



RAPPORT DE PROJET

ITESOFT

**INDUSTRIALISATION DE L'OUTIL DE REVERSIBILITE
DE PROCESSUS METIER**

**REALISE PAR
MARC HAYE**

**MAITRE DE STAGE
ALEX BROUSSARD**

**TUTEUR DE STAGE
MARC JOANNIDES**

**POUR L'OBTENTION DU BUT INFORMATIQUE
REALISATION D'APPLICATION – CONCEPTION, DEPLOIEMENT, VALIDATION (RACDV)**

**PERIODE DU 27 JANVIER AU 4 AVRIL
ANNEE UNIVERSITAIRE 2024 - 2025**

Remerciements

Au sein d'ITESOFT, j'ai eu la chance d'avoir pu suivre un excellent encadrement par de nombreux professionnels bienveillants et instructifs, qui se sont continuellement assuré du bon déroulement de cette mission. Ce sont des collaborateurs avec lesquels j'ai pu découvrir et développer de considérables nouvelles compétences techniques, ainsi que construire de solides fondations pour ma future carrière professionnelle. Pour cela, je tiens à remercier tout particulièrement :

- ALEX BROUSSARD, MON MAITRE DE STAGE ET ARCHITECTE R&D
- CHRISTOPHE LE GRAC, MANAGER R&D

La qualité de cet encadrement a également été permise grâce à l'équipe pédagogique du département informatique de l'IUT MONTPELLIER-SETE :

- MARC JOANNIDES, ENSEIGNANT ET TUTEUR UNIVERSITAIRE DE STAGE
- REMI COLETTA, RESPONSABLE UNIVERSITAIRE DES FORMATIONS ET D'ALTERNANCES

En complémentaire, je tenais également à remercier pour leurs précieux conseils et la bienveillance qu'ils m'ont apporté :

- KHEIREDDINE HADDOUR, ENGINEER R&D
- LÉO CAGNARD, SOFTWARE ENGINEER R&D
- VIRGINIE TOIRON, ENGINEER R&D
- FRÉDÉRIC ALLIN, PRODUCT MANAGER DU BLOC BU
- YVES DIETERICH, MANAGER SAAS OPS
- BAPTISTE FREMAUX, QUALITY ENGINEER QA
- LOUIS FERRAND, SYSTEMS ENGINEER ASSISTANT AU DEPARTEMENT IT
- ESTEBAN REMOND, ETUDIANT A L'IUT ET STAGIAIRE CHEZ YOOZ

Résumé

Ce document est un rapport de stage effectué au sein de l'entreprise ITESOFT et rédigé par un étudiant de l'IUT de Montpellier dans le cadre de sa deuxième année d'études. Il présente le développement logiciel effectué durant la période de stage.

Ce rapport se base sur trois thématiques qui ont été abordées durant ce stage :

1. L'analyse de l'outil existant au sein de Saas Ops*.
2. La conception d'un nouvel outil de réversibilité
3. Le développement d'une API* en TypeScript avec NestJS
4. La validation et le déploiement au client technique.

Le travail issu de ce stage constitue le début de ce qui pourra se prolonger durant une alternance.

Table des matières

Remerciements	2
Résumé	3
Table des figures	6
Glossaire	7
Définitions	8
Introduction	9
1. Présentation de l'entreprise.....	10
1.1. Contexte	10
1.2. Organisation des équipes.....	11
1.3. Equipe R&D Supplier	11
1.4. Equipe QA Supplier.....	11
1.5. Streamline for Invoices / SLI	12
1.6. Balance Omnicanal.....	14
2. Analyse.....	15
2.1. Analyse de l'existant	15
2.1.1. Contexte	15
2.1.2. Enjeux	15
2.1.3. Base de données	16
2.2. Environnement de travail.....	17
2.2.1. Docker	17
2.2.2. Postman.....	18
2.2.3. N8N	19
3. Rapport technique	20
3.1. Cahier des charges.....	20
3.1.1. Contraintes de l'existant.....	20
3.1.2. Besoins fonctionnels	22
3.1.3. Besoins non fonctionnels	22
3.2. Conception	23
3.3. Architecture logicielle	26
3.4. Réalisation.....	27
3.4.1. Préambule : Déploiement local de SLI et d'une BD Mock.....	27
3.4.2. Première tâche : Initialisation NestJS et Connexion BD	31
3.4.3. Deuxième tâche : Export des factures	33
3.4.4. Troisième tâche : Export des pièces jointes	37
3.4.5. Quatrième tâche : Refactor et Configurabilité de l'API	40
3.4.6. Cinquième tâche : Export des historiques de facture	41

3.5. Validation.....	42
3.5.1. Tests en production.....	42
3.5.2. Conteneurisation de l'outil	43
4. Manuel d'utilisation	44
5. Méthodologie et organisation du projet	44
5.1. Méthodes agiles et points réguliers.....	44
5.2. Git flow	45
5.3. Autonomie et organisation personnelle	47
5.4. Tests en développement avec Postman	48
6. Conclusion	49
6.1. Bilan du travail réalisé	49
6.2. Perspectives.....	49
6.3. Bilan des acquis techniques	49
7. Visa du maître de stage	50
Bibliographie	51
Annexes	52

Table des figures

Figure 1 - Les principales solutions proposées par ITESOFT.....	10
Figure 2 - Portail d'accès administrateur de Streamline for Invoices.....	12
Figure 3 - Schéma commercial d'aide à la décision de SLI	13
Figure 4 - Logo de Docker	17
Figure 5 - Logo de Postman	18
Figure 6 - Logo de n8n	19
Figure 7 - Diagramme d'architecture initiale de Document Exporter API au sein de SLI.....	23
Figure 8 - Diagramme d'une future architecture de Document Exporter API au sein de SLI	24
Figure 9 - Exemple d'architecture d'un projet NestJS	26
Figure 10 - Configuration d'un agent de capture.....	28
Figure 11 - Affectation d'un agent de capture à un canal fichier.....	28
Figure 12 - Capture de factures.....	28
Figure 13 - Séquence de traitement d'une facture dans SLI.....	29
Figure 14 - Erreur de traitement d'Omnicanal Capture dans SLI	29
Figure 15 - Formulaire du workflow automatisant le dépôt de facture en BD	30
Figure 16 - Module App de l'outil de réversibilité	31
Figure 17 - Configuration de Swagger	32
Figure 18 - Première architecture de l'outil.....	32
Figure 19 - Méthode exportDocument appelé par la route API	33
Figure 20 - Décorateurs de la route du contrôleur	33
Figure 21 - Aperçu du service Invoice.....	34
Figure 22 - Première version du service Test Tools	34
Figure 23 - Aperçu du service Files	35
Figure 24 - Diagramme de séquence initial d'export de factures	36
Figure 25 - Signature initial de exportAttachmentExternalFile.....	37
Figure 26 - Décorateurs Swagger.....	38
Figure 27 - Exemple d'utilisation d'un objet DTO	38
Figure 28 - Requête à l'API de Document Manager	38
Figure 29 - Résultat d'arborescence d'un export après traitement par l'API	39
Figure 30 - Diagramme de séquence fonctionnelle de réversibilité global dans l'API	40
Figure 32 - Aperçu de la partie Docker de l'outil	43
Figure 33 - Aperçu du conteneur dans les services d'IntelliJ IDEA.....	43
Figure 34 - Emploi du temps et aperçu des réunions quotidiennes	44
Figure 35 - Gitflow du développement de solution au sein d' ITESOFT	45
Figure 36 - Aperçu git de Document Exporter depuis IntelliJ IDEA	45
Figure 37 - Demande de poussée pendant le développement de Document Exporter	46
Figure 38 - Organisation dans le navigateur web.....	47
Figure 39 - Prise de notes régulière avec Microsoft OneNote	47
Figure 40 - Utilisation de Postman pour tester l'outil.....	48
Figure 41 - Réalisation des diagrammes avec Mermaid live.....	48

Glossaire

Les expressions (*) apparaissent dans l'ordre alphabétique.

API : Une interface de programmation d'application est une interface logicielle qui permet de connecter un logiciel ou un service à un autre logiciel ou service afin d'échanger des données et des fonctionnalités.

Backend / Côté Serveur : Partie de l'application qui effectue les traitements sur les données (la partie immergée de l'iceberg qui communique avec la partie Frontend (voir Frontend) qui est la partie visible par l'utilisateur.

GED ou Gestion Electronique des documents : (En anglais EDM) Logiciel visant à organiser et gérer des informations sous forme de documents électroniques au sein d'une organisation. Dans le cadre d'ITESOFT, un GED, anciennement nommé DSS puis désormais Document Manager, est utilisé pour stocker les formats fichiers des documents et processus métiers traités par le logiciel Omnicanal Capture.

Framework : Aussi appelé infrastructure de développement, à ne pas confondre avec une librairie logicielle, est un ensemble cohérent de composants logiciels structurels servant à créer les fondations de l'architecture d'un logiciel.

Frontend / Côté Interface : Désigne les éléments d'un site que l'on voit à l'écran et avec lesquels on peut interagir depuis un navigateur. Il prend principalement en compte le design d'interface(UI) et de l'expérience utilisateur (UX).

Mock : C'est un "imitateur" qui remplace temporairement un vrai composant informatique. C'est comme un acteur doublure au cinéma qui joue le rôle d'un élément qui n'est pas encore prêt ou disponible. Par exemple, si une équipe développe une application qui doit se connecter à un service météo, mais que ce service n'est pas accessible pendant les tests, ils créeront un Mock qui renvoie des données météo fictives, permettant ainsi de tester l'application en conditions réelles, sans dépendre du vrai service.

REST : REST (REpresentational State Transfer) est un style d'architecture logicielle définissant un ensemble de contraintes à utiliser pour créer des services web. Avec l'utilisation d'un protocole sans état et d'opérations standards, les systèmes REST visent la réactivité, la fiabilité et l'extensibilité.

Saas : Un logiciel SaaS (Software as a Service) est un logiciel en tant que service. Cela signifie que vous pouvez accéder à un logiciel via internet, sans avoir à installer quoi que ce soit sur son ordinateur. L'accès se fait simplement sur la page web de la solution avec un identifiant et un mot de passe.

Saas Ops : L'équipe Saas Ops, aussi appelé officieusement équipe support, est l'équipe chargée de la maintenance système. C'est le client principal de la mission de stage.

Streamline for Invoices / SLI : Voir page 8.

Définitions

Invoices : Terme anglais pour factures.

Attachments : Terme anglais pour pièces jointes.

Invoices history : Terme anglais pour historique des factures.

Merge Request : Terme anglais pour demande de fusion.

Refactor : Anglicisme répandu en programmation, consiste à reprendre de zéro le code de tout ou partie d'un programme, et le développer à nouveau de sorte à remplir une tâche de la manière la plus efficiente possible.

R&D : Acronyme de Recherche et Développement.

Introduction

Dans l'ère du numérique, la comptabilité reste une problématique majeure de notre société, tant par sa complexité que par sa nécessité. La digitalisation de document est désormais omniprésente dans tous les domaines comptables, dans notre quotidien comme pour le développement de notre civilisation. Les documents papiers ne circulant plus, le traitement de processus métier est grandement accéléré et automatisé. Cependant, malgré le progrès technique et économique qu'a permis la « Big Data », la sécurité, la fiabilité et la vitesse de ces données représentent trois problématiques majeures. Un mauvais traitement de ces données pourrait entraîner des conséquences significatives dans le secteur comptable, pouvant impacter des dizaines de millions de documents numériques.

Pour faire face à ces enjeux, les entreprises conçoivent et déploient de volumineuses bases de données pour centraliser la gestion de ces volumétries. Malgré ces solutions, à long terme, ces données deviennent au fil des années de plus en plus conséquentes, coûteuses et difficiles à maintenir. C'est dans ce contexte que ITESOFT met en place depuis près de 40 ans des outils de digitalisation de documents pour automatiser efficacement les processus métier, et ainsi améliorer l'aide à la prise de décisions de grands groupes comptables. Toutefois, leur outil, consacré à la réversibilité, ne correspond à terme plus aux besoins de performances actuelle de ses clients. C'est pourquoi cette entreprise a fait appel à mes compétences pour réaliser un nouvel outil de réversibilité des processus métiers de traitement comptable.

Ce rapport de stage présente la réalisation de l'API REST* Document Exporter, que j'ai réalisé au sein de l'entreprise ITESOFT du 27 janvier au 4 avril 2025, dans le cadre de ma formation au département informatique de l'IUT de Montpellier-Sète. Après vous avoir présenté brièvement l'entreprise et ses produits, je vous détaillerai les objectifs qui m'ont été fixés durant ces dix semaines de stage, l'analyse, la conception, la réalisation, et la validation du produit. Pour chacun des points que j'aborderai, je présenterais les choix effectués et les résultats de ces décisions. Le travail issu de ce stage constitue le début de ce qui pourra se prolonger durant une alternance.

Il est à noter que la réalisation de ce stage a été faite en autonomie avec l'aide mineure des intelligences artificielles Claude.ai et DEX AI (cette dernière étant hébergée dans le réseau interne d'ITESOFT), pour accélérer le processus de recherche de documentation technique et de comparatif de performances entre différents langages de programmation dans la partie analytique. Il faut en conclure que ce rapport a été rédigé dans son intégralité par moi-même, sauf mention contraire, et sous l'encadrement régulier de mon maître de stage.

1. Présentation de l'entreprise

1.1. Contexte

ITESOFT est une entreprise éditrice de logiciels, leader français de digitalisation et d'automatisation des processus métiers. Ses solutions traitent plus d'un milliard de documents chaque année. Elle est créée en 1984 par Didier Charpentier qui est l'actuel PDG. Le siège social, où mon stage s'est déroulé, est basé à Aimargues, mais il y a également des équipes basées à Rueil-Malmaison en banlieue parisienne ainsi qu'à Woking en Angleterre, aux environs de Londres. L'entreprise propose des solutions SaaS* qui automatisent aussi bien les processus d'engagement client (instruction de dossier, contractualisation en ligne...) que les processus fournisseurs (Procure-to-Pay, factures fournisseurs...). L'activité principale de l'entreprise est la mise en production de ses solutions SaaS. Un même logiciel peut être hébergé dans plusieurs versions personnalisées pour chaque client, ils peuvent donc ajuster chaque demande spécifique sans craindre d'impacter l'expérience des autres clients.



Figure 1 - Les principales solutions proposées par ITESOFT

L'effectif est d'approximativement 150 salariés, mais les ressources disponibles sont en réalité plus importantes, car ITESOFT fait appel aux ressources de certaines ESN, anciennement SSII, société de service et d'ingénierie informatique, pour réaliser les développements importants. ITESOFT vise les clients grands comptes, comme France Télévision (groupe M6), Randstad, Sodexo, Gifi, ou encore la Caisse d'assurance maladie. Les clients à plus petits comptes correspondent plus à la clientèle visée par Yooz, ancienne société fille maintenant concurrente d'ITESOFT.

En 2023, ITESOFT a réalisé 20,4 millions d'euros de chiffre d'affaires dont 64,7 % sont issus de la vente de logiciels, 31,5 % des prestations de maintenance et enfin 3,8 % de la négociation de matériel informatique. Les principales parts de l'entreprise (73 %) sont détenues par le groupe SARL CDML, créé par le fondateur d'ITESOFT et également basé à Aimargues. La société détient de nombreuses distinctions dans son domaine, elle obtient notamment en 2022 la certification ISO 27001 [7] - Sécurité de l'information, cybersécurité et protection de la vie privée – plus spécifiquement les systèmes de management de la sécurité de l'information.

1.2. Organisation des équipes

Dans ITESOFT, chacun des logiciels est géré par un ensemble défini d'équipes qui s'occupent de son développement, de sa maintenance et de sa personnalisation. On nomme l'ensemble des équipes travaillant sur la solution SLI* le bloc BU. Dans le cadre de mon stage, j'ai été affecté au bloc BU Supplier, celui chargé de s'occuper de la solution Streamline Invoices*. Ce bloc suit une forme hybride des méthodes agiles et Scrum vues à l'IUT. Il est conçu pour suivre une hiérarchie horizontale entre chaque équipe, qui a chacune un manager associé. Le Product manager, qui est équivalent au Product owner en agilité, est chargé de s'assurer de la compréhension globale entre toutes les équipes.

1.3. Equipe R&D Supplier

Ils sont chargés d'implémenter de nouvelles fonctionnalités et de corriger des problèmes techniques. Durant ce stage, mon intégration s'est faite au sein de cette équipe. Leur client est concrètement l'équipe Delivery, pour qui ils réalisent chaque mois une nouvelle version de SLI avec de nouvelles fonctionnalités (Feature) et des corrections de fiches de bugs techniques (BT).

1.4. Equipe QA Supplier

Cette équipe représente les clients pour R&D et fait cette intermédiaire avec notre équipe, qui est plus orientée sur le développement de fonctionnalités. Il s'assure que nos nouvelles implémentations, même si elles sont prêtes, correspondent à ce que souhaite chaque client et en fonction de leurs demandes particulières de personnalisation. Cette équipe réalise et automatise en parallèle régulièrement différents tests techniques afin de s'assurer dans un premier temps que les nouvelles fonctionnalités réalisent leur comportement adéquat, et que ces dernières ne causent pas de nouveaux problèmes à l'existant. Ces tests sont principalement organisés en plusieurs catégories : tests de charge, de régression et de conformité.

1.5. Streamline for Invoices / SLI

Streamline for Invoices, souvent agrégé SLFI ou SLI au sein de l'entreprise et sera souvent désigné ainsi dans ce rapport, est un logiciel de dématérialisation de factures, et d'automatisation de processus métier. Ma mission s'est principalement déroulée au sein de l'équipe R&D, dédié au développement de nouvelles fonctionnalités et corrections de bugs de cette solution.

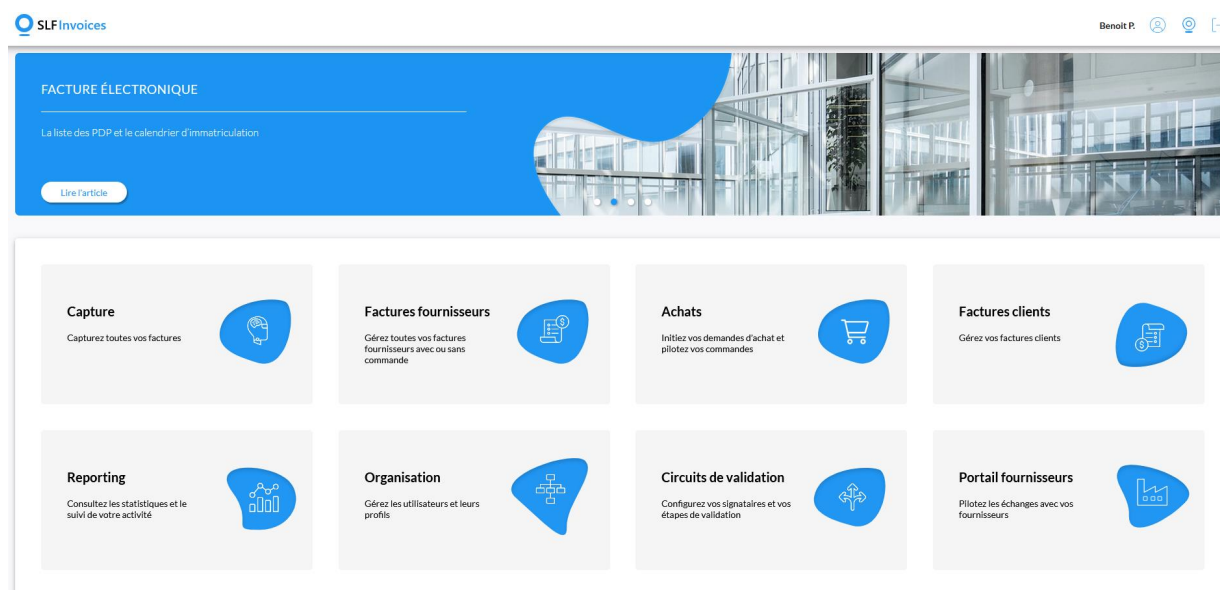


Figure 2 - Portail d'accès administrateur de Streamline for Invoices

La plupart des informations ci-dessous proviennent de sources fournies par ITESOFT, étant donc pour la plupart décrites sur un ton publicitaire et commercial. Il a été effectué au mieux une description analytique. Streamline for Invoices offre une couverture fonctionnelle importante : une acquisition omnicanale des factures (papier, courriel, PDF, Factur-X, UBL, CII), des Workflows comptables, une gestion des risques, de la conformité des documents, ainsi qu'une gestion des achats et approvisionnements. Ce logiciel permet, selon la brochure commerciale fournie par l'entreprise, « d'éviter le processus de Procure-to-Pay, considéré comme onéreux, complexe, lent et ouvert aux risques d'erreurs et fraudes ».

Concrètement, la facture ou autre document comptable est dématérialisée à l'aide de la brique logicielle Capture, produite par ITESOFT, puis est envoyée à une autre entreprise Yooz qui utilise son produit de reconnaissance optique des caractères pour extraire les informations pertinentes. Une fois ces deux étapes réalisées, Balance (une description sera faite plus bas.) reçoit les informations du document comptable et effectue les derniers traitements sur celui-ci de manière automatique si les données le permettent ou manuellement (imputation comptable ou analytique) si une intervention humaine est requise. Une fois les données complètes, le document comptable part dans un PGI (progiciel de gestion intégré) qui va finaliser le traitement du document, comme un paiement à un fournisseur.

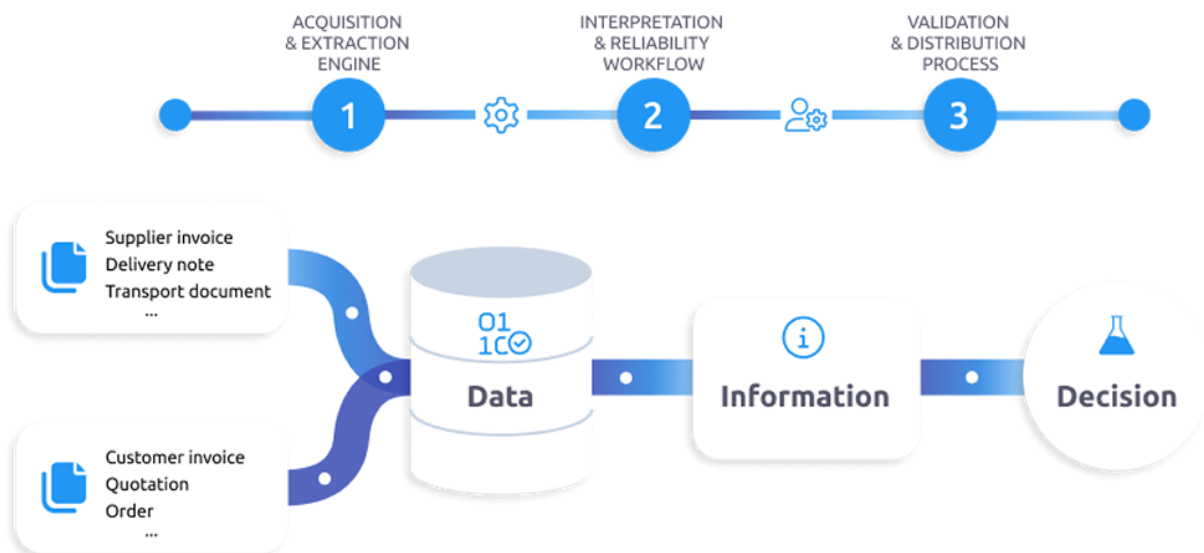


Figure 3 - Schéma commercial d'aide à la décision de SLI

Ce schéma figure dans la brochure commerciale fournie par ITESOFT. Le document papier ne circule plus dans l'entreprise, le traitement des factures est accéléré en automatisant au mieux les processus liés au métier comptable. De ce fait, les équipes sont focalisées sur des tâches à valeur ajoutée où leur savoir-faire est requis. Il y a une amélioration de la traçabilité et du suivi ainsi que de la consultation des factures, car les documents sont centralisés. La traçabilité, le suivi ainsi que la consultation sont grandement améliorés du fait de la centralisation des documents.

La partie technique de cette solution est devenue si conséquente qu'elle est divisée en différents sous modules, concrètement en plusieurs dépôts git, avec leur fonctionnement et leur architecture uniformisé. Par exemple, il a un module dédié à la partie frontend administrateur, et un module dédié au traitement de capture de documents. Ces modules sont conteneurisés avec l'aide de Docker, et sont ensuite déployés ensemble pour permettre le fonctionnement de SLI.

1.6. Balance Omnicanal

Balance est l'ancienne version de Streamline for Invoices. Cette version constitue tous les comportements du processus métier, et donc l'ensemble de la pile technique nécessaire, dans un seul module, qu'on appelait un monolithe. Cela posait de nombreux problèmes de maintenabilité et de difficulté à assimiler clairement son fonctionnement.

Depuis, SLI dans son ensemble a été divisée en de nombreux sous-modules techniques (avec chacun leur propre version) qui facilite la répartition de l'équipe et sa stabilité, et on emploie désormais le terme Balance pour désigner la partie technique de SLI qui permet de dématérialiser le traitement comptable. L'outil sur lequel porte cette mission est techniquement considéré comme un nouveau module de SLI.

L'application Balance permet de dématérialiser le traitement comptable, avec tous les avantages que cela peut avoir, notamment :

- Le document papier ne circule plus dans l'entreprise
- Le traitement des factures est accéléré en automatisant au mieux les processus liés au métier comptable. De ce fait, les équipes sont focalisées sur des tâches à valeur ajoutée où leur savoir-faire est requis.
- Amélioration de la traçabilité et du suivi ainsi que de la consultation des factures, car les documents sont centralisés.
- La traçabilité, le suivi ainsi que la consultation sont grandement améliorés du fait de la centralisation des documents.

Concrètement, la facture ou autre document comptable est dématérialisée à l'aide de la brique logicielle Capture (produite par ITESOFT), puis est envoyé à Yooz qui utilise son produit de reconnaissance optique des caractères pour extraire les informations pertinentes. Une fois ces deux étapes réalisées, Balance reçoit les informations du document comptable et effectue les derniers traitements sur celui-ci de manière automatique si les données le permettent ou manuellement (imputation comptable ou analytique) si une intervention humaine est requise. Une fois les données complètes, le document comptable par dans un PGI (progiciel de gestion intégré) qui va finaliser le traitement du document (ex : paiement à un fournisseur).

2. Analyse

2.1. Analyse de l'existant

2.1.1. Contexte

Actuellement, les factures des clients de la solution SLI sont traitées et stockées en interne dans les bases de données d'ITESOFT. Cependant, en cas de résiliation d'un de ces clients, l'entreprise doit lui restituer l'intégralité des données des documents traitées par SLI, par exemple pour qu'il puisse les charger dans une nouvelle application. Un client peut, dans la plupart des cas, avoir plusieurs millions de factures stockées en son nom. Cela nécessite un procédé considérable de traitement en continu.

Cette opération est faite par un premier outil de réversibilité, réalisé par l'équipe Saas Ops et utilisé depuis maintenant trois ans. Il est constitué d'un unique fichier PowerShell, exécuté manuellement, et effectue ce procédé technique : Il fait une requête de toutes les factures stockées dans la base de données sous format JSON, compresse ces résultats en supprimant les valeurs jugées inutiles pour le client (comme des variables sans valeur), génère trois fichiers CSV avec toutes ces informations, effectue une nouvelle requête pour récupérer les formats PDF du GED*, ainsi que les chemins d'accès de chaque facture. A la fin de ce procédé, il génère une arborescence de répertoires pour que chaque facture ait le chemin suivant (PDF et JSON) : Code société/Année/Mois/Jour/facture.json. Un membre de l'équipe Saas Ops* envoie ensuite au client par mail un lien de redirection pour accéder aux fichiers CSV et aux PDF de leurs factures.

2.1.2. Enjeux

Cette solution existante pose de nombreux problématiques d'industrialisation et de perspective d'évolution à moyen et long terme : Le choix du langage PowerShell pour la réalisation de l'outil a été fait par affinité, et n'est pas habituellement utilisé dans les procédés côté serveur* de mon équipe. Ce traitement est effectué par un seul fichier, ce qui est également difficile à maintenir s'il venait à évoluer. Ce script est exécuté manuellement dans un terminal et nécessite systématiquement une personne pour le paramétrage ainsi que le lancement de l'exécution. Cette personne doit s'assurer durant tout le procédé, pouvant durer plusieurs jours, que cela a pu se terminer sans erreur. Il n'y a d'ailleurs pas de prise en charge suffisamment industrielle des erreurs et exceptions. Dans le cas où le procédé se terminerait avec une erreur, il est relancé manuellement sans connaître précisément ce qui a causé cette exception. Cet outil n'est pas versionné, dans un dépôt distant git par exemple, et n'est donc maintenu que par un seul collaborateur. Si un problème technique venait à arriver à l'ordinateur stockant ce fichier, l'entièreté de l'outil disparaîtrait sans aucune sauvegarde existante. En parallèle de cela, il n'y a aucune documentation ni interface existante. Si une autre personne venait à devoir travailler sur cet outil (comme un stagiaire, par exemple), il devient difficile de comprendre clairement

son fonctionnement et de se l'approprier. De plus, l'outil doit communiquer avec les serveurs de l'outil SLI, notamment pour accéder aux fichiers PDF des factures. Cependant l'équipe SaaS Ops, dû à l'organisation technique de l'entreprise, n'y a pas directement accès. Ces derniers doivent contourner le réseau mis en place pour permettre son fonctionnement, ce qui accentue la dette technique de son fonctionnement. Cela peut causer à long terme des incidents techniques imprévisibles, ainsi que des failles de sécurité. Cet outil n'est donc pas vraiment ouvert aux nouvelles fonctionnalités mais plus à la modification.

Pour résumer, cet outil de réversibilité nécessite clairement une industrialisation complète, qui puisse rester fonctionnelle avec l'environnement technique existant d'ITESOFT. Ce qui a principalement fait réagir sur cette nécessité a été le besoin d'y implémenter la prise en charge des pièces jointes, qui fut sur l'outil existant difficile à développer et peu efficace par rapport au besoin réel.

2.1.3. Base de données

La base de données de SLI est principalement utilisée pour stocker toutes les informations nécessaires sur les différentes factures de chaque client. Le type de base de données existante utilisée est PostgreSQL, et le format de donnée des factures est en jsonb. C'est donc une base de données plutôt proche du non relationnel. Le format JSON est conçu pour stocker des données de type JSON (JavaScript Object Notation). Plutôt qu'avoir une ligne pour chaque facture avec un tuple pour chaque valeur associée, l'ensemble des informations est contenu dans un grand tableau "content". Cela apporte un gain en temps et en flexibilité. Les bases de données JSON, et plus généralement les bases en NoSQL se sont progressivement popularisées pour remplacer le format relationnel. Des exemples populaires sont MongoDB, utilisé par Google et Facebook, et CosmosDB, développé par Azure.

Le JSONB [1] est un format de donnée un peu plus particulier proposé par PostgreSQL. JSON et JSONB sont deux formats quasiment identiques, mais à la différence de JSON, le format jsonb ne préserve pas l'ordre exact des clés des objets et ne les duplique pas. Cela permet une grande évolutivité et flexibilité de la structure de donnée, ce qui le rend très simple à maintenir, car pas de relation clé primaire / étrangère. Si un client fait une demande spécifique sur la structure d'une facture sous forme de variable supplémentaire, on peut directement ajouter cette information sous forme de nouvel attribut dans la colonne "content". On peut accéder à ce dernier au travers d'une requête SQL avec une syntaxe spécifique au format JSON. Concernant son déploiement au sein de l'entreprise, une instance de la base de données est hébergée localement pour chaque client, car ITESOFT s'assure de répondre aux besoins de personnalisation de tous ses clients. Dans le cadre de mon stage, je serais amené à utiliser la base de données PostgreSQL existante de la solution SLI pour récupérer les données des factures, ainsi qu'accéder au DSS* existant de l'entreprise. J'ai été donc amené à comprendre comment la partie backend* de la solution SLI traite la facture dans la base de données afin de réaliser ma mission.

2.2. Environnement de travail

De nombreux outils sont utilisés en entreprise pour améliorer la productivité et la qualité du travail au sein de ce stage. Ceux-ci peuvent différer d'une entreprise à une autre et leur maîtrise, indispensable, constitue une véritable compétence qui doit être acquise rapidement pour pouvoir travailler efficacement et en respectant les formalités. Une plateforme wiki [9] interne aux serveurs est disponible pour accéder à la documentation technique des différentes réalisations des équipes du bloc BU. Vous trouverez ci-dessous une présentation des différents outils que j'ai dû découvrir et utiliser au sein de la réalisation de l'outil de réversibilité, et que j'ai utilisé tout au long de ce stage :

2.2.1. Docker



Figure 4 - Logo de Docker

Docker est un outil qui peut empaqueter une application et ses dépendances dans un conteneur isolé, qui pourra être exécuté sur n'importe quel serveur, peu importe sa configuration ou son environnement technique. Il ne s'agit pas de virtualisation, mais de conteneurisation, une forme plus légère qui s'appuie sur certaines parties de la machine hôte pour son fonctionnement. Cette approche permet d'accroître la flexibilité et la portabilité d'exécution d'une application, laquelle va pouvoir tourner de façon fiable et prévisible sur une grande variété de machines d'hôtes.

Les différents modules de la solution SLI sont décomposés en conteneurs pour permettre son déploiement. Il est donc très utilisé par l'équipe R&D pour son déploiement en local, et aide au développement de cette solution. J'ai également été amené à déployer cet outil avec Docker afin de développer l'outil d'extraction, ainsi que lors de son déploiement en production au sein des serveurs de l'entreprise, à l'équipe QA.

2.2.2. Postman



Figure 5 - Logo de Postman

Postman est une plateforme collaborative complète pour le développement, les tests et la documentation d'API. Elle permet de tester en temps réel des API, notamment durant leur phase de développement. Initialement lancé en 2012 comme une simple extension de navigateur, Postman a évolué vers une suite d'outils robuste qui accompagne l'intégralité du cycle de vie des API. Aujourd'hui, Postman est utilisé par plus de 25 millions de développeurs et 500 000 organisations à travers le monde. L'interface de Postman facilite la création et l'envoi de requêtes HTTP sans nécessiter de code, ce qui le rend accessible aussi bien aux développeurs qu'aux testeurs et aux Product managers. Cette dernière prend en charge de multiples protocoles comme REST, SOAP, GraphQL, et Web Sockets, offrant donc une polyvalence technique importante.

L'un des atouts majeurs de Postman réside dans son système d'environnements variables qui permet de passer facilement d'un contexte à l'autre (développement, test, production) en modifiant automatiquement les paramètres des requêtes. Cette fonctionnalité s'avère cruciale pour maintenir la cohérence lors des différentes phases de développement et de déploiement. Postman excelle également dans l'automatisation des tests grâce à son langage de script basé sur JavaScript. Il est possible d'implémenter des logiques pré-requête et post-requête, de définir des assertions complexes et de valider des schémas de données. Le Collection Runner permet d'exécuter des séries de requêtes de manière séquentielle ou parallèle, tandis que Newman, son équivalent en ligne de commande, facilite l'intégration dans les pipelines CI/CD.

La dimension collaborative de Postman s'exprime à travers ses espaces de travail partagés, son système de contrôle de version et ses fonctionnalités de feedback. Les équipes peuvent travailler simultanément sur les mêmes collections, suivre les modifications et maintenir une documentation à jour. La plateforme propose également des Mock* servers qui simulent des API, permettant ainsi aux équipes frontend de travailler parallèlement au développement backend. Pour les organisations soucieuses de gouvernance, Postman offre une gestion fine des accès et des permissions, un historique détaillé des actions et des mécanismes de sécurisation des données sensibles. Les API peuvent être organisées en catégories et liées à des projets spécifiques, facilitant ainsi leur découverte et leur réutilisation au sein de l'entreprise.

2.2.3. N8N



Figure 6 - Logo de n8n

N8N [6] est une plateforme d'automatisation de workflow open-source. Il permet principalement la modélisation de processus métier, l'intégration et l'interaction d'applications multiple. Grâce à des nodes ou nœuds informatiques, N8N peut créer des workflows qui mêlent plusieurs sites, services cloud, bases de données, outils ou applications. Chaque nœud permet d'effectuer une action comme lire un fichier, écrire un mail, envoyer une notification, etc., à exécuter seul ou en simultané des autres nœuds. Une autre fonctionnalité de N8N est sa capacité à créer des nœuds même avec des applications qui ne sont pas comprises dans son catalogue. En effet, malgré les plus de 350 applications disponibles, il existe des milliers d'autres outils que l'on peut utiliser pour diverses raisons et est très axé communautaire. Dans le cadre de cette mission, afin d'effectuer des tests de performances de mon outil, n8n fut utiliser pour automatiser la création de grands volumes de documents et données factures, ce traitement se fait habituellement par un client manuellement, et ainsi simuler rapidement une base de données client. Plus de détails sont apportés sur ce point dans le rapport technique.

3. Rapport technique

3.1. Cahier des charges

3.1.1. Contraintes de l'existant

Le client de cet outil est l'équipe support SaaS Ops. Une réunion fut organisée avec leur manager, qui a réalisé l'outil existant, afin de pouvoir analyser son fonctionnement et ses enjeux. Il a également pu fournir un résumé de fonctionnalités et de contraintes de l'existant. Ce texte n'étant pas rédigé par moi, il peut contenir des fautes d'orthographe.

Script PowerShell, qui effectue ce procédé:

- Effectue une requête de toutes les factures en format JSON
 - Compresse le résultat en supprimant les valeurs jugées inutiles
 - Génère un csv avec toutes ces informations
 - Effectue une nouvelle requête pour récupérer les file path de chaque facture
- Ce CSV est ensuite formaté en PDF (à confirmer) et envoyé manuellement par mail.

Objectif initial

- En cas de résiliation d'un client SLI : lui mettre à disposition l'intégralité des informations de la solution SLI

Eléments mis à disposition

- Un fichier csv <tenant>_all_invoices.csv qui contient la liste des factures avec les colonnes suivantes
 - Id : identifiant unique
 - Documenttype : type de document
 - Company_code : code de la société
 - Supplier_code : code du fournisseur
 - Supplier_name : nom du fournisseur
 - Documentdate : date du document
 - Scandate : date d'acquisition
 - Totalamount : montant total
 - Currency: devise
 - Filepath : chemin relatif d'accès au PDF de la facture
 - Un répertoire avec l'arborescence suivante :
 - Company_code
 - Année (4) de la date d'acquisition
 - Mois (2) de la date d'acquisition
 - Jour (2) de la date d'acquisition
- <guid>.pdf : le document pdf de la facture
 <guid>.invoice.json : les données simplifiées de la facture

<guid>.history.json : les données simplifiées d'historique de la facture

Etapas d'extraction

- Génération d'un fichier csv depuis un select de la base
 - Pour chaque facture
 - Récup du content json depuis la table invoice. Normalisation des données (optim taille)
- => écriture du fichier invoice
- Récup de l'historique. Normalisation des données => écriture du fichier history
 - Récup des images pdf via lien DDP (du coup image pas original) => écriture du pdf
 - Génération du fichier complet csv avec maj du champs Filepath

Contraintes/Evolutions

- Format csv : délimiter ';', pas de quote
 - Possibilité de choisir les factures => partie custom de la requête
 - Possibilité de relancer l'extraction sans "refaire" l'existant (incrémental)
 - Possibilité de lancer en mode check (compteur du nombre de factures ok/ko)
 - Config répertoire de sortie
 - Customisation de l'arborescence
 - Compatible avec toutes les versions en prod ?
-
- Extraire les fichier "originaux" ?
 - Extraire les pièces jointes
 - Extraire les DA ?

En parallèle, il a également fourni un exemple de structuration de données d'une facture, qui sont formalisé sous le format JSON :

```
{
  "header": {
    "batchName": "20210304_00008",
    "code": "5799274",
    "company_code": "ACME",
    "currency": "GBP",
    "description": "null - INV",  "docNumber": null,
    "documentDate": null,
    "documentType": "INV",
    "dueDate": null,
    "duplicated": false,
    "imagepath": null,
    "netAmount": 0,
    "receptionDate": "2021-03-04T17:50:10Z",
    "scanDate": "2021-03-04T17:50:10Z",
  }
}
```

À partir de ce paragraphe, j'ai pris la responsabilité de réaliser un cahier des charges exploitant les contraintes imposées sur la réalisation de cet outil, ce qui n'était pas imposé par l'entreprise.

3.1.2. Besoins fonctionnels

- Le client reçoit une réponse de la requête effectuée à l'API.
- Il doit pouvoir modifier la quantité de factures exportées, et s'il souhaite également exporter ses pièces jointes.

3.1.3. Besoins non fonctionnels

Performances:

- Le traitement des données récupérés doit être plus rapide et efficace que l'existant.

Accessibilité:

- Les données des documents exportés doivent pouvoir être suffisamment lisibles par l'humain, par exemple dans un fichier CSV.

Sécurité :

- Pouvoir exporter des données autres que des factures (pièces jointes, demandes d'achats) .
- Doit être sécurisé contre différentes attaques à la BD, ex: Injection SQL.

Évolutivité:

- Gestion de version avec une forge (Gitlab).
- Nécessité d'implémenter les pièces jointes et demandes d'achats .
- Il doit être 'facile' d'ajouter de nouvelles fonctionnalités avancées.

Maintenabilité:

- Rédaction d'une documentation (Avec par ex: JSDOC ou tj/dox sur GitHub).
- Commentaires réguliers dans le code de l'outil.
- Intégralité du code et de la documentation technique en anglais.

Déploiement :

- Doit être iso-fonctionnel avec les services existant en production de l'entreprise.
- Des tests de validation seront effectués sur l'outil avant d'être déployés.

3.2. Conception

La première étape de l'industrialisation de cet outil est la réécriture complète de son fonctionnement dans un nouveau langage, plus adapté au contexte de l'entreprise, plus sécurisé, plus facilement maintenable, et plus enclin à évoluer. Cet outil devra dans un premier temps répliquer le fonctionnement exact de celui de l'équipe SaaS Ops. La conception fut réalisée en autonomie, sans contraintes imposées par l'entreprise, et sous l'encadrement de mon maître de stage.

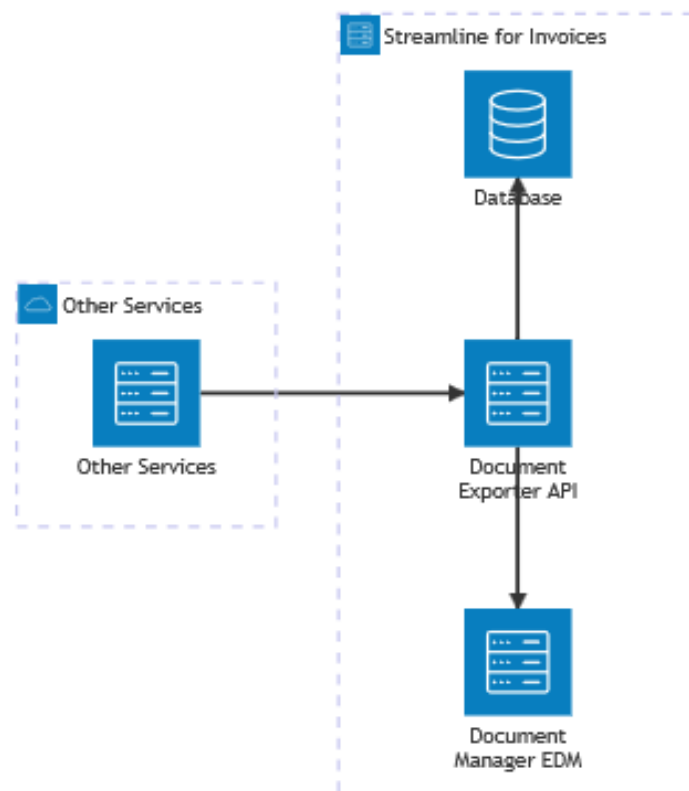


Figure 7 - Diagramme d'architecture initiale de Document Exporter API au sein de SLI

Le choix de la solution à adopter sera la réalisation d'une API REST, nommé Document Exporter par l'équipe R&D et moi-même. Elle sera l'intermédiaire entre la base de données, le gestionnaire de documents Document Manager et les personnes qui déclencheront une réversibilité (via des requêtes HTTP puis possiblement à l'avenir via l'interface web). Pour plus de clarté, il est important de ne noter que dans ce rapport technique, plusieurs dénominations : « l'outil de réversibilité », « Document Exporter », « Document Exporter API », « l'API », désigneront tous un seul et même produit réalisé durant cette mission. Cet outil devra pouvoir, à partir d'un Endpoint, lancer un procédé d'export de documents depuis la base de données, gestionnaire de documents électroniques GED* Document Manager, et recréer localement l'intégralité des données récupérées sous un format lisible par le client. La raison de ce choix est principalement de suivre l'uniformité de la gestion des micro-services de SLI, par exemple Document Manager est également constitué d'une API REST.

Le choix de la technologie (langage/framework*) n'a pas été prédéfini ni imposé par l'entreprise, et a donc été fait par mon maître de stage et moi-même, après plusieurs analyses comparatives et débats à ce sujet. Il reste néanmoins important de s'assurer que la solution technique choisie correspond aux enjeux clé de cette mission. Les deux traitements les plus coûteux en temps sont : les requêtes SQL, pour récupérer les métadonnées des factures, et les requêtes http faites au GED Document Manager, pour récupérer les fichiers des documents (PDF, xml, eml). Concernant l'optimisation de performances de la partie SQL, l'utilisation nécessaire d'un 'connection pool' fut proposé, afin d'optimiser le coût des connexions à la base de données. Ce choix n'a finalement pas été jugé nécessaire. Pour l'instant dans notre cas, ces derniers seront les clients qui résilient leur abonnement SLI, mais l'outil pourrait à l'avenir être amené à être exploité par l'entreprise dans d'autres situations moins spécifiques.

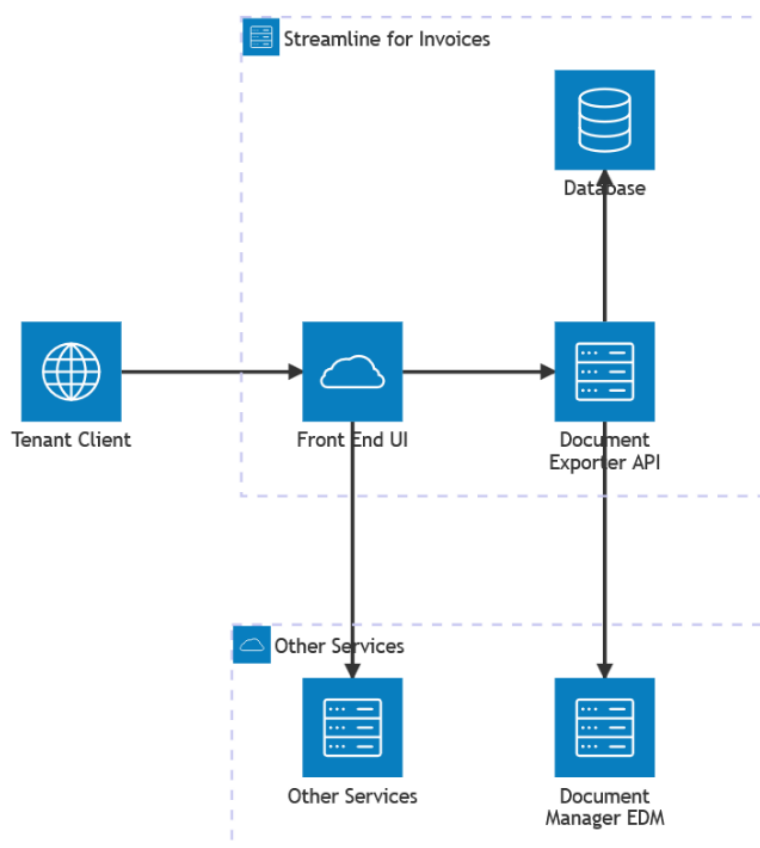


Figure 8 - Diagramme d'une future architecture de Document Exporter API au sein de SLI

Cela n'a pas pu être abordé suffisamment par rapport aux contraintes de temps pour réaliser la mission, mais en vue de l'évolutivité de l'outil, ce point reste une future possibilité pour ITESOFT de continuer vers cette possible utilisation de l'API. Par exemple dans le cas où un client actuel souhaiterait à tout moment récupérer l'intégralité de ses factures rapidement via l'interface utilisateur de Streamline for Invoices*, et choisir les factures spécifiques qu'elle souhaiterait exporter. D'autres points techniques supplémentaires ont été pensés pour répondre à toutes les problématiques d'industrialisation. L'outil sera mis sous version dans un dépôt distant sur le serveur Gitlab de l'entreprise, et une documentation technique sera générée automatiquement avec l'outil open API de Swagger [4].

Avant d'entamer ce choix de technologie, un travail comparatif de performances a été effectué pour déterminer la solution technique pour répondre aux enjeux techniques. Ces recherches ont été effectuées avec l'aide de l'intelligence artificielle Claude AI, et une réponse est apportée à la fin de chaque comparaison pour apporter un esprit critique sur ses affirmations.

Les choix retenus pour cette partie sont les suivants :

Go est généralement considéré comme l'un des meilleurs choix pour les performances http :

- Excellente gestion de la concurrence grâce aux goroutines
- Faible utilisation mémoire
- Temps de démarrage très rapide
- Compilation native

Node.js est aussi très performant pour les requêtes HTTP :

- Architecture non-bloquante et événementielle efficace
- Excellent pour gérer beaucoup de connexions simultanées
- Gère très bien la concurrence sans la complexité de la gestion des threads
- Les performances sont largement suffisantes pour la plupart des cas d'usage
- Bon pour des applications temps réel
- Grande communauté et écosystème riche

Rust offre également d'excellentes performances :

- Contrôle précis de la mémoire
- Sécurité des threads
- Performances proches du C++
- Idéal pour des systèmes critiques

Python est un peu moins performant :

- Plus lent que Go/Node.js pour des charges importantes
- Le GIL peut limiter les performances multithreads
- Mais des frameworks comme FastAPI ou aiohttp améliorent significativement les performances http

Entre Python et Node.js, spécifiquement, Node.js possède effectivement de meilleures performances pour les requêtes http. Non seulement, Node.js a été spécifiquement conçu pour l'asynchronisme, et résister à de fortes charges. Cela sera le cas pour les exports de documents intenses que devra effectuer l'outil. De plus la norme de format de données des requêtes http est le JSON ou JavaScript Object Notation, et Node.js est une plateforme logicielle en JavaScript. Le choix sera donc porté sur cet outil. Concernant la partie SQL, le choix de l'outil n'a peu voire pas d'influence sur les performances. Cela dépend en grande partie de l'optimisation des requêtes SQL, de l'utilisation réfléchie des index, et de l'implémentation de la base de données. Cette partie déjà réalisée ne concernera pas le travail à effectuer sur cet outil hormis sur ces performances. Nous ferons appel à une base de données déjà conçue et fonctionnelle avec des requêtes générées par le framework que va utiliser l'outil.

3.3. Architecture logicielle

Le framework NestJS, qui utilise le langage de programmation Type Script, structurera l'outil pour la réalisation de cette mission au sein d'ITESOFT. C'est un framework open-source lancé en 2017 pour la création d'applications backend Node.js. Il se repose sur une architecture modulaire fortement inspirée d'Angular et MVC, ce qui facilitera l'organisation du code en unités fonctionnelles cohérentes.

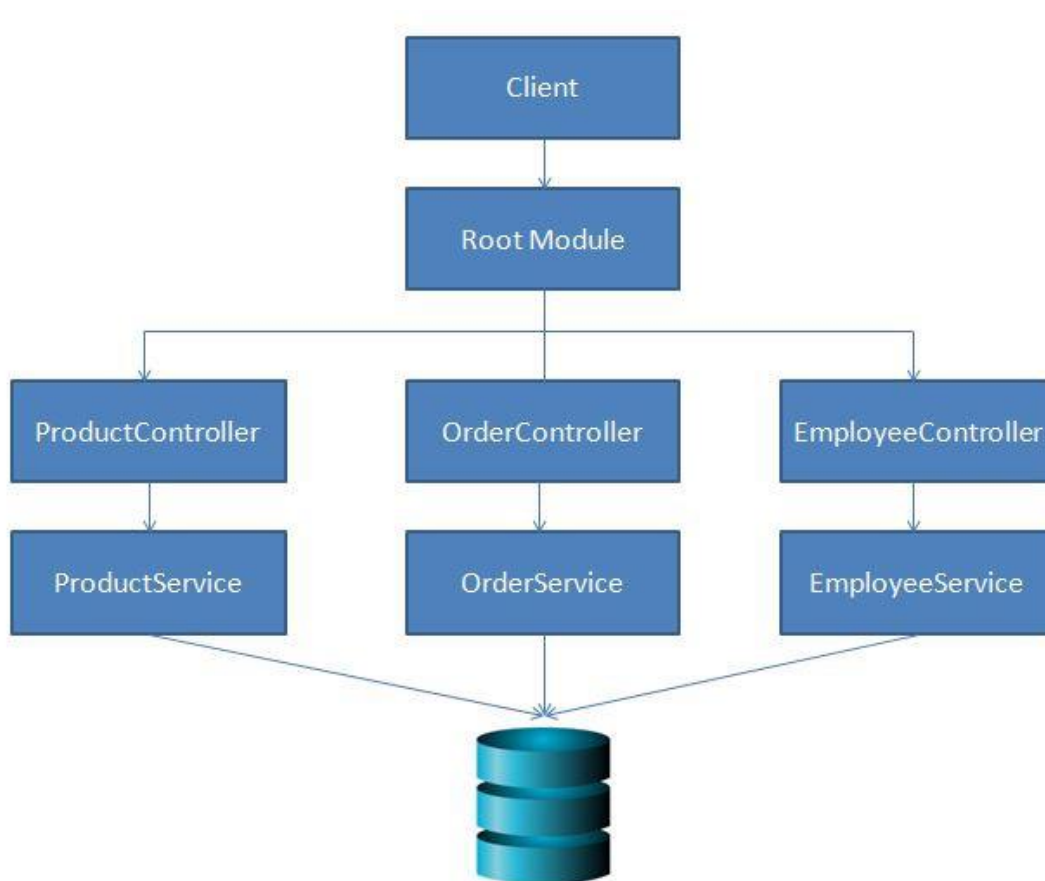


Figure 9 - Exemple d'architecture d'un projet NestJS

Cette architecture est structurée autour de trois composants fondamentaux :

- Contrôleurs : gestionnaires de routes qui traitent les requêtes http et utilisent les services
- Fournisseurs : services et autres classes qui implémentent la logique métier
- Modules : unités d'organisation qui encapsulent les contrôleurs et fournisseurs

Cette séparation permettra une structure favorisant le principe d'injection de dépendances, et sera plus ouvert à l'ajout de nouvelles fonctionnalités et fermé aux modifications, ce qui correspond bien aux besoins d'évolutivité de l'outil au-delà de cette mission.

3.4. Réalisation

3.4.1. Préambule : Déploiement local de SLI et d'une BD Mock

Pour pouvoir développer le traitement fonctionnel d'export de factures de Document Exporter, cela nécessite d'abord de déployer en local Streamline for Invoices, et de simuler un contexte de données d'un faux client avec une base données contenant une grande volumétrie de fausses factures, similaire à une situation en condition réelle. Pour accélérer le déploiement de tous les conteneurs que cela nécessite, l'équipe R&D a développé un module SLFI-Packaging, qu'il suffit de cloner et exécuter un script build-project.sh.

```
Capture:
  Downloading... http://seraimapp04:8081/repository/maven-releases-itesoft/com/itesoft/
i/capture/capture-4.14.4.zip
  Download success
  Unzipping... archive
cp: warning: behavior of -n is non-portable and may change in future; use --update=none inste
  Unzip success into package/docker/slfi/capture
  Download and unzip of capture DONE

document-proxy:
  Unstable version detected (25.3.1-master)
  Downloading... https://unreleased.raw.repo.itesoft.net/itesoft/solution/slfi/document-
cument-proxy-25.3.1-master.zip
  Download success
  Unzipping... archive
cp: warning: behavior of -n is non-portable and may change in future; use --update=none inste
  Unzip success into package/docker/slfi/document-proxy
  Download and unzip of document-proxy DONE

scpas:
  Unstable version detected (25.3.16-master)
  Downloading... https://unreleased.raw.repo.itesoft.net/itesoft/solution/slfi/scpas/25.
[#####] 62%[[124;5~
```

Ce script récupère à partir du serveur Nexus toutes les images des modules de SLI (dont Capture, le Frontend, SCPAS, Invoice Converter...).

Il y a déjà eu des erreurs de pull dû à une erreur d'attribution DNS de ma machine, notamment des accès restreint par le server hébergeant les images des modules. Cela peut être réglé en ajoutant une règle DNS dans les hosts enregistrées dans la machine. Cela se corrigeant en effectuant cette commande : **vi /etc/hosts**, et en insérant dans le fichier la ligne suivante : **votreHostname.itesoft.local**.

```
3270f4e11550 Waiting 0.8s
0f8320b63c9b Waiting 60.1s
66dd4c37ec04 Waiting 60.1s
bad7fd92aa38 Waiting 60.1s
ab5c064552ba Waiting 60.1s
c8fb1c725d1d Waiting 60.1s
b09939f45d21 Waiting 60.1s
bb201c4bab07 Waiting 60.1s
pull access denied for itesoft/slfi/routing-manager-frontend, repository does not exist or may require 'docker login': denie
d: requested access to the resource is denied
```

Il faut ensuite monter la solution, qui se fait en exécutant le script run.sh up. Cela créera à partir de plusieurs docker-compose dans un dossier docker, un répertoire package, avec tous les conteneurs de chaque module de SLI interconnectés.

Une fois déployé, on peut se connecter en tant qu'administrateur via le portail web du logiciel. Dans la rubrique capture, il faut configurer l'agent dans la partie configuration de cette section.

Figure 10 - Configuration d'un agent de capture

Il faut ensuite affecter l'agent de capture au canal de flux fichiers d'où vont passer les documents. Le canal fichier correspond au flux de fichier entrant dans Omnicanal Capture. C'est la porte d'entrée des documents physiques et numériques dans SLI. Il suffit de cocher la liaison entre ces deux intermédiaires pour les connecter.

Figure 11 - Affectation d'un agent de capture à un canal fichier

Une fois la configuration terminée, il faut envoyer manuellement les documents en traitement dans Omnicanal Capture (OC). Ce dernier va d'abord « capturer » tous les documents, puis renvoyer un résultat de ce procédé.

Figure 12 - Capture de factures

La configuration de l'agent et du canal fichier peut être sauvegarder dans un fichier JSON, qui peut être importé dans SLI pour automatiser de prochaines configurations. Elles seront automatiquement transférées à Rabbit MQ [10], qui est un système de file d'attente. Les documents vont être consommés par SCPAS (anciennement Balance), qui effectue un traitement par lot complexe des processus métier dans SLI. C'est notamment durant ce procédé que les documents sont ensuite créés et envoyés en base de données.

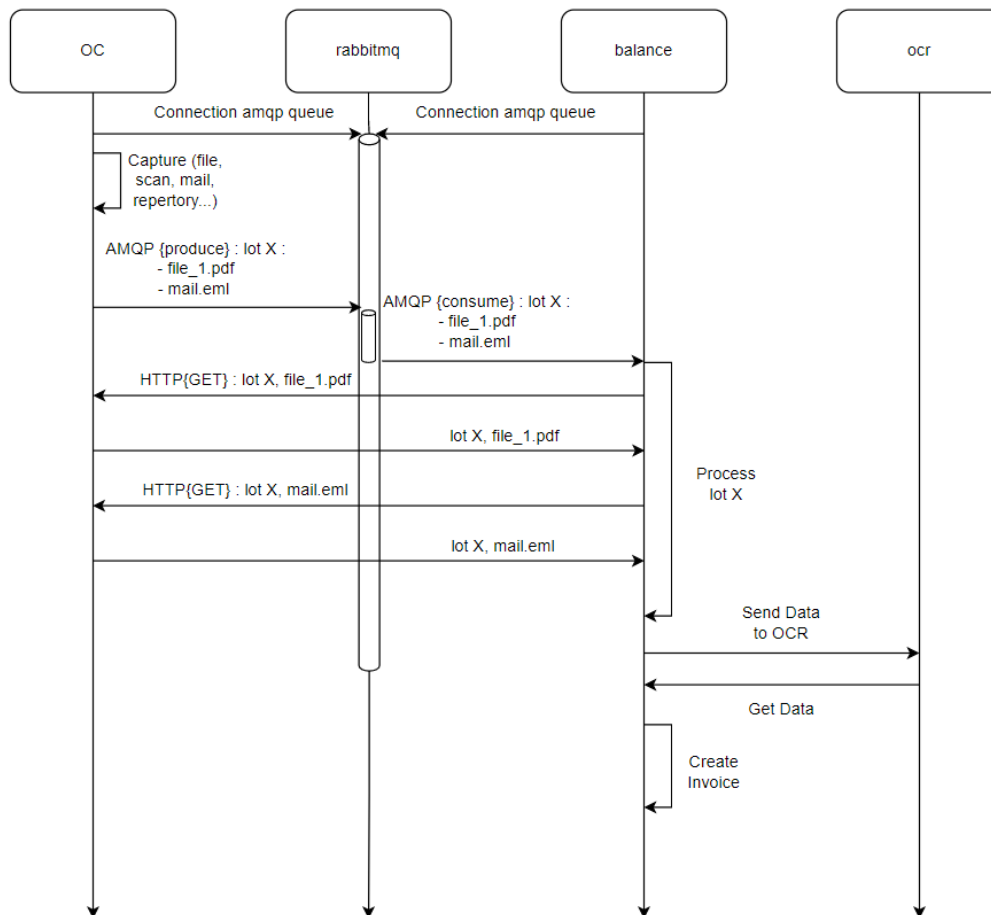


Figure 13 - Séquence de traitement d'une facture dans SLI

Au début de la mission, de nombreux problèmes internes ont eu lieu lors du traitement dans SCPAS (balance). Cela est causé par fait que dans les variables d'environnement, l'URL de callback de SCPAS par défaut est erronée au moment de cloner SLFI-Packaging. C'est parce que cette adresse a été modifiée dans son fonctionnement en production, et non dans les variables d'environnement du conteneur Scpas-rules dans SLI-Packaging.

Mes captures 2				Toutes les captures manuelles				Toutes les captures automatiques			
En cours	0	Suspendue	0	En erreur	2	Annulée	0	Terminée	0		
Nom	Flux	Canaux	Date de création	Modifiée le	Activité	Etat	Utilisateur	Mode	Actions		
20250206_00002	Flux fichier	1	06/02/2025 14:40	06/02/2025 14:55	Export	En erreur	Benoit P.	Manuel			
20250206_00001	Flux fichier	1	06/02/2025 14:40	06/02/2025 14:40	Export	En erreur	Benoit P.	Manuel			

Figure 14 - Erreur de traitement d'Omnicanal Capture dans SLI

L'envoi manuel des factures à Omnicanal Capture présente une problématique : en condition réelle d'utilisation, Document Exporter devra détenir plusieurs centaines de milliers de factures. Pour accélérer ce processus, l'équipe QA a déployé un workflow avec n8n [9], pour déposer automatiquement un nombre défini de documents en traitement, et donc en base de données.

Dépot de factur X sur une VM

Type de plateforme *

R&D

Path

Default : /srv/slfi/capture/agents/docker/volumes/input/

Number of invoices *

10000

VM Name

CLIMHY

Submit form

Form automated with n8n

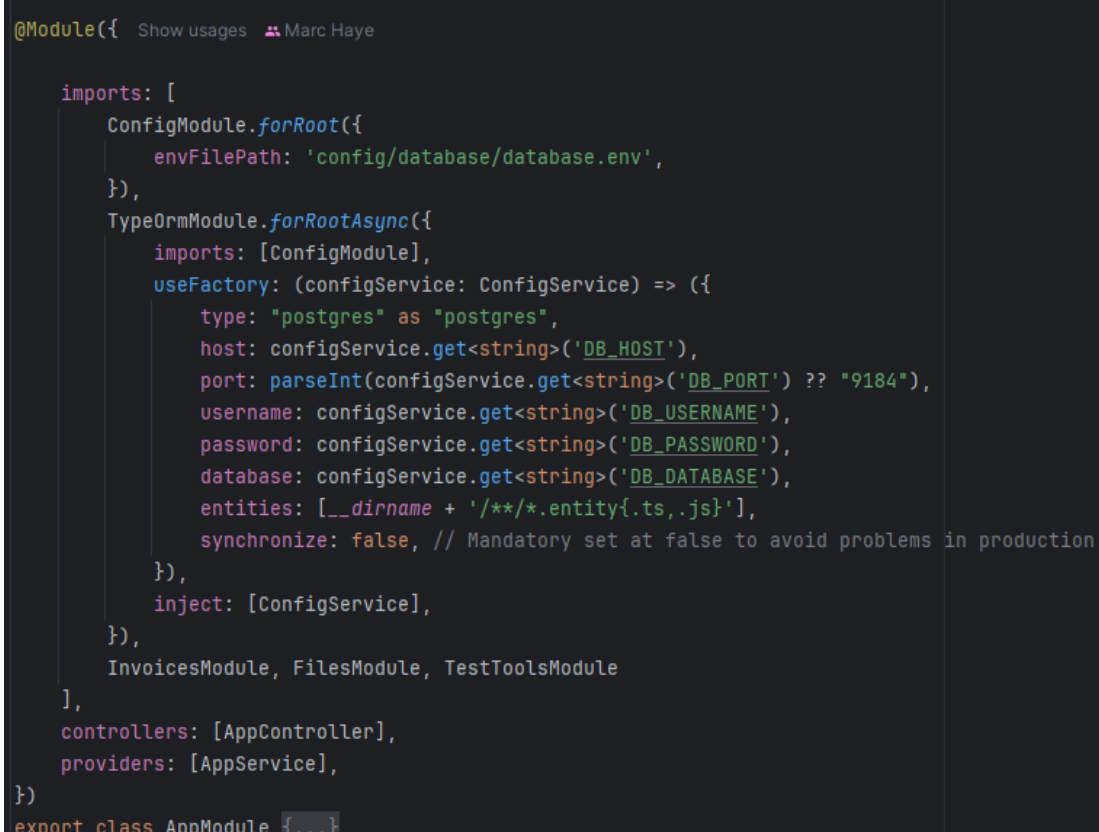
Figure 15 - Formulaire du workflow automatisant le dépôt de facture en BD

Avec ce workflow, il devient très simple et très rapide de simuler une base de données avec une grande volumétrie. Cela permettra d'effectuer des tests de l'outil sur plusieurs milliers de factures.

Une autre problématique est à considérer : la base de données est déployée automatiquement par le conteneur scpas-database. Elle est donc liée à ce dernier. Si le conteneur est supprimé, la machine hébergeant ce conteneur de SLI vient à s'éteindre, redémarrer ou faire une mise à jour, la base de données est également supprimée. L'entièreté du procédé de ce préambule devra être effectuée de nouveau. Pour pallier ce problème, SLFI-Packaging fut déployé sur une machine virtuelle interne à l'entreprise (CLIMHY dans la figure ci-dessus), mis en place par le service IT, afin de m'aider dans la réalisation de l'outil de réversibilité. Elle restera en fonctionnement en continu, et permettra de conserver le déploiement de la base de données en marche, sans dépendre de la machine sur laquelle sera développé l'outil.

3.4.2. Première tâche : Initialisation NestJS et Connexion BD

Pour réussir à réaliser un outil fonctionnel et performant en respectant les contraintes de temps, il a d'abord été important de comprendre le fonctionnement de NestJS. En commençant par initialiser un nouveau projet avec ce framework. Pour initialiser un projet Nest, il faut utiliser son interface de commandes Nest Cli. Cela créait un projet avec un package.json pour gérer les scripts et dépendances, et un dossier src/ où se trouvera la structure de l'api. La connexion de l'API à la base de données PostgreSQL se fait avec la librairie postgres [5] et TypeORM [2], une bibliothèque de mappage objet-relationnel pour TypeScript et JavaScript.



```
@Module({
  imports: [
    ConfigModule.forRoot({
      envFilePath: 'config/database/database.env',
    }),
    TypeOrmModule.forRootAsync({
      imports: [ConfigModule],
      useFactory: (configService: ConfigService) => ({
        type: "postgres" as "postgres",
        host: configService.get<string>('DB_HOST'),
        port: parseInt(configService.get<string>('DB_PORT')) ?? "9184",
        username: configService.get<string>('DB_USERNAME'),
        password: configService.get<string>('DB_PASSWORD'),
        database: configService.get<string>('DB_DATABASE'),
        entities: [__dirname + '/*.entity{.ts,.js}'],
        synchronize: false, // Mandatory set at false to avoid problems in production
      }),
      inject: [ConfigService],
    }),
    InvoicesModule, FilesModule, TestToolsModule
  ],
  controllers: [AppController],
  providers: [AppService],
})
export class AppModule { ... }
```

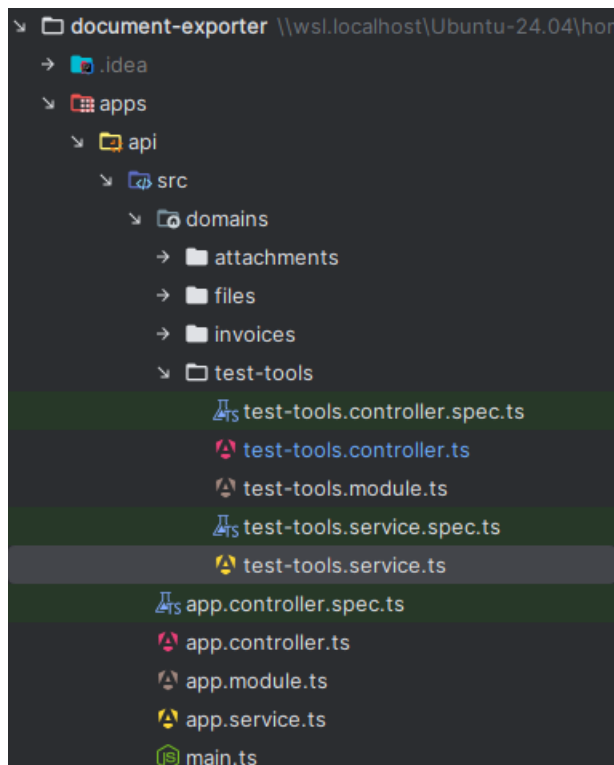
Figure 16 - Module App de l'outil de réversibilité

Pour configurer cette connexion, on utilise le module Configuration de NestJS [3] . Le ConfigService est un service proposé par NestJS pour charger une configuration spécifique (.env, .yaml, constantes...), et pour utiliser les variables chargées. J'utilise ensuite un fichier database.env pour fournir les différentes variables confidentielles de la base de données. Cela sera par la suite utilisé de façon plus générale dans le contexte de l'outil avec des constantes, par exemple pour pouvoir facilement configurer l'export, la documentation Swagger et également faciliter le déploiement. Par exemple, dans le fichier main.ts, on fournit en clair les informations à la construction de la documentation Swagger. On pourra alors à l'avenir configurer directement Swagger via une constante swagger.config interne à la structure de l'outil. Cela est également implémenté à la quatrième tâche.

```
// Swagger Creation
const swaggerConfig : Omit<OpenAPIObject, 'paths'> = new DocumentBuilder()
  .setTitle('Document Exporter REST API')
  .setDescription('Outil d'extraction et de réversibilité de données pour Streamline for Invoices.')
  .build()
```

Figure 17 - Configuration de Swagger

Pour commencer l'implémentation de la logique des services, il a fallu réfléchir à la structure du projet. La particularité de cette mission est qu'elle m'a été laissée libre à définir, et mon maître de stage évalue ensuite sa pertinence.



Pour conserver une possibilité d'évolution, j'ai choisi initialement de structurer l'outil sous un chemin apps/api/src/. Cela ne sera pas conservé par la suite durant le refactor par src/api car plus simple et non-nécessaire.

Figure 18 - Première architecture de l'outil

Résumé du travail réalisé :

- Initialisation de la configuration et structure par défaut de NestJS

- Restructuration de l'architecture spécifique au module Document Exporter
- Connexion de l'API avec BD PostgreSQL

3.4.3. Deuxième tâche : Export des factures

Cette tâche va implémenter trois traitements principaux :

- Exposer une route http pour lancer le procédé de l'API
- Récupérer en base de données les données JSON des factures
- Écrire un fichier CSV à partir de ces données.

J'ai également pris la décision de faire un contrôleur test-tools qui sera temporairement utilisé pour appeler tous les services contenant la logique. Cela sera réorganisé une fois le procédé d'export fonctionnel. Puisque l'on respecte l'injection de dépendances avec les services, il sera facile d'appeler ces services dans leur futur contrôleur adéquat.

Un contrôleur génère une route accessible avec une requête http, en l'occurrence de type POST, car elle va créer un export de documents. Avec Nest, on précise une route avec un décorateur :

```
@Post('export-documents/:tenantId')
```

Les contrôleurs doivent se contenter uniquement d'injecter les services dont ils ont besoin, et de les appeler. Les services contiennent et encapsulent toute la logique de l'API.

```
async exportDocuments(@Param('tenantId') tenantId: string): Promise<Record<string, string>> {
  await this.testToolsService.exportDocuments(tenantId);
  return {
    message: `Starting documents export for tenant id: ${tenantId}`,
  };
}
```

Figure 19 - Méthode exportDocument appelé par la route API

On renvoie une réponse précisant le déroulement avec succès de son envoi, ainsi qu'un code de retour HTTP. Selon REST, le code de retour succès d'une requête POST est 201. Ce message étant renvoyé en clair, il serait intéressant à l'avenir de pouvoir configurer les messages de retour renvoyés par l'API. Par exemple avec l'utilisation de constantes : `RESPONSE_MESSAGES.POST_SUCCESS`. Cela permettrait également d'avoir une meilleure gestion des exceptions et qu'elle soit plus configurable. Le `:tenantId` correspond à une variable saisie dans la route. Des décorateurs sont ajoutés à chaque route pour décrire son fonctionnement. Swagger récupère ces informations pour générer la documentation Swagger :

```
@Post('export-documents/:tenantId') no usages new *
@ApiOperation({
  summary: 'Start extracting all tenant's documents for processing.',
  description: 'Export documents, and generate a arborescence with PDF files, and a CSV containing all the processed information.\n\n',
})
@ApiConsumes('application/json')
@ApiHeader({
  name: 'tenantId',
  description: 'Unique identifier for the tenant/organization',
  required: true,
  example: '1',
})
async exportDocuments(@Param('tenantId') tenantId: string): Promise<Record<string, string>> {
```

Figure 20 - Décorateurs de la route du contrôleur

On récupère les données des factures du tenant dans la base de données dans un service dédié aux factures, InvoicesService, afin de correspondre au principe de responsabilité unique :

```
@Injectable() Show usages  Marc Haye *
export class InvoicesService {

  constructor( no usages  Marc Haye
    @InjectRepository(Invoice)
    private invoicesRepository: Repository<Invoice>) {
  }

  Récupère la facture avec l'id donné en paramètre.

  Params: tenantId – Identifiant du tenant/organisation

  Returns: l'objet de la facture correspondante

  async findForTenant(tenantId: string): Promise<Invoice[]> {
    return await this.invoicesRepository.findBy({
      tenant_id: tenantId
    });
  }
}
```

Figure 21 - Aperçu du service Invoice

L'objet invoicesRepository de type Repository va dans la méthode findBy() générer une requête préparé SQL sur la table 'invoice'. TypeORM convertira les données en un tableau de factures. L'appel de cette fonction est fait par le service test-tools, qui assure pour l'instant la logique de l'export.

```
export class TestToolsService {
  constructor(private readonly invoicesService: InvoicesService) { no usages  Marc Haye
  }

  Récupère toutes les données correspondant aux documents du tenant Crée un fichier CSV à
  partir des données fournies et le sauvegarde dans un dossier spécifique

  Params: tenantId – Identifiant du tenant/organisation

  async exportDocuments(tenantId: string): Promise<void> { Show usages  Marc Haye
    let invoices_data : Invoice[] = await this.invoicesService.findForTenant(tenantId);
    await this.createCSVFile(invoices_data, './', tenantId);
  }
}
```

Figure 22 - Première version du service Test Tools

Il crée ensuite un fichier CSV avec les données reçus avec une fonction « createCSVFile ». Il génère un chemin d'accès du fichier, puis crée un flux (ou stream) d'écriture de fichier dans lequel sera fourni les données de toutes les factures. Afin d'éviter une dette technique pour toutes les fonctionnalités liées aux traitements de fichiers (création fichier et dossier, écriture donnée) j'ai implémenté un service File, qui gèrera en interne tous les appels à l'API extérieur File System [11] de Node.js.

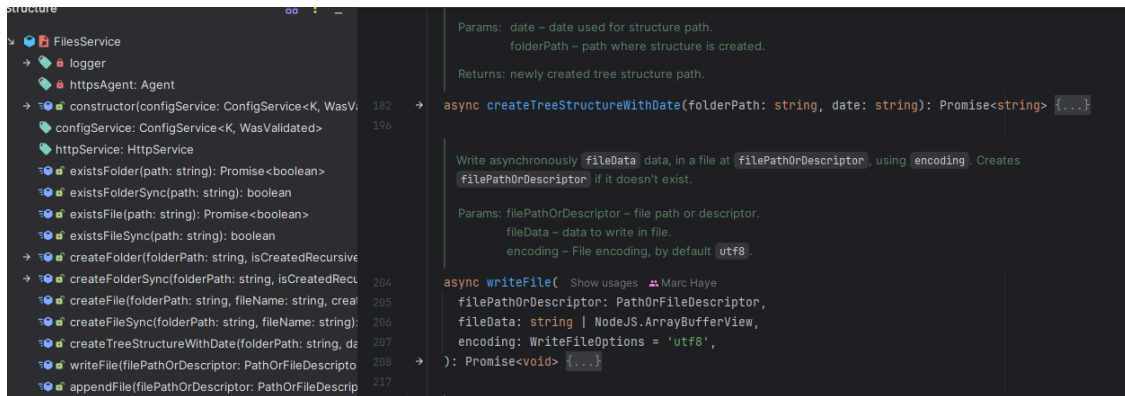


Figure 23 - Aperçu du service Files

On fait alors appel à cette librairie à chaque besoin de créer ou écrire dans un fichier, ce qui sera le cas pour la création du fichier csv.

```
// Chemin complet du fichier
const filePath: string = path.join(folderPath, `${fileName}.csv`);
// Créer un stream d'écriture
const writableStream : WriteStream = fs.createWriteStream(filePath);
```

Cette implémentation est fonctionnelle mais respecte pas le principe ouvert/fermé. Cette méthode permet la création de fichiers qu'en format CSV et uniquement avec des données d'objets de type factures. L'outil doit pouvoir par la suite exporter d'autres types de documents, comme des pièces jointes ou des historiques de factures. Il faudrait avoir une méthode plus générique, qui puisse traiter indépendamment des paramètres fournis.

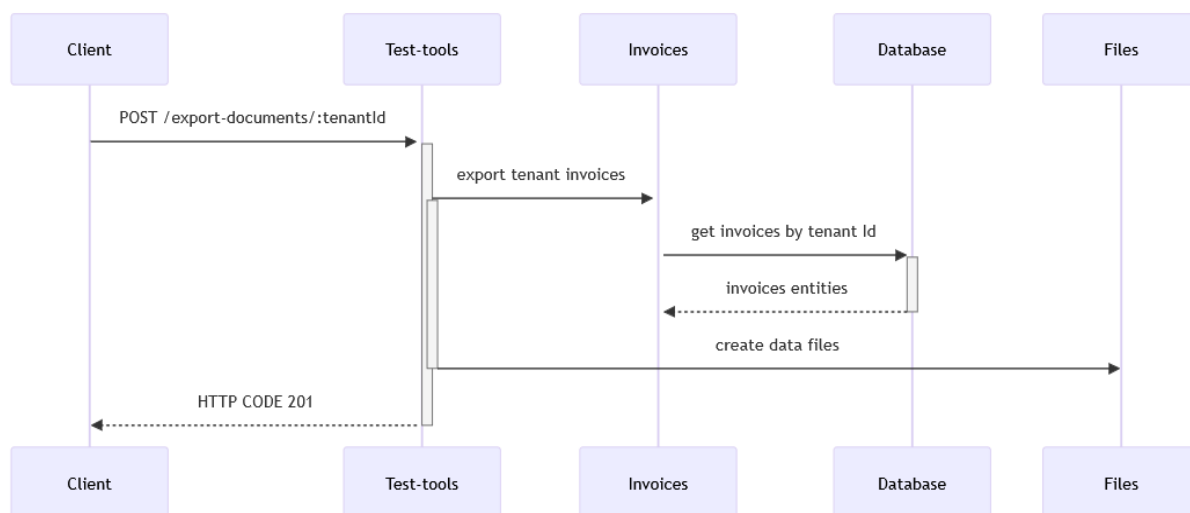
Résumé du travail réalisé :

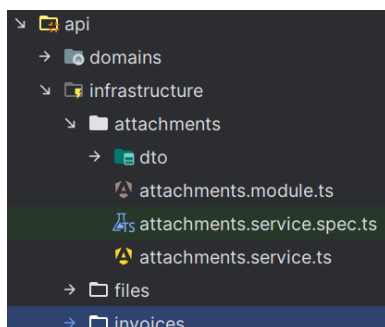
Figure 24 - Diagramme de séquence initial d'export de factures

- **Mise en place des premiers fonctionnalités temporaires :**
 - Fonctionnalité via un Endpoint, d'exportation des factures en JSON pour un tenant
 - Fonctionnalité via un Endpoint, génération de CSV
 - Fonctionnalité via un Endpoint, renvoi des CSV stockés côté back end, au client
 - Mise place de la documentation technique avec Swagger

3.4.4. Troisième tâche : Export des pièces jointes

Cette tâche va implémenter :

- Récupération des documents via requêtes http à Document Manager
- Création d'une arborescence à partir d'une date donnée
- Ecrire fichier JSON et PDF des pièces jointes.



Dans un premier temps sera créée une encapsulation Attachments (pièces jointes). Cela s'applique pour toute la structure d'un projet utilisant le framework Nest. Le module Attachment.module sera alors ensuite importé dans App.module comme l'a été Invoices.module, files.module et plus tard invoices-history et tenant.

Document Manager est le service au sein de l'entreprise qui stocke tous les fichiers documents (en format PDF, XML, EML). Il dispose d'une API par laquelle on peut récupérer les pièces jointes des factures. Nous utilisons dans un premier temps la librairie http de Node pour effectuer les requêtes GET.

```
Export attachment external PDF file by doing a request to Document Manager using
externalUrlPath].

Params: externalUrlPath – file path from Document Manager
        filePath – file path to create.

async exportAttachmentExternalFile(externalUrlPath: string, filePath: string): Promise<void> { Show usages Marc Haye
```

Figure 25 - Signature initial de exportAttachmentExternalFile

Par la suite, cela sera remplacé durant la fin de la réalisation par HttpModule, plus haut niveau et plus conforme au système de modules du framework Nest. Cette méthode posera par la suite un problème de principe SOLID d'inversion des dépendances : la méthode dépend des pièces jointes (attachements) mais devra être réutilisé par la suite pour les pièces jointes des factures. Parmi les contraintes fonctionnelles imposés, l'outil doit au moment de l'export des pièces jointes créer une arborescence selon la date de création du document. Cette valeur est stockée dans le json des pièces jointes. Un objet dto Attachment a été implémenté pour récupérer ses valeurs sous forme d'un objet manipulable.

```

@ApiProperty({
  description: 'Document creation date',
  example: '2025-03-05T15:17:09Z',
})
@Expose()
readonly creationDate: string;

@ApiProperty({
  description: 'Invoice external document url from Document Manager',
  example: '/balance/url/encodedFilePath',
})
@Expose()
readonly externalDocUrl: string;

```

Figure 26 - Décorateurs Swagger

Pour utiliser en développement ces valeurs, on peut ensuite les appeler en tant qu'objet à l'aide du langage TypeScript.

```

for (const attachment of attachments) {

  const attachmentCode : number = attachment.code;
  const creationDate : string = attachment.creationDate;
  // External Document Manager file url
  const externalDocUrl : string = attachment.externalDocUrl;

```

Figure 27 - Exemple d'utilisation d'un objet DTO

L'adresse de la pièce jointe correspond au hostname/externalDocUrl, cette valeur a pu être récupéré également à partir du JSON de la pièce. On peut donc l'utiliser pour faire une requête à l'API de Document Manager et y exporter les documents pièces jointes.

```

// TODO : faire une variable d'environnement pour l'URL par défaut
// File url to request
const fileUrl = new URL(path.join('https://climhy.itesoft.local', externalUrlPath))

```

```

// Request execution
https.get(fileUrl, options, async (response) => {

```

Figure 28 - Requête à l'API de Document Manager

Une fonction générique createTreeStructure a été créée pour automatiser et pouvoir réutiliser ce procédé.

La tâche est fonctionnelle et crée correctement en asynchrone l'arborescence des documents des pièces jointes.

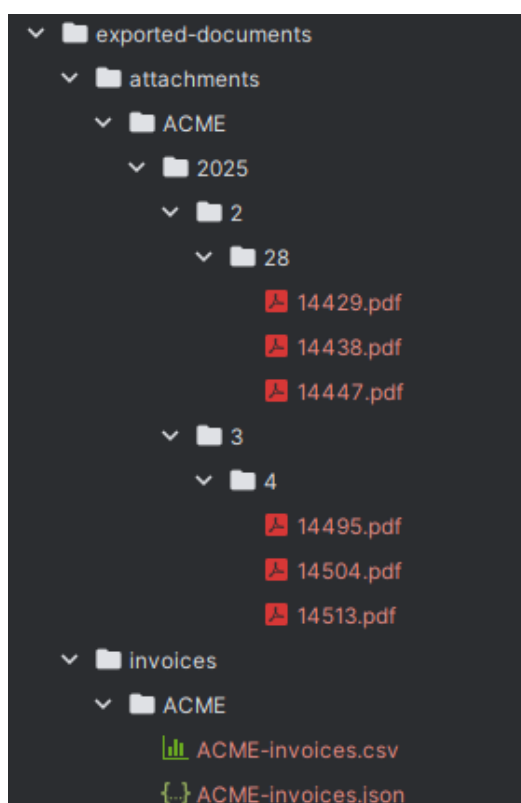


Figure 29 - Résultat d'arborescence d'un export après traitement par l'API

Résumé du travail réalisé :

• **Prise en charge automatisé des pièces jointes**

- Automatisation de l'export et de la génération de l'arborescence
- Asynchronisme général du procédé
- Utilisation de https pour récupérer données des pièces jointes via external doc url
- Ajout d'end points test-tools temporaires pour tester ces fonctionnalités

• **Implémentation single responsibility supplémentaire pour le service Files**

- Implémentation asynchrone pour création et écriture de fichier / répertoire
- Généricité des fonctions préexistantes (sera amélioré par la suite)

3.4.5. Quatrième tâche : Refactor et Configurabilité de l'API

A partir de cette étape, il est nécessaire que l'outil soit l'objet d'un important refactor, pour restructurer l'outil mais aussi afin respecter au mieux les principes SOLID. Cela permettra à l'avenir d'éviter d'accumuler une dette technique au fil de l'implémentation.

Nous utiliserons le module Configuration [3] de Nest pour rendre configurable tous les paramètres de l'API. Plusieurs constantes sous format JSON sont créées pour faire l'intermédiaire entre le développement et les variables d'environnement.

```
export const exportConfig : () => { readonly export: { readonly export... } = () => ({ Show usages Marc Hayes
  export: {
    exportDirectory: process.env.EXPORT_DIRECTORY || './exported-documents' as string,
    exportRejectUnauthorized: process.env.EXPORT_REJECT_UNAUTHORIZED === 'true',
    exportEncoding: process.env.EXPORT_ENCODING || 'utf8' as BufferEncoding,
    exportLimit: process.env.EXPORT_LIMIT || 100 as number,
  },
}) as const;
```

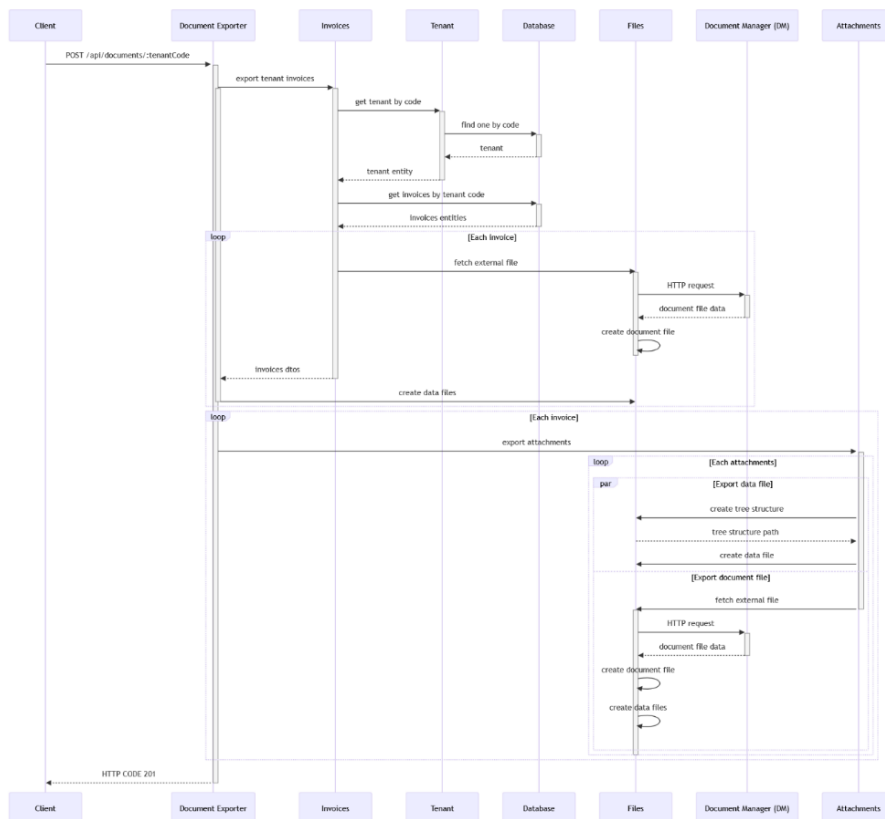


Figure 30 - Diagramme de séquence fonctionnelle de réversibilité global dans l'API

Résumé du travail réalisé :

- **Restructuration globale de l'app**

- Api séparé entre domaines pour end points, et infrastructure pour traitement BD.
- Les process d'export sont passés dans un Controller document exporter
- test tools est désormais uniquement dédié aux exemples

- **Implémentation globalisé de NestJS Configuration**

- Utilisation de configModule et configService pour app, Swagger et database
- Les variables d'environnement, déplacées dans docker/env (pour prévoir futur déploiement)

3.4.6. Cinquième tâche : Export des historiques de facture

Comme l'architecture de l'outil respecte désormais les principe SOLID « Open/Close », il a été facile d'implémenter l'export d'historiques des factures. Il existe dans la base de données une table Invoices history. Il a suffi dans un premier temps créer un nouveau module Invoices History, et de faire appel au services Files pour créer en parallèle les fichiers contenant les données des historiques.

Résumé du travail réalisé :

- **Export automatisé d'historique de facture**
- **Implémentation de invoice history**
 - Module, service, dto et entity
- **Implémentation de tenant (objet client)**
 - Module, service, dto et entity
 - Obtention d'information du tenant via bd
- **Refactor de document exporter service et invoice service**

3.5. Validation

3.5.1. Tests en production

Durant la réalisation, de premiers tests ont pu être effectués avec l'équipe QA dans les services cloud, contenant une volumétrie de copies de plusieurs millions de factures. Cela a pu permettre dans un premier de tester les limites de l'outil à ce stade de développement, mais également d'y apporter un regard différent sur les enjeux de sa réalisation. On a pu constater que l'outil était aussi performant que ce qu'il le fut en développement. Cependant, il causait une erreur après une tentative d'export de plus de dix mille factures, dû à une limitation mémoire du processus d'exécution de Node.js. En effet lors de l'export, les factures sont toutes allouées dans une variable de type tableau de factures. Node.js est certes conçu pour les tâches asynchrones, son environnement d'exécution est lancé sur un seul thread, c'est-à-dire un seul cœur de processeur. Cela cause une dette technique qui ne pourra s'améliorer si l'on souhaite effectuer des traitements plus importants, d'au moins un million de documents. L'équipe QA a donc proposé des pistes pour améliorer le traitement SQL. Plutôt que l'on effectue une requête SQL et un traitement sur toute la table de factures. Que l'on effectue des requêtes SQL en traitement par lots. Cependant, je pris la décision de ne pas implémenter cette possibilité, faute de temps avant le déploiement final en production. Cela reste cependant une perspective d'évolution à l'avenir, mais qui reste nécessaire pour assurer la validation de l'outil. Cette implémentation aurait pu se faire avec deux solutions possibles :

- Avec des offsets : Ils permettent d'effectuer une requête par lots dans une table. Il est plus facile d'implémenter cette solution mais il restera une problématique de coût technique, car on effectuerait une requête SQL par lot à la base de données.
- Avec les curseurs SQL : Utilisé en PL/SQL, on peut effectuer une seule requête dans toute la table, et traiter le résultat de la requête SQL par lots. Cette solution est plus difficile à développer mais serait le meilleur choix car une seule requête à la base de données est effectuée.

3.5.2. Conteneurisation de l'outil

L'étape de conteneurisation s'est faite avec Docker. Un docker compose se charge de monter le conteneur de l'outil, et pourra être déployé ainsi en production. Il suffira de modifier les variables d'environnement dans les fichiers .env pour mettre à jour le comportement de l'API.

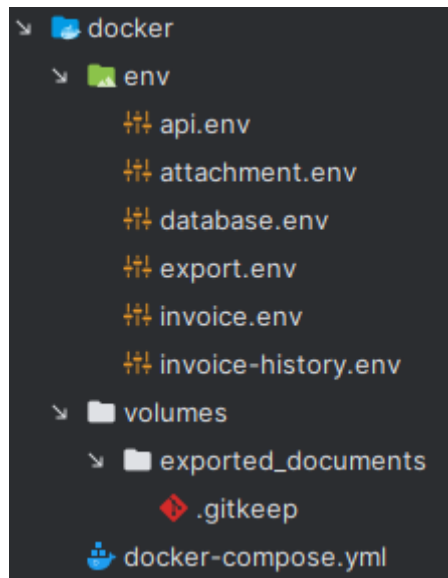


Figure 31 - Aperçu de la partie Docker de l'outil

Une fois l'image créée on peut monter le conteneur contenant tout l'environnement de l'outil. Le contenu a bien pu démarrer et est fonctionnel, il est possible d'y accéder directement via l'onglet Services de IntelliJ IDEA.

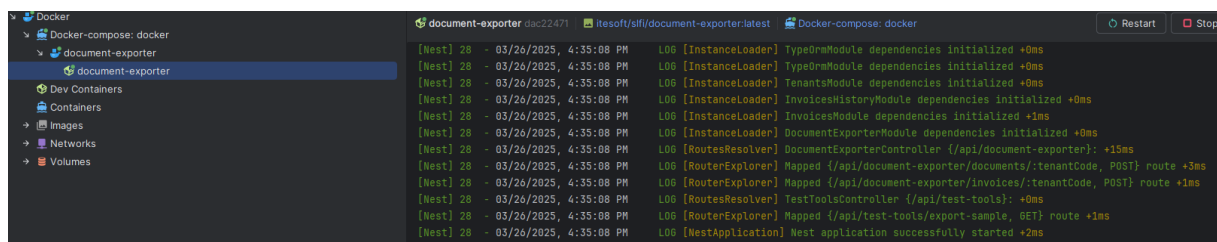


Figure 32 - Aperçu du conteneur dans les services d'IntelliJ IDEA

4. Manuel d'utilisation

L'outil de réversibilité est déployé en production et sera automatiquement interpellé par les serveurs de SLI* en effectuant une requête http. De plus, n'ayant pas d'interface, il est difficile de présenter et visualiser les résultats de l'outil. Une solution possible est de déployer localement l'outil et de le tester avec une version de SLFI-Packaging également déployée en local avec une base de données. Pour tester via Docker :

Pour créer l'image Docker :

```
docker build -t itesoft/slfi/document-exporter:latest .
```

Dans le dossier docker/ :

```
docker compose up -
```

5. Méthodologie et organisation du projet

5.1. Méthodes agiles et points réguliers

L'entreprise applique les méthodes agiles. Une réunion quotidienne, un « daily », est réalisée à 9h00 pour que chaque collaborateur R&D, moi inclus, s'exprime un à un sur ce qu'il a réalisé la veille, sur quoi porte le travail restant à réaliser, et partager les différents problèmes et obstacles rencontrés. De plus, un suivi quotidien est effectué à 10h30, seul à seul avec le maître de stage, pour exprimer l'avancement de la mission, explorer de nouveaux concepts pour la suite, et résoudre des problèmes précis lié à l'environnement de travail.

	03 Lun	04 Mar	05 Mer	06 Jeu	07 Ven
		Annulé : Annulé : SAVE THE DATE - Hackathon & Martin			
8					
9	Daily R&D SLI Réunion ↻	Daily R&D SLI Réunion ↻	Daily R&D SLI Réunion ↻	Daily R&D SLI Réunion ↻	Daily R&D SLI Réunion ↻
10	Rencontre Tuteur Réunion Alex Brous	Informatique Annulé ↻	Annulé : Annulé : Sui ↻	Annulé : Annulé : Sui ↻	Annulé : Annulé : Sui ↻
11					Annulé : Annulé : [SLI - I] Informations important
12	[SAVE THE DATE] HAPPY				

Figure 33 - Emploi du temps et aperçu des réunions quotidiennes

Les suivis quotidiens sont affichés ci-dessus comme étant annulés car la figure a été capturée à la fin de ce stage, mais ont bien eu lieu durant toute la mission. En parallèle, des réunions bimensuelles ont permis de faire des points sur le déroulement du stage.

5.2. Git flow

Le fonctionnement en entreprise de gestion de versions est assuré par Gitlab. Etant donné le nombre de collaborateurs travaillant en simultan  , s'ils envoyaient chacun des modifications sur une m  me branche distante sans aucun contr  le cela deviendrait rapidement difficile    g  rer et les probl  mes de r  gression ne tarderaient pas.

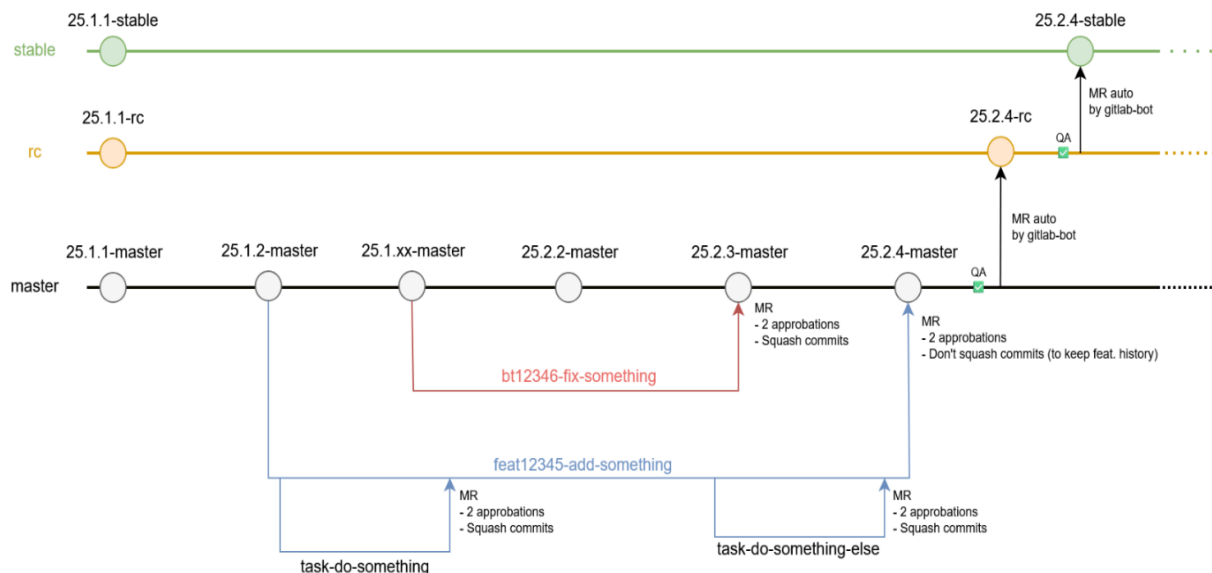


Figure 34 - Gitflow du d  veloppement de solution au sein d'ITESOFT

Ainsi, un Gitflow pr  cis est d  fini par les   quipes de l'entreprise. Le bloc BU* suit une forme hybride des m  thodes agiles et Scrum. Une version de leurs outils est d  ploy  e dans une forme « stable » apr  s en moyenne chaque mois de d  veloppement. Dans le cadre de mon stage, ma mission est d  finie en tant qu'une Feature avec un code d  fini : « feat 108623 – Restitution des donn  es ». Cette Feature est repr  sent  e par une branche feat-108623, et sera d  coup  e en diff  rentes t  ches   galement sous forme de branche. Voici un aper  u des branches de Document Exporter avec l'interface de IntelliJ IDEA :

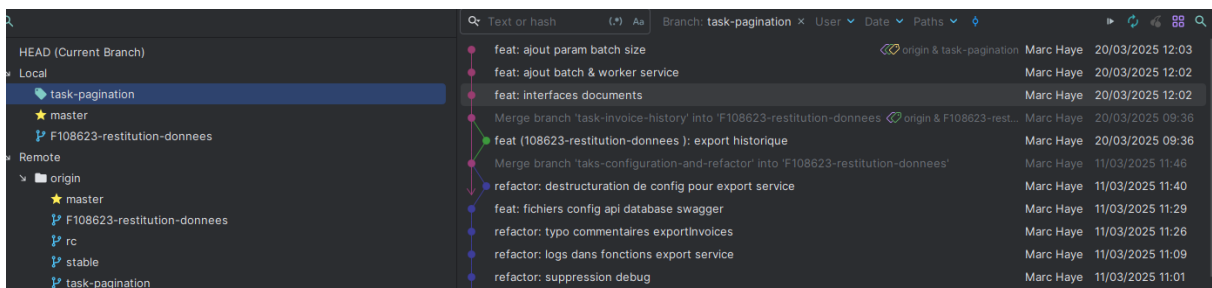


Figure 35 - Aper  u git de Document Exporter depuis IntelliJ IDEA

Il est interdit d'envoyer ses modifications sur la branche principale et les r  gles concernant le poussement de modifications locales sur le serveur distant   taient bien d  finies et strictes, englob  es dans ce qui est appel  e une demande de tirage. Par d  finition, une demande de tirage,

ou Merge Request, sert à envoyer ses modifications sur la branche principale du projet. En effet, les développeurs travaillent sur des branches dérivées de la branche principale puis demande la permission de les fusionner quand ils ont fini leurs tâches sur leurs branches. L'avantage de ce système au sein d'ITESOFT est que le travail effectué par le développeur doit être accepté par une autre personne (le plus souvent un autre développeur) avant d'être pris en compte. De plus, à chaque demande de tirage le serveur lance une génération des livrables (plus communément un « build » du projet) et la fusion des branches n'est réalisée que si tous les tests en place passent (intégration continue) et donc que la génération aboutie.

Dans le cadre de ce stage, lorsque ma tâche est considérée selon moi comme étant fonctionnelle, et complète, j'effectue une demande de poussée. Cette demande est réalisée manuellement via la plateforme Gitlab.

feat(108623-restitution-donnees): Automatisation des pieces jointes (attachments)

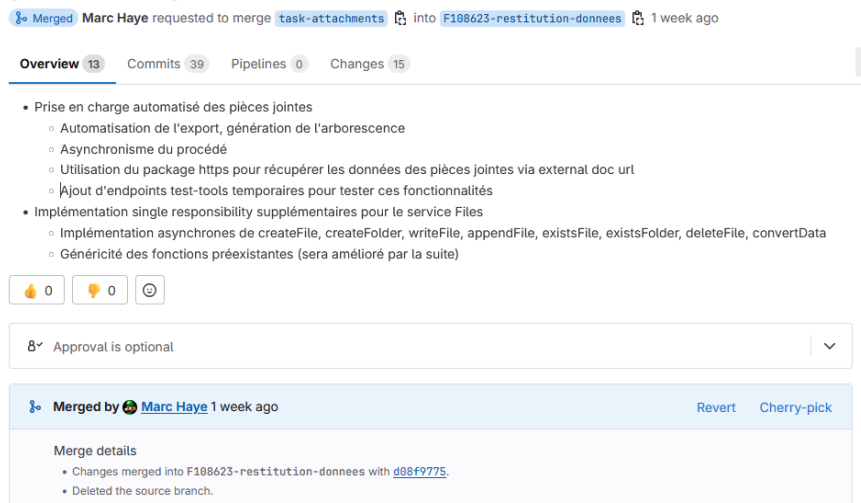


Figure 36 - Demande de poussée pendant le développement de Document Exporter

Un autre collaborateur au sein de l'équipe R&D (dans les faits ce fut mon maître de stage) se charge ensuite d'analyser toutes les modifications faites sur cette branche, et effectue ensuite différents retours sur ces modifications. Une fois satisfait, il approuve la demande, et les modifications apportés par la tâche sont poussées sur la branche de la Feature.

5.3. Autonomie et organisation personnelle

La réalisation de ce stage put être une réussite en grande partie grâce à l'encadrement de mon maître de stage, le travail en autonomie fut malgré tout un point très important. Un exemple certes anodin est d'avoir configuré l'environnement des marques pages de Firefox pour accélérer et fluidifier la navigation durant le stage.

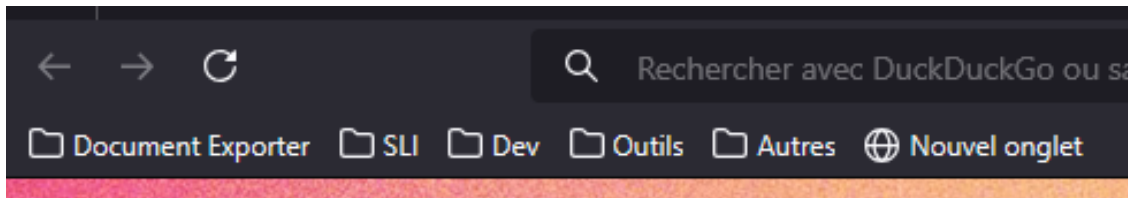


Figure 37 - Organisation dans le navigateur web

Pour la réalisation de ce rapport, des notes fut prises durant tout le stage pour documenter l'environnement de travail, ainsi que pour fluidifier et accélérer le travail personnel. Cela a demandé d'organiser une arborescence de pages, afin de pouvoir facilement y accéder.

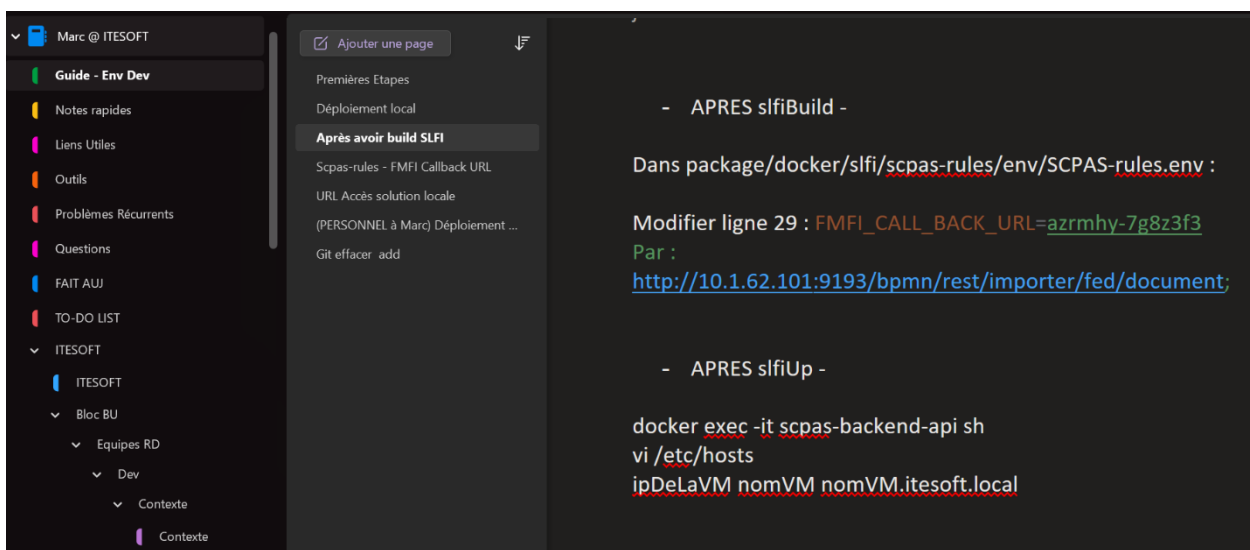


Figure 38 - Prise de notes régulière avec Microsoft OneNote

5.4. Tests en développement avec Postman

Tout au long de la mission, j'ai appris à utiliser le logiciel Postman pour tester le fonctionnement de l'API. On peut y créer des routes et configurer des variables d'environnement pour automatiser les tests.

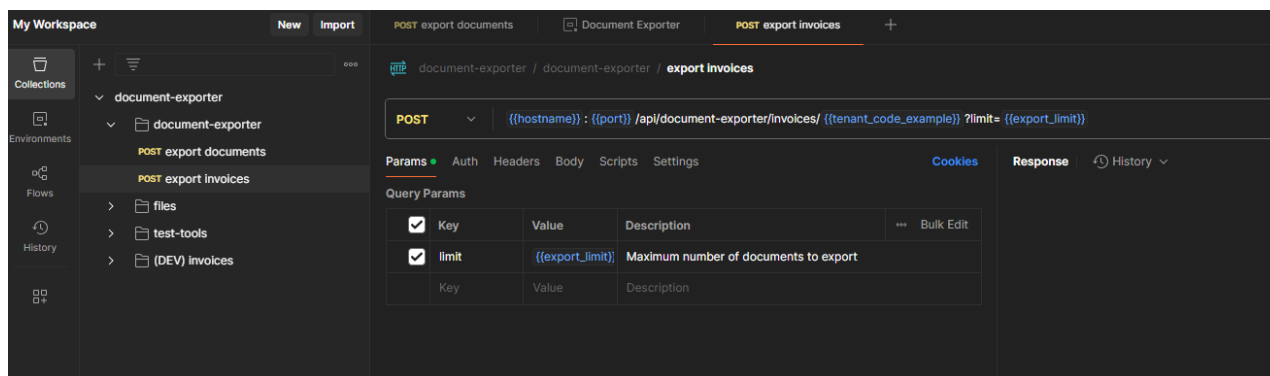


Figure 39 - Utilisation de Postman pour tester l'outil

5.5. Réalisation des diagrammes avec Mermaid

Durant la réalisation de la partie analytique, j'ai appris à utiliser l'outil Mermaid [8]. Un outil de modélisation basé sur la définition Markdown. Cet outil est très utilisé en entreprise et remplace peu à peu draw.io, principalement pour réaliser des diagrammes clairs et formalisés lors de l'industrialisation et la réalisation de logiciels. Je l'ai utilisé pour modéliser l'architecture de l'utilisation de l'outil, mais aussi pour décrire l'évolution de son séquençement fonctionnement. Concrètement, en appliquant du texte Markdown, Mermaid se charge de générer les modélisations et de les exportant sous différents formats comme SVG ou PNG.

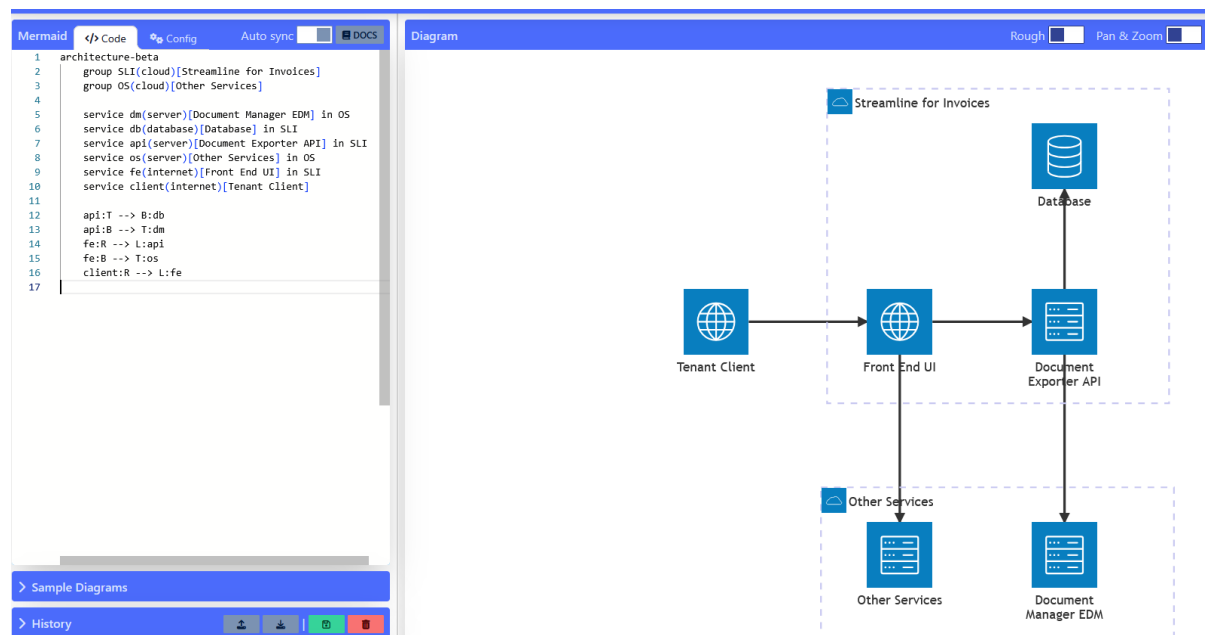


Figure 40 - Réalisation des diagrammes avec Mermaid live

6. Conclusion

Ce stage effectué au sein d'ITESOFT a non seulement été une concrétisation professionnelle des acquis de la formation que j'ai suivie à l'IUT, mais également une abondance de découvertes et de nouvelles compétences à développer.

6.1. Bilan du travail réalisé

L'industrialisation de l'outil a été achevée avec succès. J'ai pu considérablement améliorer le temps de traitement, en passant de plusieurs à seulement quelques heures. Il est également beaucoup plus structuré, plus ouvert à des évolutions futures, est versionné au sein de l'environnement technique R&D, ainsi qu'intégralement configurable et documenté. Le projet de déploiement de l'infrastructure a finalement été réalisé pendant mon stage : la documentation a été automatisée, restructurée et approfondie. La schématisation de l'infrastructure est achevée tant au niveau du matériel que des services, mais la partie de pagination et du traitement par lots n'a pas pu être réalisé. Néanmoins cela reste un point à visualiser dans une possible alternance ou repris par un autre développeur, d'où la nécessité de documentation.

6.2. Perspectives

L'outil de réversibilité pourrait être grandement amélioré sur plusieurs points. Le traitement par lots permettrait à l'outil d'atteindre ses capacités d'export initialement demandées d'un million de factures. De plus, les données des factures pourraient être écrites en parallèle dans le fichier CSV.

6.3. Bilan des acquis techniques

Une quantité considérable d'acquis a été fournis par ce stage, pour résumer les plus importants :

- Découverte de JavaScript côté serveur (Node.js) et du typage de TypeScript.
- Découverte du framework NestJS et de ses avantages d'implémentation.
- Cycle de vie d'une API : analyse, conception et réalisation et validation.
- Découverte du concept REST : Méthodes CRUD, verbes http, filtrage par offset.
- Manipulation de JSONB en base de données avec les entités Nest (entity).
- Manipulation de JSON en développement avec les DTOs Nest.
- Principe de déstructuration d'objets en JavaScript.
- Déploiement en production d'une image Docker.
- Prise de décisions en autonomie et en comprendre les différents enjeux mis en cause.
- Bonnes pratiques Git et utilisation professionnelle de Merge Request.

7. Visa du maître de stage

Vu le 27/03/2025.

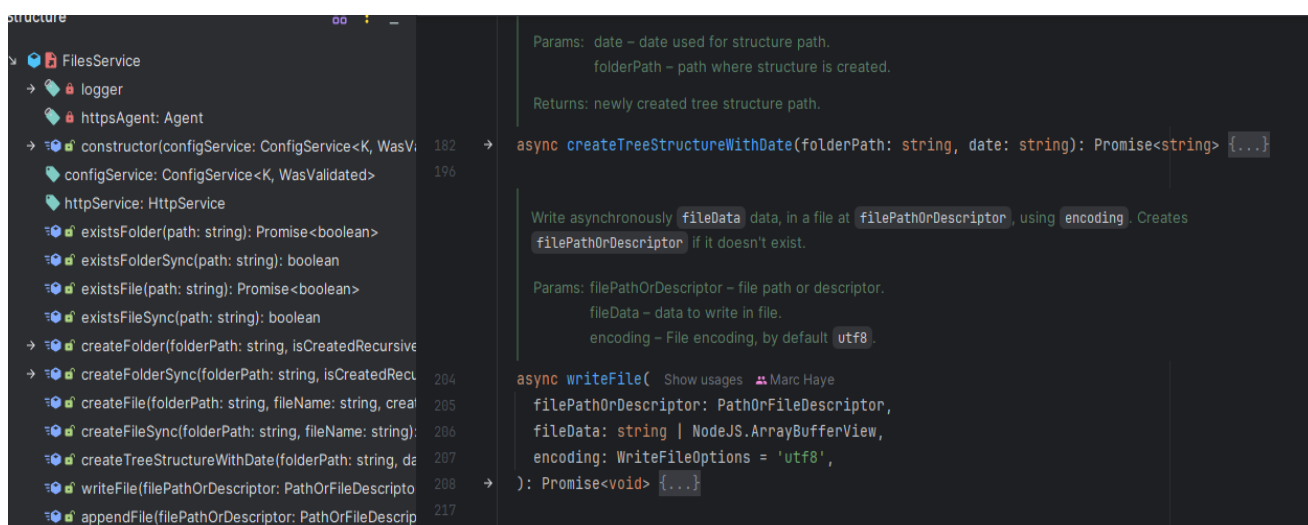
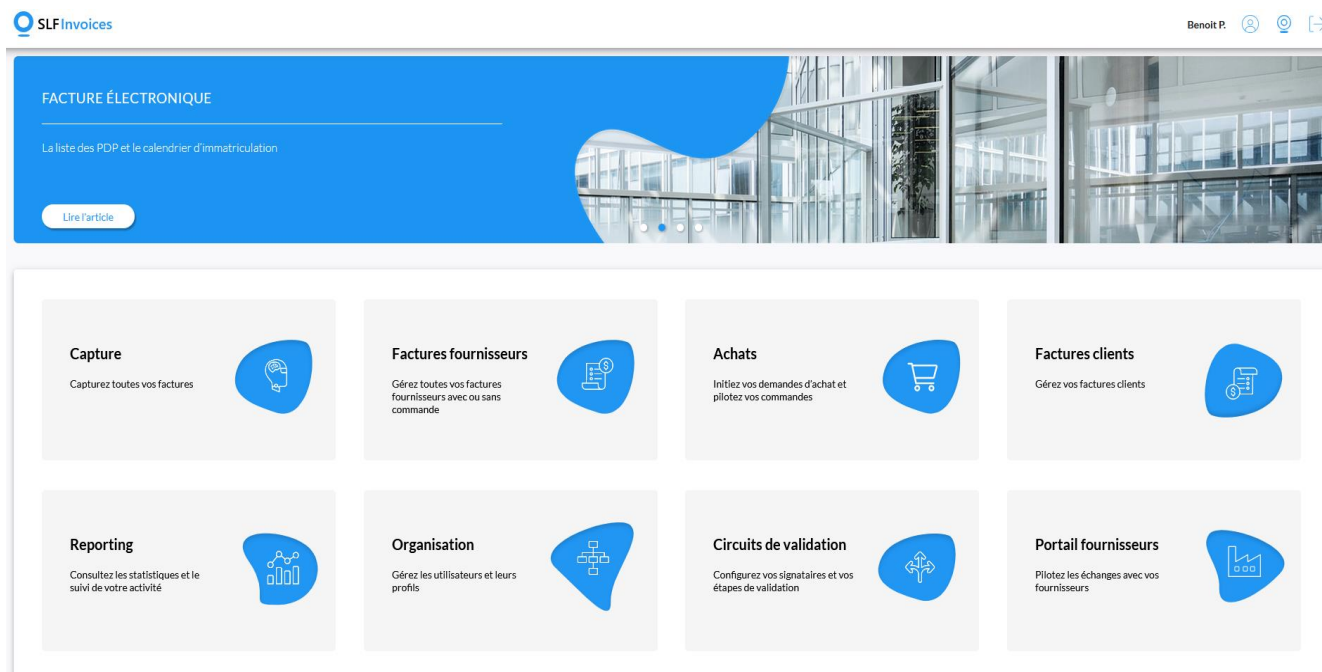
A handwritten signature in black ink, appearing to read 'M. HAYE', is written over a faint, horizontal, oval-shaped line.

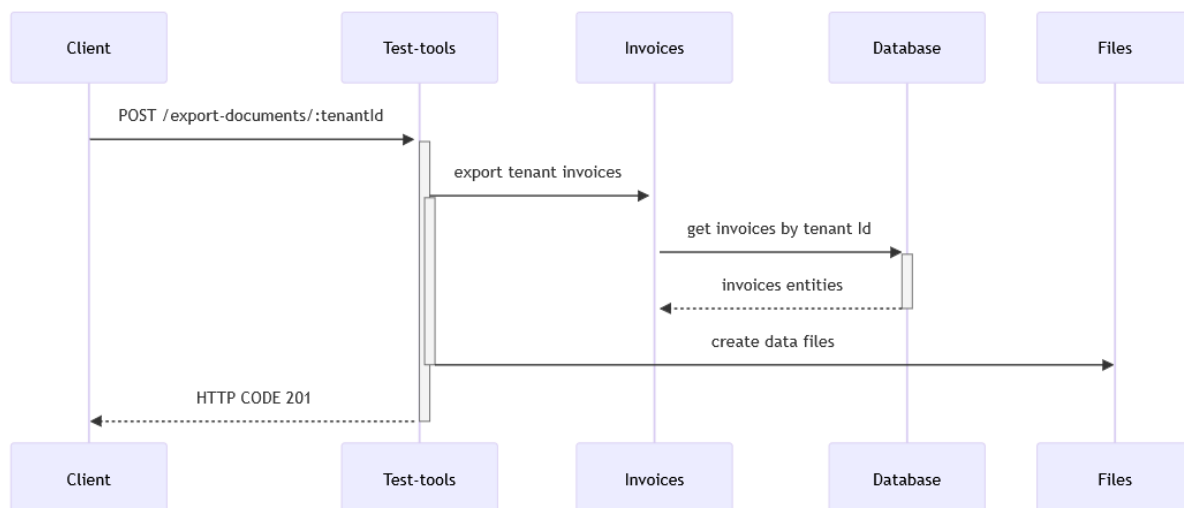
Bibliographie

Générée avec le logiciel Zotero.

- [1] PostgreSQL Global Development Group, "8.14. JSON Types - PostgreSQL Documentation," PostgreSQL, [En ligne].
Disponible : <https://www.postgresql.org/docs/current/datatype-json.html> [Consulté le 20 mars 2025].
- [2] OpenJS Foundation, "NestJS - Database," NestJS Documentation, [En ligne].
Disponible : <https://docs.nestjs.com/techniques/database> [Consulté le 26 mars 2025].
- [3] OpenJS Foundation, "NestJS - Configuration," NestJS Documentation, [En ligne].
Disponible : <https://docs.nestjs.com/techniques/configuration> [Consulté le 22 mars 2025].
- [4] SmartBear Software, "Swagger - API Documentation & Design Tools," Swagger, [En ligne]. Disponible : <https://swagger.io/> [Consulté le 21 mars 2025].
- [5] Postgres.js, "postgres - npm," npm, [En ligne]. Disponible : <https://www.npmjs.com/package/postgres> [Consulté le 21 mars 2025].
- [6] n8n.io, "Workflow Automation Software," n8n, [En ligne].
Disponible : <https://n8n.io/> [Consulté le 24 mars 2025].
- [7] International Organization for Standardization, "ISO/IEC 27001," ISO, [En ligne].
Disponible : <https://www.iso.org/isoiec-27001-information-security.html> [Consulté le 26 mars 2025].
- [8] Mermaid, "Mermaid Introduction," Mermaid.js, [En ligne].
Disponible : <https://mermaid.js.org/intro/> [Consulté le 21 mars 2025].
- [9] ITESOFT, "Wiki ITESOFT," ITESOFT, [Inaccessible].
Disponible : <https://wiki.itesoft.net/> [Consulté le 21 mars 2025].
- [10] RabbitMQ, "RabbitMQ Documentation," RabbitMQ, [En ligne].
Disponible : <https://www.rabbitmq.com/docs> [Consulté le 23 mars 2025].
- [11] Node.js, "File System Documentation," Node.js, [En ligne].
Disponible : <https://nodejs.org/api/fs.html> [Consulté le 26 mars 2025].

Annexes





localhost:3000/docs#/test-tools/TestToolsController_getExportSample

Swagger
Supported by SMARTBEAR

Document Exporter ^{1.0} OAS 3.0

Document Exporter REST API. Extract and export tool for SLI Streamline for Invoices.

document-exporter

- POST** `/document-exporter/documents/{tenantCode}` Start extracting all tenant's documents for processing.
- POST** `/document-exporter/invoices/{tenantCode}` Start extracting tenant's invoices for processing.

files

- GET** `/files/{tenantCode}` Extract processed file.

test-tools

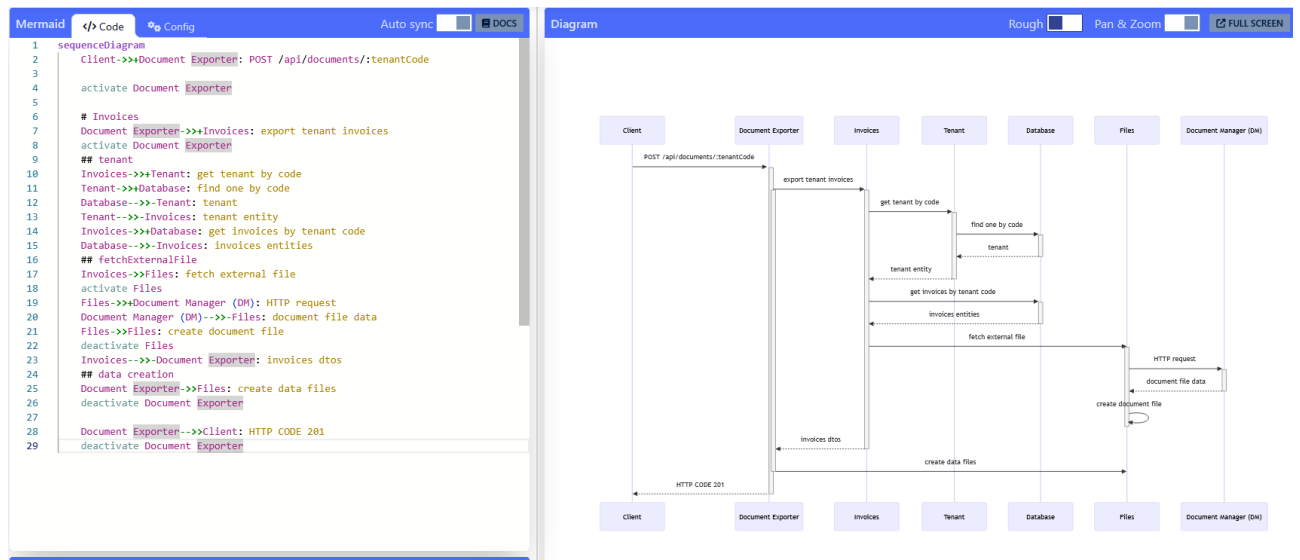
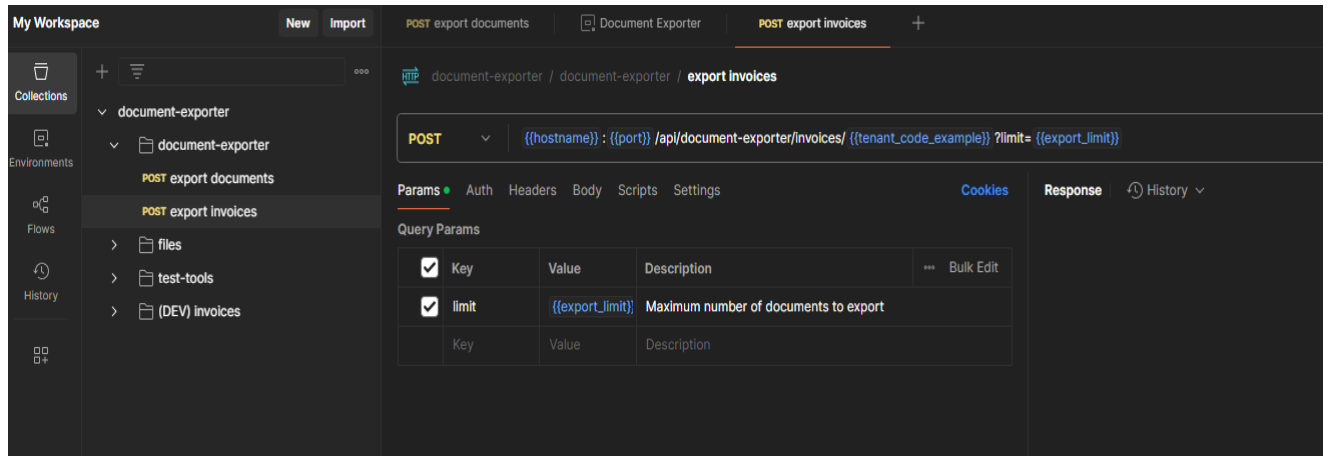
- GET** `/test-tools/export-sample` Export a generated csv containing sample invoices content.

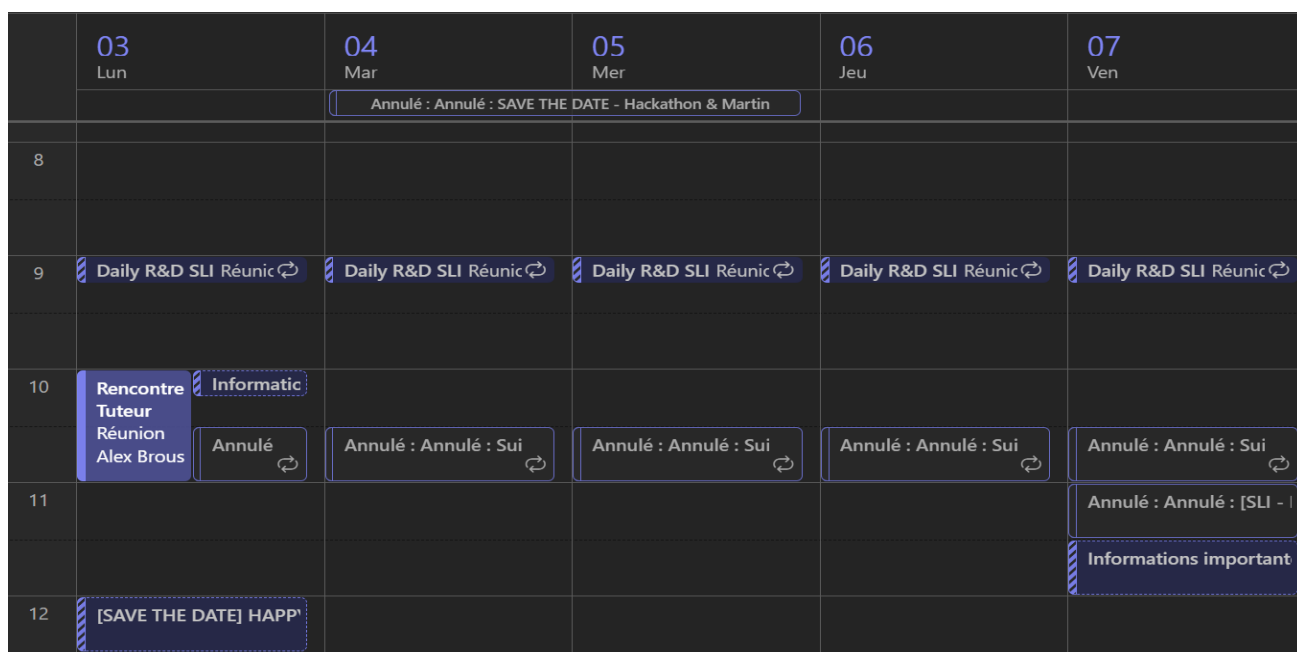
Generate a csv sample, and returns it as display.

Parameters Try it out

Name	Description
limit number (query)	Maximum number of documents to export

100





```

[Nest] 28 - 03/26/2025, 4:35:08 PM LOG [InstanceLoader] TypeOrmModule dependencies initialized +0ms
[Nest] 28 - 03/26/2025, 4:35:08 PM LOG [InstanceLoader] TypeOrmModule dependencies initialized +0ms
[Nest] 28 - 03/26/2025, 4:35:08 PM LOG [InstanceLoader] TenantsModule dependencies initialized +0ms
[Nest] 28 - 03/26/2025, 4:35:08 PM LOG [InstanceLoader] InvoicesHistoryModule dependencies initialized +0ms
[Nest] 28 - 03/26/2025, 4:35:08 PM LOG [InstanceLoader] InvoicesModule dependencies initialized +1ms
[Nest] 28 - 03/26/2025, 4:35:08 PM LOG [InstanceLoader] DocumentExporterModule dependencies initialized +0ms
[Nest] 28 - 03/26/2025, 4:35:08 PM LOG [RoutesResolver] DocumentExporterController (/api/document-exporter): +15ms
[Nest] 28 - 03/26/2025, 4:35:08 PM LOG [RouterExplorer] Mapped {/api/document-exporter/documents/:tenantCode, POST} route +3ms
[Nest] 28 - 03/26/2025, 4:35:08 PM LOG [RouterExplorer] Mapped {/api/document-exporter/invoices/:tenantCode, POST} route +1ms
[Nest] 28 - 03/26/2025, 4:35:08 PM LOG [RoutesResolver] TestToolsController (/api/test-tools): +0ms
[Nest] 28 - 03/26/2025, 4:35:08 PM LOG [RouterExplorer] Mapped {/api/test-tools/export-sample, GET} route +1ms
[Nest] 28 - 03/26/2025, 4:35:08 PM LOG [NestApplication] Nest application successfully started +2ms

```

feat(108623-restitution-donnees): Automatisation des pieces jointes (attachments)

Merged Marc Haye requested to merge task-attachments into F108623-restitution-donnees 1 week ago

Overview 13 Commits 39 Pipelines 0 Changes 15

- Prise en charge automatisé des pièces jointes
 - Automatisation de l'export, génération de l'arborescence
 - Asynchronisme du procédé
 - Utilisation du package https pour récupérer les données des pièces jointes via external doc url
 - Ajout d'endpoints test-tools temporaires pour tester ces fonctionnalités
- Implémentation single responsibility supplémentaires pour le service Files
 - Implémentation asynchrones de createFile, createFolder, writeFile, appendFile, existsFile, existsFolder, deleteFile, convertData
 - Généricité des fonctions préexistantes (sera amélioré par la suite)

👍 0 🗨️ 0 😊

8 Approval is optional

Merged by Marc Haye 1 week ago

Revert Cherry-pick

Merge details

- Changes merged into F108623-restitution-donnees with [d08f9775](#).
- Deleted the source branch.

