Gestion de stock de montres

Dossier de projet de TPI

Damien Loup
CID4B

Table des matières

1 Analyse préliminaire 3
1.1 Introduction 3
1.2 Objectifs 3
1.3 Planification initiale 4
1.3.1 Répartition du temps prévu en % 6

2 Analyse / Conception 7
2.1 Concept 7
2.1.1 Base de données 7
2.1.2 Api backend 9
2.1.3 Diagramme de flux utilisateur 10
2.1.4 Maquettes 11
2.2 Stratégie de test 14
2.2.1 Tests de bout en bout 14
2.2.2 Tests manuels 14
2.2.3 Données de test 14
2.3 Risques techniques 15
2.4 Planification 16
2.5 Dossier de conception 18
2.5.1 Technologies 18

3 Réalisation 18
3.1 Dossier de réalisation 18
3.2 Description des tests effectués 22
3.3 Erreurs restantes 23
3.4 Liste des documents fournis 23

4 Conclusions 23

5 Annexes 24
5.1 Résumé du rapport du TPI / version succincte de la documentation 24
5.2 Sources – Bibliographie 24
5.3 Journal de travail 24
5.4 Manuel d'Installation 24
5.5 Manuel d'Utilisation 24
5.6 Archives du projet 24

1 Analyse préliminaire

1.1 Introduction

Ce projet est réalisé dans le cadre du TPI de l'ETML en entreprise chez Abraxas. Le projet consiste à créer un site web de gestion de stock de montres pour divers magasins. Chaque utilisateur est soit associé à un magasin, soit omniscient et peut voir tous les stocks de tous les magasins.

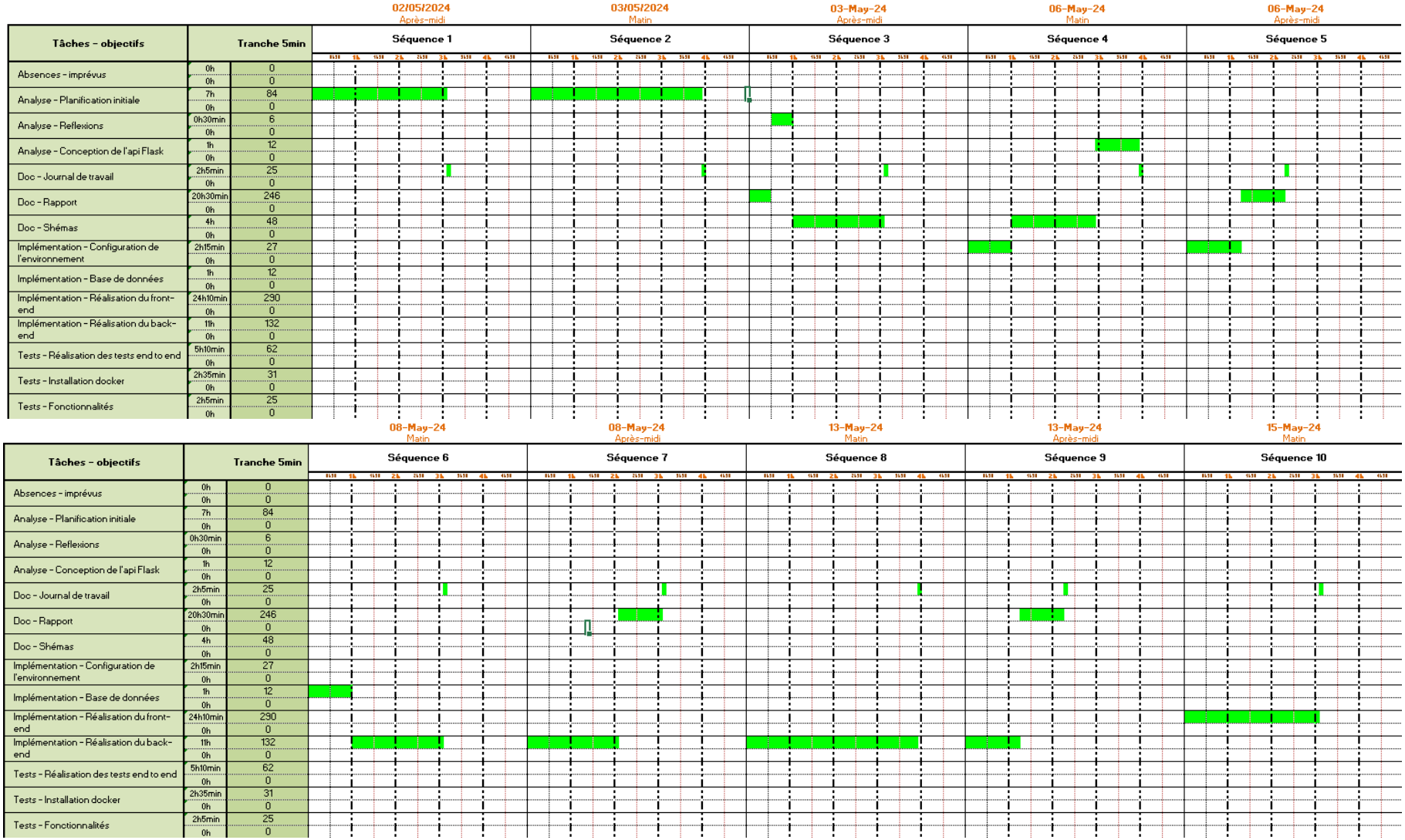
Ce projet a pour but de consolider certaines connaissances, tel que le développement web, soit les langages qui sont distinctement le typescript pour REACT et le python pour FLASK. Il en va de même pour la réalisation des tests qui utilisent la technologie CYPRESS qui permet de simuler un utilisateur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

1.2 Objectifs

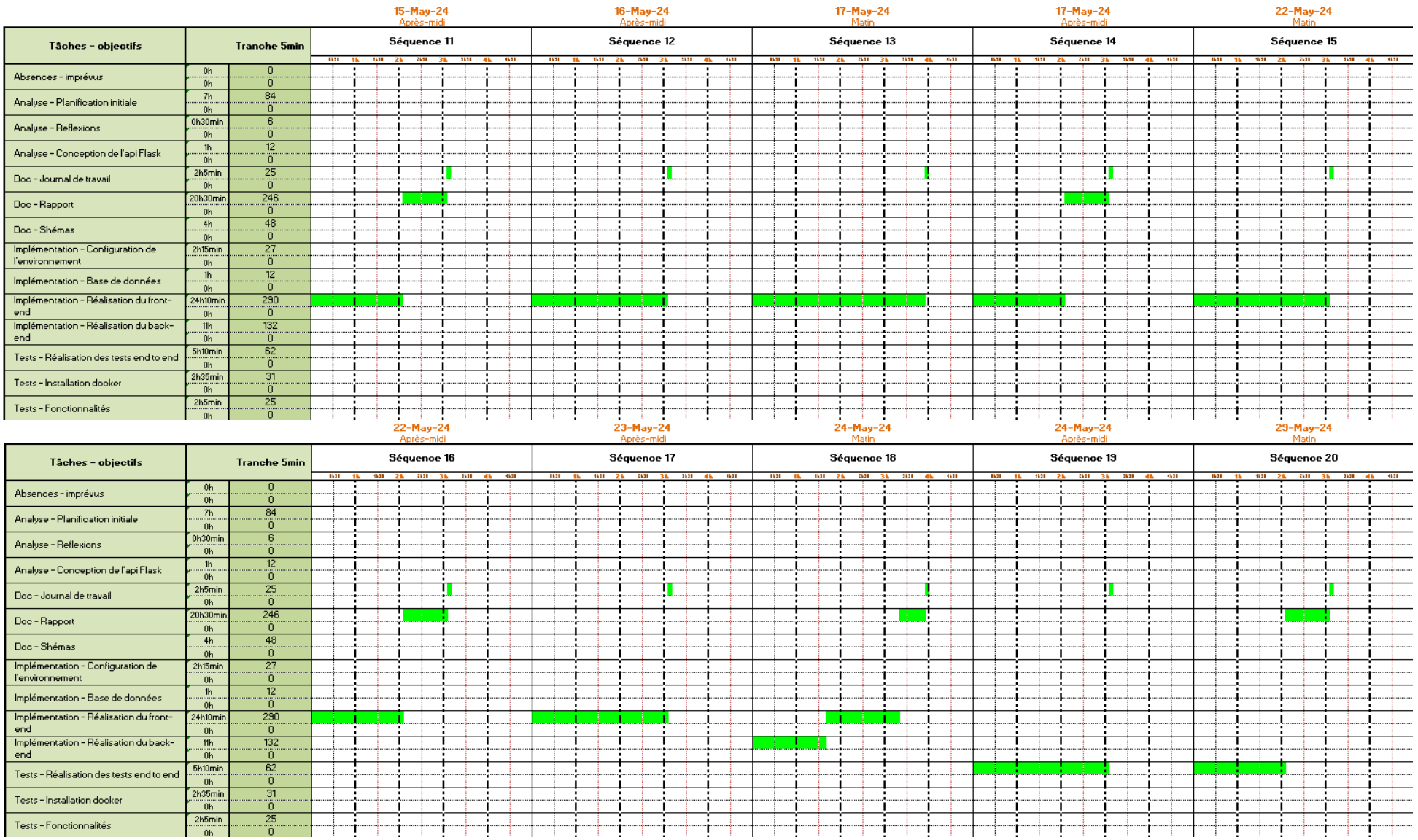
L'objectif du projet est d'avoir un site web utilisable tournant sur docker avec des tests de bout en bout contenant les fonctionnalités suivantes :

- 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 à son magasin via des fichiers Excel contenant une multitude de montres.
- Droits utilisateurs : Certains utilisateurs sont associés à des magasins et ne peuvent que voir les stocks de ceux-ci, 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 stocks : Il est possible de modifier l'état des objets dans le stock (en stock, vendu, …)

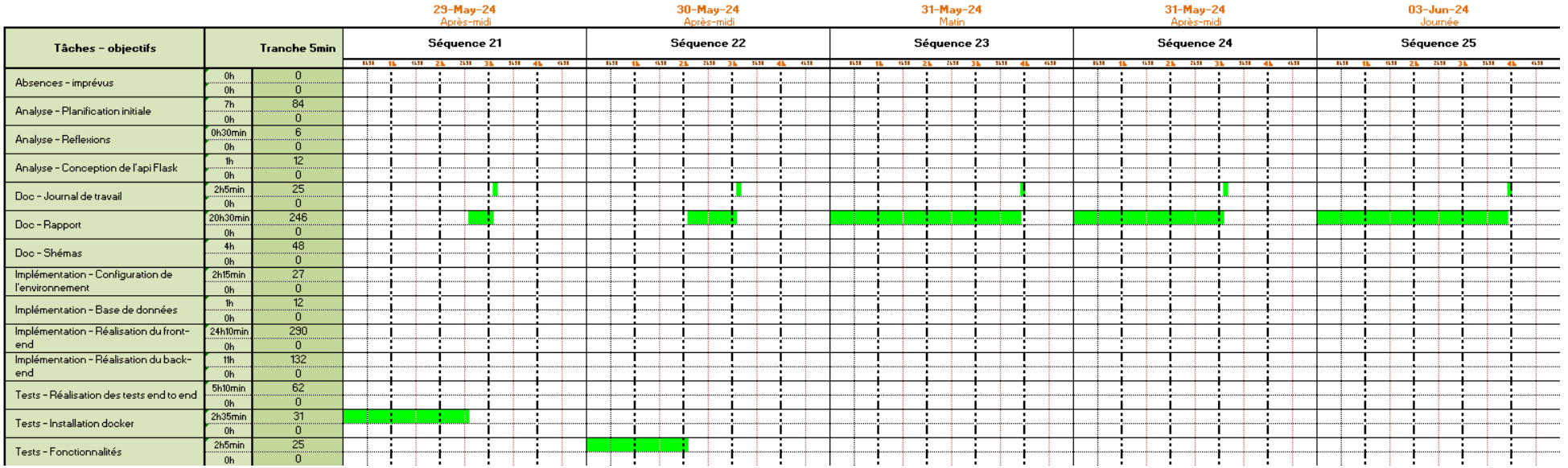
1.3 Planification initiale



Tâches – objectifs	Tranche 5min	
Absences – imprévus	0h	0
	0h	0
Analyse – Planification initiale	7h	84
	0h	0
Analyse – Reflexions	0h30min	6
	0h	0
Analyse – Conception de l'api Flask	1h	12
	0h	0
Doc – Journal de travail	2h5min	25
	0h	0
Doc – Rapport	20h30min	246
	0h	0
Doc – Shémas	4h	48
	0h	0
Implémentation – Configuration de l'environnement	2h15min	27
	0h	0
Implémentation – Base de données	1h	12
	0h	0
Implémentation – Réalisation du front-end	24h10min	290
	0h	0
Implémentation – Réalisation du back-end	11h	132
	0h	0
Tests – Réalisation des tests end to end	5h10min	62
	0h	0
Tests – Installation docker	2h35min	31
	0h	0
Tests – Fonctionnalités	2h5min	25
	0h	0



Tâches - objectifs	Tranche 5min	
Absences - imprévus	0h	0
	0h	0
Analyse - Planification initiale	7h	84
	0h	0
Analyse - Reflexions	0h30min	6
	0h	0
Analyse - Conception de l'api Flask	1h	12
	0h	0
Doc - Journal de travail	2h5min	25
	0h	0
Doc - Rapport	20h30min	246
	0h	0
Doc - Shémas	4h	48
	0h	0
Implémentation - Configuration de l'environnement	2h15min	27
	0h	0
Implémentation - Base de données	1h	12
	0h	0
Implémentation - Réalisation du front-end	24h10min	290
	0h	0
Implémentation - Réalisation du back-end	11h	132
	0h	0
Tests - Réalisation des tests end to end	5h10min	62
	0h	0
Tests - Installation docker	2h35min	31
	0h	0
Tests - Fonctionnalités	2h5min	25
	0h	0



Tâches - objectifs	Tranche 5min	
Absences - imprévus	0h	0
	0h	0
Analyse - Planification initiale	7h	84
	0h	0
Analyse - Reflexions	0h30min	6
	0h	0
Analyse - Conception de l'api Flask	1h	12
	0h	0
Doc - Journal de travail	2h5min	25
	0h	0
Doc - Rapport	20h30min	246
	0h	0
Doc - Shémas	4h	48
	0h	0
Implémentation - Configuration de l'environnement	2h15min	27
	0h	0
Implémentation - Base de données	1h	12
	0h	0
Implémentation - Réalisation du front-end	24h10min	290
	0h	0
Implémentation - Réalisation du back-end	11h	132
	0h	0
Tests - Réalisation des tests end to end	5h10min	62
	0h	0
Tests - Installation docker	2h35min	31
	0h	0
Tests - Fonctionnalités	2h5min	25
	0h	0

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

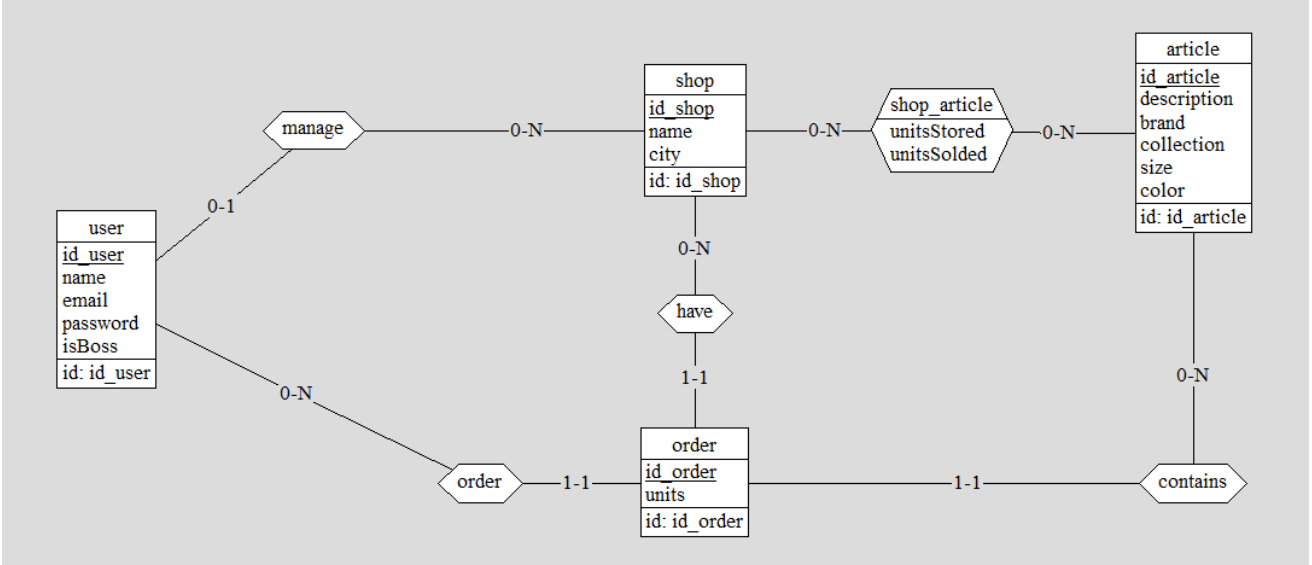
2.1 Concept

2.1.1 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.

2.1.1.1 MCD

Ce schéma permet de définir les tables et les liaisons principales entres les tables.
Ici sont définis quels utilisateurs sont managers dans quels magasins.
Chaque magasin a aussi plusieurs articles en stock et 2 d'entre eux pourraient même avoir les mêmes articles mais en quantité différentes selon les stocks indépendants.
La table « order » permet de définir les commandes qui ont été effectuées selon l'utilisateur en question et le magasin précis ce qui permet à plusieurs utilisateurs de commander des articles pour le stock de leur magasin.

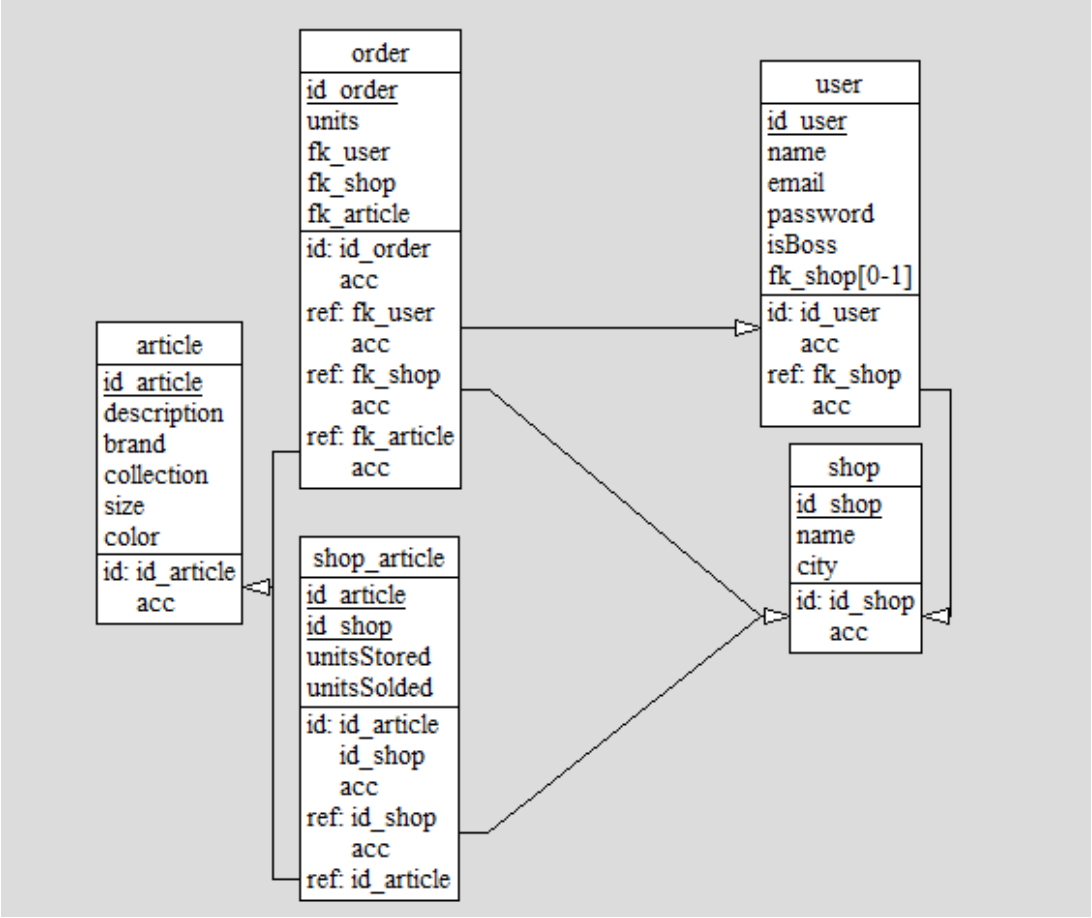


1 - Schéma mcd de la base de données

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 ⇔ 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

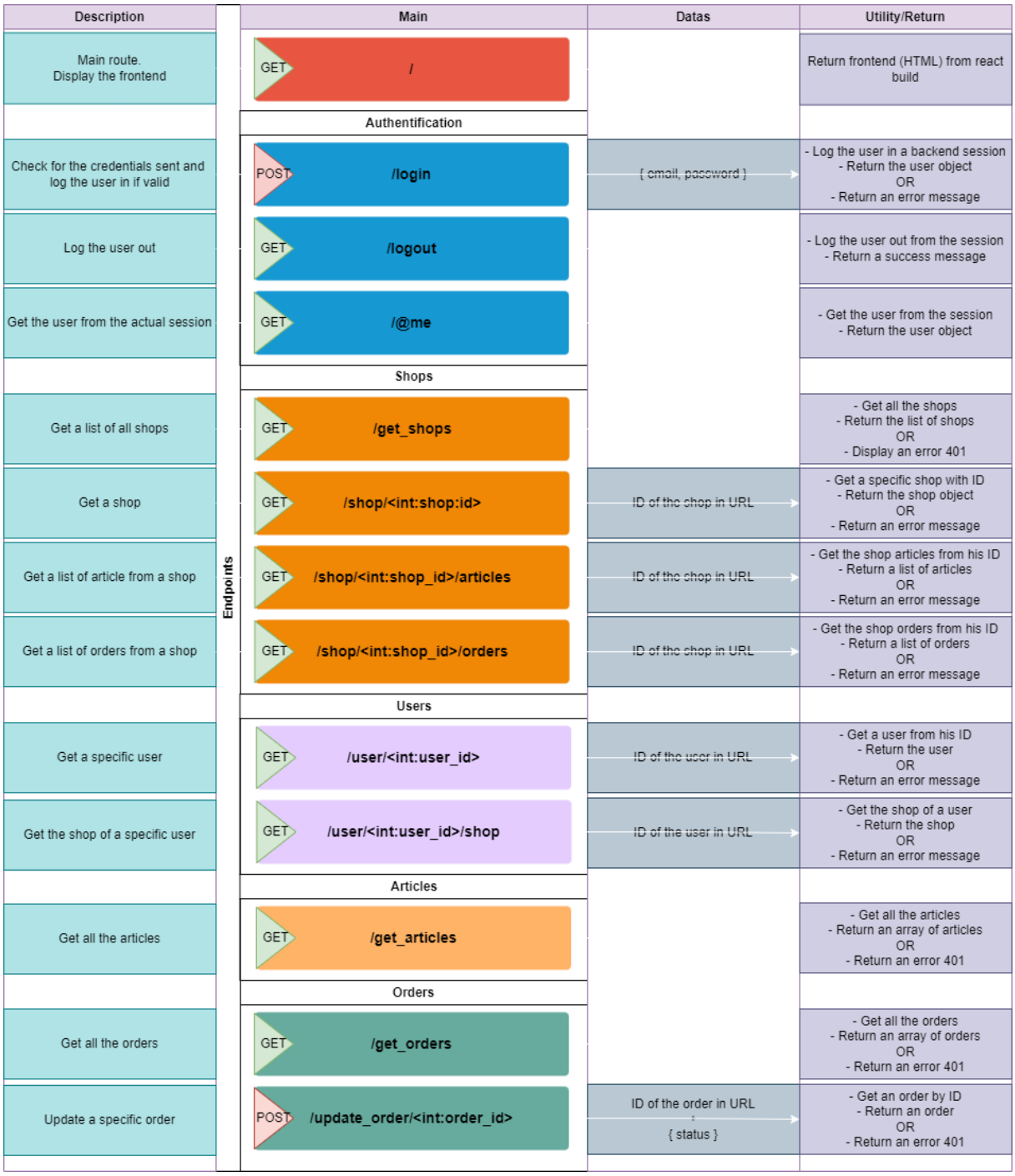
2.1.2 Api backend

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.

2.1.2.1 Routes

Voici un schéma des routes :

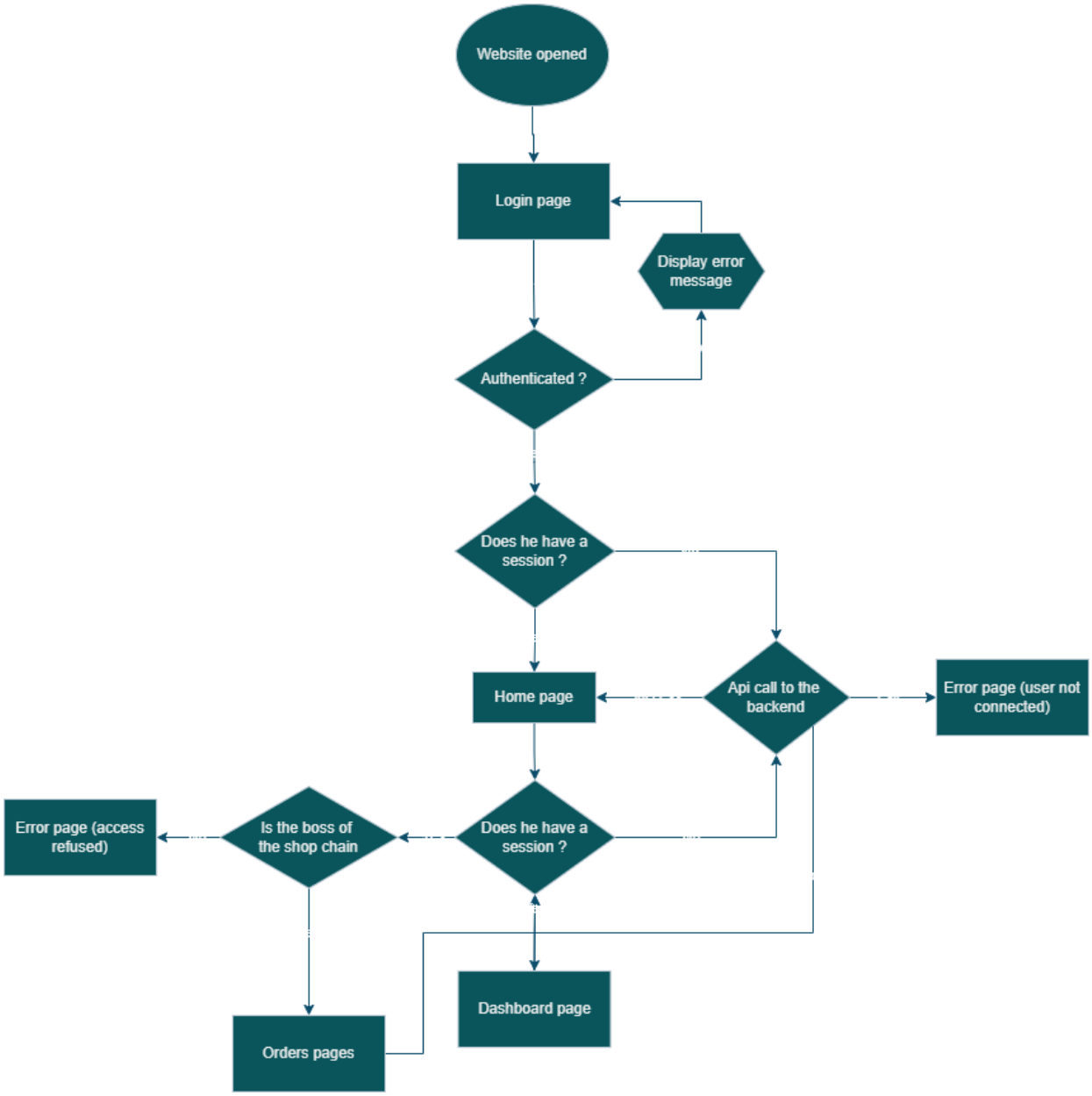


3 - Routes backend et leur fonctionnement

2.1.3 Diagramme de flux utilisateur

Durant la conception du projet un diagramme de flux à été réalisé montrant le parcours qu'un utilisateur aura lorsqu'il utilisera le site.
Ce diagramme montre non seulement comment les pages interagissent entre elles, mais aussi comment réagit le site en fonction de ce que l'utilisateur fait.

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



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

2.1.4 Maquettes

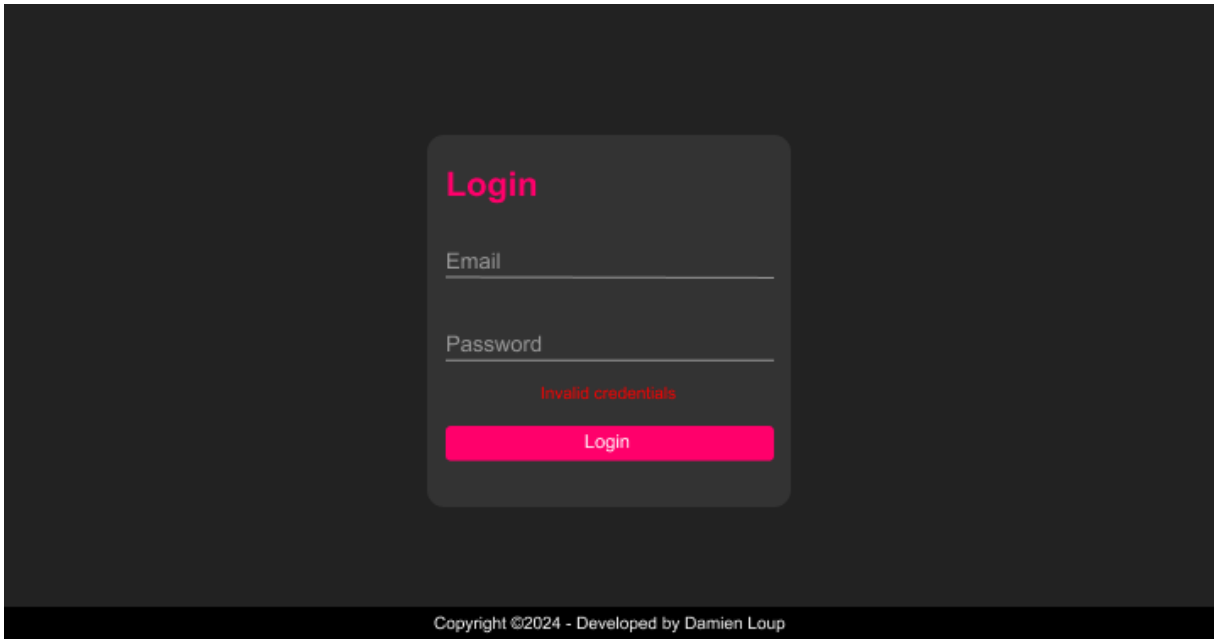
Dans le but de créer le site web, il est important de réaliser des maquettes qui reflètent le projet final. Ces maquettes ne seront pas forcément exactes par rapport au visuel final du site, mais cela permet de visualiser à quoi il devrait ressembler.

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	Eléments 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.

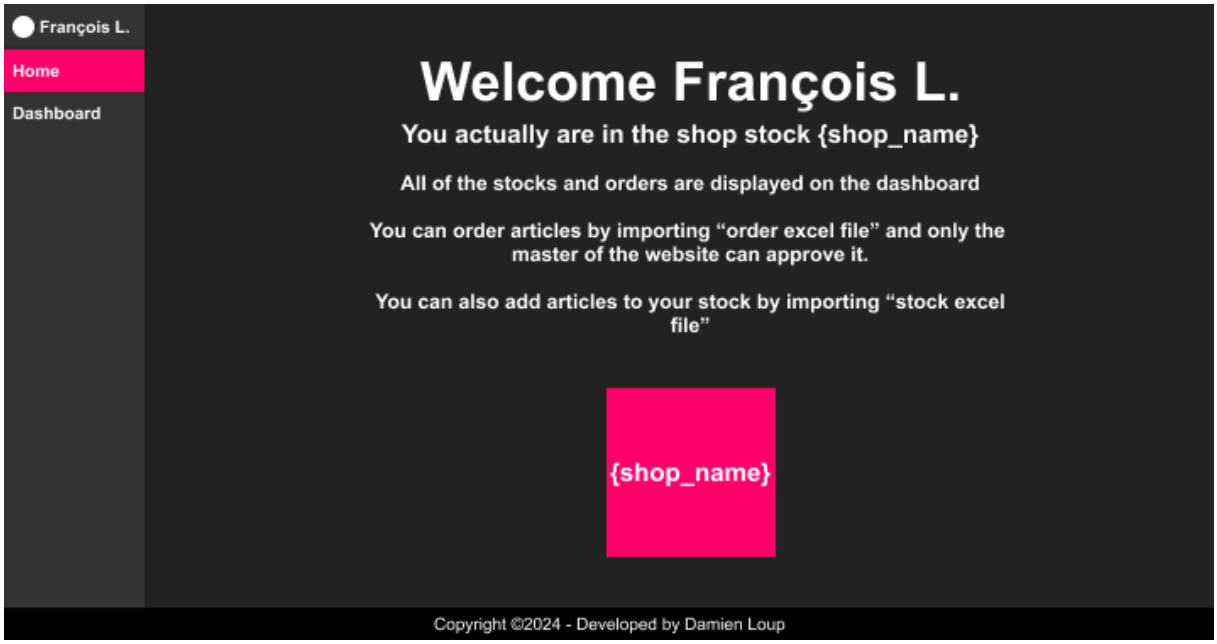


5 – Maquette page de connexion au site de stock

Une fois connecté, la page d'accueil s'affiche et comprends les informations principales sur l'utilisateur et le magasin actuel de celui-ci définit avec une icone 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, ici un manager de magasin à

uniquement accès au « Tableau de bord » qui lui permet de voir le stock et les commandes du magasin actuel.



6 - Maquette de l'accueil du site

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

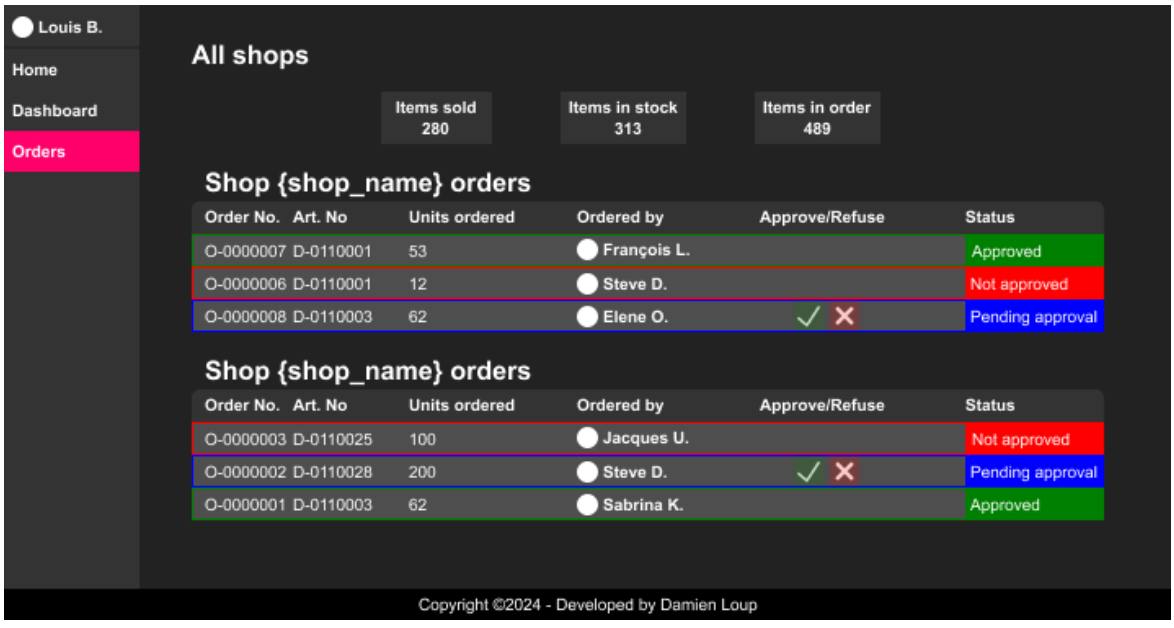
Il peut aussi importer du stock à l’aide du bouton en haut à droite « import stock ». Selon le contenu du fichier Excel, cela peut être un import direct et mettre à jour instantanément le stock ou alors exécuter une commande qui devra être approuvée par le patron de la chaine de magasins

7 - Maquette du dashboard d'un utilisateur manager

La principale différence entre les utilisateurs associés à des magasins et le patron est que lui voit les stocks et commandes de tous les magasins.

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

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



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

Sur le haut de la page, 3 éléments définissent le total de tous le stock « articles vendus », « articles en stocks » et « articles commandés » afin de simplifier l'utilisateur à voir son stock total.

En ce qui concerne les tableaux contenants le stock et les commandes, il est possible de les filtrer en cliquant sur une des en-têtes de colonne. Si l'on filtre par unités commandée, dans l'ordre 1 click permettra d'avoir les commandes décroissantes, ensuite croissantes pour revenir par défaut (numéros de commandes croissants).

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	Status
louis.u@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

Magasin :

Nom	Utilisateurs
Romanel	François L.

Romanel	Steve D.
Renens	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

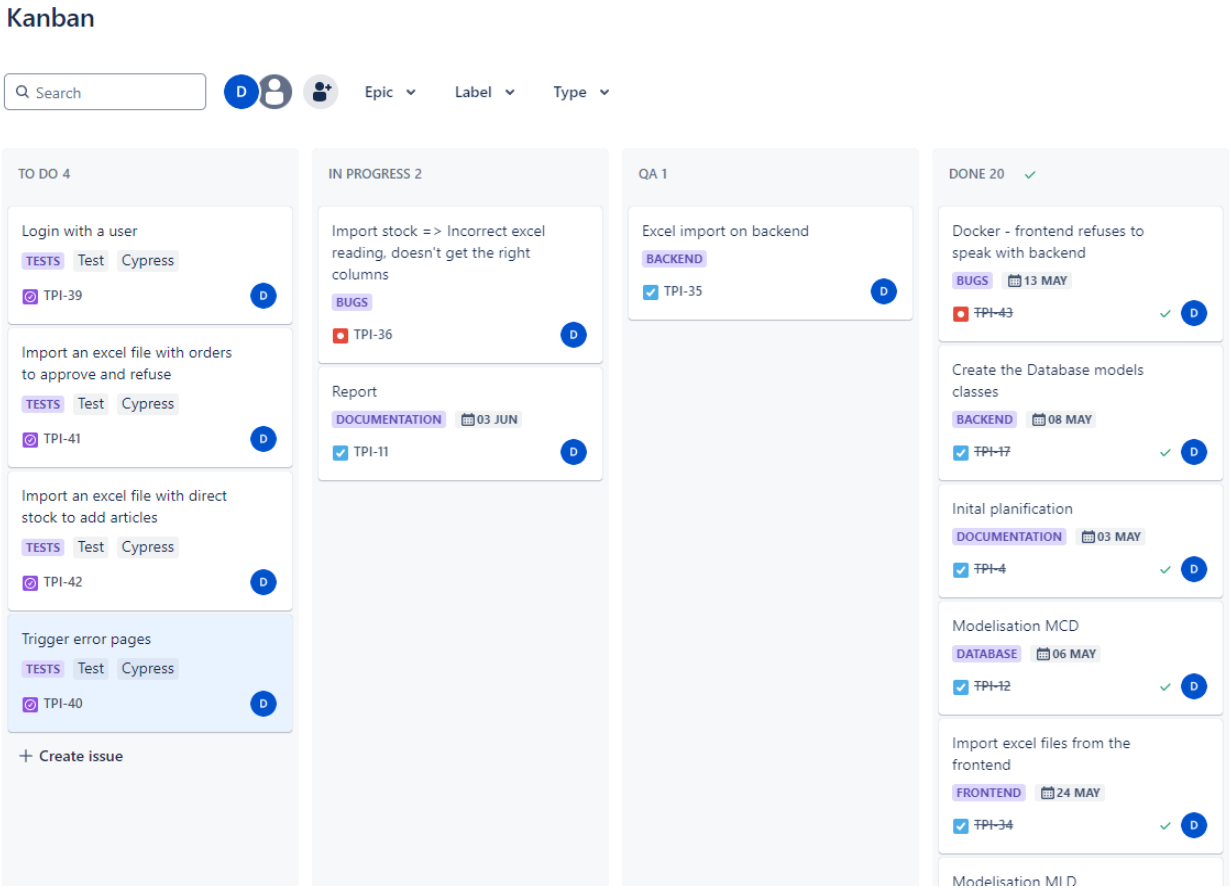
Afin de s'organiser correctement avec des tâches précises, à l'aide de JIRA dans les outils Atlassian, un projet a été créé regroupant tickets et code stocké à l'aide de git. Ce projet contient toutes les tâches et sous-tâches principales permettant la réalisation des fonctionnalités. Les tickets contiennent 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.

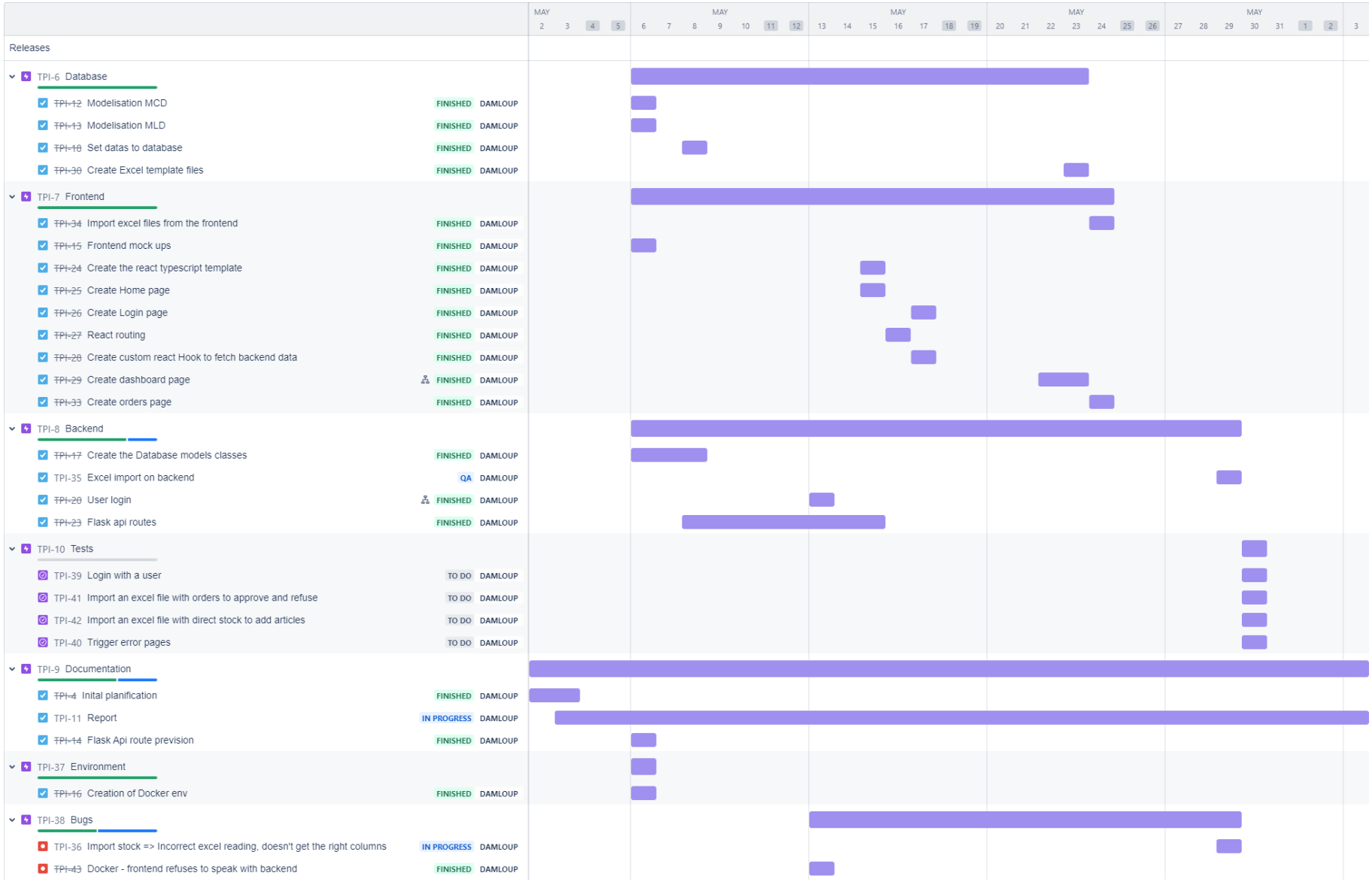
Les commits git seront aussi liés à des tickets afin de permettre la visualisation des changements.

Les aperçus des deux tableaux sont dans leur état du 29 mai 2024, afin d'avoir des exemples concrets.

Voici un aperçu du tableau KANBAN :



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



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

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

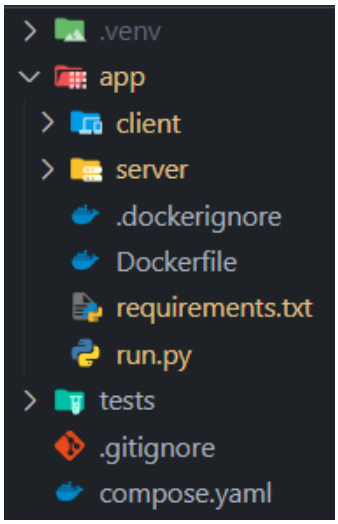
En premier lieu, le python servant à faire office de serveur et d'accès direct à la base de données. Le serveur utilise lui-même le micro-Framework nommé FLASK permettant de gérer les routes qui servent de pont entre le frontend et le backend. Ce micro-framework est très utilisé pour créer des petits projets et est donc parfait pour celui-ci.

Ensuite, vient le typescript, 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 celle-ci. Il permet donc de créer tout le design, les pages et algorithmes dans le frontend. Le framework react est donc utilisé 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 et autant de fois qu'il le faudra.

Etant donné la communication impossible de ces deux langages distincts, des appels d'api sont mis en place afin de les faire s'envoyer des données entre eux.

3.1.1 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).



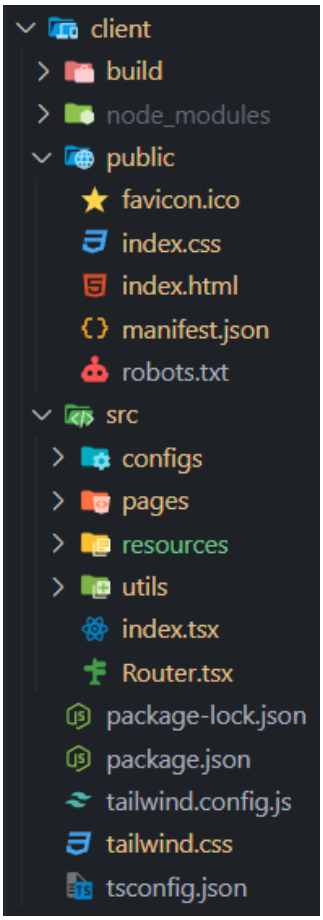
Tout d'abord, la surface est constituée d'un dossier « app » contenant l'application entière, un dossier « tests » permettant d'y mettre tous les tests automatique cypress et de fichiers tels que « .gitignore » et « compose.yaml ».
le .gitignore sert à empêcher certains fichiers ou dossiers d'être envoyés dans le dépôt git dû à la sécurité, des dossiers inutiles ou trop lourds ou alors les dossiers contenants les librairies permettant de coder (.venv pour python et node_modules pour typescript).

Par contre, compose.yaml permet lui 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.

Dans le dossier app, on y retrouve, le client et le serveur, ainsi que les fichiers docker nécessaire à la création du container de ce container-ci, ainsi que les librairies à installer contenues dans le fichier « requirements.txt ».

Et pour finir le fichier le plus important, « run.py » qui est le fichier serveur d'entrée. Il permet de lancer l'application en créant un serveur local.

3.1.1.1 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.

Le dossier « public » contient tous les fichiers de base pour que l'application fonctionne, comme un fichier html qui sera récupéré grâce à react qui modifiera son contenu en fonction du code exécuté.

Le dossier « src » contient tout le code source à la création du site. Comme son nom l'indique, « index.tsx » est la base de react et permet la récupération de la balise principale du fichier html.

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

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

« Router.tsx » est 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.

```tsx
/**
 * 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>
    );
}

export default Router;
```

10 - Composant react router contenant les pages à afficher selon l'url

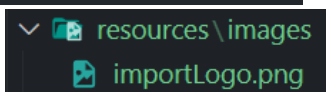
- Configurations
  Le dossier de configurations comprend la palette de couleur dans un fichier json et un fichier de configuratio en typescript. Ce même fichier json est importé dans les configurations de « tailwind » qui est un framework CSS permettant la simplicité de création du style.

11 - Dossier de configurations frontend

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

```tsx
<div className="h-60 w-60 bg-colorpalette-active flex items-center justify-center">
```

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

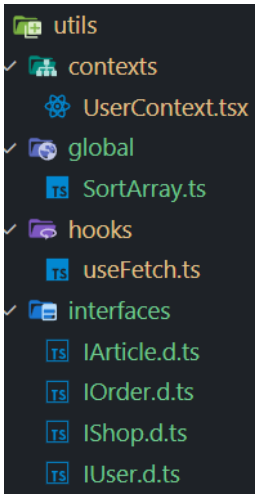


12 - Dossier de ressources frontend

- 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.



13 - Dossier d'outils frontend

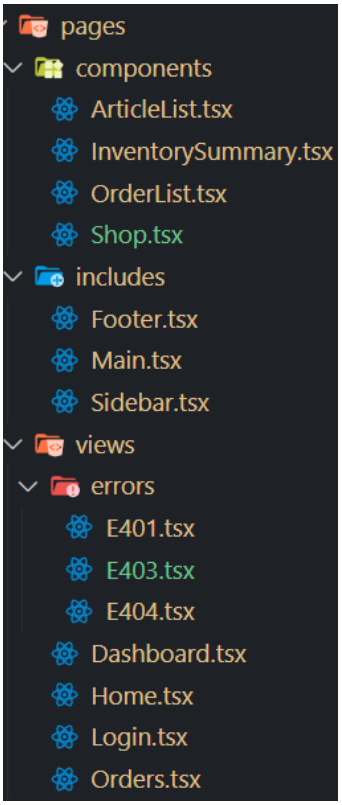
Pour finir, il y a les contextes. Dans ce cas-ci, on y trouve « UserContext.tsx ». Ce fichier, contrairement aux autres est un composant react. Il est en fait le composant chargé de la gestion de la session utilisateur et du stockage de l'objet « user » à chaque changement de page. Il permet de diffuser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

- 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 appels. Ces composants eux-mêmes récupèrent ceux qui sont contenus dans les dossiers « components » et « includes ».
  Ces fichiers afficheront l'UI aux utilisateurs et leur permettront de voyager entre les différentes pages.

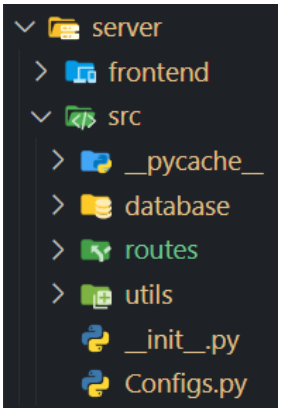
  Les erreurs sont séparées du reste afin de permettre une arborescence propre et compréhensible pour de potentiels futurs développeur qui reprendraient ce projet.

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

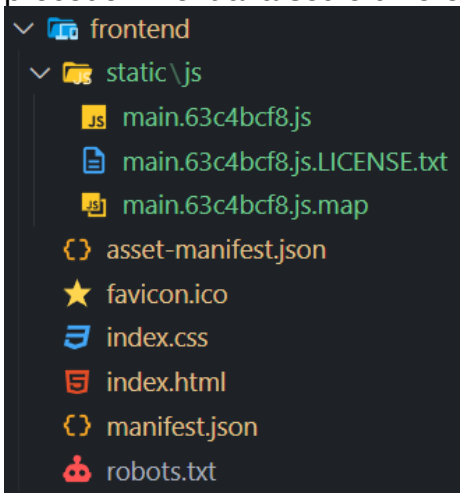


3.1.1.2 Serveur

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.

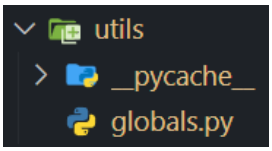


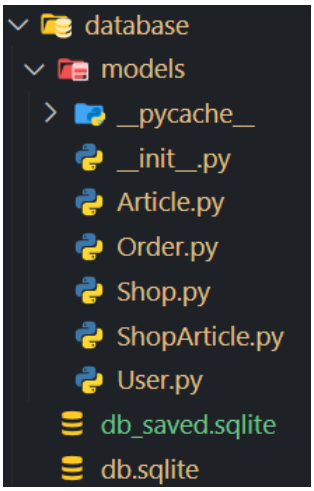
Tout d'abord, le dossier « frontend » contient le client react vu précédemment à la seule différence qu'ici, il est compilé.



Ensuite, Dans le code source du serveur, on y retrouve la base de données, les routes et des outils.

- Outils
  Ce dossier permet de gérer les outils du serveur, celui-ci ne contient qu'un fichier de fonctions globales et n'est pas nécessaire au bon fonctionnement du site.
- Base de données
  La base de données
- Routes





Décrire la réalisation "physique" de votre projet

- *les répertoires où le logiciel est installé*
- *la liste de tous les fichiers et une rapide description de leur contenu (des noms qui parlent !)*
- *les versions des systèmes d'exploitation et des outils logiciels*
- *la description exacte du matériel*
- *le numéro de version de votre produit !*
- *programmation et scripts: librairies externes, dictionnaire des données, reconstruction du logiciel - cible à partir des sources.*

NOTE : Evitez d'inclure les listings des sources, à moins que vous ne désiriez en expliquer une partie vous paraissant importante. Dans ce cas n'incluez que cette partie…

3.2 Description des tests effectués

Pour chaque partie testée de votre projet, il faut décrire:

- *les conditions exactes de chaque test*
- *les preuves de test (papier ou fichier)*

- *tests sans preuve: fournir au moins une description*

3.3 Erreurs restantes

S'il reste encore des erreurs:

- *Description détaillée*
- *Conséquences sur l'utilisation du produit*
- *Actions envisagées ou possibles*

3.4 Liste des documents fournis

Lister les documents fournis au client avec votre produit, en indiquant les numéros de versions

- *le rapport de projet*
- *le manuel d'Installation (en annexe)*
- *le manuel d'Utilisation avec des exemples graphiques (en annexe)*
- *autres…*

4 Conclusions

Développez en tous cas les points suivants:

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

5 Annexes

5.1 Résumé du rapport du TPI / version succincte de la documentation

5.2 Sources – Bibliographie

Liste des livres utilisés (Titre, auteur, date), des sites Internet (URL) consultés, des articles (Revue, date, titre, auteur)… Et de toutes les aides externes (noms)

5.3 Journal de travail

Date	Durée	Activité	Remarques

5.4 Manuel d'Installation

5.5 Manuel d'Utilisation

5.6 Archives du projet

Media, … dans une fourre en plastique