

Avrile Floro (n° étudiant : 22000086)
Aude Hennino (n° étudiant : 19011803)
Maxime Bronny (n° étudiant : 19009314)

Réalisation de programme

Le 4 juin 2024



Table des matières

1	Introduction	3
2	Analyse du problème	4
3	Conception	5
4	Implémentation	7
4.1	Langage	7
4.2	Formattage du document et lecture par pyTesseract	7
4.2.1	Redressement d'une image penchée	8
4.2.2	Traitement de documents pdf	10
4.2.3	Délimitation des zones de texte grâce à des rectangles dessinés par l'utilisateur	10
4.3	Post-traitement des données	12
4.4	API de conversion des montants en devise autre que l'euro	14
4.5	Traduction des factures	15
4.6	Base de données	17
4.7	Pattern-matcher	19
4.8	Interface Utilisateur	19
4.9	Versioning	21
4.10	Packaging avec Hatchling	22
4.11	Procédure d'installation et de configuration	23
4.11.1	Contenu du dossier projet	23
4.11.2	Installations préalables à l'exécution du projet	23
4.11.3	Lancer le programme	23
5	Tests et validation	24
5.1	Tests OpenCV et pyTesseract	24
5.2	Tests de nettoyage du texte	24
5.3	Tests de traduction du texte	25
5.4	Tests du pattern matcher	25
5.5	Tests de la base de données	26
5.6	Tests de l'UI durant le processus de développement	28
6	Axes d'amélioration	29
7	Conclusion	30
	Annexes	31
A	Cahier des charges initial	31
B	Documentation Trello	31
C	Documentation GitHub	31
D	Tests de l'UI réalisés pendant le développement	31
E	Rapport personnel - Avrile Floro	32
F	Rapport personnel - Aude Hennino	32
G	Rapport personnel - Maxime Bronny	32
H	Vidéo d'installation du logiciel	32
I	Vidéo sur les tests	32
J	Vidéo d'utilisation du logiciel	32

Réalisation de programme

Rapport de réalisation

Avrile Floro
Étudiante n°22000086

Aude Hennino
Étudiante n°19011803

Maxime Bronny
Étudiant n°19009314

1 Introduction

Ce rapport met en lumière les étapes du projet sur lequel nous avons travaillé en groupe dans le cadre du cours "Réalisation de programme" de la licence 2 d'informatique à l'IED de l'Université Paris 8. Le projet rendu est un logiciel que nous avons baptisé "FactureFacile". Il s'agit d'un outil qui s'adresse principalement aux personnes privées ou aux entrepreneurs indépendants souhaitant avoir un suivi comptable de leurs dépenses sur la base des factures reçues. FactureFacile facilite la détection automatique des éléments-clés des factures, leur enregistrement méthodique et permet donc une comptabilité simple par poste de dépenses ainsi qu'une visualisation d'éléments statistiques. Il permet également d'obtenir des traductions de factures du français vers l'anglais, l'espagnol ou l'allemand.

Avant de commencer ce projet, nous (Avrile, Aude, Maxime) ne nous connaissions pas et n'avions donc encore jamais travaillé ensemble. La première visioconférence, celle du lancement du projet, a eu lieu avec le professeur Monsieur Gilles Bernard le 26 mars 2024. Nous venions tout juste de constituer le groupe et avons adopté le projet proposé par Avrile. À partir de ce moment, nous avons communiqué via un chat créé sur Google. En outre, nous avons organisé des points réguliers par visioconférence.

Dans ce rapport, nous expliquerons les étapes de développement du projet dans sa globalité, en reflétant les aspects organisationnels et techniques. Nos rapports personnels sont joints en annexe.

2 Analyse du problème

Le cahier des charges initial (fourni en annexe) était le suivant : le projet avait pour but de développer un système capable d'extraire le texte présent dans des images pour ensuite le traduire dans une langue étrangère choisie par l'utilisateur. Ce processus se ferait à travers une interface utilisateur conviviale. Le projet s'appuierait sur des technologies de traitement d'images et de traduction de texte. Nous avons décidé de ne pas inclure de composants d'apprentissage machine, et nous souhaitions mettre l'accent sur l'utilisation d'outils et de services existants pour le traitement d'image et la traduction.

Initialement, nous souhaitions proposer des traductions de tout type d'images, incluant des menus de restaurants, des textes et des factures. Néanmoins, l'usage de notre programme n'était pas très cohérent. Nous allions développer un logiciel à utiliser sur l'ordinateur alors que les cas d'usage (par exemple pour traduire le menu d'un restaurant) se prêtaient plutôt à une application mobile. Par ailleurs, la lecture par technologie OCR nous a paru plus intéressante et complexe à mettre en oeuvre sur des factures impliquant souvent un formatage hétérogène, avec notamment des colonnes dans l'en-tête de la facture et des zones de texte que pyTesseract ne pouvait pas différencier.

Nous nous sommes donc recentrés sur la lecture de factures avec proposition de traduction, un petit "plus" pouvant notamment servir aux utilisateurs ayant une maîtrise insuffisante du français. Nous avons ensuite étendu l'usage de la lecture des factures à un enregistrement dans une base de données avec conversion des montants en euros, permettant la tenue d'une comptabilité simple des dépenses. Nous avons également réfléchi à l'introduction d'un pattern-matcher pour automatiser la recherche des éléments-clés de la facture. Ces éléments sont également utilisés pour systématiser la sauvegarde des factures originales et leur traduction par date et type de dépense, facilitant ainsi la recherche de factures ex-ante, par exemple lors d'un contrôle fiscal (ce que nous ne souhaitons à aucun de nos utilisateurs...).

3 Conception

Notre équipe s'est mise au travail très vite avec une première répartition des tâches au cours de la première réunion de groupe. Au bout de la première semaine, nous étions en mesure de valider le concept de base, à savoir la lecture de documents, le nettoyage des textes et la traduction via des API. Initialement, la répartition des tâches a été la suivante :

- AvriLe : recherche des API de post-traitement du texte et de traduction
- Aude : prise en main de pyTesseract et OpenCV et tests de lecture de menus, de textes et de factures
- Maxime : prise en main de Trello et de GitHub

Après la validation initiale, nous avons décidé de nous concentrer sur la lecture OCR de factures, ce qui nous paraissait plus complexe que la lecture de textes ou de menus, souvent formatés de telle sorte que la lecture pouvait s'effectuer de haut en bas de façon homogène. Les premiers tests avec pyTesseract ont également fait émerger le besoin de pouvoir différencier les zones de texte dans les factures afin de permettre une meilleure lecture. En parallèle, nous avons commencé à développer l'UI, la base de données et le pattern-matcher :

- AvriLe : sélection de rectangles sur une image dans tkinter, base de données
- Aude : UI onglets 1, 2 et 4, PyTesseract/OpenCV, traitement des pdf
- Maxime : pattern matcher, UI onglets 3 et 5, API de conversion de devise

En réalité, nous avons également travaillé en binôme sur certains sujets (par exemple, pattern matcher : Maxime et AvriLe ; PyTesseract/OpenCV : Aude et AvriLe).

Enfin, dans une troisième phase, des éléments supplémentaires ont été répartis :

- AvriLe : documentation technique, conception et implémentation des tests de l'UI, packaging
- Aude : documentation utilisateur, coordination du rapport de réalisation
- Maxime : consolidation des deux parties de l'UI, amélioration du pattern matcher

Ici encore, nous avons travaillé en binôme sur certains sujets (par exemple, consolidation de l'UI : Maxime et Aude ; tests de l'UI : AvriLe et Aude).

En terme d'organisation, nous n'avons pas éprouvé le besoin de déterminer un chef pour les raisons suivantes :

- Petite équipe (3 membres), réduisant les besoins de coordination entre les différents membres
- Chacun des membres était ouvert aux propositions et aux compromis, il n'y a pas eu besoin de médiation et nous n'avons pas eu de situation de décision difficile à prendre collectivement
- Chaque membre de l'équipe testait la version de l'UI sur laquelle il venait de travailler avant de la mettre en ligne, et la personne reprenant l'UI pour y faire ses modifications le communiquait par chat Google. Toute erreur trouvée était également signalée dans le chat.

En effet, la communication au quotidien sur nos avancées respectives ou les bugs repérés et à corriger s'est principalement effectuée dans notre chat Google. De plus, nous faisions une réunion par semaine, chacune de ces visioconférences hebdomadaires durant entre 1h30 et 2h (mais nous avons fait également plus court, une fois!). Il n'a pas toujours été simple de trouver un horaire convenant à tout le monde, entre nos obligations professionnelles et le fait d'habiter sur des fuseaux horaires différents.

Nous avons également tenu un inventaire des tâches et de leur répartition grâce à l'outil Trello, qui nous a paru suffisant et efficace sur ce projet (figure 1). Maxime avait préparé un document expliquant au reste du groupe comment utiliser Trello (document mis en annexe).

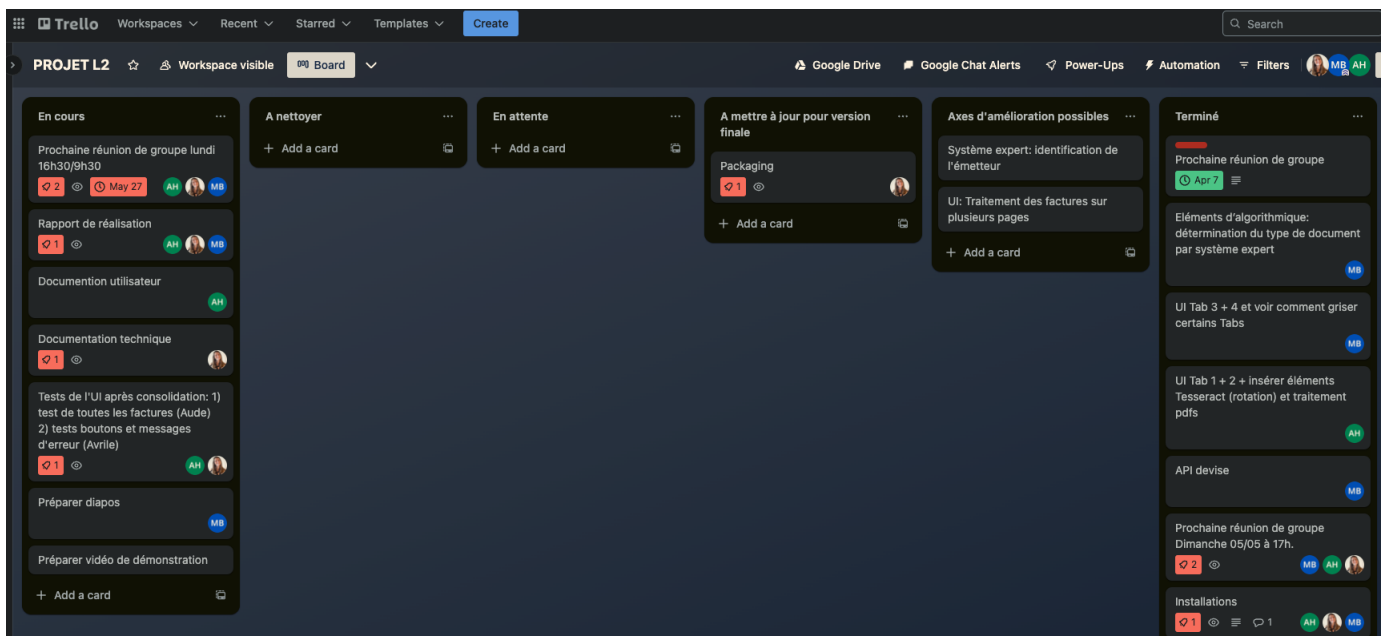


FIGURE 1 – Capture d’écran de notre tableau de bord collaboratif sur Trello.

4 Implémentation

Nous présentons ci-dessous les principales étapes et choix réalisés lors de la mise en oeuvre du concept et la phase de programmation.

4.1 Langage

Nous avons posé les bases du langage très en amont, dans la mesure où l'idée initiale consistait à lire des images grâce à l'outil OCR Tesseract, ce qui impliquait de préparer les images permettant une meilleure lecture par Tesseract. Il s'avère que la version de Tesseract pour Python, pyTesseract, conjuguée à l'utilisation d'OpenCV et de Pillow, permet un traitement et une lecture efficace des images. Par ailleurs, nous savions que le nombre de données à traiter serait relativement limité, tant par la longueur des textes traités que par le nombre de textes traités et le nombre de variables recueillies. Enfin, nous avons vérifié la possibilité d'utiliser une interface graphique dans Python, tkinter ainsi que de faire appel à la base de données MySQL. Ainsi, Python nous a paru le langage le plus adapté à ce projet.

4.2 Formattage du document et lecture par pyTesseract

Il est apparu lors des tests qu'une lecture directe des factures par pyTesseract donnaient des résultats peu probants dans un certain nombre de cas, du fait de l'hétérogénéité du format des factures. Nous avons testé l'utilisation de fonctionnalités d'OpenCV pour isoler les zones de texte indépendantes les unes des autres par formation de rectangles (figure 2) en appliquant un processus de recherche de zones automatique, notamment par dilatation du texte (figure 3).

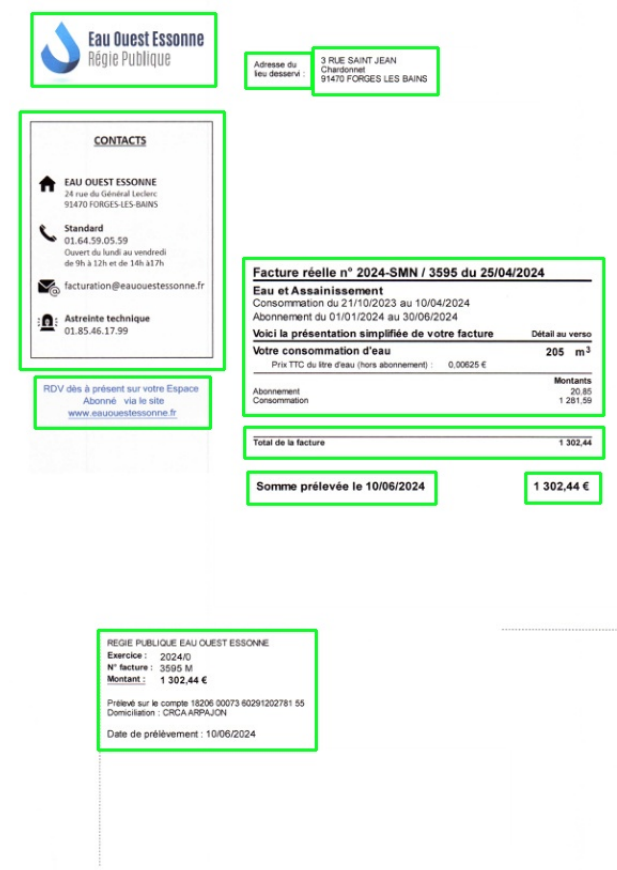


FIGURE 2 – Détection automatique de zones de texte indépendantes les unes des autres

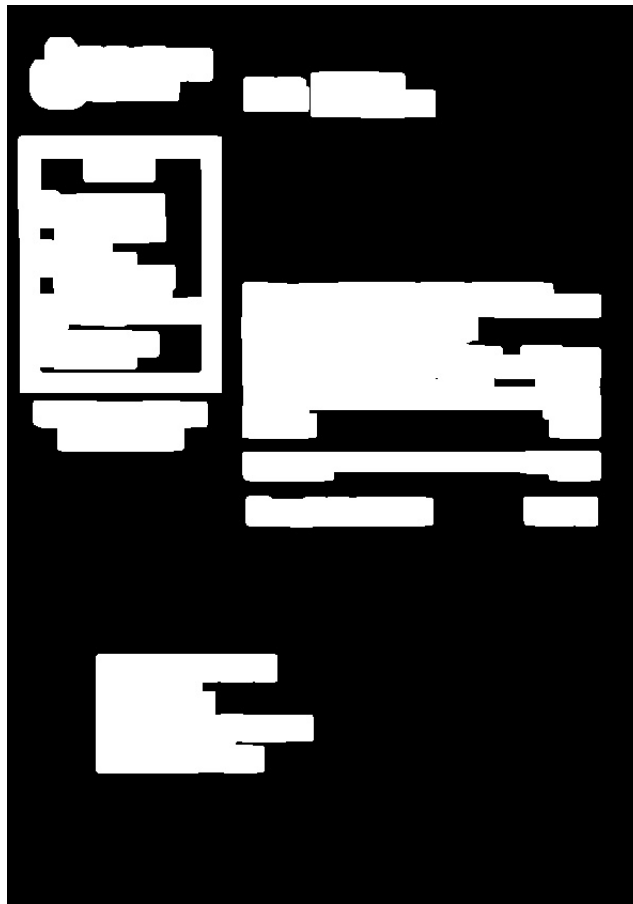


FIGURE 3 – Dilatation du texte pour faire ressortir les différentes zones à séparer pour la lecture

Après un certain nombre de tests, nous en avons conclu que la recherche automatique des zones de texte n'était pas optimale. Nous avons préféré une solution manuelle, où l'utilisateur pourrait sélectionner les zones de texte par lui-même. Nous avons néanmoins laissé la possibilité pour le programme de lire une facture dans son entièreté si l'utilisateur ne sélectionne pas de zone de texte.

4.2.1 Redressement d'une image penchée

Dans la mesure où la photo de certaines factures entrantes peut être légèrement penchée, il nous paraissait important de redresser ces factures afin que les lignes présentées à Tesseract soient les plus horizontales possibles. Nous nous sommes inspirés pour ce faire de fonctions trouvées sur internet, et les avons adaptées à notre situation.

Dans un premier temps, une fonction détecte l'angle de rotation de la facture originale présentée en utilisant le même principe que dans l'étape présentée précédemment. Il s'agit d'identifier et de délimiter les différentes zones de texte, notamment par dilatation des caractères. Après cette étape, on applique la fonction `minAreaRect`, qui détecte le rectangle le plus petit contenant toutes les informations d'une zone de texte. La fonction initiale n'appliquant `minAreaRect` qu'à la zone de texte la plus grande de l'image, il s'est avéré que dans certains cas, l'angle de rotation était détecté comme non nul alors que la facture était en réalité droite (exemple figure 4). Nous avons donc étendu l'application de `minAreaRect` aux huit zones de texte les plus grandes pour chaque facture (sauf pour les cas où le nombre de zones de texte détectées est inférieur à huit), et de ne conserver que l'angle de rotation le plus fréquent.

Dans un deuxième temps, une fonction permet d'appliquer l'angle de rotation à l'image par calcul du point central de l'image et rotation autour de ce point pour la redresser.

Les fonctions de calcul de l'angle de rotation et de redressement de l'image se trouvent dans le fichier `tesseract.py`.

4.2.2 Traitement de documents pdf

De nos jours, beaucoup de factures sont reçues par email sous la forme d'un document pdf. Il fallait donc pouvoir étendre la fonction de lecture de documents à ce format car nous n'avions initialement pris en considération que les documents entrants au format image. La solution initialement mise en place pour la lecture de documents au format pdf impliquait la conversion de la facture au format image en utilisant la bibliothèque pdf2image. Cependant, nous avons remarqué au moment du packaging que cette bibliothèque sur la base d'une technologie qui n'était pas en lien avec Python (et qu'il fallait télécharger à part). Cette solution ne pouvait donc pas être intégrée dans la solution de packaging choisie, à moins d'actions supplémentaires d'installation de l'utilisateur. Nous l'avons donc remplacée par la bibliothèque PyMuPDF permettant également la conversion de documents pdf en images, et pouvant être installée avec le reste du packaging, sans nécessiter d'installation supplémentaire.

4.2.3 Délimitation des zones de texte grâce à des rectangles dessinés par l'utilisateur

Pour améliorer la précision de l'extraction de texte sur les factures, nous avons implémenté une fonctionnalité permettant aux utilisateurs de dessiner des rectangles sur le canvas afin de délimiter manuellement les zones de texte à traiter. Cette approche offre une solution adaptable aux mises en page variées des factures, en permettant un traitement séparé de chaque zone définie.

Problématique

L'extraction du texte des factures est effectuée grâce à OpenCV et Tesseract, utilisant des technologies de reconnaissance optique de caractères (OCR). Cependant, l'utilisation de ces technologies a montré certaines limites avec certains types de documents, en particulier les factures. Les factures présentent de nombreuses zones de texte hétérogènes avec plusieurs colonnes et mises en page sur une même page, rendant les réglages généralisés de Tesseract assez peu efficaces.

Pour répondre à cette problématique, nous avons convenu qu'il était nécessaire de laisser à l'utilisateur la possibilité de sélectionner lui-même certaines zones de texte sur la facture. De cette manière, les résultats obtenus par Tesseract et OpenCV sont beaucoup plus satisfaisants, car chaque zone de texte peut être traitée de façon séparée.

Initialement, nous avons envisagé de demander à l'utilisateur de sélectionner certains types d'informations précises sur la facture (par exemple l'émetteur, la date, le prix, etc.). Cependant, cette approche s'est révélée assez peu efficace, en raison de la grande hétérogénéité des factures et des informations présentées. Nous avons donc opté pour la solution des rectangles, permettant à l'utilisateur de dessiner des rectangles sur le canvas pour délimiter les zones de texte à traiter. Il s'agissait de l'implémentation la plus simple possible nous permettant de récupérer les zones de texte.

Implémentation au sein du projet

Pour permettre à l'utilisateur de dessiner des rectangles sur le canvas, plusieurs fonctions ont été implémentées dans le fichier `UI.py` :

- `on_canvas_click` : Cette fonction est appelée lors du clic sur le canvas. Elle stocke la position initiale du clic et commence à dessiner un rectangle minuscule.
- `on_canvas_drag` : Cette fonction est appelée lors du glissement de la souris avec le bouton pressé. Elle met à jour les coordonnées du rectangle pour l'agrandir en fonction de la position actuelle de la souris.
- `on_canvas_release` : Cette fonction est appelée lors du relâchement du bouton de la souris. Elle finalise les coordonnées du rectangle et ajoute le rectangle à une liste pour un traitement ultérieur.

- **clear_last_rectangle** : Cette fonction permet d'effacer le dernier rectangle ajouté. Elle vérifie s'il y a des rectangles à effacer, puis enlève le dernier rectangle de la liste et le supprime du canvas.
- **validate_all** : Cette fonction est utilisée pour valider tous les rectangles dessinés et extraire le texte des zones correspondantes. Elle trie les rectangles par ordre vertical et horizontal, et utilise Tesseract pour extraire le texte de chaque zone délimitée par les rectangles. Le texte extrait est ensuite nettoyé et sauvegardé.

L'utilisateur peut ainsi dessiner des rectangles sur le canvas pour sélectionner les zones de texte à traiter. Une fois les rectangles dessinés, ils peuvent être validés pour extraire et traiter le texte de manière plus précise. Cette approche améliore significativement les résultats obtenus par Tesseract et OpenCV, en permettant un traitement séparé de chaque zone de texte.

Les événements du canvas sont liés aux fonctions correspondantes :

- Le clic sur le canvas appelle la fonction **on_canvas_click**.
- Le glissement de la souris avec le bouton pressé appelle la fonction **on_canvas_drag**.
- Le relâchement du bouton de la souris appelle la fonction **on_canvas_release**.

Ces fonctionnalités permettent à l'utilisateur de manipuler facilement les zones de texte à traiter.

4.3 Post-traitement des données

Dans le cadre de notre projet, nous avons estimé qu'un post-traitement des données était nécessaire. L'extraction de texte par OCR (Optical Character Recognition) peut en effet conduire à des caractères indésirables, des erreurs typographiques, et des anomalies qui peuvent altérer la qualité de la traduction. Pour garantir une traduction de qualité, il est crucial que le texte à traduire soit propre et exempt d'erreurs.

Décision d'avoir recours à une IA générative

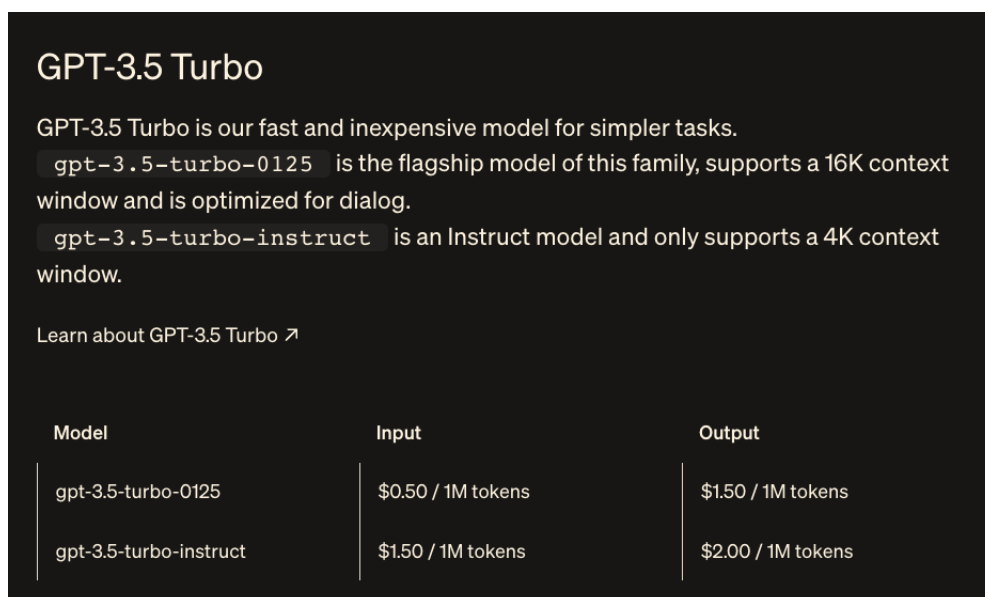
Nous n'avons pas trouvé de bibliothèque permettant un post-traitement efficace du texte extrait, c'est la raison pour laquelle nous avons décidé de faire appel à une IA générative. Notre objectif initial était de rester sur des solutions gratuites.

Choix de Gemini

C'est Avrilie qui a recherché les IA génératives disponibles avec un tiers gratuit. Elle est résidente en Colombie. Elle avait initialement choisi Gemini, l'IA de Google. L'offre proposée par Gemini était parfaite pour notre usage, avec une gratuité jusqu'à 15 requêtes par minute, ce qui correspondait exactement à nos besoins. Cependant, lors de tests plus approfondis du projet, nous avons découvert que l'API de Gemini est interdite d'utilisation en Europe en raison de la législation sur la protection des données. Cette contrainte nous a forcés à chercher une alternative accessible en Europe.

Utilisation définitive de OpenAI

Après avoir exploré diverses options gratuites sans résultats satisfaisants, nous nous sommes résolus à utiliser l'API payante d'OpenAI, c'est-à-dire son célèbre Chat GPT 3.5 Turbo. Bien que cela ait contredit notre souhait initial de gratuité, nous avons pris 15\$ de crédit auprès de cette API, ce qui dépasse largement nos besoins en post-traitement (5 à 10\$ auraient probablement été suffisants). Cette API payante, bien qu'à un prix très abordable, nous a permis d'obtenir des résultats de très bonne qualité et délivrés de façon rapide, ce qui est non négligeable.



GPT-3.5 Turbo

GPT-3.5 Turbo is our fast and inexpensive model for simpler tasks.

`gpt-3.5-turbo-0125` is the flagship model of this family, supports a 16K context window and is optimized for dialog.

`gpt-3.5-turbo-instruct` is an Instruct model and only supports a 4K context window.

[Learn about GPT-3.5 Turbo ↗](#)

Model	Input	Output
<code>gpt-3.5-turbo-0125</code>	\$0.50 / 1M tokens	\$1.50 / 1M tokens
<code>gpt-3.5-turbo-instruct</code>	\$1.50 / 1M tokens	\$2.00 / 1M tokens

FIGURE 5 – La tarification de l'API GPT-3.5 Turbo.

Appel à l'API au sein du programme

La fonction de correction de texte `nettoyage_texte_txt`, qui fait appel à l'API de OpenAI, se trouve dans le document `openai_deepl.py`. Elle est appelée après l'extraction initiale du texte et avant le renvoi à l'utilisateur, de façon à ce que ce dernier soit exposé directement à un rendu aussi propre que possible. L'utilisateur a lui-même l'opportunité de modifier ensuite le contenu si nécessaire.

La documentation de l'API fournit des exemples de fonctions à utiliser pour extraire et identifier les réponses de l'API (source : <https://platform.openai.com/docs/guides/text-generation/json-mode>).

4.4 API de conversion des montants en devise autre que l'euro

RapidAPI est fréquemment utilisé pour la conversion de devises en raison de sa vaste sélection d'APIs, y compris celles offrant des taux de change en temps réel. Les développeurs apprécient RapidAPI pour sa facilité d'utilisation et sa documentation complète, ce qui facilite l'intégration des fonctionnalités de conversion de devises dans les applications ou les sites web.

De plus, RapidAPI propose souvent des plans tarifaires flexibles, adaptés à une large gamme de projets, allant des petits développements expérimentaux aux grandes applications commerciales. En outre, RapidAPI prend en charge de nombreux langages de programmation, comme illustré ci-dessous. Dans notre cas, nous avons particulièrement privilégié les options gratuites lorsque cela était possible. Grâce à RapidAPI, une API permettant jusqu'à 1 000 conversions de devises par mois a été trouvée, alors que d'autres solutions comme CurrencyLayer et Fixer offraient entre 100 et 250 conversions gratuites par mois.

The screenshot displays the RapidAPI interface for the 'Conversion des devises et taux de change' API. The top navigation bar includes links for 'er des API', 'Centre d'API', 'Organisations', 'applications', and 'Mes API'. The API details section shows the API name, a 'GRATUITMIUM' badge, a 'Vérifié' status, a popularity score of 9,8 / 10, a latency of 304 ms, a service level of 100%, and a health status of N / A. The 'Optional Parameters' section shows the 'from' parameter set to 'USD' and the 'to' parameter set to 'EUR,GBP'. A sidebar on the right lists supported programming languages: C, Clojure, C#, Go, HTTP, Java, JavaScript, Kotlin, Node.js, Objective-C, OCaml, PHP, Powershell, Python, R, and RapidQL. The bottom of the interface shows the footer with copyright information and social media links.

FIGURE 6 – Présentation de l'interface lors de la sélection d'une API, ainsi que la répartition des fonctionnalités. À droite, on peut voir les différents langages de programmation pris en charge par l'API.

Voici les prix proposés par cette API :

Conversion des devises et taux de change

Par API principales | Mis à jour il y a un mois | Finance

GRATUITIUM

Vérifié

Popolarité

Latence

Niveau de service

Bilan de santé

9,8 / 10

304 ms

100%

N / A

Points de terminaison
À propos
Tutoriels
Discussions
Tarifs

Choisissez le forfait qui vous convient

RapidAPI s'associe directement aux fournisseurs d'API pour vous proposer une tarification transparente et simple. Trouvez un plan qui correspond le mieux à l'échelle dont vous avez besoin pour votre application.

Maxime Bro

Objets	Basique 0,00 \$/mois	Pro 10,00 \$/mois	Ultra 25,00 \$/mois	Méga 50,00 \$/mois
	S'abonner	S'abonner	S'abonner	S'abonner
Demandes ☺	1 000 / mois Limite stricte	10 000 / mois + 0,01 \$US les uns les autres	50 000 / mois + 0,01 \$US les uns les autres	300 000 / mois + 0,01 \$US les uns les autres
Caractéristiques				
Point final de la série chronologique	×	×	✓	✓
Tarification personnalisée ☺	×	×	×	×
Caractéristiques	1000 requêtes par heure			

FIGURE 7 – Présentation des plans tarifaires proposés par le développeur.

4.5 Traduction des factures

Nous avons souhaité offrir à l'utilisateur la possibilité de traduire le contenu de ses factures, dans l'hypothèse d'un utilisateur maîtrisant mal le français ou d'un environnement international.

Choix de l'API de traduction

Nous souhaitions utiliser une solution gratuite pour la traduction. Nous avons testé plusieurs options, y compris l'API de Google Translate. Cependant, après plusieurs essais, nous avons conclu que l'API de DeepL était la plus efficace. Cette API offre une meilleure qualité de traduction, particulièrement lorsque le texte est complexe et nécessite de conserver certains mots dans leur langue d'origine, tels que les noms propres.

Le principal inconvénient de l'API de DeepL est sa limitation à 500 000 caractères par mois (dans sa version gratuite), ce qui peut être restrictif. Pour contourner cette limitation, nous avons obtenu deux clés API. Avant chaque traduction, nous vérifions le nombre de caractères déjà traduits depuis le début du mois. En fonction de cette vérification, nous choisissons l'une ou l'autre clé, garantissant ainsi que nous ne dépassons pas la limite mensuelle de 500 000 caractères par clé. Cela a pour désavantage de ralentir le programme avec un appel supplémentaire à la base de données pour effectuer ce contrôle.

Nous avons estimé qu'une quantité d'un million de caractères traduits par mois serait suffisante, étant donné que toutes les factures ne nécessitent pas de traduction.

Implémentation au sein du projet

Nous avons intégré la fonction de traduction `traduire_texte` dans le document `openai_deepl.py`. Le texte post-traité, que l'utilisateur a eu l'opportunité de corriger, est ensuite passé à l'API de traduction. Cela garantit un résultat de traduction encore plus satisfaisant.

Trois langues sont disponibles pour la traduction : l'anglais, l'espagnol et l'allemand. Cette fonctionnalité est accessible via l'interface utilisateur, permettant aux utilisateurs de choisir la langue de traduction souhaitée.

4.6 Base de données

Il était essentiel de choisir une solution d'hébergement de base de données qui soit à la fois fiable, gratuite et permettant une connexion à distance. Après avoir évalué plusieurs options, nous avons opté pour Microsoft Azure, en utilisant l'offre "Azure for Students". Cette décision nous a permis de bénéficier d'un hébergement gratuit et d'une connexion à distance, répondant ainsi parfaitement à nos besoins tout en minimisant les coûts.

Choix du serveur et de l'hébergeur

Pour héberger notre base de données, nous avons choisi Microsoft Azure, plus précisément l'offre "Azure for Students". Cette décision a été motivée principalement par :

- **Gratuité pour les étudiants** : L'offre Azure pour les étudiants est gratuite (dans une certaine limite), ce qui correspond parfaitement à notre besoin de minimiser les coûts.
- **Connexion à distance** : De nombreuses offres gratuites ne permettent pas la connexion à distance (*remote MySQL*), ce qui était crucial pour notre projet.

Voici la configuration de notre base de données : **Burstable**, **B1ms**, **1 vCores**, **2 GiB RAM**, **32 GiB storage**.

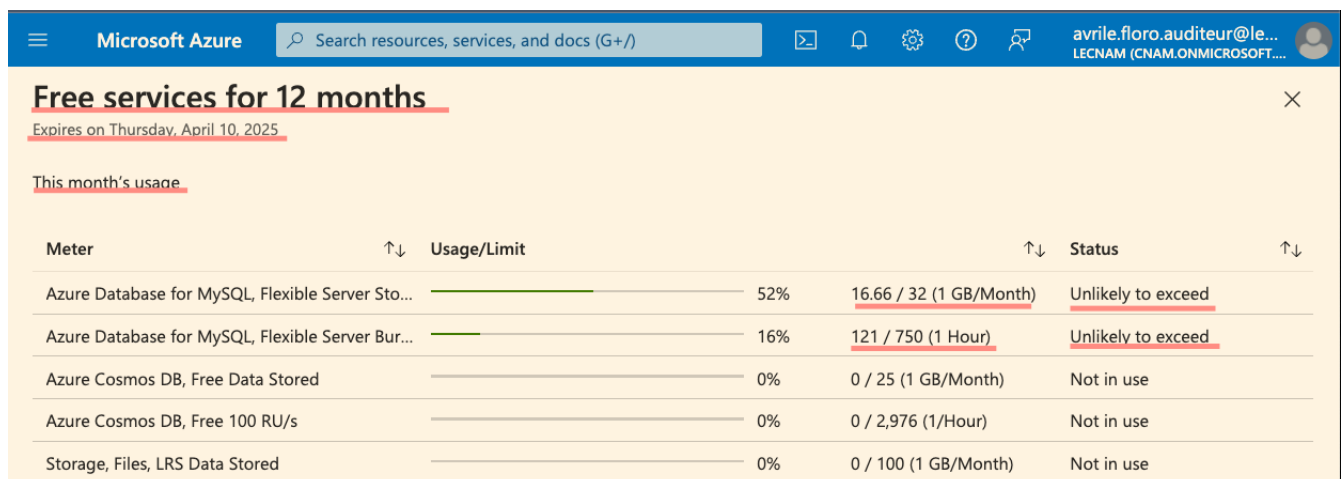


FIGURE 8 – L'offre "Azure for Students" permet d'utiliser gratuitement des services pendant 12 mois. Il est possible de suivre sa consommation.

Choix de MySQL

Pour gérer les données de notre projet, nous avons choisi MySQL pour plusieurs raisons :

- **Limitation des API gratuites** : Nous étions limités par le nombre de caractères traduits mensuellement avec la version gratuite de l'API de DeepL et le nombre de conversions de devises effectuées sur une période de 30 jours avec l'API gratuite de RapidAPI. Il était donc nécessaire de garder une trace précise du nombre de conversions effectuées et du nombre de caractères traduits.
- **Stockage et suivi des mouvements** : MySQL nous permet de stocker toutes les données de manière centralisée et de suivre les mouvements effectués. Contrairement à SQLite, qui ne retient les données qu'en local, MySQL nous offre une solution pour suivre nos données à distance.
- **Intégration avec Python** : MySQL est facile à utiliser avec Python grâce à des bibliothèques bien supportées comme `mysql.connector`, ce qui simplifie l'implémentation et la gestion de la base de données dans notre projet.

Microsoft Azure Search resources, services, and docs (G+)

Home > **projet-trad** Azure Database for MySQL flexible server

Search Connect View process list Delete Reset password Restore Restart

Overview

- Activity log
- Access control (IAM)
- Tags
- Diagnose and solve problems
- Learning center

Settings

- Compute + storage
- Networking
- Databases
- Connect
- Server parameters
- Replication

Essentials

Subscription ([move](#)) [Azure for Students](#)

Subscription ID: 80218779-a071-4e17-b342-233566be8efe

Resource group ([move](#)): [projet-trad](#)

Status: Available

Location: France Central

Tags ([edit](#)) [Add tags](#)

[JSON View](#)

Server name: projet-trad.mysql.database.azure.com

Server admin login name: headofappli

Configuration: [Burstable, 81ms, 1 vCores, 2 GiB RAM, 32 GiB st...](#)

MySQL version: 8.0

Availability zone: --

Created On: 2024-04-10 21:37:29.6956583 UTC

FIGURE 9 – Sur le site de Microsoft Azure, nous pouvons consulter les informations relatives à la base de données, notamment le serveur, le nom de l’administrateur, la configuration de la base de données et la version de MySQL utilisée.

Ainsi, l’utilisation de Microsoft Azure pour héberger notre base de données MySQL a répondu à nos besoins en offrant une option gratuite et une connexion à distance. Le choix de MySQL a facilité l’intégration avec Python et permis un suivi à distance de nos données.

4.7 Pattern-matcher

Initialement, nous avons envisagé de développer un système expert pour notre projet sans avoir une idée précise des fonctionnalités à intégrer. Dans un premier temps, nous avons opté pour une approche générale afin de comprendre l'utilité d'un système expert et son rôle potentiel dans un projet de ce type. Nous avons décidé d'incorporer un système expert en utilisant la bibliothèque Python `experta`.

Finalement, nous avons opté pour un pattern-matcher, car nous avons estimé qu'un système expert n'était pas nécessaire dans notre cas. De plus, la bibliothèque `experta` que nous utilisions posait des problèmes lors du packaging car elle n'était plus maintenue.

Le principe du pattern-matcher consiste à identifier des motifs spécifiques dans des ensembles de données. Un motif peut être une séquence ou une structure particulière que l'on souhaite repérer. Par exemple, les expressions régulières (regex) permettent de trouver des mots précis dans un texte, tandis que les arbres syntaxiques analysent des structures dans des langages de programmation ou des phrases en langage naturel. Cela nous a permis de rechercher des similarités au sein des factures que nous souhaitions traiter. Les fonctionnalités que nous avons initialement prévues pour le système expert ont été transformées en fonctions simples. Ces fonctions nous permettent d'extraire le montant, la date, la catégorie, et le type de devises, ainsi que d'enregistrer automatiquement chaque fichier de manière triée par date et type, puis de renommer le fichier en utilisant les informations obtenues lors du traitement de chaque facture par le logiciel.

Les fonctions de pattern matching se trouvent dans le fichier `pattern_matcher.py`.

4.8 Interface Utilisateur

L'interface graphique, que nous appellerons "UI", a été implémentée avec l'outil tkinter. Avant de commencer à travailler sur tkinter, qui s'est avéré être un nouvel outil pour nous, nous avons préparé un schéma simplifié du layout attendu de l'interface graphique, qui nous a servi de guide lors de la programmation de l'UI (figure 10).

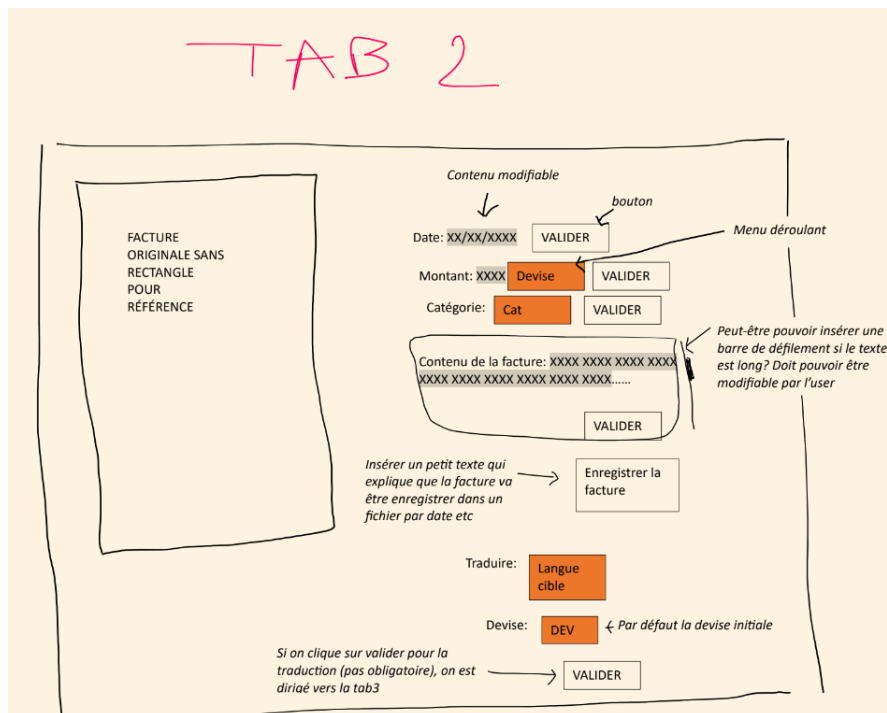


FIGURE 10 – Exemple de sketch initial nous ayant servi de modèle de développement pour l'UI.

Tkinter est largement choisi pour le développement d'interfaces graphiques (GUI) en Python en raison de plusieurs facteurs clés. Tout d'abord, son intégration native avec Python est un avantage majeur, étant inclus dans la distribution standard de Python, ce qui signifie qu'aucune installation supplémentaire n'est nécessaire. De plus, connu pour sa facilité d'utilisation, simple et intuitive qui la rend accessible même aux débutants en programmation GUI. Les concepts de bases comme les fenêtres, les boutons et les champs de texte sont facilement réalisables.

Un autre avantage est sa nature multiplateforme, elle permet aux applications développées avec Tkinter de fonctionner de manière cohérente sur Windows, macOS et Linux, assurant ainsi une portabilité importante. Enfin, la communauté active qui entoure Tkinter est un atout supplémentaire, offrant aux développeurs un accès à des astuces, des conseils et des bibliothèques supplémentaires pour étendre les fonctionnalités de Tkinter selon leurs besoins spécifiques.

Pour notre projet, Tkinter était donc parfaitement adapté à l'application que nous souhaitions développer.

Dans tkinter, il y a plusieurs méthodes pour positionner les objets et les widgets. Nous avons choisi d'utiliser grid, qui nous paraissait plus simple d'approche que pack, et qui nous a permis de réaliser les mises en page que nous souhaitions.

L'UI a été construite autour de 5 onglets principaux. Par ailleurs, certaines fonctionnalités (activées par des boutons) ouvrent des fenêtres supplémentaires, par exemple dans l'onglet 1 (Manuel d'utilisation) et l'onglet 4 (Voir une facture et Voir la traduction d'une facture).

L'UI utilise en backend un dossier temporaire appelé "factures_temporaire", qui est créé à chaque ouverture de l'UI et détruit à chaque fermeture de l'UI. De plus, l'UI sauvegarde toute facture validée dans l'onglet 2, ainsi que sa traduction, le cas échéant, dans un dossier appelé "trie_doc". Ce dossier est structuré de manière à trier les factures par année, puis par mois, puis par catégorie de dépenses. Cette logique de classification nous a paru la plus appropriée dans le cas d'un contrôle fiscal, car le contrôleur demandera les factures liées à une année donnée. La sous-division par mois et type de dépense permet d'affiner la classification.

Les onglets 1 à 3 utilisent un certain nombre de variables globales en commun, dans la mesure où ils traitent une facture à enregistrer sélectionnée par l'utilisateur dans l'onglet 1. Ces variables sont réinitialisées à chaque chargement d'une nouvelle facture dans le premier onglet. Le troisième onglet est optionnel car il permet de visualiser la traduction d'une facture. Ainsi, cet onglet n'est chargé qu'au moment où l'utilisateur confirme sa demande de traduction dans l'onglet 2. Cela permet une économie de mémoire lors de l'utilisation de l'UI.

Nous avons fait en sorte de griser certains onglets ainsi que certains boutons de validation tant que certaines étapes n'étaient pas franchies. Ainsi, l'onglet 2 est inaccessible tant que l'utilisateur n'a pas validé une nouvelle facture chargée dans le premier onglet. De même, l'onglet 3 est inaccessible tant que l'utilisateur n'a pas confirmé qu'il souhaitait une traduction en bas du deuxième onglet. Dans l'onglet 2, par exemple, le bouton permettant de visualiser le texte (pour le vérifier en cas de traduction demandée) n'est accessible que si l'utilisateur a confirmé tous les éléments-clés de la facture (ce qui déclenche l'enregistrement de la facture dans la base de données).

L'onglet 1 fait appel à la fonction de calcul de l'angle de la facture (pour la redresser si nécessaire), à l'API de nettoyage du texte retourné par l'application de pyTesseract sur l'image, ainsi qu'au pattern matcher afin de détecter les éléments-clés de la facture après lecture par Tesseract. Ces éléments-clés sont affichés dans l'onglet 2. L'onglet 2 fait appel à la base de données MySQL en écriture (entrée d'une nouvelle facture avec affectation d'un identifiant facture unique, et inscription d'informations complémentaires en cas de traduction). Il fait également appel à l'API de traduction. Les onglets 1, 2 et 3 écrivent dans le dossier temporaire "factures_temporaire", et les onglets 2 et 3 dans le dossier de classification "trie_doc".

L'onglet 4 est indépendant des autres et permet une visualisation d'ensemble de la comptabilité ainsi que d'appeler le détail de postes de dépenses et enfin, de visualiser les détails par facture. Il fonctionne avec des boutons de validation et ne se met donc pas à jour de façon continue. Il fait appel à la base de données MySQL principalement en lecture (affichage des montants par poste de dépense, affichage du détail d'un poste de dépense sélectionné, affichage des détails d'une facture, recherche du chemin où est sauvegardée la facture et sa traduction par identifiant facture), mais également en écriture (suppression d'une facture). Cet onglet intervient en lecture (visualisation d'une facture et de sa traduction) et écriture (suppression d'une facture et de la traduction) du dossier de classification "trie_doc".

L'onglet 5 est également indépendant des autres et, contrairement à l'onglet 4, il se met à jour chaque fois que l'utilisateur l'affiche ou modifie une date. Le contenu affiché dans cet onglet est actualisé en temps réel en interrogeant la base de données MySQL, garantissant que les résultats présentés sont toujours à jour. Comme l'onglet 4, il se concentre principalement sur la lecture de données depuis MySQL. Les boutons de sélection de dates sont conçus pour être aussi intuitifs que possible : si l'utilisateur choisit une année, le bouton du mois revient par défaut à "Choisir un mois", affichant les résultats uniquement par année sans sélection de mois et désactivant complètement la possibilité de sélectionner un mois.

4.9 Versioning

Nous nous sommes penchés dès le départ sur la prise en main de GitHub, avec lequel aucun d'entre nous n'avait encore travaillé dans le cadre d'un projet collaboratif. Nous n'avons pas réellement utilisé GitHub de façon continue au cours du projet car il nous a paru plus simple de partager nos différentes versions sur un Google Drive, mais nous avons ouvert un répertoire dédié au programme sur GitHub et l'avons testé occasionnellement afin de quand même nous familiariser avec cette manière de gérer nos versions. Maxime avait préparé un document expliquant au reste du groupe comment utiliser GitHub et une présentation de ces fonctionnalités basiques (document mis en annexe).

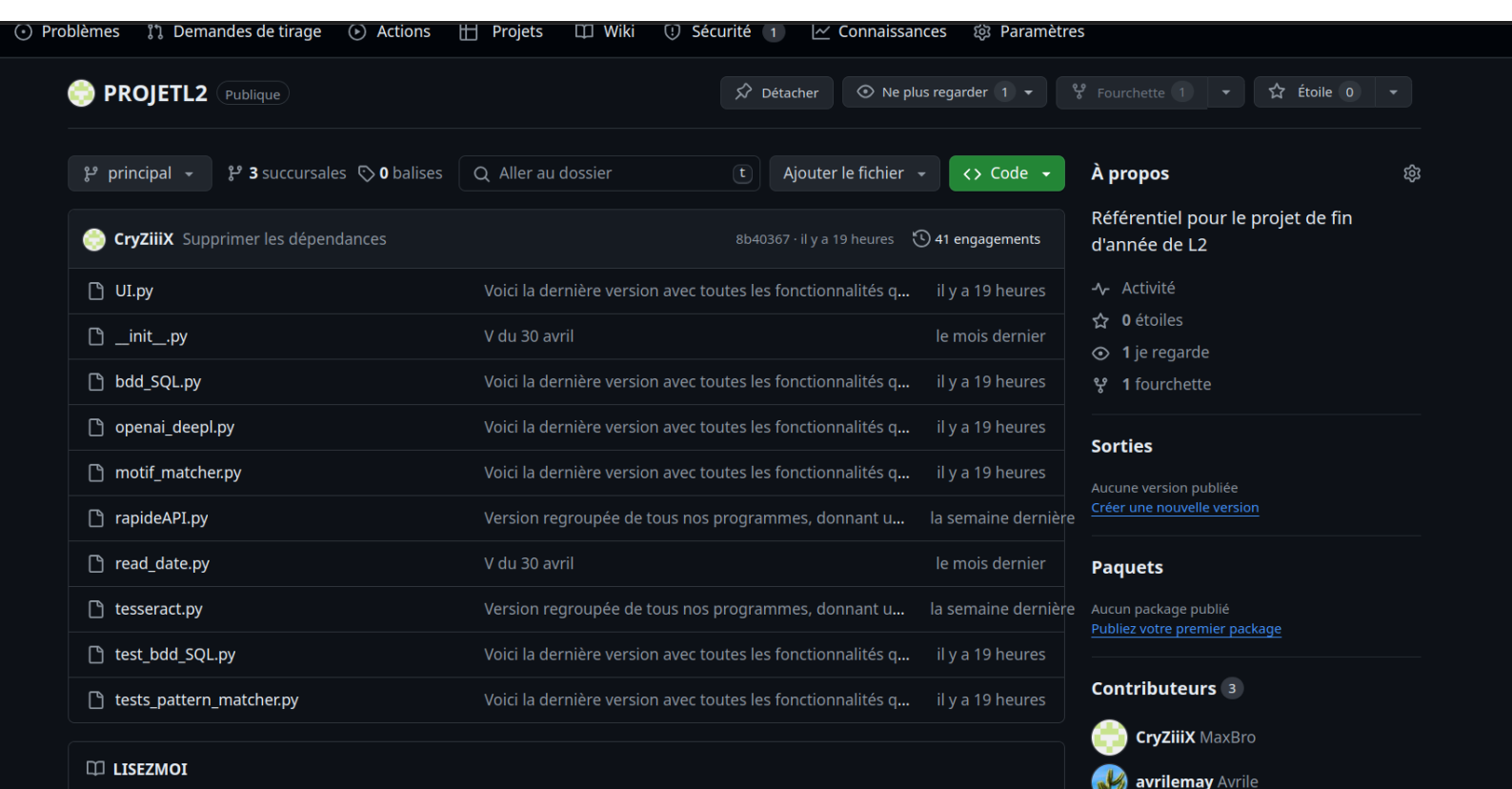


FIGURE 11 – Interface GitHub des fichiers du projet mis en ligne.

4.10 Packaging avec Hatchling

Nous avons décidé de créer un packaging afin de faciliter l'utilisation et la distribution de notre projet. Le packaging apporte plusieurs avantages, dont la simplicité et l'absence de difficultés d'installation pour les utilisateurs finaux. Le packaging assure une expérience utilisateur fluide, sans besoin de configuration complexe ou de gestion manuelle des dépendances.

Voici le contenu du fichier `pyproject.toml` utilisé pour créer le packaging :

```
1 [build-system]
2 requires = ["hatchling"]
3 build-backend = "hatchling.build"
4
5 [project]
6 name = "FactureFacile"
7 version = "0.1.1"
8 authors = [
9   { name = "Avrile Floro", studentID = "22000086" },
10  { name = "Aude Hennino", studentID = "19011803" },
11  { name = "Maxime Bronny", studentID = "19009314" }
12 ]
13
14 requires-python = ">=3.10"
15 dependencies = [
16   "mysql-connector-python>=8.4.0",
17   "Pillow>=10.3.0",
18   "opencv-python>=4.9.0.80",
19   "pytesseract>=0.3.10",
20   "deepl>=1.18.0",
21   "requests>=2.31.0",
22   "dateparser>=1.2.0",
23   "tkcalendar>=1.6.1",
24   "openai>=1.25.1",
25   "PyMuPDF>=1.24.2",
26 ]
27
28 [tool.hatch.build.targets.wheel]
29 packages = ["src"]
30
31 classifiers = [
32   "Programming Language :: Python :: 3",
33   "License :: OSI Approved :: MIT License",
34   "Operating System :: OS Independent",
35 ]
```

Outils utilisés

Pour réaliser le packaging, nous avons décidé d'utiliser Hatchling comme outil de construction backend. Nous avons suivi pas à pas des instructions trouvées en ligne (source : <https://packaging.python.org/en/latest/tutorials/packaging-projects/>). Les métadonnées du projet ont été définies de manière conventionnelle, en suivant les indications données par notre source. Nous avons également veillé à définir les dépendances en utilisant les versions les plus récentes disponibles (mis à part pour Python).

4.11 Procédure d'installation et de configuration

4.11.1 Contenu du dossier projet

Voici les étapes à suivre pour installer les dépendances nécessaires à l'exécution du programme :

1. Télécharger le fichier `.tar.gz` sur GitHub au lien suivant : <https://github.com/CryZiiiX/PROJECTL2>
2. Extraire le contenu de l'archive téléchargée. Un dossier **projet** qui se décompose comme suit sera obtenu :
 - Un script `install.sh` qui permet de procéder aux installations nécessaires à l'exécution du programme.
 - Un fichier `.wheel` qui permet de télécharger les dépendances python.
 - Un répertoire `src` qui contient le code source.
 - Un répertoire `src_avec_tests` qui contient le code source ainsi que les fichiers nécessaires pour l'exécution des tests.
 - Un répertoire `factures` proposant des factures à utiliser avec le programme.

4.11.2 Installations préalables à l'exécution du projet

Le script `install.sh` lance l'installation des composants logiciels nécessaires (`tesseract`, `pip`, `tkinter`) et va ensuite procéder à l'installation des dépendances via l'installation du paquet `.wheel`. Voici le contenu du script :

```
1 #!/bin/bash
2
3 # mettre à jour les dépôts et installer Python3-pip, Tesseract OCR et Tkinter
4 sudo apt-get update
5 sudo apt-get install -y python3-pip tesseract-ocr tesseract-ocr-fra python3-tk
6
7 # installer les dépendances Python
8 pip install facturefacile-0.1.2-py3-none-any.whl
9
10 echo "Installation terminée avec succès."
```

Afin de procéder aux installations nécessaires, via le script `install.sh`, il convient de se placer dans le répertoire où se trouve le script. La commande suivante permet d'attribuer au script des droits d'exécution :

```
1 chmod +x install.sh
```

Afin d'exécuter le script, il faut se placer dans le répertoire adéquat (le répertoire où se trouve le script) et lancer la commande :

```
1 ./install.sh
```

4.11.3 Lancer le programme

Pour lancer le projet, il faut exécuter le fichier `UI.py` qui se trouve dans le répertoire `src`.

5 Tests et validation

Durant le processus de développement de notre projet, nous avons accordé une importance particulière aux tests afin de nous assurer que les fonctionnalités principales de notre application étaient correctement implémentées et n'étaient pas altérées par les modifications successives. Étant donné que l'utilisateur doit interagir avec l'UI, seuls certains tests ont pu être automatisés. Les tests de l'UI nécessitaient une intervention humaine pour vérifier certaines fonctionnalités spécifiques, telles que dessiner des rectangles et sélectionner des boutons. Cette section décrit les différentes étapes de nos tests.

Les principaux tests menés lors du développement du projet ont été les suivants :

1. Tests de la partie préparation de l'image et lecture de l'image avec OpenCV et pyTesseract : tests menés avec sélection des zones de texte par l'utilisateur ainsi que sans sélection sur plusieurs types de documents.
2. Tests du nettoyage du texte (incluant l'API de OpenAI) sur plusieurs sorties pyTesseract au format texte
3. Tests de la traduction du texte (incluant l'API de DeepL) sur plusieurs sorties pyTesseract après nettoyage du texte
4. Tests des fonctions de pattern-matcher sur plusieurs sorties pyTesseract (détermination de la date, du montant, de la devise et du type de dépense)
5. Tests de la base de données
6. Tests de l'UI

5.1 Tests OpenCV et pyTesseract

Les tests réalisés sur le rendu du traitement des images par OpenCV et la lecture de ces images par pyTesseract ont été menés sur Jupyter Notebook. Cela a permis l'affichage des factures à chaque étape intermédiaire. L'échantillon des documents à lire comprenait un texte, des menus et des factures, le tout en français. Les tests ont notamment porté sur :

- La lecture d'images présentées sans sélection préalable de zones de texte pré-définies. Cela a fait ressortir la difficulté pour OpenCV, dans certains cas, d'isoler les zones de texte indépendantes les unes des autres par formation de rectangles, même en appliquant un processus de recherche de zones automatique, notamment par dilatation du texte.
- La lecture d'images avec sélection préalable manuelle des zones de texte par l'utilisateur. Ces tests donnaient de meilleurs résultats que ceux portant sur la détection automatique présentés ci-dessus.
- La lecture d'images légèrement penchées, qui a conduit à la mise en place d'une fonction de recherche de l'angle de rotation à appliquer ainsi que d'une fonction de redressement de l'image. En effet, pyTesseract ne peut lire une image que si les lignes sont organisées horizontalement et sans biais.

5.2 Tests de nettoyage du texte

Pour les tests de nettoyage de texte, nous avons commencé par extraire des textes via Tesseract et nous avons ensuite fait traiter ces extraits par Gemini. Gemini répondait parfaitement à nos critères initiaux. Les tests étaient concluants et la qualité du nettoyage était satisfaisante. Toutefois, Gemini ajoutait des ****** autour des parties considérées importantes, les mettant en gras. Pour corriger cela, nous avons ajouté une fonction pour supprimer ces ****** après le post-traitement. Finalement, nous avons abandonné l'utilisation de Gemini. Nous n'avons donc plus eu besoin de cette fonction supplémentaire de nettoyage.

Nous avons ensuite recherché une autre API pour le post-traitement et effectué des tests directement avec l'UI, alors partiellement fonctionnelle. Ces tests en conditions réelles ont révélé que Mistral de Hugging Face ne fournissait presque aucune correction satisfaisante. Nous avons donc opté pour l'API de OpenAI avec ChatGPT. Les résultats étaient bien meilleurs.

Pendant nos tests, nous avons constaté que lorsque les entrées envoyées à l'API de ChatGPT étaient vides ou contenaient seulement un ou deux caractères, l'API renvoyait un message indiquant que le texte à corriger était nul. Ce message était affiché dans notre UI, ce qui n'était pas convenable. Nous avons donc ajouté une étape de vérification avant l'appel à l'API. Si le texte extrait par Tesseract contient uniquement des espaces ou a une longueur inférieure à 20 caractères, alors le post-traitement renvoie le contenu initial, sans appeler l'API de OpenAI.

5.3 Tests de traduction du texte

Pour la traduction, nous avons testé plusieurs API, notamment DeepL et Google Translate API. Les tests ont été effectués à partir de documents réels, comme un article de journal sur l'enseigne Au Vieux Campeur et des menus de restaurant.

Nous avons constaté que DeepL était plus efficace pour les traductions complexes. Par exemple, DeepL identifiait correctement les noms propres comme "Au Vieux Campeur", tandis que Google Translate traduisait tout en anglais, y compris le nom de la boutique. C'est pourquoi nous avons choisi DeepL.

5.4 Tests du pattern matcher

Le fichier en charge des tests du pattern matcher est `tests_pattern_matcher.py`.

Pour faciliter la vérification du fonctionnement de notre programme et en particulier l'extraction de texte, nous avons placé dans un dossier à part les extractions textuelles des factures utilisées pour les tests. Ces extractions de texte ont été obtenues via l'UI. Une fois l'extraction textuelle obtenue, nous pouvions travailler sur le pattern matcher sans intervention humaine pour la sélection de texte. Nous avons automatisé les tests pour obtenir les données extraites par notre pattern matcher sur chaque facture et les comparer aux résultats attendus, facilitant ainsi le suivi des erreurs et l'évaluation des corrections effectuées.

Les principaux types de tests automatisés incluent :

- Extraction de la date, du montant, de la devise et de la catégorie à partir du texte extrait.
- Comparaison des valeurs extraites avec les valeurs correctes pour chaque facture.
- Suivi des erreurs et affichage des informations incorrectes pour faciliter le diagnostic.

```
Fichier: facture34_nettoye.txt, Date extraite: 01-04-2024, Montant extrait: 220.23, Devise extraite: EUR, Catégorie extraite: Travaux, Mot correspondant: TRAVAUX
Informations incorrectes:
Date extraite: 01-04-2024, date correcte: 06-03-2024

Fichier: facture8_nettoye.txt, Date extraite: 01-03-2023, Montant extrait: 1.0, Devise extraite: EUR, Catégorie extraite: Téléphonie, Mot correspondant: Internet
Informations incorrectes:
Montant extrait: 1.0, montant correct: 63.99

Fichier: facture28_nettoye.txt, Date extraite: None, Montant extrait: 56.94, Devise extraite: EUR, Catégorie extraite: Alimentaire, Mot correspondant: CARREFOUR
Informations incorrectes:
Date extraite: None, date correcte: 28-03-2024
Montant extrait: 56.94, montant correct: 50.94
Catégorie extraite: Alimentaire, catégorie correcte: Autre

Fichier: facture13_nettoye.txt, Date extraite: 28-02-2024, Montant extrait: 712.47, Devise extraite: EUR, Catégorie extraite: Autre, Mot correspondant: None
Informations incorrectes:
```

FIGURE 12 – Les tests permettent de mettre en valeur les informations extraites par le pattern matcher et d’identifier rapidement les erreurs commises lors de l’extraction.

5.5 Tests de la base de données

Le fichier en charge des tests de la base de données est `test_bdd_SQL.py`.

Concernant la base de données, des tests ont été créés dès l’étape de développement pour vérifier les différentes fonctionnalités. Nous avons utilisé `unittest` pour permettre l’exécution des tests de manière isolée, évitant ainsi des ralentissements dus aux multiples interactions avec la base de données. L’espace mémoire de cette dernière étant limité, nous avons décidé de le préserver autant que possible.

Lors des tests de l’UI, nous avons remarqué que la base de données ne gérât pas correctement les valeurs `null`, ce qui nous a conduit à apporter des modifications au code source et à développer des tests spécifiques pour cette situation.

Les principales fonctionnalités testées incluent :

- Connexion à la base de données et vérification de la connexion.
- Enregistrement et suppression des factures, avec vérification de leur gestion correcte.
- Mise à jour et vérification des informations de traduction.
- Calcul du nombre de factures traitées, du prix moyen des factures, et de la somme des montants par catégorie.

- Calcul du nombre total de caractères traduits et la fréquence d'utilisation des différentes langues cibles.

Cette structure de tests nous a permis de nous assurer que chaque modification apportée n'introduisait pas de nouvelles erreurs dans le système.

```
✓ Tests passed: 1 of 1 test - 4 sec 159 ms

/Users/avri1e/PycharmProjects/fetetravail_venv/bin/python3.10 ↵
↳/Applications/PyCharm.app/Contents/plugins/python/helpers/pycharm_
↳/_jb_unittest_runner.py --target test_bdd_SQL.TestStringMethods_
↳.test_afficher_date_facture
Testing started at 5:48 PM ...
Launching unittests with arguments python -m unittest test_bdd_SQL
.TestStringMethods.test_afficher_date_facture in
/Users/avri1e/PycharmProjects/V16

testing afficher_date_facture
Date de la facture: 2022-07-11

Ran 1 test in 4.160s

OK

Process finished with exit code 0
```

FIGURE 13 – Les tests permettent de vérifier la correcte exécution des différentes fonctionnalités de la base de données.

5.6 Tests de l’UI durant le processus de développement

Tout au long du développement, nous avons continuellement testé l’UI. En utilisant l’interface, nous avons pu identifier des éléments à corriger. En effectuant des tests, nous avons découvert certaines incohérences dans l’interface. Par exemple, pour rendre plus clair le parcours progressif au sein de l’interface au fur et à mesure de la sélection des options, nous avons décidé de griser certaines parties de l’UI.

Des tests ont été effectués pour vérifier les fonctionnalités de chaque fenêtre de l’application. Par exemple, sur la Tab 1 (Affichage image), des tests ont été réalisés pour l’importation des images, le dessin et l’effacement des rectangles, ainsi que la validation des données sans téléchargement. Sur la Tab 2 (Vérification), nous avons testé la validation des données de facturation, l’affichage et la validation du texte pour la traduction, ainsi que la traduction elle-même. La Tab 3 (Traduction) a été testée pour la sauvegarde du texte, tandis que la Tab 4 (Comptabilité) a été vérifiée pour les combinaisons de mois et d’année, la catégorisation des dépenses, la vérification des ID des factures et la consultation ou la suppression des factures par ID. Enfin, la Tab 5 (Statistiques) a été testée pour les choix de mois et d’année, ainsi que pour la logique de sélection des options.

Les tests suivants ont identifié des résultats anormaux avant correction :

- Importer une image au mauvais format
- Choisir un mois sans année
- Choisir une année sans mois
- Validation d’une traduction sans langue définie
- Ajout de rectangles de très petite taille sur le canvas

Ces anomalies ont été identifiées et corrigées pour assurer le bon fonctionnement de l’application.

Un extrait des tests effectués est disponible en annexe (voir [D](#)).

6 Axes d'amélioration

Le logiciel que nous rendons nous semble déjà relativement complet, et nous y avons intégré les fonctionnalités qui nous paraissaient importantes. Néanmoins, nous avons également mis de côté certains aspects, soit par manque de temps, soit car la mise en oeuvre ne nous paraissait pas aisée et aurait demandé une analyse plus approfondie. Les axes d'amélioration possibles que nous avons identifiés sont les suivants :

1. Lecture de documents au format pdf de plusieurs pages (actuellement seule la première page est lue)
2. Neutralisation du redressement automatique de l'image, au cas où celui-ci ne donne pas le résultat attendu
3. Proposition automatique des rectangles sur la facture pour que l'utilisateur n'ait plus qu'à confirmer
4. Détection automatique de l'émetteur de la facture (mais cela nous paraît difficile à automatiser)
5. Possibilité d'exporter les résultats des onglets 4 et 5 dans un tableur ou sur un fichier pdf
6. Possibilité de centraliser sur un serveur les fichiers des factures passées au logiciel, afin de les récupérer depuis n'importe quel ordinateur.
7. Proposition de versions du logiciel en anglais, allemand ou espagnol puisqu'une partie des utilisateurs attendus, notamment ceux utilisant le service de traduction, ne maîtrisera pas forcément le français.
8. Possibilité de sauvegarder des factures et leur traduction sur un serveur commun (plutôt que sur le disque dur de l'utilisateur) pour permettre l'utilisation du logiciel par plusieurs personnes à la fois, ou par une seule personne mais depuis différents terminaux.
9. Enfin, le pattern-matcher pourrait être largement amélioré en tenant compte des différentes devises que l'on peut rencontrer, il est probable que certains cas spécifiques n'aient pas été pris en compte.

Parmi ces propositions, celle qui paraît la plus difficile à mettre en oeuvre est la reconnaissance automatique de l'émetteur de la facture, car cet élément est rarement lié à un mot-clé spécifique, et il peut être situé à différents endroits de la facture (mais en général plutôt dans l'en-tête). Par ailleurs, si pour l'oeil humain la distinction entre le destinataire et l'émetteur est aisée, elle est beaucoup plus compliquée pour l'ordinateur.

7 Conclusion

Le projet rendu a évolué par rapport au cahier des charges initial, notamment afin de refléter une utilisation réaliste dans le cadre d'une problématique bien définie : tenue d'une comptabilité simple et structurée des dépenses sur la base de factures avec possibilité de traduction desdites factures à l'aide d'un logiciel installé sur l'ordinateur individuel de l'utilisateur. Cela nous a également permis d'introduire des éléments d'algorithmique, que nous n'avions pas intégrés dans la proposition initiale. En tenant compte du cahier des charges légèrement redéfini, nous avons atteint notre objectif avec un produit facile à installer, simple et intuitif d'utilisation.

Annexes

A Cahier des charges initial

Le cahier des charges initial est un document fourni en annexe

B Documentation Trello

La documentation Trello préparée pour le groupe se trouve en annexe

C Documentation GitHub

La documentation GitHub préparée pour le groupe se trouve en annexe

D Tests de l'UI réalisés pendant le développement

Voici un exemple des différents tests effectués durant la phase de développement pour vérifier les fonctionnalités de l'UI.

TESTS TAB 1 - Importer image

Importer une image au format jpg (normal)
Importer une image au format jpeg (normal)
Importer une image au format png (normal)
Importer une image au format pdf (normal après correction)
Importer une image au bon format penché (normal)
Importer une image au mauvais format (not normal mais désiré + à corriger)
> Transformer le print en « Message Box »
Dessine un rectangle en dehors de l'image (not normal mais désiré)
Dessine un rectangle (normal)
Efface un rectangle (normal)
Valide sans télécharger (not normal mais désiré)
Valide sans rectangle (normal)
Valide avec rectangle (normal)

TESTS TAB2 - Vérification

Valider données sans émetteur facture (not normal mais désiré)
Valider données sans date facture (not normal mais désiré)
Valider données sans montant (not normal mais désiré)
Valider données sans devise (not normal mais désiré)
Valider données avec toutes les données (normal)
Valider catégorie sans catégorie (not normal mais désiré)
Valider catégorie avec catégorie (normal)
Afficher le texte pour traduction (normal)
Valider le texte (normal)
Traduire en ... (normal)

TESTS TAB3 - Traduction

Sauvegarder le texte (normal)

TESTS TAB4 - Comptabilité

Choix mois tous et année (normal)
Choix mois avril et année (normal)
Catégorie de dépense (normal)
ID de la facture qui n'existe pas (normal)
ID de la facture qui existe (normal)

TESTS TAB5 - Statistiques

Choix mois sans année NOT NORMAL À CORRIGER

> Erreur: Une fois qu'il y a une erreur, ça ne se remet plus à jour.

Choix année sans mois NOT NORMAL À CORRIGER

> Erreur: Il faut activement sélectionner « choisir un mois », ce n'est pas logique

E Rapport personnel - Avrile Floro

Les rapports personnels se trouvent en annexe.

F Rapport personnel - Aude Hennino

Les rapports personnels se trouvent en annexe.

G Rapport personnel - Maxime Bronny

Les rapports personnels se trouvent en annexe.

H Vidéo d'installation du logiciel

Une vidéo détaillant l'installation du logiciel FactureFacile se trouve en annexe.

I Vidéo sur les tests

Une vidéo détaillant une partie des tests se trouve en annexe.

J Vidéo d'utilisation du logiciel

Une vidéo montrant l'utilisation du logiciel pas à pas du point de vue de l'utilisateur se trouve en annexe.