

Gestion de stock de montres

Dossier de projet de TPI

Damien Loup

CID4B

Table des matières

1	Analyse préliminaire	4
1.1	Introduction	4
1.2	Objectifs.....	4
1.3	Planification initiale	5
1.3.1	Répartition du temps prévu en %.....	7
2	Analyse / Conception.....	8
2.1	Concept	8
2.1.1	Conception de la base de données	8
2.1.2	Api backend	11
2.1.3	Diagramme de flux utilisateur	12
2.1.4	Maquettes.....	13
2.2	Stratégie de test.....	16
2.2.1	Tests de bout en bout	16
2.2.2	Tests manuels	16
2.2.3	Données de test.....	16
2.3	Risques techniques	16
2.4	Planification	17
2.5	Dossier de conception	19
2.5.1	Technologies	19
3	Réalisation.....	19
3.1	Dossier de réalisation	19
3.1.1	Langages utilisés	19
3.1.2	Arborescence.....	20
3.1.3	Base de données.....	24
3.1.4	Communication frontend et backend	25
3.1.5	Pages	34
3.2	Description des tests effectués.....	36
3.3	Erreurs restantes	39
3.4	Liste des documents fournis	39
4	Conclusions	39
4.1	Atteinte des objectifs.....	39
4.2	Points positifs	39
4.3	Point négatif.....	40
4.4	Difficultés	40
4.4.1	Mise en place de docker	40
4.5	Evolutions et améliorations possibles du projet	40
5	Glossaire	40
6	Annexes.....	41
6.1	Sources – Bibliographie.....	41
6.1.1	Documentations officielles des outils utilisés	41
6.1.2	Stack overflow pour les compléments	41
6.2	Résumé du rapport du TPI	42
6.2.1	Situation de départ.....	42
6.2.2	Mise en œuvre.....	42

6.2.3	Résultats.....	42
6.3	Journal de travail	43
6.4	Manuel d'Installation	53
6.5	Manuel d'Utilisation.....	53

1 Analyse préliminaire

1.1 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.

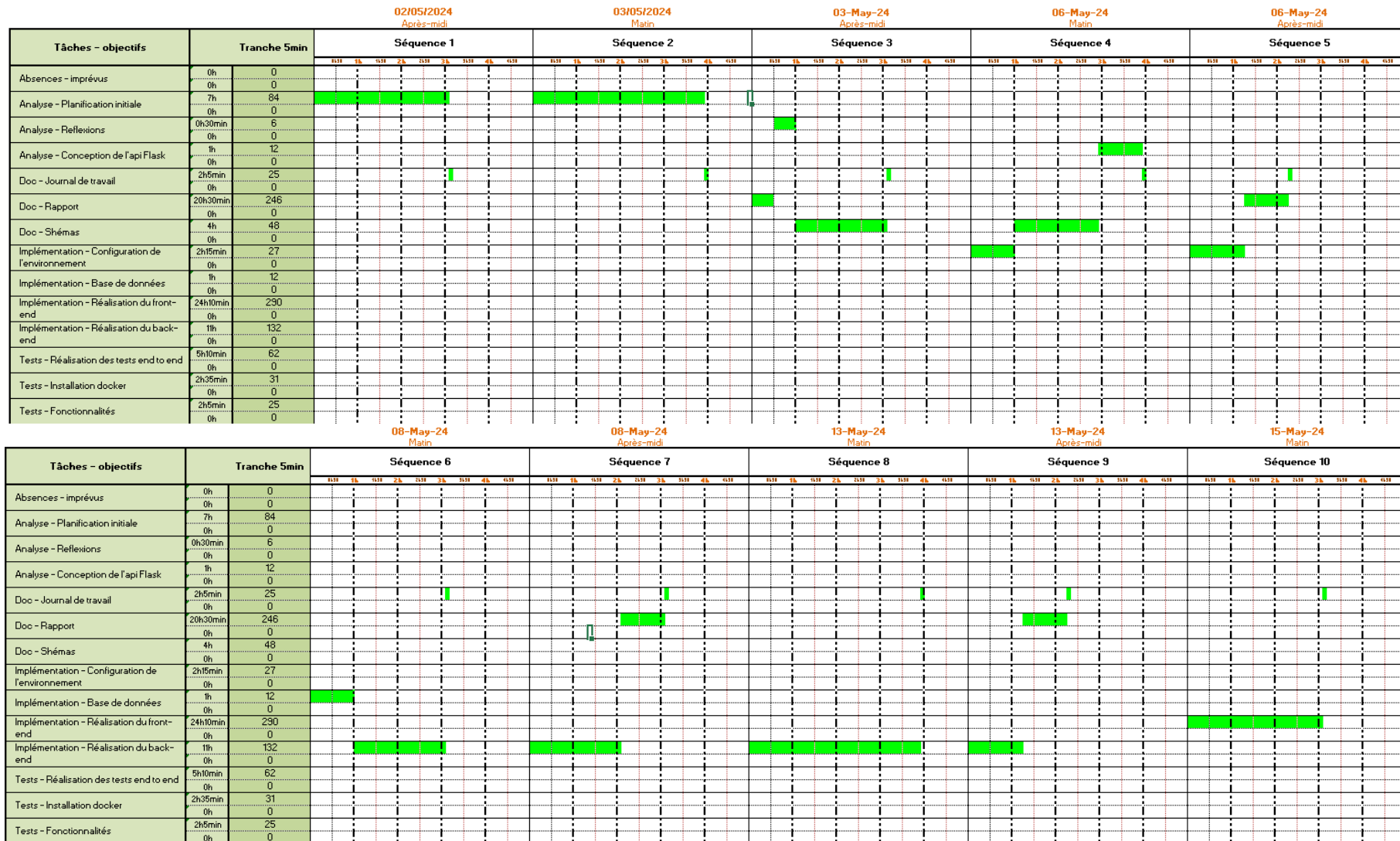
Ce projet a pour but de consolider certaines connaissances, tel que le développement web, soit les langages qui sont distinctement le typescript pour REACT et le python pour FLASK. Il en va de même pour la réalisation des tests qui utilisent la technologie CYPRESS qui permet de simuler un utilisateur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

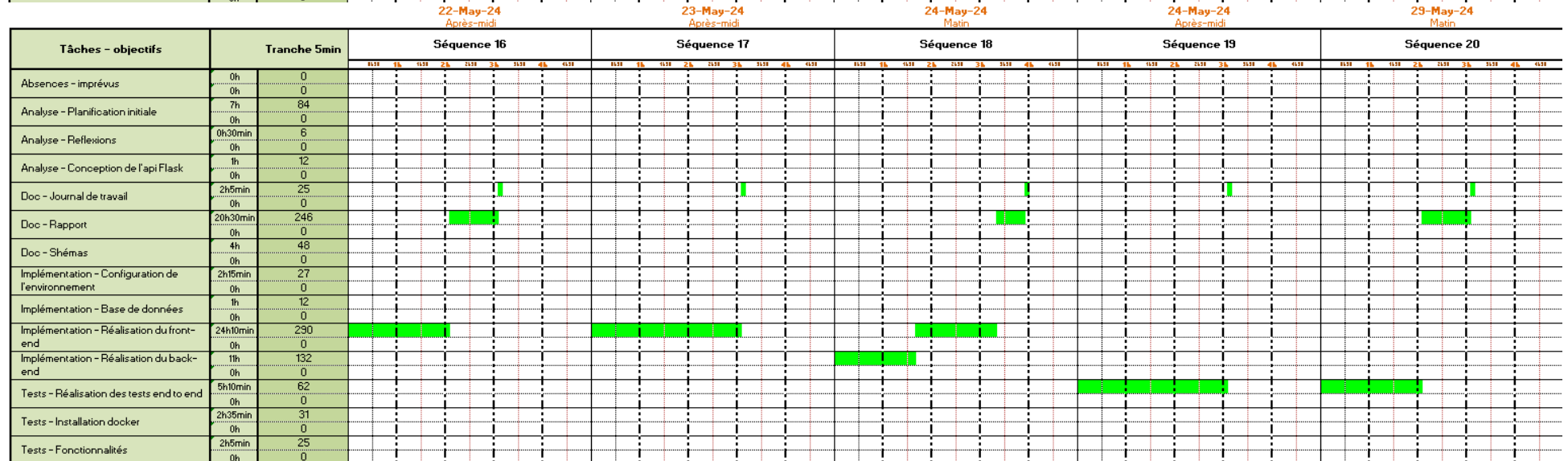
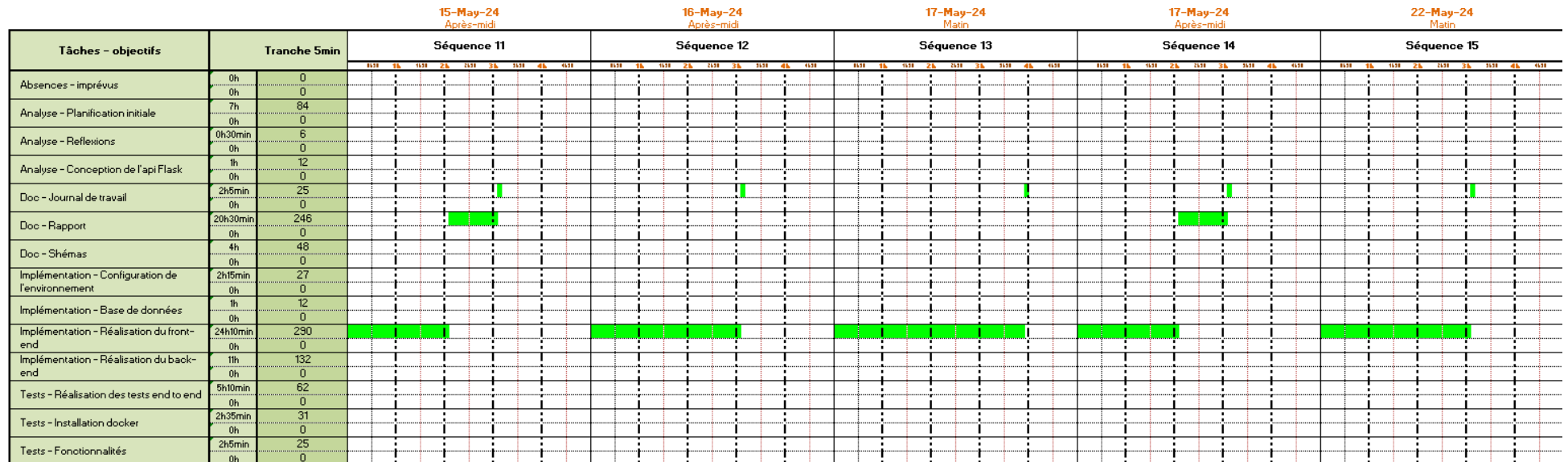
1.2 Objectifs

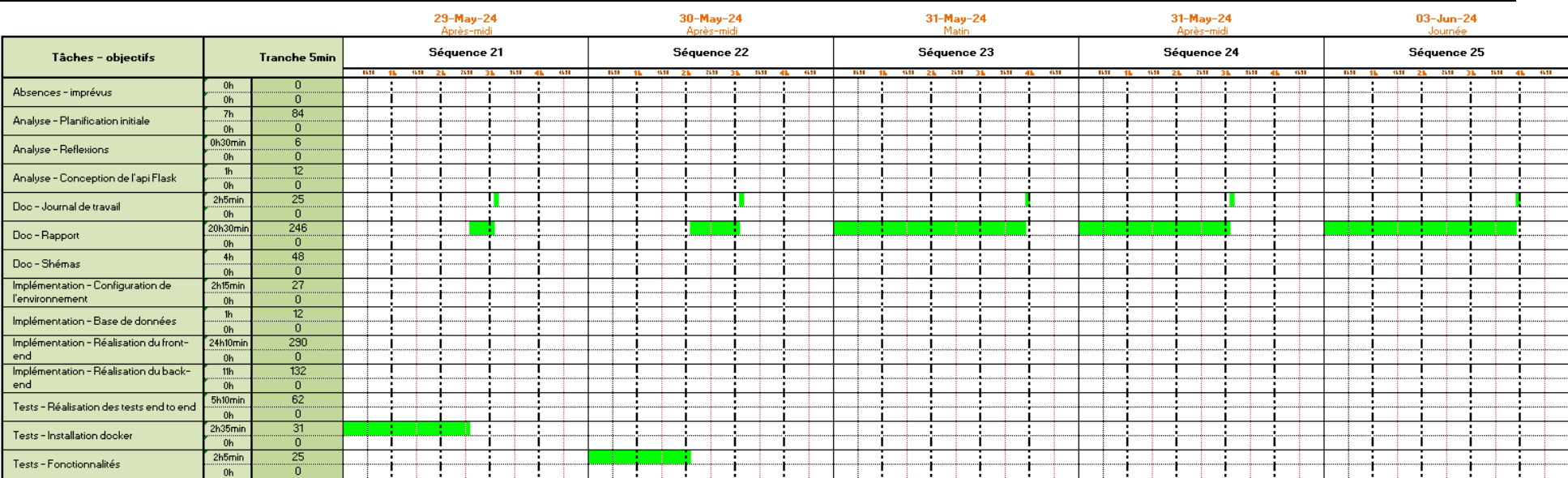
L'objectif du projet est d'avoir un site web utilisable tournant sur docker avec des tests de bout en bout contenant les fonctionnalités suivantes :

- 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.

1.3 Planification initiale







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

1.3.1 Répartition du temps prévu en %

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

2 Analyse / Conception

2.1 Concept

2.1.1 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 chaîne. 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 stockées 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.

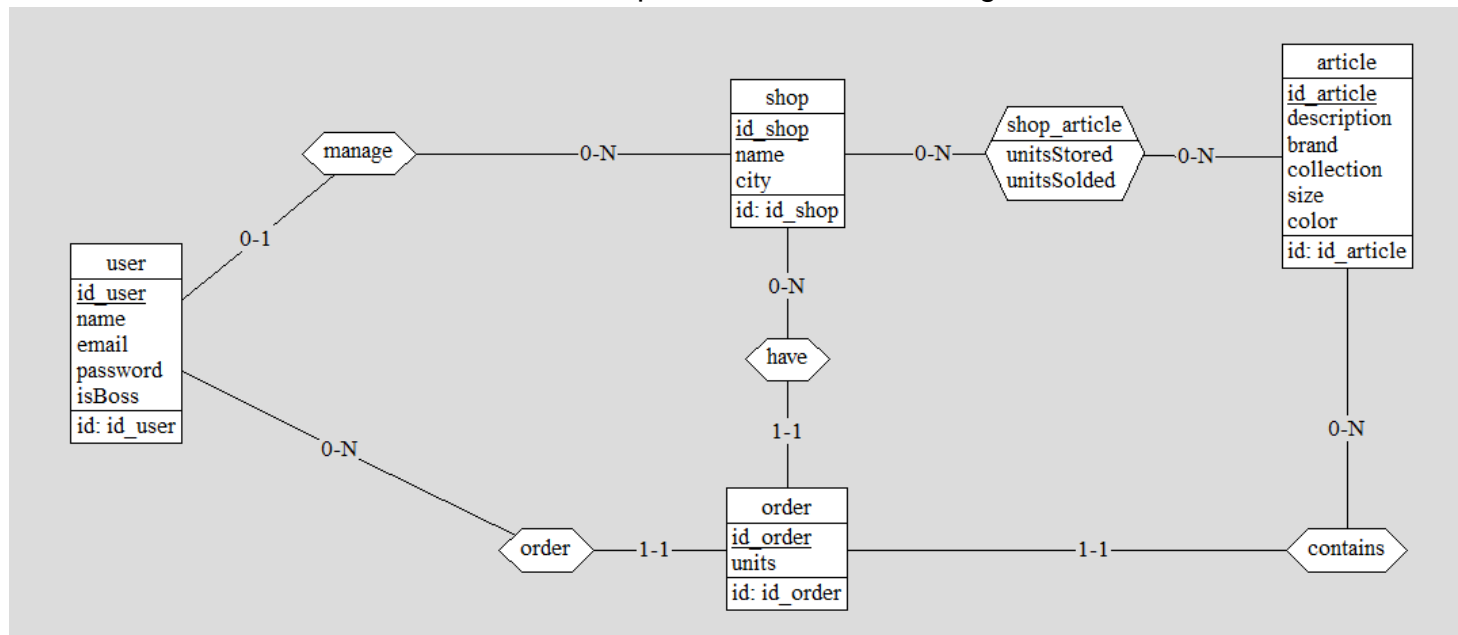
2.1.1.1 MCD

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



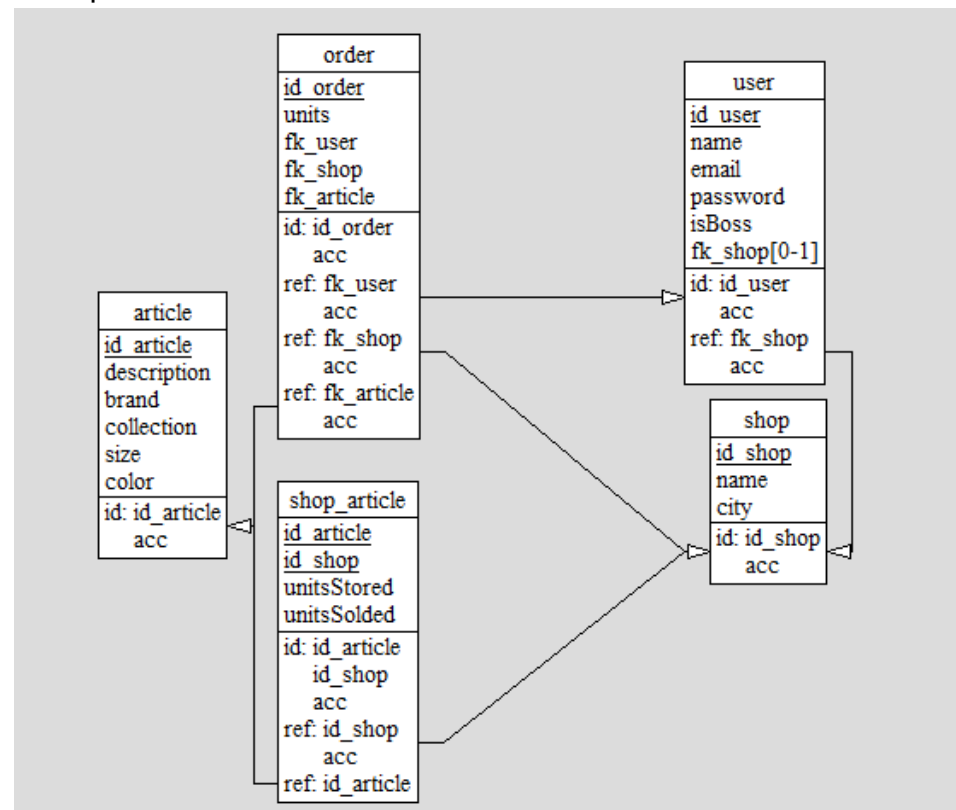
1 - Schéma mcd de la base de données

2.1.1.2 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 \Leftrightarrow 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.



2 - Schémas MLD de la base de données














2.1.2 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.

2.1.2.1 Routes

Description	Main	Datas	Utility/Return
Main route. Display the frontend	GET  /		Return frontend (HTML) from react build
	Authentication		
Check for the credentials sent and log the user in if valid	POST  /login	{ email, password }	- Log the user in a backend session - Return the user object OR - Return an error message
Log the user out	GET  /logout		- Log the user out from the session - Return a success message
Get the user from the actual session	GET  /@me		- Get the user from the session - Return the user object
	Shops		
Get a list of all shops	GET  /get_shops		- Get all the shops - Return the list of shops OR - Display an error 401
Get a shop	GET  /shop/<int:shop_id>	ID of the shop in URL	- Get a specific shop with ID - Return the shop object OR - Return an error message
Get a list of article from a shop	GET  /shop/<int:shop_id>/articles	ID of the shop in URL	- Get the shop articles from his ID - Return a list of articles OR - Return an error message
Get a list of orders from a shop	GET  /shop/<int:shop_id>/orders	ID of the shop in URL	- Get the shop orders from his ID - Return a list of orders OR - Return an error message
	Users		
Get a specific user	GET  /user/<int:user_id>	ID of the user in URL	- Get a user from his ID - Return the user OR - Return an error message
Get the shop of a specific user	GET  /user/<int:user_id>/shop	ID of the user in URL	- Get the shop of a user - Return the shop OR - Return an error message
	Articles		
Get all the articles	GET  /get_articles		- Get all the articles - Return an array of articles OR - Return an error 401
	Orders		
Get all the orders	GET  /get_orders		- Get all the orders - Return an array of orders OR - Return an error 401
Update a specific order	POST  /update_order/<int:order_id>	ID of the order in URL { status }	- Get an order by ID - Return an order OR - Return an error 401

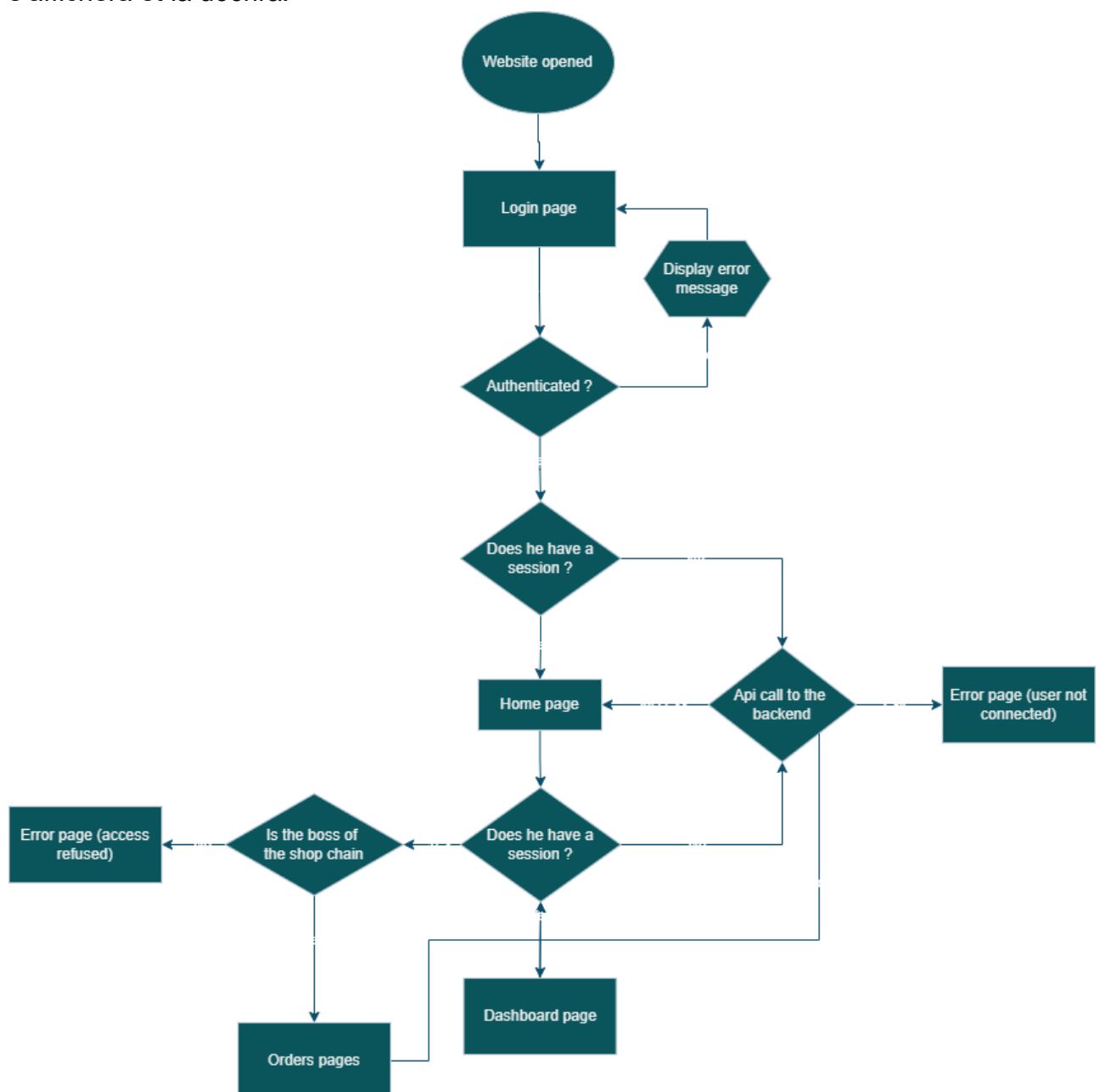
3 - Routes backend et leur fonctionnement

2.1.3 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 a 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.



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

2.1.4 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.

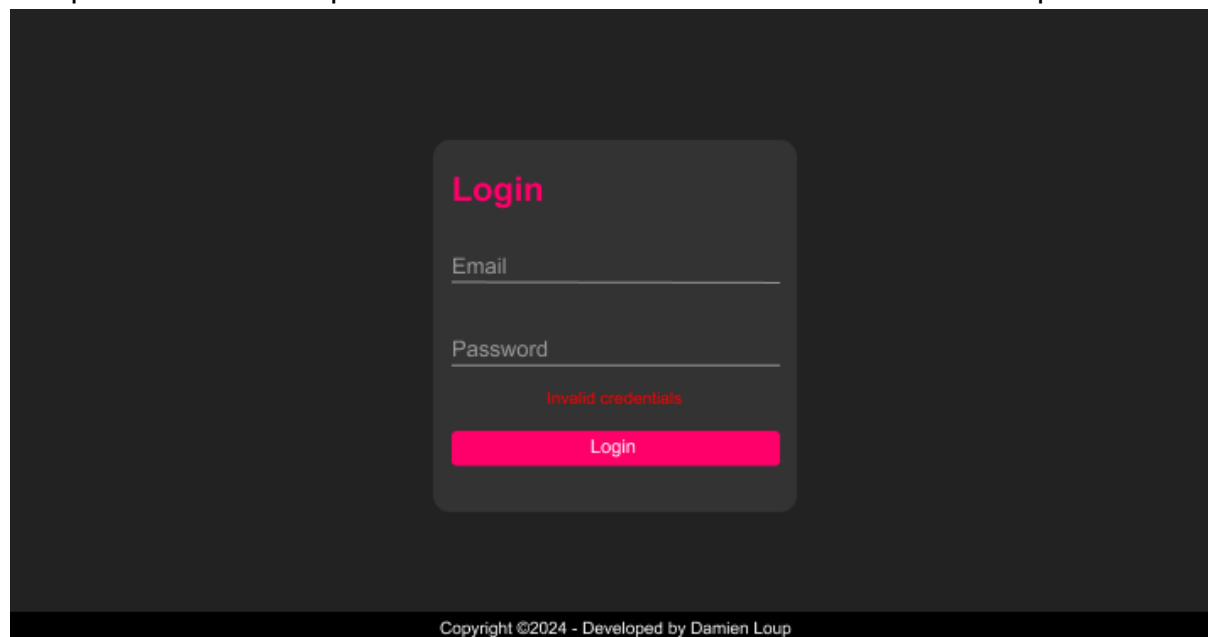
Elles ont été créées à l'aide de l'application *Figma* qui est un logiciel de design permettant de créer des maquettes de toute sorte.

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

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

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

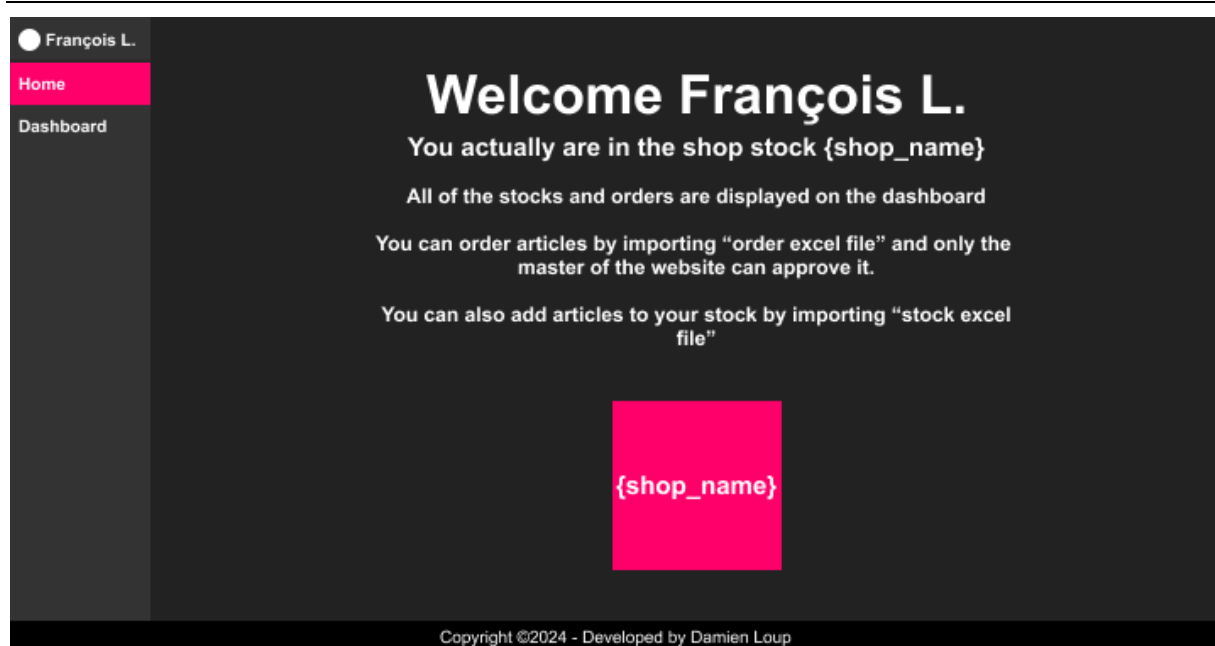
En premier lieu, il y a la page de connexion. Elle s'affiche lorsque l'utilisateur se rend sur le site. L'utilisateur peut donc se connecter à l'aide de son email et d'un mot de passe. Lorsqu'il y a une erreur, un texte rouge apparaît en dessous des champs pour indiquer à l'utilisateur qu'il a commis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



7 - Maquette du dashboard d'un utilisateur manager

La principale différence entre les utilisateurs est que les patrons 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 chaîne, 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.

Maquette du dashboard d'un utilisateur patron de la chaîne. Le menu de gauche contient 'Home', 'Dashboard' (surligné) et 'Orders'. Le titre principal est 'All shops'. En haut à droite, trois cartes indiquent : 'Items sold 280', 'Items in stock 313' et 'Items in order 489'. Deux tableaux de stock sont affichés :

Art. No.	Description	Brand	Collection	Size	Color	Units in stock	Units sold	Status
D-0110001	Watch 1	Swatch	Summer	44mm	Red	47 -	6	In Stock
D-0110002	Watch 2	Rolex	Sport	40mm	Blue	42 -	0	In Stock
D-0110003	Watch 3	Swatch	Sport	45mm	Green	0 -	168	Out of Stock

Art. No.	Description	Brand	Collection	Size	Color	Units in stock	Units sold	Status
D-0110024	Watch 1	Rolex	Winter	30mm	Orange	200	0	In Stock
D-0110025	Watch 2	Rolex	Sport	40mm	Blue	3	73	In Stock
D-0110026	Watch 3	Swatch	Sport	45mm	Green	21	43	In Stock

Copyright ©2024 - Developed by Damien Loup

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

Maquette de la page des commandes d'un utilisateur patron de la chaîne. Le menu de gauche contient 'Home', 'Dashboard' et 'Orders' (surligné). Le titre principal est 'All shops'. En haut à droite, trois cartes indiquent : 'Items sold 280', 'Items in stock 313' et 'Items in order 489'. Deux tableaux de commandes sont affichés :

Order No.	Art. No.	Units ordered	Ordered by	Approve/Refuse	Status
O-0000007	D-0110001	53	● François L.		Approved
O-0000006	D-0110001	12	● Steve D.		Not approved
O-0000008	D-0110003	62	● Elene O.	✓ ✗	Pending approval

Order No.	Art. No.	Units ordered	Ordered by	Approve/Refuse	Status
O-0000003	D-0110025	100	● Jacques U.		Not approved
O-0000002	D-0110028	200	● Steve D.	✓ ✗	Pending approval
O-0000001	D-0110003	62	● Sabrina K.		Approved

Copyright ©2024 - Developed by Damien Loup

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

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 contenant 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.

2.2 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.

2.2.1 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.

2.2.2 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.

2.2.3 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	Louis1234	Patron de la chaine
alice.s@gmail.com	Alice1234	Patronne de la chaine
françois.l@gmail.com	F-L1234	Manager d'un magasin
steve.d@gmail.com	S-D1234	Manager d'un magasin
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.

2.3 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.

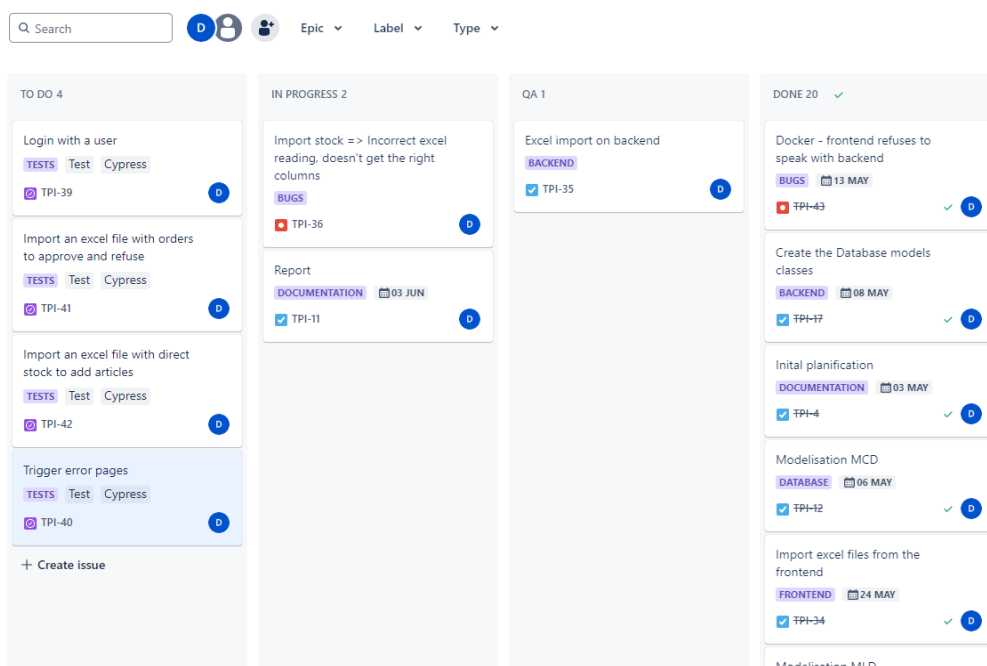
Le plus grand risque étant le peu de connaissance sur docker lors du lancement du TPI qui pourrait amener à un léger retard sur la planification initiale, ce qui demande d'apprendre d'autant plus comment correctement l'utiliser avec des tutoriels, des vidéos et de la documentation en ligne afin que l'ensemble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

2.4 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.

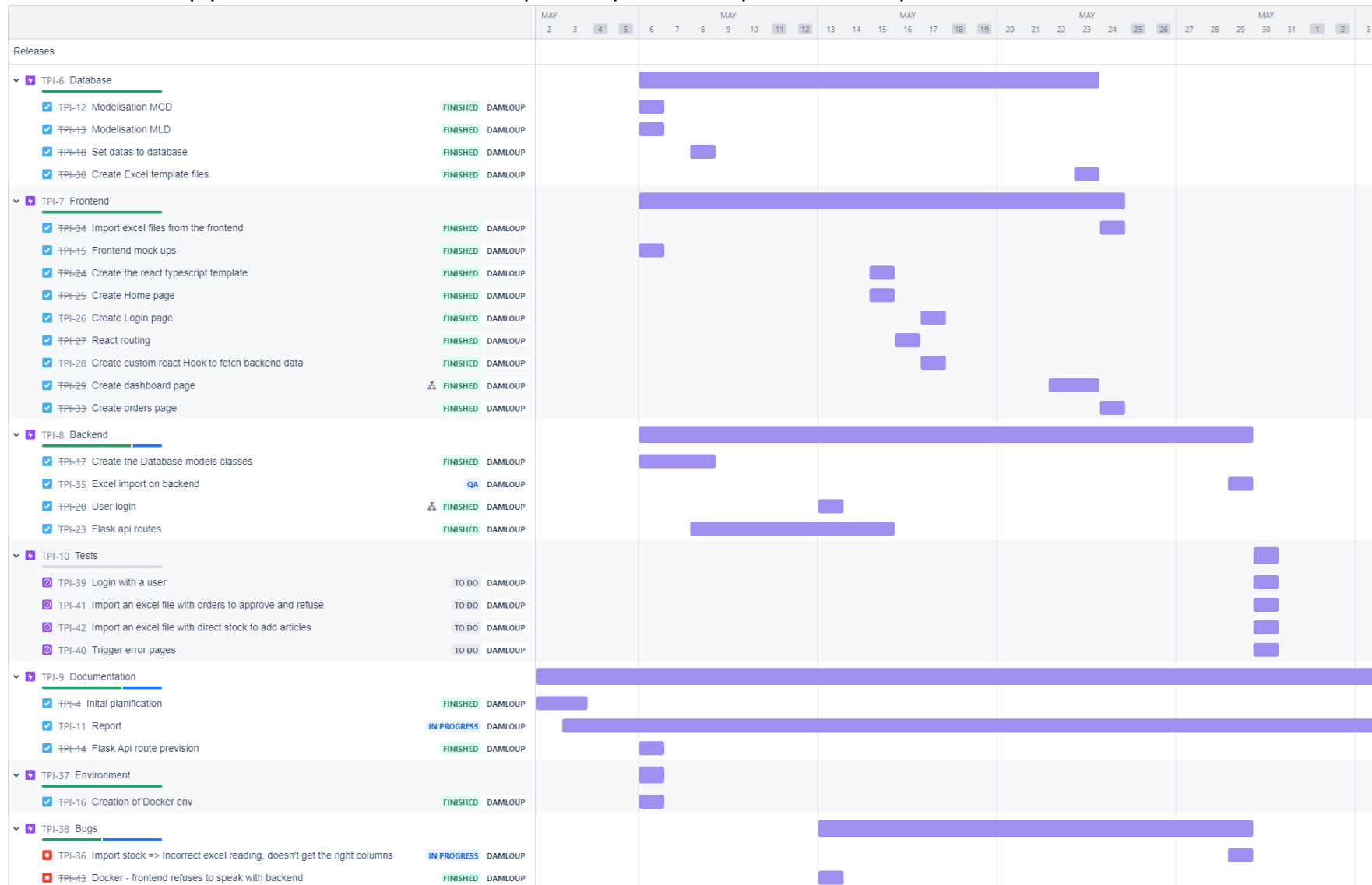
La méthode utilisée est KANBAN. Cela permet de visualiser chaque tâche en fonction de son statut d'évolution (To do, In progress, QA, Done). La colonne QA permet de réaliser un test manuel sur le ticket réalisé afin de le vérifier. Si le resultat du test est OK, le ticket finit « Done », sinon, le ticket 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.

Kanban



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

2.5 Dossier de conception

2.5.1 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

3 Réalisation

3.1 Dossier de réalisation

3.1.1 Langages utilisés

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

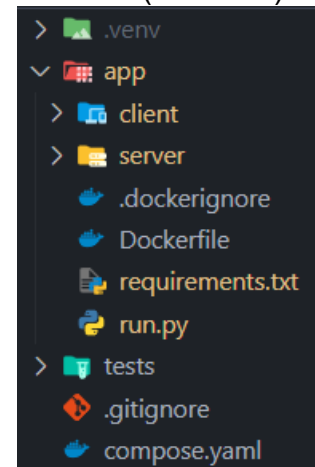
- **Python** : Utilisé pour le backend, le python permet de faire office de serveur. Il utilise néanmoins le micro-Framework nommé FLASK permettant de gérer les routes qui servent de pont entre le frontend et le backend. Ce micro-framework est très utilisé pour créer des petits projets et est donc parfait 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 typescript, des appels d'api sont mis en place afin de les faire s'envoyer des données entre eux à travers le réseau.

3.1.2 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) :

- 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.



12 - Arborescence du projet

- Ensuite le dossier « tests » permet d'y mettre tout le code servant à lancer les tests automatique cypress.
- 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 contenant 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.

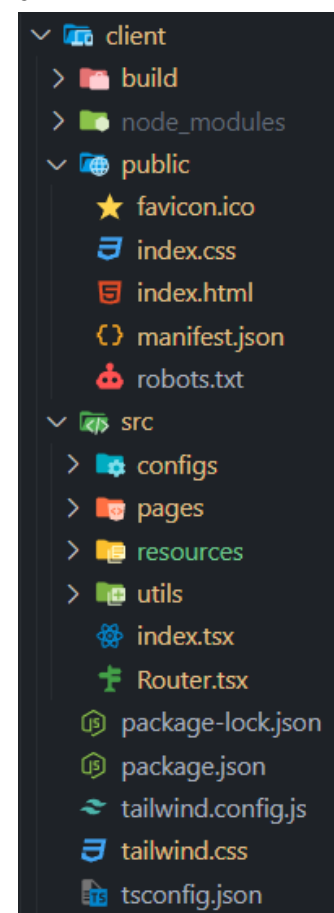
3.1.2.1 Client

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 affichés 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.

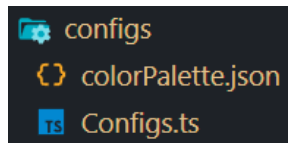


13 - Arborescence frontend

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.

- **Configurations**

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.



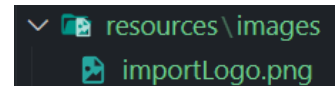
14 - Dossier de configurations frontend

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

```
<main className={`bg-colorpalette-backgrounds-primary text-colorpalette-texts-default min-h-screen ${location.pathname !== "/" && "flex"}}>
  {children}
</main>
```

- **Ressources**

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

- **Outils**

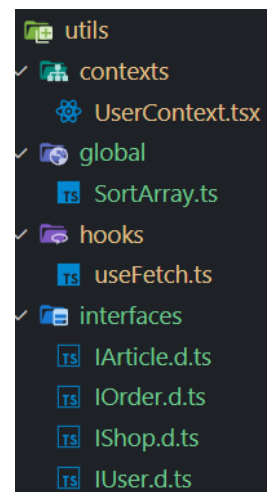
Les outils contiennent quelque peu plus de fichiers.

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

• Pages

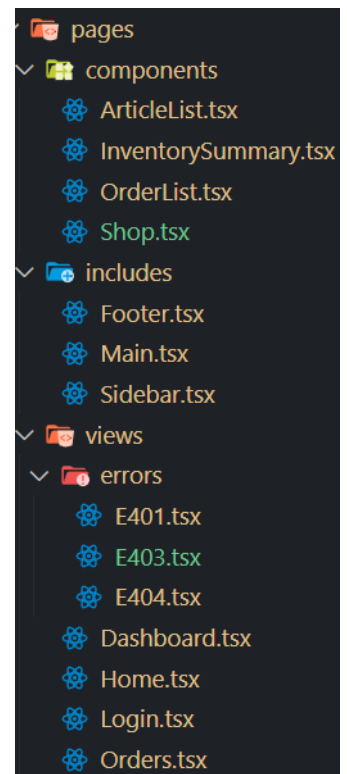
Ce dossier contient tous les composants react qui ont un rapport avec les pages du site web.

Concernant ces répertoires, le dossier « views » possède la base des pages. C'est-à-dire les composants affichés lorsque le « router » les appelle. 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 utilisés 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.

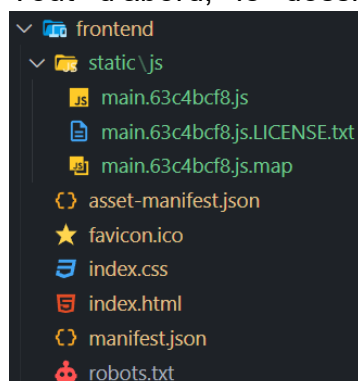


17 - Pages du frontend

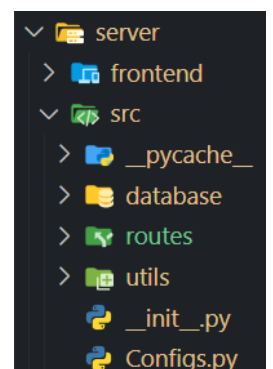
3.1.2.2 Serveur

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.

Tout d'abord, le dossier « frontend » contient le client react vu précédemment à la seule différence qu'ici, il est compilé.



19 - Frontend compilé

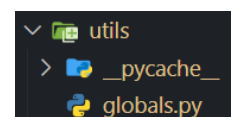


18 - Arborescence serveur

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.

• Outils

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.

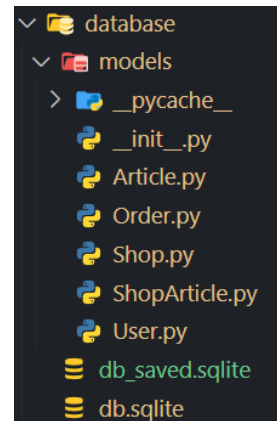


20 - Outils backend

- **Base 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 vraies bases de données 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.

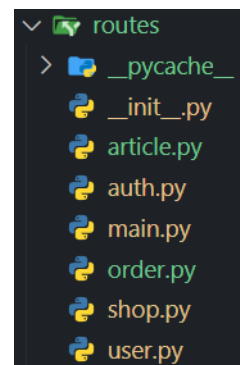


21 - Base de données du backend

- **Routes**

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.

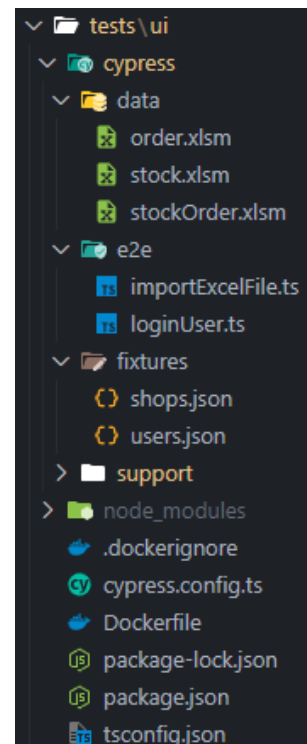


22 - Routes de l'api backend

3.1.2.3 Tests

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.



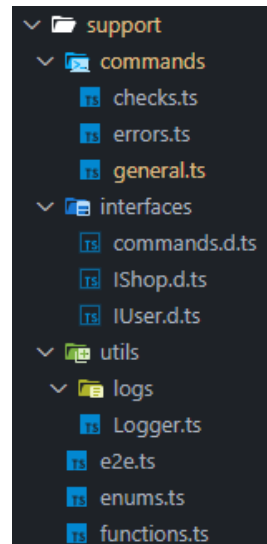
23 - Arborescence tests cypress

Afin 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

3.1.3 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.

```
# Init sqlalchemy
db = SQLAlchemy()
```

25 - Déclaration de la base de données

Voici un exemple de table :

```
class User(UserMixin, db.Model):
    """ ...

    # Create a table in the db
    __tablename__ = 'user'

    # Columns
    id = Column(Integer, primary_key=True)
    name = Column(String(255), index=True)
    email = Column(String(255), unique=True, index=True)
    password = Column(String(255))
    isBoss = Column(Integer)

    # Foreign keys
    fk_shop = Column(Integer, ForeignKey('shop.id'))

    # Relationships
    shop = db.relationship("Shop", back_populates="users")
    orders = db.relationship("Order", back_populates="user")
```

user
id user
name
email
password
isBoss
fk_shop[0-1]
id: id_user
acc
ref: fk_shop
acc

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.

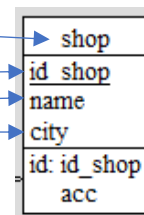
Comme toute base de données relationnelle, SQLite n'échappe pas à la règle. Il est donc possible d'avoir accès à des données liées à un utilisateur. Par exemple le magasin auquel il est affecté ou alors les commandes qu'il a passé à l'aide de l'importation de fichiers.

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).

```
class Shop(db.Model):
    """ ... """
    # Create a table in the db
    __tablename__ = 'shop'

    # Columns
    id = Column(Integer, primary_key=True)
    name = Column(String(255), unique=True, index=True)
    city = Column(String(255))

    # Relationships
    shop_articles = db.relationship("ShopArticle", back_populates="shop")
    orders = relationship("Order", back_populates="shop")
    users = relationship("User", back_populates="shop")
```



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 :

```
user: User = User.query.filter_by(name="Louis").first()
```

3.1.4 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.

3.1.4.1 Routes

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

```
# Save the blueprints
from .routes.main import main as main_blueprint
from .routes.auth import auth as auth_blueprint
from .routes.shop import shop as shops_blueprint
from .routes.user import user as users_blueprint
from .routes.article import article as articles_blueprint
from .routes.order import order as orders_blueprint

# Register the blueprints
app.register_blueprint(main_blueprint)
app.register_blueprint(auth_blueprint)
app.register_blueprint(shops_blueprint)
app.register_blueprint(users_blueprint)
app.register_blueprint(articles_blueprint)
app.register_blueprint(orders_blueprint)
```

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.

```
# Create the main blueprint
main = Blueprint("main", __name__, static_folder=STATIC_FOLDER, static_url_path='/')

@main.app_errorhandler(404)
def not_found(e):
    """
    ...

    return get_frontend()

@main.route("/", methods=["GET"])
def index():
    """
    ...

    return get_frontend()

def get_frontend():
    """
    ...

    # Check if the frontend url is set otherwise display the static html
    if FRONTEND_URL:
        return redirect(FRONTEND_URL)

    # Display the static html
    return main.send_static_file("index.html")
```

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.

```
@shop.route("/shop/<int:shop_id>", methods=["GET"])
@login_required
def get_shop_by_id(shop_id: int):
    """
    ...

    # Get the shop
    shop: Shop = Shop.query.get(shop_id)

    # Check if shop exists
    if not shop:
        return jsonify({"message": "Shop not found"})

    # Return the shop
    return jsonify({"shop" : shop.to_dict_raw()}), 200
```

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

La 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.

```
@login_manager.unauthorized_handler
def unauthorized():
    """
    ...

    Block the user when not logged in
    """

    # Return unauthorized message
    return jsonify({'message': 'Unauthorized'}), 403
```

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

33 - Erreur lorsque l'utilisateur est censé être connecté

La route permettant de mettre à jour une commande n'est accessible que lorsque l'utilisateur a 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.

```
@order.route("/update_order/<int:order_id>", methods=["POST"])
@login_required
def update_order(order_id: int):
    """...
    # Check if the user is an admin
    user: User = current_user
    if not user.isBoss:
        return jsonify({"error": "You are not an admin"}), 401

    # Get the data from the request
    status: bool = request.json.get("status")

    # Get the order
    order: Order = Order.query.filter_by(id=order_id).first()
    if order is None:
        return jsonify({"error": "Order not found"})

    # Modify the order
    order.status = "Approved" if status else "Rejected"

    # If the order is approved, add the units to the shop article
    if status:
        shop_article: ShopArticle = ShopArticle.query.filter_by(fk_article=order.fk_article, fk_shop=order.fk_shop).first()
        shop_article.unitsStored += order.units

    # Commit the changes
    db.session.commit()

    # Return the success message
    return jsonify({"message": "order successfully updated"}), 200
```

34 - Code permettant de mettre à jour une commande

3.1.4.2 Importation de fichier excel

L'importation de fichier excel est la partie la plus importante de ce site de gestion de stock. Il permet d'importer très rapidement un grand nombre d'articles ou de commandes.

Voici un exemple d'à quoi il peut ressembler :

	A	B	C	D	E	F	G	H
	ID	Description	Brand	Collection	Size	Color	Stock	Order
1	1	Rose Gold Watch	Daniel Wellington	Eco-Drive	35mm	Silver	649	0
2	2	Smartwatch with GPS	Rolex	Premier	45mm	Gold	0	0
3	3	Digital Sports Watch	Fossil	Carrera	44mm	White	0	322
4	4	Elegant Quartz Watch	Tag Heuer	Premier	38mm	Black	0	0
5	5	Diver's Watch	Apple	Carrera	43mm	Rose Gold	0	0
6	6	Fashion Watch	Tag Heuer	Parker	43mm	Silver	0	0
7	7	Rose Gold Watch	Citizen	Carrera	36mm	White	0	0
8	8	Smartwatch with GPS	Omega	Seamaster	38mm	Rose Gold	0	0
9	9	Diver's Watch	Michael Kors	Oyster Perpetual	42mm	Space Gray	0	0
10	10	Luxury Leather Watch	Fossil	Series 7	45mm	Pink	232	0
11	11	Minimalist Watch	Daniel Wellington	G-Shock	35mm	Black	0	0
12	12	Luxury Leather Watch	Tag Heuer	Series 7	36mm	Blue	0	0
13	13	Elegant Quartz Watch	Michael Kors	Classic Petite	43mm	Rose Gold	0	0
14	14	Minimalist Watch	Fossil	Premier	32mm	Pink	0	4
15	15	Classic Analog Watch	Tag Heuer	Premier	39mm	Space Gray	0	0
16	16	Chronograph Watch	Casio	Seamaster	45mm	Blue	0	0
17	17	Luxury Leather Watch	Omega	Carrera	36mm	Silver	0	0
18	18	Elegant Quartz Watch	Apple	G-Shock	43mm	Pink	0	0
19	19	Smartwatch with GPS	Omega	Carrera	42mm	Blue	0	0
20	20	Smartwatch with GPS	Rolex	G-Shock	36mm	Pink	0	311
21	21	Diver's Watch	Citizen	Jacqueline	39mm	White	0	0
22	22	Classic Analog Watch	Omega	Series 7	45mm	Brown	0	0
23	23	Chronograph Watch	Seiko	Series 7	35mm	Pink	76	0
24	24	Elegant Quartz Watch	Citizen	Oyster Perpetual	39mm	Space Gray	0	0
25	25	Minimalist Watch	Omega	Seamaster	36mm	Space Gray	0	0
26	26	Minimalist Watch	Tag Heuer	Jacqueline	32mm	Silver	0	0
27	27	Luxury Leather Watch	Daniel Wellington	Parker	38mm	Silver	0	0
28	28	Classic Analog Watch	Daniel Wellington	Premier	35mm	Brown	0	0
29	29	Classic Analog Watch	Rolex	Parker	35mm	White	0	0
30	30	Fashion Watch	Fossil	Jacqueline	39mm	White	0	0
31	31	Elegant Quartz Watch	Tag Heuer	Premier	43mm	Space Gray	0	0
32	32	Minimalist Watch	Seiko	Seamaster	32mm	Pink	45	0
33	33	Elegant Quartz Watch	Citizen	Jacqueline	45mm	Brown	0	0
34	34	Luxury Leather Watch	Fossil	Eco-Drive	36mm	White	0	0

```
@shop.route("/import_stock", methods=["POST"])
@login_required
def import_stock():
    """ ...

    # region GET ALL DATA AND READ THE FILE -----
    # Get the file and check if it exists
    if not "import_file" in request.files:
        return jsonify({"error": "No file found"})
    file = request.files["import_file"]
    if not file:
        return jsonify({"error": "File doesn't exists"})

    # Get the current user
    user: User = current_user
    if user.isBoss:
        return jsonify({"error": "You can't import a file as a boss"})

    # Get the shop of the user
    shop: Shop = user.shop
    if not shop:
        return jsonify({"error": "No shop found"})

    # Read the excel file with pandas
    xl_file = pandas.ExcelFile(file)
    data = xl_file.parse(xl_file.sheet_names[0])
```

La fonction « import_stock » n'est accessible que par des utilisateurs connectés et utilise la méthode POST. Elle reçoit le fichier excel grâce à l'objet de requêtes. Dans le cas où le fichier n'existerait pas, un message d'erreur serait renvoyé.

Afin de lire le fichier excel, l'utilisation de la librairie « pandas » est requise.

Une fois le fichier excel converti en dictionnaire, il est possible d'ajouter les éléments en base de données.

```
# Get the rows of the file
for index, row_row in data.iterrows():
    # Convert the row to a dictionary
    row = row_row.to_dict()

    # Get the article if it exists
    article = Article.query.filter_by(
        description=row.get("Description"),
        brand=row.get("Brand"),
        collection=row.get("Collection"),
        size=row.get("Size"),
        color=row.get("Color")).first()

    if not article:
        # Set the article as a new article
        article = Article(
            description=row.get("Description"),
            brand=row.get("Brand"),
            collection=row.get("Collection"),
            size=row.get("Size"),
            color=row.get("Color"))
        db.session.add(article)
        db.session.commit()

    # Get the shop article if it exists
    shop_article: ShopArticle = ShopArticle.query.filter_by(
        fk_shop=shop.id,
        fk_article=article.id).first()

    if not shop_article:
        # Set the shop article
        new_shop_article = ShopArticle(
            fk_shop=shop.id,
            fk_article=article.id,
            unitsStored=row.get("Stock"),
            unitsSold=0)
        db.session.add(new_shop_article)
        db.session.commit()
    else:
        # Update the shop article
        shop_article.unitsStored += row.get("Stock")
        db.session.commit()

    # Check if there is an order on this article
    if row.get("Order") > 0:
        order = Order(
            user_id=user.id,
            article_id=article.id,
            shop_id=shop.id,
            units=row.get("Order"))
        db.session.add(order)
        db.session.commit()
```

La fonction itère ensuite sur le dictionnaire reçu par le fichier excel.

Pour chaque article, il est vérifié si l'article existe ou non déjà et s'il existe aussi dans le magasin actuel en tant que stock ou commande.

- Si oui, le stock ou commandes demandés est ajouté à la somme de base.

- Si l'article existe, mais n'est pas présent dans le magasin, l'article existant sera ajouté au magasin ciblé avec les valeurs demandées dans le fichier.
- Si non, un nouvel article est créé en base de données avec des valeurs de base comme précisé dans le fichier.

3.1.4.3 Fetch de données

```
/**
 * Custom hook to fetch data from the API
 * @returns => A function to fetch data from the API, the data fetched, the loading state and the error
 *
 * @example
 * const [ makeApiCall, data, loading, error ] = useFetch<{shop: IShop, shops: IShop[]}>();
 */
export const useFetch = <DataType> (): [ makeApiCall: <T extends DataType>(url: string, method: string, body: any = null, contentType: string = 'application/json') => Promise<T>, data: T | null, loading: boolean, error: string ] => {
  const [data, setData] = useState<DataType>[null](null); // Data fetched from the URL
  const [loading, setLoading] = useState<boolean>(false); // Loading state
  const [error, setError] = useState<string>(); // Error state

  const location = useLocation(); // Get the current location

  /**
   *
   * @param url URL to fetch data from
   * @param method Method to use for fetching data
   * @param body Body to send with the request
   * @returns
   */
  const makeApiCall = <T extends DataType> (url: string, method: string, body: any = null, contentType: string = 'application/json') => Promise<T> {
    // ...
  }

  // Return the data, loading state and error
  return [ makeApiCall, data, loading, error, setData ];
}
```

Afin de permettre la communication entre le serveur et le frontend une fonction react personnalisée de « hook » nommé « useFetch » est définie 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

35 - Hook permettant les appels d'API

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 :

```
const [ shopApiCall, shopData, shopLoading, shopError ] = useFetch<{shop: IShop, shops: IShop[]}>();
```

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 comme montré ci-dessous.

```
/**
 * Function to fetch data from the API
 * @param url URL to fetch data from
 * @param method Method to use for fetching data
 * @param body Body to send with the request
 * @returns
 */
const makeApiCall = <T extends DataType> (url: string, method: string, body: any = null, contentType: any = null, displayError: Boolean = true): Promise<T> =>
{
  // If no URL is provided, return
  if (!url) return Promise.reject("No URL provided");

  // Set the API URL
  const API_URL = Configs.API_URL + url;

  // Fetch the data from the URL sent
  return fetch(API_URL,
  {
    method: method,
    headers: contentType !== null ?
    {
      "Content-Type": contentType,
    } : undefined,
    body: body instanceof FormData ? body : (body ? JSON.stringify(body) : null),
  })
}
```

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 »

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

```
.then((res) =>
{
    setLoading(true);
    setError("");

    // Get the status code
    const status = res.status.toString();

    // If the status code is not starting with 2, display the error
    if(!status.startsWith("2") && displayError && location.pathname !== `/${status}`)
    {
        // Redirect to the associated error page
        switch (status)
        {
            case "401":
                window.location.href = "/401";
                break;
            case "403":
                window.location.href = "/403";
                break;
            case "404":
                window.location.href = "/404";
                break;
        }

        // Set the error
        res.json().then((data) => displayError && setError(data.message));
        return;
    }

    // Return the data
    return res.json();
})
.then((data) =>
{
    // Set the data and return it
    setData(data);
    return data;
})
.finally(() => setLoading(false));
```

38 - Résultats de l'appel d'API

3.1.4.4 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.

```
@auth.route('/login', methods=['POST'])
def login():
    """
    Login the user to the server
    """
    # Get the email and password from the request
    email: str = request.json.get('email')
    password: str = request.json.get('password')

    # Get the user
    user: User = User.query.filter_by(email=email).first()

    # Check if user exists and if the password is correct
    if not user or not user.check_password(password):
        return {'message': 'Invalid credentials'}, 405

    # Log the user in
    login_user(user, remember=True, force=True)

    # Return logged in message
    return jsonify({'message': 'Logged in', 'user': user.to_dict_raw()}), 200
```

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...

Ensuite, 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.

```
def to_dict_raw(self):
    """
    Get the dictionary representation of the user

    Returns:
        dict: The dictionary representation of the user
    """
    return {
        "id_user": self.id,
        "name": self.name,
        "email": self.email,
        "isBoss": self.isBoss == 1,
        "fk_shop": self.fk_shop
    }
```

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 d'obtenir les variables d'états récupérées.

```
// Get makeApiCall function, data, loading and error from useFetch hook
const [makeApiCall, data, loading, error] = useFetch<{user: IUser}>();
```

41 - Utilisation de "useFetch" pour la connexion utilisateur

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

```
// Set the states of email and password
const [email, setEmail] = useState<string>("");
const [password, setPassword] = useState<string>("");
```

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.

```
// Return html elements
return (
  <UserContext.Provider value={{ actualUser, login, logout, loading, error }}>
    {children}
  </UserContext.Provider>
)
```

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 :

```
// Get user context
const { login, loading, error } = useContext<IUserContext>(UserContext as any);
```

44 - Récupération des variables du contexte

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

```
const login = (email: string, password: string) => makeApiCall("/login", "POST", { email, password }, "application/json")
```

45 - Fonction de connexion à l'utilisateur

```
/**
 * Check User Session and redirect if needed
 * @param data => Data fetched from the API
 *
 * @returns {void}
 *
 * @example
 * checkUserSession(data);
 */
function checkUserSession(data: {user: IUser}|any): void {
  // If data fetched and user is set, set the user
  if (data && data.user) {
    // Set the user fetched
    setUser(data.user);

    // Check if user is already logged in
    if (location.pathname === "/" ) {
      window.location.href = "/home";
    }
  }
}
```

Il 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

```
// Check for the user session on page load
useEffect(() => {
  // Fetch user data
  makeApiCall("/@me", "GET", null, null, location.pathname !== "/").then((data) => checkUserSession(data));
}, []);
```

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.

```
@auth.route('/@me', methods=['GET'])
@login_required
def me():
    """ ...

    # Return the current user
    return jsonify({'user': current_user.to_dict_raw()}), 200
```

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.

```
actualUser && actualUser.isBoss &&
<td className="p-2">
  {
    order.status.toLowerCase() === "pending" &&
    <>
      <button id={`approve-${order.id_order}`} onClick={(e) => handleStatus(e)}
        className="bg-colorpalette-status-ok bg-opacity-50 w-10 p-1 mx-1 hover:bg-opacity-80">
        ✓
      </button>
      <button id={`refuse-${order.id_order}`} onClick={(e) => handleStatus(e)}
        className="bg-colorpalette-status-notOk bg-opacity-50 w-10 p-1 mx-1 hover:bg-opacity-80">
        X
      </button>
    </>
  }
</td>
```

49 - Droits utilisateur, visibilité des boutons

3.1.5 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.

3.1.5.1 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.

```
// Import includes
import Main from "../includes/Main";
import Sidebar from "../includes/Sidebar";
```

50 - Importation des composants

```
// Return the dashboard component
return (
  <Main>
    <Sidebar />
    <div className="w-full p-16">
      <div className="flex">
        <h1 className="text-4xl flex-1">
          { shopData?.shop ?
            shopData.shop.name : "All shops"}
          </h1>
        </div>
        <article>
          <div className="p-5"></div>
          {articleData?.articles && orderData?.orders &&
            <InventorySummary articles={articleData.articles}
              orders={orderData.orders} />
          }
          {shopData?.shops && shopData.shops.map((shop) =>
            <Shop shop={shop} apiEndpoint={"orders"} />
          )}
        </article>
      </div>
    </Main>
  );
```

51 - Tableau de bord utilisant les composants Main et Sidebar

3.1.5.2 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.

```
/**
 * Manage the routes of the application
 * @returns => Router component
 */
function Router()
{
  // Return the router
  return (
    <BrowserRouter>
      <UserContext>
        <Routes>
          <Route path="/" element={<Login />} />
          <Route path="/home" element={<Home />} />
          <Route path="/dashboard" element={<Dashboard />} />
          <Route path="/orders" element={<Orders />} />
          <Route path="*" element={<E404 />} />
          <Route path="/401" element={<E401 />} />
          <Route path="/403" element={<E403 />} />
        </Routes>
      </UserContext>
      <Footer />
    </BrowserRouter>
  );
}
```

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 « Sidebar » introduit précédemment.

```
<nav className="font-bold flex-1">
  <ul className="relative h-full">
    <li>
      <NavLink className={({ isActive }) =>
        `${isActive ? "bg-colorpalette-active" : ""} py-4 pl-2 block` to={"/home"}>
        Home
      </NavLink>
    </li>
    <li>
      <NavLink className={({ isActive }) =>
        `${isActive ? "bg-colorpalette-active" : ""} py-4 pl-2 block` to={"/dashboard"}>
        Dashboard
      </NavLink>
    </li>
    {actualUser && actualUser.isBoss &&
      <li>
        <NavLink className={({ isActive }) =>
          `${isActive ? "bg-colorpalette-active" : ""} py-4 pl-2 block` to={"/orders"}>
          Orders
        </NavLink>
      </li>
    }
    <li className="absolute inset-x-0 bottom-0">
      <button id="btnLogout" onClick={() => logout()}
        className="w-full h-full text-left py-4 pl-2 bg-colorpalette-button-disconnect hover:bg-opacity-80">
        Logout
      </button>
    </li>
  </ul>
</nav>
```

53 - Contenu du composant Sidebar pour la navigation

3.2 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.

Nom du test	Nom du sous-test	Conditions de lancement /prérequis	Etape	Resultat attendu	Résultat	Remarques
Importation de fichier excel	Importation de stock	Aucun article stocké en base de données OU Base de donnée déjà remplie	Se rendre sur le site	Affichage de la page de connexion	Ok	
			Se connecter à l'aide d'un utilisateur Manager	Connexion réussie	Ok	
			Se rendre sur le tableau de bord	Affichage du tableau de bord avec succès	Ok	
			Itérer sur tout le tableau et récupérer les unités stockées par article			
			Récupérer le fichier excel contenant le stock a ajouter et stocker les valeurs			
			Importer un fichier excel de stock sur le site			
			Récupération des nouvelles valeurs du tableau			
			Comparaison des anciennes valeurs stockée par rapport aux nouvelles	Les nouvelles valeurs doivent être égales à : anciennes valeurs + valeurs à rajouter du fichier excel	Ok	
	Importation de commandes	Aucune commande stockée en base de données OU Base de donnée déjà remplie	Se rendre sur le site	Affichage de la page de connexion	Ok	Relancer une deuxième fois les tests si la base de données n'était pas préalablement remplie pour tester les 2 cas
			Se connecter à l'aide d'un utilisateur Manager	Connexion réussie	Ok	
			Se rendre sur le tableau de bord	Affichage du tableau de bord avec succès	Ok	
			Itérer sur tout le tableau et récupérer les unités stockées par article			
			Récupérer le fichier excel contenant les commandes a ajouter et stocker les valeurs			
			Importer un fichier excel de commandes sur le site			
			Se connecter à l'aide d'un utilisateur Patron	Connexion réussie	Ok	
			Se rendre sur la page des commandes	Affichage de la page des commandes	Ok	
			Accepter et refuser 1 commande sur 2	Le statut de la commande à dû changer	Ok	
			Récupération des nouvelles valeurs du tableau			
			Comparaison des anciennes valeurs stockée par rapport aux nouvelles	Les nouvelles valeurs doivent être égales à : anciennes valeurs + valeurs à rajouter du fichier excel	Ok	

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

Connexion utilisateur	Manager	Base de données remplie avec les utilisateurs requis	Se connecter à un compte Manager	Connexion réussie	Ok
			Vérifier le texte de la page d'accueil	La page doit contenir tout le texte destiné à un utilisateur de type Manager	Ok
			Vérifier le contenu du tableau de bord	Le tableau de bord doit contenir 2 tableaux séparément pour les articles et les commandes	Ok
			Vérifier si l'utilisateur peut se rendre sur la page des commandes	L'utilisateur ne devrait pas pouvoir s'y rendre et retrouver une page d'erreur	Ok
			Quitter la page d'erreur en cliquant sur le bouton de retour de celle-ci	La page d'accueil devrait s'afficher	Ok
			Essayer de déclencher toutes les pages d'erreurs		
			Se rendre sur une page inexistante	Affichage de la page d'erreur 404	Ok
			Se rendre sur une page dont les droits ne permettent pas	Affichage d'une page d'erreur	Ok
			Se déconnecter du compte	Deconnexion réussie	Ok
	Patron	Base de données remplie avec les utilisateurs requis	Se connecter à un compte Patron	Connexion réussie	Ok
			Vérifier le texte de la page d'accueil	La page doit contenir tout le texte destiné à un utilisateur de type Patron	Ok
			Vérifier le contenu du tableau de bord	Le tableau de bord doit contenir X tableaux en fonction du nombre de magasins dans la base de données	Ok
			Vérifier si l'utilisateur peut se rendre sur la page des commandes	L'utilisateur devrait pouvoir s'y rendre	Ok
			Essayer de déclencher toutes les pages d'erreurs	Une page d'erreur 404 devrait s'afficher	Ok
			Se déconnecter du compte	Deconnexion réussie	Ok
	Utilisateur inexistant	Aucun	Se connecter à un compte inexistant	Reste sur la page de login (n'est pas redirigé)	Ok
			Essayer de déclencher toutes les pages d'erreurs		
			Se rendre sur une page inexistante	Affichage de la page d'erreur 404	Ok
			Se rendre sur une page dont les droits ne permettent pas	Affichage d'une page d'erreur 401 ou 403	Ok

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.

```
Running: importExcelFile.ts (1 of 2)

Login to a manager and import stock and orders that will be accepted and refused by the boss, then verify the data
  login to a manager user, get the actual stored units, import a stock, check again the new values
    ✓ Get the actual stored units (2320ms)
    ✓ Import a stock with an excel file (7095ms)
    ✓ Check if the datas are correctly displayed with the new values (2217ms)
  login to a manager user, get the actual stored units, import an order, accept and refuse with a boss and check again the new values
    ✓ Get the actual stored units (2051ms)
    ✓ Import orders excel file (7136ms)
    ✓ Login as a boss and accept or refuse the orders (4608ms)
    ✓ Check if the datas are correctly displayed with the new values (2094ms)
7 passing (28s)
```

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

```
Running: loginUser.ts (2 of 2)

Login multiple users and check for his rights are correct
  Login with a manager user and make him go on multiple pages that he has rights and not
    ✓ Check the home page text, user and shop name (1034ms)
    ✓ Check the user dashboard page, and check the content (6226ms)
    ✓ Check if the user can go to the orders page (898ms)
    ✓ Try to trigger all the error pages that the user can have (7276ms)
  Login with a boss user and make him go on multiple pages that he has rights and not
    ✓ Check the home page text (969ms)
    ✓ Check the user dashboard page, and check the content (6129ms)
    ✓ Check if the user can go to the orders page (5911ms)
    ✓ Try to trigger all the error pages that the user can have (6091ms)
  Try to login with non existant user and trigger errors
    ✓ Try to login a non existant user (10782ms)
    ✓ Try to trigger all the error pages that the user can have (7690ms)
10 passing (54s)
```

57 - Resultat de tests automatique cypress de connexion utilisateur

3.3 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.

3.4 Liste des documents fournis

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

4 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 fini 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.

4.1 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 peut 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.

4.2 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.

4.3 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.

4.4 Difficultés

4.4.1 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, 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 était 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.

```
from flask_cors import CORS; CORS(app, origins=[FRONTEND_URL])
```

58 - Tentative d'autorisation des CORS

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.

4.5 Evolutions et améliorations possibles du projet

Le projet bien que complètement fonctionnel peut être sujet à des améliorations et évolutions.

- **Prix des articles** : L'évolution la plus visible serait d'ajouter les prix de chaque article. Cela permettrait une gestion plus complète des stocks. Il serait donc possible de voir le chiffre d'affaires entier et de chaque magasin sachant que le compte des articles vendus est déjà implémenté.
- **Graphiques** : Ajouter des graphiques permettant de traquer les ventes sur l'années. Cela est utile afin d'effectuer des analyses et définir les articles les plus vendu selon la date, la saison ou encore l'année.

5 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 caractères avec une longueur fixe à partir d'un autre ensemble de caractères 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.

6 Annexes

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

6.1 Sources – Bibliographie

6.1.1 Documentations officielles des outils utilisés

<https://www.docker.com>

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

<https://tailwindcss.com>

<https://fr.react.dev>

6.1.2 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>

6.2 Résumé du rapport du TPI

6.2.1 Situation de départ

Le projet de base consiste à créer une application web de stockage de montres afin d'y gérer les stocks de divers magasins d'une chaîne.

L'application doit avoir un système de connexion avec différents utilisateurs qui peuvent être Manager ou Patron. Il est ensuite possible de voir les stock et commandes des différents magasins selon l'utilisateur connecté (tous lorsque l'utilisateur est un patron).

Le frontend doit être codé en JavaScript / Typescript avec la librairie react et doit suivre les maquettes de l'application faites préalablement avec FIGMA.

Les données du site doivent être gérées à l'aide d'une API en backend codée en python à l'aide du micro-framework flask et la base de données doit être un fichier SQLite.

Pour finir, ce projet doit entièrement pouvoir tourner sur un environnement docker.

6.2.2 Mise en œuvre

Lorsque l'utilisateur se rend sur le site, la page de connexion s'affiche. Sur cette page, il peut entrer ses informations personnelles (email et mot de passe) afin de se connecter. Une fois connecté, l'utilisateur peut avoir un compte patron ou manager et est capable de naviguer entre les différentes pages du site.

Le tableau de bord et la page de commandes contiennent deux ou plus de tableaux contenant des informations concernant les stocks ou les commandes.

Dans le cas où il est manager, les tableaux contiennent le stock et les commandes de son magasin. Dû à son rôle, il ne peut se rendre dans la page des commandes.

Cependant s'il est patron, il verra tous les stocks de tous les magasins sur la même page. Etant un patron, il a les accès pour se rendre dans la page des commandes. Il voit toutes les commandes effectuées de chaque magasin et peut approuver ou non celles qu'il veut.

Pour en arriver là, un diagramme de flux et des maquettes ont été réalisés pour concevoir la réalisation du projet. Ils ont été créés à l'aide de Draw.io et de Figma.

Les technologies principalement utilisées sont : Visual Studio Code, Git, Jira, Postman et Docker.

6.2.3 Résultats

L'application a pu être terminée et contient toutes les fonctionnalités demandées. Tout le système est fluide et les bonnes informations sont affichées. Les maquettes ont été respectées avec succès et quasiment aucun changement notable n'est présent.

Lorsqu'un utilisateur n'a pas les droits de faire quelque chose, le site affiche un message d'erreur ou est automatiquement redirigé sur une page d'erreur.

Les appels d'API fonctionnent correctement et renvoient les bonnes valeurs. Ainsi l'importation de stock et de commandes ajoute les éléments du fichier excel demandés et crée l'article s'il n'existe pas.

Les patrons peuvent accepter ou refuser des commandes. Cette fonctionnalité ajoute les bonnes valeurs de chaque article accepté et n'ajoute rien si refusé.

6.3 Journal de travail

Séquence 1			Date: 02 May 2024		Après-midi
Tâche	Tranche [5min]		Explications: qu'est-ce qui se fait et comment ?	Liens, références, ...	
Analyse - Planification initiale	37	3h5min	Réalisation de la planification initiale		
Doc - Journal de travail	1	0h5min	Remplissage journal de travail		
Total tranche	38	3h10min			
Séquence 2			Date: 03 May 2024		Matin
Tâche	Tranche [5min]		Explications: qu'est-ce qui se fait et comment ?	Liens, références, ...	
Analyse - Planification initiale	12	1h	Entretien avec le premier expert de projet		
Analyse - Planification initiale	35	2h55min	Réalisation de la planification initiale		
Doc - Journal de travail	1	0h5min	Remplissage du journal de travail		
Total tranche	48	4h			
Séquence 3			Date: 03 May 2024		Après-midi
Tâche	Tranche [5min]		Explications: qu'est-ce qui se fait et comment ?	Liens, références, ...	
Doc - Rapport	12	1h	Mise en place du rapport		
Analyse - Reflexions	6	0h30min	Reflexions sur la conception de la base de données		
Doc - Schémas	19	1h35min	Création des schémas des pages du site web		
Doc - Journal de travail	1	0h5min	Remplissage journal de travail		
Total tranche	38	3h10min			

Séquence 4			Date: 06 May 2024		Matin
Tâche	Tranche [5min]		Explications: qu'est-ce qui se fait et comment ?	Liens, références, ...	
Implémentation - Configuration de l'environnement	12	1h	Préparation de l'environnement Atlassian, jira et de l'emplacement du code source et des fichiers de documentation		
Doc - Shémas	24	2h	Création des schémas des pages du site web (Accueil, Login, Stock d'un magasin (utilisateur magasin), Stock total de tous les magasins)		
Doc - Shémas	6	0h30min	Création des schémas de base de données MCD, MLD et MPD		
Analyse - Conception de l'api Flask	5	0h25min	Prévision des routes et des besoins pour l'api Flask		
Doc - Journal de travail	1	0h5min	Remplissage journal de travail		
Total tranche	48	4h			
Séquence 5			Date: 06 May 2024		Après-midi
Tâche	Tranche [5min]		Explications: qu'est-ce qui se fait et comment ?	Liens, références, ...	
Implémentation - Configuration de l'environnement	21	1h45min	Mise en place de l'environnement docker (containers et images) du backend, du frontend, de la base de données sqllite et des tests cypress		
Doc - Rapport	6	0h30min	Mise à jour du rapport		
Doc - Journal de travail	1	0h5min	Remplissage journal de travail		
Total tranche	28	2h20min			
Séquence 6			Date: 08 May 2024		Matin
Tâche	Tranche [5min]		Explications: qu'est-ce qui se fait et comment ?	Liens, références, ...	
Implémentation - Base de données	12	1h	Remplissage de la base de données avec des données temporaires		
Implémentation - Réalisation du backend	25	2h5min	Implémentation des routes backend et de l'accès à la base de données		
Doc - Journal de travail	1	0h5min	Remplissage journal de travail		
Total tranche	38	3h10min			

Séquence 7			Date: 08 May 2024		Après-midi
Tâche	Tranche [5min]		Explications: qu'est-ce qui se fait et comment ?	Liens, références, ...	
Implémentation - Réalisation du back-end	31	2h35min	Implémentation des routes backend et de l'accès à la base de données		
Doc - Rapport	6	0h30min	Mise à jour du rapport		
Doc - Journal de travail	1	0h5min	Remplissage journal de travail		
Total tranche	38	3h10min			
Séquence 8			Date: 13 May 2024		Matin
Tâche	Tranche [5min]		Explications: qu'est-ce qui se fait et comment ?	Liens, références, ...	
Implémentation - Réalisation du back-end	47	3h55min	Implémentation du login et gestion des utilisateurs		
Doc - Journal de travail	1	0h5min	Remplissage journal de travail		
Total tranche	48	4h			
Séquence 9			Date: 13 May 2024		Après-midi
Tâche	Tranche [5min]		Explications: qu'est-ce qui se fait et comment ?	Liens, références, ...	
Implémentation - Réalisation du back-end	24	2h	Avancement sur le backend de l'application (petits problèmes avec docker)		
Doc - Rapport	3	0h15min	Mise à jour rapport		
Doc - Journal de travail	1	0h5min	Remplissage journal de travail		
Total tranche	28	2h20min			

Séquence 10			Date: 15 May 2024	Matin
Tâche	Tranche [5min]		Explications: qu'est-ce qui se fait et comment ?	Liens, références, ...
Implémentation - Réalisation du back-end	37	3h5min	Finalisation du backend de l'application	
Doc - Journal de travail	1	0h5min	Remplissage journal de travail	
Total tranche	38	3h10min		
Séquence 11			Date: 15 May 2024	Après-midi
Tâche	Tranche [5min]		Explications: qu'est-ce qui se fait et comment ?	Liens, références, ...
Implémentation - Réalisation du front-end	25	2h5min	Commencement du frontend avec react en typescript et préparation des fichiers frontend nécessaires et création de l'accueil	
Doc - Rapport	12	1h	Mise à jour rapport	
Doc - Journal de travail	1	0h5min	Remplissage journal de travail	
Total tranche	38	3h10min		
Séquence 12			Date: 16 May 2024	Après-midi
Tâche	Tranche [5min]		Explications: qu'est-ce qui se fait et comment ?	Liens, références, ...
Implémentation - Réalisation du front-end	37	3h5min	Mise en place du routing react pour le frontend de l'application avec tous les pages principales	
Doc - Journal de travail	1	0h5min	Remplissage journal de travail	
Total tranche	38	3h10min		

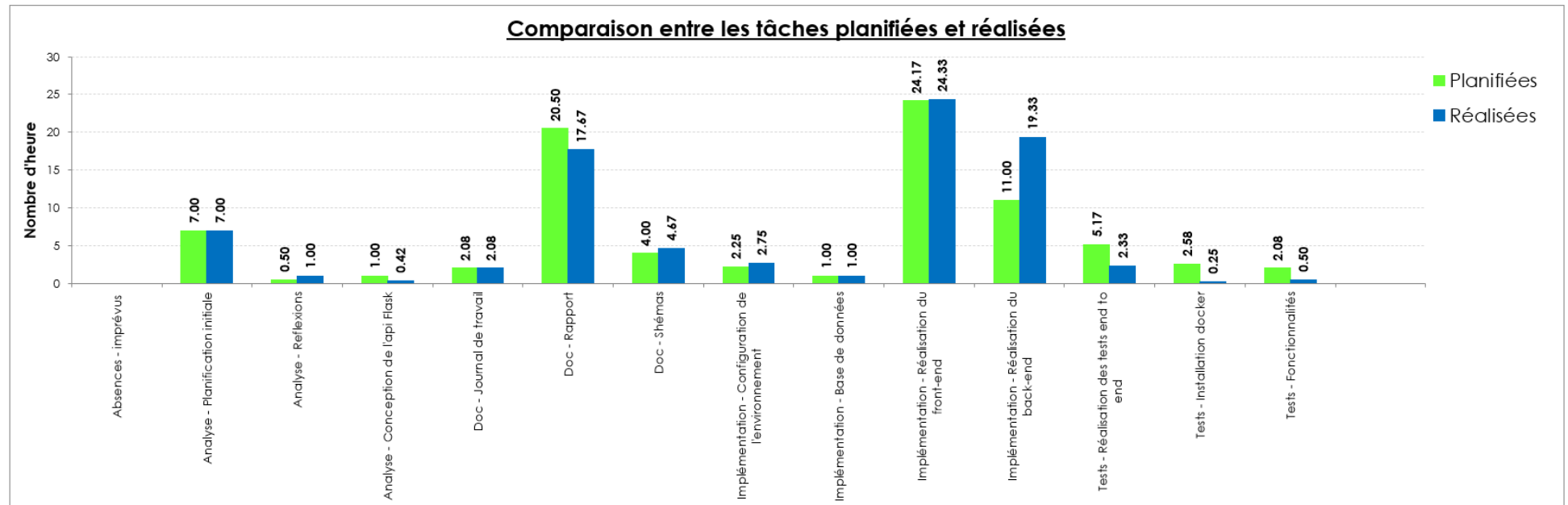
Séquence 13			Date: 17 May 2024		Matin
Tâche	Tranche [5min]		Explications: qu'est-ce qui se fait et comment ?	Liens, références, ...	
Implémentation - Réalisation du front-end	47	3h55min	Via le frontend, création d'un fichier et fonctions permettant la connexion avec le backend et mise en place des éléments nécessaires à cela et création du login utilisateur		
Doc - Journal de travail	1	0h5min	Remplissage journal de travail		
Total tranche	48	4h			
Séquence 14			Date: 17 May 2024		Après-midi
Tâche	Tranche [5min]		Explications: qu'est-ce qui se fait et comment ?	Liens, références, ...	
Implémentation - Réalisation du front-end	18	1h30min	Finalisation du login, de la session et sécurisation de celui-ci. Gérer les droits utilisateurs (Utilisateurs magasins / Utilisateur du site de gestion de stock)		
Doc - Shémas	7	0h35min	Création du schémas de flux utilisateur		
Doc - Rapport	12	1h	Mise à jour du rapport		
Doc - Journal de travail	1	0h5min	Remplissage journal de travail		
Total tranche	38	3h10min			
Séquence 15			Date: 22 May 2024		Matin
Tâche	Tranche [5min]		Explications: qu'est-ce qui se fait et comment ?	Liens, références, ...	
Implémentation - Réalisation du front-end	25	2h5min	Début d'affichage du stock d'un magasin selon les schémas à l'aide d'un utilisateur "magasin"		
Implémentation - Réalisation du back-end	6	0h30min	Correction de quelques routes en backend		
Doc - Rapport	6	0h30min	Mise à jour du rapport		
Doc - Journal de travail	1	0h5min	Remplissage journal de travail		
Total tranche	38	3h10min			

Séquence 16			Date: 22 May 2024	Après-midi
Tâche	Tranche [5min]		Explications: qu'est-ce qui se fait et comment ?	Liens, références, ...
Implémentation - Réalisation du front-end	31	2h35min	Finalisation de l'affichage du stock et du design selon les maquettes et début de création des fichiers excel afin de faire l'importation de stock	
Doc - Rapport	6	0h30min	Mise à jour du rapport	
Doc - Journal de travail	1	0h5min	Remplissage journal de travail	
Total tranche	38	3h10min		
Séquence 17			Date: 23 May 2024	Après-midi
Tâche	Tranche [5min]		Explications: qu'est-ce qui se fait et comment ?	Liens, références, ...
Implémentation - Réalisation du front-end	37	3h5min	Mise en place des fichiers excel et finalisation de la modélisation de la page de stock et des commandes	
Doc - Journal de travail	1	0h5min	Remplissage journal de travail	
Total tranche	38	3h10min		
Séquence 18			Date: 24 May 2024	Matin
Tâche	Tranche [5min]		Explications: qu'est-ce qui se fait et comment ?	Liens, références, ...
Implémentation - Réalisation du front-end	47	3h55min	Affichage du dashboard et des commandes sur les comptes admins (patron)	
Doc - Journal de travail	1	0h5min	Remplissage journal de travail	
Total tranche	48	4h		

Séquence 19			Date: 24 May 2024		Après-midi
Tâche	Tranche [5min]		Explications: qu'est-ce qui se fait et comment ?	Liens, références, ...	
Implémentation - Réalisation du front-end	25	2h5min	Finalisation du frontend		
Doc - Rapport	12	1h	Correction des maquettes et ajout de la maquette de routes		
Doc - Journal de travail	1	0h5min	Remplissage journal de travail		
Total tranche	38	3h10min			
Séquence 20			Date: 29 May 2024		Matin
Tâche	Tranche [5min]		Explications: qu'est-ce qui se fait et comment ?	Liens, références, ...	
Analyse - Reflexions	6	0h30min	Entrevue avec le 2ème expert et point de vue sur le TPI, problèmes, questions, fin et rendu.		
Implémentation - Réalisation du back-end	31	2h35min	Correction des endpoints avec les noms des route de la maquette. Ajout de la dernière fonctionnalité manquante permettant d'importer un fichier excel afin d'ajouter ou de commander du stock. Lecture et analyse du fichier Excel, définition des commandes et du stock à importer, création des commandes à afficher à l'utilisateur patron et ajout du stock. Si un article n'existe pas de base, celui-ci est ajouter automatiquement en tant que nouvel article.		
Doc - Journal de travail	1	0h5min	Remplissage du journal de travail		
Total tranche	38	3h10min			
Séquence 21			Date: 29 May 2024		Après-midi
Tâche	Tranche [5min]		Explications: qu'est-ce qui se fait et comment ?	Liens, références, ...	
Implémentation - Réalisation du back-end	31	2h35min	Correction des derniers bugs backend concernant l'importation de stock. Lecture du fichier incorrecte, ne recupère pas les données des bonnes colonnes (1 colonne de décalage)		
Doc - Rapport	6	0h30min	Modification de la mise en page du rapport, ajout des risques techniques et présentation de la planification de ticketing JIRA sans les images. Mise en place de screenshots pour l'écriture du chapitre 3 de la réalisation.		
Doc - Journal de travail	1	0h5min	Remplissage du journal de travail		
Total tranche	38	3h10min			

Séquence 22			Date: 30 May 2024	Après-midi
Tâche	Tranche [5min]		Explications: qu'est-ce qui se fait et comment ?	Liens, références, ...
Tests - Installation docker	3	0h15min	Tests d'installation de l'environnement docker à l'aide des fichiers compose.yaml et dockerfile. Tests avec la commande "docker compose up" pour mettre en place le système et afficher le site sur un port local concluant et tous fonctionne correctement, ainsi que la commande "docker compose watch" permettant d'effectuer des modification directement dans le container sans avoir à le redémarrer concluant.	
Tests - Réalisation des tests end to end	28	2h20min	Réalisation de tous les tests end to end avec cypress : 1) Connexion et droits utilisateur => Avec les données de test, la simulation essaye de se connecter à plusieurs utilisateurs dont des faux. Permet de détecter si un message d'erreur s'affiche lorsque le mot de passe ou l'email est incorrect, regarde si la page contient les bons éléments en fonction des droits utilisateur, essaie de se diriger sur une page dont seul les utilisateurs "patron de la chaine" sont autorisés à aller. 2) Pages d'erreur => Test essayant de se rendre sur une page qui n'existe pas ou que l'utilisateur actuel n'a pas les droit et test les boutons de retour de celles-ci. 3) Importation de stock direct => Importe un fichier excel ne contenant que des articles à ajouter directement en stock et vérifie si l'importation s'est effectuée correctement en comparant les nombres affichés avant et après l'importation. 4) Commande de stock => Importe un fichier excel avec plusieurs commandes, vérifie si la commande a été ajoutée dans le tableau de bord avec le status "Pending". Se connecte avec un utilisateur "patron" et accepte une commande et en refuse une autre. Elle finit par vérifier si le stock à bel est bien été importé.	
Tests - Fonctionnalités	6	0h30min	Test de quelques fonctionnalités à la main pour vérifier correctement le rendu (connexion, tri par colonne des tableaux de stock et commandes et importation de stock et commandes)	
Doc - Journal de travail	1	0h5min	Remplissage du journal de travail	
Total tranche	38	3h10min		
Séquence 23			Date: 31 May 2024	Matin
Tâche	Tranche [5min]		Explications: qu'est-ce qui se fait et comment ?	Liens, références, ...
Doc - Rapport	47	3h55min	Finalisation du rapport : Prise de toutes les captures d'écran nécessaires et synthèse de tous les chapitres et textes à ajouter le 31 mai et 3 juin, afin de permettre l'écriture du reste le 3 juin sans problème rapidement si besoin.	
Doc - Journal de travail	1	0h5min	Remplissage du journal de travail	
Total tranche	48	4h		

Séquence 24			Date: 31 May 2024		Après-midi
Tâche	Tranche [5min]		Explications: qu'est-ce qui se fait et comment ?	Liens, références, ...	
Doc - Rapport	37	3h55min	Finalisation du rapport : Ecriture du chapitre 3 de la réalisation, arborescence, client et serveur à l'aide de la synthétisation du matin. Mise en place des images et ajout de leurs légendes		
Doc - Journal de travail	1	0h5min	Remplissage du journal de travail		
Total tranche	38	3h10min			
Séquence 25			Date: 03 June 2024		Journée
Tâche	Tranche [5min]		Explications: qu'est-ce qui se fait et comment ?	Liens, références, ...	
Doc - Rapport	47	3h55min	Finalisation du rapport : Ecriture des derniers chapitres à l'aide de la synthétisation de la semaine précédente.		
Doc - Journal de travail	1	0h5min	Ecriture de la conclusion et du résumé du TPI et ajout des images correspondantes		
Total tranche	48	4h			



Total heures planifiées 83h20min

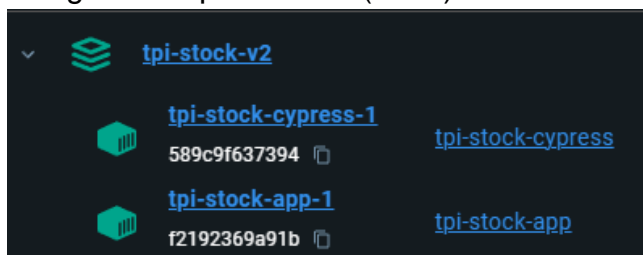
Total heures réalisées 83h20min

Différence 0h

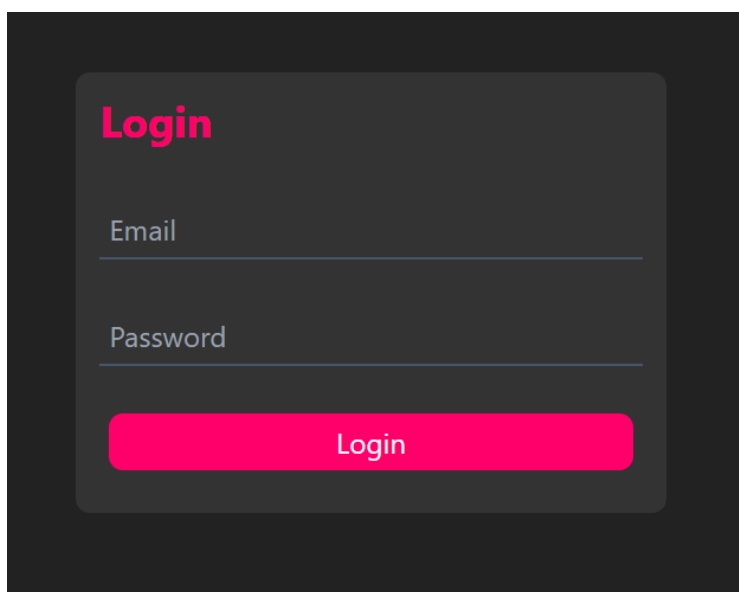
6.4 Manuel d'Installation

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 ».

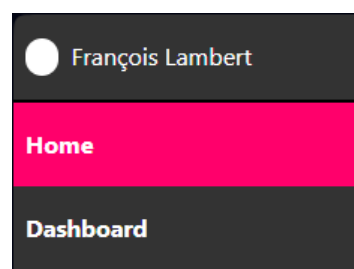
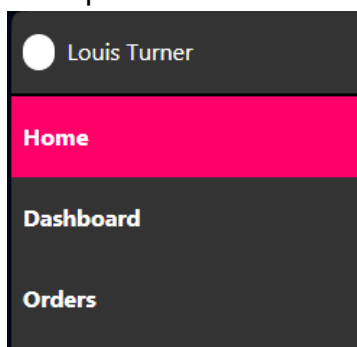
Une fois le tout compilé, les deux containers devraient être en action. Il est donc possible ensuite de voir les logs cypress en allant dans son container et d'accéder au site grâce au port ouvert (5000)



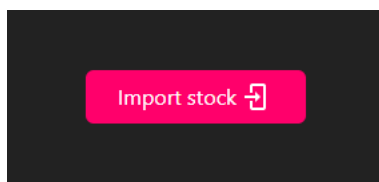
6.5 Manuel d'Utilisation



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



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



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.

Items sold 0	Items in stock 5347	Items in order 0
-----------------	------------------------	---------------------

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

Orders		ion	Brand ▼	Collect
Order No.	Art. No. ▲			
O-7	A-48	ports Watch	Apple	Oyster
O-14	A-48	Watch	Apple	Premie
O-21	A-48	Watch	Apple	Eco-Dr
		Watch	Apple	Jacque

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

Approve/Refuse
✓ X
✓ X
✓ X
✓ X
✓ X
✓ X
✓ X

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



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