ECOLE PRATIQUE DES HAUTES ETUDES COMMERCIALES

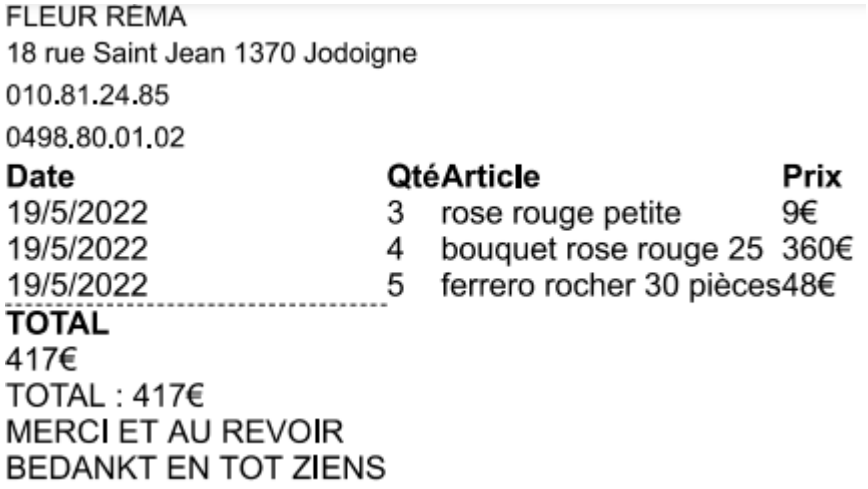
Avenue du Ciseau, 15

1348 Louvain-la-Neuve

Développement application desktop : Point de vente et de gestion de stock de fleur

Année académique 2021-2022

Travail de fin d'études présenté en vue de l'obtention du diplôme de bachelier en Informatique et Systèmes orientation Technologie de l'informatique



FLEUR REMA
18 rue Saint Jean 1370 Jodoigne
010.81.24.85
0498.80.01.02

Date	Qté	Article	Prix
19/5/2022	3	rose rouge petite	9€
19/5/2022	4	bouquet rose rouge 25	360€
19/5/2022	5	ferrero rocher 30 pièces	48€

TOTAL
417€
TOTAL : 417€
MERCI ET AU REVOIR
BEDANKT EN TOT ZIENS

Étudiant : LUK Brian

Rapporteuse : VROMAN Marie-Noël

Remerciement

Je tiens à remercier toutes les personnes qui ont contribué et qui m’ont soutenu au cours de mon travail de fin d’étude et qui m'ont aidé lors de la rédaction de ce rapport.

Tout d'abord, je tiens à remercier Mme Dochart, ma cliente ainsi qu’à tous les employés de l’enseigne « Fleur Réma » pour l’idée et le temps qu’ils m’ont consacré pour me décrire leurs besoins ainsi que pour leurs retours.

J'adresse également mes remerciements à ma promotrice de stage, Mme Marie-Noël Vroman. Sans elle, je n’aurais pas pu en arriver ici. Ses aides et ses conseils m’ont énormément aidé, je lui en suis très reconnaissant car elle a toujours pu rapidement se rendre disponible pour répondre mes questions et me faire un retour sur mon avancement du projet.

Enfin, je remercie ma famille et mes amis qui m’ont soutenu dans ce travail, mais aussi tout au long de mon apprentissage à l’Ephec.

Table des matières

Introduction

Dans le cadre de mon TFE, je vous parlerai du développement de mon application d'un point de vente et de gestion de stock pour une fleuriste. En effet, Mme Dochart est gérante de la boutique « Fleur Réma » depuis plus d'un an et elle m'a fait part de ses soucis de caisse enregistreuse lors de ses débuts et m'a chargé de trouver une solution à son problème.
Effectivement, elle souhaitait à ces débuts, un logiciel sur son ordinateur qui, en plus de servir de caisse enregistreuse, était capable de gérer les stocks des fleurs qu'elle possédait afin d'avoir un aperçu direct de son stock.
La gérante a déjà eu l'occasion de tester certains logiciels de point de vente sur internet mais aucun ne la convenait, les problèmes qu'elle rencontrait personnellement étaient :

- Leur complexité : les logiciels de point de vente ont énormément d'option car leur but est de satisfaire un maximum de besoins, de ce fait, ces logiciels ont en plus d'avoir besoin d'un paramétrage important, s'avèrent être assez compliqué à utiliser

- Leur prix : beaucoup de ces logiciels offrent au maximum 1 mois gratuit et deviennent ensuite payant avec un système d'abonnement à payer par mois qui peut revenir cher. En outre, si on part vers les solutions avec de vrais logiciels et non des applications, le prix d'un pack d'une caisse enregistreuse avec un gestion de stock peut s'élever à des milliers d'euros (prix des machines)

- Leur besoin d'une connexion constante : la majorité des logiciels de point de vente avec une démo sont des applications web qui exigent une connexion internet constante, cela peut s'avérer problématique dans le cas où elle perd sa connexion

C'est pourquoi j'ai proposé à Mme DOCHART une application multiplateforme de bureau qui satisferait uniquement ses besoins pour que celle-ci ne se retrouve pas submergée par un amas de fonctionnalités qui ne lui seraient pas utiles. Cette application ne demanderait aucune connexion car elle sera liée à la base de données qui elle sera en local.

Pour le développement de mon application, j’ai appliqué la méthodologie Agile scrum avec des réunions avec la cliente toutes les 2-3 semaines (sprint review) afin qu’elle puisse voir l’avancé de l’application, ainsi que de tester les fonctionnalités de celui-ci.

Bien entendu, avant même de commencer le développement de la solution, j’ai élaboré avec la cliente un cahier des charges avec toutes les fonctionnalités importantes dont elle avait besoin et les user stories. J’ai également essayé au maximum de prendre rendez-vous avec ma rapporteuse Mme VROMAN au moins une fois toutes les 3-4 semaines pour faire le point sur l’avancée de mon application et les retours de ma cliente. Elle m’a énormément aidé et m’a suggéré beaucoup d’idées que j’ai faite parvenir à ma cliente.

Vous retrouverez sur les pages ci-après une synthèse de mon cahier des charges.

Besoins de la cliente

Pour les besoins de la cliente, elle a d'abord décrit ses besoins pour l'application sous forme de liste que je vais énumérer ci-dessous que j'ai ensuite retranscrits en user stories dans les pages suivantes :

- Application qui nécessite aucune connexion (application offline)
- Simple d'utilisation
- Fonction de caisse enregistreuse avec une caisse, des images et un moyen d'imprimer les tickets de caisse
- Moyen de notification lorsque le stock d'un produit passe en dessous d'un certain seuil
- Fonction pour modifier les données d'un produit
- Fonction pour ajouter un produit dans la caisse
- Un graphique qui montre les données des ventes
- Modification de l'état d'une vente (ex : passer de 'réservation' à 'annuler')

Fonctionnalités des user stories

US1 – connexion

En tant que gérante du magasin, je dois pouvoir d'abord m'identifier pour pouvoir utiliser l'application afin d'avoir une sécurité pour que tout le monde ne puisse pas utiliser l'application.

Fonctionnalité

Pour que l'utilisateur ait accès à toutes les fonctionnalités de l'application, celui-ci doit d'abord entrer le bon identifiant, ainsi que le bon mot de passe.

Si l'utilisateur se trompe d'identifiant ou/et mot de passe, la page se réinitialise jusqu'à ce qu'il réussisse.

US2 – caisse enregistreuse

En tant que gérant du magasin, je dois pouvoir être capable d'utiliser une fonction de caisse enregistreuse de l'application afin de faire le compte sur les produits qu'achètent mes clients. Les produits achetés seront décrémentés dans *L'US4* gestion de stock*.

Fonctionnalité

L’interface de cette user story est séparée en deux parties principales :

- La partie de droite qui affiche les images et en dessous de chaque image, se trouveront le nom du produit, leur prix et leur quantité actuelle.

- La partie gauche affiche quant à elle les noms des produits sélectionnés, le nombre du produit que le client achète, le prix des produits par rapport au nombre ainsi que le prix total en dessous et un bouton pour imprimer le ticket.

Image 1 : maquette de l'interface de la caisse enregistreuse

L’utilisateur doit être capable de choisir le nombre d’articles achetés par le client. Une fois que le ticket a été imprimé, le nombre de stock des produits affiché sera modifié et décrémenté par rapport au nombre d’articles achetés.

L’utilisateur sera également capable de changer les produits actuellement affichés avec un système de pagination où chaque page contient une quinzaine d’articles.

Les produits seront séparés en plusieurs catégories (fleur, consommable et décoration) qui seront visuellement différenciés par des couleurs, l’utilisateur doit être capable de choisir une catégorie pour que cela affiche uniquement les produits qui appartiennent à la catégorie choisie.

Pour éviter toute erreur, l’utilisateur ne sera pas capable de choisir un nombre qui est supérieur au nombre de stock que l’article possède actuellement, si l’utilisateur choisit un nombre qui est supérieur, le choix ne sera pas sauvegardé dans la caisse.

L’utilisateur ne sera également pas capable de choisir un produit donc le stock est à 0, un message ‘produit non disponible’ sera affiché si celui-ci clique sur un article qui ne possède plus de stock.

US3 – reçus de caisse

En tant que gérant du magasin, je dois pouvoir imprimer des reçus de caisse afin de donner aux clients une preuve de leurs achats en cas de remboursement.

Fonctionnalité

En cliquant sur le bouton imprimer qui se trouve sur *l’US2* caisse enregistreuse*, l’utilisateur est capable d’imprimer un reçu de caisse pour le client. L’utilisateur sera également capable d’avoir un aperçu du ticket avant son impression.

Le reçu doit afficher les informations de base du magasin (nom de l’enseigne, l’adresse et le numéro de téléphone), les noms des produits achetés, le nombre d’articles achetés, le prix en fonction de l’article * le nombre achetés, la date de l’achat, ainsi que le prix total de tous les articles.

US4 – gestion de stock

En tant que gérante du magasin, je dois pouvoir avoir un aperçu visuel des stocks de produits restants afin de savoir le nombre de produits qu’il me reste. Le nombre doit être visible facilement.

Fonctionnalité

Sous *l’US2* caisse enregistreuse*, se trouve le nombre d’articles qui restent en stock pour chaque produit, ce nombre est décrémenté à chaque impression de ticket.

Un produit dont le stock est à zéro n’est plus disponible, un message « produit plus disponible » apparaît pour avertir l’utilisateur quand il essaie de sélectionner cet article dans la caisse.

Un produit avec un stock assez bas est surligné d’une couleur voyante pour avertir l’utilisateur.

US5 – ajouter produits

En tant que gérante du magasin, je dois pouvoir ajouter un produit dans l'application de manière simple et rapide.

Fonctionnalité

L'utilisateur doit remplir le formulaire du produit à ajouter, il doit contenir le nom du produit, le prix, un lien url vers l'image, la quantité en stock ainsi que la date et heure.

Le champ du lien de l'image est rempli de base sur une « image non-disponible » au-cas où la gérante ne possède pas d'image.

En validant le formulaire, un message de confirmation apparaît pour avertir l'utilisateur de l'ajout.

US6 – modifier produits

En tant que gérante du magasin, je dois pouvoir modifier et/ou supprimer un produit et ses informations si elles sont erronées ou si je dois appliquer une mise à jour des données.

Fonctionnalité

L'application doit fournir une liste de tous les articles en affichant leur nom, leur prix, leur stock et leur catégorie.

La liste contient également une barre de recherche qui permet de retrouver plus facilement les produits que l'utilisateur désire modifier.

En sélectionnant l'article voulu, cela renvoie une page de formulaire similaire à la page d'ajout de *l'US5* ajouter produits* avec les champs préremplis avec les données du produit.

L'utilisateur peut donc modifier les données et valider à l'aide d'un bouton, un message de confirmation apparaît après la validation des modifications. Les données qui peuvent être modifiées sont le nom, le prix, l'image, la catégorie et le stock.

Un bouton pour supprimer les produits existe également, une fois supprimés, le produit n'apparaît plus dans la caisse et se retrouve dans un état « indisponible ». Il est possible de restaurer un produit « indisponible » dans *l'US10* restauration des produits*

US7 – notification

En tant que gérante du magasin, je dois pouvoir avoir un moyen de notification qui me prévient lorsque la quantité restante d'un produit passe sous un certain seuil afin de pouvoir refaire le stock auprès d'un fournisseur.

Fonctionnalité

Un point rouge doit être présent dans la partie « caisse enregistreuse » de l'application, cet indicateur doit être assez visible pour que l'utilisateur le remarque facilement (couleur voyante). Ce point est présent uniquement si le stock de produit est suffisamment bas, sous un certain seuil.

US8 – bloc-notes

En tant que gérante du magasin, je dois avoir une partie bloc-notes dans l'application pour pouvoir écrire des informations et les sauvegarder. Les produits avec peu de stock sont ajoutés de base dans le bloc-notes.

Fonctionnalité

Un bouton pour pouvoir choisir le fichier à lire est présent dans le bloc-notes, une fois le fichier texte sélectionné, l'application affiche le contenu du fichier texte.
Un champ de texte est présent pour pouvoir écrire les informations et les sauvegarder.

Les produits qui ont un seuil de produits suffisamment bas sont automatiquement ajoutés dans le champ de texte du bloc-notes, cela permet à la gérante de posséder immédiatement une liste exhaustive de tous les produits à devoir refournir sans les ajouter elle-même ou les compter.

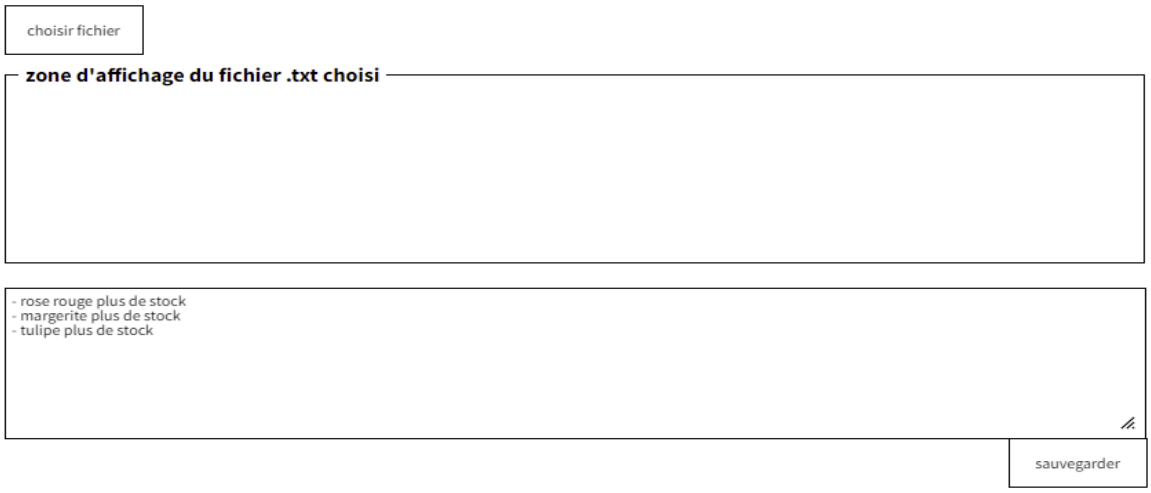


Image 2 maquette de l'interface du bloc note

US9 – historique des modifications

En tant que gérante du magasin, je dois pouvoir avoir une liste de toutes modifications effectuées pour pouvoir retrouver des informations passées.

Fonctionnalité

Affichage de la liste de toutes les modifications des produits effectués, contient le numéro d'identification du produit, le nom, la catégorie, le prix, le stock, le statut ainsi que la date et heure de la modification.

Une barre de recherche est présente pour pouvoir retrouver plus facilement les articles.

Un bouton pour supprimer définitivement l'historique du produit est présente.

US10 – restauration des produits

En tant que gérante du magasin, je dois pouvoir faire revenir un produit supprimé en cas de réapprovisionnement de l'article

Fonctionnalité

Affichage de la liste de tous produits supprimés, contient le numéro d'identification du produit, le nom, la catégorie, le prix, le stock, le statut ainsi que la date et heure de la suppression de l'article.

Une barre de recherche est présente pour pouvoir retrouver plus facilement les articles.

Un bouton « restaurer le produit » permet de ramener le produit supprimé dans la caisse.

US11 – liste des commandes

En tant que gérante du magasin, je dois pouvoir voir toutes les ventes effectuées afin de savoir les profits faits.

Fonctionnalité

Affichage de la liste de toutes les commandes, les informations affichées sont le numéro d'identification de la commande, le total du paiement, la date et heure de la commande, ainsi que l'état de la commande (payé, réservation et annulation).

Une commande annulée n'est pas prise en compte dans le total des gains dans le graphique de la *US11* graphiques des ventes*

La liste est séparée par un système de pagination où chaque page contient une vingtaine de commande.

Un bouton de sélection est présent pour pouvoir changer à tout moment l’état de la commande en cas d’annulation d’une commande (ex. : passer d’un état réservation à annuler).

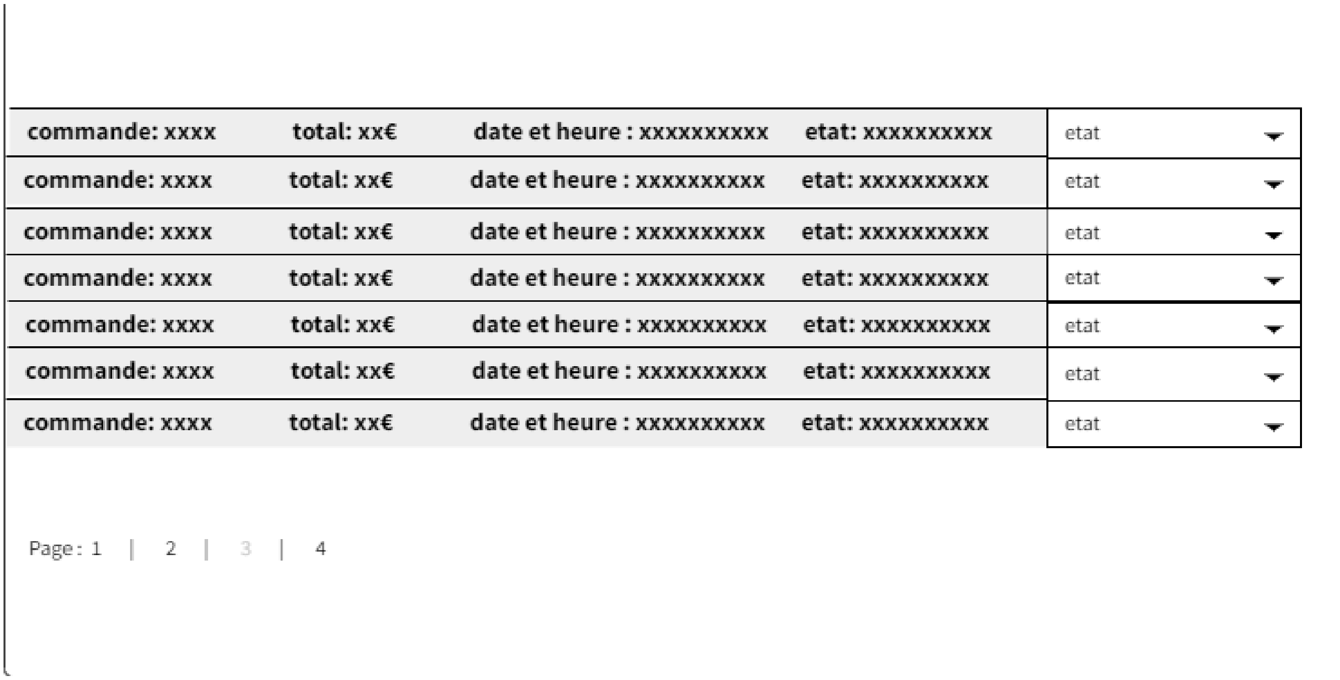


Image 3 : maquette de la page des commandes

US12 – graphiques des ventes

En tant que gérante du magasin, je dois pouvoir afficher des graphiques de mes ventes pour avoir un aperçu clair de mes profits.

Fonctionnalité

L’interface affiche plusieurs graphiques des ventes :

- Un graphique par semaine (un graphique des profits par jour en 1 semaine)
- Un graphique par mois (en un an)
- Un graphique par année

Les graphiques prennent en compte uniquement les commandes donc leur état (*US11* liste des commandes)* sont vendu ou réservé, les commandes annulées ne sont pas ajoutées dans le total des ventes du graphique.

Choix techniques

Choix du langage

Une fois le cahier des charges écrit avec la cliente, j'ai dû me pencher sur les choix technologiques pour mon application. Ceux qui ont retenu le plus mon attention sont :

- Java
- C#
- Python
- Javascript

Voyons dans un premier temps quelles sont les possibilités, avantages et inconvénients de chacun d'entre eux.

Java

Java est un des langages orientés objet le plus utilisé dans le monde qui fonctionne bien pour les applications desktop et qui est assez simple à prendre en main. Il a l'avantage de sortir un fichier .jar exécutable pour utiliser l'application et qui permet donc un déploiement facile de l'application.

L'inconvénient avec Java, est que le GUI proposé par Swing ou SWT (Standard Widget Toolkit) n'est pas très attractif, il est également assez lent comparé à d'autre langage comme le C ou le C++.

De plus, Java est un langage qui consomme énormément et est plus adapté pour des applications petites et simple. L'avantage est que j'ai étudié le langage et que je possède les bases de celui-ci.

C#

C# est un langage plus facile à apprendre et à coder que le C++ mais reste un langage de haut niveau, il demande moins de mémoire que Java mais en consomme malgré tout. Il possède un excellent IDE gratuit (Visual C# Express) mais uniquement disponible sur Windows.

Le GUI de C# est plus attractif que celle proposé par Java, celle-ci est également assez facile à créer et à personnaliser.

Le désavantage est que C# n'est pas le meilleur langage multiplateforme, il dépend énormément de .NET et n'est pas reconnu pour ses performances.

Python

Python est un langage open-source interprété de haut niveau qui permet le travail en orienté
Objet. La syntaxe est simplifiée et n'est pas compliquée, il s'agit d'un bon langage pour programmer des applications de bureau et possède de nombreux outils pour parvenir aux résultats voulus.

Bien que Python soit un langage interprété, il a été conçu pour être lent. De plus, il ne supporte pas le multithreading car il utilise un mécanisme appelé GIL (Global Interpreter Lock) qui assure le thread unique. Python demande également beaucoup de mémoire.

JavaScript

Bien que JavaScript soit un langage interprété, sa vitesse d'exécution est plus rapide que Java qui est un langage semi-interprété et semi-compilé. C'est un langage facile à apprendre, extrêmement populaire et bien documenté.

Bien que JavaScript soit un langage de programmation web, celui-ci reste populaire pour écrire des applications de bureau. Il possède un très large écosystème de frameworks et d'outils pour n'importe quelle situation.

L'inconvénient est que malgré la quantité astronomique de frameworks à sa disposition, beaucoup d'entre-eux sont loin d'être aussi bons que les outils des autres langages.

Tableau de comparaison

Langages	*Vitesse d'exécution*	*Mémoire*	*Facilité d'apprentissage*	*Popularité*	*Utilisation d'énergie*
JavaScript	Assez rapide	Moyenne	Facile	Très populaire, le plus demandé au monde	Moyenne
Java	Rapide	Lourd	Moyenne	Moins populaire, dans le top 5	Très peu
Python	Très lent	Moyenne	Facile	Très populaire, un des plus demandé au monde	Beaucoup
C#	Rapide	Assez léger	Très facile	Moins populaire, dans le top 10	Peu

Conclusion du choix

Après beaucoup de réflexion, j'ai décidé de partir sur Javascript car il s'agit d'un langage très puissant bien qu'il soit interprété, en utilisant le framework Electron (supporté et développé par GitHub), qui me permet de créer une application de bureau, je peux créer une GUI proche de celui du web qui est plus attractif que celui de Java. Il est également plus facile de personnaliser l'interface à l'aide de l'HTML et du CSS. De plus, Electron est multiplateforme, il fonctionne donc également sur Linux et macOS. Des grosses applications très utilisées comme Discord, Visual Studio Code ou Slack utilise Electron.

Il existe également énormément d'applications de pointe de vente mais en version web, le choix de Javascript me semble donc approprié dans ce cas-ci.

Choix du frontend

React

J’ai décidé de choisir React pour le frontend, à l’aide de la DOM virtuelle, ces performances sont nettement meilleures que les autres frameworks. React est très flexible grâce de ses nombreux packages en Javascript. L’utilisation du JSX qui facilite énormément la programmation de rendu HTML et sa documentation et communauté gigantesque permette de faciliter la recherche d’information.

En outre, ayant déjà utilisé Angular lors de mon stage et de mon projet d’intégration, je l’ai trouvé plus simple à mes yeux car je le maîtrisais davantage. React reste tout de même un des framework les plus populaires et nombreuses sont les sociétés qui l’utilisent. C’est pourquoi, j’ai décidé de me forcer à utiliser cette technologie que je pense ne pas suffisamment connaître.

Choix du backend

Node js

J’ai choisi Node js car il s’associe extrêmement bien avec Electron pour créer des applications de bureau. Il est leader en technologie backend, très performant et rapide, possède une forte présence communautaire pour les développeurs et a une documentation riche pour répondre aux besoins.

Ayant choisi du Javascript pour le développement de l’application, Node js est presque incontournable. Il est également idéal pour ce projet car il associe frontend et backend donc full stack ce qui permet un gain de temps énorme.

Choix de la base de données

MySQL

Je suis parti sur MySQL pour le choix de la base de données car il reste un des gestionnaires de base de données le plus connu et populaire au monde. Celui-ci est maintenu par Oracle et est très rapide et performant. Les grandes entreprises utilisent également MySQL comme Slack qui est codé sur Electron. La combinaison Electron et MySQL fonctionne donc et ceux-ci s’associent parfaitement avec Node js.

Étant donné que MySQL existe depuis plus longtemps que la plupart des autres systèmes de base de données, on a accès à un plus grand nombre de besoins d'assistance. Ceci comprend la documentation réelle, sites de questions-réponses et, bien sûr, tutoriels en ligne.

MySQL est aussi relativement performant pour les récupérations de données des index et des jointures, cependant, cette performance faiblit à mesure que les requêtes augmentent. Ce défaut ne pose pas de problème dans le cas de l’application car celle-ci ne demande que des requêtes assez simples. Le manque de sécurité de MySQL ne pose également pas trop de soucis car l’application ne possède pas de données sensibles car il n’y a aucune table « client ».

Réalisation de l’application

Pour la réalisation de mon projet, j’ai tout d’abord commencé à créer mon backend en connectant une base de données MySQL à partir d’un schéma que j’avais conçu. Au fil du temps, les besoins de ma cliente ont changé et la base de données s’est retrouvée modifiée à plusieurs reprises au cours de la réalisation.

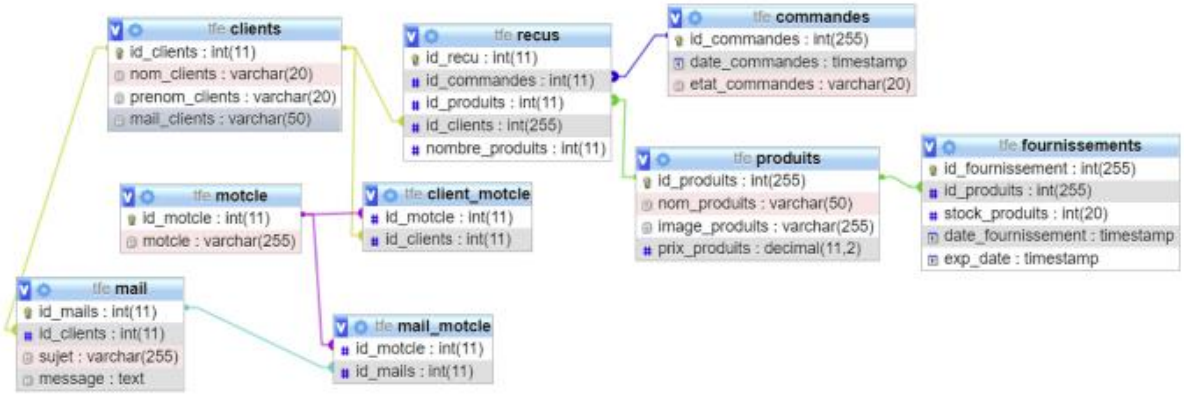


Image 4 : Première version de la base de données

Dans un premier temps, ma cliente a décidé de retirer complétement une des fonctionnalités de départ qui est d’envoyer à certains clients de la publicité ciblée. Le but était que lorsque l’utilisateur ajoutait un client dans la base de données, il pouvait associer des mots clés aux clients et lorsque la gérante envoyait une publicité via mail, elle pouvait préciser des mots clés et le mail était envoyé uniquement aux clients avec les mots clés associés à eux.

Par exemple un client qui avait comme mot clé « rose » et « tulipe », ne pouvait recevoir le mail uniquement si la gérante précisait un des 2 mots clés lors de la rédaction de son mail.

La raison pour laquelle elle a décidé de retirer cette fonctionnalité était que de base elle n’ajoutait que très peu de clients. En effet, la gérante ne propose pas de points de fidélité, la démarche de demander les informations comme le mail ou le numéro de téléphone ne représente donc assez peu d’intérêt aux clients sauf pour les réservations mais qui peuvent se faire juste en appelant l’enseigne.

De ce fait, l’idée de la publicité ciblée a vite été abandonnée par ma cliente. Elle m’a cependant demandé d’autres ajouts dans la réalisation du TFE comme l’ajout de catégorie dans les articles et l’affichage de code couleur dans la caisse pour les différentes catégories de produit.

De plus, lors de mes réunions avec Mme Vroman, qui m’a considérablement aidé en m’apportant des idées dont j’ai fait part à ma cliente qui les a beaucoup appréciées. Elle m’a par exemple conseillé d’encoder dans la base de données les dates partout où c’était possible pour toujours avoir une trace de quand se sont faits les modifications ou les ajouts. Elle m’a aussi donné l’idée d’une table « Historique » qui ajoute dans ses données chaque ajout et modification de produit à l’aide de triggers qui se déclenchent après les événements comme une insertion ou une modification de données dans la table « produits ».

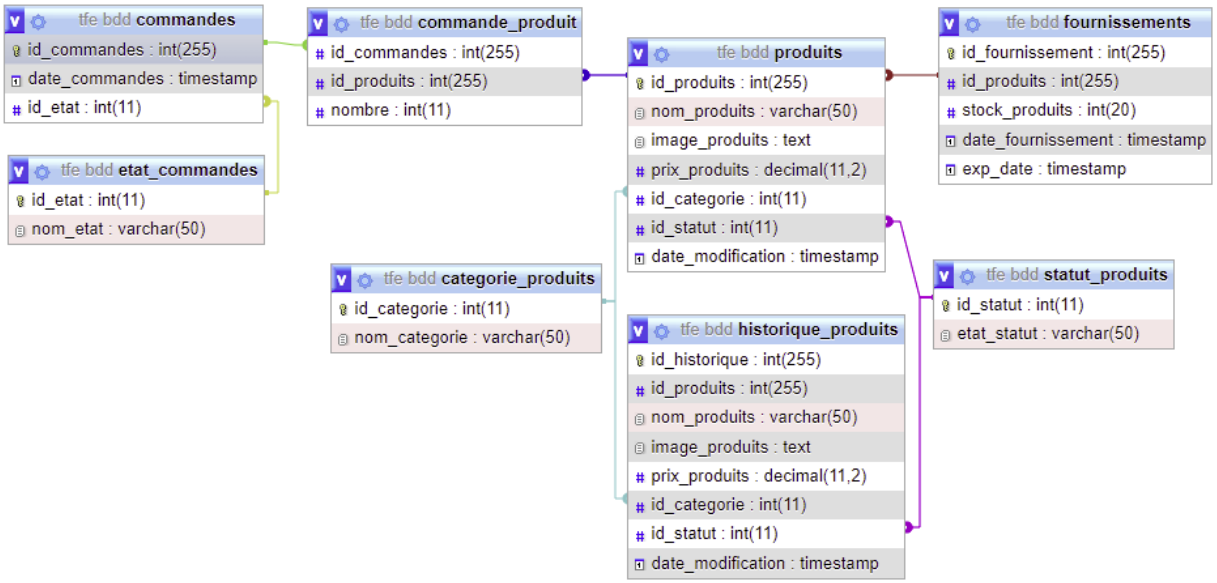


Image 5 : Version finale de la base de données

Interface de connexion

Après avoir fini avec le backend, j'ai commencé le frontend en commençant par faire la partie connexion. Avant d'avoir fait mon choix sur Electron, j'ai d'abord essayé de tester pour m'assurer que cela fonctionnait avec React, n'ayant eu aucun problème j'ai validé mes choix. Malheureusement, quand j'ai commencé le frontend en novembre dans un nouveau projet, l'application Electron se plantait brusquement avec les routers React. Après plusieurs jours de recherche, la solution trouvée était d'utiliser une ancienne version de React retrouvée dans une issue react-router[1] dans GitHub.

J'ai décidé pour le TFE, de ne pas créer de table utilisateur et de juste hard coder le nom d'utilisateur et le mot de passe requis pour accéder à l'application. La raison est que je n'étais pas sûr de la pertinence d'ajouter une table pour n'y avoir que les données d'une personne car uniquement ma cliente utilise l'application. Malgré tout, pour une meilleure sécurité, il est à noter qu'ajouter une table « utilisateur » avec des mots de passe hashés et salés est une piste d'amélioration.

Interface ajout de produits

Ensuite, je suis passé vers la partie d'ajout de produit qui fut assez simple, il s'agissait tout simplement d'un formulaire d'ajout tout de ce qu'il y a de plus normal. J'ai décidé pour les images de simplement mettre des liens url, pour mon stage, les images étaient des données blob, l'avantage de celles-ci était que les images étaient stockées dans la base de données ce qui signifie un affichage certain de ceux-ci contrairement à des liens url qui peuvent expirer.

La raison de ce choix est que les images n'étaient pas si importantes aux yeux de ma cliente qu'aux yeux de mon maître de stage. De plus stocker les données Blob prenait énormément de place dans la base de données. L'ajout de lien URL était donc plus simple.

[1] https://github.com/remix-run/react-router/issues/8252

Interface de la caisse enregistreuse

Une fois le component de d'ajout de produit achevé, je suis parti vers la plus grosse partie et la plus importante qui est la caisse enregistreuse, j'ai d'abord séparé l'interface en deux avec une partie ticket de caisse et une partie affichage de produits. J'ai décidé pour éviter d'avoir une longue liste d'articles à afficher, d'ajouter de la pagination dans la caisse. Après des réunions avec ma cliente, elle m'a alors conseillé un ratio d'une quinzaine d'articles par page et pour encore mieux retrouver les produits, d'ajouter des catégories ainsi que des boutons pour n'afficher que les produits avec la catégorie choisie.

L'affichage des produits et la pagination ne m'ont pas posé trop de problème, mais la partie Modal fut un peu plus compliquée. En effet, il me fallait des boites de dialogue pour afficher les informations et choisir le nombre de stock pour un produit, les modules comme *bootstrap* et *material-ui* n'arrivaient pas à afficher les Modal, j'ai dû donc écrire avant ma classe de caisse, une fonction qui me permettrait d'afficher une boite de dialogue en fonction de l'article choisi.

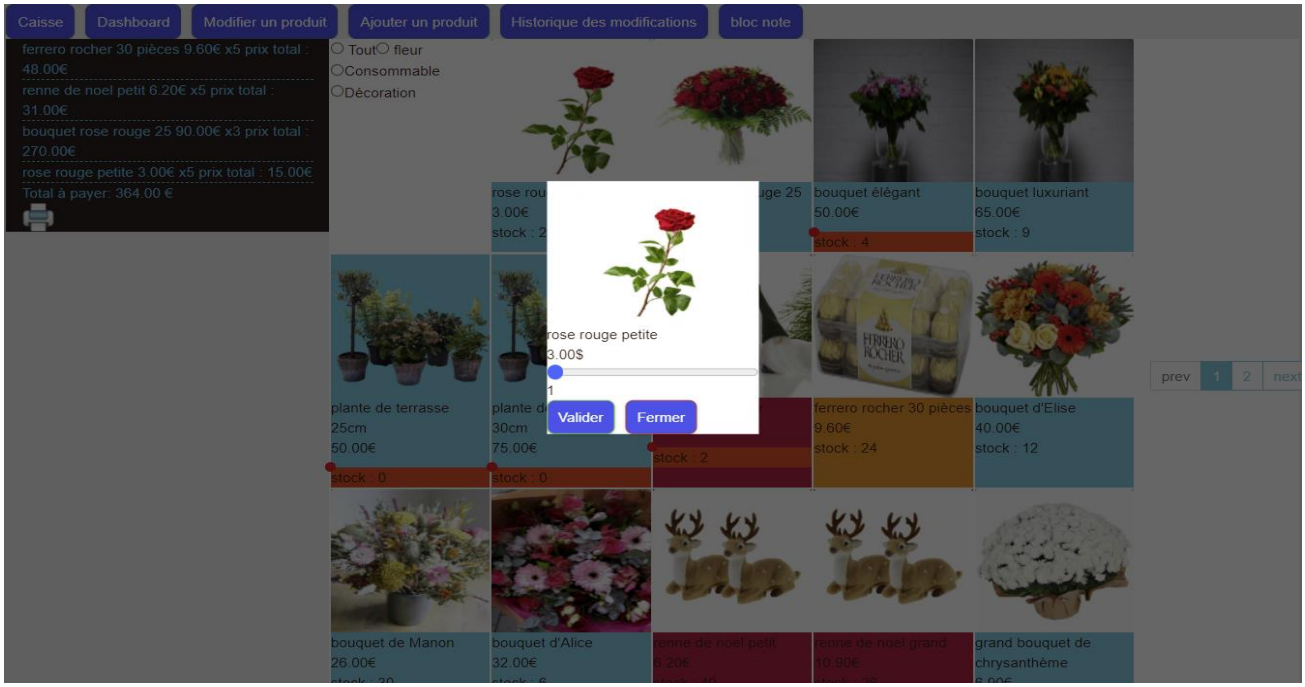


Image 6 : Interface de la caisse enregistreuse avec affichage du Modal

Lors du choix du nombre d'articles achetés, j'ai décidé de me limiter à 10. La raison pour laquelle j'ai choisi ce nombre est par rapport à l'expérience de ma cliente. En effet, les clients achètent assez rarement plus d'une dizaine de fois un même produit. Généralement, un client achète deux à trois fleurs ou part vers les bouquets de fleurs que proposent et préparent les employés du magasin. Par ailleurs, si un client commande plus de dix produits, l'utilisateur est toujours capable d'ajouter plusieurs fois le même article.

Après avoir fini avec la partie Modal, je me suis attaqué aux calculs du prix total dans la caisse, ce ne fut pas extrêmement compliqué. Il m'a juste suffi de stocker les données des prix de chaque article dans la caisse multiplié par le nombre dans une variable *prixTotal* qui était initialisé à 0 à chaque début de caisse. Celle-ci se mettait à jour à chaque ajout d'article que le client s'apprête à acheter. Une fois la caisse finie, il suffit de cliquer sur le bouton d'imprimante pour se rediriger dans l'interface du reçu de caisse.

À savoir que chaque « caisse » corresponde à une commande d'un client, si on change d'interface et qu'on revient vers l'interface caisse, la caisse se réinitialise.
Pour les articles dont le stock est inférieur à 5, j'ai fait en sorte qu'ils soient affichés dans une couleur orange pour qu'on puisse bien voir qu'ils sont proches de 0, de plus j'ai ajouté également un voyant de notification rouge à côté du nombre.

En conclusion, il s'agit de l'user story la plus compliquée et la plus longue car elle représente énormément de petites et de grandes fonctions qui doivent fonctionner entre elles, cette interface est aussi le cœur de l'application, celle où l'utilisateur est censé passer le plus de temps.

Interface ticket de caisse

Etonnamment, la partie imprimante de mon application fut plus compliquée que prévu, les fonctions Javascript d'impression ne fonctionnaient pas à cause d'Electron, il s'agit apparemment d'un bug assez connu[2]. Après avoir testé un bon nombre de modules d'imprimante et de solutions proposées par les utilisateurs de StackOverflow, aucune ne semblait fonctionner pour cause de comptabilité avec Electron pour certains et des modules pas mis à jour régulièrement pour d'autres. Il m'aura fallu plusieurs jours pour coder cette partie le temps de bien lire la documentation du module React-to-Print et de le tester pour confirmer son bon fonctionnement.

J'ai malheureusement perdu énormément de temps pour cette partie, les différents modules ne fonctionnaient pas pour la plupart et ceux qui fonctionnaient n'étaient pas ceux qui étaient les plus adaptés pour ce projet. J'ai dû également visionner de nombreuses vidéos mais certaines étaient trop anciennes et plus à jour.

Interface bloc-notes

Par la suite, j'ai commencé la partie bloc-notes de l'application. J'ai également éprouvé quelques difficultés pour la conception de cette user story, en réalité, je n'avais pas assez réfléchi sur le moyen de sauvegarder les données écrites dans le bloc-notes. Dans un premier temps, j'avais pensé à sauvegarder les données dans un state React. Le problème était que les states ne gardaient pas les données une fois l'interface quittée.

Une des solutions pensées, est celle où j'ajoutais une table « bloc-notes » dans ma base de données, à l'aide d'une méthode POST, il me suffisait d'envoyer les données dans ma table pour garder les informations mais comme pour la partie connexion utilisateur (cf. dernier paragraphe *interface de connexion* p18), je n'étais pas certain de la pertinence d'une table juste pour ajouter du texte. Une autre solution était celle où je pouvais créer un fichier texte, l'avantage de celle-ci était que l'utilisateur pouvait créer plusieurs notes et facilement les retrouver. Ce fut finalement la solution choisie pour « sauvegarder » les données.

[2] https://stackoverflow.com/questions/52333939/window-print-to-generate-pdf-with-electron

Néanmoins, je pense que la solution d'ajouter une table « bloc-notes » n'était pas mauvaise et peut être sujet de point d'amélioration si on creuse un peu plus sur ce sujet. Ma cliente m'avait l'air satisfaite lors de la réunion et m'a demandé de ne pas trop m'étaler sur le sujet car ma solution fonctionnait et que cela lui convenait.

Interface modification des produits

Pour cette partie, je suis parti vers un affichage de liste avec une barre de recherche pour retrouver plus facilement les produits à modifier. Une fois l'article trouvé et choisi, l'application nous redirige vers une page similaire à la page d'ajout de produits, les champs texte sont cependant remplis par les informations de l'article. L'utilisateur est libre de modifier les informations et de les confirmer ensuite, une boîte de dialogue s'affiche pour confirmer la modification.

Image 7 : modification des produits sous forme de liste

Un autre problème que Electron possède et qui n'a pas été résolu est l'affichage de la méthode *alert()*[3] de Javascript. En effet, pour confirmer la modification j'avais prévu de juste afficher une boîte de dialogue créée par la méthode *alert()* malheureusement cette méthode fait planter l'application Electron et ne me permet plus de remplir les formulaires, ce qui est assez dérangeant si l'utilisateur souhaite ajouter ou modifier un produit. Il s'agit d'un problème qui n'est toujours pas résolu par Electron[4] comme indiqué dans *l'issue 19977* du GitHub officiel d'Electron qui est toujours ouvert.

La solution que j'ai trouvée fut celle où j'ai repris la méthode des Modal de la caisse (cf. deuxième paragraphe *interface de la caisse enregistreuse* p19) et d'afficher un message de confirmation à la place des produits. De plus, j'ai également ajouté un bouton qui permet de supprimer un produit dans la caisse ce qui permet de ne pas garder des produits que la gérante n'utilise plus ou qu'elle ne possède plus en stock. Plus tard, le bouton a été modifié pour ne plus supprimer un produit mais de passer son état de « disponible » à « non-disponible », un produit non-disponible n'est pas affiché dans la caisse.

Dans les pistes d'améliorations à ajouter que ma cliente souhaiterait est qu'au lieu d'afficher les produits sous forme de liste, on les afficherait sous forme d'images comme pour l'interface de la caisse (cf. image *interface de la caisse enregistreuse* p19). Ma cliente a beaucoup apprécié l'affichage de la caisse et un

[3] https://www.w3schools.com/jsref/met_win_alert.asp
[4] https://github.com/electron/electron/issues/19977

affichage plus visuel sous forme d'images est plus agréable pour l'œil. Malheureusement, je n'ai pas encore eu le temps de modifier le code pour pouvoir modifier l'interface.

Interface historique des produits

Dans une des réunions avec Mme Vroman, je lui ai montré l'état actuel de mon application. Après avoir vu la partie de la modification des articles, elle m'a fait remarquer qu'il était préférable de toujours garder une trace des anciennes données en cas de besoin. J'ai fait parvenir cette idée à ma cliente qui n'y avait pas pensé et qui l'a validée.

Par la suite, j'ai réfléchi par quel procédé j'allais devoir passer pour pouvoir afficher toutes les anciennes modifications. Après quelques recherches, j'ai pensé à ajouter une table « historique » dans laquelle se trouvent toutes les informations anciennes des produits ainsi que nouvelles. J'ai donc utilisé des méthodes triggers MySQL, il s'agit d'une méthode où une action se déclenche après un événement. J'ai configuré deux triggers, une qui s'active après une insertion dans la table « produits » et une autre après une modification dans la table « produits », l'action est de copier dans la table « historique » les données entrées dans la table « produits ».

Trigger	Event	Table	Statement	Timing
copie_historique	INSERT	produits	INSERT INTO historique_produits (id_produits, nom_...	AFTER
ajout_historique	UPDATE	produits	INSERT INTO historique_produits (id_produits, nom_...	AFTER

Image 8 : triggers de la table « historique »

Une fois que les triggers ont été configurés, je suis parti afficher les données de la table « historique » sous forme de liste comme pour l'interface modification de produit (cf. image *interface modification des produits* p21), j'ai également ajouté un bouton pour pouvoir supprimer définitivement l'historique.

Après avoir entendu l'idée de l'historique, ma cliente m'a demandé de ne pas supprimer complétement les produits dans la caisse, en effet en tant que fleuriste, les fleurs et les bouquets vendus dépendent des saisons, c'est pourquoi il peut lui arriver de réajouter un ancien produit.

J'ai donc ajouté dans l'urgence une table « état » liée à la table « produits » qui indique si un produit est disponible ou non, les produits non-disponibles n'apparaissent pas dans l'interface de la caisse. J'ai ajouté dans l'interface historique une section « produits supprimés », cette section affiche tous les articles non-disponibles et un bouton s'y trouve pour pouvoir le restaurer et les réafficher dans la caisse.

Dans un point d'amélioration, j'ai pensé à ajouter dans les produits un colonne « saison » qui s'affiche en fonction de la date et saison actuelle. Je n'ai pas eu le temps d'ajouter cette fonctionnalité car cela aura un impact sur de nombreuses fonctions dans le code dans les autres interfaces.

En conclusion, ce fut une user story assez difficile à concrétiser car elle n'était pas prévue de base et est venue assez tard au niveau du timing, de plus ma cliente a imaginé d'autres fonctionnalités et pour pouvoir réaliser j'ai dû modifier ma base de données comme l'ajout des états dans un produit. Malgré tout, j'ai beaucoup appris l'utilisation du trigger de la base de données que je ne connaissais pas.

Interface graphique des ventes

Pour cette dernière partie, j'ai d'abord construit sous forme de liste les commandes, en cliquant sur une des commandes, une boite de dialogue s'affiche indiquant des informations supplémentaires de la commande comme les produits achetés, leur prix à l'unité et ainsi que leur nombre.

Dans les informations présente dans la liste, il y a également l'état de la commande qui est indiqué en différentes couleurs (vert pour « vente », rouge pour « annulé » et bleu pour « réservation ») pour mieux les différencier en regardant tout le répertoire. La liste est séparée par une pagination au nombre de maximum dix commandes par page. Il existe également une fonctionnalité qui permet de modifier dynamiquement l'état de la commande en sélectionnant dans les options l'état puis de cliquer sur un bouton « changer état ». Les commandes qui sont annulées ne sont pas prise en compte dans le graphique.

Image 9 : Interface des commandes

Il y a dans cette interface 3 boutons qui nous renvoient vers 3 graphiques différents, un qui montre les 7 derniers jours et qui montre les ventes totales des jours de la semaine, un qui nous montre les ventes des 12 derniers mois et la dernière qui montre les ventes par année.

Pour cette partie j'ai rencontré un problème, mon interface affichait bien les graphiques mais les données n'avaient pas le temps d'être prises en compte et les graphiques restaient vide à moins de « mettre à jour » la page en modifiant par exemple la taille de la fenêtre. Pour pallier le souci, j'ai ajouté un state React *isFetching* dans une de mes fonctions qui renvoie *true* uniquement quand les données ont fini d'être reçues, tant que mon state est *false*, l'html renvoi un chargement et une fois le state retourne *true*, cela affiche mes graphiques. Bien que la solution fût simple, trouver le problème était plus difficile car il n'y eu aucun message d'erreur et que je n'ai pas su trouver le problème en recherchant sur internet. J'ai dû donc deviner moi-même la nature du problème. Une fois trouvé, j'ai rapidement trouvé la solution pour la résoudre.

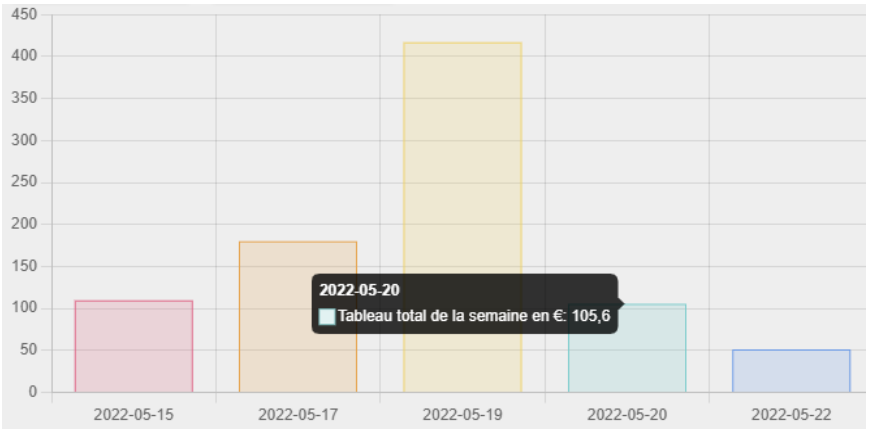


Image 10 : graphique des ventes de la semaine.

Sécurité

Pour la sécurité de l'application, j'ai examiné les différentes données de l'application pour savoir quels pourraient être les impacts de la perte ou vol des données ainsi que la probabilité que cela arrive.

Les données qui sont stockées dans l'application sont les informations des produits, leurs stocks ainsi que les commandes des clients. Ces données représentent peu de valeur car il ne s'agit que d'une petite enseigne. De plus, aucune donnée d'utilisateur n'est sauvegardée, ce qui signifie un intérêt faible de vol car il n'y a aucune donnée personnelle ou sensible. Les informations sont en plus conservées dans une base de données locale sur l'ordinateur servant de caisse et sont accessibles uniquement par la propriétaire. Le peu de sécurité que propose MySQL est tout à fait acceptable dans ce cas-ci.

J'en ai déduit un risque assez faible de vol des données de l'application, à noter que le quartier où se trouve le magasin n'a que très rarement eu des cas de vols. Cependant, un risque de vol de l'ordinateur reste possible. Dans ce cas-ci, le voleur est tout à fait capable d'accéder à l'application. Néanmoins, il doit d'abord passer par la partie connexion de l'application qui est présente et donc le mot de passe et le nom de l'utilisateur est uniquement connu par la gérante du magasin.

Un autre point de la sécurité, est les backups pour la base de données. En effet, il vaut toujours mieux avoir un backup régulier en cas de perte de celui-ci. Heureusement, en utilisant MySqlBackupFTP, on est capable d'automatiser un backup de la base de données locale n'importe où. Dans mon cas, j'ai choisi, par exemple, un OneDrive Business, cela permet de créer un fichier de la base de données tous les jours et d'effacer les plus anciens datant de plus d'un mois.

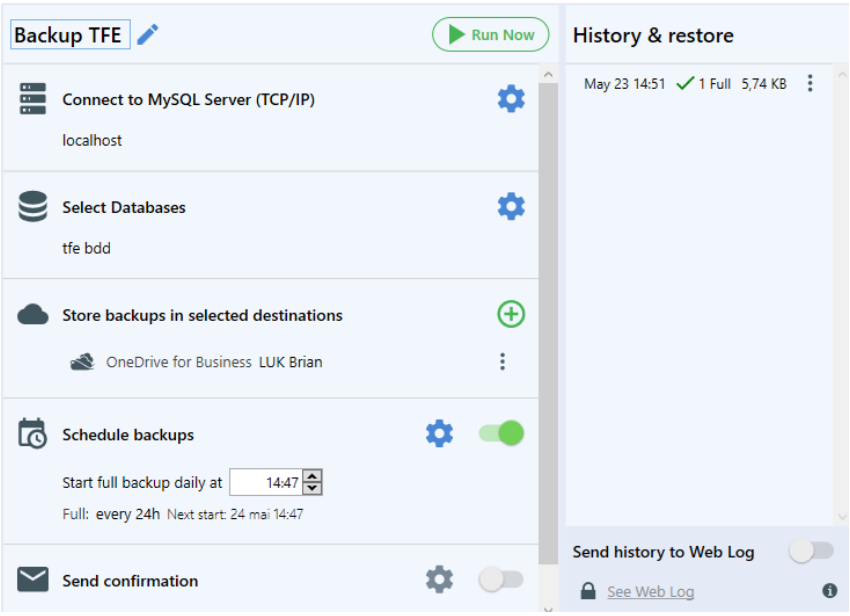


Image 11 : backup de la base de données

En conclusion, on peut dire que le risque global est assez faible, la bonne utilisation des mises en sécurité moyen de l'ordinateur (mise en veille après x minutes d'inactivité, présence de différents mots de passe forts pour chaque logiciel, …etc.) reste le meilleur moyen de sécurité.

Tests

Test unitaire

Pour les tests unitaires, j'ai testé ces différents cas :

- Le rendu de la page sans qu'il n'y ait de problème
- L'affichage de l'interface caisse ainsi que la bonne présence d'un texte HTML présent dans le code
- L'affichage du component du message de confirmation avec les bonnes fonctions
- Le snapshot pour assurer aucun changement d'UI entre 2 snapshots prises
- Une valeur dans un champ de recherche équivalente à ce que l'utilisateur écrit

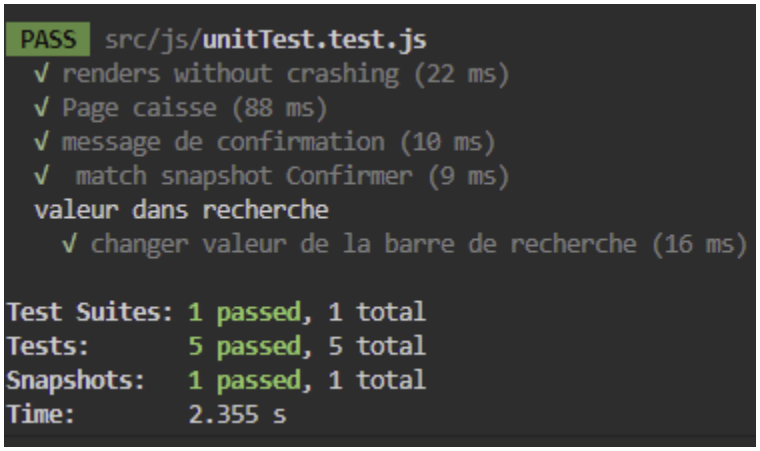


Image 12 : résultats test unitaire

Conclusion

Analyse personnelle

J'ai beaucoup apprécié travailler sur ce TFE, je suis personnellement assez fier de l'application que j'ai pu produire, bien évidement il y a quelques aspects que je regrette un peu de ne pas avoir eu le temps d'améliorer que je reprendrais plus loin dans la conclusion dans le point « Point d'amélioration ».

Je suis content d'avoir pu retravailler sur un projet avec le framework React car malgré avoir utilisé celui-ci dans mon projet Web, je sentais que je ne le maitrisais pas complètement et je préférais un autre framework que j'ai utilisé pour mon stage qui est Angular. Mais maintenant, après avoir pu travailler seul sur ce frontend, je suis fier d'avoir pu mieux le maitriser et l'apprendre plus en profondeur, j'ai appris à mieux apprécier React.

Au terme de mon programme, mon application correspond aux besoins de ma cliente, contrairement à d'autres programmes de point de vente qu'on peut trouver sur internet, il ne demande pas de paramétrage comme *Odoo Point Of Sale* qui est assez compliqué à configurer. L'interface que je propose est plus sobre et moins complexe. De plus, comparé à *extendaGO*, mon application ne demande pas d'abonnement à payer par mois et surtout ne demande pas une connexion internet constante.

J'ai bien évidement rencontré quelques problèmes donc je suis ravi d'avoir pu les résoudre seul comme l'affichage des graphiques qui ne fonctionnait pas et donc j'ai pu identifier et résoudre les difficultés moi-même. Ce ne fut également pas facile de satisfaire tous les besoins de ma cliente qui changeait au fil du temps comme l'ajout de l'historique ou l'ajout de catégorie dans les produits qui n'étaient pas prévu de base.

Malgré tout, je suis fier et soulagé de mon projet et je pense avoir suffisamment abouti mon programme pour le fournir à ma cliente en tant que première version du produit fini.

Point d'amélioration

Importante

- Insertion d'une table utilisateur pour avoir une meilleure sécurité pour la partie connexion avec des mots de passe hashé et salé.

- Ajout des catégories « saisonnière » en fonction des produits de type fleur

Moyenne

- Modifier l'interface « modification Produits » pour afficher sous forme d'images ressemblant à celle de la caisse au lieu d'une liste.

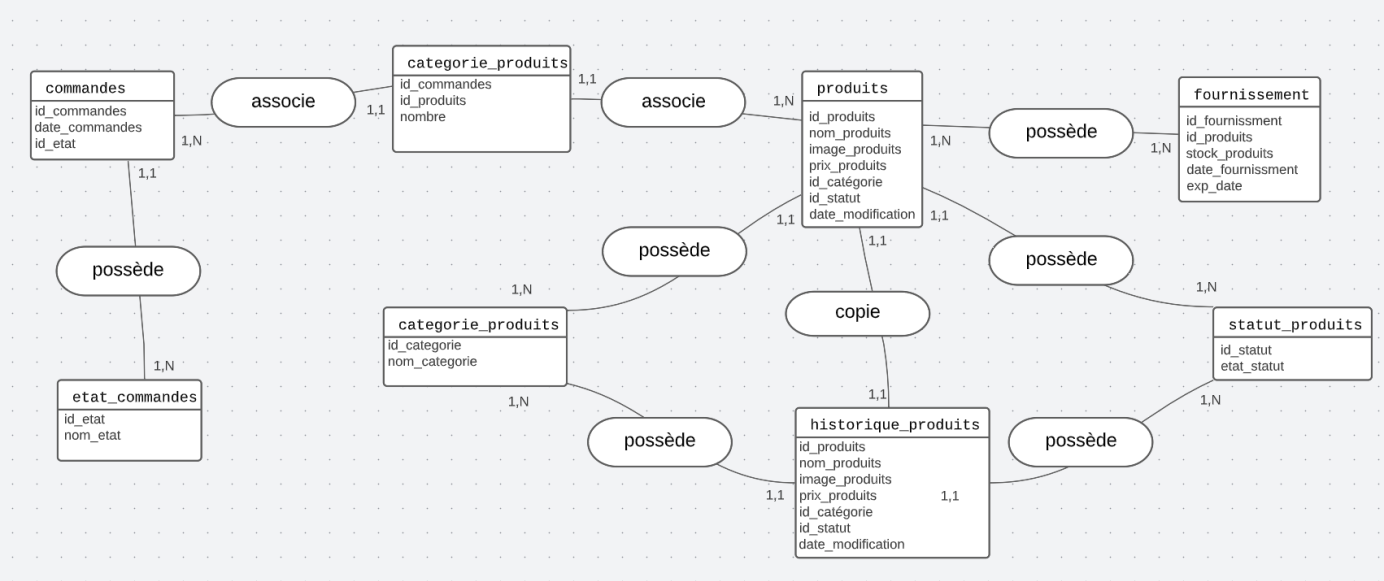
Négligeable

- CSS plus « professionnel » et agréable.

- Afficher sous une meilleure forme les produits achetés par les clients dans l'interface « commande » quand on souhaite les détails des commandes.

Annexes

Schéma entité-association



Code source

Le code source se trouve dans le repository suivant :
https://github.com/BrianLuk10/tfe2021

JsDoc

La JsDoc est comprise dans le code source et se trouve dans le lien suivant :
https://github.com/BrianLuk10/tfe2021/tree/main/document

Documentation API Postman

La documentation API Postman se trouve sur ce lien :
https://documenter.getpostman.com/view/10744771/Uz5CKcoS

Test unitaire

```
12  it("renders without crashing", () => {
13    const div = document.createElement("div");
14    ReactDOM.render(<App></App>, div);
15  });
16
17  test("Page caisse", () => {
18    render(
19      <MemoryRouter>
20        <Caisse></Caisse>
21      </MemoryRouter>
22    );
23    const divText = screen.getByTestId("Test");
24    expect(divText).not.toBeNull();
25    expect(divText).toBeInTheDocument();
26    expect(divText).toHaveTextContent("Total à payer: ");
27  });
28
29  test("message de confirmation", () => {
30    const confirmer = () => {
31      console.log("test");
32    };
33    const valeurTrue = true;
34    render(<Confirmer show={valeurTrue} confirmer={confirmer}></Confirmer>);
35    const divElement = screen.getByText("Action effectué");
36    expect(divElement).not.toBeNull();
37  });
38
39  test(" match snapshot Confirmer", () => {
40    const confirmer = () => {
41      console.log("test");
42    };
43    const tree = renderer
44      .create(<Confirmer show={true} confirmer={confirmer}></Confirmer>)
45      .toJSON();
46    expect(tree).toMatchSnapshot();
47  });
48
49  describe("valeur dans recherche", () => {
50    it("changer valeur de la barre de recherche", () => {
51      const { queryByPlaceholderText } = render(
52        <MemoryRouter>
53          <ModifierProduit></ModifierProduit>
54        </MemoryRouter>
55      );
56      const searchInput = queryByPlaceholderText("Rechercher...");
57      fireEvent.change(searchInput, { target: { value: "test" } });
58      expect(searchInput.value).toBe("test");
59    });
```

Bibliographie

Choix technique

- Greenlab, *Energy Efficiency across Programming Languages*,https://greenlab.di.uminho.pt/wp-content/uploads/2017/09/paperSLE.pdf, consulté le 28 juin 2022
- Journal du net, choisir *une technologie de développement en 2020*, https://www.journaldunet.com/web-tech/developpeur/1493621-choisir-une-technologie-de-developpement-en-2020/, consulté le 2 novembre 2021
- Decipherzone, *top programming language for desktop app in 2021*, https://www.decipherzone.com/blog-detail/top-programming-languages-for-desktop-apps-in-2021, consulté le 2 novembre 2021
- ElectronJs, *Quick start Electron*, https://www.electronjs.org/docs/latest/tutorial/quick-start, consulté le 2 novembre 2021
- Zeste de savoir, *vos applications avec electron*, https://zestedesavoir.com/tutoriels/996/vos-applications-avec-electron/, consulté le 3 novembre 2021
- YouTube Eincode, *Electron with ReactJs*, https://www.youtube.com/watch?v=VCl8li22mrA&ab_channel=Eincode, consulté le 3 novembre 2021
- YouTube Formation Facile, *ElectronJs : créer des logiciels avec JAVASCRIPT pour Windows, Mac et Linux*, https://www.youtube.com/watch?v=ReIx2T5DIhU&t=743s&ab_channel=Formationfacilefr, consulté le 3 novembre 2021
- GitHub Electron packager, *Electron-packager*, https://github.com/electron/electron-packager, consulté le 3 novembre 2021
- GitHub officiel Electron, *Electron*, https://github.com/electron/electron, consulté le 3 novembre 2021
- Altexsoft, *Comparing Database Management Systems: MySQL, PostgreSQL, MSSQL Server, MongoDB, Elasticsearch, and others*, https://www.altexsoft.com/blog/business/comparing-database-management-systems-mysql-postgresql-mssql-server-mongodb-elasticsearch-and-others/, consulté le 6 novembre 2021
- Altexsoft, *The Good and the Bad of C# Programming* , https://www.altexsoft.com/blog/c-sharp-pros-and-cons/, consulté le 6 novembre 2021

Sécurité

- Vaadata, *Comment évaluer les risques d'une faille de sécurité ?*, *https://www.vaadata.com/blog/fr/comment-evaluer-les-risques-dune-faille-de-securite/* consulté le : 26 mai 2022
- YouTube V-Tech Channel, *How to Backup MySQL Database using MySQLBackupFTP*, https://www.youtube.com/watch?v=No-FaH_WRlY&ab_channel=V-TechChannel, consulté le 28 mai 2022

- Mysqlbackupftp, *backup automatically*, https://mysqlbackupftp.com/mysql-blog/backup-phpmyadmin-automatically/phpmyadmin-mysqlbackupftp/, consulté le 28 mai 2022
- Sqlbackupandftp, *How to automate MySql Database backups in Windows*, https://blog.sqlbackupandftp.com/how-to-automate-mysql-database-backups-in-windows, consulté le 28 mai 2022

Electron/Javascript

- GitHub, *the input-box lose focus after call window.alert('...')*, https://github.com/electron/electron/issues/19977, consulté le 20 janvier 2022
- YouTube coderJeet, *Reading & Writing Files in Electron JS - Electron Tutorial* , https://www.youtube.com/watch?v=1rDvNDvZrnA&ab_channel=coderJeet, consulté le 23 février 2022
- Electronjs, *dialog*, https://www.electronjs.org/docs/latest/api/dialog, consulté le 20 mai 2022
- Lifesaver codes, *routes not working on production*, https://lifesaver.codes/answer/routes-not-working-on-production, consulté le 20 mai 2022
- Stackoverflow, *Easiest CSS for "red notification badge" with count*, *https://stackoverflow.com/questions/5747863/easiest-css-for-red-notification-badge-with-count*, consulté le 10 mai 2022
- Stackoverflow, *How to get first and last day of the current week in JavaScript*, https://stackoverflow.com/questions/5210376/how-to-get-first-and-last-day-of-the-current-week-in-javascript, consulté le 11 mars 2022
- Javascript plain English, *Building a Point of Sale system with Node & React*, https://javascript.plainenglish.io/building-a-point-of-sale-system-with-node-react-c2c0395ccaca, consulté le 30 novembre 2022
- Npmjs, *Electron pos printer*, https://www.npmjs.com/package/electron-pos-printer, consulté le 14 avril 2022
- GitHub gregnb, *React-to-print*, https://github.com/gregnb/react-to-print, consulté le 18 mars 2022
- Dev.to Francisco Mendes, *How to Create a Modal in React*, https://dev.to/franciscomendes10866/how-to-create-a-modal-in-react-3coc, consulté le 30 janvier 2022
- Stackoverflow, *I have a chart but it is not updating the chart with react chart-js-2*, https://stackoverflow.com/questions/71001503/i-have-a-chart-but-it-is-not-updating-the-chart-with-react-chart-js-2, consulté le 3 mai 2022

Base de données

- Siteground, *What are MySQL triggers and how to use them?*, https://www.siteground.com/kb/mysql-triggers-use/#:~:text=A%20MySQL%20trigger%20is%20a,before%20or%20after%20the%20event., consulté le 10 mars 2022
- Sqlsh, *jointure*, https://sql.sh/cours/jointures, consulté le 29 novembre 2022

- Dev MySQL, *11.2.2 The DATE, DATETIME, and TIMESTAMP Types*, https://dev.mysql.com/doc/refman/8.0/en/datetime.html#:~:text=MySQL%20retrieves%20and%20displays%20DATE,both%20date%20and%20time%20parts., consulté le 8 janvier 2022
- Tutorials point, *MySQL - create event statement*, https://www.tutorialspoint.com/mysql/mysql_create_event.htm#:~:text=A%20MySQL%20Event%20is%20nothing,and%20schedule%20an%20MYSQL%20event., consulté le 29 mars 2022

Test unitaire

- Jestjs, *test snapshot*, https://jestjs.io/fr/docs/snapshot-testing#:~:text=Les%20tests%20snapshot%20sont%20tr%C3%A8s,stock%C3%A9%20%C3%A0%20c%C3%B4t%C3%A9%20du%20test., consulté le 14 mai 2022
- GitHub, *jest Electron hustcc*, https://github.com/hustcc/jest-electron, consulté le 9 mai 2022
- YouTube Grafikart.fr, *React : Chapitre 28, Comment tester ?*, https://www.youtube.com/watch?v=ZEE3AyEep64&t=351s&ab_channel=Grafikart.fr, consulté le 9 mai 2022
- YouTube Grafikart.fr, *Tutoriel JavaScript : Jest*, https://www.youtube.com/watch?v=_9JTTGI9-K0&ab_channel=Grafikart.fr, consulté le 9 mai 2022
- YouTube techsith, *React unit testing with Jest & React-testing-library*, https://www.youtube.com/watch?v=3e1GHCA3GP0&ab_channel=techsith, consulté le 12 mai 2022