparer mon application en une API créée avec FastAPI, et un frontend avec un autre framework. Il est vrai que j'aurais pu réunir les deux services en un seul serveur fastapi, qui servirait à la fois l'api et l'application web, avec des routes différentes, mais je préférais séparer complètement les deux services, pour travailler avec une architecture un peu plus "classique".
Pour la partie frontend, j'ai choisi d'utiliser flask pour avoir une application web assez simple et qui s'adapte facilement à mes besoins.




Mon projet est donc divisé en un backend fast API qui réalise les traitements des données, utilise les services d'ocr et réalise le clustering, en utilisant la base de données, et un frontend flask qui utilise des templates html pour interagir avec les factures via une interface graphique.
Certaines routes du backend et du frontend sont aussi protégées d'accès par une authentification configuré avec OAuth2 avec un Bearer Token JWT, récupérable en s'identifiant sur la route /token de l'api. Les utilisateurs sont stockés avec leur mot de passe haché sur la base de données.

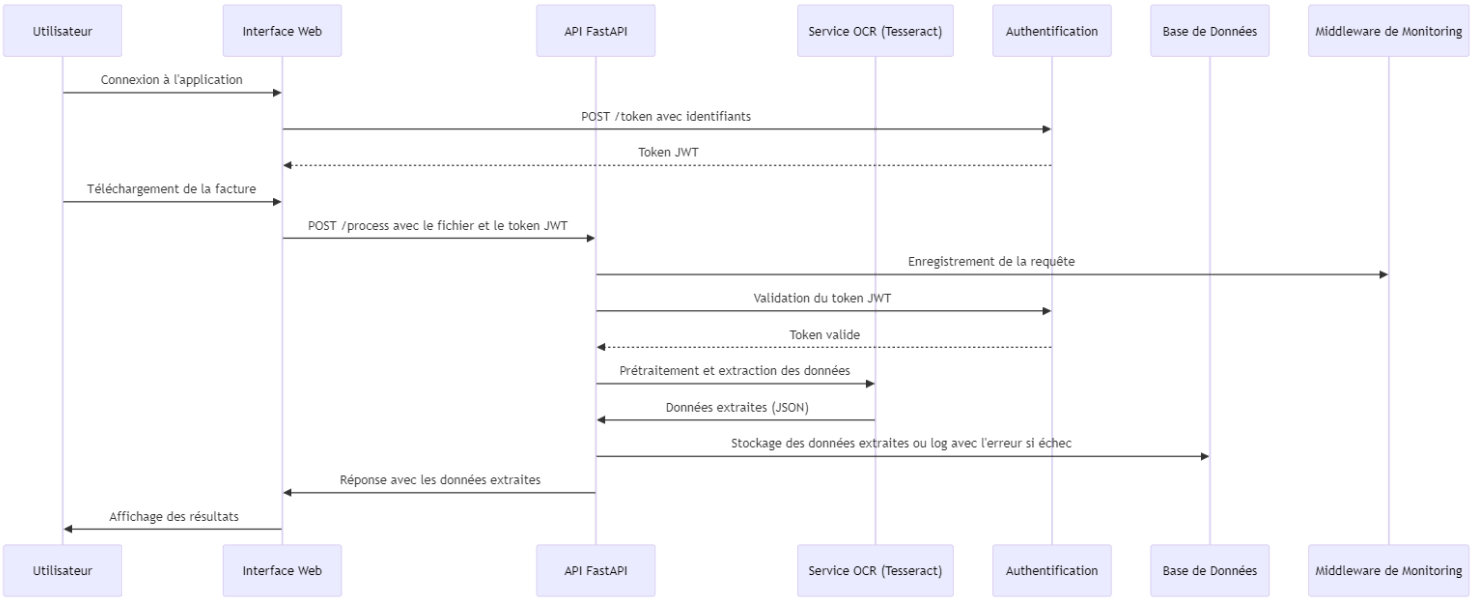
Cette base de données est une base de données PostGreSQL sur Azure cloud, puisque c'est ce qui m'était fourni.

Enfin, le service OCR de mon backend utilise pytesseract, qui interagit avec Tesseract installé localement sur la machine faisant tourner le serveur (ou dans le conteneur docker donc).

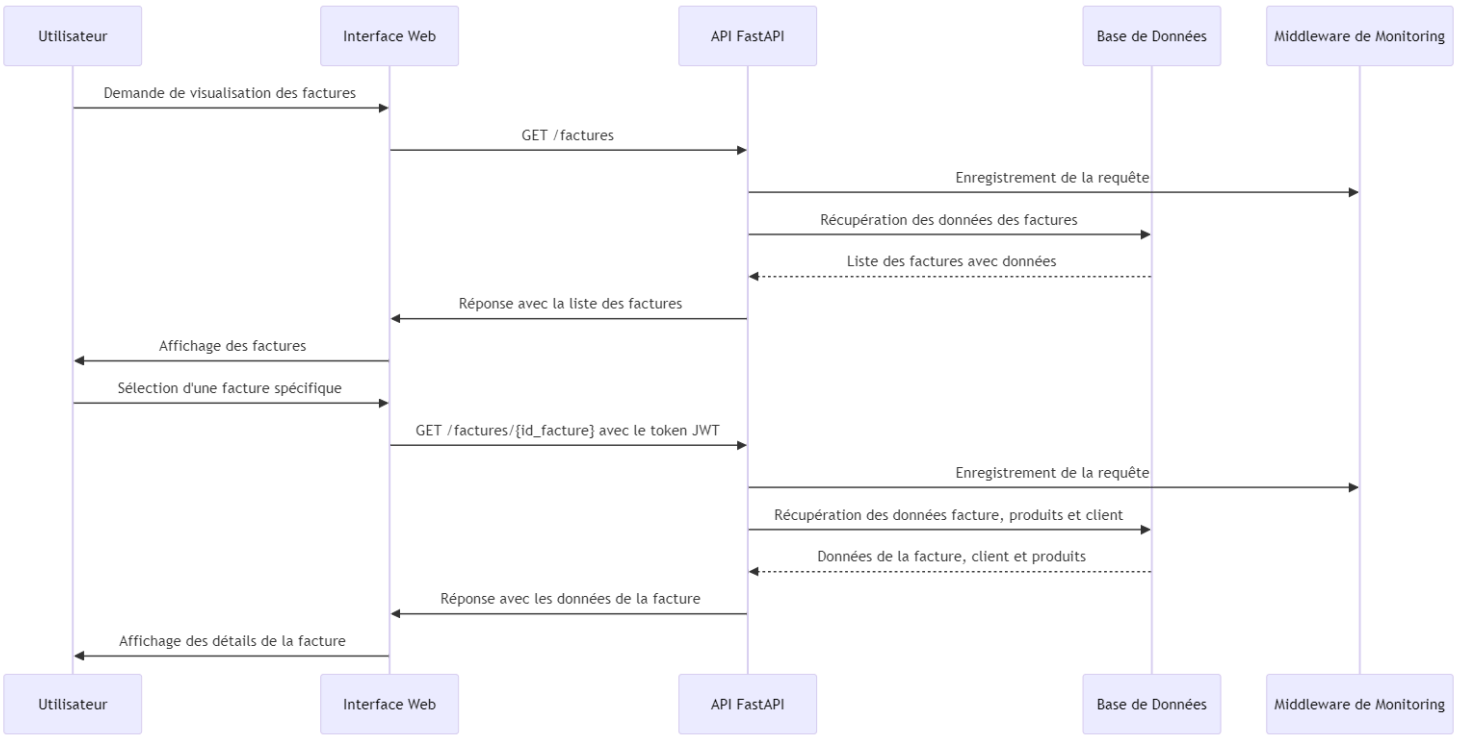
# 2. - Flux de fonctionnement

J’ai schématisé le fonctionnement de l’application, et les intéractions entre les différents composants de l’application, sous forme de diagramme de séquences.

## 2.1. - Téléchargement d’un fichier et traitement OCR

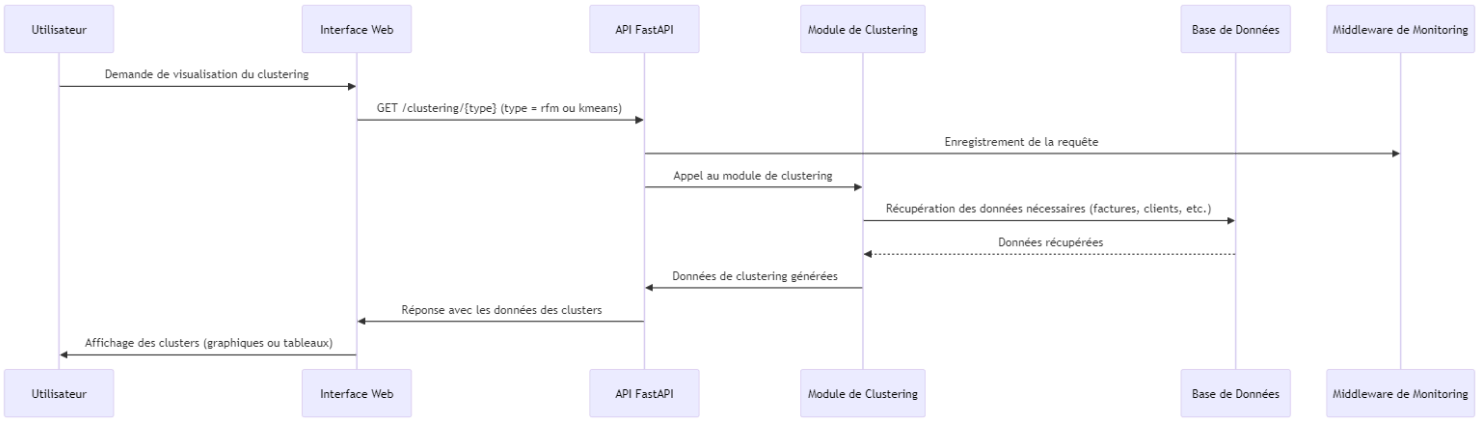



## 2.2. - Affichage d’une ou des factures

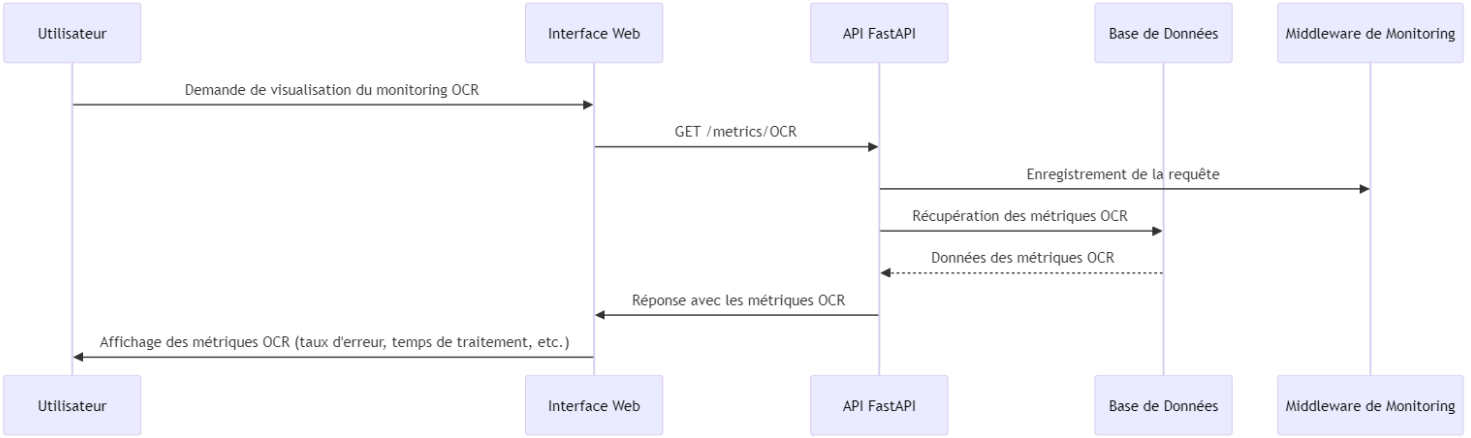


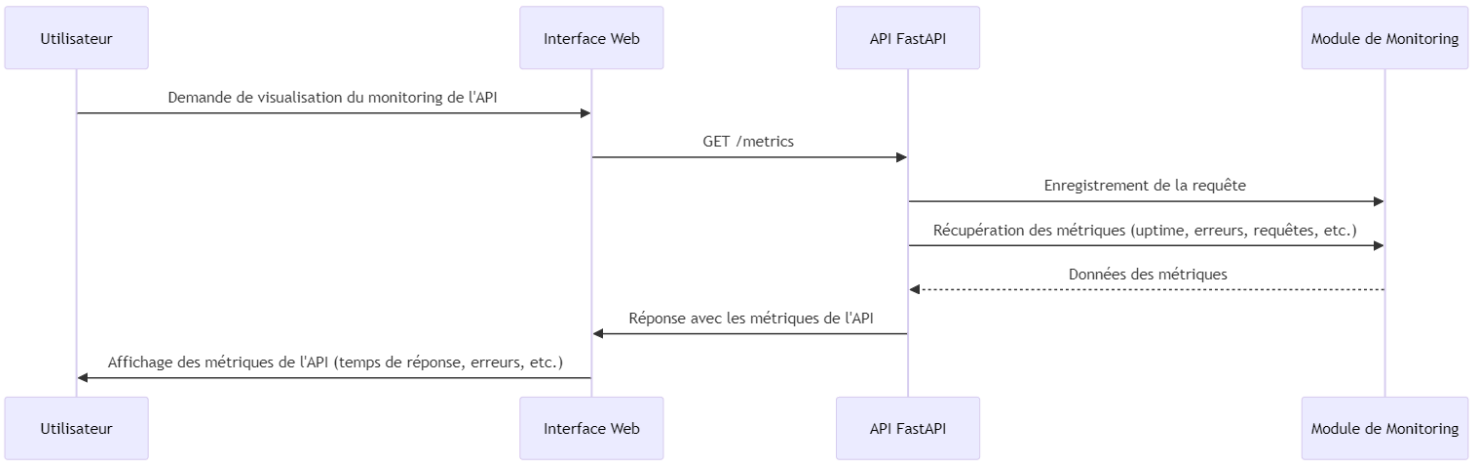
## 2.3. - Visualisations du clustering

Concernant le clustering, j'ai choisi de réaliser ce clustering avec 2 méthodes pour les comparer.
J'ai d'abord utilisé la méthode RFM (Récence, Fréquence, Montant), pour classer les clients selon des catégories prédéfinies communes.
Puis, j'ai créé un autre clustering en utilisant cette fois ci un modèle kmeans avec scikit-learn, et en reprenant les même variables que le RFM (donc nombre de jours depuis dernier achat, nombre d'achat, montant total d'achat), mais en ajoutant l'âge qui peut être une variable intéressante pour catégoriser les clients.
Mon prétraitement des données était seulement composé d'une standardScaler pour normaliser les données.




## 2.4. - Visualisation du monitoring

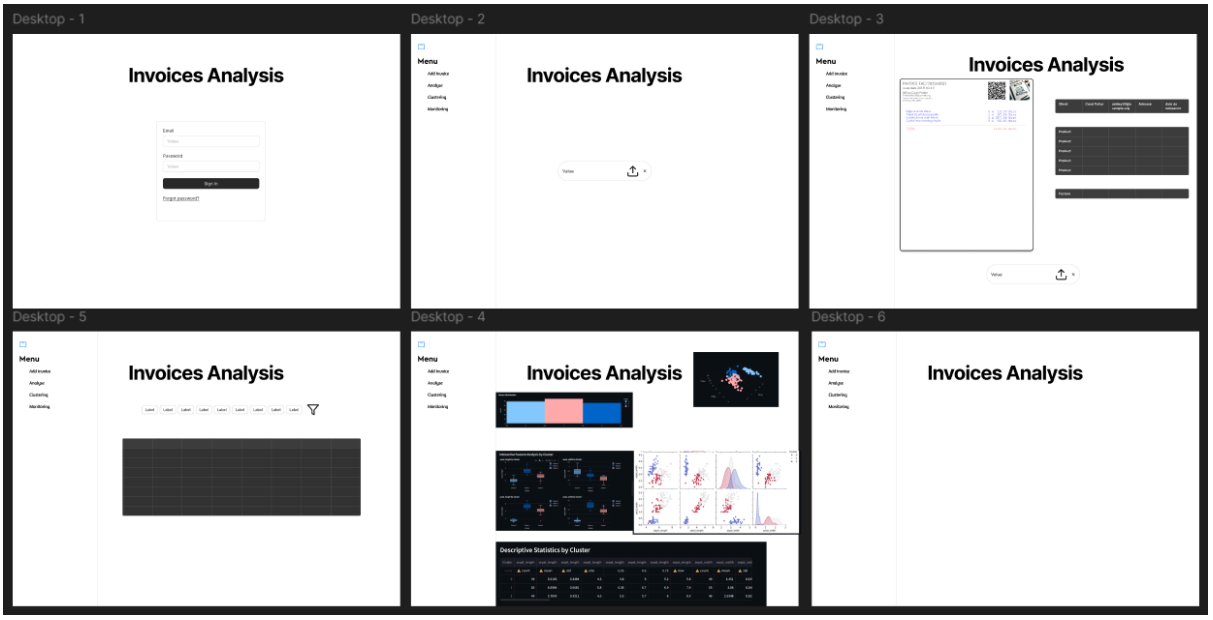




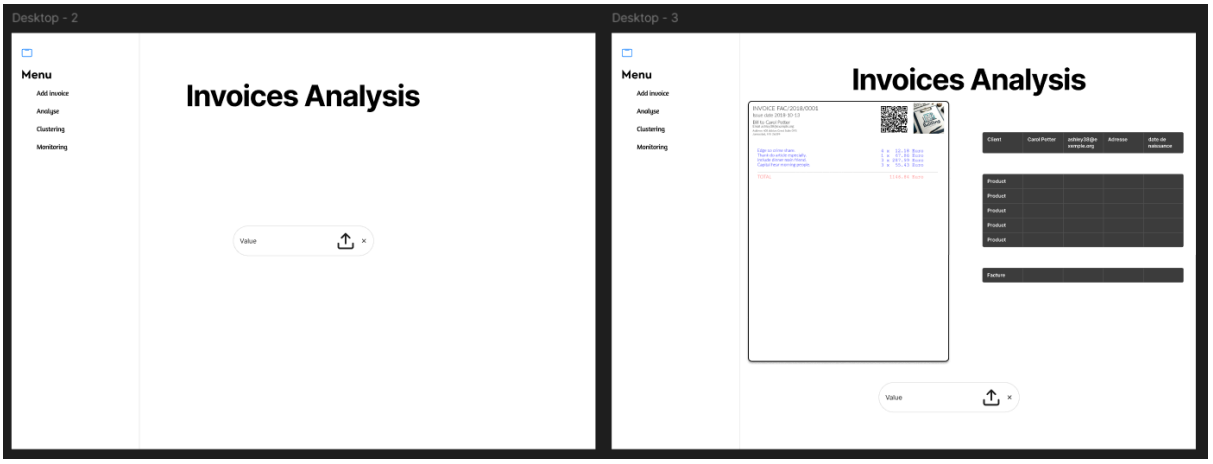

# 3. - Maquette et interface utilisateur

## 3.1. - Maquette

Avant de créer le frontend de mon application, j'ai imaginé cette maquette d'interface:



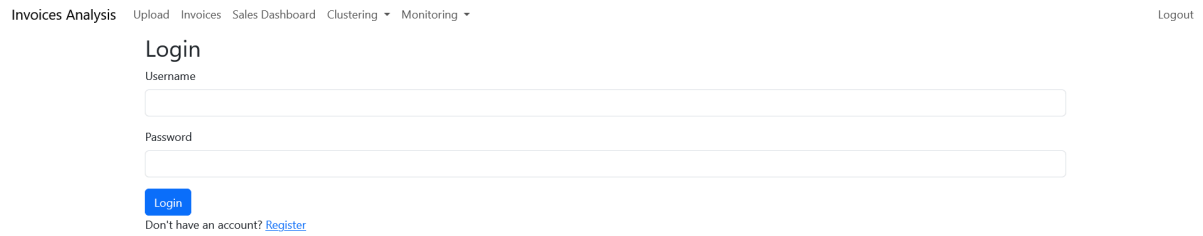

Avec en particulier donc ces pages pour l'upload d'une facture et l'extraction et affichage de ses données:




J’ai finalement une interface assez différente. En particulier j’ai choisi une barre de navigation horizontale, parce que j'utilise bootstrap pour la mise en page de mon application, et que bootstrap intègre plus nativement une barre de navigation horizontale.
Je n’ai aussi pas intégré de filtres sur mon tableau des factures, mais seulement des boutons pour trier par colonne, par manque de temps.
Pour finir, je n’ai finalement qu’une graphique circulaire pour interpréter le clustering, ainsi qu’une table simple des stats par clusters, aussi par manque de temps.

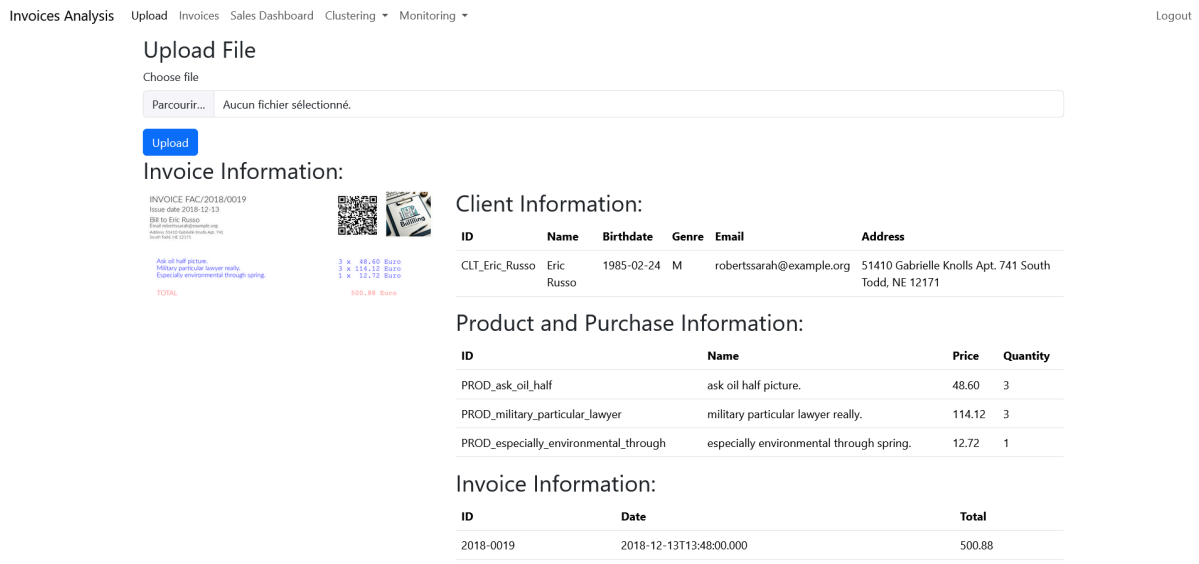
## 3.2. - Interface utilisateur

L’accès à l’application commence par nous emmener à la page de connexion (la création d’un compte n’est pas encore configurée et donc fonctionnelle).

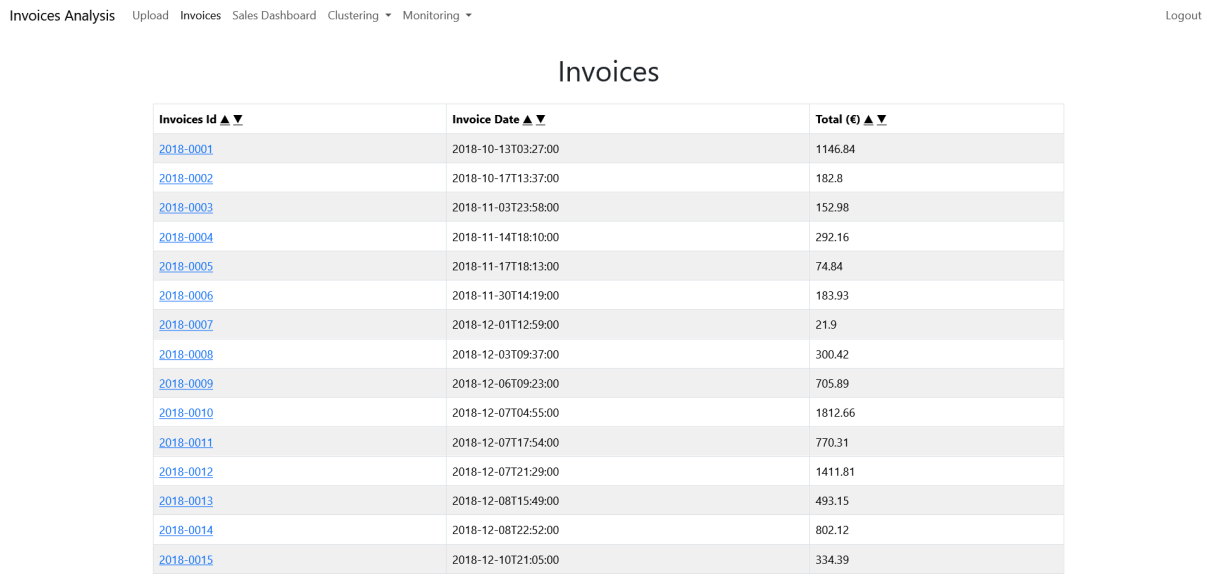



Après connexion, on est redirigé sur la page sur laquelle on peut uploader une fichier d’une facture.


Après avoir cliqué sur upload, l'image de la facture est affichée, ainsi que les informations extraites via OCR. Les informations ont aussi automatiquement été ajoutées dans la base de données si elles n'existaient pas déjà.

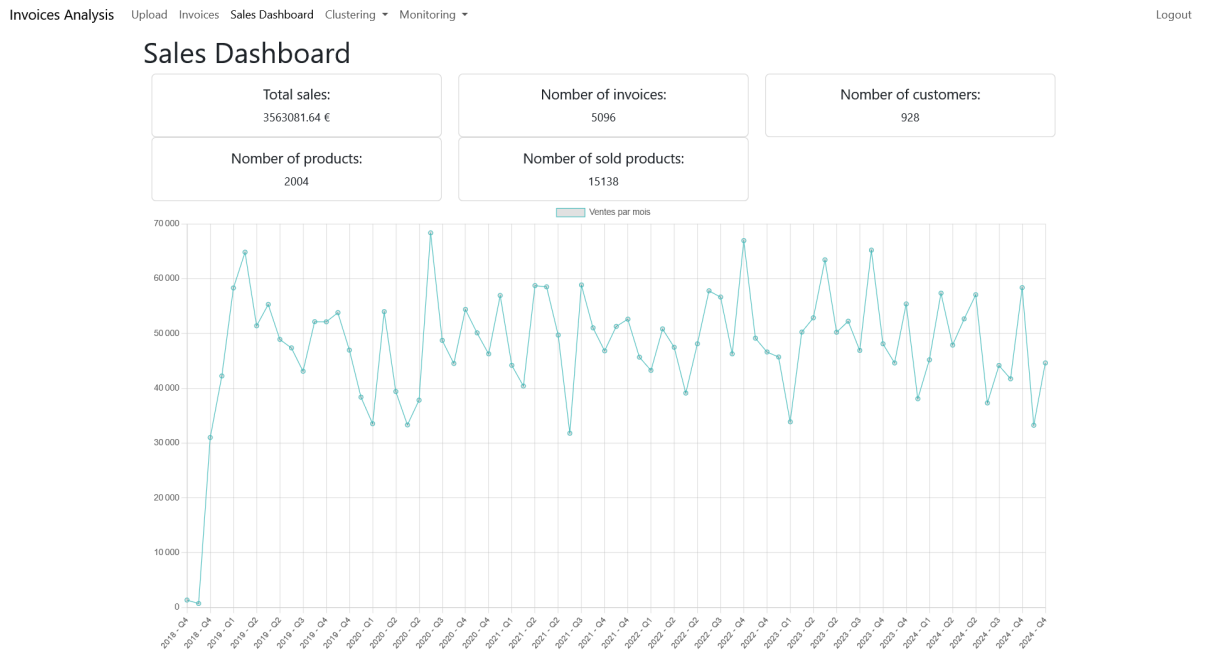



On peut aussi accéder à une table listant l'ensemble des factures en base de données. Cliquer sur un id de facture permet d'accéder à une page affichant les informations de cette formation, comme la partie droite de la page précédente.




| Invoices Id ▲▼ | Invoice Date ▲▼ | Total (€) ▲▼ |
|---|---|---|
| 2018-0001 | 2018-10-13T03:27:00 | 1146.84 |
| 2018-0002 | 2018-10-17T13:37:00 | 182.8 |
| 2018-0003 | 2018-11-03T23:58:00 | 152.98 |
| 2018-0004 | 2018-11-14T18:10:00 | 292.16 |
| 2018-0005 | 2018-11-17T18:13:00 | 74.84 |
| 2018-0006 | 2018-11-30T14:19:00 | 183.93 |
| 2018-0007 | 2018-12-01T12:59:00 | 21.9 |
| 2018-0008 | 2018-12-03T09:37:00 | 300.42 |
| 2018-0009 | 2018-12-06T09:23:00 | 705.89 |
| 2018-0010 | 2018-12-07T04:55:00 | 1812.66 |
| 2018-0011 | 2018-12-07T17:54:00 | 770.31 |
| 2018-0012 | 2018-12-07T21:29:00 | 1411.81 |
| 2018-0013 | 2018-12-08T15:49:00 | 493.15 |
| 2018-0014 | 2018-12-08T22:52:00 | 802.12 |
| 2018-0015 | 2018-12-10T21:05:00 | 334.39 |

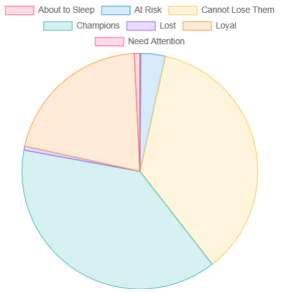
La partie Dashboard des factures permet de visualiser rapidement des informations statistiques sur les factures rentrées en base de données, avec des kpis comme les ventes totales ou le nombre de produits, ainsi qu'une graphique des ventes par mois.




Via la partie clustering de la barre de navigation, on peut accéder au clustering réalisé avec une segmentation RFM, ainsi que celui réalisé via un clustering k-means. Sur ces pages sont affichés un graphique circulaire représentant les différents groupes de clients et leur proportion, un tableau des statistiques moyennes de chaque groupe, ainsi qu'une table montrant tous les clients avec leur groupe, que l'on peut filtrer sur un groupe en cliquant sur la partie correspondante du graphique circulaire.

Invoices Analysis Upload Invoices Sales Dashboard Clustering Monitoring Logout

## Clusters based on RFM (Recency, Frequency, Monetary)




| Cluster | Total Expenses | Number of Orders | Days Since Last Invoice | Customer Id |
|---|---|---|---|---|
| Champions | 18269.51 | 7.42 | 186.19 | 356 |
| Cannot Lose Them | 7496.34 | 3.33 | 1074.49 | 334 |
| Loyal | 16684.05 | 6.32 | 379.32 | 193 |
| Need Attention | 26372.31 | 4.71 | 376.14 | 7 |
| Lost | 19143.15 | 1.8 | 1034.0 | 5 |
| At Risk | 14828.65 | 2.58 | 1029.26 | 31 |
| About to Sleep | 17713.8 | 2.0 | 349.0 | 1 |

| Customer | Total Expenses | Score Expenses | Number of Orders | Score Frequency | Days Since Last Invoice | Score Recency | Segment |
|---|---|---|---|---|---|---|---|
| CLT_Allen_Nelson | 37497.23 | 1 | 6 | 3 | 465 | 3 | Need Attention |
| CLT_Angela_Hughes | 26875.62 | 1 | 5 | 3 | 343 | 3 | Need Attention |
| CLT_Chad_Valdez | 26594.68 | 1 | 5 | 3 | 328 | 3 | Need Attention |
| CLT_Courtney_Irwin | 25013.46 | 1 | 6 | 3 | 473 | 3 | Need Attention |
| CLT_Pam_Perry | 7926.9 | 3 | 1 | 1 | 321 | 3 | Need Attention |
| CLT_Victor_Garrison_Jr. | 31696.39 | 1 | 6 | 3 | 391 | 3 | Need Attention |
| CLT_William_Ramsey | 29001.89 | 1 | 4 | 3 | 312 | 3 | Need Attention |

Sur la dernière partie de la barre de navigation, on accède aux pages de monitoring de l'OCR et du monitoring de l'api.
Le monitoring OCR nous permet de voir les performances de notre service OCR via le nombre de requêtes, le pourcentage d'erreurs, et le détail des erreurs.

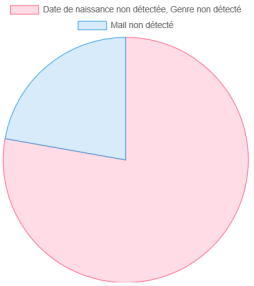
Invoices Analysis Upload Invoices Sales Dashboard Clustering Monitoring Logout

## Metrics

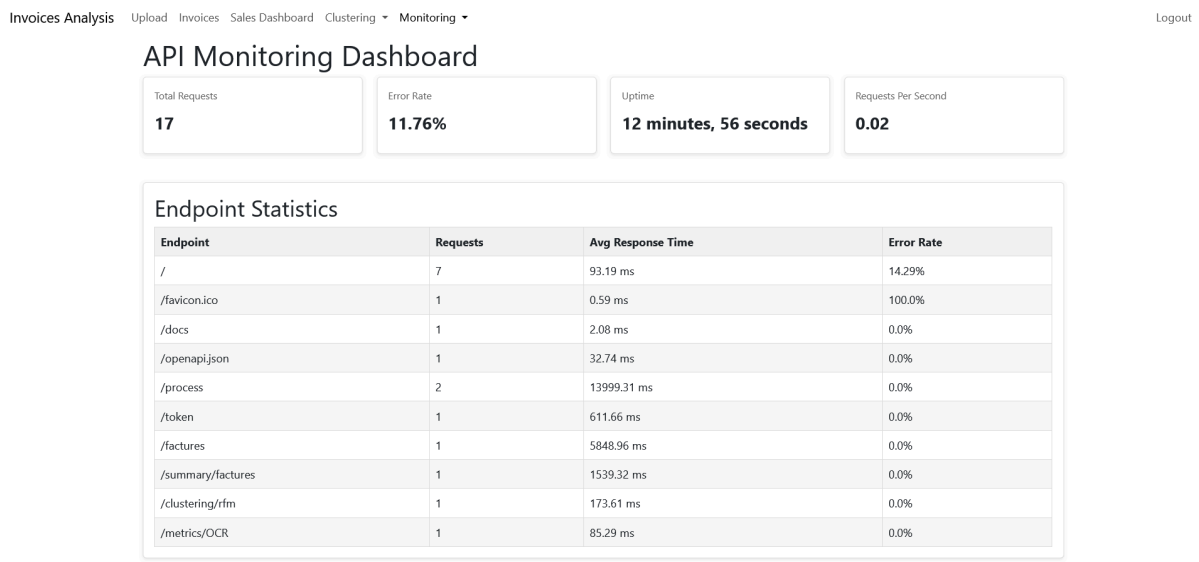
Total Requests
5096

Error Rate
0.53 %

### Errors Distribution




Enfin, La page de monitoring de l'api est reprise d'une page non faite par moi, et permet d'afficher les détails des requêtes sur notre api, et les éventuelles erreurs.




# 4. - Pipeline CI/CD

Pour garantir la qualité et la stabilité de l'application, j'ai intégré des tests automatisés dans un pipeline CI/CD à l'aide de GitHub Actions. Ce pipeline était configuré pour s'exécuter automatiquement à chaque push ou pull request sur la branche main.

Les tests incluent des tests unitaires pour vérifier le bon fonctionnement des différentes fonctionnalités de l'application, ainsi que des tests d'intégration pour valider les interactions entre les composants (API, base de données, etc.).

Cependant, j'ai rencontré des problèmes de cohérence et de compatibilité entre l'exécution des tests en local et sur GitHub Actions. Par exemple, certains tests qui passaient en local échouaient sur GitHub en raison de différences dans l'environnement d'exécution (versions de dépendances, configuration de la base de données, etc.).

Par manque de temps, j'ai dû mettre de côté cette intégration CI/CD pour me concentrer sur d'autres aspects du projet. Une amélioration future serait de résoudre ces problèmes de compatibilité pour réactiver le pipeline CI/CD et garantir une validation automatique des modifications

# 5. - Améliorations possibles

Mon application est encore en phase de développement et mériterait encore de nombreuses améliorations.

Tout d’abord, la gestion des erreurs est pour l’instant incomplète, et de nombreuses erreurs ne sont pas traitées.
Par exemple, si les mauvais identifiants sont rentrés sur la page de connexion, pour l’instant, il s’affiche juste une erreur inconnue, la mauvaise authentification n’est pas du tout gérée.

Ensuite, il est pour l’instant impossible de créer un nouveau compte via l’API, et donc impossible via l’application. Il n’existe donc pour l’instant qu’un seul compte, le compte par défaut.

Les analyses pourraient aussi être plus poussées, avec par exemple des graphiques représentant de façon plus visuelles les différences entre les différents clusters. On pourrait aussi ajouter des filtres sur la table listant les factures pour simplifier cette table très longue.

Enfin, les temps de chargement sont relativement longs, et les performances de l'application sont donc mauvaises. Il serait je pense possible d’améliorer ces temps de chargement en optimisant les traitements des données. De plus, on pourrait charger une partie de la page en attendant le retour de l’api au lieu de charger toute la page après le retour de l’api, pour éviter que l’utilisateur reste bloqué sur la page précédente. Stocker les données en cache serait par exemple aussi une solution pour réduire les temps de chargement.