Dossier Projet

Titre Concepteur Développeur Web et Web Mobile

Application internet pour l'amélioration d'une application de gestion de micro-entreprise de la société ID Formation.

Encadrée par : Stéphane PIOTROWSKI
Elaboré par **:** SONDES ELLOUZI

Résumé

Le présent rapport est le fruit du travail réalisé pendant notre stage de fin d'études au sein d'ID Formation pour l'obtention du titre Concepteur développeur informatique.

Le projet réalisé vise à améliorer une application Web qui offre un tableau de bord complet avec des indicateurs clés, et une gestion d'entreprise complète, couvrant clients, offres de service, factures, finances, et calendrier, le tout interconnecté grâce à des API REST pour une expérience utilisateur fluide.

Le stage était une bonne opportunité de développer une excellente maitrise des langages de programmation ainsi la découverte de nouvelles technologies telles que Java, React native.

Abstract

This report is the fruit of the work carried out during our end-of studies internship within ID Formation to obtain **Professional title web and mobile developer**.

The project carried aims to improve a web application that offers a complete dashboard with key indicators, and complete business management, covering customers, service offers, invoices, finances, and calendar, all interconnected thanks to APIs REST for a smooth user experience.

The internship was a good opportunity to develop an excellent command of programming languages as well as the discover of new technologies such as Java, React native …

Introduction

Le présent rapport est composé de trois chapitres structurés comme suit :

- Le premier chapitre : présente le cadre général du projet en décrivant l'organisme d'accueil, la problématique du sujet ainsi que la solution proposée.
- Le deuxième chapitre : se concentre sur l'analyse fonctionnelle et technique du projet, la conception et les différents diagrammes d'UML utilisés ainsi que le modèle logique de données.
- Le troisième chapitre : se centralise autour de la présentation de différentes interfaces de l'application avec une description de chacune d'elles.

Chapitre I : Contexte général du projet

1.1 Introduction

Cette première partie du rapport donne une vision globale du projet, sa problématique ainsi que son cadre général.

1.2 Présentation du lieu de stage

ID Formation est une Société Coopérative Ouvrière de Production (SCOP) présente à l'échelle nationale, avec plus de 117 centres de formation. Fondée en 1994, sa mission principale est de d'accompagner les personnes dans leur projet de transition professionnelle et d'insertion. En tant qu'Organisme de formation certifié QUALIOPI, Centre de bilan de compétences et Centre de formation d'apprentis, ID Formation s'engage également en faveur de l'intégration des personnes en situation de handicap.

Les valeurs d'ID Formation sont :

- Professionnelle Accessible à Tous : Avec une présence nationale et 213 centres de formation, nous proposons une variété de formations professionnelles pour soutenir votre évolution et votre succès. Que vous ayez besoin d'un bilan de compétences, d'une réorientation professionnelle ou de conseils pour évoluer dans votre carrière, nous couvrons un large éventail de besoins pour aider chaque individu à atteindre ses objectifs.
- Un Engagement à l'Échelle Nationale et Régionale : Implantés dans plusieurs régions françaises, notamment les Hauts-de-France, le Grand Est, le Centre-Val de Loire, l'Île-de-France, l'Auvergne-Rhône-Alpes, la Provence-Alpes-Côte d'Azur et la Normandie, nous mettons notre expertise à la disposition des salariés et des demandeurs d'emploi. Notre objectif est de les aider à réussir leurs projets professionnels en leur offrant un soutien adapté à leurs besoins spécifiques.
- Impact et Réussite : En 2022, nous avons dispensé plus de 4 millions d'heures de formation, accueilli près de 125 000 stagiaires, et avons enregistré un taux de réussite aux examens de 74%, avec une satisfaction globale des stagiaires atteignant 93%. Notre équipe de 807 salariés se mobilise chaque jour pour votre réussite. Notre siège social est basé dans la région Hauts-de-France, avec des centres de formation répartis sur tout le territoire.

Site web

http://www.id-formation.com

Secteur

Formation professionnelle et coaching

Taille de l'entreprise

501-1 000 employés

1 220 sur LinkedIn Inclut des membres avec ID FORMATION comme employeur actuel, y compris les postes à temps partiel.

Siège social

Lille, Nord-Pas-de-Calais

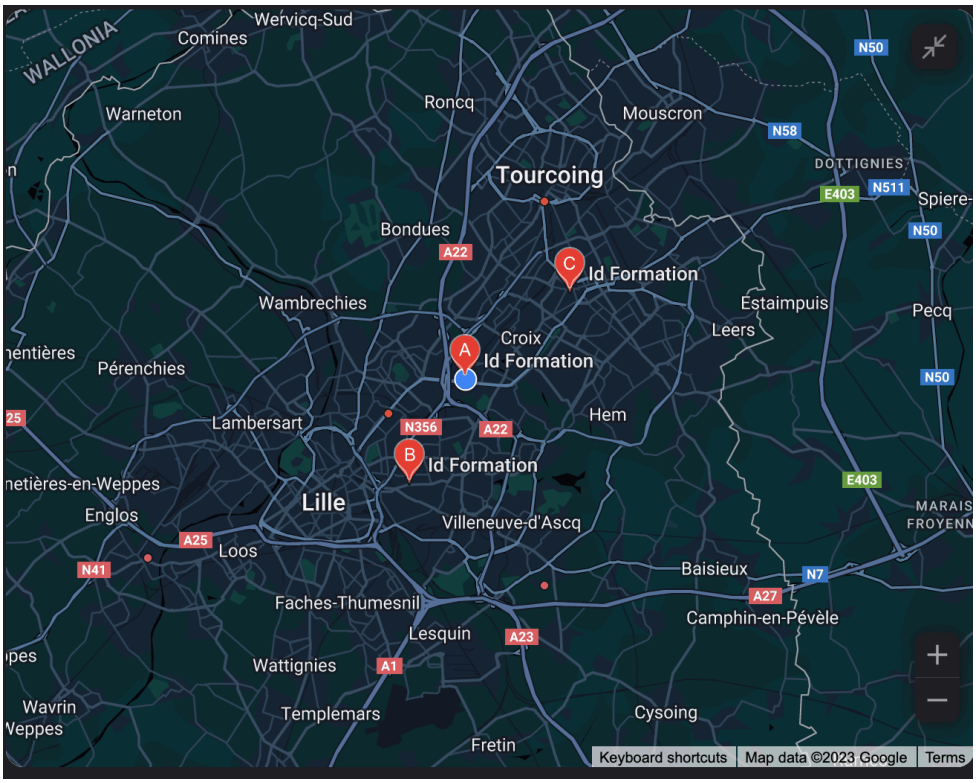

Figure 1 : Adresse ID formation

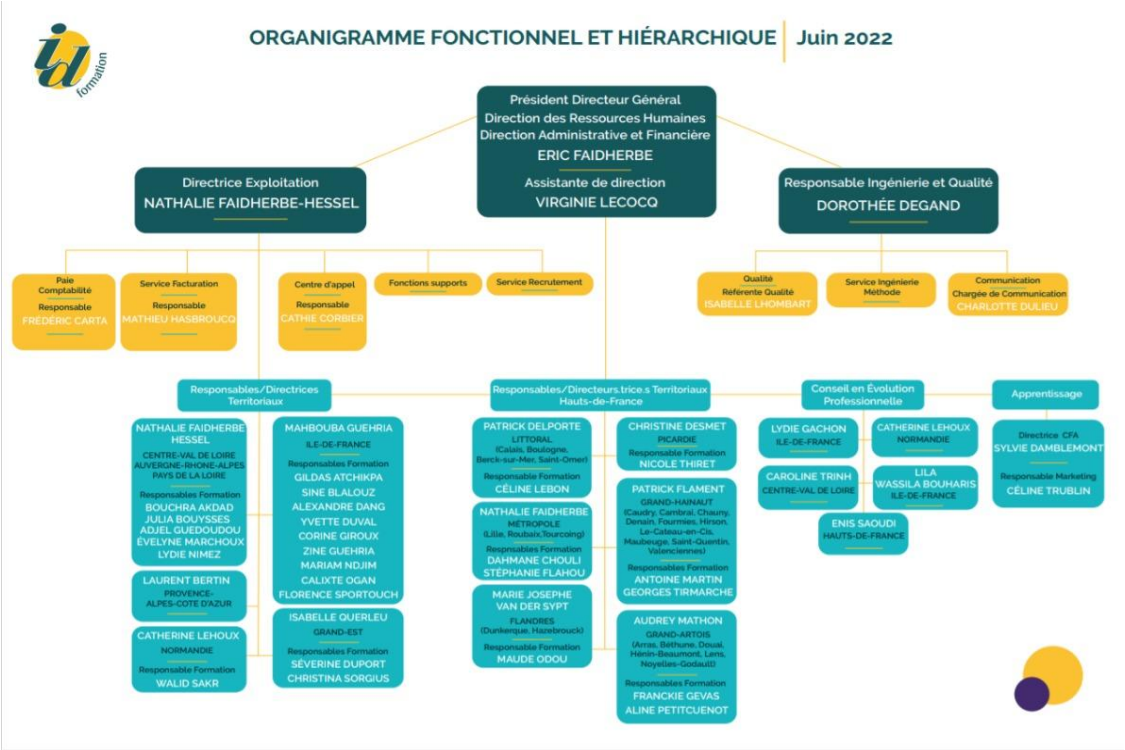

Figure 2: Organigramme ID Formation

1.3 Description du projet

1.3.1 Etude de l'existant

L'entreprise utilise actuellement un logiciel interne pour la gestion de la comptabilité, des clients, de la facturation, etc.

Architecture de l'application

L'architecture de l'application de gestion est divisée en deux parties.

- **Front-end :** Cette partie est développée en utilisant React.js avec le langage TypeScript. Le front-end est l'interface utilisateur avec laquelle les utilisateurs interagissent. Il offre une expérience interactive et dynamique aux utilisateurs finaux.
- **Back-end :** Le back-end alimente le front-end en fournissant plusieurs API REST sécurisées. Techniquement, il est développé en Java avec Spring Boot (version 3.1.2 au moment de la rédaction de ce document). Le back-end gère les requêtes provenant du front-end, traite les données et les renvoie au format approprié pour être affichées à l'utilisateur.

Les données de l'application sont stockées dans une base de données MySQL, hébergée sur un ordinateur de l'entreprise. Cette base de données agit comme un référentiel central pour toutes les informations nécessaires au bon fonctionnement de l'application.

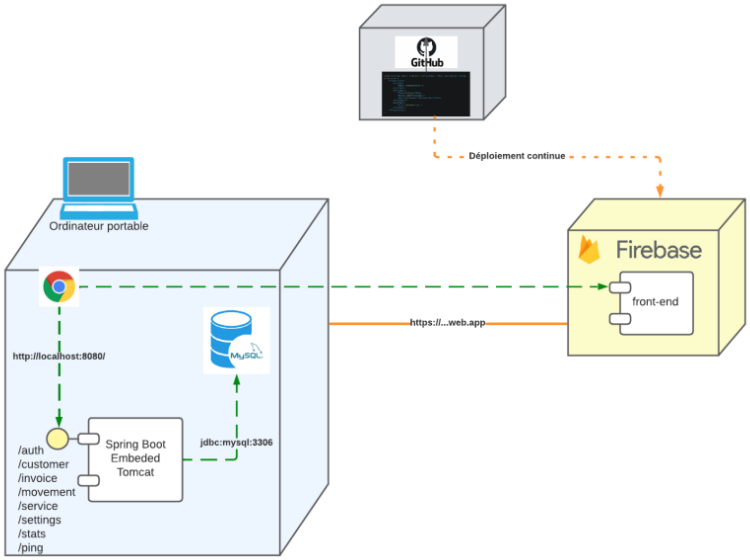

Figure 3: Architecture de l'application

Fonctionnalités de l'application

En ce qui concerne les fonctionnalités de l'application, nous avons :

- **Authentification** : Un système d'authentification sécurisé pour garantir l'accès approprié aux utilisateurs autorisés.
- **Tableau de bord** : Affiche les principaux indicateurs clés de l'activité de l'entreprise sur l'année, tels que le nombre de clients actifs, le nombre de nouveaux clients, le chiffre d'affaires annuel, le chiffre d'affaires moyen et le bénéfice net mensuel moyen.
- **Page Clients** : Présente les informations relatives aux clients de manière structurée et tabulaire pour une gestion efficace.
- **Offres de Services** : Une page dédiée qui présente les différentes offres de services proposées par l'entreprise.
- **Gestion des Factures** : Affiche toutes les factures émises, permettant un suivi précis des transactions financières.
- **Calendrier** : Une fonctionnalité similaire à Google Calendar, permettant de planifier et de suivre les événements et les rendez-vous importants.

1.3.2 Cahier charge et évolution

Bien que la solution actuelle ait été efficace jusqu'à présent, diverses évolutions et améliorations sont désormais nécessaires en raison du contexte de l'entreprise.

Nous avons identifié les fonctionnalités suivantes à développer :

Gestion de la TVA

L'application doit permettre de comptabiliser le montant de TVA à déclarer entre deux déclarations et afficher ce montant sur le tableau de bord.

Seules les factures payées sont concernées et c'est la date à laquelle la facture a été effectivement payée qui doit être prise en compte.

Emission d'une facture

L'application doit permettre d'envoyer un mail au client suite à l'émission d'une facture et de la création du PDF contenant la facture.

La mail envoyer doit je souhaite que le client mail s'ouvre automatiquement avec l' email pré-saisi suivant:

- Le destinataire est l'email de contact attaché au client de la facture
- Le sujet est « facturation <mois> <année> »
- Le contenu est :

  « Bonjour,

  Veuillez trouver ci-joint la facture pour mon activité de ce mois ainsi que l'attestation habituelle.

  Cordialement »

  Si une facture est en retard de paiement (facture non payée et échéance passée), une nouvelle option doit apparaître dans le menu présent pour chaque facture. Celle-ci doit ouvrir le client mail avec l'email pré-saisi suivant:

- Le destinataire est l'email de contact attaché au client de la facture
- Le sujet est « retard de paiement facture <num> », où <num> est le numéro de la facture

Emission d'une facture impayée

L'application doit permettre d'envoyer un mail au client si une facture est en retard de paiement (facture non payée et échéance passée).

• Le contenu est :

« Bonjour,

Je n'ai, à ce jour, pas reçu le règlement pour la facture <num> émise le <date d'émission de la facture> . Pouvez-vous vérifier que le traitement de cette facture a été correctement effectué de votre côté et, dans le cas contraire procéder sans délai au versement correspondant ?

Cordialement »

Suppression d'une facture

L'application doit permettre la suppression (avec pop-up de confirmation) dans le menu présent pour chaque facture.

L'action est disponible que pour les factures en mode « brouillon », c'est à dire celles qui n'ont pas encore été émises.

1.4 Gestion du projet

L'application Gestion micro-entreprise est réalisé dans le cadre de projet fin de formation au sein de IdFormation.

Le but est de concevoir et développer une application d'une micro-entreprise Pour ceci, nous avons pu mettre en place des bonnes pratiques rencontrées au cours de la formation.

1.4.1 AGILE/SCRUM et Trello

Étant donné que notre période de stage était limitée à un mois, nous avions besoin de flexibilité, ce que l'approche AGILE/SCRUM nous a offert.

Il s'agit d'un cadre de gestion de projet qui segmente les projets en plusieurs phases dynamiques appelées "sprints".

À la fin de chaque sprint, nous discutions des axes d'amélioration, des éléments bloquants et des facteurs ralentisseurs ("parachute") à considérer pour le sprint suivant. Nous tenions également des "dailies" – de courtes réunions nous permettant de faire le point sur les tâches réalisées et celles prévues pour la journée. De plus, en cas de problème, nous n'attendions pas la fin du sprint pour le signaler. Ce processus favorisait une compréhension mutuelle du projet et un partage efficace des informations.

Les sprints constituaient aussi des moments propices pour revoir les tickets ou en créer de nouveaux. Ces tickets représentaient des demandes à traiter, segmentant ainsi le projet en plusieurs sous-parties. Nous utilisions Trello pour les gérer.

Chaque ticket est structuré en trois parties : la demande initiale (celle du client), la description et les "besoins" ou solutions potentielles pour sa mise en œuvre.

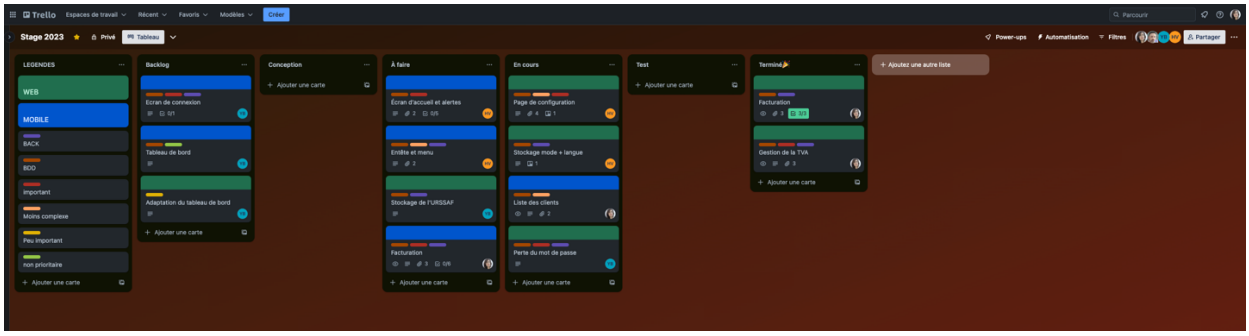

Figure 4: Tableau Trello des taches

Le ticket est décomposé en trois parties : la demande initiale (celle du client), la description et les "besoins" ou solutions potentielles pour réaliser le ticket.

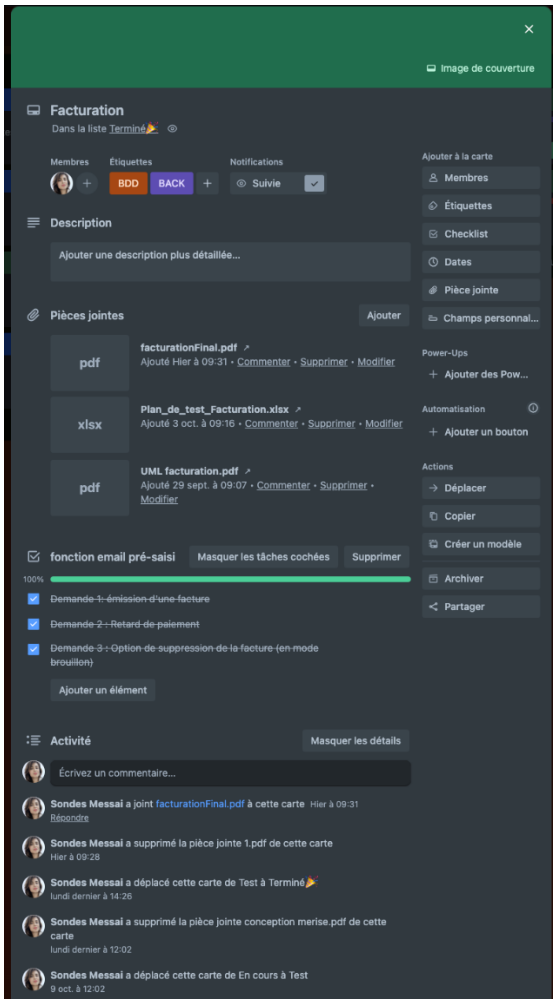

Figure 5: Exemple de tache effectué

1.4.2 GIT

Nous avions établi une organisation précise pour Git, notamment en ce qui concerne la nomenclature des branches. Une branche était dédiée à chaque ticket sur lequel nous travaillions.

Lorsque je développais une fonctionnalité, je fusionnais la branche associée avec une branche de "pré-production". Plusieurs branches "pré-production" étaient présentes, chacune correspondant au numéro du sprint en cours.

Finalement, nous disposions d'une branche "production" (la branche principale) qui accueillait les fusions jugées "finales". En d'autres mots, une fois que toutes les fusions d'un sprint étaient opérationnelles et compatibles, je les intégrais à cette branche principale

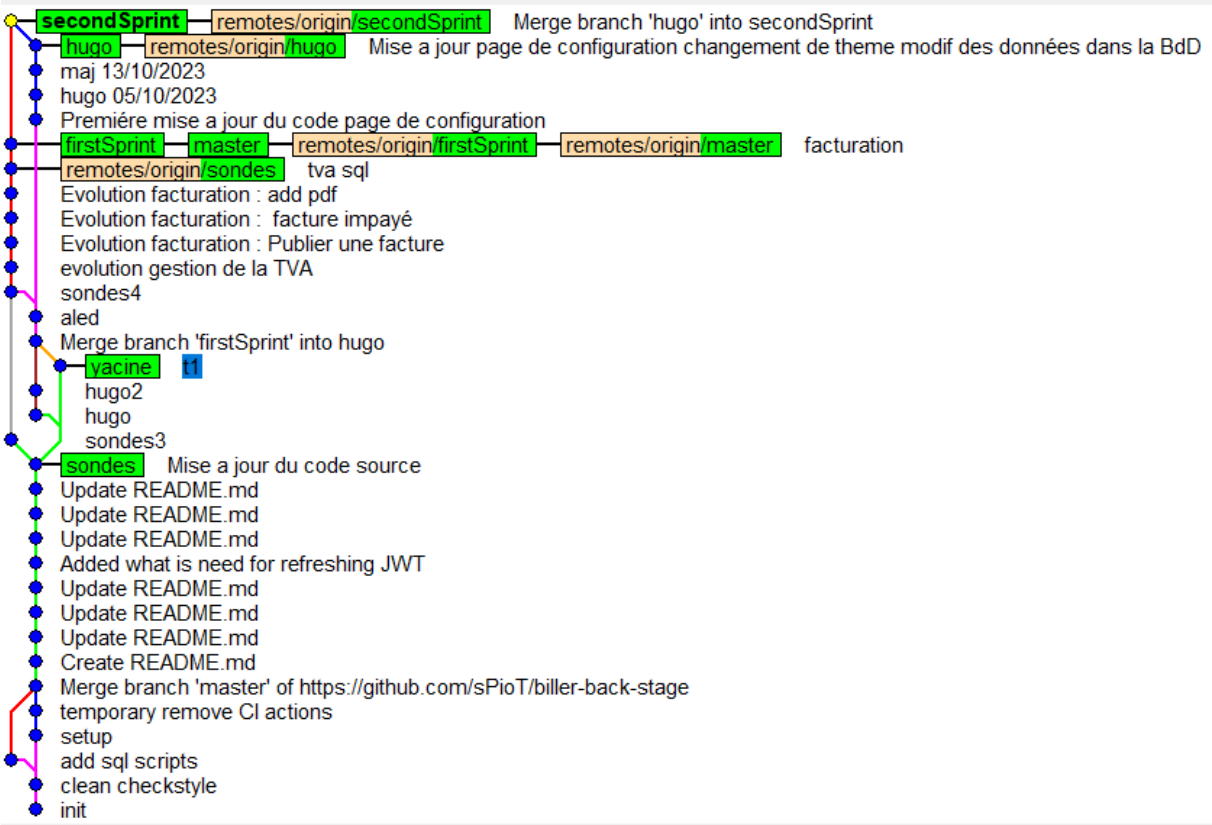

Figure 6: Exemple de commit git

Chapitre II : Modélisation et Conception

1. Introduction

La conception permet de décrire de manière non ambiguë, le plus souvent en utilisant un langage de modélisation, le fonctionnement futur du système, afin d'en faciliter la réalisation. Méthodes de conception et de modélisation utilisées :

Merise [Méthode d'Etude et de Réalisation Informatique pour les Systèmes d'Entreprise]

UML [Langage de Modélisation Unifié]

Dans ce chapitre nous allons se baser sur une analyse conceptuelle des besoins ; commençons par une étude conceptuelle et logique des données en utilisant le model conceptuelle de données (MCD) et le model logique de données (MRD ou MLD). Ensuite nous réaliserons une modélisation des traitements.

En effet la réalisation d'une application puissante n'est qu'un fruit d'une bonne démarche, bonne étude et une conception adéquate, ceux-ci s'avèrent donc précieux et primordiale dans le cycle de vie de notre application.

Pour terminer, c'est dans ce chapitre que s'appliquent les techniques de modélisation.

1.2 Analyse des besoins

L'objectif principal d'un système logiciel est de rendre service à ses utilisateurs,donc il faut bien comprendre leurs désirs et besoins.

1.2.1 Besoins fonctionnels

Les besoins fonctionnels sont définis comme étant ; les services attendus par l'utilisateur de produit, et qu'ils doivent accomplir un ensemble complet et cohérent des buts.

Les besoins fonctionnels demandés sont :

☑ ***Gestion de la TVA :*** Afficher le cumul des tva sur le mois encours.

☑ ***Gestion des factures :***

- Emission d'une facture
- Emission d'une facture impayée
- Suppression d'une facture.

1.2.2 Besoins non fonctionnels

Les besoins non fonctionnels se basent sur le respect des normes ergonomiques etsur les interactions homme/machine qu'on fournit à l'application. Notre application doit nécessairement assurer ces besoins :

☑ ***L'ergonomie :*** aboutir une interface caractérisée par une perception attirante et professionnelle.

☑ ***La fiabilité :*** exécution correcte et optimiser de tous les fonctions du système.

☑ ***La rapidité de traitement :*** le système doit garantir la rapidité d'exécution des traitements.

☑ ***L'intégrité :*** les fonctionnalités offertes à chaque utilisateur doivent être restreintes à celles qui lui sont autorisées. L'information n'est modifiée que parles personnes y ayants droit. Dans ce cadre, nous définissons pour chaque utilisateur ces droits d'accès au système.

☑ ***La portabilité :*** minimiser l'effort de la configuration de l'application avecun nouveau serveur.

☑ ***L'extensibilité :*** l'application devra être extensible, c'est-à-dire qu'il pourray avoir une possibilité d'ajouter ou de modifier de nouvelles fonctionnalités.

☑ ***La sécurité*** : le système doit traiter les failles de sécurité d'où le besoin d'un login et d'un mot de passe pour accéder au système. Les messages d'erreurs doivent identifier tous les cas d'erreurs de saisie et leur source. En plus de ça, la récupération de mot de passe s'il y a un problème.

1.3 Modélisation des traitements avec UML

1.3.1 Définition

Le langage de modélisation unifié, de l'anglais Unified Modeling Language (UML), est un langage de modélisation graphique à base de pictogrammes conçu pour fournir une méthode normalisée pour visualiser la conception d'un système. Il est couramment utilisé en développement logiciel et en conception orientée objet.

UML est à présent un standard défini par l'Object Management Group (OMG). De très nombreuses entreprises de renom ont adopté UML et participent encore aujourd'hui à son développement.

1.3.2 Diagramme de classe

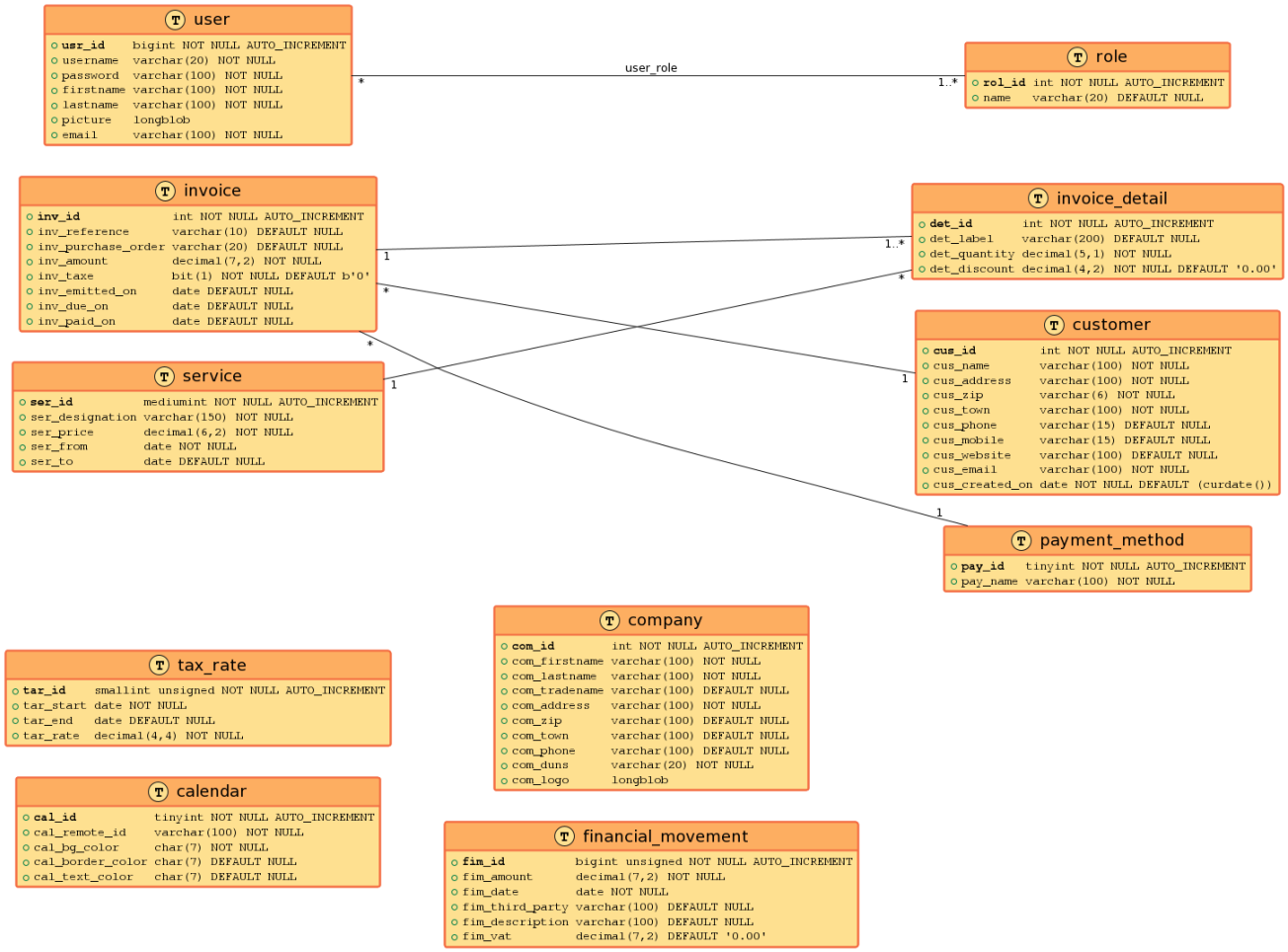

Figure 7: Diagramme de classe

1.3.3 Diagrammes de cas d'utilisation

Les diagrammes de cas d'utilisation sont des diagrammes UML utilisés pourdonner une vision globale sur le comportement fonctionnel de l'application.

Gestion de factures

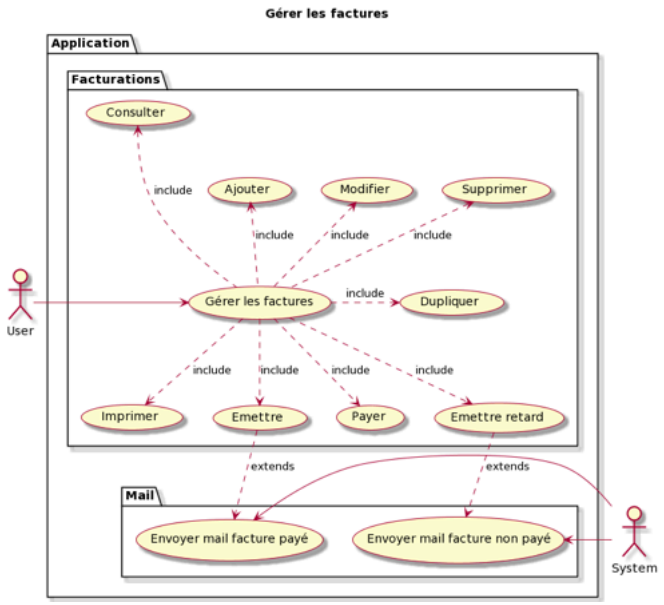

Figure 8: Diagramme de cas d'utilisation factures

Gestion TVA

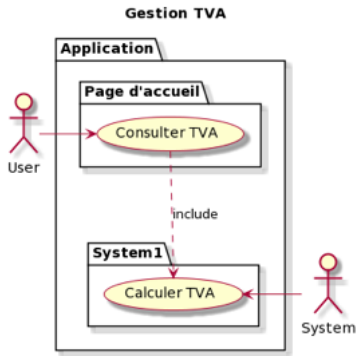

Figure 9: Diagramme de cas d'utilisation TVA

1.3.4 Diagrammes de séquences

Gestion de factures

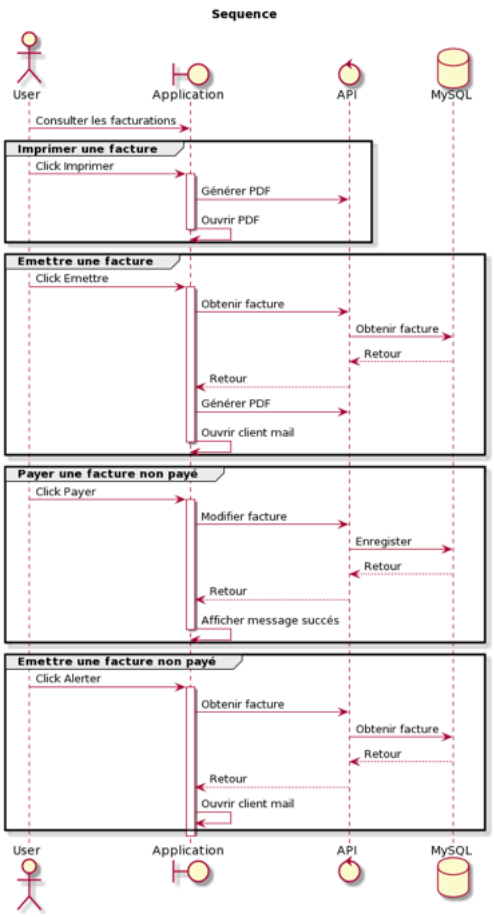

Figure 10: Diagramme de cas séquence factures

Gestion TVA

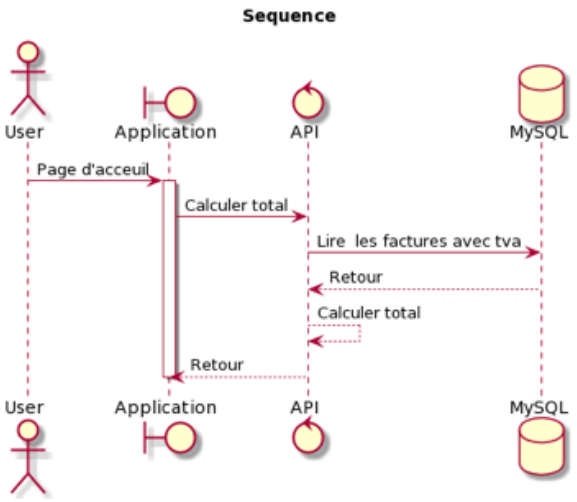

Figure 11: Diagramme de cas séquence TVA

1.4 Maquettage

Le maquettage d'une application est le processus de création de schémas visuels rudimentaires qui représentent l'interface utilisateur d'une application, y compris la disposition des éléments, la navigation et l'apparence générale.

Il sert à planifier la structure et le design avant le développement, permettant aux concepteurs et aux développeurs de mieux comprendre les besoins et les fonctionnalités de l'application. Les maquettes sont généralement des prototypes non fonctionnels qui aident à visualiser et à valider les idées avant la création de la version finale de l'application.

1.4.1 Gestion de factures

ZONING

Liste des factures :

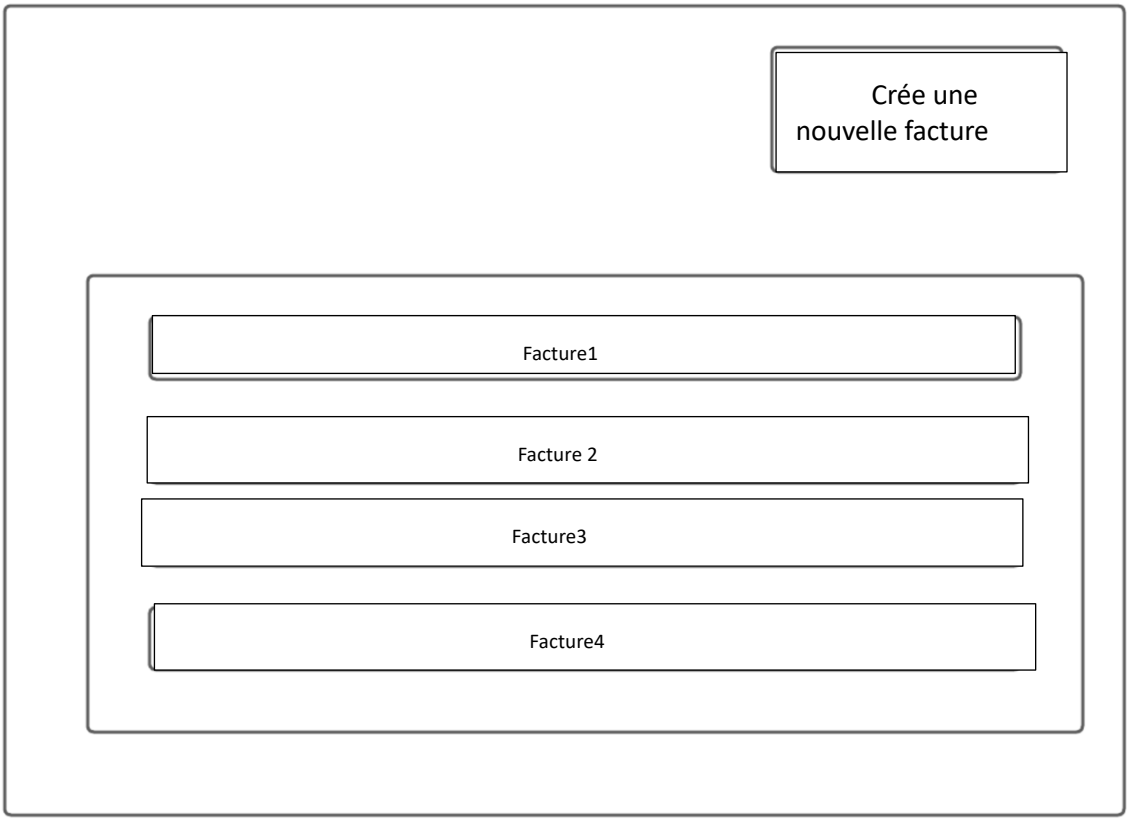

Figure 12: Maquette liste de facture

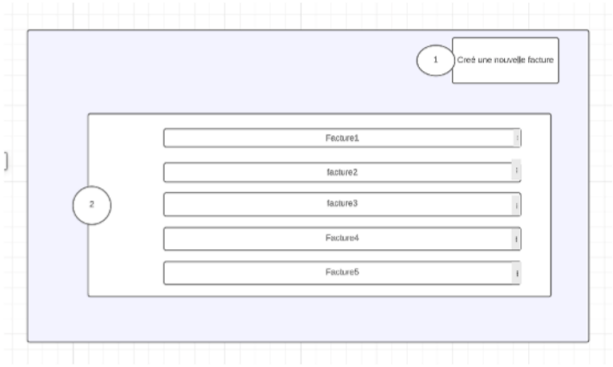

Figure 13: Maquette liste de facture

#	Description	Action	R.G
1	Bouton créer une nouvelle facture	Afficher l'écran créer une facture	
2	Liste des factures déjà crée	Liste d'affichage des factures sont déjà crée	

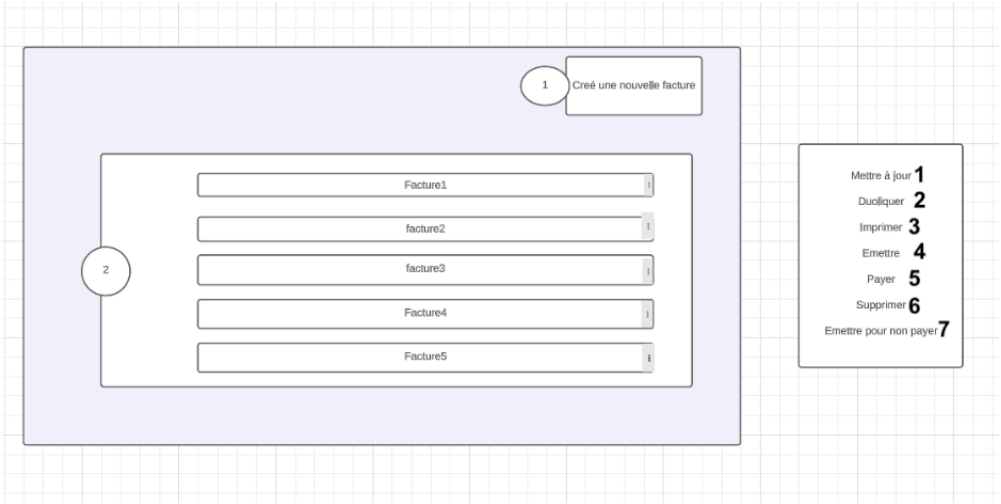

Figure 14: Maquette liste de facture

#	Description	Action	R.G
1	Consulter	Afficher l'écran consulter	RG1
2	Mettre à jour	Afficher l'écran modifier	RG2
3	Dupliquer	Dupliquer la facture	RG3
4	Imprimer	Afficher le bon de commande	RG4
5	Emettre	Envoyer la facture	RG5
6	Payer	Marquer la facture comme payer	RG6
7	Emettre retard	Envoyer alerte retard facture	RG7
8	Supprimer	Supprimer la facture	RG8
9			
10			

➢ **Voici les règles de gestion possibles pour les fonctionnalités de la facturation :**

#	
RG1	Les factures peuvent être consulter, si elles sont dans un état « Emise ou Payé ou non Payé ».
RG2	Les factures peuvent être modifier, si elles sont dans un état « Brouillon ».
RG3	Les factures peuvent être dupliquer, si elles sont dans un état « Brouillon ou Emise ou Payé ou non Payé ».
RG4	Les factures peuvent être imprimer, si elles sont dans un état « Brouillon ou Emise ou Payé ou non Payé ».
RG5	Les factures peuvent être émise, si elles sont dans un état « Brouillon ».
RG6	Les factures peuvent être payer, si elles sont dans un état « Emise ».
RG7	Les factures peuvent être non payé, si elles sont dans un état « Emise ».
RG8	Les factures peuvent être supprimées, mais uniquement si elles sont dans un état « brouillon ».

Ajouter/Modifier facture :

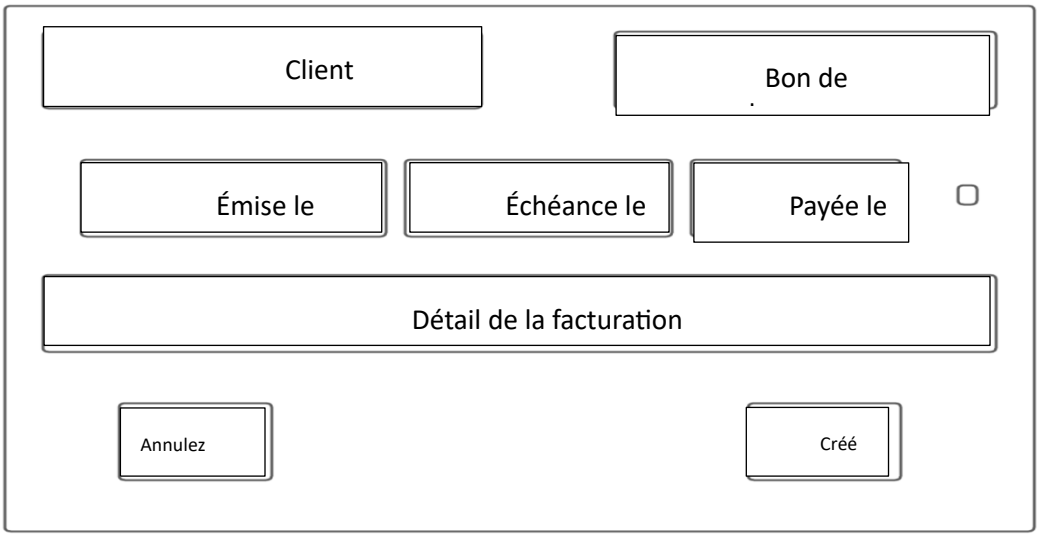

Figure 15: Maquette détail/ajouter/modifier facture

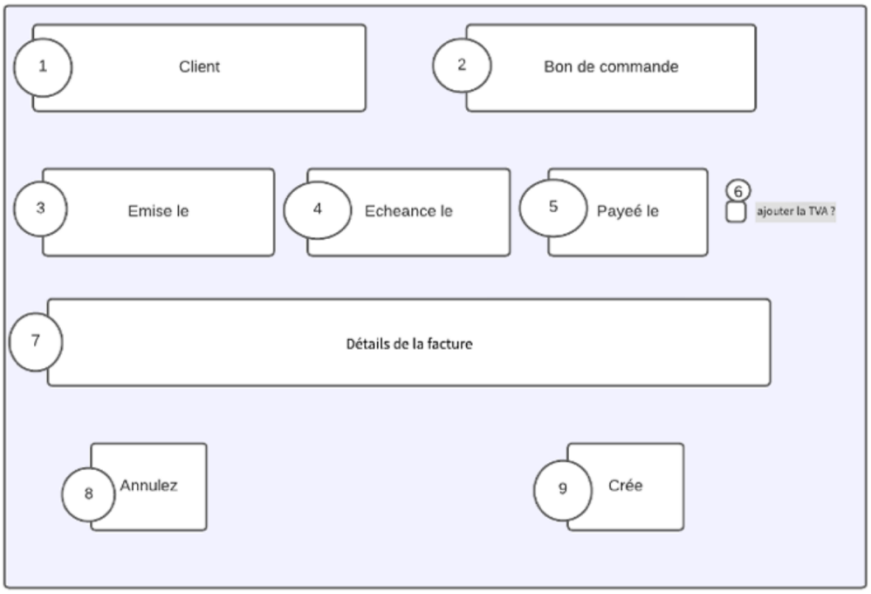

Figure 16: Maquette détail/ajouter/modifier facture

#	Description	Action	R.G
1	Liste des clients	Une liste déroulante des client	RG1
2	Bon de commande	Bon de commande	RG2
3	Date d'émise	Saisir la data d'Emission	RG3
4	Date d'échéance	Saisir Echéance le	RG4
5	Date a été payée	Saisir Payé le	RG5
6	Le check de la TVA	Sélectionner si TVA appliqué	N/A

Détail d'une facture

#	Description	Action	R.G
1	Liste des offres	Choisir une Offre	RG6
2	Description	Saisir Description	N/A
3	Prix unitaire	Saisir Prix unitaire	RG7
4	Quantité	Saisir Quantité	RG8
5	Remise	Saisir Remise	N/A
6	Total HT	Saisir Total HT	RG9
7	Total TTC	Saisir Total TTC	RG10
8	Bouton -	Supprimer ligne	RG11

#	Description	Action	R.G
1	Total HT	Calculer total HT	RG9, RG12
2	Total TTC	Calculer total TTC	RG10, RG13
3	Bouton +	Ajouter une ligne détail	RG14
4	Bouton Valider	Confirmer la création d'une nouvelle facture	
5	Bouton retour	Annuler la facture	
6			

➢ **Voici les règles de gestion possibles pour les fonctionnalités de la facturation :**

RG1	Le client concerné doit être sélectionné à partir d'une liste de clients existants.
RG2	Un bon de commande client peut être associé à la facture, s'il existe.
RG3	La date d'émission de la facture doit être renseignée automatiquement à la création de la facture.
RG4	Une échéance de paiement doit être définie pour chaque facture.
RG5	La date réelle de paiement doit être mise à jour lorsque la facture est payée.
RG6	L'offre concerné doit être sélectionné à partir d'une liste de offres existants.
RG7	Le prix unitaire est obligatoire
RG8	La quantité est obligatoire
RG9	Afficher si « L'indicateur TVA » n'est pas sélectionné.
RG10	Afficher si « L'indicateur TVA » est sélectionné.
RG11	Je peux supprimer une ligne détail sauf la 1ere ligne.
RG12	Est égale à la somme des montant total HT des lignes détails.
RG13	Est égale à la somme des montant total TTC des lignes détails.
RG14	Je peux ajouter une ligne détail

Emettre une facture

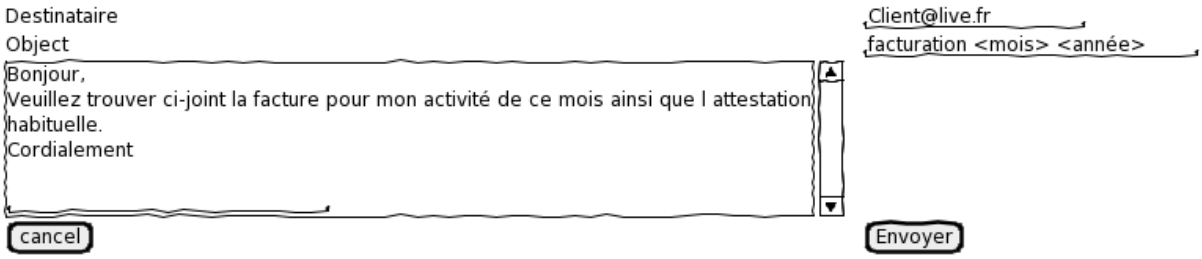

Figure 17: Maquette émettre facture

#	Description	Action	R.G
1	Destinataire	Afficher le mail client	RG1
2	Object	Afficher « facturation » + mois + année	RG2
3	Body	Afficher le mail	RG3
4	Pièce jointe	Inclure mon bon de commande	RG4
5	Bouton retour	Annuler l'envoie	N/A
6	Bouton valider	Envoyer le mail	N/A
			N/A

➢ **Voici les règles de gestion possibles pour les fonctionnalités de la facturation :**

RG1	Le destinataire est l'email de contact attaché au client de la facture
RG2	Le sujet est « facturation <mois> <année> »
RG3	Le contenu est :

	« Bonjour, Veuillez trouver ci-joint la facture pour mon activité de ce mois ainsi que l'attestation habituelle. Cordialement »
RG4	La pièce jointe est le pdf généré

Emettre une facture impayée

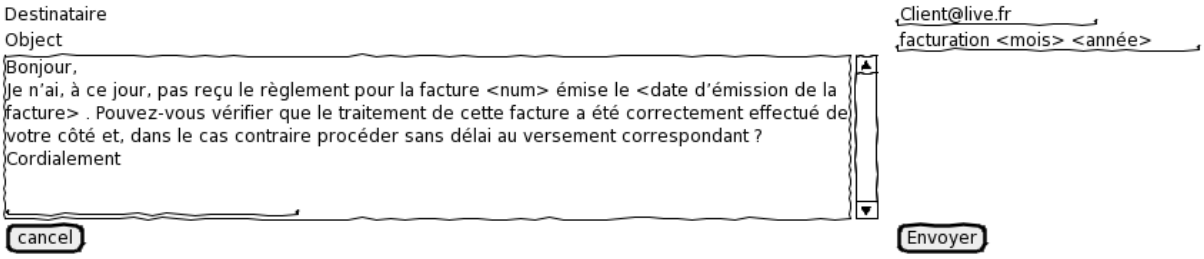

Figure 18: Maquette émettre facture

#	Description	Action	R.G
1	Destinataire	Afficher le mail client	RG1
2	Object	Afficher « facturation » + mois + année	RG2
3	Body	Afficher le mail	RG3
4	Bouton retour	Annuler l'envoie	N/A
5	Bouton valider	Envoyer le mail	N/A
6			N/A

➢ **Voici les règles de gestion possibles pour les fonctionnalités de la facturation :**

RG1	Le destinataire est l'email de contact attaché au client de la facture
RG2	Le sujet est « retard de paiement facture <num> », où <num> est le numéro de la facture
RG3	Le contenu est : « Bonjour, Je n'ai, à ce jour, pas reçu le règlement pour la facture <num> émise le <date d'émission de la facture>. Pouvez-vous vérifier que le traitement de cette facture a été correctement effectué de votre côté et, dans le cas contraire procéder sans délai au versement correspondant ? Cordialement »
RG4	

WIREFRAM

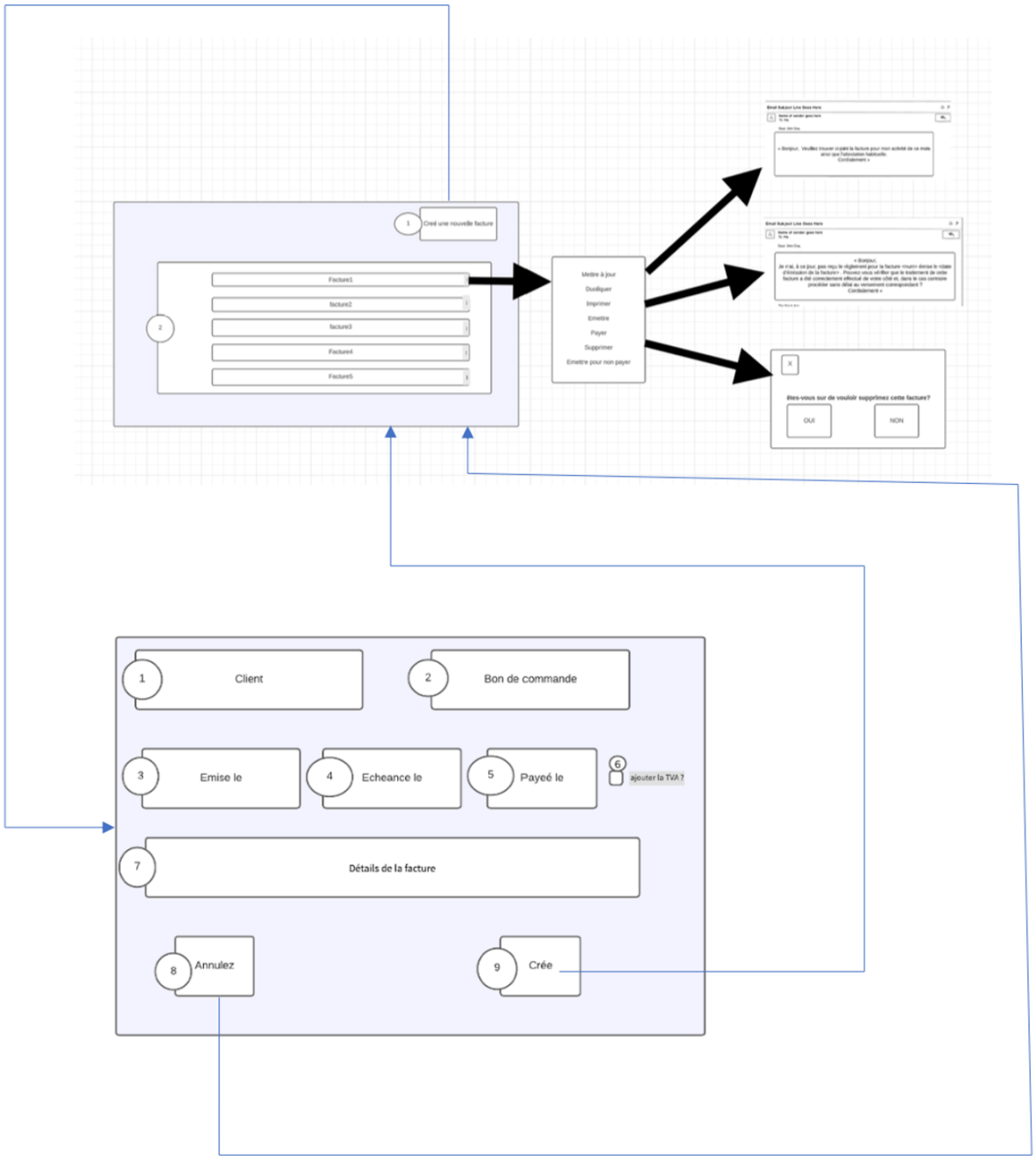

Figure 19: Maquette facture

1.4.2 Gestion de TVA

ZONING

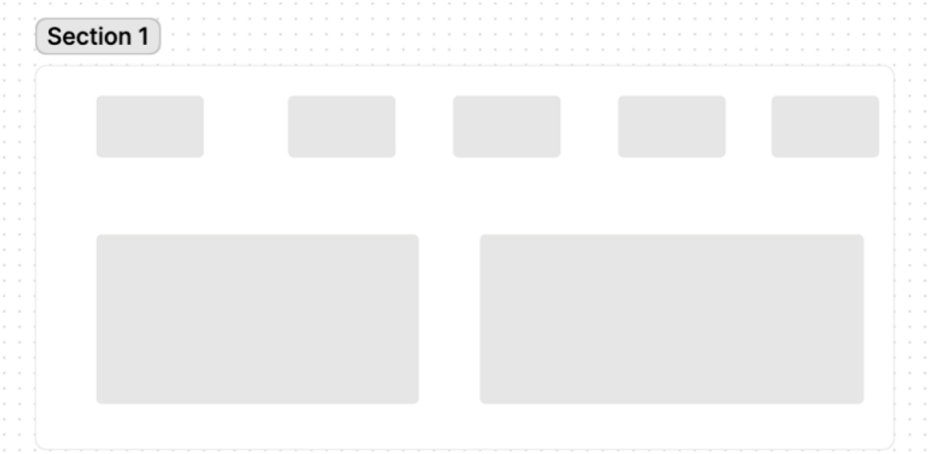

Figure 20: Maquette dashboard

WIREFRAM

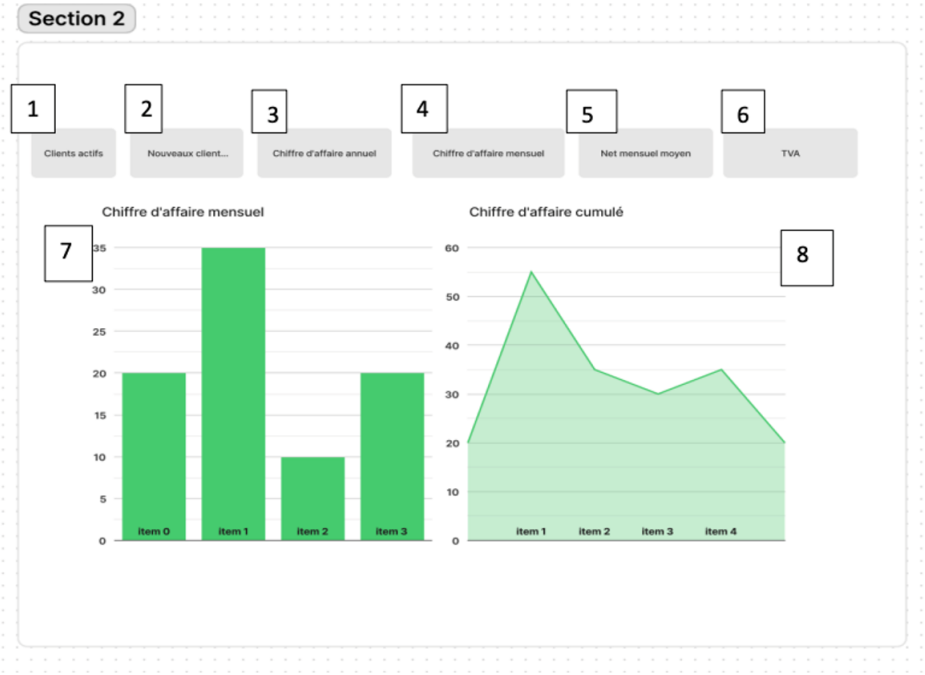

Figure 21: Maquette dashboard

#	Description	Action	R.G
1	Clients active	Les nombre de client active	
2	Nouveaux clients	Nombre de clients récemment ajoutés	
3	Chiffre d'affaires annuel	Somme des factures payer annuel en €	
4	Chiffre d'affaires mensuel	Somme des factures payées ce mois-ci	
5	Net mensuel moyen	Net mensuel moyen en €	

6	TVA	Tva en €	RG1
7	Chiffre d'affaires mensuel	Le chiffre d'affaires est le total des ventes ou des recettes générées par une entreprise sur une période donnée, généralement mensuelle.	
8	Chiffre d'affaires Cumulé	Le chiffre d'affaires cumulé représente la somme totale des ventes ou des recettes d'une entreprise sur une période donnée, en tenant compte de l'accumulation sur plusieurs mois ou années.	

➢ **Voici les règles de gestion possibles pour les fonctionnalités de la tableaux bord :**

#	
RG1	Calculer le total TVA pour le mois en cours de déclaration à partir du champ **inv_amount** de table **invoice** ou l'indication **inv_taxe** est à true.

Chapitre III : Réalisation de l'application

1. Introduction

Dans ce chapitre nous allons présenter le détail de l'implémentation, commençant par une présentation de différentes technologies utilisées dans le développement, et terminant par une présentation de quelques interfaces de l'application réalisée.

1.2 Environnement du travail

Dans cette partie, nous présentons l'environnement de la réalisation. Nous présentons ainsi les caractéristiques des machines sur lesquelles nous avons travaillé ainsi que les logiciels utilisés pour la réalisation.

1.2.1 Environnement matériel

Pour la réalisation de notre projet nous avons à notre disposition le matériel suivant :

Un PC portable MacBook Air 13-inch équipé d'un processeur M1, d'un disque dur de 256 GB et de 8,00 Go de mémoire vive.

1.2.2 Environnement logiciel

Nous présentons dans cette partie tous les logiciels utilisés au cours du développement de notre site.

Système d'exploitation : (MacOs Monterey) Ce système apporte à la fois la puissance et la sécurité nécessaire, avec une interface graphique qui offre, grâce à ses nombreuses fonctionnalités, un environnement solide et ergonomique.

1.2.3 Environnement de développement

Nous présentons dans cette partie tous les logiciels utilisés au cours du développement de notre application.

Visual Studio Code

C'est un éditeur de code Extensible développé par Microsoft Pour Windows, Linux Et macOS.

Les fonctionnalités incluent la prise en charge du débogage, la mise en évidence de la syntaxe, la complétion intelligente du code, les snippets, la refactorisation Du code et Git Intégré. Les utilisateurs peuvent modifier le thème, les raccourcis clavier, les préférences et installer des extensions Qui ajoutent des fonctionnalités supplémentaires.

Dans le sondage auprès des développeurs réalisé par Stack Overflow En 2021, Visual Studio Code a été classé comme l'outil d'environnement de développement Le plus populaire, avec 71,06 %. Et aussi

Eclipse

C’est un IDE open source populaire, principalement utilisé pour le développement Java, mais extensible pour d'autres langages grâce à des plugins. Il offre une interface utilisateur personnalisable, une vaste communauté de développeurs et des outils intégrés pour le développement, le débogage et la gestion de projets logiciels.

Système de gestion de bases de données (DBever)

Afin de stocker les données de l’application j’ai choisi d’utiliser DBever principalement, car j’ai l’habitude d’utiliser ce serveur de base de données.

1.2.4 Langage de programmation

CSS (Cascading Style Sheets)

C’est un langage informatique qui décrit la présentation des documents HTML. L’extension est « nom-de-la-page.css ».

Type Script

C’est un langage de programmation open source développé par Microsoft. Il sert d'extension à JavaScript, ajoutant des fonctionnalités de typage statique au langage. Les développeurs peuvent spécifier des types de données pour les variables, les paramètres de fonction, les propriétés d'objet, etc. Cela facilite la détection des erreurs de typage pendant la phase de développement.

TypeScript se compile en JavaScript standard pour être exécuté dans un navigateur web ou sur un serveur

Java

Java est un langage de programmation polyvalent et orienté objet, reconnu pour sa portabilité grâce à la machine virtuelle Java (JVM). Il offre une sécurité renforcée, une gestion automatique de la mémoire, prend en charge le multithreading et propose une vaste bibliothèque standard. Utilisé dans le développement web, mobile (notamment pour Android), les applications de bureau et les solutions d'entreprise, Java demeure un choix populaire en raison de sa stabilité et de son adoption étendue au sein de la communauté de développement.

1.2.5 Framework

Un Framework est, comme son nom l’indique en anglais, un “cadre de travail“. L’objectif d’un Framework est généralement de simplifier le travail des développeurs informatiques, en leur offrant une architecture “prête à l’emploi”.

React

C’est une bibliothèque frontale open source, basée sur des composants et gérant exclusivement la couche d'affichage de l'application, a été développée par Facebook. Cette bibliothèque utilise un mécanisme virtuel basé sur le DOM pour insérer des données (vues) dans le DOM HTML. Le DOM virtuel fonctionne rapidement car il ne modifie que les éléments DOM individuels, évitant ainsi de recharger l'intégralité du DOM à chaque fois

React Native

C’est un un framework open source créée par Facebook, permet aux développeurs de concevoir des applications mobiles pour Android et iOS en utilisant JavaScript et React, un framework JavaScript populaire.

Grâce à React Native, une grande partie du code peut être partagée entre les deux plateformes, réduisant ainsi le temps de développement et les coûts. De plus, il offre un accès direct aux composants natifs, offrant des performances proches de celles des applications natives. Il est largement adopté pour le développement d'applications multiplateformes, offrant ainsi une solution efficace et rapide pour les développeurs.

Spring Boot

C'est un framework Java open source,qui a été conçu pour simplifier le développement d'applications Java. Il offre une approche rapide et efficace en réduisant la configuration et en fournissant des dépendances préconfigurées. Intégré au framework Spring, il facilite le développement d'applications web, de microservices et de solutions d'entreprise.

Spring Boot améliore la productivité des développeurs grâce à ses fonctionnalités prêtes à l'emploi tout en garantissant la robustesse et les performances des applications Java. Il demeure un choix populaire dans le domaine du développement logiciel Java.

MUI Core

Material-UI Core, également connu sous le nom de Material-UI Core, est une bibliothèque open source de composants d'interface utilisateur (UI) développée en utilisant React, c'est un framework JavaScript qui propose une variété de composants prêts à l'emploi, suivant les principes du design Material Design de Google.

MUI Core simplifie la création d'applications web modernes en offrant une esthétique cohérente et une expérience utilisateur agréable. En raison de sa flexibilité et de sa facilité d'intégration avec React, elle est largement utilisée dans le développement web. La bibliothèque est constamment mise à jour pour suivre les dernières tendances en matière de design et de développement web.

Jwt (JSON Web Token)

C'est un format de token sécurisé utilisé pour l'authentification et l'autorisation dans les applications web et mobiles. Il se compose de trois parties encodées en JSON (en-tête, charge utile, signature) et est signé numériquement pour garantir l'intégrité des données, permettant ainsi l'échange sécurisé d'informations d'identification et d'autorisation entre le client et le serveur.

Checkstyle :

C'est un outil d'analyse statique de code en Java utilisé pour vérifier si le code source respecte un ensemble de conventions de style et de règles de codage prédéfinies. Pour l'utiliser, vous devez configurer un fichier XML avec les règles de codage souhaitées, puis exécuter Checkstyle sur votre code pour détecter les violations. Une fois les violations identifiées, vous pouvez les corriger pour maintenir un code propre et cohérent.

Checkstyle est un outil précieux pour assurer la qualité et la lisibilité du code Java.

1.3 Mise en place de la base de données

1.3.1 Scripts

Figure 22: Code creation DB

➢Add fresh token:

```
    CREATE TABLE `refresh_token` (
      `token` varchar(36) CHARACTER SET utf8mb4 COLLATE utf8mb4_0900_ai_ci
NOT NULL,
      `user_id` bigint NOT NULL,
      `expirity_date` timestamp NOT NULL,
      PRIMARY KEY (`token`),
      KEY `refresh_token_FK` (`user_id`),
      CONSTRAINT `refresh_token_FK` FOREIGN KEY (`user_id`) REFERENCES
`user` (`id`)
    );
```

Figure 23: Code creation DB

1.3.2 ER Diagramme

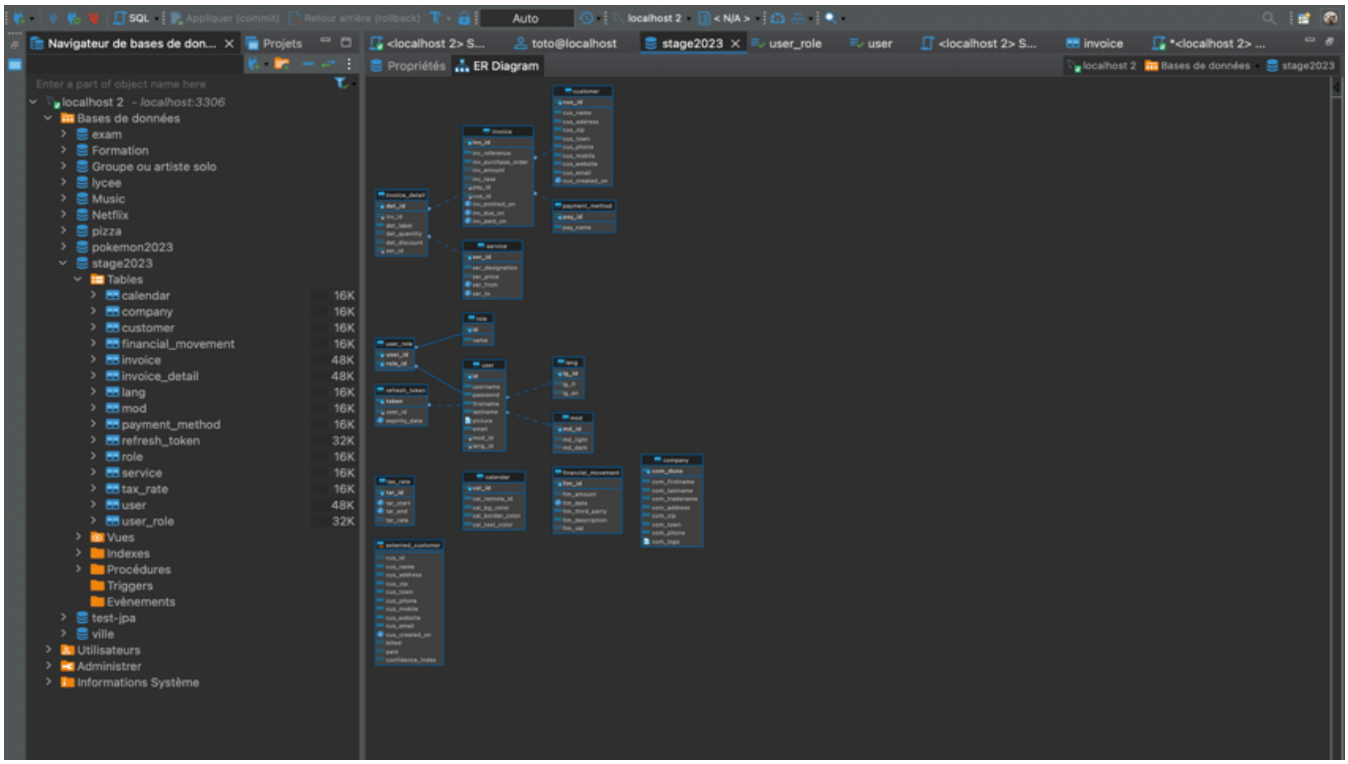

Figure 24: ER Diagramme

1.3.3 Privilège du schéma

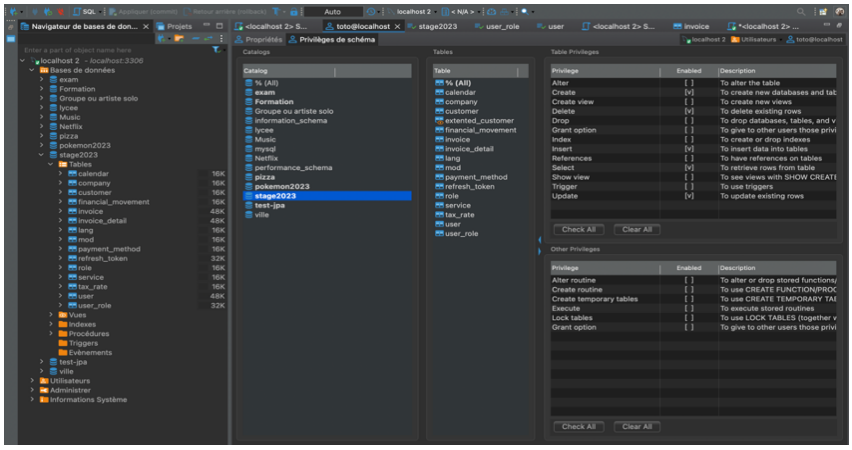

Figure 25: Privilège du schéma BD

J'ai Utilisé la fonction bcrypt pour hacher le mot de passe et crée un utilisateur :

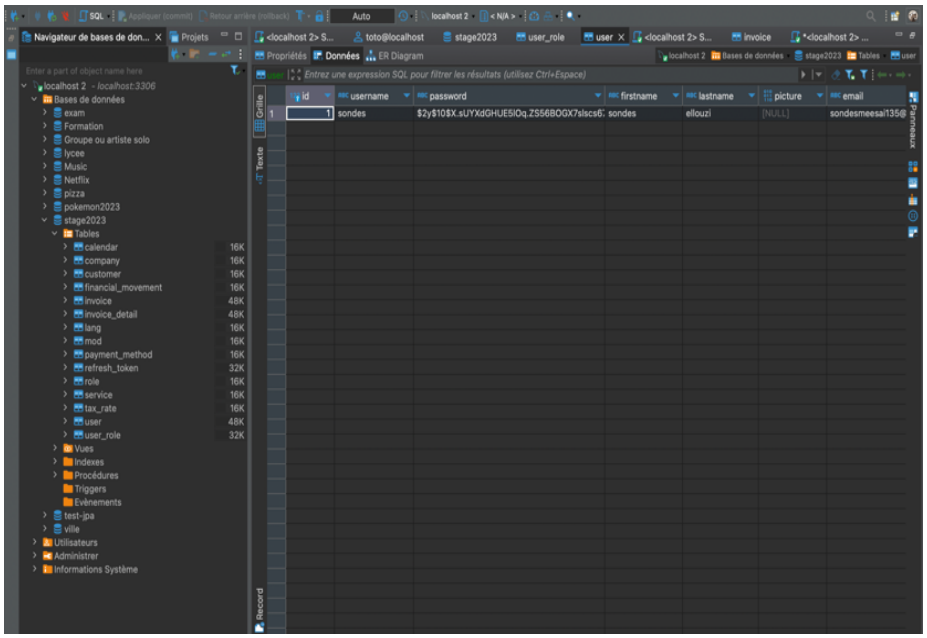

Figure 26: Création utilisateur BD

1.4 Structure du projet

1.4.1 Backend

Les éléments clés de la structure de notre projet backend sont :

- **Les Dtos** représentent les classes de transfert de données (Data Transfer Objects) utilisées pour transmettre les données entre le client et le serveur.
- **Les Repositories** représentent les interfaces qui étendent JpaRepository pour l'accès à la base de données (Repositories). Ils prennent en charge la persistance des données, en encapsulant l'accès à la base de données et en permettant des opérations de création, lecture, mise à jour et suppression (CRUD) sur les modèles.
- **Les modèles** représentent la structure des données au cœur de l'application, définissant leur forme et leurs relations.
- **Les services** abritent la logique métier de l'application, organisée en méthodes et algorithmes, favorisant la réutilisation et la facilité de maintenance. Les mappers assurent la conversion fluide des objets modèles entre la base de données et l'application, garantissant une interaction transparente des données.
- **Les contrôleurs** opèrent comme des points d'entrée, orchestrant les requêtes des utilisateurs et facilitant la communication harmonieuse entre les différentes composantes.

Cette démarche basée sur la POO favorise une distinction claire des responsabilités, contribuant ainsi à une meilleure maintenabilité, une plus grande réutilisation du code et une meilleure compréhension de l'application dans le contexte de la programmation orientée objet.

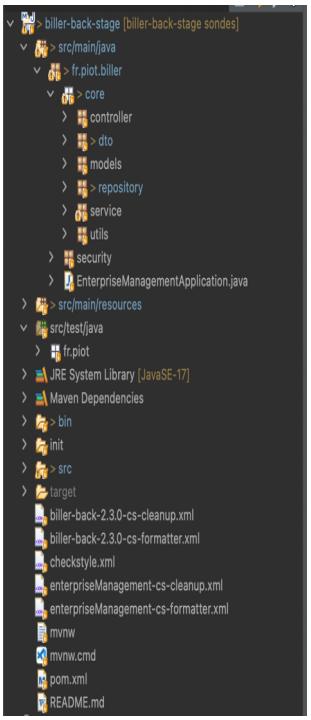

Figure 27: Structure projet backend

1.4.2 Frontend

La structure fondamentale du projet React se compose des éléments clés suivants :

- **Les composants** représentent les entités gèrent l'interface utilisateur de l'application. Chaque composant est responsable d'une partie spécifique de l'interface et peut être réutilisé à travers l'application pour assurer la cohérence visuelle.
- **Les helpers de données** représentent les aides (ou data helpers) prennent en charge la gestion des données. Elles sont responsables de l'acquisition, du traitement et de la manipulation des données avant de les transmettre aux composants ou aux modèles.
- **Les modèles** définissent la structure des données utilisées dans l'application. Ils représentent les données de manière abstraite et fournissent un schéma clair pour les données manipulées par l'application.
- **Les pages** définissent les différentes vues de l'application. Chaque page est généralement associée à un chemin d'URL spécifique et englobe les composants nécessaires pour rendre cette vue particulière.
- **Les services** sont responsables de la gestion des opérations complexes de l'application. Cela peut inclure des requêtes réseau, des calculs avancés ou d'autres tâches nécessitant une logique métier complexe.

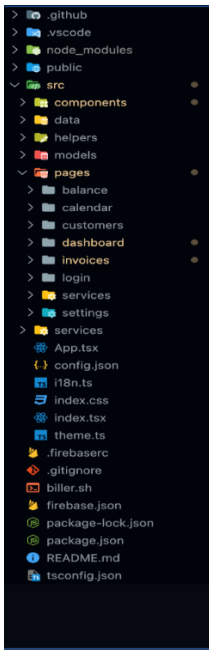

Figure 28: Structure projet frontend

L'organisation de l'application de cette manière permet une modélisation claire des données, une manipulation efficace des données à travers les composants, et une création d'interfaces utilisateur cohérentes. Cette approche favorise la création d'applications React modulaires et bien structurées, facilitant ainsi la maintenance et l'extension du code.

1.5 Mise place des fonctionnalités

1.5.1 Gestion TVA

Intégration dans le Backend

Couche DTO

J'ai créé un DTO pour faciliter le transfert de données entre les couches d'une application

```
/*-
package fr.piot.biller.core.dto;

/**
 * @author sondesEllouzi
 *
 */
public class CumulatedTaxeDto {

    /** the month. */
    private int month;

    /** the year. */
    private int year;

    /** the tzzg. */
    private Double total = 0.;

    /**
     * A constructor.
     *
     * @param pYear  a Year
     * @param pMonth a Month
     * @param pTotal a Total
     */
    public CumulatedTaxeDto(final int pYear, final int pMonth, final Double pTotal) {
        super();
        this.year = pYear;
        this.month = pMonth;
        this.total = pTotal;
    }

    /**
     * @return the month
     */
    public int getMonth() {
        return month;
    }

    /**
     * @param month the month to set
     */
    public void setMonth(int month) {
        this.month = month;
    }

    /**
     * @return the year
     */
    public int getYear() {
        return year;
    }

    /**
     * @param year the year to set
     */
    public void setYear(int year) {
        this.year = year;
    }

    /**
     * @return the total
     */
    public Double getTotal() {
        return total;
    }

    /**
     * @param total the total to set
     */
    public void setTaxe(Double total) {
        this.total = total;
    }
}
```

Figure 29: Code backend – Gestion TVA – DTOs

Couche Repository et services

J'ai étendu les fonctionnalités du repository Invoice en introduisant une nouvelle fonction nommée "getTaxe".

```
44    /**
45     *
46     * @param year  a year
47     * @param month a month
48     * @return the taxe for the given year and month
49     */
50    Double getTaxe(int year, int month);
51
```

Figure 30: Code backend – Gestion TVA – Services

J'ai créé une méthode au sein du service Invoice qui utilise cette fonction du repository pour obtenir les informations fiscales nécessaires.

```
74    @Override
75    public Double getTaxe(final int year, final int month) {
76        return invoiceRepo.getTaxe(year, month);
77    }
78
```

Figure 31: Code backend – Gestion TVA – Services

Et Cette requête SQL native récupère la somme des valeurs de la colonne "inv_amount" de la table "invoice" pour les enregistrements ayant une taxe égale à 1, ayant une date de paiement non nulle, et correspondant à une année et un mois spécifique. Si la somme est nulle, elle renvoie 0, puis divise le résultat par 6. Les valeurs de l'année et du mois sont passées en tant que paramètres à la requête.

```
65    /**
66     *
67     * @param year  a year
68     * @param month a month
69     * @return the taxe for the given year and month
70     */
71    @Query(nativeQuery = true, value = "SELECT IFNULL(SUM(inv_amount  ), 0)/6 " + "FROM invoice s "
72            + "WHERE inv_paid_on IS NOT NULL AND inv_taxe = 1"
73            + " AND YEAR(inv_paid_on)=:year AND MONTH(inv_paid_on) =:month ")
74    Double getTaxe(@Param("year") int year, @Param("month") int month);
75
```

Figure 32: Code backend – Gestion TVA – Repositories

Ce code utilise une annotation `@Query` dans le contexte d'un framework de persistance, probablement Hibernate, pour définir une requête SQL native à exécuter sur une base de données. Voici une explication ligne par ligne :

`@Query(nativeQuery = true, value = "...")`: Cette annotation est utilisée pour indiquer au framework que la méthode contient une requête SQL native plutôt qu'une requête JPQL (Java Persistence Query Language). `value` contient la requête SQL à exécuter, et `nativeQuery = true` précise que c'est une requête SQL native.

"SELECT IFNULL(SUM(inv_amount), 0)/6": C'est la partie de la requête qui effectue une opération SQL. Elle calcule la somme (SUM) des valeurs de la colonne "inv_amount" dans la table "invoice" et renvoie cette somme. Si la somme est NULL, elle est remplacée par 0. Ensuite, elle divise le résultat par 6.

"FROM invoice s": Cette partie de la requête spécifie que les données doivent être extraites de la table "invoice" et qu'elle est désignée par l'alias "s".

"WHERE inv_paid_on IS NOT NULL AND inv_taxe = 1 AND YEAR(inv_paid_on) = :year AND MONTH(inv_paid_on) = :month": C'est la clause WHERE qui filtre les résultats. Elle indique que seules les lignes où la colonne "inv_paid_on" n'est pas NULL et où la colonne "inv_taxe" a la valeur 1 sont incluses. De plus, il effectue des filtres sur l'année et le mois en utilisant les paramètres ":year" et ":month", ce qui signifie que ces valeurs sont probablement passées en tant qu'arguments à la requête.

Couche contrôleur

En ce qui concerne le contrôleur des statistiques, j'ai ajouté une nouvelle API qui fait appel au service Invoice pour récupérer les données pertinentes.

```
68●     /*
69       * Compute monthly taxe statistics for a specefic year.
70       *
71       * @param year the year to compute
72       *
73       * @param month the month to compute
74       */
75●     @GetMapping("/taxe/{year}/{month}")
76      CumulatedTaxeDto getCumulatedTaxeStats(@PathVariable("year") final int year,
77              @PathVariable("month") final int month) {
78
79          CumulatedTaxeDto data = new CumulatedTaxeDto(year, month, 0.0);
80          data.setTaxe(invoiceService.getTaxe(year, month));
81          return data;
82      }
83 |
```

Figure 33: Code backend – Gestion TVA – Contrôleurs

L'intégration dans le frontend

Couche views

1. J'ai ajouté l'importe nécessaire :
    a. StatsService à partir du fichier ../../services/StatsService, qui est un service pour obtenir les données statistiques.
    b. CumulatedTaxe à partir du fichier ../../models/statistics/CumulatedTaxe, qui est le modèle de données pour stocker les statistiques de la taxe.

```
import { Box, IconButton, Typography, useTheme } from "@mui/material";
import React, { useEffect, useState } from "react";
import StatsService from "../../services/StatsService";
import CumulatedTaxe from "../../models/statistics/CumulatedTaxe";
```

Figure 34: Code frontend – Gestion TVA – React Import

2. J'ai défini deux états (taxe et taxeLastYear) sont initialisés à l'aide de l'état useState. Ils sont de type CumulatedTaxe et sont tous deux initialisés avec des objets CumulatedTaxe ayant des valeurs par défaut de 0.

```
const Dashboard = () => {
  const [taxe, setTaxe] = useState<CumulatedTaxe>(new CumulatedTaxe(0, 0, 0
  const [taxeLastYear, setTaxeLastYear] = useState<CumulatedTaxe>(
    new CumulatedTaxe(0, 0, 0)
  );
```

Figure 35: Code frontend – Gestion TVA – React états

3. J'ai ajouté un useEffect pour effectuer des opérations de côté client après le rendu initial du composant.

À l'intérieur du useEffect, il y a un bloc try où le service StatsService est appelé deux fois avec les années actuelle et précédente ainsi que le mois en cours pour obtenir les statistiques de la taxe.

Les valeurs de la taxe actuelle et de l'année précédente sont mises à jour à l'aide des fonctions setTaxe et setTaxeLastYear respectivement.

```
  useEffect(() => {
    try {
      /* Obtenir la taxe du mois en cours */
      const month = new Date().getMonth();
      const year = new Date().getFullYear();

      StatsService.getTaxe(year, month)
        .then((value: CumulatedTaxe) => setTaxe(value))
        .catch((reason) => console.error(reason));

      StatsService.getTaxe(year-1, month)
        .then((value: CumulatedTaxe) => setTaxeLastYear(value))
        .catch((reason) => console.error(reason));
    } catch {}
  }, []);
```

Figure 36: Code frontend –Gestion TVA – React actions

4. J'ai ajouté dans l'élément <Box> de MUI un composant StatBox qui contient.
   - L'affiche le total de la TVA en utilisant la méthode toLocaleString pour formater le nombre avec deux décimales.
   - Un calcule et affichage de l'augmentation de la TVA par rapport à l'année précédente, sous forme de pourcentage. Cependant, si la taxe de l'année précédente est zéro ou moins, il affiche undefined.

```
<StatBox
  title={turnover.total?.toLocaleString(undefined, {
    maximumFractionDigits: 2,
    minimumFractionDigits: 2,
  })}
  unformattedTitle={turnover.total.toString()}
  subtitle={t("dashboardscreen.TotalTurnover")}
  progress={turnover.total / turnover.previousYearTotal}
  increase={
    turnover.previousYearTotal > 0
      ? 100 * (turnover.total / turnover.previousYearTotal - 1)
      : undefined
  }
  icon={
    <EuroOutlinedIcon
      sx={{
        color: theme.palette.secondary.main,
        fontSize: "40px",
      }}
```

Figure 37: Code frontend –Gestion TVA – React views

1.5.2 Emission d'une facture

Intégration dans le Backend :

➢Pom.xml :

Ces dépendances sont utilisées dans un projet Java basé sur Spring Boot pour la gestion des e-mails. La première dépendance, "spring-boot-starter-mail", inclut les fonctionnalités de base pour envoyer des e-mails avec Spring Boot. La deuxième dépendance, "javax.mail:mail:1.4.3", est une bibliothèque Java standard pour la manipulation des e-mails, et elle est utilisée en conjonction avec Spring Boot pour les opérations plus avancées, telles que la construction de messages personnalisés et l'accès aux pièces jointes. Ces dépendances facilitent le développement d'applications qui nécessitent des fonctionnalités d'envoi et de réception d'e-mails. La version 1.4.3 de javax.mail est spécifiée pour garantir la compatibilité avec Spring Boot.

```
94          <dependency>
95              <groupId>org.springframework.boot</groupId>
96              <artifactId>spring-boot-starter-mail</artifactId>
97          </dependency>
98          <dependency>
99              <groupId>javax.mail</groupId>
100             <artifactId>mail</artifactId>
101             <version>1.4.3</version>
102         </dependency>
103     </dependencies>
104
```

Figure 38: Code backend

➢Src/main/application.properties :

Ces paramètres de configuration sont destinés à permettre à une application Spring d'envoyer des e-mails via le serveur SMTP de Gmail (smtp.gmail.com). Les informations d'identification (nom d'utilisateur et mot de passe) sont fournies pour authentifier l'application, et le port 587 est spécifié pour la communication. De plus, l'option "starttls" est activée pour assurer une communication sécurisée avec le serveur SMTP de Gmail.

```
13
14 # Email Configuration
15 spring.mail.host=smtp.gmail.com
16 spring.mail.username=appstage5@gmail.com
17 spring.mail.password=ggac ocag oxgm tdao
18 spring.mail.port=587
19 spring.mail.properties.mail.smtp.auth=true
20 spring.mail.properties.mail.smtp.starttls.enable=true
21
```

Figure 39: Code backend – Emission d'une facture – Mail properties

Couche DTO

J'ai ajouté un Data Transfer Object (DTO) appelé **InvoiceMailDto** au sein de la couche service, ce qui nous permet de mieux gérer les données que nous envoyons par e-mail.

```
1  /**[
4  package fr.piot.biller.core.dto;
5
6  /**
7   * @author [redacted]
8   *
9   */
10 public class InvoiceMailDto {
11
12     /** the to. */
13     private String recipient;
14
15     /** the subject. */
16     private String subject;
17
18     /** the Body. */
19     private String body;
20
21     /**
22      *
23      */
24     public InvoiceMailDto() {
25         // TODO Auto-generated constructor stub
26     }
27
28     /**
29      * @return the Recipient
30      */
31     public String getRecipient() {
32         return recipient;
33     }
34
35     /**
36      * @param recipient the Recipient set
37      */
38     public void setRecipient(String recipient) {
39         this.recipient = recipient;
40     }
41
42     /**
43      * @return the subject
44      */
45     public String getSubject() {
46         return subject;
47     }
48
49     /**
50      * @param subject the subject to set
51      */
52     public void setSubject(String subject) {
53         this.subject = subject;
54     }
55
56     /**
57      * @return the body
58      */
59     public String getBody() {
60         return body;
61     }
62
63     /**
64      * @param body the body to set
65      */
66     public void setBody(String body) {
67         this.body = body;
68     }
69
70 }
71
```

Figure 40: Code backend – Emission d'une facture – Dtos

Couche Repository et services

J'ai enrichi notre application en introduisant un tout nouveau service d'envoi d'e-mails.

Cette interface déclare une méthode `send` pour envoyer un e-mail avec une pièce jointe, prenant comme arguments un objet `InvoiceMailDto`,

```java
package fr.piot.biller.core.service;

import java.io.ByteArrayInputStream;

public interface IMailService {

    // To send an email with attachment
    void send(InvoiceMailDto invoice, String

}

```

Figure 41: Code backend – Emission d'une facture – Services

Ce code représente une implémentation d'un service d'envoi d'e-mails avec des pièces jointes. Il gére l'envoi d'e-mails, en injectant `JavaMailSender`. La méthode `send` crée un e-mail avec une pièce jointe à partir des informations fournies dans l'objet `InvoiceMailDto`, puis envoie cet e-mail. En cas d'erreur, des exceptions sont gérées pour la notification.

```java
package fr.piot.biller.core.service.impl;

import java.io.ByteArrayInputStream;

@Service
public class MailService implements IMailService {

    @Autowired
    private JavaMailSender mailSender;

    // To send an email with attachment
    @Override
    public void send(InvoiceMailDto invoice, String fileName, ByteArrayInputStream fileContent) {

        // be send
        try {
            // Creating a mime message
            MimeMessage mimeMessage = mailSender.createMimeMessage();
            MimeMessageHelper mimeMessageHelper;

            mimeMessageHelper = new MimeMessageHelper(mimeMessage, true);

            mimeMessageHelper.setTo(invoice.getRecipient());
            mimeMessageHelper.setText(invoice.getBody());
            mimeMessageHelper.setSubject(invoice.getSubject());
            DataSource dataSource = new ByteArrayDataSource(fileContent.readAllBytes(), "application/pdf");
            mimeMessageHelper.addAttachment(fileName, dataSource);
            // Sending the mail
            mailSender.send(mimeMessage);
        } catch (MessagingException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        } catch (Exception e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
    }

    private ByteArrayInputStream getAttachmentAsInputStream(ByteArrayInputStream fileContent) {
        return new ByteArrayInputStream(fileContent.readAllBytes());
    }
}

```

Figure 42: Code backend – Emission d'une facture – Services

`InvoiceMailer` utilisée pour envoyer des e-mails avec des pièces jointes. Il utilise la bibliothèque Spring pour gérer l'envoi d'e-mails, en injectant `JavaMailSender`. La méthode `send` crée un e-mail avec une pièce jointe à partir des informations fournies dans l'objet `InvoiceMailDto` et envoie cet e-mail. Notez qu'il semble y avoir une erreur concernant la déclaration de la variable `mailSender`, car elle est déclarée en tant que statique.

```java
package fr.piot.biller.core.utils;

import java.io.ByteArrayInputStream;

/**
 * @author sondesEllouzi
 *
 */
public class InvoiceMailer {

    @Autowired
    private static JavaMailSender mailSender;

    // To send an email with attachment
    public static void send(InvoiceMailDto invoice, String fileName, ByteArrayInputStream fileContent) {

        // be send
        try {
            // Creating a mime message
            MimeMessage mimeMessage = mailSender.createMimeMessage();
            MimeMessageHelper mimeMessageHelper;

            mimeMessageHelper = new MimeMessageHelper(mimeMessage, true);

            mimeMessageHelper.setFrom("");
            mimeMessageHelper.setTo(invoice.getRecipient());
            mimeMessageHelper.setText(invoice.getBody());
            mimeMessageHelper.setSubject(invoice.getSubject());

            mimeMessageHelper.addAttachment(fileName, new InputStreamResource(fileContent),
                    MediaType.APPLICATION_PDF.toString());

            // Sending the mail
            mailSender.send(mimeMessage);
        } catch (MessagingException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
    }
}
```

Figure 43: Code backend – Emission d'une facture – Services

Couche contrôleur

De plus, j'ai effectué une mise à jour importante dans le contrôleur InvoiceController. Plus précisément, j'ai intégré un appel à ce service d'envoi d'e-mails au sein de la méthode "publishInvoice," qui est responsable d'émettre une facture. Cette modification nous permet d'automatiser l'envoi d'e-mails à chaque fois qu'une facture est émise.

```java
    /** import the mail service. */
    @Autowired
    private IMailService mailService;
```

Figure 44: Code backend

```java
    @PutMapping(value = "/publish/{id}")
    InvoiceDto publish(@PathVariable("id") final Long id, @RequestBody final InvoiceMailDto mail) {
        Invoice invoice = invoiceService.publishInvoice(id);
        Company company = companyService.getCompany();

        String fileName = StringUtils.defaultString(invoice.getInvoiceRef(), "Devis") + ".pdf";
        ByteArrayInputStream fileContent = InvoicePrinter.print(invoice, company);
        mailService.send(mail, fileName, fileContent);

        return InvoiceMapper.toDto(invoice);
    }
```

Figure 45: Code backend – Emission d'une facture – Backend

Intégration dans le frontend

Couche components

Afin de développer cette fonctionnalité j'ai commencé par créer un composant React qui crée une modal de suppression. Dans ce composant :

1. J'ai ajouté l'importations de différents modules nécessaires.
    a. InvoiceService service qui gère les opérations liées aux factures.

b. InvoiceMail et Invoice sont des modèles de données utilisés pour représenter les informations d'une facture et d'un courrier électronique, respectivement.

```
import InvoiceService from "../../services/InvoiceService";
import InvoiceMail from "../../models/invoices/InvoiceMail";
import Invoice from "../../models/invoices/Invoice";
```

Figure 46: Code frontend – Emission d'une facture – React import

2. J'ai déclaré une interface props qui définit les propriétés attendues par le composant.
   - show: Un booléen indiquant si la modal doit être affichée ou non.
   - setShowModal: Une fonction pour mettre à jour l'état de la modal.
   - invoice: Une instance de la classe Invoice (ou undefined si non spécifié).
   - updateInvoiceList: Une fonction pour mettre à jour la liste des factures.

```
interface props {
  show: boolean;
  setShowModal: Function;
  invoice?: Invoice;
  updateInvoiceList: Function;
}
```

Figure 47: Code frontend – Emission d'une facture – React properties

3. J'ai déclaré le composant PublishModal, Il reçoit des propriétés (props) conformes à l'interface définie précédemment.

```
const PublishModal = (params: props) => {
  // ...
};
```

Figure 48: Code frontend – Emission d'une facture – React components

Le composant PublishModal est défini ici. Il reçoit des propriétés (props) conformes à l'interface définie précédemment.

4. J'ai ajouté le hook useState pour gérer l'état de l'ouverture/fermeture de la modal, et useEffect pour mettre à jour l'état open lorsque la propriété show change:

```
const [open, setOpen] = useState<boolean>(false);
useEffect(() => {
  setOpen(params.show);
}, [params.show]);
```

Figure 49: Code frontend – Emission d'une facture – React états

5. J'ai utilisé useFormik pour gérer le formulaire à l'intérieur de la modal. J'ai initialisé les valeurs du formulaire avec l'ID de la facture, le destinataire , le sujet et le contenue du mail, et j'ai définit une fonction onSubmit avec un appel au service externe (**InvoiceService.publishInvoice**) pour supprimer une facture qui sera exécutée lorsque le formulaire est soumis

```
const formik = useFormik({
  initialValues: {
    id: params.invoice?.id,
    recipient: params.invoice?.customer?.email,
    subject: "Facturation " + month + "-" + year,
    body:
      "Bonjour," +
      " \n" +
      "Veuillez trouver ci-joint la facture pour mon activité de ce mois air
      " \n" +
      "Cordialement",
  },
  enableReinitialize: true,
  onSubmit: (values) => {
    // Logique de soumission de formulaire
  },
});
```

Figure 50: Code frontend – Emission d'une facture – React actions

```
onSubmit: (values) => {
  if (values.id) {
    var mail = new InvoiceMail(
      values.recipient ? values.recipient : "",
      values.subject,
      values.body
    );
    InvoiceService.publishInvoice(values.id, mail)
      .then((invoice) => {
        console.log(invoice);
        params.updateInvoiceList(invoice);
        handleClose();
      })
      .catch((reason) => console.error(reason));
  }
},
```

Figure 51: Code frontend – Emission d'une facture – React actions

6. J'ai ajouté le composant rend une modal avec un formulaire avec des champs pour l'adresse e-mail du destinataire, le sujet et le corps du courrier électronique. L'utilisateur peut annuler l'action ou envoyer la facture par courrier électronique en cliquant sur les boutons correspondants.

```
return (
  <Modal
    aria-labelledby="transition-modal-title"
    aria-describedby="transition-modal-description"
    open={open}
    onClose={handleClose}
    closeAfterTransition
    BackdropComponent={Backdrop}
    BackdropProps={{
      timeout: 500,
    }}
  >
    {/* Contenu de la modal */}
  </Modal>
);
```

Figure 52: Code frontend – Emission d'une facture – React views

Couche views

1. J'ai ajouté l'importations de différents modules nécessaires.

```
import Invoice from "../../models/invoices/Invoice";
import PublishModal from "../../components/invoice/PublishModal";
```

Figure 53: Code frontend – Emission d'une facture – React import

2. À l'intérieur du composant Invoices, j'ai ajouté les états à l'aide du Hook useState de React. Ces états incluent :
   a. invoices, qui est un tableau de factures (Invoice[]).
   b. showPublishModal, un booléen qui contrôle l'affichage du modèle de publication.
   c. invoiceToPublish, qui stockera une facture à publier.

```
const [invoices, setInvoices] = useState<Invoice[]>([]);
const [showPublishModal, setShowPublishModal] = useState<boolean>(false);
const [invoiceToPublish, setInvoiceToPublish] = useState<Invoice>();
```

Figure 54: Code frontend – Emission d'une facture – React états

3. J'ai ajouté une fonction **publishInvoice** qui prend un paramètre id (supposément l'ID de la facture à publier). Cette fonction est utilisée pour préparer une facture spécifique à la publication en mettant à jour l'état **invoiceToPublish** avec la facture correspondante trouvée dans le tableau invoices. De plus, elle définit l'état **showPublishModal** sur true, ce qui affiche le modèle de publication.

```
const publishInvoice = (id: number) => () => {
  let invoice = invoices.find((elm) => elm.id === id);
  setInvoiceToPublish(invoice);
  setShowPublishModal(true);
};
```

Figure 55: Code frontend – Emission d'une facture – React actions

4. J'ai ajouté la déclaration du modale « PublishModal »

```
<PublishModal
  show={showPublishModal}
  invoice={invoiceToPublish}
  setShowModal={setShowPublishModal}
  updateInvoiceList={updateInvoiceList}
/>
```

Figure 56: Code frontend – Emission d'une facture – React views

1.5.3 Emission d'une facture impayée

Intégration dans le Backend

Couche DTO

N/A

Couche Repository et services

Service `InvoiceService` : Vous avez modifié le service `InvoiceService` pour y ajouter un appel à l'API "unpaid". Cela signifie que le service est désormais capable de communiquer avec la méthode "Unpaid" du contrôleur `InvoiceController`.

Couche contrôleur

Contrôleur `InvoiceController` : j'ajoute une nouvelle méthode appelée "Unpaid" dans le contrôleur `InvoiceController`. Cette méthode semble faire appel à un service de messagerie avec en entrée un objet `InvoiceMailDto`. Cette méthode pourrait être utilisée pour gérer des factures impayées.

```java
 * @param id   the invoice's id
 * @param mail
 * @return the update invoice as a InvoiceDto
 */
@PutMapping(value = "/publish/{id}")
InvoiceDto publish(@PathVariable("id") final Long id, @RequestBody final InvoiceMailDto mail) {
    Invoice invoice = invoiceService.publishInvoice(id);
    Company company = companyService.getCompany();

    String fileName = StringUtils.defaultString(invoice.getInvoiceRef(), "Devis") + ".pdf";
    ByteArrayInputStream fileContent = InvoicePrinter.print(invoice, company);
    mailService.send(mail, fileName, fileContent);

    return InvoiceMapper.toDto(invoice);
}

/**
 * Set an invoice as unpaid (=emitted).
 *
 * @param id   the invoice's id
 * @param mail
 * @return the invoice as a InvoiceDto
 */
@PutMapping(value = "/unpaid/{id}")
InvoiceDto unpaid(@PathVariable("id") final Long id, @RequestBody final InvoiceMailDto mail) {
    Invoice invoice = invoiceService.getInvoiceById(id);
    Company company = companyService.getCompany();

    String fileName = StringUtils.defaultString(invoice.getInvoiceRef(), "Devis") + ".pdf";
    ByteArrayInputStream fileContent = InvoicePrinter.print(invoice, company);
    mailService.send(mail, fileName, fileContent);

    return InvoiceMapper.toDto(invoice);
}
```

Figure 57: Code backend – Emission d'une facture impayée – Contrôleurs

Intégration dans le frontend

Couche components

Afin de développer cette fonctionnalité j'ai commencé par créer un composant React qui crée une modal de suppression. Dans ce composant :

1. J'ai ajouté l'importations de composants et de bibliothèques : le code importe divers composants et bibliothèques nécessaires à la construction de la modal, y compris des composants MUI (Material-UI) tels que Modal, Backdrop, Fade, Button, Typography, et d'autres éléments

```java
import React, { useEffect, useState } from "react";
import Backdrop from "@mui/material/Backdrop";
import Modal from "@mui/material/Modal";
import Fade from "@mui/material/Fade";
import Button from "@mui/material/Button";
import Typography from "@mui/material/Typography";
import { Box, TextField, useTheme } from "@mui/material";
import { tokens } from "../../theme";
import { useFormik } from "formik";
import InvoiceService from "../../services/InvoiceService";
import { useTranslation } from "react-i18next";
import Invoice from "../../models/invoices/Invoice";
```

Figure 58: Code frontend – Emission d'une facture impayée – React Import

2. J'ai déclaré une interface props qui définit les propriétés attendues par le composant.
   - show: Un booléen indiquant si la modal doit être affichée ou non.
   - setShowModal: Une fonction pour mettre à jour l'état de la modal.
   - invoice: Une instance de la classe Invoice (ou undefined si non spécifié).
   - updateInvoiceList: Une fonction pour mettre à jour la liste des factures.

```
interface props {
  show: boolean;
  setShowModal: Function;
  invoice?: Invoice;
  updateInvoiceList: Function;
}
```

Figure 59: Code frontend – Emission d'une facture impayée – React properties

3. J'ai déclaré le composant DeleteModal, Il reçoit des propriétés (props) conformes à l'interface définie précédemment.

```
const UnpaidModal = (params: props) => {
  // ...
};
```

Figure 60: Code frontend – Emission d'une facture impayée – React components

Le composant UnpaidModal est défini ici. Il reçoit des propriétés (props) conformes à l'interface définie précédemment.

4. J'ai ajouté les Hooks React : tels que useTheme() pour obtenir le thème actuel, et useTranslation() pour la gestion de la traduction dans l'application.

```
const theme = useTheme();
const colors = tokens(theme.palette.mode);
const { t } = useTranslation();
```

Figure 61: Code frontend – Emission d'une facture impayée – React états

5. J'ai ajouté le hook useState pour gérer l'état de l'ouverture/fermeture de la modal, et useEffect pour mettre à jour l'état open lorsque la propriété show change:

```
const [open, setOpen] = useState<boolean>(false);
useEffect(() => {
  setOpen(params.show);
}, [params.show]);
```

Figure 62: Code frontend – Emission d'une facture impayée – React états

6. J'ai utilisé useFormik pour gérer le formulaire à l'intérieur de la modal. J'ai initialisé les valeurs du formulaire avec l'ID de la facture , le destinataire , le sujet et le contenue du mail, et

```
const formik = useFormik({
  initialValues: {
    id: params.invoice?.id,
    invoice: params.invoice,
  },
  enableReinitialize: true,
  onSubmit: (values) => {
    // ...
  },
});
```

Figure 63: Code frontend – Emission d'une facture impayée – React actions

7. J'ai défini une fonction onSubmit avec un appel au service externe (**InvoiceService.unpaidInvoice**) pour supprimer une facture qui sera exécutée lorsque le formulaire est soumis

```
onSubmit: (values) => {
  if (values.id) {
    // Crée une nouvelle instance de InvoiceMail
    var mail = new InvoiceMail(
      values.recipient ? values.recipient : "",
      values.subject,
      values.body
    );
    // Appelle le service InvoiceService pour traiter la facture impayée
    InvoiceService.unpaidInvoice(values.id, mail)
      .then((invoice) => {
        // Met à jour la liste des factures
        params.updateInvoiceList(invoice);
        handleClose();
      })
      .catch((reason) => console.error(reason));
  }
},
```

Figure 64: Code frontend – Emission d'une facture impayée – React actions

8. J'ai ajouté le composant rend une modal utilisant les composants MUI (Modal, Backdrop, Fade). Le contenu de la modal comprend un titre, des boutons "Annuler" et "Confirmer", et il est géré par le formulaire défini précédemment.

```
return (
  <Modal
    aria-labelledby="transition-modal-title"
    aria-describedby="transition-modal-description"
    open={open}
    onClose={handleClose}
    closeAfterTransition
    BackdropComponent={Backdrop}
    BackdropProps={{
      timeout: 500,
    }}
  >
    {/* Contenu de la modal */}
  </Modal>
);
```

Figure 65: Code frontend – Emission d'une facture impayée – React views

Couche views

1. J'ai ajouté l'importations de différents modules nécessaires.

```
import Invoice from "../../models/invoices/Invoice";
import UnpaidModal from "../../components/invoice/UnpaidModal";
```

Figure 66: Code frontend – Emission d'une facture impayée – React import

2. À l'intérieur du composant Invoices, j'ai ajouté les états à l'aide du Hook useState de React. Ces états incluent :
   a. invoices, qui est un tableau de factures (Invoice[]).
   b. showUnpaidModal, un booléen qui contrôle l'affichage du modèle de publication.
   c. invoiceToUnpaid, qui stockera une facture à publier.

```
const [invoices, setInvoices] = useState<Invoice[]>([]);
const [showUnpaidModal, setShowUnpaidModal] = useState<boolean>(false);
const [invoiceToUnpaid, setInvoiceToUnpaid] = useState<Invoice>();
```

Figure 67: Code frontend – Emission d'une facture impayée – React états

3. J'ai ajouté une fonction **unpaidInvoice** qui prend un paramètre id (supposément l'ID de la facture à publier). Cette fonction est utilisée pour préparer une facture spécifique à envoyer en mettant à jour l'état **invoiceToUnpaid** avec la facture correspondante trouvée dans le tableau invoices. De plus, elle définit l'état **showUnpaidModal** sur true, ce qui affiche le modèle de publication.

```
const unpaidInvoice = (id: number) => () => {
  let invoice = invoices.find((elm) => elm.id === id);
  setInvoiceToUnpaid(invoice);
  setShowUnpaidModal(true);
};
```

Figure 68: Code frontend – Emission d'une facture impayée – React actions

4. J'ai ajouté la déclaration du modale « UnpaidModal » et j'ai ajouté une grid actions.

```
return (
  <Box margin="0 20px">
    {/* ... */}
    <GridActionsCellItem
      icon={<UnpaidOutlinedIcon />}
      label={t("invoiceScreen.Unpaid")}
      onClick={unpaidInvoice(params.row.id)}
      showInMenu
      disabled={params.row.invoiceRef !== undefined}
    />
    <UnpaidModal
      show={showUnpaidModal}
      invoice={invoiceToUnpaid}
      setShowModal={setShowUnpaidModal}
      updateInvoiceList={updateInvoiceList}
    />
  </Box>
```

Figure 69: Code frontend – Emission d'une facture impayée – React views

1.5.4 Suppression d'une facture

Intégration dans le frontend

Couche components

Afin de développer cette fonctionnalité j'ai commencé par créer un composant React qui crée une modal de suppression. Dans ce composant :

1. J'ai ajouté l'importations de composants et de bibliothèques : le code importe divers composants et bibliothèques nécessaires à la construction de la modal, y compris des composants MUI (Material-UI) tels que Modal, Backdrop, Fade, Button, Typography, et d'autres éléments

```
import React, { useEffect, useState } from "react";
import Backdrop from "@mui/material/Backdrop";
import Modal from "@mui/material/Modal";
import Fade from "@mui/material/Fade";
import Button from "@mui/material/Button";
import Typography from "@mui/material/Typography";
import { Box, TextField, useTheme } from "@mui/material";
import { tokens } from "../../theme";
import { useFormik } from "formik";
import InvoiceService from "../../services/InvoiceService";
import { useTranslation } from "react-i18next";
import Invoice from "../../models/invoices/Invoice";
```

Figure 70: Code frontend - Suppression d'une facture – React Import

2. J'ai déclaré une interface props qui définit les propriétés attendues par le composant.

```
interface props {
  show: boolean;
  setShowModal: Function;
  invoice?: Invoice;
  updateInvoiceList: Function;
}
```

Figure 71: Code frontend - Suppression d'une facture – React Properties

3. J'ai déclaré le composant DeleteModal, Il reçoit des propriétés (props) conformes à l'interface définie précédemment.

```
const DeleteModal = (params: props) => {
  // ...
};
```

Figure 72: Code - Code frontend - Suppression d'une facture – React Déclaration

Le composant DeleteModal est défini ici. Il reçoit des propriétés (props) conformes à l'interface définie précédemment.

4. J'ai ajouté les Hooks React : tels que useTheme() pour obtenir le thème actuel, et useTranslation() pour la gestion de la traduction dans l'application.

```
const theme = useTheme();
const colors = tokens(theme.palette.mode);
const { t } = useTranslation();
```

Figure 73: Code frontend - Suppression d'une facture – React Hooks

5. J'ai ajouté le hook useState pour gérer l'état de l'ouverture/fermeture de la modal, et useEffect pour mettre à jour l'état open lorsque la propriété show change:

```
const [open, setOpen] = useState<boolean>(false);
useEffect(() => {
  setOpen(params.show);
}, [params.show]);
```

Figure 74: Code frontend - Suppression d'une facture – React états

6. J'ai utilisé useFormik pour gérer le formulaire à l'intérieur de la modal. J'ai initialisé les valeurs du formulaire avec l'ID de la facture et la facture elle-même, et j'ai définit une fonction onSubmit avec un appel au service externe (**InvoiceService.deleteInvoice**) pour supprimer une facture qui sera exécutée lorsque le formulaire est soumis

```
const formik = useFormik({
  initialValues: {
    id: params.invoice?.id,
    invoice: params.invoice,
  },
  enableReinitialize: true,
  onSubmit: (values) => {
    // ...
  },
});
```

Figure 75: Code frontend - Suppression d'une facture – React Actions

7. J'ai ajouté le composant rend une modal utilisant les composants MUI (Modal, Backdrop, Fade). Le contenu de la modal comprend un titre, des boutons "Annuler" et "Confirmer", et il est géré par le formulaire défini précédemment.

```
return (
  <Modal
    aria-labelledby="transition-modal-title"
    aria-describedby="transition-modal-description"
    open={open}
    onClose={handleClose}
    closeAfterTransition
    BackdropComponent={Backdrop}
    BackdropProps={{
      timeout: 500,
    }}
  >
    {/* Contenu de la modal */}
  </Modal>
);
```

Figure 76: Code frontend - Suppression d'une facture – React Views

Couche views

1. J'ai ajouté l'importations de différents modules nécessaires.

```
import Invoice from "../../models/invoices/Invoice";
import DeletedModal from "../../components/invoice/DeletedModal";
```

Figure 77: Code frontend - Suppression d'une facture – React Import

2. À l'intérieur du composant Invoices, j'ai ajouté les états à l'aide du Hook useState de React. Ces états incluent :
    a. showDeletedModal, un booléen qui contrôle l'affichage du modèle de suppression.
    b. invoiceToDeleted, qui stockera une facture à publier.

```
const [invoices, setInvoices] = useState<Invoice[]>([]);
const [showDeletedModal, setShowDeletedModal] = useState<boolean>(false);
const [invoiceToDeleted, setInvoiceToDeleted] = useState<Invoice>();
```

Figure 78: Code frontend - Suppression d'une facture – React états

3. J'ai ajouté une fonction **deleteInvoice** qui prend un paramètre id (supposément l'ID de la facture à supprimer).

Cette fonction est utilisée pour préparer une facture spécifique à envoyer en mettant à jour l'état **invoiceToDelete** avec la facture correspondante trouvée dans le tableau invoices. De plus, elle définit l'état **showDeleteModal** sur true, ce qui affiche le modèle de publication.

```
const deletedInvoice = (id: number) => () => {
  let invoice = invoices.find((elm) => elm.id === id);
  setInvoiceToDeleted(invoice);
  setShowDeletedModal(true);
};
```

Figure 79: Code frontend - Suppression d'une facture – React actions

4. J'ai ajouté la déclaration du modale « **Delete**Modal ».

```
return (
  <Box margin="0 20px">
    {/* ... */}
    <GridActionsCellItem
      icon={<DeletedOutlinedIcon />}
      label={t("invoiceScreen.Deleted")}
      onClick={deletedInvoice(params.row.id)}
      showInMenu
      disabled={params.row.invoiceRef !== undefined}
    />
    <DeletedModal
      show={showDeletedModal}
      invoice={invoiceToDeleted}
      setShowModal={setShowDeletedModal}
      updateInvoiceList={updateInvoiceList}
    />
  </Box>
);
```

Figure 80: Code frontend - Suppression d'une facture – React views

1.5.5 Traduction

La traduction JSON consiste à convertir des données depuis le format JSON (JavaScript Object Notation) vers une autre langue ou format de données, et vice versa. Elle permet d'interagir efficacement entre différentes applications et systèmes en utilisant un langage de données universel. Cela facilite l'échange d'informations structurées de manière cohérente.

English

```
"Create": "Create",
   "Save": "Save",
   "Update": "Update",
   "Send": "Send",
   "month": {
     "1": "January",
     "2": "February",
   @@ -74,7 +75,8 @@
   "monthlyTurnover": "Monthly turnover",
   "cumulatedTurnover": "Cumulated turnover",
   "comparedTo": "{{diff}} compared to {{ref}}",
   "AverageNet": "Average Monthly Net",
   "CumulatedTaxe": "Cumulated Taxe"
 },
 "customerScreen": {
   "Title": "Customers",
   @@ -174,6 +176,12 @@
   "TotalWoTaxes": "Total (VAT exc.)",
   "TotalWTaxes": "Total (VAT inc.)"
 },
 "publishModal": {
   "Title": "Send invoice to client",
   "recipient": "To",
   "subject": "Subject",
   "body": "Body"
 },
 "balanceScreen": {
   "Title": "Balance",
   "Subtitle": "Manage your company's financial movements",
```

Figure 81: Traduction anglais

Français

```
"Create": "Créer",
   "Save": "Enregistrer",
   "Update": "Mettre à jour",
   "Send": "Envoyer",
   "month": {
     "1": "Janvier",
     "2": "Février",
   @@ -74,7 +75,8 @@
   "monthlyTurnover": "Chiffre d'affaire mensuel",
   "cumulatedTurnover": "Chiffre d'affaire cumulé",
   "comparedTo": "{{diff}} comparé à {{ref}}",
   "AverageNet": "Net mensuel moyen",
   "CumulatedTaxe": "Total taxe"
 },
 "customerScreen": {
   "Title": "Clients",
   @@ -153,6 +155,12 @@
   "latePayment": "en retard de paiement",
   "Delete": "Supprimer"
 },
 "publishModal": {
   "Title": "Envoyer la facture au client",
   "recipient": "Destinataire",
   "subject": "Object",
   "body": "Text"
 },
 "invoiceForm": {
   "CreateTitle": "Créer une facture",
```

Figure 82: Traduction français

1.6 Démonstration et tests

1.6.1 Gestion TVA

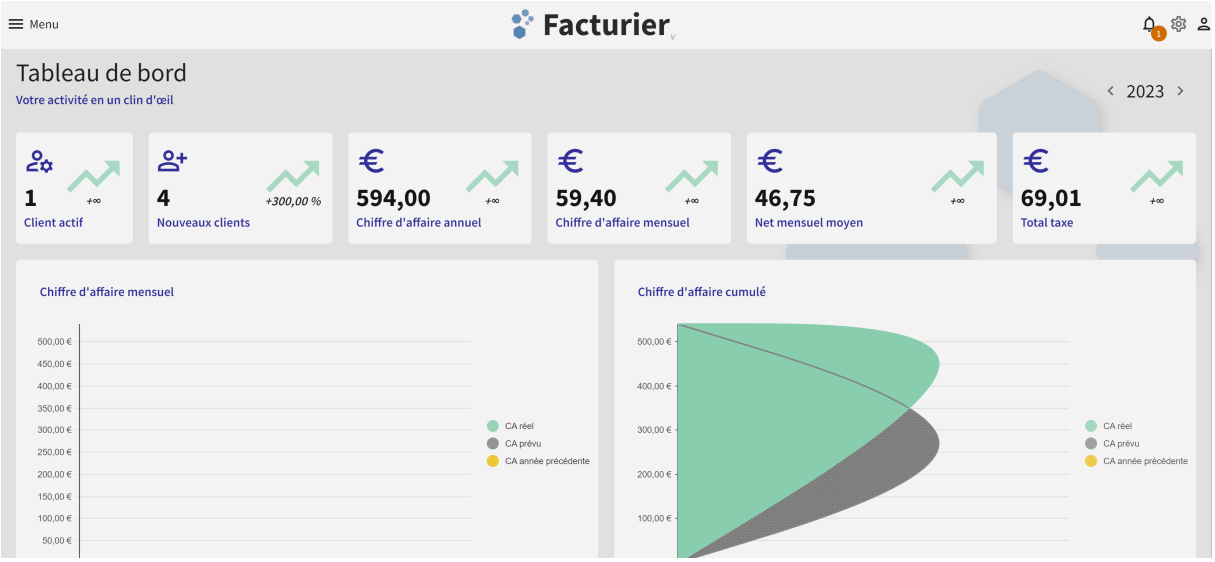

Figure 83: Interface dashboard

1.6.2 Emission d'une facture

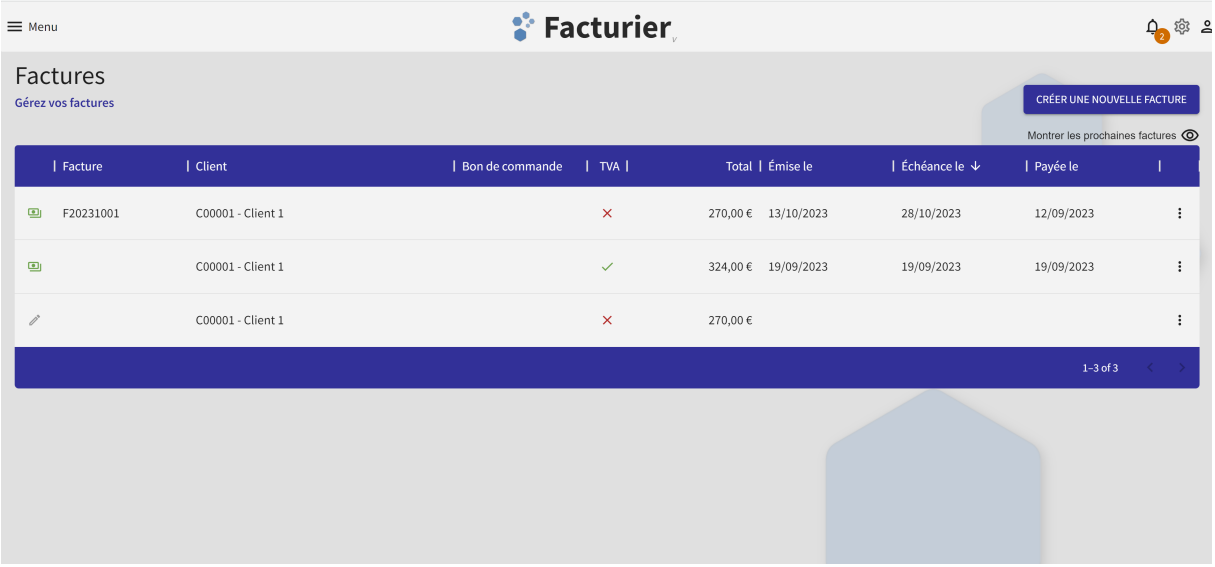

Figure 84: Interface pour émettre une facture

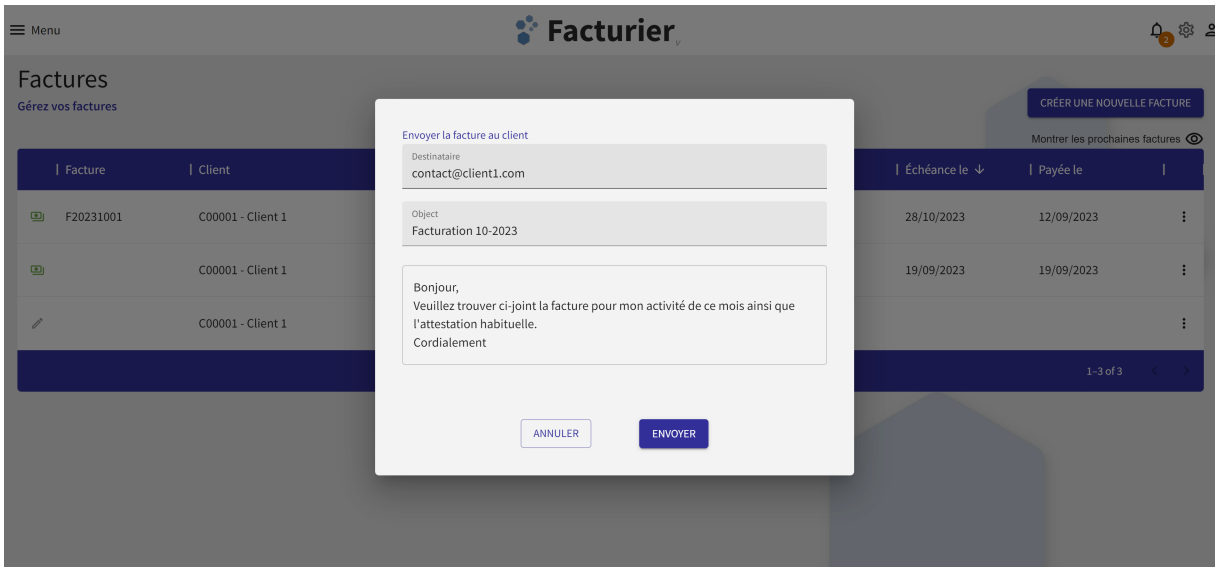

Figure 85: Interface pour émettre une facture

1.6.3 Emission d'une facture impayée

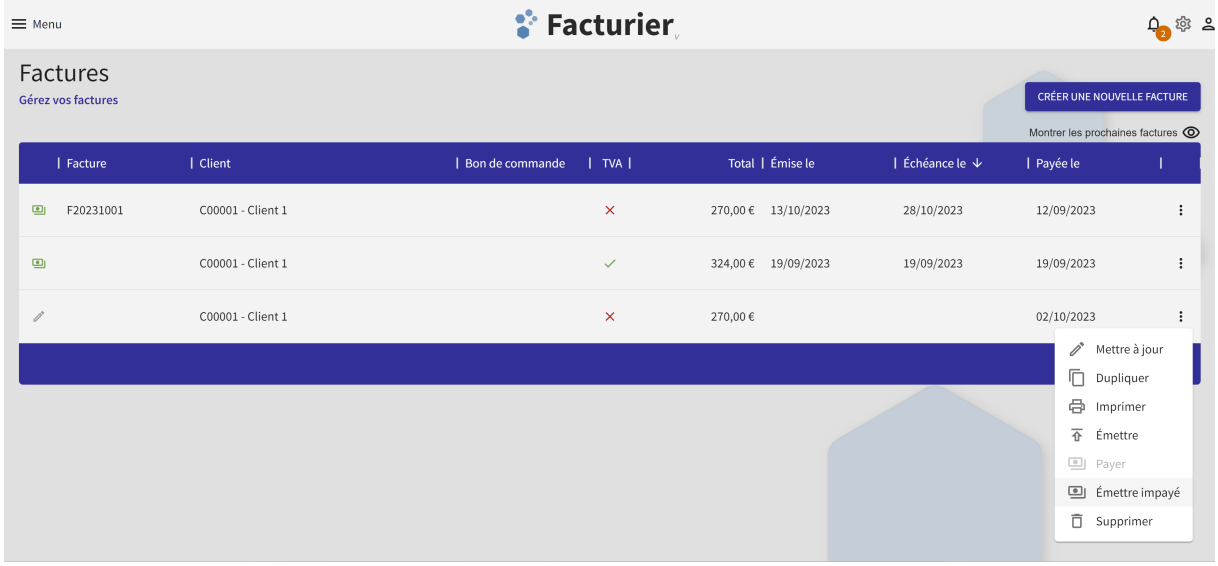

Figure 86: Interface pour émettre une facture impayée

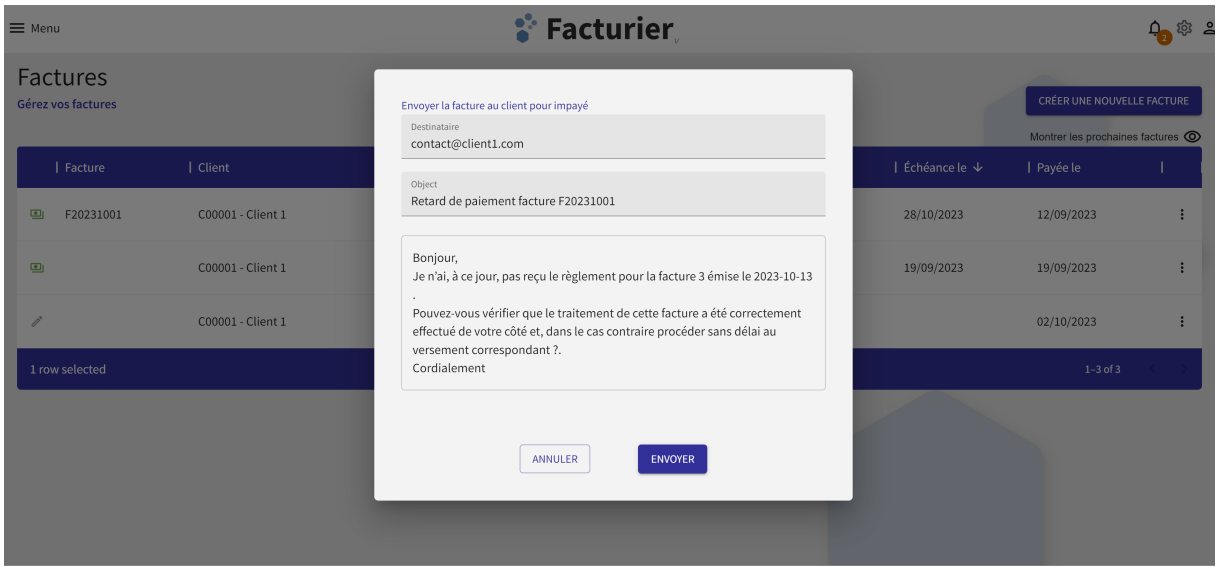

Figure 87: Interface pour émettre une facture impayée

1.6.4 Suppression d'une facture

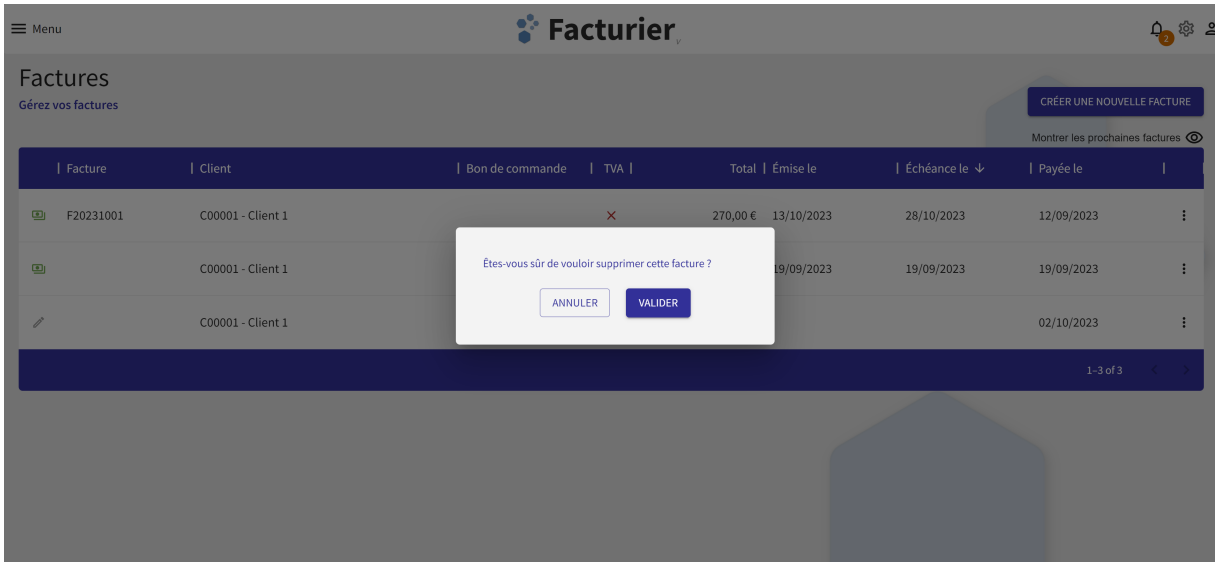

Figure 88: Interface de suppression d'une facture

1.6.5 Présentation du jeu d'essai

☑ **Tests fonctionnels :** sont des tests qui évaluent le comportement global d'une application en simulant des interactions utilisateur.

Ils sont généralement automatisés pour vérifier si l'application répond aux spécifications fonctionnelles attendues.

	Envoyer la Facture avec Génération PDF et Envoi par Courrier Électronique	Vérifier que le système génère correctement une facture avec le bon numéro automatique (ex:bdm-Facture-Pxx), génère un PDF	1. Utilisateur non connecté à l'application de gestion des factures.		1. Sélectionner la facture à Modifier Utilisateur accède à la liste des factures enregistrées. Utilisateur identifie la facture avec le statut d'échéance passée et le statut de paiement vide.	1. Vérifications Vérifier que le statut de la facture est correctement indiqué dans le sujet du courrier électronique. Vérifier que le statut d'émission de la facture est correctement indiqué dans le corps du courrier électronique. Vérifier que le champ "À" du courrier électronique est pré-remplissez l'adresse électronique du client associée à la facture.			
	Envoyer la Facture en retard de Paiement avec par Courrier Électronique	Vérifier que le système identifie correctement les factures en retard de paiement, génère un rappel de paiement au PDF, envoie le courriel électronique avec les informations précédentes pour le destinataire le sujet et le contenu.	2. Une facture a été ajoutée dans le système et un rappel a le processus d'ajout de factures.		Aller à l'Processus d'Ajoute pour Rappel de Paiement L'utilisateur clique sur le bouton "Envoyer" pour initier le processus d'ajoute pour retard de paiement.	2. Résultat Final Une fenêtre de courrier électronique s'ouvre automatiquement avec les champs préremplis pour le client pour signaler le retard de paiement de la facture.			
					3. L'utilisateur clique sur le bouton "Envoyer".				
	Suppression d'une Facture en Mode Brouillon	Vérifier que le système permet la suppression d'une facture en mode "brouillon" non émise,uniquement, et affiche une popup de confirmation pour éviter les suppressions accidentelles	1. Utilisateur non connecté à l'application de gestion des factures.		1. Sélectionner la facture à Supprimer Utilisateur accède à la liste des factures enregistrées. Utilisateur identifie et clique sur la facture en mode "brouillon" à supprimer.	1. Vérifications Vérifier que le pop-up de confirmation s'affiche correctement avec le message approprié. Vérifier que la facture en mode "brouillon" est supprimée de la base de données si l'utilisateur clique sur "Confirmer". Vérifier que la facture en mode "brouillon" reste inchangée si l'utilisateur clique sur "Annuler".			
			2. Une facture a été ajoutée dans le système et un rappel a le processus d'ajoute de factures.		Aller à l'Processus de Suppression L'utilisateur clique sur l'option de menu "Supprimer" ou un bouton similaire associé à la facture sélectionnée.	2. Résultat Final La facture en mode "brouillon" est supprimée de la base de données si l'utilisateur a confirmé la suppression.			
					3. Afficher le Popup de Confirmation Le système affiche une pop-up de confirmation avec le message : "Êtes-vous sûr de vouloir supprimer cette facture ?" Le pop-up propose les boutons "Confirmer" et "Annuler".				
					4. Confirmer la Suppression Si l'utilisateur clique sur "Confirmer", la facture en mode "brouillon" est supprimée de la base de données.				

Figure 89: Tests fonctionnels

☑ **Tests unitaires :** Afin de m'assurer que les différentes fonctionnalités de mon application fonctionnent correctement tout au long du développement, j'ai mis en place des tests unitaires et fonctionnels. Ces tests sont des scripts qui ont pour but de réaliser des actions afin de s'assurer qu'une action ou une fonctionnalité de l'application fonctionne correctement.

Le framework springboot permet de mettre en place facilement ce genre de tests.

```java
package fr.piot.biller.core.repository;

import static org.junit.jupiter.api.Assertions.assertEquals;
import static org.junit.jupiter.api.Assertions.assertNotNull;
import static org.mockito.ArgumentMatchers.any;
import static org.mockito.Mockito.mock;
import static org.mockito.Mockito.times;
import static org.mockito.Mockito.verify;
import static org.mockito.Mockito.when;

import java.util.ArrayList;
import java.util.List;
import java.util.Optional;

import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;
import org.springframework.boot.test.context.SpringBootTest;

import fr.piot.biller.core.models.Customer;
import fr.piot.biller.core.models.Invoice;
import fr.piot.biller.core.models.InvoiceDetail;
import fr.piot.biller.core.models.PaymentMethod;
import fr.piot.biller.core.models.Service;

@SpringBootTest
public class InvoiceRepositoryTest {

    private IServiceRepository serviceRepository;
    private ICustomerRepository customerRepository;
    private IInvoiceRepository invoiceRepository;

    @BeforeEach
    public void setup() {
        // Créez des mocks pour les repositories
        serviceRepository = mock(IServiceRepository.class);
        customerRepository = mock(ICustomerRepository.class);
        invoiceRepository = mock(IInvoiceRepository.class);

        Service service = new Service();
        service.setId(1L);
        service.setDesignation("Service 1");
        service.setPrice(100.0);

        when(serviceRepository.findById(1L)).thenReturn(Optional.of(service));

        Customer customer = new Customer();
        customer.setId(1L);
        customer.setName("Client 1");
        customer.setAddress("Test");

        when(customerRepository.findById(1L)).thenReturn(Optional.of(customer));
    }

    @Test
    public void add_Invoice_OK() throws Exception {
        Service service = serviceRepository.findById(1L).get();
        Customer customer = customerRepository.findById(1L).get();

        Invoice invoice = new Invoice();
        InvoiceDetail invoiceDetail = new InvoiceDetail();
        List<InvoiceDetail> listInvoiceDetails = new ArrayList<>();

        invoice.setId(1L);
        invoice.setCustomer(customer);
        invoice.setAmount(300.0);
        // Créez un mock pour PaymentMethod, car il n'est pas inclus dans votre exemple.
        PaymentMethod paymentMethod = mock(PaymentMethod.class);
        when(paymentMethod.getId()).thenReturn(1L);
        when(paymentMethod.getName()).thenReturn("Carte");
        invoice.setPaymentMethod(paymentMethod);
        invoice.setTaxe(true);
        invoiceDetail.setId(1L);
        invoiceDetail.setInvoice(invoice);
        invoiceDetail.setService(service);
        invoiceDetail.setQuantity(3.0);
        invoiceDetail.setDiscount(0.0);

        listInvoiceDetails.add(invoiceDetail);
        invoice.setDetails(listInvoiceDetails);

        when(invoiceRepository.save(any(Invoice.class))).thenReturn(invoice);

        Invoice savedInvoice = invoiceRepository.save(invoice);

        assertNotNull(savedInvoice);
        assertEquals(1L, savedInvoice.getId());
        assertEquals(customer, savedInvoice.getCustomer());
        assertEquals(300.0, savedInvoice.getAmount());
        // Vérifiez les autres attributs de l'objet invoice si nécessaire.

        verify(invoiceRepository, times(1)).save(invoice);
    }

    @Test
    public void publish_Invoice_OK() throws Exception {
        // Reproduisez le même processus pour la création de l'invoice.
        // Vous pouvez réutiliser la logique de création d'invoice du test précédent.

        Service service = serviceRepository.findById(1L).get();
        Customer customer = customerRepository.findById(1L).get();

        Invoice invoice = new Invoice();
        InvoiceDetail invoiceDetail = new InvoiceDetail();
        List<InvoiceDetail> listInvoiceDetails = new ArrayList<>();

        invoice.setId(2L); // Utilisez un ID différent pour ce test
        invoice.setCustomer(customer);
        invoice.setAmount(500.0); // Montant différent
        PaymentMethod paymentMethod = mock(PaymentMethod.class);
        when(paymentMethod.getId()).thenReturn(2L); // ID différent
        when(paymentMethod.getName()).thenReturn("Chèque"); // Nom différent
        invoice.setPaymentMethod(paymentMethod);
        invoice.setTaxe(false); // Taxe différente
        invoiceDetail.setId(2L); // ID différent
        invoiceDetail.setInvoice(invoice);
        invoiceDetail.setService(service);
        invoiceDetail.setQuantity(4.0); // Quantité différente
        invoiceDetail.setDiscount(0.1); // Remise différente

        listInvoiceDetails.add(invoiceDetail);
        invoice.setDetails(listInvoiceDetails);

        when(invoiceRepository.save(any(Invoice.class))).thenReturn(invoice);

        Invoice savedInvoice = invoiceRepository.save(invoice);

        assertNotNull(savedInvoice);
        assertEquals(2L, savedInvoice.getId());
        assertEquals(customer, savedInvoice.getCustomer());
        assertEquals(500.0, savedInvoice.getAmount());
        // Vérifiez les autres attributs de l'objet invoice si nécessaire.

        verify(invoiceRepository, times(1)).save(invoice);
    }
}
```

Figure 90: Tests unitaires

Dans le cas où toutes les entrées ont des résultats pertinents, on obtient alors un résultat de test positif comme visible sur la capture. Sinon c'est qu'on a mal évalué les résultats ou alors que la fonction ne fait pas ce qu'elle devrait réaliser.

1.7 Description de la veille

La veille pour un projet React Spring Boot est essentielle pour rester à jour avec les technologies et les meilleures pratiques. Voici une description de ce que cela implique :

- ☑ **Technologies React et Spring Boot : Suivez les dernières mises à jour de React** (bibliothèque JavaScript) et Spring Boot (framework Java) en consultant les blogs, les forums de développeurs et les annonces officielles. Assurez-vous d'adopter les versions stables et de prendre en compte les changements majeurs.
- ☑ **Sécurité** : Restez informé des dernières vulnérabilités de sécurité associées à React, Spring Boot et leurs dépendances. Assurez-vous de mettre en place des pratiques de sécurité robustes pour protéger votre application.

- ☑ **Nouveaux outils et bibliothèques** : Explorez de nouveaux outils, bibliothèques et packages liés à React et Spring Boot qui peuvent améliorer le développement, la productivité ou les performances de votre projet.
- ☑ **Bonnes pratiques de développement** : Gardez un œil sur les meilleures pratiques de développement pour React et Spring Boot, telles que les modèles d'architecture, la gestion de l'état, la structuration du code, et assurez-vous de les appliquer à votre projet.
- ☑ **Performance** : Surveillez les conseils et astuces pour optimiser les performances de votre application React Spring Boot, en accordant une attention particulière au chargement initial, à la gestion de l'état et à la réactivité de l'interface utilisateur.
- ☑ **Tendances du marché** : Comprendre les tendances du marché en matière de développement web et mobile, telles que les PWA (Progressive Web Apps), les technologies de conteneurisation, les architectures microservices, etc., et évaluez comment elles peuvent affecter votre projet.
- ☑ **Actualités de l'écosystème** : Restez à l'écoute des actualités de l'écosystème JavaScript, Java et des technologies connexes, car elles peuvent influencer les choix architecturaux et les intégrations de votre projet.
- ☑ **Communauté et forums** : Participez à des communautés de développeurs, des forums de discussion et des groupes de réseaux sociaux dédiés à React, Spring Boot et aux technologies connexes pour poser des questions, partager des expériences et bénéficier de l'expertise collective.
- ☑ **Formation et documentation** : Explorez les ressources de formation en ligne, les tutoriels et la documentation officielle de React et Spring Boot pour approfondir vos compétences et résoudre des problèmes spécifiques.
- ☑ **Tests et mises à jour** : Assurez-vous de maintenir votre application à jour avec les dernières versions des dépendances, de réaliser des tests réguliers pour détecter les problèmes potentiels et de mettre en place une stratégie de gestion des mises à jour.

1.8 Description d'une situation de travail ayant nécessité une recherche

Pour améliorer et rendre professionnel votre paragraphe, vous pouvez le réorganiser et l'éditer pour qu'il soit plus clair et fluide. Voici une version améliorée :

"La réalisation de ce projet a nécessité de nombreuses recherches pour résoudre divers défis techniques. Pour l'affichage des pop-ups en utilisant le framework MUI, j'ai consulté la documentation officielle de MUI, en particulier pour la gestion des pop-ups, dont voici le lien : [lien MUI](https://mui.com/base-ui/react-number-input/).

En ce qui concerne la manipulation d'e-mails en Java, j'ai utilisé la documentation du site Baeldung, une ressource précieuse pour les opérations liées à l'envoi d'e-mails en Java : [lien Baeldung](https://www.baeldung.com/java-email). De plus, j'ai consulté la documentation de Jean-Michel Doudoux, qui est également une excellente source d'informations pour la programmation Java, notamment pour l'utilisation de JavaMail : [lien Jean-Michel Doudoux](https://www.jmdoudoux.fr/java/dej/chap-javamail.htm).

Pour la gestion des bases de données en SQL, j'ai utilisé le site SQL.sh, qui propose des tutoriels et des ressources pour apprendre et travailler avec SQL : [lien SQL.sh](https://sql.sh/).

Enfin, pour la gestion de la localisation et de la traduction (i18n), j'ai utilisé la documentation fournie par i18next, un outil utile pour la prise en charge de multiples langues dans une application : [lien i18next](https://www.i18next.com/).

L'ensemble de ces ressources m'a permis d'améliorer mon projet et de le présenter de manière professionnelle."

Conclusion

Mon stage a été une expérience exceptionnelle, dépassant largement mes attentes initiales. Travailler sur un code existant avec les frameworks Spring Boot et React a été un défi captivant, m'offrant des compétences cruciales en développement web. J'ai appris à décortiquer le code existant grâce au "reverse engineering" et à maîtriser des outils essentiels tels que git et NPM pour la gestion du projet.

La création d'une API REST, la communication entre le backend et le frontend, la conception de SPA, et l'utilisation de React et Material pour les formulaires et le filtrage ont également enrichi mon savoir-faire. Au-delà des compétences techniques, cette expérience m'a immergé dans le monde professionnel, avec ses contraintes de temps et ses exigences changeantes, renforçant ma détermination à poursuivre une carrière en développement web, tout en privilégiant le travail en équipe.

Je tiens à exprimer ma profonde gratitude envers l'entreprise qui m'a chaleureusement accueilli et m'a poussé intellectuellement. Mon stage a été une étape déterminante dans mon parcours, me préparant à relever avec succès les défis futurs du développement web.