|  |
| --- |
| Gestion de stock de montres |

**Dossier de projet de TPI**

**Damien Loup**

**CID4B**

Table des matières

[1 Analyse préliminaire 3](#_Toc168265450)

[1.1 Introduction 3](#_Toc168265451)

[1.2 Objectifs 3](#_Toc168265452)

[1.3 Planification initiale 4](#_Toc168265453)

[1.3.1 Répartition du temps prévu en % 6](#_Toc168265454)

[2 Analyse / Conception 7](#_Toc168265455)

[2.1 Concept 7](#_Toc168265456)

[2.1.1 Conception de la base de données 7](#_Toc168265457)

[2.1.2 Api backend 10](#_Toc168265458)

[2.1.3 Diagramme de flux utilisateur 11](#_Toc168265459)

[2.1.4 Maquettes 12](#_Toc168265460)

[2.2 Stratégie de test 15](#_Toc168265461)

[2.2.1 Tests de bout en bout 15](#_Toc168265462)

[2.2.2 Tests manuels 15](#_Toc168265463)

[2.2.3 Données de test 15](#_Toc168265464)

[2.3 Risques techniques 15](#_Toc168265465)

[2.4 Planification 16](#_Toc168265466)

[2.5 Dossier de conception 18](#_Toc168265467)

[2.5.1 Technologies 18](#_Toc168265468)

[3 Réalisation 18](#_Toc168265469)

[3.1 Dossier de réalisation 18](#_Toc168265470)

[3.1.1 Arborescence 19](#_Toc168265471)

[3.1.2 Base de données 23](#_Toc168265472)

[3.1.3 Communication frontend et backend 24](#_Toc168265473)

[3.1.4 Pages 31](#_Toc168265474)

[3.2 Description des tests effectués 33](#_Toc168265475)

[3.3 Erreurs restantes 36](#_Toc168265476)

[3.4 Liste des documents fournis 36](#_Toc168265477)

[4 Conclusions 36](#_Toc168265478)

[4.1 Points positifs 36](#_Toc168265479)

[4.2 Points négatifs 36](#_Toc168265480)

[4.3 Difficultés 36](#_Toc168265481)

[4.3.1 Mise en place de docker 36](#_Toc168265482)

[4.4 Evolutions et améliorations possibles du projet 37](#_Toc168265483)

[5 Glossaire 38](#_Toc168265484)

[6 Annexes 38](#_Toc168265485)

[6.1 Résumé du rapport du TPI / version succincte de la documentation 38](#_Toc168265486)

[6.2 Sources – Bibliographie 38](#_Toc168265487)

[6.2.1 Documentations officielles des outils utilisés 38](#_Toc168265488)

[6.2.2 Stack overflow pour les compléments 38](#_Toc168265489)

[6.3 Journal de travail 38](#_Toc168265490)

[6.4 Manuel d'Installation 39](#_Toc168265491)

[6.5 Manuel d'Utilisation 39](#_Toc168265492)

# Analyse préliminaire

## Introduction

Ce projet est réalisé dans le cadre du TPI de l’ETML en entreprise chez Abraxas. Il consiste à développer un site web de gestion de stock de montres pour divers magasins. Le système permettra de gérer différents types d’utilisateurs. Les managers sont associés chacun à un magasin et les patrons sont omniscients et peuvent voir tous les stocks de tous les magasins, ainsi que les commandes.

Ce projet a pour but de consolider certaines connaissances, tel que le développement web, soit les langages qui sont distinctement le typescript pour REACT et le python pour FLASK. Il en va de même pour la réalisation des tests qui utilisent la technologie CYPRESS qui permet de simuler un utilisateur humain, afin de vérifier les fonctionnalités sur la base d’un projet débuté en 2023 pendant le stage en entreprise.

## Objectifs

L’objectif du projet est d’avoir un site web utilisable tournant sur docker avec des tests de bout en bout contenant les fonctionnalités suivantes :

* Système de connexion d’utilisateur : Lors de l’ouverture du site, l’utilisateur sera capable de se connecter à son compte afin d’avoir les informations de stock.
* Ajout de stock : L’utilisateur sera capable d’ajouter du stock ou des commandes à son magasin via des fichiers Excel contenant une multitude de montres.
* Droits utilisateurs : Certains utilisateurs sont associés à des magasins et ne peuvent voir les stocks d’autrui, contrairement aux utilisateurs omniscients qui peuvent voir les stocks de tous les magasins présents.
* Les tests de bout en bout couvrent la quasi-entièreté du site web, afin de pouvoir tester toutes ses fonctionnalités.
* Gestion des commandes : Les patrons seront capables d’accepter ou de refuser les commandes en attente et cela ajoutera ou non du stock selon les articles commandés.
* Concevoir un système facilement réutilisable et améliorable permettant des ajouts futurs et de nouvelles fonctionnalités.

## Planification initiale

Une image contenant texte, capture d’écran, nombre, ligne

Description générée automatiquementUne image contenant texte, ligne, nombre, capture d’écran

Description générée automatiquement

Une image contenant texte, nombre, ligne, capture d’écran

Description générée automatiquementUne image contenant texte, ligne, nombre, Tracé

Description générée automatiquement*Une image contenant texte, ligne, nombre, capture d’écran

Description générée automatiquementLa planification initiale est sous forme de diagramme de Gantt sur Excel, ici découpé par 5 séquences par image. Une séquence définit une demi-journée.*

### Répartition du temps prévu en %

|  |  |
| --- | --- |
| Analyse | 10% |
| Implémentation | 40% |
| Tests | 15% |
| Documentation | 25% |

# Analyse / Conception

## Concept

### Conception de la base de données

En ce qui concerne la conception de la base de données, la maquette initiale a été réalisée à l’aide de l’application *db-main*. Ces schémas découpés en deux phases, MCD (Modèle conceptuel de données) et MLD (Modèle logique de données) permettent de définir toutes les tables ainsi que les attributs de celle-ci.

* Utilisateurs (user) : Cette table contient les informations des utilisateurs du site. Elle stocke les informations de connexion et permet d’identifier le rôle de celui-ci. Les rôles définissent les niveaux d’accès et permettent de déterminer les manager et les patrons.
* Magasins (shop) : Contient tous les magasins présents de la chaine. Elle est la table pivot entre toutes les autres gérant des relations avec les utilisateurs, les articles et les commandes.
* Articles (article) : Englobe une liste d’articles uniques pouvant être affecté à des magasins spécifiques. Elle comprend toutes les informations nécessaires aux données d’un article passant par sa description, sa marque ou sa couleur.
* Commandes (order) : Toutes les commandes effectuées par des utilisateurs sont stockés dans cette table, comprenant l’article en question, l’utilisateur qui a passé la commande et le magasin associé. Le nombre d’unités sont stockées aussi.

#### MCD

Ce schéma permet de définir les tables et les liaisons principales entres les tables.

Ici sont définis quels utilisateurs sont managers dans quels magasins.

Chaque magasin a aussi plusieurs articles en stock et 2 d’entre eux pourraient même avoir les mêmes articles mais en quantité différentes selon les stocks indépendants.

La table « order » permet de définir les commandes qui ont été effectuées selon l’utilisateur en question et le magasin précis ce qui permet à plusieurs utilisateurs de commander des articles pour le stock de leur magasin.

Une image contenant texte, diagramme, capture d’écran, ligne

Description générée automatiquement

1 - Schéma mcd de la base de données

#### MLD

Ce schéma-ci définit les liaisons définitives de la base de données et crée des nouvelles tables si besoin (par exemple quand 2 tables sont liée en 0-N ⬄ 0-N).

Il permet aussi de définir les références de clef secondaires et détermine les schémas réels de la base de données.

Ces donc sur la base de ce schéma que la base de données a été créée dans le code.

Une image contenant texte, diagramme, capture d’écran, ligne

Description générée automatiquement

2 - Schémas MLD de la base de données

### Api backend

Cette api, développée en python avec le micro-Framework flask permet de gérer la base de données, afin de mettre en place une connexion utilisateur avec une session, récupérer les magasins, les stocks et tous les besoins du site web.

Afin de la conceptualiser, un schéma a été réalisé afin de définir les routes de base ainsi que les données que chacune d’entre elles doivent récupérer et retourner.

Ceci fait partie de la conceptualisation initiale, certaines routes pourraient donc disparaître du code ou bien être ajoutées.

#### Routes

#### Une image contenant texte, capture d’écran, Caractère coloré, Parallèle Description générée automatiquement

3 - Routes backend et leur fonctionnement

### Diagramme de flux utilisateur

Durant la conception du projet un diagramme de flux a été réalisé montrant le parcours qu’un utilisateur aura lorsqu’il utilisera le site.

Ce diagramme montre non seulement comment les pages interagissent entre elles, mais aussi comment réagit le site en fonction de ce que l’utilisateur fait.

Comme il est possible de voir, lorsque l’utilisateur ouvre le site, il atterrit instantanément sur la page de connexion. S’il envoie des informations erronées, le site enverra automatiquement une erreur pour le lui signaler. Ensuite, pour ce qui est des pages, il est important de détecter si l’utilisateur à une session active afin de rester connecté et fera des appels d’api quand besoin il y aura. A la moindre erreur d’api ou d’un utilisateur qui n’a pas les droits se rendre sur une page précise, une page d’erreur

s’affichera et la décrira.  
Une image contenant texte, capture d’écran, diagramme, conception

Description générée automatiquement

4 - Diagramme de flux sur l'interaction entre les pages

### Maquettes

Dans le but de créer le site web, il est important de réaliser des maquettes qui reflètent le projet final. Ces maquettes ne seront pas forcément exactes par rapport au visuel final du site, mais cela permet de visualiser à quoi il devrait ressembler.

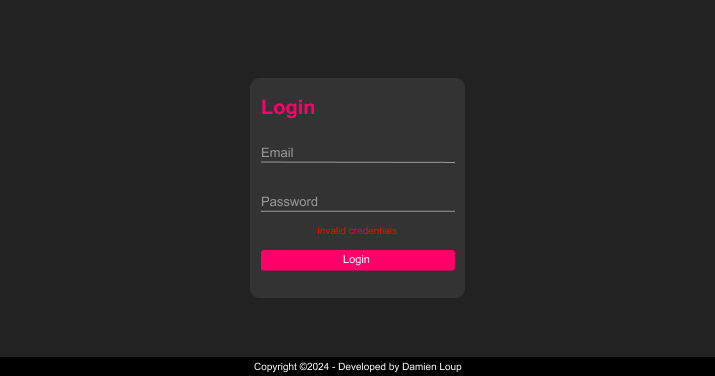
Elles ont été créées à l’aide de l’application *Figma* qui est un logiciel de design permettant de créer des maquettes de toute sorte.

|  |  |
| --- | --- |
| #FF006B | Boutons |
| #333333 | Eléments ressortant du fond |
| #222222 | Fond |
| #FFFFFF | Textes |
| Vert/Bleu/Rouge | Statut des articles et commandes de stock |

Une palette de couleur a donc dû être choisie qui est celle-ci :

*A l’exception du texte « login » dans la page de connexion qui prends la première couleur pour plus d’immersion.*

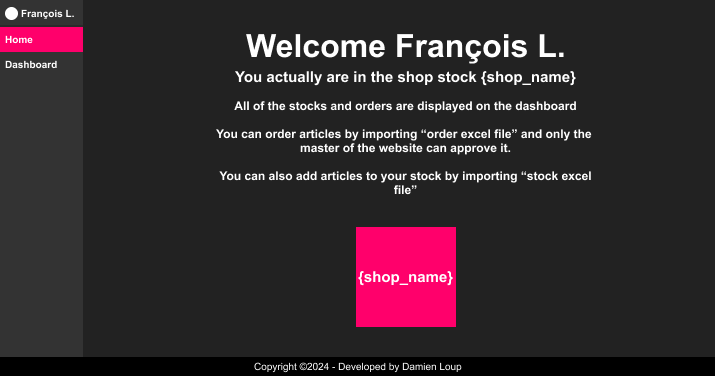
En premier lieu, il y a la page de connexion. Elle s’affiche lorsque l’utilisateur se rend sur le site. L’utilisateur peut donc se connecter à l’aide de son email et d’un mot de passe. Lorsqu’il y a une erreur, un texte rouge apparaît en dessous des champs pour indiquer à l’utilisateur qu’il a commis une faute dans l’email ou le mot de passe.



5 – Maquette page de connexion au site de stock

Une fois connecté, la page d’accueil s’affiche et comprend les informations principales sur l’utilisateur et le magasin actuel de celui-ci définit avec une icône en bas de page.

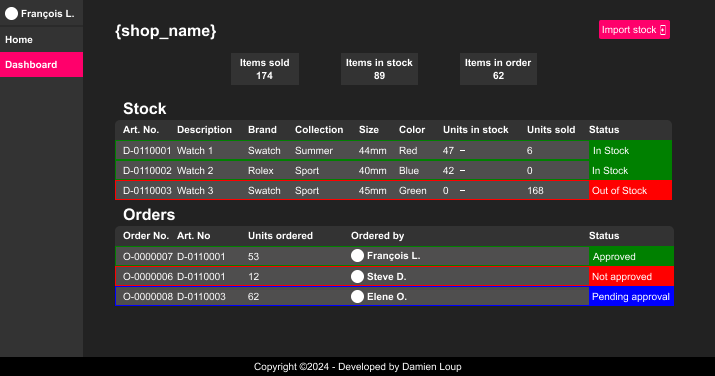
Un volet de navigation est aussi affiché à gauche et permet à l’utilisateur de voyager entre les pages dont il a accès. Par exemple, un manager de magasin à uniquement accès au « Tableau de bord » qui lui permet de voir le stock et les commandes du magasin actuel.



6 - Maquette de l'accueil du site

Ensuite, lorsqu’il se rend sur son Tableau de bord, il peut voir le stock actuel du magasin, et les commandes de stock effectuées ainsi que la personne ayant exécuté celles-ci.

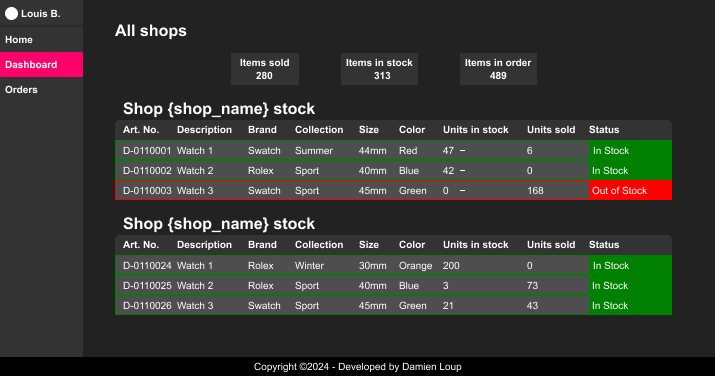
Il peut aussi importer du stock à l’aide du bouton en haut à droite « import stock ». Selon le contenu du fichier Excel, cela peut être un import direct et mettre à jour instantanément le stock ou alors exécuter une commande qui devra être approuvée par le patron de la chaine de magasins



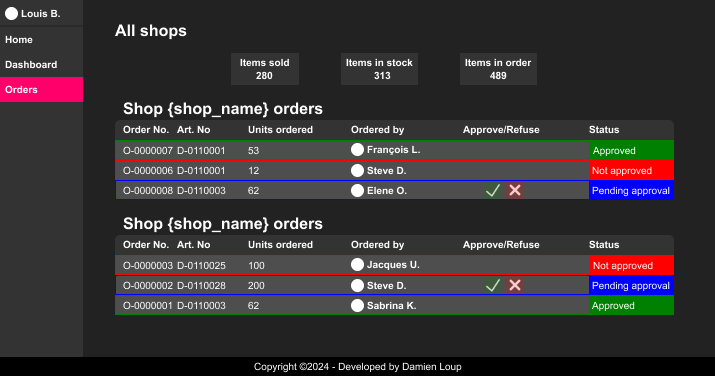
7 - Maquette du dashboard d'un utilisateur manager

La principale différence entre les utilisateurs est que les patron peuvent accéder aux stocks et commandes de tous les magasins.

En ce qui concerne les patrons, ils voient la même chose que les autres utilisateurs à la différence qu’il a tous les magasins de la chaine, cependant, lui a 2 onglets qui sont distinctement « le tableau de bord » afin de voir les stocks de chaque magasin, ainsi que les « commandes », afin de pouvoir approuver celles qu’il juge correctes ou non.



8 - Maquette du dashboard d'un utilisateur patron de la chaine



9 - Maquette de la page des commandes d'un utilisateur patron de la chaine

Sur le haut de la page, 3 éléments définissent le total de tous le stock « articles vendus », « articles en stocks » et « articles commandés » afin de simplifier l’utilisateur à voir son stock total. La valeur est différente selon le compte utilisé (Un magasin ou tous les magasins).

En ce qui concerne les tableaux contenants le stock et les commandes, il est possible de les filtrer en cliquant sur une des en-têtes de colonne. Si l’on filtre par unités commandée, un click sert à définir l’ordre pour trier (croissant/décroissant), le deuxième inversera le tri et ainsi de suite.

Il est aussi possible de cliquer sur le nom du magasin (Ex. « Shop Romanel orders ») pour réduire le tableau afin de pouvoir créer une bonne organisation.

## Stratégie de test

La plupart des tests seront effectués à l’aide de cypress qui permet d’exécuter des tests de bout en bout pour simuler un utilisateur humain. Cypress est une grande partie de ce projet afin de s’assurer du bon fonctionnement de l’application et certains seront fait manuellement afin de s’assurer de la sécurité mise en place.

### Tests de bout en bout

Des tests de bout en bout seront mis en place à la fin du projet afin de garantir une couverture assez globale des fonctionnalités proposées.

Ils permettront de tester chaque fonctionnalité au lancement des scripts ce qui permet une plus grande efficacité.

### Tests manuels

Des tests manuels seront effectués à chaque modification et ajout de fonctionnalité tout au long du projet afin de permettre un avancement contrôlé du développement.

Ceux-ci garantiront une couverture assez exhaustive du code.

### Données de test

Afin de pouvoir tester toutes les fonctionnalités voulues, une liste de données de test est prévue.

Le but étant de pouvoir récupérer ces données afin de pouvoir les utiliser directement sur le site en tant que données de test.

Utilisateurs :

|  |  |  |
| --- | --- | --- |
| Email | Mot de passe | Rôle |
| [louis.t@gmail.com](mailto:louis.t@gmail.com) | Louis1234 | Patron de la chaine |
| [alice.s@gmail.com](mailto:alice.s@gmail.com) | Alice1234 | Patronne de la chaine |
| [françois.l@gmail.com](mailto:françois.l@gmail.com) | F-L1234 | Manager d’un magasin |
| [steve.d@gmail.com](mailto:steve.d@gmail.com) | S-D1234 | Manager d’un magasin |
| [sabrina.k@gmail.com](mailto:sabrina.k@gmail.com) | S-K1234 | Manageuse d’un magasin |

Magasins :

|  |  |
| --- | --- |
| Nom | Utilisateurs |
| Horology Haven | François Lambert, Steve Davis |
| O'Clock | Sabrina K. |

Des fichiers Excel seront aussi utilisés afin de permettre l’ajout ou la commande de stock.

## Risques techniques

Ce projet est effectué à l’aide de Docker. Une application de virtualisation de conteneurs permettant de déployer et tester rapidement et efficacement des applications. Cela tient compte d’un conteneur pour l’application et de l’autre pour les tests cypress.

Le plus grand risque étant le peu de connaissance sur docker lors du lancement du TPI qui pourrait amener à un léger retard sur la planification initiale, ce qui demande d’apprendre d’autant plus comment correctement l’utiliser avec des tutoriels, des vidéos et de la documentation en ligne afin que l’ensemble du projet puisse être utilisé sur n’importe quelle machine.

## Planification

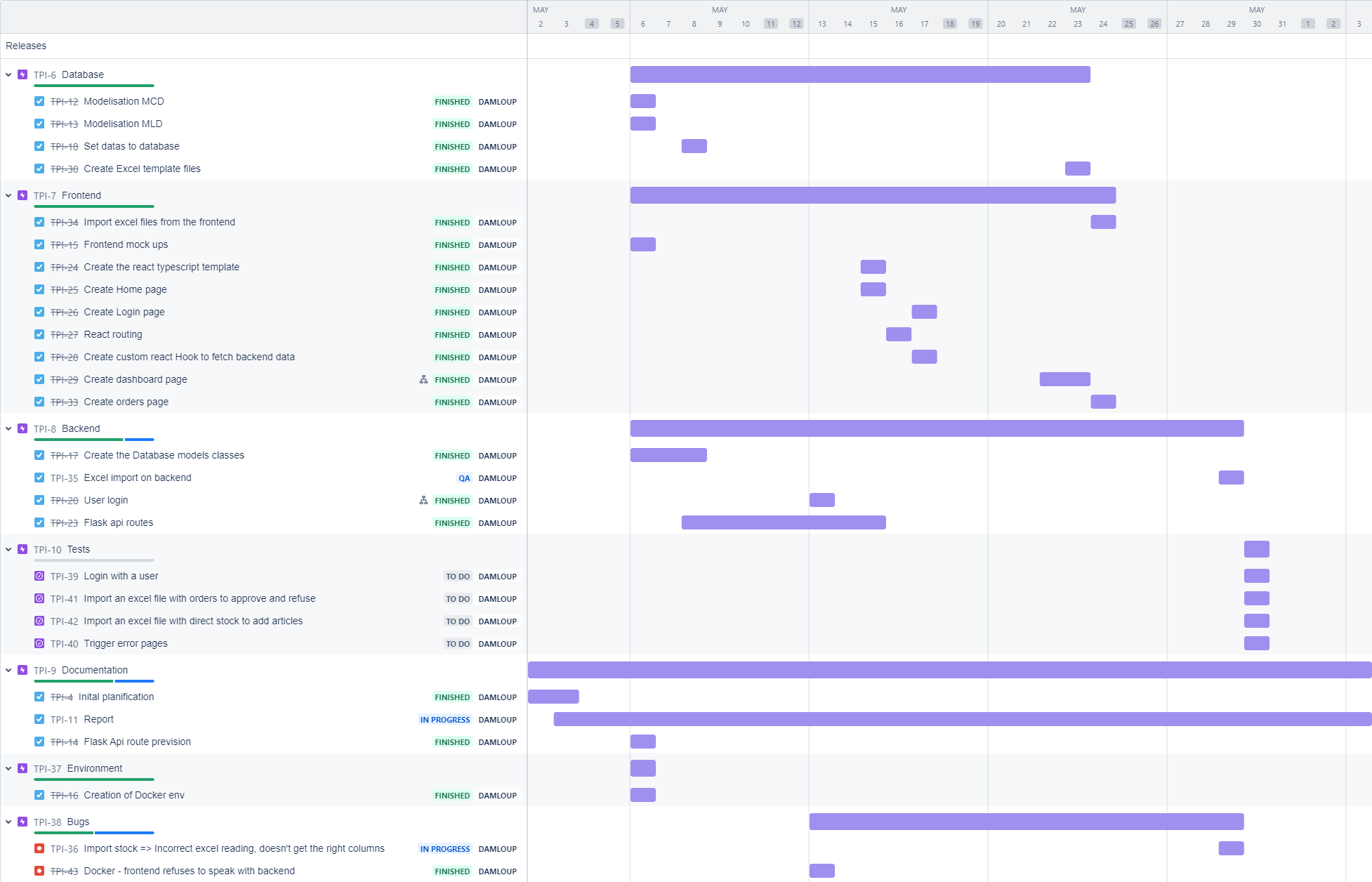
Afin de s’organiser correctement avec des tâches précises, à l’aide de JIRA dans les outils Atlassian, un projet a été créé regroupant tickets et code stocké à l’aide de git. Ce projet contient toutes les tâches et sous-tâches principales permettant la réalisation des fonctionnalités. Les tickets contiennent au moins un titre, un label définissant leur catégorie et deux dates pour définir leur début et fin ce qui permet de les afficher à l’aide d’un diagramme de Gantt.

La méthode utilisée est KANBAN. Cela permet de visualiser chaque tâche en fonction de son statut d’évolution (To do, In progress, QA, Done). La colonne QA permet de réaliser un test manuel sur le ticket réalisé afin de le vérifier. Si le resultat du test est OK, le ticket finit « Done », sinon, le ticket est réouvert ou alors un autre contenant l’étiquette « bug » est ouvert avec un message décrivant l’erreur afin de garder des traces des éléments corrigés ou non.

Une image contenant texte, capture d’écran, logiciel, Icône d’ordinateur

Description générée automatiquement

10 - Tableau KANBAN sur jira

Voici la Roadmap permettant de voir la durée que chaque tâche à prit ou devrait prendre sur le mois :

11 - Roadmap des tâches sur Jira

## Dossier de conception

### Technologies

Les principales technologie et logiciel utilisés pour la réalisation de l’application :

* **Visual Studio Code** pour la conception du code de l’application
* **Docker** pour faire tourner l’application (python et typescript) en local
* **Atlassian** pour la gestion de projet
* **Jira** pour la gestion des tâches
* **Git** pour la gestion et stockage des fichiers de projet
* **Postman** pour tester l’api backend du projet

Les logiciels utilisés pour la documentation :

* **Office**pour la rédaction du journal de travail et du rapport
* **Figma** pour la création de maquettes
* **DB-Main** pour la conceptualisation de la base de données

Matériel à la réalisation du projet

* **1 PC** windows 10 standard avec connection internet

# Réalisation

## Dossier de réalisation

### Langages utilisés

Le projet est basé sur deux principaux langages qui communiquent ensemble.

* **Python :** Utilisé pour le backend, le python permet de faire office office de serveur. Il utilise néanmoins le micro-Framework nommé FLASK permettant de gérer les routes qui servent de pont entre le frontend et le backend. Ce micro-framework est très utilisé pour créer des petits projets et est donc parfait pour celui-ci.
* **Typescript :** il est l’évolution syntaxique et sécurisée du javascript. Ce langage permet de forcer le type des variables, afin d’accroître la sécurité ou la modification inattendue de type de celles-ci. Il permet donc de créer tout le design, les pages et algorithmes dans le frontend. Le framework utilisé est « React » afin de réaliser toute la partie visuelle du projet. React est très utilisé pour faire des sites dû à sa simplicité de créer des pages et des composants amovibles utilisables partout
* **HTML/TailwindCSS :** Ces technologies ont été utilisés pour concevoir le visuel des pages du site, garantissant une présentation conforme aux maquettes.

Etant donné la communication impossible entre le python et le typescriot, des appels d’api sont mis en place afin de les faire s’envoyer des données entre eux à travers le réseau.

### Arborescence

Les répertoires sont séparés de manière à différencier clairement des tests automatiques et de l’application, ainsi que du client (frontend) et du serveur (backend) :

Une image contenant texte, capture d’écran, Police, logiciel

Description générée automatiquement03/06/2024

* Tout d’abord, la surface est constituée d’un dossier « app ».

Il contient l’entièreté de l’application dont le client et le serveur. Des fichiers relatifs à docker sont aussi entreposés rendant l’utilisation de l’application possible afin de mettre en place l’environnement ainsi que les librairies à installer contenues dans le fichier « requirements.txt ».

Le fichier le plus important, « run.py » est le fichier serveur d’entrée. Il permet de lancer l’application en créant un serveur local.

* Ensuite le dossier « tests » permet d’y mettre tout le code servant à lancer les tests automatique cypress.

12 - Arborescence du projet

* Le fichiers « .gitignore » sert à empêcher certains fichiers ou dossiers d’être envoyés dans le dépôt git dû à la sécurité, des dossiers inutiles ou trop lourds ou alors les dossiers contenants les librairies permettant de coder (.venv pour python et node\_modules pour typescript).
* En revanche, « compose.yaml » permet de définir les containers qui seront créés avec docker. Il permet aussi de mettre en place des actions automatique si le mode « watch » est activé lors du lancement de la commande de build.

#### Une image contenant texte, capture d’écran, Police, menu Description générée automatiquementClient

Le client est composé d’énormément de fichiers permettant la mise en place du frontend comme « package.json » qui contient toutes les librairies à installer avant de lancer le projet.

* **public/**: Le dossier contient tous les fichiers de base pour que l’application fonctionne, comme un fichier html qui sera récupéré grâce à react et modifiera son contenu en fonction du code exécuté.
* **src/**: Il contient tout le code source à la création du site. Comme son nom l’indique, « index.tsx » est la base de react et permet la récupération de la balise principale du fichier html.

Ici, il est possible de voir que les dossiers ont été séparés en fonction de leur utilité première :

* **configs/**: pour les configurations globales du site web.
* **resources/** : pour les images et autres fichiers servant à être afficher sur le site
* **utils/** : pour tous les outils typescript hors du code principal, comme des composants de la librairie react personnalisés, des interfaces ou des fonctions globales et servant à être utilisés de partout.
* **pages/** : pour tous les composants react affichants les pages web voulues contenant des algorithmes et du html.

13 - Arborescence frontend

**Router.tsx**: Le composant react le plus important dans ce projet. C’est lui qui permet de changer de page en fonction de l’url sur laquelle se trouve l’utilisateur.

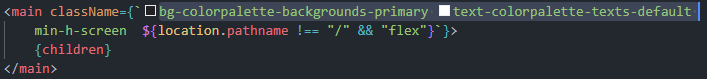
* **Une image contenant texte, Police, capture d’écran, nombre

  Description générée automatiquementConfigurations**

Le dossier de configurations comprend la palette de couleur dans un fichier json et un fichier de configurations en typescript. Ce même fichier json est importé dans les configurations de « tailwind » qui est un framework CSS permettant la simplicité de création du style.

14 - Dossier de configurations frontend

Exemple d’utilisation de la palette de couleur avec tailwind :



* **Une image contenant Police, texte, capture d’écran, Graphique

  Description générée automatiquementRessources**

Les ressources ne contiennent rien de plus qu’une seule image qui permet la création du bouton d’importation de stock.

15 - Dossier de ressources frontend

* Une image contenant texte, capture d’écran, Police, conception

  Description générée automatiquement**Outils**

Les outils contiennent quelque peu plus de fichiers.

Tout d’abord, les fonctions globales, ici « SortArray.ts » permettant de trier les tableaux d’articles et de commandes par colonne.

Ensuite, les interfaces. Celles-ci servent à définir des objets précis renvoyés par l’api en backend. Elles sont utiles pour le typage et la lisibilité du code.

Après cela, il y a un « hook », crochet en français, nommé « useFetch.ts ». Cet élément est le point central entre le frontend et le backend, il est la connexion entre ces deux parties.

Pour finir, il y a les contextes. Dans ce cas-ci, on y trouve   
« UserContext.tsx ». Ce fichier, contrairement aux autres est un composant react. Il est en fait le composant chargé de la gestion de la session utilisateur et du stockage de l’objet « user » à chaque changement de page. Il permet de diffuser cette variable dans tout le code, afin que n’importe quel composant puisse le récupérer.

16 - Dossier d'outils frontend

* **Une image contenant texte, capture d’écran, menu, Police

  Description générée automatiquementPages**

Ce dossier contient tous les composants react qui ont un rapport avec les pages du site web.

Concernant ces répertoires, le dossier « views » possède la base des pages. C’est-à-dire les composants affichés lorsque le « router » les appels. Ces composants eux-mêmes utilisent ceux qui sont contenus dans les dossiers « components » et « includes ».

Ces fichiers affichent l’UI aux utilisateurs et leur permettent de voyager entre les différentes pages.

Les erreurs sont séparées du reste afin de permettre une arborescence propre et compréhensible pour de potentiels futurs développeur qui reprendraient ce projet.

Les composants du dossier « includes » ne sont pas forcément utilisées par les vues, mais peuvent aussi être utilisés pour être affiché en tout temps sur le site, ce qui est le cas du bas de page.

17 - Pages du frontend

#### Une image contenant texte, capture d’écran, Police, nombre Description générée automatiquementServeur

Le serveur est composé de beaucoup moins de fichiers dû à son utilité réduite à communiquer avec le frontend et accéder à la base de données.

Une image contenant texte, capture d’écran, Police

Description générée automatiquementTout d’abord, le dossier « frontend » contient le client react vu précédemment à la seule différence qu’ici, il est compilé.

18 - Arborescence serveur

19 - Frontend compilé

Ensuite, dans le code source du serveur, on y retrouve la base de données, les routes et des outils au même titre que le frontend.

* **Une image contenant texte, Police, capture d’écran, nombre

  Description générée automatiquementOutils**

Ce dossier permet de gérer les outils du serveur. Celui-ci ne contient qu’un seul fichier contenant des fonctions globales en python. Il n’est pas nécessaire au bon fonctionnement du site, mais est utile dans certains cas.

20 - Outils backend

* **Une image contenant texte, capture d’écran, Police, conception

  Description générée automatiquementBase de données**

L’entièreté de la base de données est contenue dans un seul fichier nommé « db.sqlite ». Il utilise le système de gestion de base de données **SQLITE** et permet un développement facile avant d’utiliser de vraie base donnée complètes pour la production (ex. MySQL).

Dans le dossier « models » se trouvent toutes les classes équivalentes à chaque table et leurs liaisons permettant l’accès à la base de données.

21 - Base de données du backend

* **Une image contenant texte, capture d’écran, Police, conception

  Description générée automatiquementRoutes**

Le dossier de routes contient tous les fichiers qui gèrent les routes accessibles depuis un url. Elles renvoient toutes du json que le frontend peut récupérer et lire.

Les fichiers définissent le sujet qu’ils vont traiter du fait de leur nom.

22 - Routes de l'api backend

#### Une image contenant texte, capture d’écran, Police, conception Description générée automatiquementTests

Le dossier de tests contient des fichiers en rapport avec l’arborescence de base de Cypress.

* Le fichier « cypress.config.ts » permet de définir les configurations de base de cypress. Les fichiers en rapport avec docker permettent la virtualisation d’un container, afin de lancer les tests automatiquement.
* Le dossier « data » contient toutes les données excel qui peuvent être importée directement depuis le site web.
* Tous les fichiers contenus dans le dossier « e2e » sont les tests automatiques eux-mêmes.
* Les fixtures sont les données utilisées lors des tests qui contiennent les magasins et les utilisateurs du site.
* Le dossier « support » contient du code typescript réutilisable de plusieurs manières dans les tests.

23 - Arborescence tests cypress

Une image contenant texte, capture d’écran, Police, conception

Description générée automatiquementAfin de permettre la réutilisation du code, il existe un dossier « support » contenant toutes les commandes générales de cypress, les énumérations, des fonctions et des tâches.

Certaines interfaces sont érigées dans les dossiers « interfaces » qui définissent les commandes possibles et certains objets utiles.

Un outil de log est aussi présent afin de suivre la progression des tests.

Pour finir, l’entièreté de ce projet est dotée d’une arborescence logique et facilement réutilisable pour reprendre le projet et tous les fichiers sont utilisés à bon escient.

24 - Arborescence cypress

### Base de données

La base de données doit pouvoir être gérée grâce à des models afin de récupérer et envoyer des données. Ce sont donc des classes en python définies grâce à la librairie « sqlalchemy » qui sont utilisées pour la création et l’interaction de la DB.



25 - Déclaration de la base de données

Voici un exemple de table :

Une image contenant texte, capture d’écran, Police, nombre

Description générée automatiquementUne image contenant texte, capture d’écran, Police

Description générée automatiquement

26 - Schéma de la table "utilisateur"

27 - Modèle "utilisateur"

Dans ce code, on peut y apercevoir tous les champs présents dans la table « user ». A défaut d’être le fichier qui gère la table, il permet entre autres, de la créer lorsqu’elle n’existe pas.

Afin qu’une classe soit définie comme une table de la base de données, elle doit impérativement hériter de « db.Model » qui provient de la même instance de la variable « db » déclarée au départ.

La table « User » hérite aussi de « UserMixin ». Il s’agit d’une classe provenant de la librairie « flask\_login ». Cela permet la gestion de la session utilisateur et n’affecte que cette table précisément.

Chaque champ est défini par un type de valeur et peut être limité en taille, comme pour le nom, l’email et le mot de passe.

Comme toute base de données relationnelle, SQLite n’échappe pas à la règle. Il est donc possible d’avoir accès à des données liées à un utilisateur. Par exemple le magasin auquel il est affecté ou alors les commandes qu’il a passé à l’aide de l’importation de fichiers.

Le champ « shop » de la table « User » est de type « Shop » mais ne fait pas à proprement partie de la table. Cependant, il fait le lien avec la table « Shop » où le même champ au ’’pluriel’’ est déclaré (users).

Une image contenant texte, Police, capture d’écran, nombre

Description générée automatiquementUne image contenant texte, capture d’écran, Police, logiciel

Description générée automatiquement

28 - Schéma de la table "shop"

29 - Modèle "shop"

Comme il est possible de le voir ici, il est possible de récupérer les utilisateurs du magasin en question, les commandes ou même les articles et leur nombre.

Il est possible d’effectuer une recherche d’un utilisateur dans la base de données comme ceci :

### Communication frontend et backend

Afin que le frontend puisse communiquer avec backend, il faut définir une liste de routes que le frontend peut appeler afin d’envoyer des données et/ou d’en recevoir.

Autant dans le frontend que dans le backend, la structure est similaire et possèdent tous des fonctions pour les requêtes d’API.

#### Routes

Afin de structurer les routes, des Blueprint sont créés pour chaque type de route.

Une image contenant texte, capture d’écran, Police

Description générée automatiquement

30 - Ajout de blueprint pour l'application

Les routes principales se trouvent dans le blueprint « main » et celles-ci permettent de retourner directement l’html du site grâce à la fonction « get\_frontend ». Il permet aussi de gérer les erreurs 404.

Une image contenant texte, capture d’écran, logiciel

Description générée automatiquement

31 - Blueprint "main"

Les routes accessibles à l’aide d’une url sont définies grâce à un décorateur contenant « l’Endpoint » et des méthodes d’appel (GET ou POST) permettant l’accès depuis le serveur local.

Pour récupérer un magasin spécifique, il suffit d’appeler l’API et d’y passer l’id de celui-ci. Postman a été utilisé afin de vérifier leur bon fonctionnement.

Une image contenant texte, capture d’écran, Police

Description générée automatiquement

32 - Exemple de route en GET

**Gestion des erreurs**

Les erreurs et succès sont tous définis par des codes spécifiques traductibles :

**200 :** Succès lors de la requête.

**401 :** L’utilisateur n’a pas les permissions nécessaires.

**403 :** L’utilisateur n’est pas autorisé à accéder.

**404 :** La page n’existe pas

Une image contenant texte, capture d’écran, Police

Description générée automatiquementLa quasi-totalité des routes comprennent un décorateur « login\_required ». Celui-ci permet de stopper tout appel d’api venant d’un utilisateur qui n’est pas préalablement connecté.

Lorsque cela arrive, le système de gestion de login redirige automatiquement dans une fonction d’erreur.

33 - Erreur lorsque l'utilisateur est censé être connecté

Cette fonction « unauthorized » renvoie automatiquement un message suivis d’une erreur 403.

La route permettant de mettre à jour une commande n’est accessible que lorsque l’utilisateur à le rôle patron. Dans ce cas, le code récupère l’utilisateur actuel et vérifie son rôle. S’il n’est pas patron, la route renverra un message d’erreur suivis d’un code d’erreur 401.

Malgré tout, si l’utilisateur est un patron, le code renvoyé sera 200.

Une image contenant texte, capture d’écran, logiciel

Description générée automatiquement

34 - Code permettant de mettre à jour une commande

#### Fetch de données

Une image contenant texte, capture d’écran, logiciel

Description générée automatiquementAfin de permettre la communication entre le serveur et le frontend une fonction react personnalisée de  
 « hook » nommé « useFetch » est définit afin d’y récupérer les données reçues, le statut de chargement et une potentielle erreur.

Le fait que les variables soient déclarées en tant que variables d’état permet d’utiliser un « useFetch » pour chaque appel d’API spécifique et nommer les variables reçues selon la route. Il est donc possible de les utiliser en tout temps et effectuer d’autres appels qui les mettront automatiquement à jour :

35 - Hook permettant les appels d'API



36 - Récupération de la fonction de requêtes et des données

La méthode « shopApiCall » permet l’appel API sur le serveur.

Lorsque la fonction « shopApiCall » (de son nom originel « makeApiCall ») est appelée, l’url et la méthode (GET/POST) doivent être précisés.

Une image contenant texte, capture d’écran

Description générée automatiquement

37 - Fonction d'appel d'API

Le corps n’est utilisé que lorsque la requête est exécutée en POST afin d’envoyer des informations en JSON, un fichier ou autres contenus. Le type de contenu n’est présent que pour définir le type d’élément envoyé et la variable « displayError » permet, peu importe les circonstances, de ne pas afficher d’erreur.

Les appels d’api sont exécutés à l’aide de la fonction native à typescript « fetch » et retourne toutes les informations trouvées afin de mettre à jour les variables d’état.

La réponse est récupérée, met à jour le statut de chargement et retire toute erreur potentielle.

Il vérifie ensuite le statut de la requête, dans le cas où le code ne serait pas dans les 200, l’utilisateur est redirigé sur une page d’erreur. L’erreur reçue est donc définie dans la variable d’état « error »

Par contre, si aucune erreur n’est reçue, les données sont ajoutées à la variable d’état associée et le statut de chargement est stoppé.

Une image contenant texte, capture d’écran, logiciel

Description générée automatiquement

38 - Resultats de l'appel d'API

#### Connexion et droits utilisateurs

Tous les appels d’API vu jusqu’à maintenant sont utilisables dans 100% des cas, dont la connexion et la gestion de la session de l’utilisateur. La majorité d’entre eux sont des requêtes GET, malgré cela, la route utilisée pour la connexion utilisateur est en POST afin d’y envoyer l’email et le mot de passe.

Une image contenant texte, capture d’écran

Description générée automatiquement

39 - Route de connexion au compte dans le backend

Pour se connecter au site, le minimum requis est d’entrer un email et un mot de passe. La fonction login récupère ces deux éléments à l’aide de l’objet « request » qui est une variable native du micro-framework flask. Elle permet l’accès au corps de la requête et donc à des objets convertis en JSON, etc...

Une image contenant texte, capture d’écran, Police

Description générée automatiquementEnsuite, l’utilisateur est récupéré dans le cas où il existe. Sinon, une erreur est renvoyée contenant le message « Invalid credentials » signalant à l’utilisateur que les informations sont erronées

Finalement, si les informations de connexion sont correctes, l’utilisateur est ajouté dans une session puis la route retourne du JSON contenant les informations principales de l’utilisateur à l’aide de la fonction

« to\_dict\_raw » retournant la représentation d’un utilisateur en tant que dictionnaire de données.

40 - Transformation de l'objet en dictionnaire

Désormais, pour que le backend reçoive ces informations, il faut que le frontend les envoie. C’est pourquoi « useFetch » est utilisé afin de récupérer les variables d’états récupérée.



41 - Utilisation de "useFetch" pour la connexion utilisateur

Ensuite email et mot de passe sont récupérés séparément dans des variables d’état :

Une image contenant texte, Police, capture d’écran, ligne

Description générée automatiquement

42 - Variables d'états email et mot de passe

Cependant, un élément important entre en jeu. Il s’agit d’un « contexte » permettant le passage d’une variable à travers tout le frontend à l’aide d’un seul composant.

Une image contenant texte, capture d’écran, Police, ligne

Description générée automatiquement

43 - Composant "UserContext" gérant la session

Le « UserContext » est le composant servant à gérer toute la partie utilisateur. Il renvoie en valeur un objet contenant des données d’appel d’API et des fonction propres au composant. Il permet donc au composant « Login » de récupérer les variables suivantes grâce au hook « useContext » natif a react :



44 - Récupération des variables du contexte

La fonction « login » connecte l’utilisateur en effectuant un appel d’API comme suit : 

45 - Fonction de connexion à l'utilisateur

Une image contenant texte, capture d’écran, Police

Description générée automatiquementIl est ensuite soumis à une fonction « checkUserSession » dont le but est de vérifier si l’utilisateur a été reçu et mettre à jour la variable d’état stockant ses données.

Dans le cas où l’utilisateur se rendrait sur la page de connexion en étant déjà connecté, il serait automatiquement redirigé sur la page d’accueil.

La raison à cela est qu’à chaque changement de page, le site requiert un nouvel appel d’API afin de récupérer l’utilisateur connecté.

C’est le cas de la route « /@me ».

46 - Fonction de vérification de session

Une image contenant capture d’écran, texte, Police

Description générée automatiquement

47 - Appel d'API "/@me" à chaque rechargement de page

Celle-ci permet la récupération de l’objet complet de l’utilisateur actuellement connecté sur la session. Cette fonction est énormément utilisée dans le cas d’un changement de page ou de rechargement.

Une image contenant texte, capture d’écran, Police

Description générée automatiquement

48 - Route permettant la récupération de l'utilisateur actuel

A partir de ce moment précis, l’utilisateur est soumis à une limite de droits en fonction de son rôle. Les utilisateurs managers n’ont pas accès aux boutons d’acceptation ou de refus d’une commande, à l’inverse des patrons.

Une image contenant texte, capture d’écran, Police, logiciel

Description générée automatiquement

49 - Droits utilisateur, visibilité des boutons

### Pages

Les pages sont toutes gérées de la même manière à l’aide de composants distincts. Ceux-ci se distinguent à l’aide de leur extension « .tsx » et certains d’entre eux sont réutilisés à plusieurs endroits afin de ne pas avoir de redondance dans le code.

#### Structure des pages

Les exemples les plus simples sont les composant « Main » et « Sidebar ». Ils sont réutilisés dans chacune des « Views » du projet selon leur utilité ou non.

* **Main :** Définit la couleur de fond gère le style principal similaire dans chaque page.
* **Sidebar :** Affiche la barre de navigation latérale pour la navigation entre les pages.

Une image contenant texte, Police, capture d’écran, Graphique

Description générée automatiquement

50 - Importation des composants

Une image contenant texte, capture d’écran, Police

Description générée automatiquement

51 - Tableau de bord utilisant les composants Main et Sidebar

#### Navigation

React a à sa disposition une librairie nommée « react-router-dom » permettant la navigation entre les pages du site ajoutant des routes frontend. Celles-ci sont complètement indépendantes et n’ont rien à voir avec celles du backend.

Une image contenant texte, capture d’écran, Police

Description générée automatiquement

52 - Router du frontend (Contenu du fichier « Router.tsx »)

Le composant « Router » est le point central de l’application et est accompagné de balises « Navlink » pré-faites aussi. Elles permettent la redirection d’url à travers le site.

Ces balises sont contenues dans le composant « Sidear » introduit précédemment.

Une image contenant texte, capture d’écran

Description générée automatiquement

53 - Contenu du composant Sidebar pour la navigation

## Description des tests effectués

L’entièreté des tests effectués tiennent sur 2 scripts Cypress. Ensemble, ils couvrent l’intégralité des fonctionnalités du site qui sont : la gestion utilisateur et l’importation de stock et commandes.

Ces tests ont été réalisés manuellement dans un premier temps avant d’avoir été lancés avec Cypress.

Les résultats attendus et les conditions de lancement des tests étaient définis comme suit.

Une image contenant texte, capture d’écran, nombre, Police

Description générée automatiquement

54 - Première partie des tests à effectuer (importation fichier excel)

Une image contenant texte, ligne, nombre, Police

Description générée automatiquement

55 - Seconde partie des tests effectués (Connexion utilisateur)

Les résultats officiels des tests Cypress proviennent directement de la console du container docker utilisé pour les lancer.

Une image contenant texte, capture d’écran, Police

Description générée automatiquement

56 - Resultats du test automatique cypress d'importation de fichier excel

Une image contenant texte, capture d’écran, Police

Description générée automatiquement

57 - Resultat de tests automatique cypress de connexion utilisateur

## Erreurs restantes

Aucune erreur n’est restée. Les bugs et problèmes trouvés ont été résolus aussi vite que possible et correctement. Rien n’exclut la possibilité que certaines erreurs apparaissent, mais aucune n’a été trouvée à ce jour en testant en boucle les fonctionnalités du site.

## Liste des documents fournis

* Planification initiale
* Rapport de projet
* Journal de travail
* Fichier ZIP contenant le code source et autres contenus du projet

# Conclusions

En conclusion, tous les objectifs demandés du projet ont été accomplis. Malgré quelques problèmes avec docker qui ont créé un léger retard pour l’implémentation de l’application, tout fonctionne correctement et aucun problème visible n’est apparu.

Le rapport a été quelque peu négligé en début de TPI mais a été finit sur le fil malgré tout. La difficulté du projet était raisonnable et a abordé plusieurs sujets dont le sujet principal utilisé durant tout le stage qui est Cypress.

## Atteinte des objectifs

Tous les objectifs cités dans le point 1.2 ont été entièrement réalisés.

* La connexion utilisateur fonctionne et les informations affichées sont correctes
* Un utilisateur peut ajouter du stock ou des commandes dans le magasin via des fichiers excel. Si des articles ou commandes n’existent pas, le backend les ajoutera automatiquement en tant que nouvelles entrées dans la base de données
* Les droits utilisateurs sont gérés correctement.
* Les tests de bout en bout couvrent toutes les fonctionnalités demandées par le cahier des charges.
* Un patron peu accepter ou refuser une commande et le stock sera mis à jour selon le bouton cliqué.
* Le système contient une arborescence réutilisable et améliorable à souhait.

## Points positifs

* Ce projet m’a permis d’apprendre docker et de comprendre les fondamentaux de la création d’environnement avec des containers.
* J’ai mis à profit mes connaissances découvertes durant le stage, étant Cypress, Jira, etc…
* Le projet offrait une liberté des choix de maquettes, de l’approfondissement des fonctionnalités et de la méthodologie de travail.

## Point négatif

* Le projet avait besoin de tourner sur docker, un domaine encore inconnu pour moi avant le projet ce qui m’a fait perdre quelque peu de temps afin de régler les problèmes dont je ne comprenais pas tout.

## Difficultés

### Mise en place de docker

Le projet entier devait tourner sur docker. Lors des premiers jours, la mise en place de l’environnement avait été planifiée. Tout d’abord, le projet comptabilisait 3 containers pour chacun, le frontend, le backend et Cypress.

La première difficulté apparue était en rapport avec le container Cypress. Lors du build des containers, celui de Cypress ne le faisait pas correctement et ne trouvait pas les fichiers de configuration Cypress, alors qu’ils y étaient. La solution trouvée et utilisée était de supprimer l’arborescence de base et en refaire une à neuf.

La plus grande difficulté avec docker a finalement été avec l’application elle-même. Au départ, le frontend et le backend tournaient sur deux serveurs distincts (port 3000 et 5000). La solution pour faire communiquer les deux étaient d’utiliser des appels d’API classiques entre les deux serveurs. Cependant, le frontend était bloqué à cause des CORS (Cross-origin resource sharing) permettant de partager des données qui peuvent être récupérée par un autre domaine (partager des infos entre domaines/serveurs). Malgré les tentatives d’autorisation, certaines requêtes sur l’API étaient tout de même restreintes.



La solution finale qui fut concluante a été de supprimer le container frontend afin de réunir l’application entière en un seul nommé « app ». Lors du build de celui-ci, en premier lieu, il utilise une image « node.js », installe les librairies javascript, compile le frontend et ajoute le css grâce à tailwind.

Ensuite il change l’image pour python3.12.3, installe les librairies et démarre le serveur dont lui-même retournera directement l’html compilé précédemment.

## Evolutions et améliorations possibles du projet

Le projet a été réalisé de façon qu’il puisse évoluer.

*Développez en tous cas les points suivants:*

* *Objectifs atteints / non-atteints*
* *Points positifs / négatifs*
* *Difficultés particulières*
* *Suites possibles pour le projet (évolutions & améliorations)*

# Glossaire

**API :** “Application programming interface” en anglais, est une interface avec des interactions délimitées par ses développeurs offrant des services à d’autres logiciels.

**Hash :** Résultat d’une fonction mathématique permettant de produire un ensemble de charactères avec une longueur fixe à partir d’un autre ensemble de charactères de longueur variable. Par exemple, le hachage pour la phrase ‘Mot de passe’ : *f0401823bc8e44fcfd99399be0c763ff6f1d195031382272b2808ce6addf56eb*

**Model / Modèle :** Fichier contenant des informations permettant la communication avec la base de données.

**View :** Fichier contenant l’interface de l’application. Il contient des composants permettant la création du design du site.

**Route :** Fonction associée à un url pouvant recevoir des informations dû à un appel d’api.

**Framework :** Modèle développé par un développeur ou une entreprise permettant d’avoir une structure pour organiser le code et les fichiers de l'application, ainsi que des fonctionnalités courantes telles que la gestion des bases de données, la gestion des sessions utilisateur et la sécurité.

**Frontend :** Partie d'une application ou d'un site web avec laquelle les utilisateurs interagissent directement. (Design / Visuel du site web).

**Backend :** Partie d’une application ou d’un site web qui fonctionne en arrière-plan sur un serveur. Il permet l’accès à la base de données ou encore la gestion des requêtes HTTP. Il permet entre autres l’affichage du frontend.

# Annexes

Ce chapitre contient tous les autres fichiers et bibliographie utiles du projet.

## Résumé du rapport du TPI

### Situation de départ

### Mise en œuvre

### Résultats

## Sources – Bibliographie

### Documentations officielles des outils utilisés

<https://www.docker.com>

<https://flask.palletsprojects.com/en/3.0.x/>

<https://tailwindcss.com>

<https://fr.react.dev>

### Stack overflow pour les compléments

<https://stackoverflow.com/questions/28089344/docker-what-is-it-and-what-is-the-purpose>

<https://stackoverflow.blog/2021/10/20/why-hooks-are-the-best-thing-to-happen-to-react/>

<https://stackoverflow.com/questions/39565706/post-request-with-fetch-api>

## Journal de travail

## Manuel d'Installation

Afin d’installer l’environnement, le prérequis est d’avoir l’application « docker ».

Une fois l’application démarrée, il suffit d’aller dans le répertoire de projet où se trouve le fichier « compose.yaml » et d’exécuter la commande « docker compose up » ou « docker compose watch ».

Une fois le tout compilé, les deux containers devraient être en action. Il est donc possible ensuite de voir les logs cypress en allant dans son container et d’accéder au site grâce au port ouvert (5000)

Une image contenant texte, capture d’écran, Police

Description générée automatiquement

## Manuel d'Utilisation

Une image contenant texte, capture d’écran, Police, conception

Description générée automatiquement

La première étape en se rendant sur le site est de se connecter en remplissant les champs avec les bonnes informations.

Une image contenant texte, capture d’écran, Police, Téléphone mobile

Description générée automatiquement Une image contenant texte, capture d’écran, Police

Description générée automatiquement

Il est ensuite possible selon le rôle de l’utilisateur de se rendre sur le tableau de bord et les commandes.

Une image contenant texte, capture d’écran, Police, Graphique

Description générée automatiquement

Sur le tableau de bord, il est possible d’importer du stock si l’utilisateur n’est pas patron ou des commandes si l’utilisateur est uniquement un Manager.

Une image contenant capture d’écran, texte, Police

Description générée automatiquement

Il est possible de voir le total des unités de chaque tableau, en haut de la page.

Une image contenant texte, capture d’écran, Police, nombre

Description générée automatiquementUne image contenant texte, capture d’écran, nombre, Police

Description générée automatiquement

Une fois sur le tableau de bord, il est possible de trier par colonne les tableaux d’articles et de commandes

Une image contenant texte, capture d’écran, Police, Graphique

Description générée automatiquement

Si l’utilisateur est patron, il est alors possible d’approuver ou de refuser des commandes avec les boutons ci-dessus.

Une image contenant capture d’écran, texte, Police, Graphique

Description générée automatiquement

Pour finir il est aussi possible de se déconnecter de son compte.