DOSSIER DE PROJET

Création d’un site de E-commerce : Steamfunk

Présenté et soutenu par :

Romain Boudet

En vue de l'obtention du

Titre Professionnel Concepteur Développeur Web

Table des matières

REMERCIEMENTS

Cette dernière année fut intense et riche. J'ai fait le choix d'une reconversion professionnelle et ce projet en est le fruit.

Mais ce projet est également le résultat d'encouragements et de soutiens, sans lesquels tout cela n'aurait peut-être pas vu le jour.

Merci à l'école M2Itech, pour avoir permis le cadre de cette formation en alternance. Je remercie les différents formateurs que j'ai pu avoir au cours de cette année, de nous avoir armé au mieux pour notre futur métier malgré le temps court de cette formation. Et ce dans une bonne ambiance, permettant un apprentissage serein et structuré.

Merci à Jordan, pour son attention et ses remarques constructives tout au long de ce projet

Et enfin, merci à mes proches, amis et parents, qui de par leurs encouragements et soutien tout au long de ce projet, m'ont permis de me sentir prêt pour cette formation et cette reconversion.

Résumé

This e-commerce website project was born from the need of an artisan who creates lamps using recycled music instruments; he wants a website in order to develop his business. The request was also to increase visibility on social networks and, if possible, incorporate a blog into the e-commerce website to document the creation of his works.

The artisan had only a Facebook page to promote his products. The artisan's request was quite complex : to be able to sell his products online, with multiple payment methods, to have a back-office allowing him to check his new orders (preferably through receiving a simple SMS), manage his inventory, add products, and view a summary of his sales with various statistics. Additionally, he wanted to enable customers to buy online and pick up items at the store, and have a potentially scalable website if he wanted to sell other categories of products in the future. And of course, it should be SEO-friendly.

All technical choices were made by myself. I started from scratch on this project and worked during personal time (as the project at work didn't allow me to cover the range of skills required for this diploma). The project is not yet completed at the moment.

To facilitate the viewing of some pictures, QR codes are included in this report. They link to the original documents stored on a cloud that belongs to me.

COMPÉTENCES DU RÉFÉRENTIEL

Les spécifications techniques sont décrites plus en détail dans le chapitre spécifications techniques de ce rapport (cf. page 19), néanmoins un petit rappel afin de faciliter la lecture et la compréhension des compétences du référentiel en lien avec notre projet. Celui-ci a été réalisé via Javascript avec un front React/NextJs, un back sous Node.js et des base de données sous PostgresQL et Redis.

Activité type 1 - Développer la partie front-end d'une application web ou web mobile en intégrant les recommandations de sécurité

CP 1 - Maquetter une application

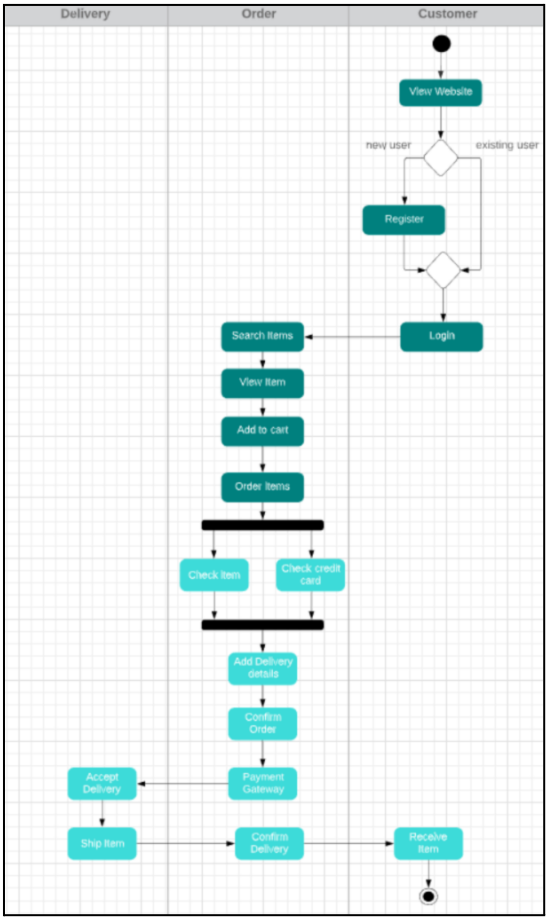


Diagramme d'activité

Après avoir échangé avec l'artisan, et comme pour tout site de e-commerce récent, j'ai opté pour la mise en place d'un site avec un design responsive. Des wireframes ont été conçus via l'outil Figma. Cette étape de la conception est indispensable pour structurer notre interface et vérifier que tous s'articulent correctement.

Un diagramme d'activité a également été maquetté via LucidChart (https://lucid.app/) , pour représenter au mieux ce qu'allaient être les interactions des utilisateurs sur ce site de e-commerce.

CP 3 - Développer des composants d'accès aux données

Dans le cadre de la connexion à la base de données PostgreSQL, le package ‘PG’ a été utilisé tandis que pour l’utilisation de Redis, le package ‘ioredis’ a été utilisé.

```
/**
 * db est un pool de connecteurs de base de données
 * @module - permet le lien avec la base de données postgreSQL
 */

import pg from "pg";
const { Pool } = pg;

// ici, les informations de connection sont récupérées dans l'environnement
// PGHOST pour l'hôte
// PGUSER pour l'utilisateur
// PGPASSWORD pour le mot de passe
// PGDATABASE pour la base de données

const db = new Pool({});

export default db;
```

Connecteur Postgres

Le pool postgres est initialement créé vide et créera de nouveaux clients au besoin. Ici les éléments de configuration nécessaire pour créer un nouveau pool, sont récupérés directement dans les variables d’environnement (si elles portent les bons noms dans le fichier .env) par mesure de sécurité.

L'architecture choisie a été celle de l'Active Record (et non du dataMapper), où le contrôleur interroge les modèles, avec chaque modèle qui dispose de ses propres accès CRUD (Create Read Update Delete) à la BDD, puis le modèle renvoie les infos obtenues au contrôleur.

Dans notre application, les modèles gèrent l'accès aux données manipulées par l'application et s'occupent du stockage (ils constituent l'ensemble des classes qui définissent les objets manipulés par l'application), tandis que nos contrôleurs gèrent l'action demandées, récupèrent les données via le model, et renvoient en format json au front l'information voulue. Aucun ORM n’a été utilisé dans ce projet, (comme Sequelize pour Node.js / Postgres) afin de ne pas être limité avec l'interaction avec la BDD pour des jointures et groupements plus complexes que le classique CRUD. Cela limite également l’import de code tierce à maintenir, et potentiellement source de failles de sécurité .

CP 4 - Développer la partie front-end d'une interface utilisateur web

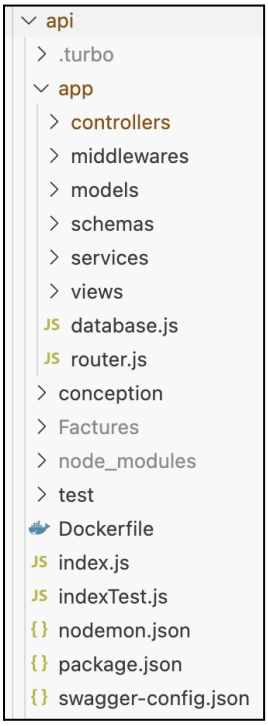
Pour une interface réactive et interactive j'ai utilisé la librairie REACT avec le framework Next.js.

React gère les données de l'application (le state) de façon dynamique. Au moindre changement de ce state (l'utilisateur clique sur un lien, une donnée est mise à jour, un timer se déclenche...), les éléments du DOM virtuel de React sont recalculés et le DOM mis à jour est affiché. Le DOM (Document Object Model) est une représentation de la structure de l'affichage à l'écran. C'est le DOM qui est interprété par le moteur de rendu des navigateurs internet pour être ensuite affiché. React est une librairie flexible et performante, notamment grâce à un DOM virtuel et en ne mettant à jour le rendu qu'en cas de nécessité.

Tandis que Next permet de nombreuses choses : optimisation des images (optimisations d'image automatiques avec des constructions instantanées), une internationalisation plus facile, un rendu hybride statique et serveur, d'une pré-lecture de route, et surtout une prise en charge du rendu côté serveur, la génération statique et le rendu côté client.

D'une manière générale, j'ai fait en sorte que ce front soit accessible depuis un navigateur, responsive, que le code soit documenté, que les informations envoyées à l'API soit au préalable vérifiées et filtrées (schéma de validation).

CP 5 - Développer la partie back-end d'une interface utilisateur web



Le back repose sur un serveur Express derrière un proxy Nginx qui renvoie des données en format JSON au front. La logique back-end de l'application est située dans les contrôleurs. J'ai mis en place 16 contrôleurs pour 38 modèles, certaines parties de la logique ayant été déportées dans un contrôleur dédié (comme l'authentification qui n'est pas dans le "userController".).

Les méthodes dans les contrôleurs s'occupent de récupérer, traiter et renvoyer l'information nécessaire au front. Dans notre application, les données envoyées au front sont variées : détail d'un produit, données sur un utilisateur, information sur un paiement, une commande, autorisation d'authentification, etc.

Pour le reste de l'architecture, j'ai des middlewares, pour l'authentification des différents rôles, et la sécurité (sanitizer le body).

Divers services sont présents pour gérer l'accès à Redis, les paiement en lignes, les envois d'emails, la génération de factures, les envois de SMS, l'écriture des dates, la mise en cache de certaines routes, etc..

Les 23 schémas de validation Joi permettent également de définir très finement le format de données qui est autorisé à rentrer dans les controllers.

Les views, sont des templates pour les emails (édité avec le moteur de template Handlebars). Les factures en pdf sont générées automatiquement lors de chaque paiement et sont stockées sur le serveur. Enfin les 150 routes de l'API sont documentées avec une documentation swagger (https://steamfunk.shop/api-docs).

Activité type 2 - Développer la partie back-end d'une application web ou web en intégrant les recommandations de sécurité

CP 7 - Mettre en place une base de données

Avant la création à proprement parler d'un script DDL (Data Définition Language), j'ai réfléchi à la structure de la Base de Données (BDD).

J'ai élaboré les Users Stories du projet, puis le Model Conceptuel de Données (MCD) via JMerise puis j'ai effectué un passage du MCD au Modèle Logique de Données (MLD). Le passage du MCD au MLD s'est effectué en suivant certaines règles :

- Chaque entité du MCD devient une table, son identifiant devient la clé primaire de la relation. Les autres propriétés deviennent les attributs de la relation.
- Une association avec la cardinalité 1:N (avec les cardinalités maximales positionnées à " 1 " d'un côté de l'association et à "N" de l'autre coté) va se traduire par la création d'une clé étrangère dans la relation correspondant à l'entité coté 1 (c'est cette table qui porte la relation). Cette clé étrangère référence la clé primaire de la relation correspondant à l'autre entité.
- Une association avec la cardinalité N:N, va se traduire par la création d'une table spéciale dont les attributs seront des clés étrangères référençant les relations correspondant aux entités liées par l'association.
- Dans le cas d'une cardinalité 1:1, la clé étrangère pourra être dans l'une ou l'autre des deux tables.

Un dictionnaire de données a également été conçu pour lister toutes les colonnes de nos tables de manière visuelle.

Associé à Node.js il me fallait un SGBDR efficace. J'ai fait le choix de PostgreSQL (libre sous licence BSD). Parce qu'il est reconnu pour son comportement stable et ses possibilités de programmation étendues, et aussi parce qu'il s'agissait du SGBDR avec lequel j'ai le plus de pratique.

Conformément à un bon usage de PostgreSQL, j'ai respecté les contraintes d'intégrité avec des clés (en pratique une clé "id" ayant une valeur numérique auto-incrémentée dans

notre cas avec le type IDENTITY) et des clés étrangères, afin d'assurer la cohérence du modèle de données.

Pour respecter l'unicité de la donnée, de nombreuses clés ont été caractérisées avec la valeur UNIQUE afin de nous préserver de tout doublon non souhaité (la présence de la valeur UNIQUE sur la clé "email" nous assure qu'un seul utilisateur avec cet email sera présent en base). Pour maintenir une qualité des données en base, un typage fin des colonnes a permis d'apporter un premier niveau de contrôle (NOT NULL si on ne veut pas de valeur null, CHECK pour ajouter des contrôles sur le contenu des données,etc.). Avec PostgreSQL, tous les ordres de type DDL sur des objets d'une base étant transactionnels, je n'ai pas oublié d'encadrer mes script DDL par les ordres BEGIN et COMMIT, m'assurant de ne rien écrire en BDD en cas de problème à la lecture du script.

Enfin, d'un point de vue sécurité en production, l'image docker postgres sera modifiée pour donner un mot de passe à un rôle (ALTER ROLE monrôle WITH PASSWORD "…"). Tandis que d'un point de vue pratique, en back, j'ai travaillé avec pgAdmin4. pgAdmin4 est un outil de gestion pour PostgreSQL, Il peut être exécuté en tant qu'application Web ou de bureau. Comme le reste de la stack, il a été intégré dans le docker-compose pour pouvoir être utilisé et connecté automatiquement à la BDD du projet, lors de chaque lancement de la stack.

A l'avenir, pour faciliter les migration de la base et le versioning de la base, sqitch (https://sqitch.org/about/) sera mis en place sur le projet.

Activité type 3 - Concevoir et développer une application multicouche répartie en intégrant les recommandations de sécurité

CP 10 - Concevoir une application

Cette application a été réalisée en relation directe avec le client. Après la traduction des besoins du client en diagramme UML. A partir du diagrammes UML, de la conception du MCD et du MCD, l'application a été conçue en respectant les besoins de sécurité de l'application. Un backlog a été réalisé sous trello pour organiser les tâches.
Cette application a une approche de conception centrée sur la base de données qui se base principalement sur le modèle conceptuel de données (MCD) plutôt que sur le diagramme de classe. Les aspects fonctionnels et comportementaux ont également été pris en compte dans le processus de conception, mais la structure de la base de données à joué un rôle prépondérant.

CP 11 - Développer des composants métier

De nombreux composants métier ont été développés, spécifiquement pour un site e-commerce:

- des composants pour les paiements en ligne, via carte bancaire ou virement SEPA,
- des formulaires pour insérer les données de l'utilisateur avant envoi des produits commandés,
- des composants pour générer automatiquement des factures après paiement.

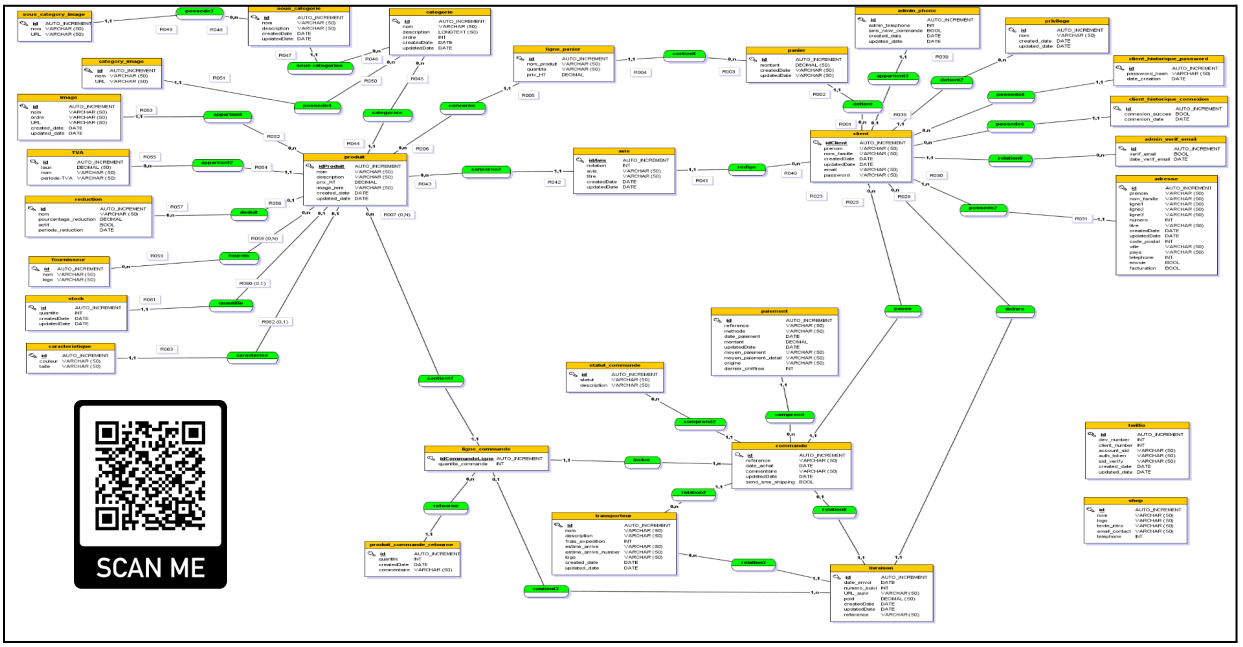
CP 12 - Construire une application en couches

Les deux parties de notre application sont réparties en couches. Le front est constitué en couches avec des composants, des hook, du style et un dossier app, bien séparé, tandis que dans le back, l'architecture de type MVC répartit le code dans des controllers, des modèles, des services et des schémas.

CAHIER DES CHARGES

La conceptualisation de l'application

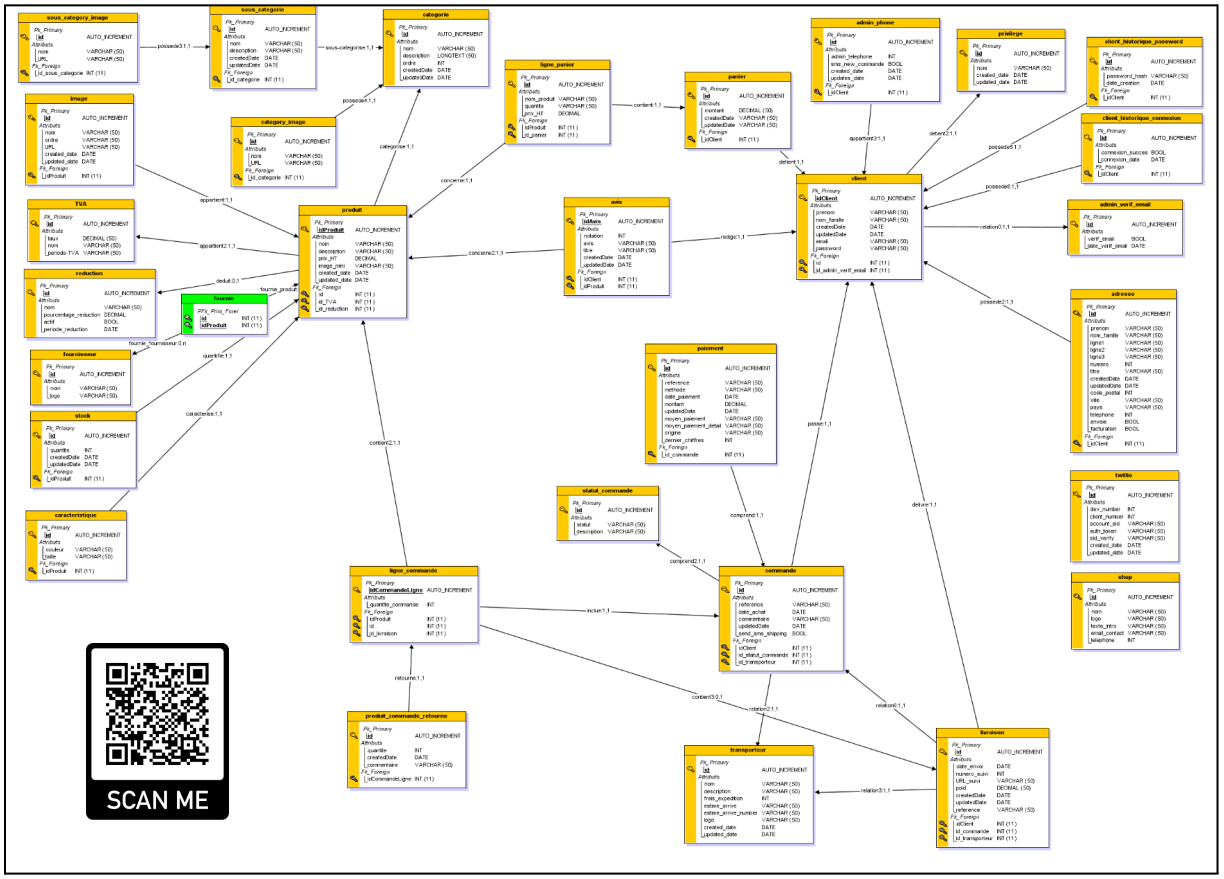
Afin de définir la structure de ma BDD, après un travail de réflexion j'ai créé un MCD via jMerise pour le Minimal Viable Product, (les règles de gestion sont en annexes) :



Modèle Conceptuel de Données du projet

(disponible en Annexe en pleine page ou en ligne en scannant le QR Code)

Et son MLD associé, en suivant les 4 règles précédemment éditées (cf. page 9 "CP 7 – Mettre en place une base de données") :



Modèle Logique de Données du projet

(disponible en Annexe en pleine page ou en ligne en scannant le QR Code)

Le Workflow de l'application

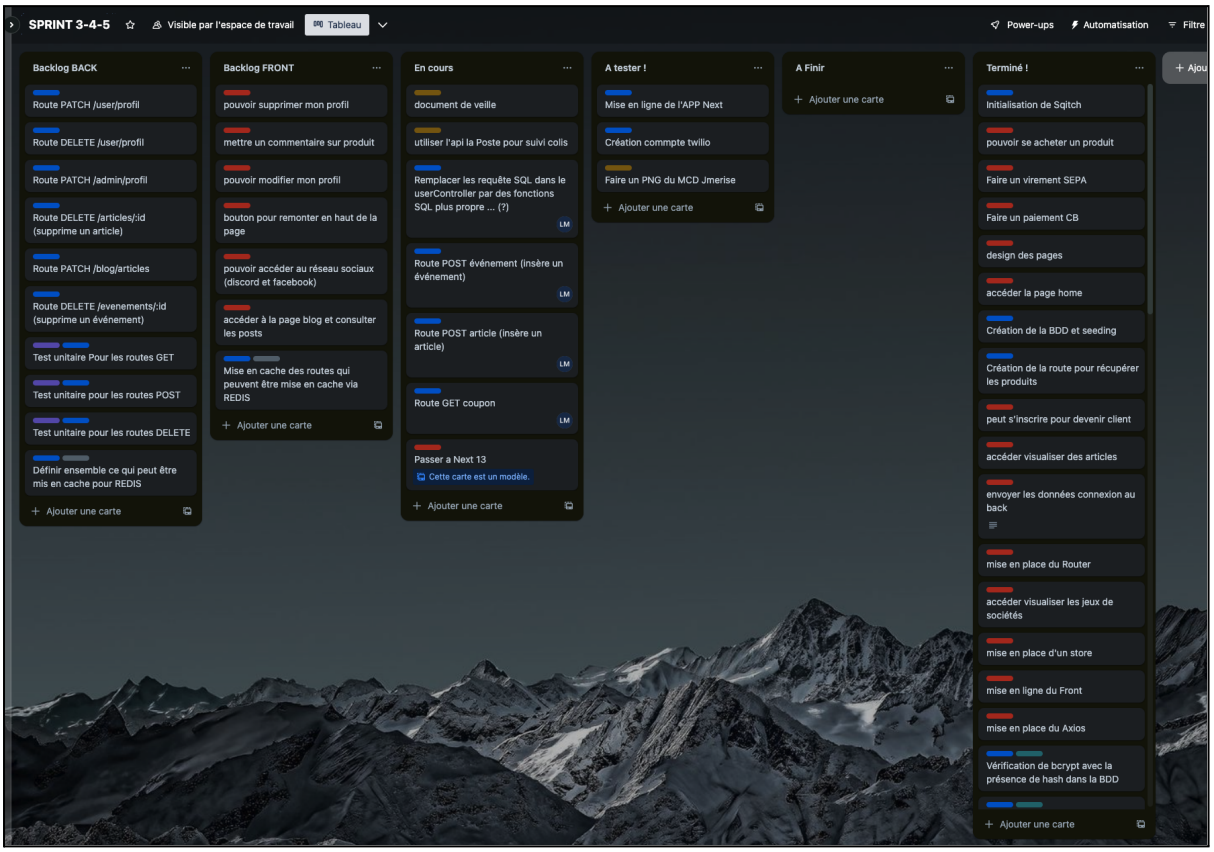
Pour la réalisation du site, j'ai utilisé la méthode agile SCRUM basée sur la communication, et un développement évolutif. Dans le cadre de cette méthode, j'ai réalisé un backlog priorisé (via Trello) où j'ai pu organiser mes idées, des sprints, (soit des intervalles de temps pendant lesquels j'ai complété un certain nombre de tâches du backlog). Au fur et à mesure que le statut des tâches fixées dans le backlog évoluaient (en cours, à tester, fait), j'ai déplacé les tâches dans le trello. Entre chaque sprint des réunions avec le client ont permis d'ajuster les besoins du client.

Ce projet a été réalisé sur 7 sprints : (ici, chaque journée de travail correspond bien souvent à quelques heures en fin de journée, après une journée en entreprise.)

- Sprint 0 : Initialisation du projet, réflexion technique, choix des technos (10 jours).

- ❖ Sprint 1: Réalisations des maquettes sous Figma (10 jours).
- ❖ Sprint 2: Élaboration des documents liés à la BDD avant la création de script de création de la BDD (20 jours).
- ❖ Sprint 3, 4 et 5 : Mise en place des principales fonctionnalités, temps dédié au code (100 jours).
- ❖ Sprint 6 et 7 : Finition des dernières fonctionnalités en cours, test, refactoring, correction des bugs, test finaux, déploiement sur le serveur d'hébergement (20 jours).

Le projet n'étant pas terminé, je suis toujours dans une phase de développement de feature, les sprints 6 et 7 n'étant pas commencés.



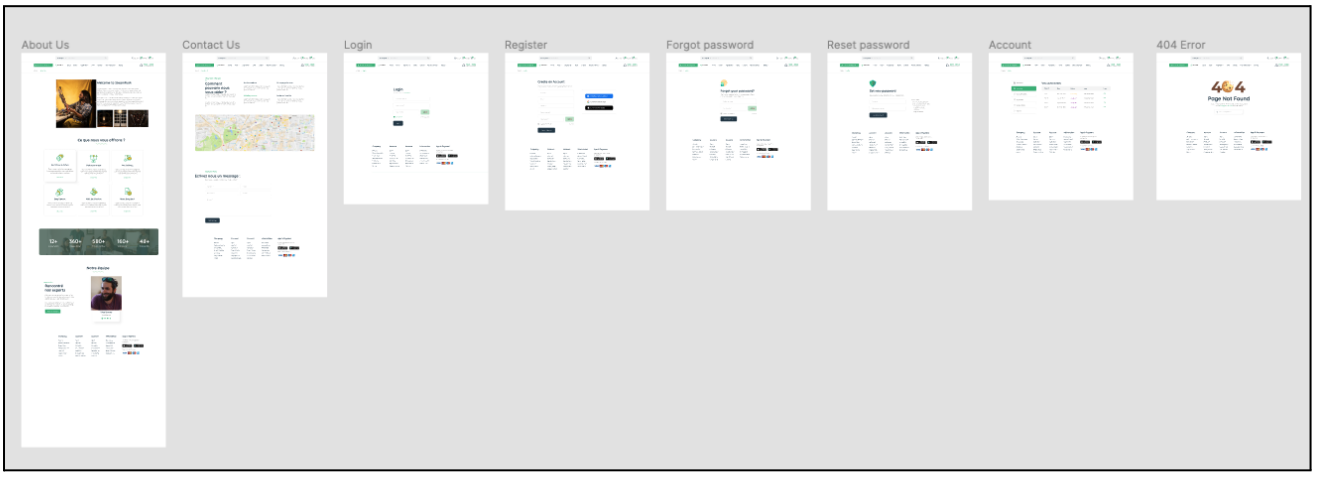
Trello et le backlog du projet

La mise en dimension de l’application

Les Wireframes de l’application

Les wireframes ont été réalisés via l’outil Figma (https://figma.com). 10 Wireframes ont été réalisés pour la partie desktop. Chaque maquette permet une représentation précise du futur site.

Le détail de ces wireframes est disponible en annexe. En voici un aperçu :



Wireframe du projet

La création de la charte graphique

L’artisan ne possède pas à proprement parler de charte graphique que j'aurais pu suivre, mais juste un logo. Je me suis rapproché des thématiques et colori assez clairs et simples de nombreux sites de e-commerce, et je m’en suis inspiré pour créer le graphisme et les maquettes du site.

Utilisateurs et user stories

Le public visé et les rôles

Ce site est ouvert à tous les publics et est configuré pour 4 types de rôles : les clients, les clients non connectés, les administrateurs et le développeur.

Toutes les fonctionnalités reliées ne sont pas encore mises en place dans le Front.

- **Les utilisateurs non connectés** : ils ont accès aux produits, aux articles du blog et aux différentes catégories de produits. Mais ils ne peuvent avoir accès à leur page de profil, leurs comptes, leur historique de commandes.
- **Les utilisateurs connectés (inscrit)** : ils peuvent faire tout ce que peut faire un utilisateur non connecté mais également avoir accès/modifier leur profil et leur compte, leurs anciennes commandes, consulter et télécharger leur factures en pdf, réaliser des achats avec différents moyens de paiement.
- **Les administrateurs** : ils peuvent supprimer et modifier tous ce qui est relatif aux produits (des catégories, des images, des descriptions, la TVA, le stock, etc) valider des commandes, effectuer des remboursement, gérer des coupons, changer l'état d'avancement des commandes, consulter la listes des clients, avoir accès au statistiques de ventes, effectuer des réductions sur des produits pendant des temps définis, configurer leur compte administrateur pour recevoir ou non des sms à chaque nouvelle commande ou pouvoir envoyer des ordres au server par sms.
- **Le développeur** : il pourra faire tout ce que font les rôles précédents mais aussi supprimer des comptes, donner des droits supplémentaires à certains membres, vérifier la balance Stripe et demander des virements de Stripe au compte souhaité .

Les User Stories

En vue de définir les différentes fonctionnalités du projet qu'il allait falloir développer, J'ai établi des User Stories.

Users Stories sf

En tant que	Je veux
client non connecté	Accepter ou ne pas accepter une politique de cookies
client non connecté	Accéder aux différentes catégorie de produit vendue sur la page d'acceuil
client non connecté	Accéder aux détails d'un produit : (*)
client non connecté	* Voir la description des produits vendus
client non connecté	* Voir les photos des produits vendus
client non connecté	* Pouvoir zoomer sur les photos présentées
client non connecté	* Pouvoir faire le tour en 3D des produit vendus
client non connecté	* Voir le prix des produits vendus
client non connecté	* Voir la référence du produit vendu

SCAN ME

Extrait des users Stories du projet

(Version complète en Annexe ou en ligne en scannant le QR Code)

Actuellement, de nombreuses fonctionnalités sont présentes en back et ne sont pas encore implémentées en front.

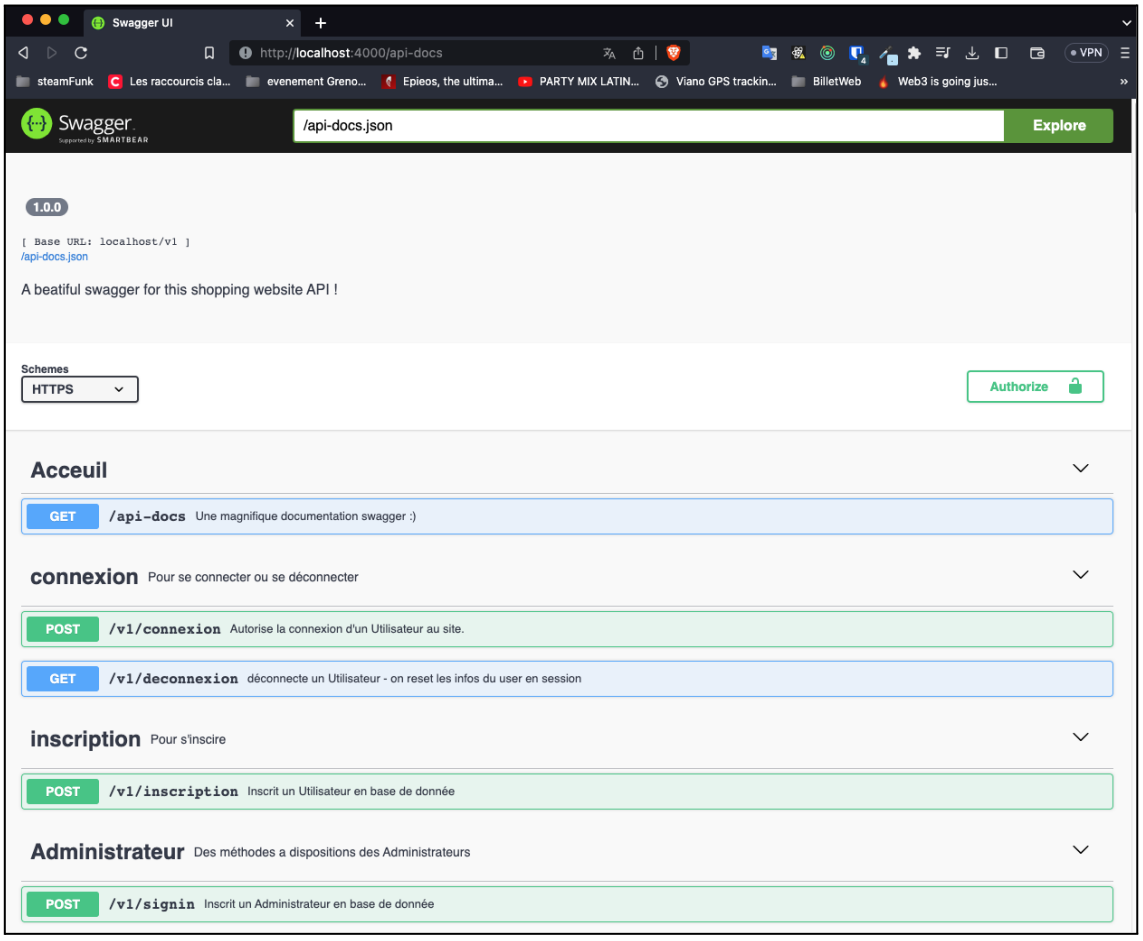
Documentation

J'ai mis en place une documentation Swagger sur le router. Pour rappel, Swagger UI est un logiciel permettant de générer une documentation en utilisant les spécifications d'OpenAPI (https://swagger.io/specification/).

Pour la documentation du router, et la description précise des routes présentes et de leurs caractéristiques majeures (comme le format de données qu'elles acceptent, leurs réponses, etc.), une documentation auto-formatée par Swagger est donc disponible en ligne quand l'API est en production. Elle est disponible en ligne : https://steamfunk.shop/.

Cette implémentation, mise à jour au fil du développement du router, permet ainsi d'avoir une idée détaillée des routes présentes et exploitables pour le front à tout moment.

L'API possède environ 150 routes opérationnelles. La documentation automatiquement générée à ce format :



Extrait de la documentation auto-générée par Swagger

(Le reste de la documentation swagger est présent en Annexe)

SPÉCIFICATIONS TECHNIQUES

Le Versionning

Pour la programmation, j'ai utilisé Github pour sauvegarder mon code et centraliser les données. Avec comme organisation un monorepo et des branches : une branche Main toujours fonctionnelle représentant le code prêt à être mis en production, une branche dev-test et une dev-back, également fonctionnelle, dédiée au front ou au back, et enfin des branches par features.

A l'avenir, pour l'évolution de la base, j'utiliserai Sqitch que j'ai découvert tardivement. Sqitch est un système de migrations permettant de versionner une base de données. Une migration Sqitch est l'équivalent d'un commit Git : elle recense les instructions SQL permettant de passer d'un état donné à l'état suivant (script SQL dans le dossier "deploy") ainsi que les instructions SQL pour revenir à l'état initial (script SQL dans le dossier "revert"). Il est aussi possible de fournir un script de test pour confirmer la validité du déploiement (dans le dossier "verify"). Une migration c'est donc 3 scripts SQL supplémentaires que l'on remplit après avoir fait la commande :

sqitch add <nom-de-la-migration> -n 'description'

Après seulement il nous est possible d'écrire nos scripts de déploiement que l'on pourra lancer avec la commande : sqitch deploy db:pg:<nom_de_la_DB> et nos réversions si nécessaire seront toutes autant aisées avec la commande sqitch revert db:pg:<nom_de_la_DB>.

L'éfficacité de Sqitch réside dans sa simplicité. Un projet fonctionnant avec Sqitch possède seulement 2 petits fichiers indispensables :

- Un fichier "sqitch.conf" qui détaille la configuration du projet (SGBD utilisé et nom de la base de données, principalement)
- Un fichier "sqitch.plan" qui recense l'ordre des migrations du projet

j'ai édité un pense-bête Github, afin de centraliser en un seul document les principales commandes github nécessaires dans ce projet.

Ce récapitulatif est disponible dans les annexes.

Monorepo

A la racine du monorepo, l'application est conteneurisée via Docker et un fichier 'docker compose' orchestre les conteneurs suivants : postgres, redis, pgadmin4 (interface graphique pour la gestion de postgres), redisInsight (interface graphique pour la gestion de Redis), mailhog (Pour recevoir, en environnement de développement, les emails sur une interface graphique en évitant de spammer les emails perso), Nginx (comme reverse proxy). Pour la mise en production, L'API et le front seront aussi conteneurisés pour un déploiement plus facile sur n'importe quel serveur.

Pour rappel, Docker est une plateforme de virtualisation légère qui permet d'emballer des applications et leurs dépendances dans des conteneurs isolés, fournissant ainsi une portabilité et une gestion simplifiée. Les conteneurs Docker offrent une solution rapide et cohérente pour exécuter des applications sur différents environnements sans se soucier des dépendances et des configurations système.

Le bundler utilisé est Turbo, (https://turbo.build/repo, conçu par Vercel, comme Next.js), un bundler incrémental, optimisé pour JavScript et TypeScript écrit en Rust, avec un système de mise en cache lors de chaque build qui permet d'économiser du temps de build.

Les technologies du Back

Pour la partie back j'ai utilisé l'environnement d'exécution Node.js créé par Ryan Dahl en 2009.

Sur celui-ci, plusieurs autres technologies sont venues se greffer :

- PostgreSQL : comme SGBDR, pour la base de données.
- REDIS : base de données clé valeur, Nosql, très haute performance, utilisé pour la mise en cache de certaines routes et la mise en cache des sessions.
- Ngrok pour exposer mon API qui tourne en local, en ligne, pour permettre l'accès à mon API au webhook Stripe et Twilio
- Express : Mise en place du serveur HTTPS
    - Joi : pour la validation de données d'une query ou d'un body.
    - Bcrypt : pour chiffrer et vérifier les mots de passe.
    - Express-sanitizer et Email-validator pour se protéger au mieux des attaques XSS.
    - Swagger : pour la documentation du router.
    - JWT : pour générer des tokens utilisés dans la création de liens temporaires.
    - NodeMailer : pour les envois d'emails.
    - Stripe pour les Paiements en ligne

- Twilio pour l'envoi de sms de la part du server ou l'envoi d'ordres au server par commande SMS
- dayjs pour la gestion des date et heures
- I18n pour l'internationalisation
- mocha et Chai pour les tests unitaires
- EsLint pour le linter

Les technologies du Front

Pour le front j'ai utilisé la librairie REACT avec framework NextJs avec :

- React-redux : fait le lien entre le store et React, pour la gestion centralisée des états et des actions des composants react,
- Axios : pour faire les appels avec l'API,
- Redux : pour créer un store des variables dynamiques à modifier,
- Tailwind en framework CSS,
- react-swipeable pour détecter les mouvements
- react-hot-toast pour les notifications

La sécurité de l'application

J'ai essayé d'être vigilant tout au long du projet à sécuriser l'API au maximum. En lien avec le top 10 des failles reconnues par le Open Web Application Security, j'ai essayé d'être vigilant contre 4 attaques fréquentes, à savoir : les failles XSS, CRSF, injection SQL et Brute-Force. Dans cette partie à venir, je vous expliquerai rapidement ce en quoi consiste ces attaques, afin de mieux percevoir les stratégies mises en place pour s'en protéger.

1. Les attaques Cross Site Scripting (XSS), consistent à injecter du contenu dans une page, ce qui provoquera des actions de la part des navigateurs qui la liront. Pour s'en prémunir, j'ai suivi la règle NTUI, Never Trust User Input. C'est-à-dire la sécurisation de tout ce qui rentre dans l'API, avec un schéma Joi (toute donnée doit respecter un certain format défini par le schéma), le middleware Express-Sanitizer pour échapper tous les caractères spéciaux qui passent par le body et les query mais aussi Email-validator pour vérifier les formats des emails, et enfin j'ai mis en place une particularité d'express qui permet de mettre des mini regex sur nos routes avec params.
2. Les attaques Cross Site Request Forgery (CRSF), visent à faire faire une requête à l'insu d'un utilisateur, sur un site ou il est déjà connecté. L'attaquant devra néanmoins connaître le endpoint et la query à modifier pour mener son attaque. Il est important

de se rappeler qu’une propriété des navigateurs web est qu'ils incluent automatiquement et de manière invisible, tous les cookies utilisés par un domaine donné, dans toute demande web envoyée à ce domaine. Les attaquants exploitent ainsi cette propriété par des attaques CSRF puisque toute demande web effectuée par un navigateur inclura automatiquement tous les cookies (y compris les cookies de session et autres) créés lorsqu'une victime se connecte à un site Web. Pour s'en prémunir on envoie un token dans le local storage et on envoie ce même token dans un cookie (signé) supplémentaire. Lors d'une requête sur l'API, ces deux données devront être présentes, le token du local storage envoyé via le header et l'autre via le cookie. Si les deux tokens sont bien envoyés par ces deux biais distincts et qu’ils sont identiques en back, on en déduit l'absence d'attaque CSRF. Dans cette configuration, nous partons du principe que le local storage est peu sensible aux attaques CSRF, mais dans le cas où une attaque XSS aurait abouti malgré nos défenses, alors celle-ci pourrait dérober le token du local storage et faire aboutir une attaque CSRF...

Afin de contrer efficacement ces deux attaques, un paramétrage précis des options de configuration pour les CORS et les cookies est également indispensable :

Configuration pour les cookies :

- Les cookies sont envoyés uniquement en HTTPS avec l’option "httpOnly: true" dans la configuration d'express-session.
- Les cookies sont envoyés uniquement sur HTTPS et pas au javascript client avec l'option "secure: true".
- L'envoi de cookie est interdit dans le cas d'un accès au site via un lien externe, avec l'option "Samesite: "strict"". En n'oubliant pas que les clients utilisant Firefox ne pourront pas bénéficier de cette option.
- L'utilisation de cookie signé, pour le cookie porteur du csrf token avec l'option "signed: true", (en oubliant pas de récupérer le cookie dans le MW d'authentification avec la méthode "req.signedCookies" dans ce cas..)

Configuration pour les Cross Origine Ressource Sharing (dans notre cas de figure avec l'utilisation du paquet NPM cors) :

- CORS : Cross Origine Ressource Sharing, le fait que d'autres sites que notre front ne puissent pas interroger notre API, avec la valeur "origin: "mondomain.prod"", a un nom de domaine précis et unique correspondant à notre front. En local la valeur était "https://localhost:3000" sans oublier de passer également à true l'option "credentials", permettant d'envoyer des cookies et des en-têtes d'autorisation et d'autoriser notre header à transporter notre xrsfToken avec "allowedHeaders : 'x-xxrsf-token'".

3. L'attaque par injection SQL est une attaque visant à injecter du langage SQL en vue d'essayer de communiquer directement avec la BDD. Pour s'en prémunir j’ai mis place

des requêtes paramétrées dans les modèles : "...WHERE id=$1;', [id]", ou la valeur de la variable n'est pas disposée directement dans la query.

4. L'attaque par Brute-Force vise à tester un très grand nombre de mots de passe pour tenter de trouver le bon. Pour m'en prémunir j'ai inclus un module permettant de limiter, pour une adresse ip donnée, le nombre de tentatives de connexion sur la route /connexion. Néanmoins, cette méthode serait contournable par un vpn qui pourrait changer d'adresse IP à volonté. Dans notre cas, le module est configuré pour permettre uniquement 100 connexions en 10 heures par adresse ip sur la route /connexion. Un mot de passe robuste a également été demandé à nos utilisateurs (8 caractères minimum, une lettre minuscule, une lettre majuscule, un nombre, et un caractère spécial parmi !*#$%^&). Il serait également préférable de ne lancer la règle des 100 connexions en 10h qu'après le premier essai infructueux uniquement.

Et d'une manière générale j'ai aussi opté pour certaines bonnes pratiques permettant d'accroître la sécurité générale de l'application :

- La mise en place du protocole HTTPS, avec un certificat auto-signé quand je travaille en développement via mkcert (pratique car accepté comme s'il était émis par une CA via les navigateurs) et signé avec une vrai CA (Certificate Authority) pour la mise en production, via Let's encrypt. Le protocole de sécurisation des données, entre la couche transport TCP/UDP et la couche application HTTP est le TLS 1.3 (dernière version du Transport Layer Security, remplaçant le Secure Socket Layer). L'API est également compatible HTTPS2 et Speedy (protocole de Google pour augmenter les capacités du http, notamment en multiplexant le transfert, intégré dans HTTP2). Seule l'API est en HTTP2, Le front est resté en HTTP.
- Une configuration fine de mes en-têtes. Comme le fait de supprimer l'en-tête X-powered-by (pour que personne ne sache que j' utilise Express, et ainsi cibler les attaques), définir l'en-tête Content-Sécurity-Policy pour la protection contre certaines attaques XSS et CRSF, imposer des connexions au serveur sécurisé via SSL/TLS via l'en-tête Strict-Transport-Security, etc. Pour en savoir plus, cf. page 38 "Jeu d'essai".
- Le rangement des variables sensibles et des "secrets" dans un fichier d'environnement. (.env).
- Suivi et veille en termes de sécurité en auditant nos paquets npm et la configuration de nos headers. Pour en savoir plus, cf. page 44 "Veille sur les vulnérabilités de sécurité".
- La mise à jour régulière des paquets qui le nécessitent.
- Le stockage des mots de passe sous forme de hash en BDD.
- L'utilisation d'algorithmes de chiffrement sûrs et reconnus, HS512 dans notre application via les JWT et SHA 256 pour l'algorithme de signature du certificat (DES, 3DES et fonction de hashage MD5, SHA1 sont à bannir, fonctionnalités obsolètes encore présentes dans le TLS 1.2).
- Une clé secrète complexe, avec plus de 100 caractères.

- Un rôle utilisateur Postgres sur le serveur de production avec des droits réduits en accès sur la BDD (dans le conteneur Docker).
- Un backup régulier de la BDD via une commande pg_dump associée à une tâche CRON qui envoie, via un email (en utilisant les paquets linux postfix mailutils) la BDD dans un fichier compressé et chiffré. Le script associé à la tâche cron écrit est disponible en Annexe.

Pour ce qui est de la sécurité propre à la technologie Docker :

- J'ai évité de faire tourner les conteneurs Docker avec l'autorisation root,
- On peut utiliser des registres de confiance comme Docker Trusted Registry pour bénéficier d'un registre privé sécurisé pour stocker et gérer les images Docker (le DTR étant payant, je n'ai pas mis en pratique cette règle) .
- Dans la mesure du possible, utiliser des images Docker les plus légères possible (Alpine), ne contenant que le nécessaire, évitant des failles de sécurité potentielles dans du code tierce.
- Définir des quotas de ressources pour chaque conteneur, afin d'empêcher un seul conteneur de consommer trop de ressources système. Cela permet également d'améliorer la sécurité en bloquant les attaques qui cherchent à perturber des services en accaparant un grand nombre de ressources.
- Il est recommandé d'utiliser des outils de surveillance et de scans des conteneurs.
- On peut également utiliser des plateformes de gestion de conteneurs qui offrent un environnement pour orchestrer et automatiser les opérations liées aux conteneurs comme Kubernetes ou Terraformer pour l'infrastructure (souvent utilisé en amont de Kubernetes pour provisionner l'infrastructure nécessaire à l'exécution de clusters Kubernetes).

Autres services utilisés

Pour la mise en production, le projet sera déployé sur un VPS dans un premier temps pour être testé puis sur une instance EC2 d'Amazon Web Service pour bénéficier de la scalabilité.

Réalisations personnelles détaillées

Dans les extraits de code ci-dessous, la JS doc, les commentaires et certains console.log ont été retirés afin de rendre ces extraits plus compacts en vue de l'impression. Par ailleurs, l'indentation a été optimisée pour réduire le nombre de lignes ne contenant qu'une parenthèse et/ou accolade. Ils restent à votre entière disposition en format original dans VSCode le jour de l'examen.

Mise en place des accès CRUD et du mécanisme d'inscription

Pour la mise en place de la BDD, un script DDL a été créé, avec un fichier de seeding pour injecter des données de test dans la base.

Une fois la base de données en place, et l'architecture express de l'API construite, je me suis appliqué à mettre en place la majorité des routes présentes et à gérer le formulaire d'inscription.

Mise en place du userController et du modèle associé :

Dans un premier temps j'ai mis en place le clientController qui possède les méthodes dédiées aux utilisateurs pour les opérations classiques CRUD, de manière asynchrone avec une méthode asynchrone contenant un ou plusieurs await, (la fonction qu'on await va attendre toutes les fonctions asynschrones qu'elle contient) permettant de traiter en parallèle d'autres taches. Et on l'encadre avec un try – catch pour obtenir l'erreur de la fonction asynchrone. Une fois l'information récupérée du modèle, si pas d'erreur, l'information est envoyée en format json avec un statut 200 indiquant que tout s'est bien passé.

```
getAll: async (req, res) => {
        try {

            const clients = await Client.findAll();

            res.status(200).json(clients);
        } catch (error) {
            console.trace('Erreur dans la méthode getAllUser du clientController :',
                error);
                res.status(500).end();
            }
    },
```

Méthode getAll du clientController

Tandis que dans le modèle, qui va interroger la BDD, on déclare la classe puis les propriétés de cette classe, allant de l'id à idPrivilége pour nos clients. Nos setters (accesseurs) permettent de passer du snake-case du langage sql à l'écriture camel-case de javascript.

La méthode constructor permet de passer des arguments à la création et ainsi d'appeler new Client avec des arguments (la méthode constructor est toujours présente mais l'appeler nous permet de lui passer des arguments), et ainsi créer une nouvelle instance à partir de notre classe Client. On utilise une boucle for in (pour des objets énumérables) afin de boucler sur chaque propriété, et pour stocker les valeurs reçues en arguments dans l'objet final, on utilise le mot clé this pour y accéder. Ce qui nous permet de désigner l'objet Client en cours de "fabrication" (dans ce contexte).

```
class Client {

  id;
  prenom;
  nomFamille;
  email;
  password;
  createdDate;
  updatedDate;
  idPrivilege;

  set created_date(val) {
    this.createdDate = val;
  }

  set updated_date(val) {
    this.updatedDate = val;
  }

  set nom_famille(val) {
    this.nomFamille = val;
  }

  set id_privilege(val) {
    this.idPrivilege = val;
  }


  /**
   * @constructor
   */
  constructor(data = {}) {
    for (const prop in data) {
      this[prop] = data[prop];
    }
  }
```

Et on déclare nos méthodes d'instance(image suivante) sans utiliser de fonction fléchée pour des notions de portée, puisque this dans la méthode d'instance représente l'instance courante, alors que dans une méthode statique, elle représente la classe elle-même. Le mot static devant async indique une méthode statique, indiquant une méthode non pas rattachée à l'objet fabriqué mais à la classe Client elle-même, et elle sera partagée avec toutes ces instances. Avec await db.query j'attends le retour de ma requête sql lancé à ma BDD via le connecteur db qui va faire le lien avec ma BDD. On notera au passage la présence de requêtes paramétrées pour se prémunir de toutes attaques par injection sql. Query est une méthode de db, elle-même une nouvelle instance de Pool extrait du module "pg", dans le fichier database. S'il n'y a pas de ligne de résultats, donc si l'index 0 n'existe pas (s'il y en a au moins un qui existe, c'est celui-là), on l'envoie dans une nouvelle instance de la classe Error, avec son message en argument. Si tout va bien, Map va pour chaque élément du tableau, renvoyer un nouvel objet. Et pour chaque objet (utilisateur en BDD), on va créer une instance de User qui contient les infos de la BDD.

Constructor et accesseur du modèle client

```
static async findAll() {
    const {
      rows
    } = await db.query('SELECT * FROM sf.client ORDER BY client.id ASC');

    if (!rows[0]) {
      throw new Error("Aucun client dans la BDD");
    }
    consol.model(
      `les informations des ${rows.length} clients ont été demandé !`
    );

    return rows.map((client) => new Client(client));
  }
```

Méthode static findAll dans le user modèle

On étudie le sens de l'information pour déterminer si on utilise une méthode d'instance ou une méthode statique : si on part de JS pour mettre à jour la BDD, alors on utilise une méthode d'instance. Si on part de la BDD pour mettre à jour des infos dans JS, ce sera une méthode statique. C'est pourquoi, seulement la méthode save, update, updatePwd et delete ne sont pas statiques, ce sont les seules méthodes d'instance qui mettent à jour la BDD. Toutes les autres sont en statique.

```
async save() {
    const {
      rows,
    } = await db.query(
      `INSERT INTO sf.client (prenom, nom_famille, email, password) VALUES ($1,$2,$3,$4) RETURNING*;`,
      [this.prenom, this.nomFamille, this.email, this.password]
    );

    this.id = rows[0].id;
    this.createdDate = rows[0].created_date;
    consol.model(
      `le client id ${this.id} avec comme nom ${this.prenom} ${this.nomFamille} a été inséré à la date du ${this.createdDate} !`
    );
    return new Client(rows[0]);

  }
```

méthode d'instance : save() du model client

Ce qui caractérise nos différentes opérations du CRUD, ce sont les différentes requêtes sql dans nos méthodes qui vont interagir de manière différente avec la BDD :

- INSERT INTO… pour insérer en BDD => Create
- SELECT * FROM… pour aller chercher une info en BDD => Read
- UPDATE… pour mettre à jour la BDD => Update
- DELETE FROM… pour supprimer un enregistrement de la BDD => Delete

La logique entre les autres controllers et leurs modèles est identique à celle décrite précédemment.

Maintenant que sont décrit les principales opérations de base que j'ai mises en place pour que le ClientController interagisse avec la BDD, voyons plus en détail la méthode handleSignupForm, du clientController, qui va prendre en charge l'inscription en BDD d'un client qui souhaiterait s'inscrire.

```
signIn: async (req, res) => {
        try {
            const {
                prenom,
                nomFamille,
                email,
                password,
                passwordConfirm
            } = req.body;

            if (!validator.isEmail(email)) {
                return res.status(403).json({message:'Le format de l\'email est incorrect'});
            }
            if (!validator.isStrongPassword(password)) {
                return res.status(403).json({message:'Le format du mot de passe est incorrect : Il doit contenir au minimum 8
            }
            if (password !== passwordConfirm) {
                return res.status(403).json({message:
                    'La confirmation du mot de passe est incorrecte'
            });
            }
            const userInDb = await Client.findByEmail(email);
            if (userInDb !== null) {
                return res.status(403).json({message:'Cet email n\'est pas disponible'});
            }
            const hashedPwd = await bcrypt.hash(password, 10)
```

méthode signin du clientController

Cette méthode reçoit du schéma de validation Joi (middleware présent en amont du controller, pour s'assurer d'un format prédéfini, strict, des données reçues), 5 informations dans le body. Avec la package validator on s'assure que le format de l'email est correct, et que le mot de passe est robuste, à noter que cette vérification devrait dans tous les cas passer, puisque Joi l'a vérifié avant, dans le schéma de validation. A ce stade, nous sommes certains que les données envoyées sont valides. Mais certains points restent à vérifier :

- On vérifie que les deux mots de passe entrés par l'utilisateur sont bien identiques avant de faire des appels à la BDD.

- Avec la méthode findByEmail de la classe Client on vérifie que l'email proposé n'existe pas déjà en BDD, et on fait de même avec le pseudo proposé via findByPseudo. On retourne une erreur en json si c'est déjà le cas, avec un message clair.
- Si pas d'erreur, on convertit le mot de passe de notre utilisateur en hash via bcrypt avec 10 itérations du salage pour renforcer la sécurité du mot de passe.
- Je construis un objet qui reprend toutes les propriétés de notre nouvel utilisateur.

```
const newUser = {
                email: email,
                password: hashedPwd,
                prenom,
                nomFamille,
            };


            const userNowInDb = new Client(newUser);
            const user = await userNowInDb.save();
```

suite de la méthode signin du clientController

- Je crée une nouvelle instance de User qui possède désormais la méthode save que je peux utiliser.

- J'envoie les infos à la méthode d'instance save de manière asynchrone. Notre utilisateur est bien inscrit en BDD désormais.

- Dans le cadre de la mise en place de paiement par Stripe, on peut informer Stripe de l'apparition d'un nouveau client, pour ce faire on envoie à l'API de Stripe les informations sur le client et je stocke le custumer id dans redis pour y avoir accès plus tard, en fournissant la clé (l'email de l'utilisateur) adéquat.

```
let custumer;
            try {
                custumer = await stripe.customers.create({
                    description: 'Un client sfSHOP',
                    email: user.email,
                    name: `${user.prenom} ${user.nomFamille}`,
                    balance: 0,
                });
            } catch (error) {
                console.log("erreur dans le clientController, lors de la creation du client STRIPE ==>> ", error);
                // on renvoie pas une 500.. on laisse filer..
            }


            console.log("custumer dans le clientController, creation du client STRIPE ==>> ", custumer);

            await redis.set(`sf/clientStripe:${user.email}`, custumer.id);
```

insertion d'un client Stripe dans la méthode signin du clientController

- Enfin, j'appelle le service sendEmail qui utilise le package nodemailer pour envoyer un mail de bienvenue au client nouvellement inscrit sur le site.

```

const subject = `Bienvenue sur le site SteamFunk !`;
const text = `Bonjour ${userNowInDb.prenom} ${userNowInDb.nomFamille}.`;
const html = `<h3>Bonjour <span class="username"> ${userNowInDb.prenom} ${userNowInDb.nomFamille}, </span> </h3> <br>`;
sendEmailWithoutTemplate(userNowInDb.email, subject, text, html);
```

envoi d'un email dans la méthode signin du clientController

Authentification, mise en place de la méthode reset password,

Dans ce prochain paragraphe, je vous ferai part de ma réflexion qui dans un premier temps m'a fait installer une méthode d'authentification basée sur le JWT, puis suite à d'autres lectures, mon point de vue a évolué et un mécanisme de session avec cookie stocké côté serveur a été mis en place.

Suite à des expériences passées de la gestion des tokens et refresh tokens présents dans le framework Oauth2.0, je me suis tourné vers une authentification via JWT, pour l'authentification des utilisateurs. En voyant les avantages des JWT intéressants pour mon projet :

- Ils n'ont pas à être stockés côté serveur, le client s'en charge.
- Ils sont peu gourmands en mémoire pour le serveur.
- Le JWT stocké dans le local Storage en front nous préserve d'une attaque CSRF, ce que ne fait pas un cookie envoyé automatiquement.
- Une authentification robuste et sécurisée, et qui, associée à l'envoi d'un refresh token, n'impose pas à l'utilisateur de se reconnecter à nouveau.
- Associée au package JWT-permission, la gestion des droits est très facile à implémenter.

C'est pourquoi, dans un premier temps, la méthode d'authentification a été le JWT en BDD. Tandis que la gestion des droits était gérée par le paquet Express-JWT-permission, me permettant d'utiliser ces méthodes, avec guard.check('admin') par exemple pour restreindre l'accès d'une route au propriétaire d'un rôle précis.

Néanmoins j'avais sous-estimé l'importance de gérer l'intégralité du cycle de vie du token. Et mes premières interrogations sur la pertinence de cette méthode dans mon application se sont révélées lorsque j'ai souhaité gérer la révocation de la session. Jusqu'à présent, pour la déconnexion, le front s'occupait de détruire le token du local storage. Néanmoins, et même si j'avais défini un access token avec un temps d'expiration court, pour un refresh token avec un temps de vie long, si le local storage avait fait l'objet d'une attaque, ou si l'utilisateur l'avait sauvegardé, j'étais dans l'incapacité de lui refuser l'accès à mon API. La solution aurait pu consister à stocker les access tokens dans Redis, mais la gestion des refresh tokens ne faisant déjà pas partie de l'implémentation standard, cela me paraissait une architecture bien plus volumineuse et contraignante que prévu à mettre en place. De plus, d'autres inconvénients se sont manifestés :

- Dans le cas où un utilisateur veut changer de mot de passe avec une authentification effectuée juste avant, le JWT aurait toujours été valide. Il ne pouvait certes plus se connecter avec son ancien mot de passe sur la page de connexion, néanmoins, le JWT généré avec l'ancien mot de passe sera toujours valide.

- Sans stockage des access tokens en BDD, pour qu'un utilisateur soit réellement bloqué par le serveur sur les routes où les ressources sont protégées, j'aurais dû attendre que le JWT expire.
- Comparé à un stockage de session sur Redis (qui peut avoir des délais quasi infimes de 5ms), il semblerait que les JWT, plus volumineux en octet, consomment plus de ressources du processeur pour calculer les signatures cryptographiques et soient en pratique, beaucoup plus lents que des sessions traditionnelles.
- Peu facilement compréhensible si on ne se plonge pas un peu dans le cycle de fonctionnement du token et du refresh token.

  Et au final, pour un mécanisme que l'on dit stateless (pas besoin du serveur), il y avait beaucoup de choses à configurer dans le serveur pour une implémentation correcte …

L'utilité d'un tel mécanisme, semble beaucoup résider dans le cas d'utilisation d'un même compte utilisateur sur plusieurs plateformes, ce qui pour mon site de e-commerce, n'a que peu d'intérêt actuellement. Par ailleurs, express-session fonctionne en production depuis des années sur des milliers de serveurs, bien utilisé et configuré, tout me pousse à penser que c'est une valeur sûre, or, l'implémentation que j'allais installer semblait en fin de compte se rapprocher d'une session classique comme pourrait le faire express-session avec cookie.

Au vue de la petite taille de mon application, de son architecture modeste, d'une sécurité complémentaire qu'il était possible de mettre en place pour lutter contre les attaques CSRF, de la possibilité de stocker les sessions via Redis, de la possibilité de gérer plus facilement l'authentification sans avoir à ajouter de tables à la BDD, de la robustesse d'express-session bien implémenté, je suis revenu à un système de session avec cookie stocké côté serveur. Un mécanisme plutôt sûr, permettant de révoquer un utilisateur à tout moment, sécurisé avec des options robustes (décrites dans la partie sécurité de l'application), et également plus simple à comprendre pour toute l'équipe, ce qui était également important pour moi.

Bien entendu, les MW pour la gestion des droits ont également évolué pour que l'on conserve une protection contre les attaques CSRF, la méthode est la suivante :

- Je vérifie la présence dans l'objet req d'un cookie signé, envoyé, par express lors de la connexion, renvoyé par le navigateur de l'utilisateur.
- Après récupération du header par déstructuration, Je récupère mon custom header qui a comme en tête "x-xsrf-token", un token qui provient du local storage de l'utilisateur. Au préalable, lors de la connexion, je l'ai envoyé dans le body et mis dans le local storage via "localStorage.setItem('xsrfToken', response.data.xsrfToken)".

- Je compare les deux. S'ils ne sont pas identiques, on pourrait être en présence d'une attaque CSRF, avec un cookie envoyé par le navigateur sans que l'utilisateur ne le veuille vraiment, on retourne une 401.

```
const client = async (req, res, next) => {

    try {

      const {
        headers
      } = req;

      const cookieXsrfToken = req.signedCookies.xsrfToken;

      if (!cookieXsrfToken) {
        console.log('Il n\'y a pas de token dans le cookie de session')
        return res.status(401).json('Vous n\'êtes pas connecté. merci de vous connecter.');
      }

      if (!headers || !headers['x-xsrf-token']) {
        console.log('Il n\'y a pas de token CRSF dans le header')
        return res.status(401).json('Vous n\'êtes pas connecté. merci de vous connecter.');
      }

      const headerXsrfToken = headers['x-xsrf-token'];

      if (headerXsrfToken !== cookieXsrfToken) {
        console.log(`Probléme de token csrf dans le auth MW, les deux tokens son différent, headerXsrfToken ${headerXsrfToken} et cookieXsrfToken ${cookieXsrfToken}`);
        return res.status(401).json(
          'Vous n\'êtes pas connecté. merci de vous connecter.'
        );
      }
```

- Je vérifie que mon utilisateur est bien connecté et a le bon rôle pour accéder à la ressource.
- Je laisse la main au MW suivant.

```
if (!req.session.user) {
        return res.status(403).json(
          'Vous n\'êtes pas connecté. merci de vous connecter.'
        );
      }

      if (req.session.user.privilege !== 'Developpeur' && req.session.user.privilege !== 'Administrateur' && req.session.user.privilege !== 'Client') {
        return res.status(403).json({
          message: 'Vous n\'avez pas les droit nécéssaires pour accéder a la ressource.'
        })
      }

      next();

    } catch (err) {
      console.trace(
        'Erreur dans la méthode Client du MW Client :',
        err);
      return res.status(401).json({
        message: 'Erreur lors de l\'autentification'
      });
    }
  }
```

Enfin, durant cette phase de développement sur les méthodes d'authentification , un des derniers passages qui a nécessité une certaine réflexion sur la méthode à utiliser, a été la mise en place d'une méthode pour l'utilisateur, de renouveler son mot de passe. Le mécanisme, en deux temps, est composé d'une première route reliée à un formulaire en front, qui renvoie une adresse email à l'API.

Si celle-ci appartient bien à un utilisateur de la BDD, un email est envoyé à l'utilisateur, lui offrant l'accès à une route qui le renvoie vers un formulaire en front pour rentrer son nouveau mot de passe (et sa confirmation). Et dans un second temps, ces infos sont transmises à l'API sur une seconde route, qui aura pour but d'insérer ce nouveau mot de passe en BDD.

La problématique ici est de pouvoir faire un lien dans l'email qui ne serait pas valide jusqu'à l'expiration du token, mais qui serait invalide à la seconde même où un utilisateur changerait son mot de passe en BDD (tout en mettant quand même, une date de validité). Ce qui signifierait que toute personne souhaitant utiliser ce lien à postériori serait bloqué. Pour ce faire, la solution a consisté à créer une clé secrète de JWT dynamique, et unique pour chaque utilisateur. Et pour ce faire, j'ai chiffré le JWT avec un secret composé du hash de son mot de passe actuel concaténé à sa date d'inscription sur le site. Cette clé secrète me permet de m'assurer qu'à la seconde où l'utilisateur changera son mot de passe dans la BDD, son nouveau hash écrasera l'ancien, qui est notre clé secrète. Le lien précédent devient caduque car le JWT ne pourra plus être déchiffré, et ainsi la route /reset_pwd le renverra vers une 401.

Concaténer sa date d'inscription me permet de m'assurer, dans le cas où un utilisateur utiliserait un mot de passe unique pour plusieurs sites web, et si son mot de passe aurait été découvert par un attaquant sur un autre site, absolument personne ne pourrait reconstituer la clé secrète car personne ne peut savoir l'heure et la seconde précise de création du compte de l'utilisateur.

Cette méthode est fonctionnelle en back, testé sous Postman, mais je ne l'ai pas encore implémentée en front. Ce sera bientôt fait.

```
new_pwd: async (req, res) => {

        try {
            const {
                email
            } = req.body;
            if (!validator.isEmail(email)) {
                console.log("le format du mot de passe ne convient pas au validator")
                return res.status(403).json(
                    'le format de l\'email est incorrect'
                );
            }
            const userInDb = await Client.findByEmail(email);
            if (userInDb === null) {
                console.log(`l'email ${email} n'existe pas en BDD !`);
                res.status(404).json({message: "cet email n'existe pas, assurez vous de l'avoir écris correctement."});
            }
            const secret = `${userInDb.password}_${userInDb.createdDate}`
            const jwtOptions = {
                issuer: `${userInDb.prenom} ${userInDb.nomFamille}`,
                audience: 'envoiresetpwd',
                algorithm: 'HS512',
                expiresIn: '1h' // si l'utilisateur ne valide pas un new password dans l'heure, le token sera invalide.
            };
            const jwtContent = {
                userId: `${userInDb.id}`,
                jti: userInDb.id + "_" + crypto.randomBytes(9).toString('base64'),

            };
            const newToken = await jsonwebtoken.sign(jwtContent, secret, jwtOptions);
            const host = req.get('host');
            const link = `https://${host}/v1/user/reset_pwd?userId=${userInDb.id}&token=${newToken}`;
            const subject = `${shop.nom} : Changement de votre mot de passe`;
            const text = `Bonjour ${userInDb.prenom} ${userInDb.nomFamille}, merci de cliquer sur le lien pour vérifier votre email auprés du site ${shop.nom}. ${link}`;
            const html = `<h3>Bonjour <span class="username"> ${userInDb.prenom} ${userInDb.nomFamille}, </span> </h3> <br>
            <p>Vous souhaitez réinitialiser votre mot de passe du site ${shop.nom}.</p> <br>
            <p>Merci de cliquer sur le lien pour changer votre mot de passe. </p> <br>
            <a href="${link}">cliquez ici pour changer votre mot de passe. </a><br>`;
            sendEmailWithoutTemplate(userInDb.email, subject, text, html);

            res.status(200).json({message:"Merci de consulter vos emails et de cliquer sur le lien envoyé pour renouveller votre mot de passe."});
```

méthode new_pwd pour envoyer un lien temporaire par mail

Mise en place du cache via Redis

Afin de gagner du temps lors de la demande de certaines ressources à l'API, j'ai mis en place Redis pour du cache, avec un middleware composé de deux méthodes, une pour la mise en cache, et une pour vider les données (on parle de fluch) afin que ce qu'il y a en cache dans Redis reflète bien ce qu'il y a dans postgreSQL (on parle d'invalidation de cash événementiel et temporaire). De manière générale pour la mise en cache, je vais déjà vérifier s'il existe une clé dans Redis avec les données, si oui, je court-circuite postgreSQL, je prends les données dans Redis et je les renvoie au front. S'il n'y a pas de données dans Redis, je vais les chercher dans PostgreSQL via Active Reccord (et les modèles), puis lors du renvoi de la réponse attendue au front, je récupère ces données au passage et je les stocke dans Redis, prêtes à être envoyées depuis Redis à leur prochaine demande.

Voyons tout ça un peu plus en détail :

Pour connecter Redis à mon application, j'utilise un client Redis. Après avoir require Redis, on créé un client :

```
1  import { Redis } from "ioredis"
2  const redis = new Redis();
3
4  export default redis;
5
6  //https://luin.github.io/ioredis/index.html#RedisOptions
7
```

Le connecteur Redis

Les mots clés Redis sont disponibles sous la méthode du client Redis (del, get, set, etc…), Les clés n'étant pas indexées dans Redis (Redis ne tient pas d'index des clés à jour), on va donc gérer nous même un index.

La fonction client.keys ('*') que l'on pourrait utiliser est très lente, et plus notre nombre de clés va grandir plus ce sera long. La commande Keys va parcourir toute la mémoire de Redis à la recherche de clés, ce qui pourrait être très long en cas de serveur en fonction depuis longtemps, avec un grand nombre de données. L'index des clés Redis va permettre de savoir à tout moment quelles sont les clés utilisées par mon application dans le cache de Redis. J'instancie un nouvel objet Set qui me permet de stocker des valeurs uniques pour créer mon index (comme un array mais avec une contrainte d'unicité). Un array n'aurait pas pu convenir ici, puisque on va avoir de nombreuses clés identiques, je ne veux surtout pas qu'il me stocke plusieurs valeurs avec la même clé…

Pour pouvoir continuer à utiliser le cache et nodemon ensemble en développement, j'ai dû créer un petit fichier de configuration pour nodemon (nodemon.json) à la racine, qui automatiquement, quand on démarre le back ou quand nodemon relance le back automatiquement, efface toutes les données dans Redis (ensemble clés - valeurs). Ainsi, on est toujours en adéquation entre ce qu'il y a dans notre index des clés (en js) et les clés réellement contenues dans Redis. Sinon Redis contient toujours des clés sans que notre

index le sache, et s'il n'y pas de clés dans l'index, pas de flush qui boucle sur le Set d'index possible, puisqu'il est vide, et Redis retourne des données erronées, différentes de ce que contient postgreSQL (Redis n'a pas pu flush et recharger avec des données fraîches).

```
{
    "events": {
        "restart": "docker exec ecommerce_redis sh -c \"redis-cli --scan --pattern 'sf:*' | xargs redis-cli DEL\""

    }

}
```

Le fichier nodemon.json

"Cache" et "flush" sont des méthodes de mon middleware, ils contiennent donc de manière universelle "request", "response" et "next".

Je définis une clé dynamique et paramétrable, qui correspond à chaque fois à une route différente via l'URL et l'option pour le préfix de la clé. Je vais chercher l'URL dans request, c'est une de ses propriétés (selon la doc d'Express). J'ai ici pris req.originalURL qui est l'URL demandée à la base mais attention à ne pas prendre req.baseUrl qui est l'URL sur laquelle j'ai monté le router (ici /v1, pour faciliter le passage vers une v2 si besoin à l'avenir). Je conserve la bonne pratique Redis qui consiste à mettre un préfix pour nos clés et je le définis comme un paramètre que je pourrai ainsi modifier à ma guise dans le router.

Si la clé existe, je la sors du registre et je réponds directement à l'utilisateur.

J'utilise await car tout ça retourne des promesses et je préfère attendre que Redis ait fait son travail, avant de renvoyer la réponse au front. Mon connecteur Redis va donc aller chercher la clé si elle existe. Redis ne stockant pas des objets mais des strings, on convertit en string avant de le stocker dans Redis et on reconvertit tout en json à la sortie de Redis pour le renvoyer à l'utilisateur en utilisant json.parse sur toutes nos données retournées (la version écrite est équivalente à : JSON.parse(await redis.get(theKey));).

```
const cacheGenerator = (options) => {

    return {
        cache: async (request, response, next) => {

            const theKey = `${options.prefix}:${request.originalUrl}`;

            if (await redis.exists(theKey)) {
                const theValue = await redis.get(theKey).then(JSON.parse);

                response.status(200).json(theValue);

```

Extraction des données de Redis si elles ont déjà été mises en cache

Si la version n'existe pas dans le cache, on intercepte les données :

```
} else {

                const originalResponseSend = response.send.bind(response);

                response.send = (theResponse) => {

                    keysIndex.add(theKey);

                    redis.setex(theKey, options.ttl, theResponse);

                    consol.redis(`la valeur ${theKey} n'est pas dans Redis, on la renvoie depuis postgreSQL`);

                    originalResponseSend(theResponse);
                }
                next();
            }
```

Je veux désormais que response.send, mette dans le cache et réponde, alors qu'avant cela ne faisait que répondre.

Je fais une copie de response.send et je le bind à "response" car je le stocke dans une fonction et non dans une méthode (pour préciser que quand j'utilise originalResponseSend il sera bindé à l'objet response).Je veux que cette fonction continue de fonctionner même si je la déracine de son contexte response. Après je définis l'objet response.send comme étant un stockage dans le cache qui à la fin appelle la version originale de response.send. Ici, j'intercale une mise en cache avant de renvoyer response.send. Quand response.send sera exécuté dans le prochain middleware, c'est notre response.send qui sera en fait lancé.

Je garde mon index des clés à jour, en mettant cette nouvelle clé dans mon objet Set. Redis.setex permet de définir une clé avec une valeur (qui est notre variable theResponse) et un time to live (temps au delà duquel la donnée n'existe plus : invalidation temporaire) défini en option. La méthode redis.setex n'est pas await car sinon je dois async response.send et si je procède ainsi, je la dénature, ce que je ne veux pas, elle n'est pas par défault async, donc pour la garder dans l'état, je ne l'async pas. Dans le cas où l'enregistrement du cache n'est pas terminé, si une deuxième requête arrive très rapidement, la clé n'existera pas encore et ça repassera par le "else" et response.send.

Response.json applique JSON.stringify avant d'appeler response.send. En prenant response.send et non response.json, je n'ai plus besoin de JSON.stringify car c'est déjà appliqué.

Dès qu'on touche à la base de données (donc UPDATE, POST, PATCH, DELETE), on ne prend pas le risque de proposer un contenu périmé à l'utilisateur, aucune donnée invalide, je vide le cash via ma méthode "flush" :

```
flush: async (request, response, next) => {
            for (const key of keysIndex) {
                await redis.del(key);
                consol.redis("on flush dans Redis");

                keysIndex.delete(key);
            }
            next();
```

Pour supprimer les données en cache, on parcourt les clés de notre index ("keysIndex"), on les supprime une par une et on supprime également la clé de l'index via la méthode "delete", toujours à chaque boucle.

Et ainsi dans notre router, on pourra définir quelles routes pourront être mises en cache ou ‘flusher’ les données :

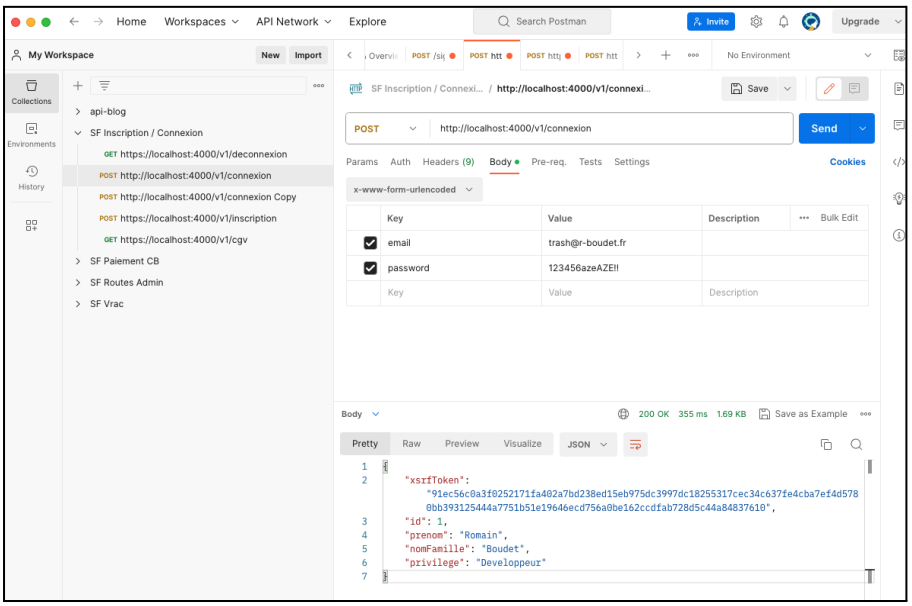
```
/**
 * Une route pour voir tous les articles //! Attention l'adresse de la route est lié à la clé REDIS dans le searchController.
 * route accessible a tous
 * @route GET /user/produits
 * @group Produit - Gestion des produits
 * @summary Affiche tous les articles
 * @returns {JSON} 200 - Les données de tous les produits
 */
router.get('/user/produits', cache, produitController.getAll);
```

Jeu d’essai

Test divers via Postman

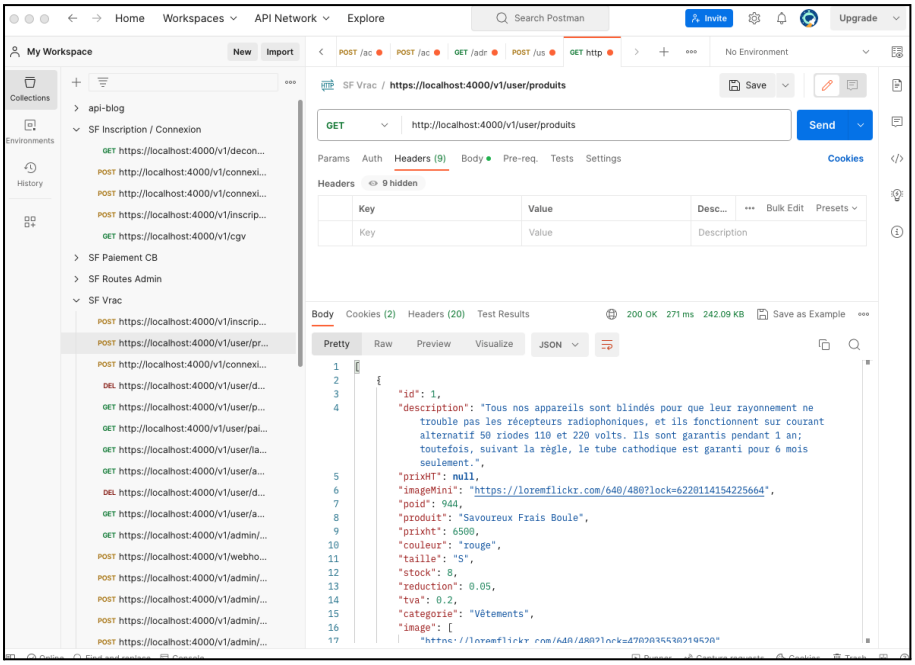
L'API contenant plus de routes que le front n'en utilise, Postman a été utilisé pour tester toutes les routes avec toutes les configurations possibles, en envoyant un header permettant de simuler une authentification avec un x-xsrf-token, et les cookies adéquats. Le x-xsrf-token est envoyé lors de la première connexion :

:



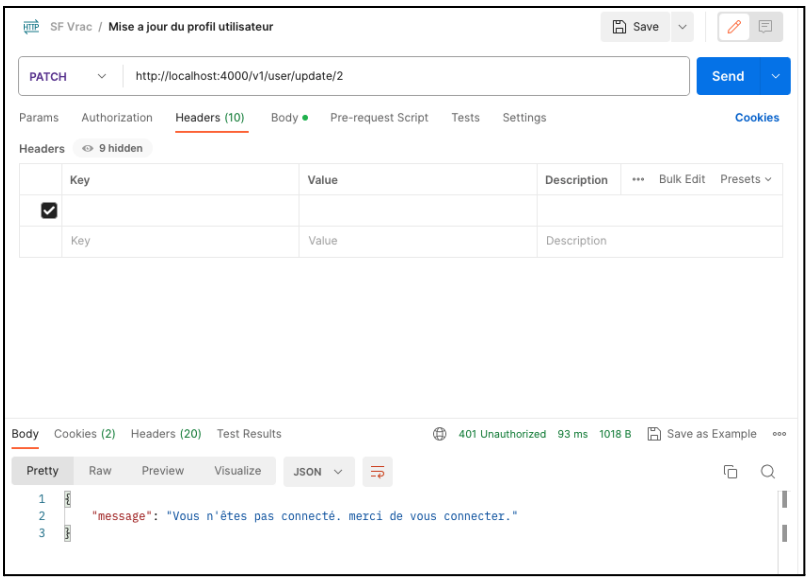
Test d'une connexion :

Le token nous est rendu dans le json.



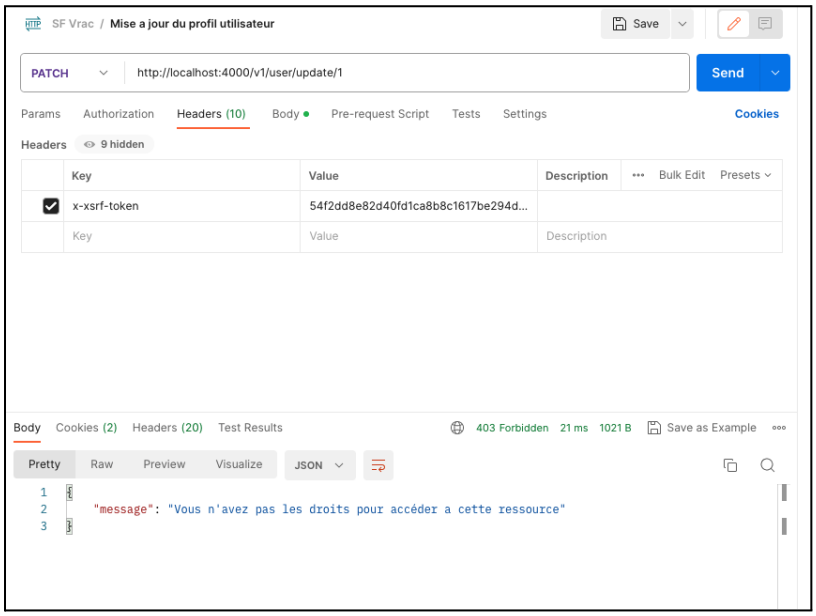
Test d'une récupération de produit (route sans restriction) :

Nous sommes bien autorisés sans x-xsrf-token dans le header.



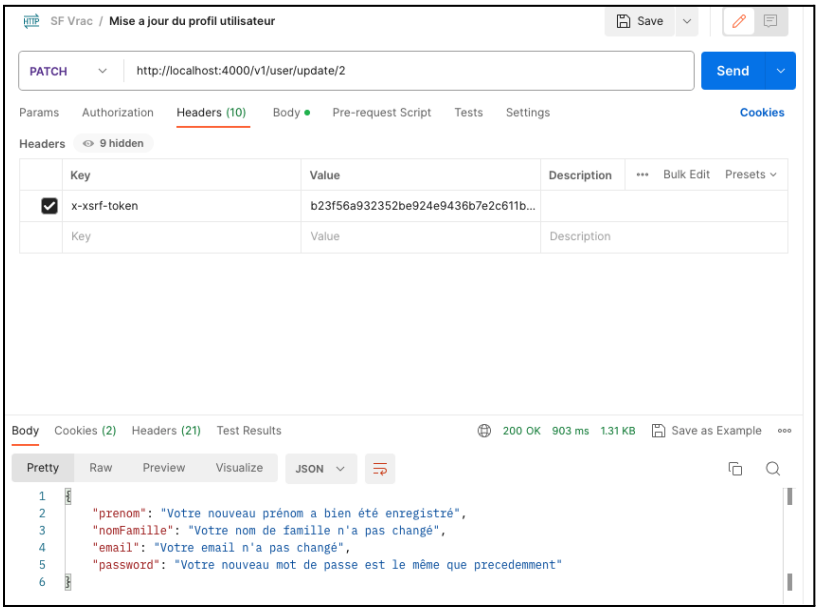
Test de modification du profil utilisateur mais sans token =>

Nous sommes bien bloqués (par le MW client).



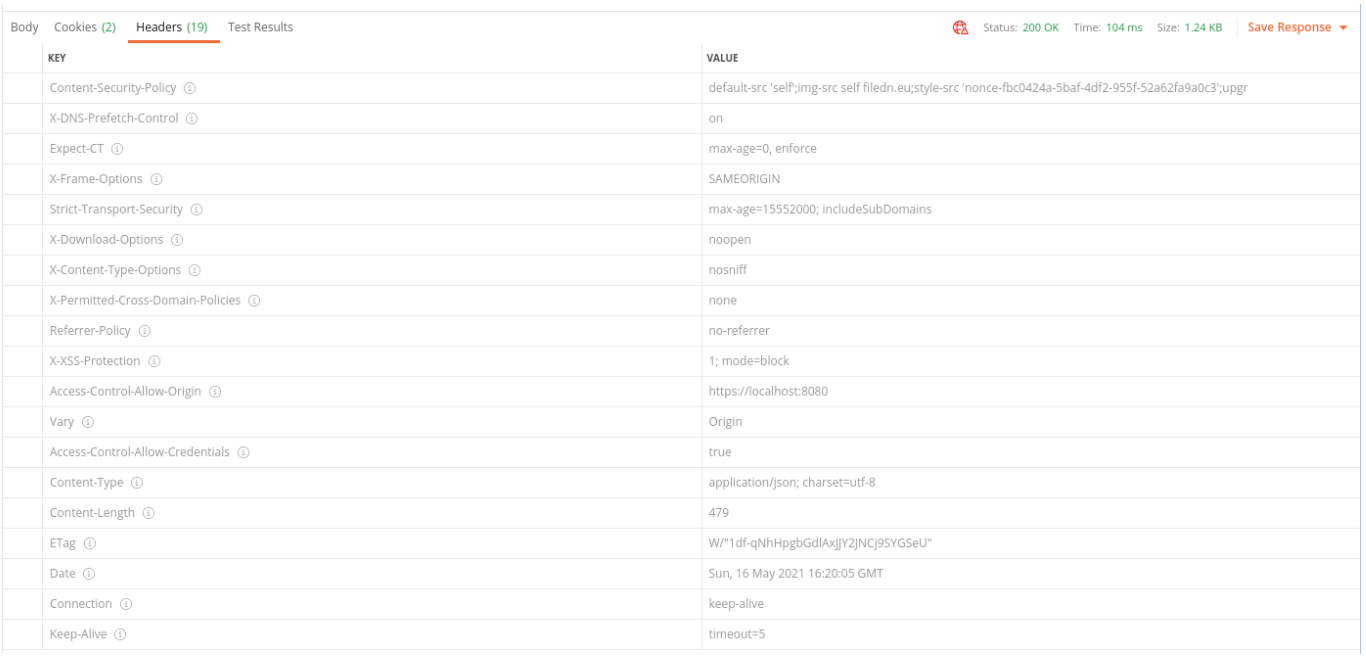
Test de modification du profil utilisateur avec un token client valide mais sans les droits du bon utilisateur =>

Nous sommes bien bloqués (par le controller).



Test de modification du profil utilisateur avec un token client valide et les droits du bon utilisateur =>

Notre modification a bien été prise en compte !



Body Cookies (2) Headers (19) Test Results — Status: 200 OK Time: 104 ms Size: 1.24 KB — Save Response

KEY	VALUE
Content-Security-Policy	default-src 'self';img-src self filedn.eu;style-src 'nonce-fbc0424a-5baf-4df2-955f-52a62fa9a0c3';upgr
X-DNS-Prefetch-Control	on
Expect-CT	max-age=0, enforce
X-Frame-Options	SAMEORIGIN
Strict-Transport-Security	max-age=15552000; includeSubDomains
X-Download-Options	noopen
X-Content-Type-Options	nosniff
X-Permitted-Cross-Domain-Policies	none
Referrer-Policy	no-referrer
X-XSS-Protection	1; mode=block
Access-Control-Allow-Origin	https://localhost:8080
Vary	Origin
Access-Control-Allow-Credentials	true
Content-Type	application/json; charset=utf-8
Content-Length	479
ETag	W/"1df-qNhHpgbGdlAxJJY2JNCj9SYGSeU"
Date	Sun, 16 May 2021 16:20:05 GMT
Connection	keep-alive
Keep-Alive	timeout=5

<u>On remarque que la configuration de nos headers laisse peu de place à la récolte d'informations et offre une première défense contre les attaques les plus basiques, avec dans l'ordre présenté sous Postman :</u>

- La CSP est strictement définie (le nom de domaine pour mes images est défini, toute requête http est upgradée en HTTPS, attribut "nonce" avec uuid dynamique).
- Optimisation des temps de chargement des pages (surtout sur mobiles) en autorisant la résolution de noms de domaine en parallèle de la récupération du contenu de la page via X-DNS-prefetch-control sur on.
- Expect-CT pour l'attente de la transparence de certificat SSL et TLS. Afin que le navigateur refuse toute future connexion qui viole la politique de transparence des certificats, s'il y a problème de certificat, pas de connexion.
- L'en-tête X-frame-Option est définie de façon à ce que seulement mon site puisse utiliser les balises frame, iframe, etc. (protection contre le clic-jacking).
- La Strict-Transport-Security qui prévient les navigateurs que sur ce site on préfère du HTTPS, avec l'option max-age= 15552000 pour que les navigateurs se remémorent cette préférence pendant 180 jours.
- X-download-Option est défini pour Explorer 8 (Il force l'enregistrement des téléchargements potentiellement dangereux, ce qui atténue l'exécution du HTML dans le contexte du site).
- Le reniflage de type MIME est désactivé via "nosniff".
- Une politique CORS strictement définie.
- L'en-tête Referrer (qui contient normalement l'adresse d'une demande) est annulé via la configuration "no-referrer" évitant toute fuite d'information potentiellement non désirée par un utilisateur.
- L'header X-powered-by a disparu, ainsi personne ne sait que j'utilise Express.

Détail et Test d'un paiement

Voyons en détail une fonctionnalité du site, le cas d'un paiement en carte bancaire :

Données en entrée : les coordonnées bancaires d'un utilisateur.

Données attendues : L'utilisateur reçoit un mail confirmant sa commande, l'administrateur du site reçoit un sms et un mail, lui indiquant qu'une nouvelle commande a été passée sur le site (avec le contenu de la commande). Tandis que dans la base de données, le stock du produit commandé est mis à jour en soustrayant les produits récemment achetés, l'information est insérée dans la table "commande" (qui insère les infos générales de la commande: date, commentaire, choix de l'envoi d'un sms quand la commande passe au statut envoyé, et toutes les clés étrangères), dans la table "ligne-commande" (qui fait le lien entre un produit et sa quantité demandée), dans la table "paiement", et enfin créer une facture en pdf disponible dans le dossier "Factures" du serveur. Enfin, le dashboard Stripe contient bien un nouveau paiement.

Le front n'étant pas encore totalement relié à l'API (en cours de réalisation), on simule une première partie du parcours utilisateur sur le site via Postman. On lance une inscription puis une connexion d'un client en ligne via les routes POST '/inscription', puis POST 'connexion', on ajoute des articles dans un panier et on ajoute une adresse d'envoi au client via les routes GET '/user/addPanier/109' et POST '/client/adresse/new'. On insère le choix d'un transporteur via la route POST '/client/livraisonChoix'. On est désormais prêt à réaliser un paiement. Le client va cliquer sur le bouton "Procéder au paiement par CB", ça activera la route '/user/paiementCB' qui en interne, après les vérifications d'usage, va initialiser le paiement, via la méthode Stripe 'paymentIntents.create' :

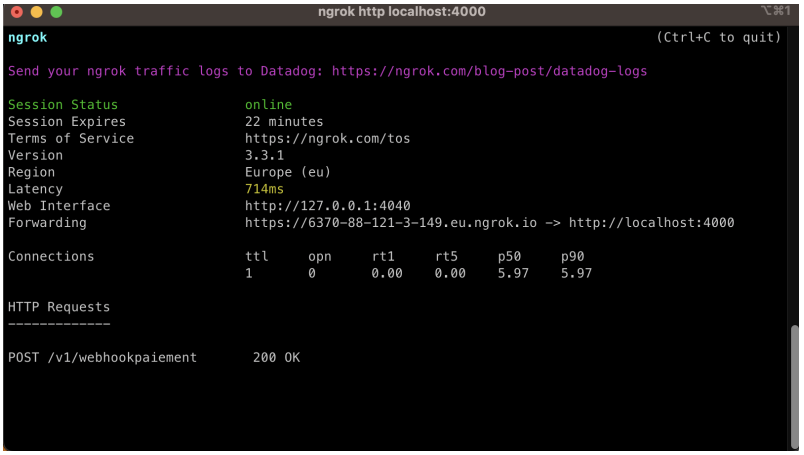
```
const paymentIntent = await stripe.paymentIntents.create({
              amount: req.session.totalStripe, // total en centimes et en Integer
              currency: "eur",
              customer: idClientStripe,
              payment_method_types: ["card"],
              setup_future_usage: "on_session",
              receipt_email: req.session.user.email,
              statement_descriptor: "sf Shop", // Le libellé de relevé bancaire qui app
              metadata: {
                date: `${formatLong(new Date())}`,
                articles: articlesBought,
                client: `${req.session.user.prenom} ${req.session.user.nomFamille}`,
                idClient: req.session.idClient,
                email: req.session.user.email,
                session: req.sessionID,
                amount: req.session.totalStripe,
                ip: req.ip,
                coupon,
              },
```

La méthode Stripe pour initier un paiement

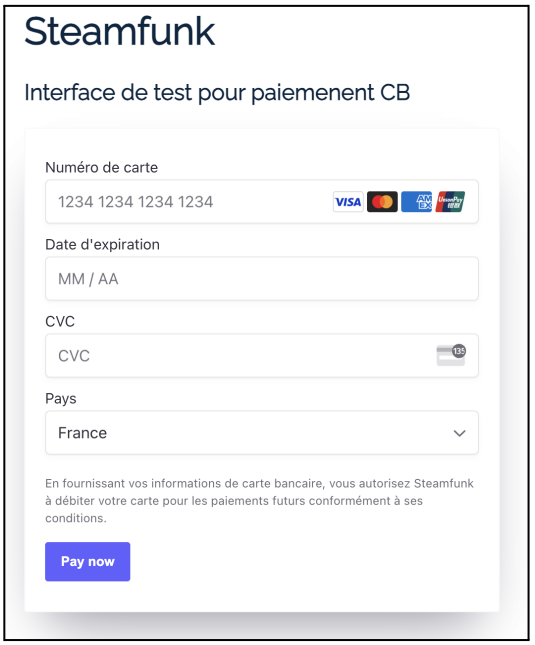
Dorénavant, l'API de Stripe est au courant que mon API souhaite effectuer un paiement dans un futur proche. Stripe nous renvoie un objet "PayementIntent" qui comprend notamment un "id" et un "client_secret", ces deux données sont stockées en session.

En phase de développement, on n'oublie pas de lancer Ngrok pour que le webhook de Stripe puisse contacter l' API, et bien sûr, on configure le webhook, dans le dashboard de Stripe, avec le nom de domaine fourni par Ngrok.

En front, après avoir cliqué sur le bouton "Procéder au paiement par CB", le client va pouvoir renseigner ses coordonnées bancaires :



Ngrok va ouvrir notre API en local sur internet.



Composant react pour l'interface de paiement bancaire

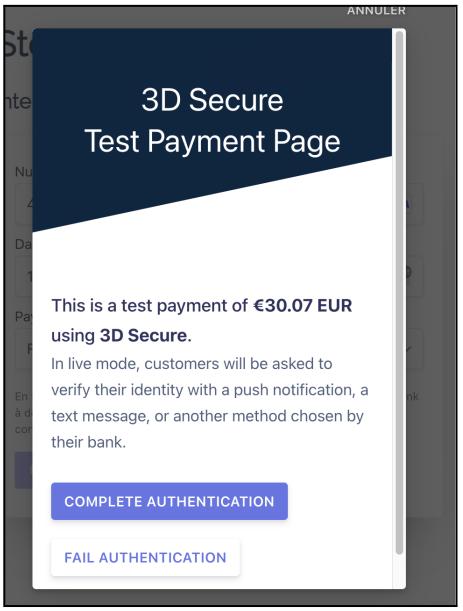
Le front va venir taper sur la route GET 'user/paiementkey', au premier render du composant grâce au "useEffect", qui va lui livrer le 'client_secret' qui va être passé durant le paiement :

```
1  useEffect(() => {
2
3      const fetchData = async () => {
4
5        const options = {
6          withCredentials: true,
7          mode: 'cors',
8        };
9
10       const clientSecret = await axios.get("http://localhost:4000/v1/user/paiementkey", options);
11       setClientSecret(clientSecret.data.client_secret);
12
13     }
14     fetchData();
15   }, []);
```

Méthode de récupération du 'client_secret' en front.

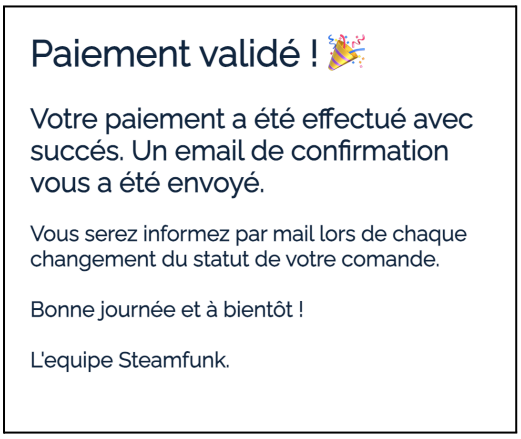
Quand le client va cliquer sur “Pay now”, les coordonnées bancaires du client, le “client_secret” en provenance de mon API et la clé publique Stripe (déjà présente dans notre front), vont permettre à Stripe d’authentifier ce paiement.

Stripe nous permet de tester le 3D secure, avec des codes de carte bancaire spécifiques :

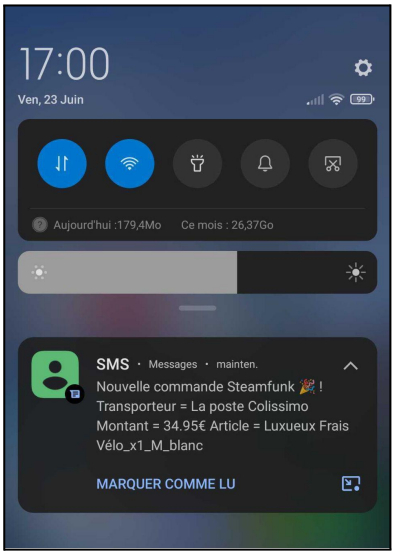


Interface React pour le paiement avec 3D secure

En front, un message nous confirme le bon déroulé du paiement CB :

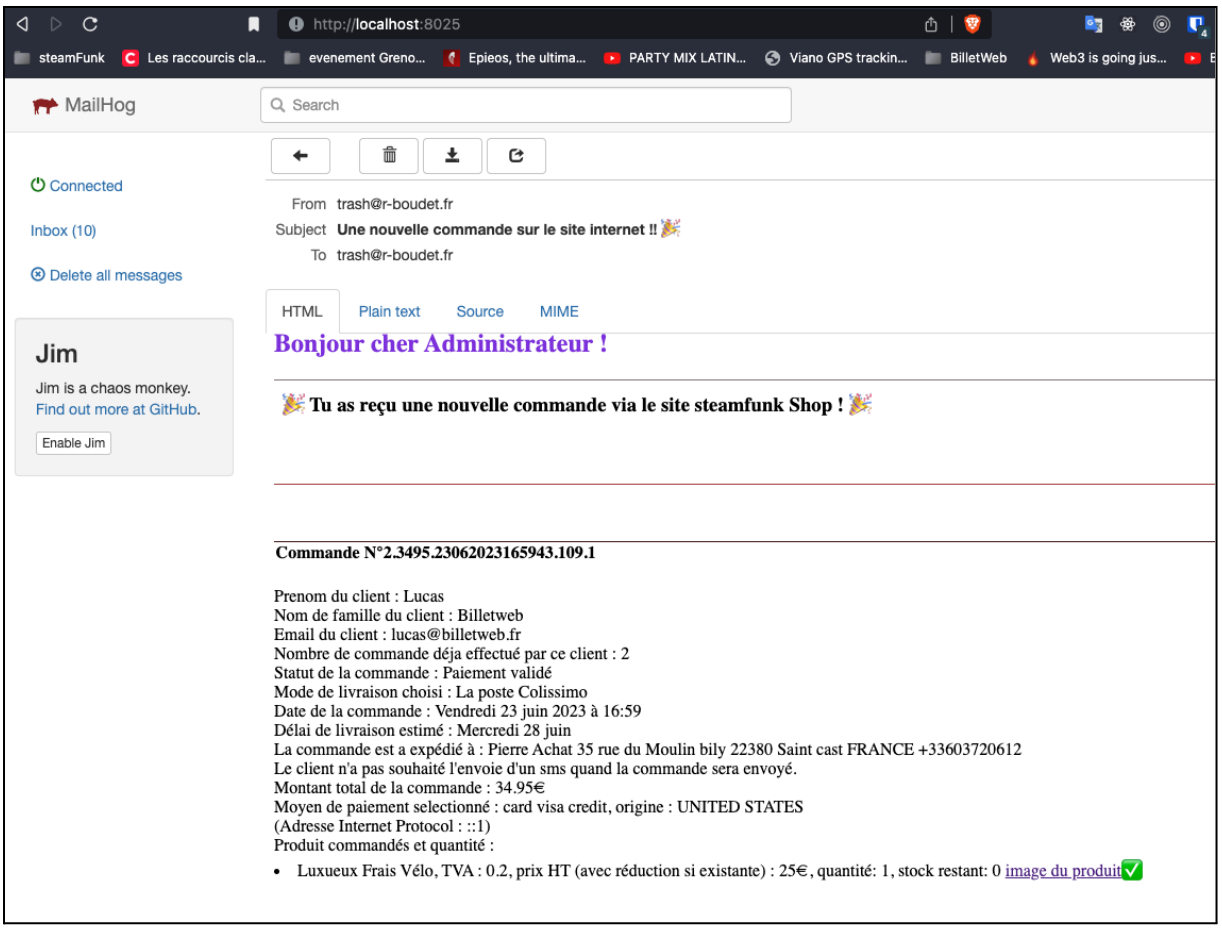


Message en front renvoyé au client



Sms envoyé à l'administrateur

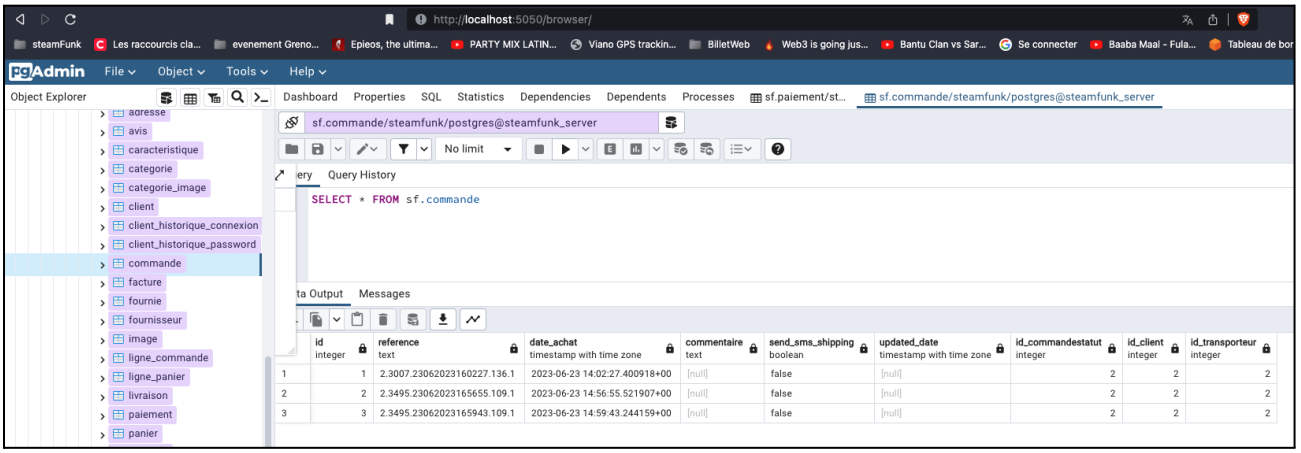
Tandis qu’un email et un sms sont envoyés à l’administrateur, le client reçoit également un email lui confirmant sa commande.



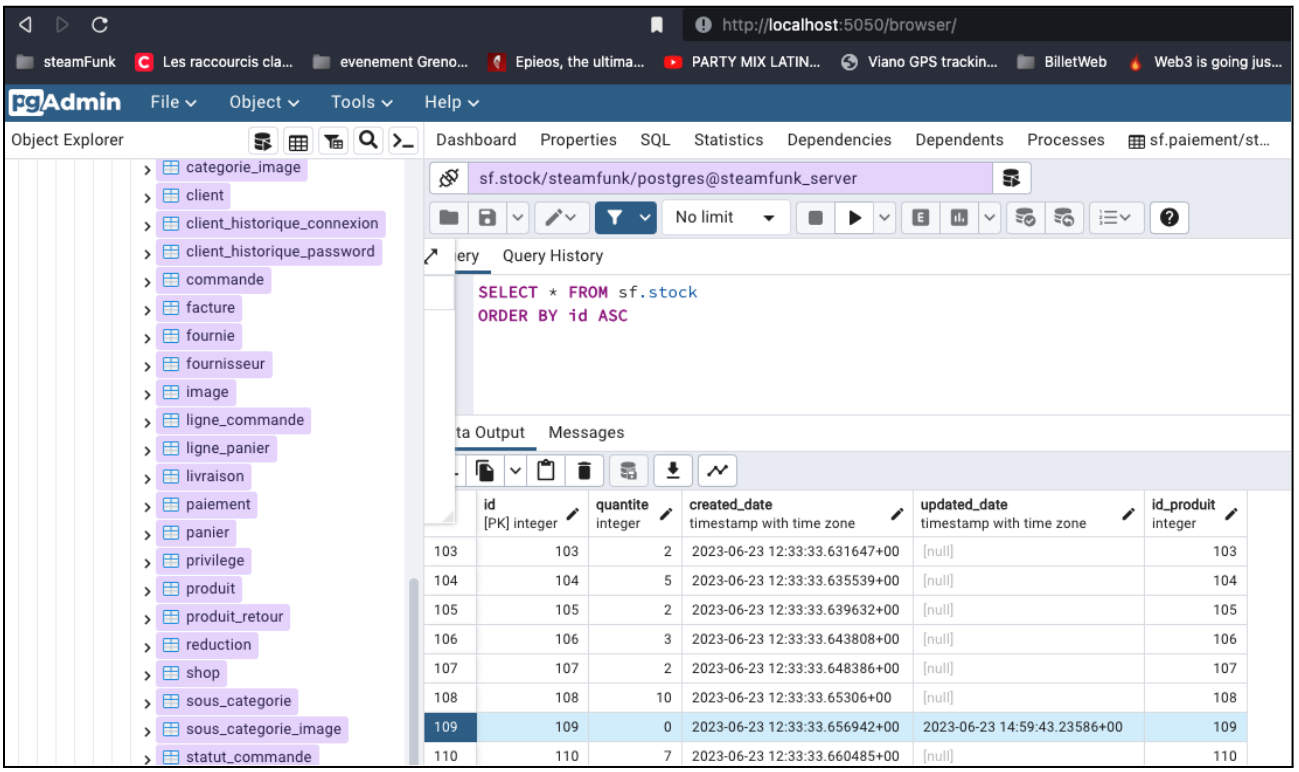
L'email envoyé à l'administrateur lui permettant de préparer sa commande en connaissant le mode de transport choisi par le client, son adresse et les produits à lui envoyer.

Si on re-rentre dans le back pour comprendre le fonctionnement, une fois le paiement effectué par Stripe, il recontacte l'API sur la route POST "/webhookpaiement" pour lui signaler l'état du paiement. Après la vérification de la signature Stripe, on récupère les informations du paiement, leur état, et leur source (paiement SEPA ou CB, d'autre moyen de paiement seront développés plus tard). Selon les cas de figures, si Stripe nous indique que le paiement s'est bien passé, on lance les méthodes pour insérer les données dans les tables 'Commande', 'Paiement', Ligne_Commande', 'Stock', on appel le service Facture pour générer une facture en pdf dans le dossier Factures du server, puis on lance les méthodes pour envoyer des emails, sms à l'administrateur et au client. Dans le cas de paiement refusé, annulé ou en attente (SEPA), d'autres logiques sont activées.

On peut vérifier les tables dans le conteneur Docker de PgAdmin4 sur le port 5050, pour être certain qu'elles ont bien été mise à jour :



Deux nouvelles commandes ont été insérées dans la table commande



Suite à l'achat de l'article 109, dans la table 'stock', la quantité de l'article est passé à zéro et l'attribut updated_date a bien été changé

Données obtenues : l'utilisateur a validé son paiement.

Résultats : la donnée obtenue est identique à la donnée attendue, le test est réussi et notre utilisateur n'a plus qu'à attendre son produit tandis que l'administrateur sait exactement le produit qu'il doit envoyer et à quelle adresse l'envoyer. Et un paiement validé est présent sur le dashboard Stripe.

VEILLE ET TROUBLESHOOTING

Veille technologique et veille sur les vulnérabilités de sécurité

Comprendre l'utilisation des JWT

De manière générale, de très nombreuses ressources ont été utilisées pour la mise en place de l'application. Bien souvent, la lecture quasi-systématique de la documentation des paquets npm, de la documentation d'Express, de Node.js, de REACT et la consultation de nombreux échanges sur Stackoverflow a été nécessaire.

Pour appréhender ce que sont les JWT et quand les utiliser à bon escient, de nombreuses ressources, la plupart en anglais, ont été consultées. Notamment la documentation officielle, des blogs spécialisés, et Stackoverflow :

- https://jwt.io/
- https://www.vaadata.com/blog/fr/jetons-jwt-et-securite-principes-et-cas-dutilisation/
- https://stackoverflow.com/questions/43452896/authentication-jwt-usage-vs-session
- https://ponyfoo.com/articles/json-web-tokens-vs-session-cookies
- https://developer.okta.com/blog/2017/08/17/why-jwts-suck-as-session-tokens
- http://cryto.net/~joepie91/blog/2016/06/13/stop-using-jwt-for-sessions/
- https://medium.com/devgorilla/how-to-log-out-when-using-jwt-a8c7823e8a6
- https://stackoverflow.com/questions/31919067/how-can-i-revoke-a-jwt-token
- https://fusionauth.io/learn/expert-advice/tokens/revoking-jwts/

Veille sur les vulnérabilités de sécurité

Je me suis appuyé sur site snyk.io (snyk.io/vuln/?type=npm) qui recense les failles de sécurité connues des packages npm. Suite à cette veille, tous les packages npm utilisés sont exempts de faille connue à ce jour. Il a été vérifié que le package express-jwt, était bien supérieur à la version 6.00 étant donné que toutes les versions précédentes possédaient de graves failles de sécurité (classées hight severity).

La bonne application de mes options de configuration pour les headers a été analysée via https://observatory.mozilla.org/.

Description d’une situation de travail ayant nécessité une recherche

Pour réaliser ce projet, j’ai dû réaliser de très nombreuses recherches, allant de la simple expression regex, ou de la compréhension de Tailwind CSS, au fonctionnement du framework Next, ou pour une query complexe en SQL.

Prenons le cas de la configuration Docker, dans le fichier docker-compose qui permet de conteneuriser le projet. Comment créer un volume sur le serveur pour la base de données Postgres, sachant que les consignes de la documentation officielle mènent à une erreur. Pour commencer, j’ai effectué une recherche avec les mot clés suivants : “docker compose postgres set volumes”. Mais les résultats n'étaient pas probants, alors j’ai transformé ma recherche en incluant des données de l’erreur : “docker postgres var/lib/postgresql/data permission denied”.

Le premier lien envoie sur stackoverflow avec des réponses intéressantes mais non applicables dans le docker-compose (https://stackoverflow.com/questions/60619659/postgres-mounting-volume-in-docker-permission-denied).

Par contre le deuxième lien (https://github.com/docker-library/postgres/issues/116), qui pointe vers une issue github de l’image docker / postgres, contenait un post d’internaute qui avait fait face au même problème que moi, et expliquait la solution qu’il avait trouvé. En testant sa solution, mon erreur a disparu pour le résultat souhaité.

CONCLUSION

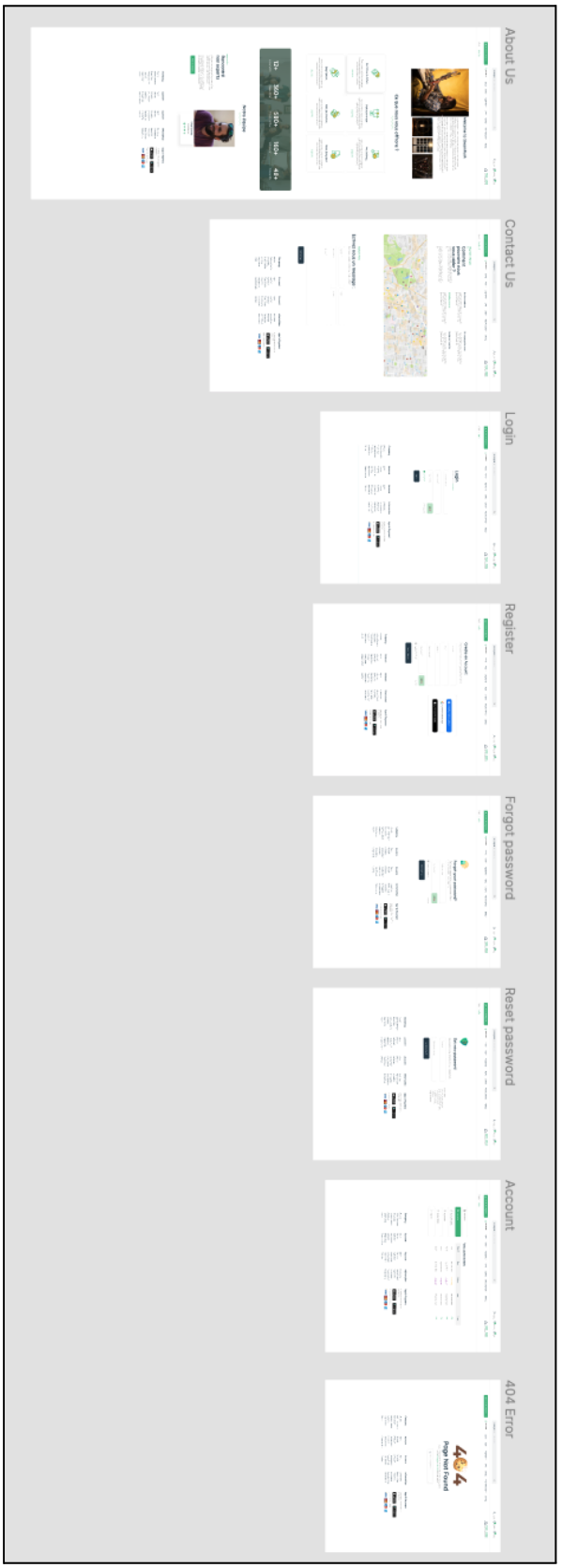
La création de ce site internet, qui n'est pas encore terminé, m'a permis de monter en compétence sur de nombreux points. En front, la manipulation de REACT au travers le framework Next et de sa configuration m’ont permis de perfectionner mes compétences en REACT, tandis que j'ai une vision claire du back, de sa structure et des points de sécurité importants sur lesquels il faut veiller.

La mise en pratique de mes connaissances par la création d'un projet de bout en bout m'a donné un avant-goût de la vie de développeur que j'ai vraiment apprécié. La communication et la coopération avec le client , allant de la réflexion jusqu’à la présentation, toutes ces étapes mises en pratique dans ce projet ont été très précieuses.

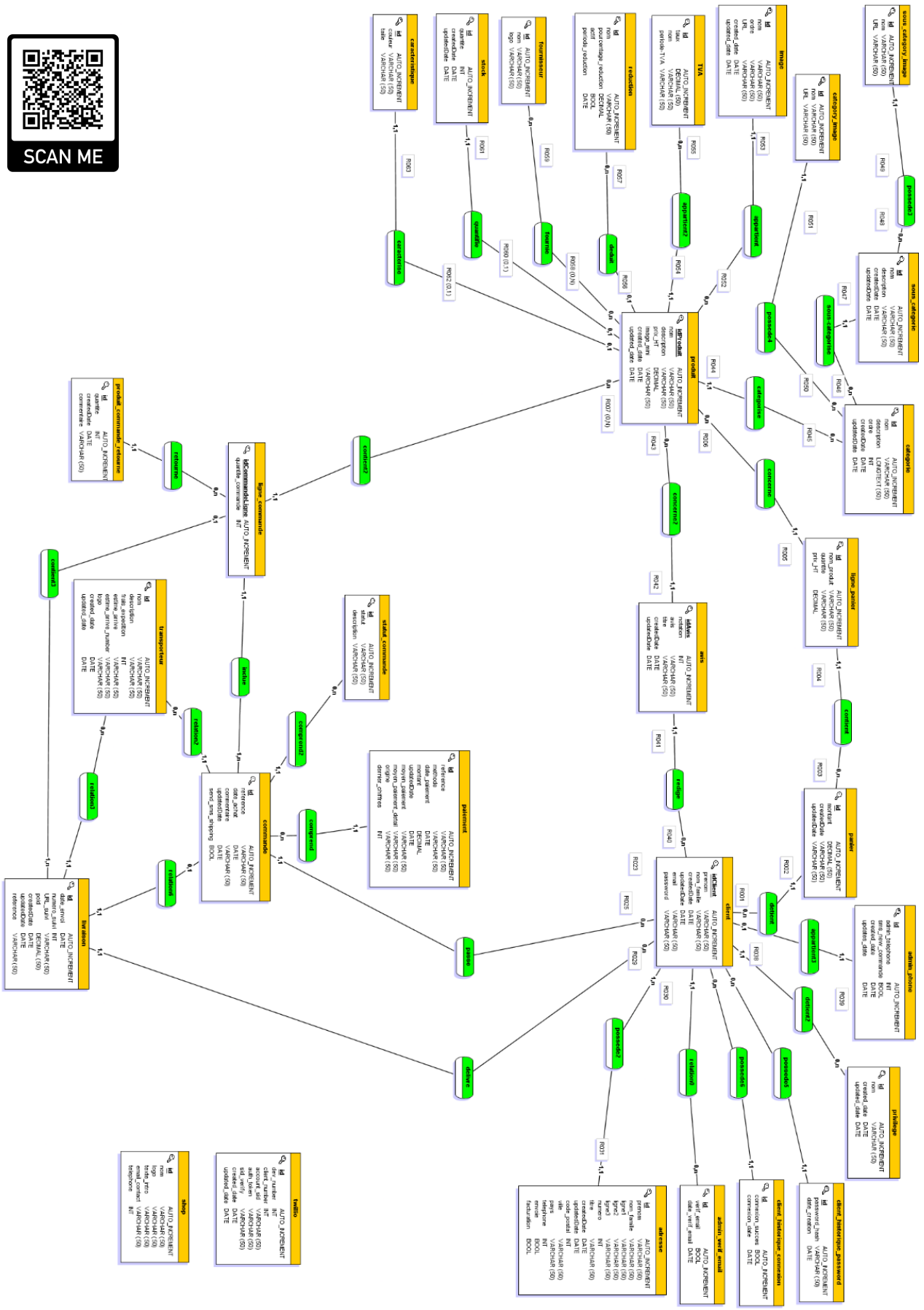
Il me reste de nombreuses choses à apprendre au cours de ma future vie de développeur junior, mais au vu du plaisir pris lors de cette phase de développement, je sais néanmoins que l'envie et la motivation seront toujours là.

ANNEXES

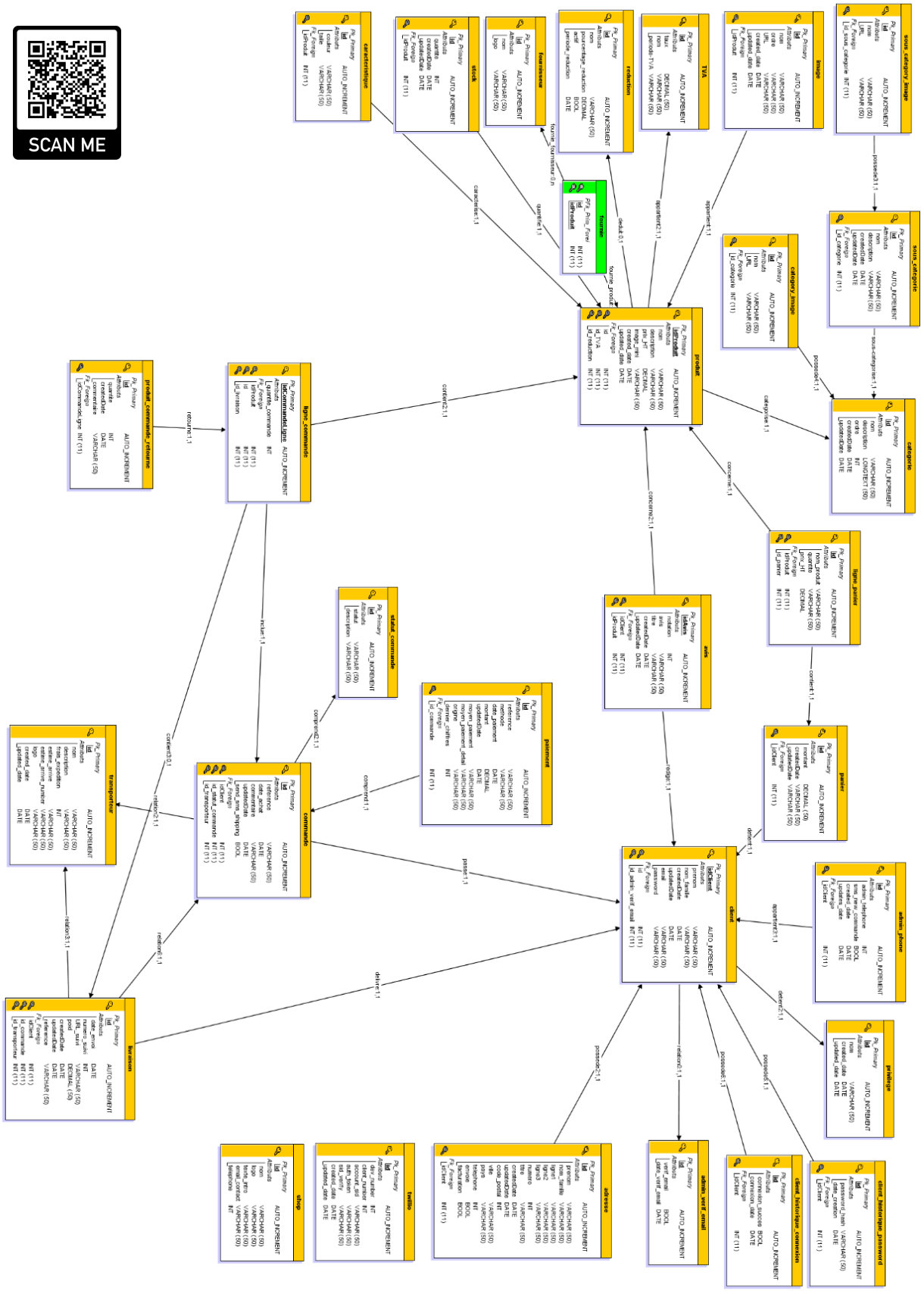
WIREFRAMES



MCD (https://filedn.eu/lD5jpSv048KLfgLMlwC2cLz/MCD%20V31.png)



MLD (https://filedn.eu/lD5jpSv048KLfgLMlwC2cLz/MLD%20V31.png)



Règles de gestion :

Règles de gestion MCD (en partant de "client" vers "panier")

R001 Un client peut avoir 0 ou plusieurs paniers.

R002 Un panier appartient à un, et un seul client.

R003 Un panier peut contenir aucun produit (ligne_panier) ou plusieurs ligne_panier.

R004 Une ligne_panier appartient à un et un seul panier.

R005 Une ligne_panier concerne un et un seul produit.

R006 Un produit peut être dans aucune ou plusieurs ligne_panier.

R007 Un produit peut être dans 0 ou plusieurs lignes (d'une commande / facture / livraison).

R008 Une ligne contient un et un seul produit.

R009 / R010 / R011 => "ligne_livraison", "ligne_commande" et "ligne_facture" hérite de "ligne" et posséde tous les trois les mêmes attributs hérités de "ligne". Les identifiants de ces trois entités sont dans l'entité ligne.

R012 Une ligne de livraison est toujours le fruit d'une ligne de commande.

R013 Une ligne de commande entraîne aucune ou une seule livraison. Pas de livraison multiple ici et une ligne de commande peut ne pas se traduire par une livraison si le produit n'est plus en stock (erreur de stock) ou si le client vient le chercher directement en boutique.

R014 Une ligne de commande peut n'engendrer aucune ligne de facture (si erreur de stock) ou une seule ligne de facture.

R015 Une ligne de facture provient d'une et une seule ligne de commande.

R016 Une ligne de livraison appartient à une et une seule livraison.

R017 Une livraison peut contenir une ou plusieurs lignes de livraison.

R018 Une ligne de commande appartient à une et une seule commande.

R019 Une commande peut contenir une ou plusieurs lignes de commande.

R020 Une ligne de facture est comprise dans une et une seule facture.

R021 Une facture possède de une à plusieurs lignes de facture.

R022 Une livraison délivre un et un seul client. Il s'agit d'une entité faible qui dépend du "client". Bonne pratique contre I/O Bound.

R023 Un client ne peut recevoir aucune livraison.

R024 Une commande est passée par un et un seul client. Il s'agit d'une entité faible qui dépend du "client".

R025 Un client passe aucune ou plusieurs commandes.

R026 Une commande comprend un ou plusieurs paiements. Les paiements multiples ne seront pas mis en place mais si une erreur survient suite à un premier paiement, le client pourra refaire un paiement lié à cette même commande.

R027 Un paiement appartient à une et une seule commande.

R028 Une facture est destinée à un seul client. Il s'agit d'une entité faible qui dépend du "client".

R029 Un client peut détenir aucune ou plusieurs factures.

R030 Un client possède une adresse minimum à plusieurs adresses.

R031 Une adresse appartient à un et un seul client.

R032 Une adresse doit avoir une seule ville.

R033 Une ville ne peut avoir aucune adresse.

R034 Une ville est présente dans un et un seul pays.

R035 Un pays ne contient aucune ou plusieurs villes.

R036 Un code postal peut être associé à une ou plusieurs villes (exemple 54490 => 7 communes).

R037 Une ville peut avoir un ou plusieurs codes postaux (exemple => Metz a 3 codes postaux).

R038 Un client détient un et un seul privilège (rôle).

R039 Un privilège peut être possédé par aucun ou plusieurs clients.

R040 Un client peut ne rédiger aucun avis ou plusieurs avis.

R041 Un avis est rédigé par un et un seul client.

R042 Un avis ne concerne qu'un et un seul produit.

R043 Un produit ne peut posséder aucun ou plusieurs avis.

R044 Un produit est forcément dans une, et une seule catégorie.

R045 Une catégorie ne possède aucun ou plusieurs produits.

R046 Une catégorie peut avoir aucune ou plusieurs sous-catégorie.

R047 Une sous-catégorie appartient à une et une seule catégorie.

R048 Une sous catégorie ne peut posséder aucune ou plusieurs images.

R049 Une "sous-catégorie image" fait référence à une et une seule sous catégorie.

R050 Une catégorie peut ne posséder aucune ou plusieurs images.

R051 Une catégorie image fait référence à une et une seule catégorie.

R052 Un produit ne possède aucune ou plusieurs images.

R053 Une image produit fait référence à un et un seul produit.

R054 Un produit a une et une seule TVA.

R055 Une TVA ne peut s'appliquer à aucun ou plusieurs produits.

R056 Un produit peut avoir zéro à plusieurs réductions.

R057 Une réduction peut s'appliquer à plusieurs produits.

R058 Un produit peut avoir aucun fournisseur (fait maison ou fournisseur inconnu, ou que l'on ne veut pas mettre en BDD) ou plusieurs.

R059 Un fournisseur ne fournit aucun produit (si le produit n'est plus en stock mais qu'il va bientôt le devenir, on veut garder le fournisseur) ou plusieurs produits.

R060 Un produit peut avoir zéro stock ou un seul.

R061 Un stock concerne un produit et un seul.

R062 Un produit peut avoir aucune caractéristique ou une seule.

R063 Une caractéristique (qui peut en comprendre plusieurs : taille et couleur ici) appartient à un et un seul produit.

Users Stories :

Users Stories sf

En tant que	Je veux
client non connecté	Accepter ou ne pas accepter une politique de cookies
client non connecté	Accéder aux différentes catégorie de produit vendue sur la page d'acceuil
client non connecté	Accéder aux détails d'un produit : (*)
client non connecté	* Voir la description des produits vendus
client non connecté	* Voir les photos des produits vendus
client non connecté	* Pouvoir zoomer sur les photos présentées
client non connecté	* Pouvoir faire le tour en 3D des produit vendus
client non connecté	* Voir le prix des produits vendus
client non connecté	* Voir la référence du produit vendu
client non connecté	* Voir le stock de produit disponible ou si oui ou non ce produit est en stock
client non connecté	* Voir les avis des produits vendu
client non connecté	* Pouvoir augmenter / diminuer le nombre de produit que je veux mettre dans mon pannier
client non connecté	* Pouvoir ajouter des produits dans mon pannier
client non connecté	* Pouvoir voir une notification courte me confirmant qu'un produit a bien été ajouté dans mon panier
client non connecté	* Pouvoir voir mon panier
client non connecté	* Pouvoir continuer mes achats aprés avoir vu mon panier
client non connecté	* Pouvoir commander aprés avoir vu mon panier => qui m'enméne sur la page de connexion si pas cannecté
client non connecté	* Accéder a des catégories de produit sur la gauche de la page d'acceuil
client non connecté	* Accéder a des propositions de produits complémentaires / accessoires sous mon produit
client non connecté	Créer mon compte
client non connecté	Me connecter a mon compte

1 / 6

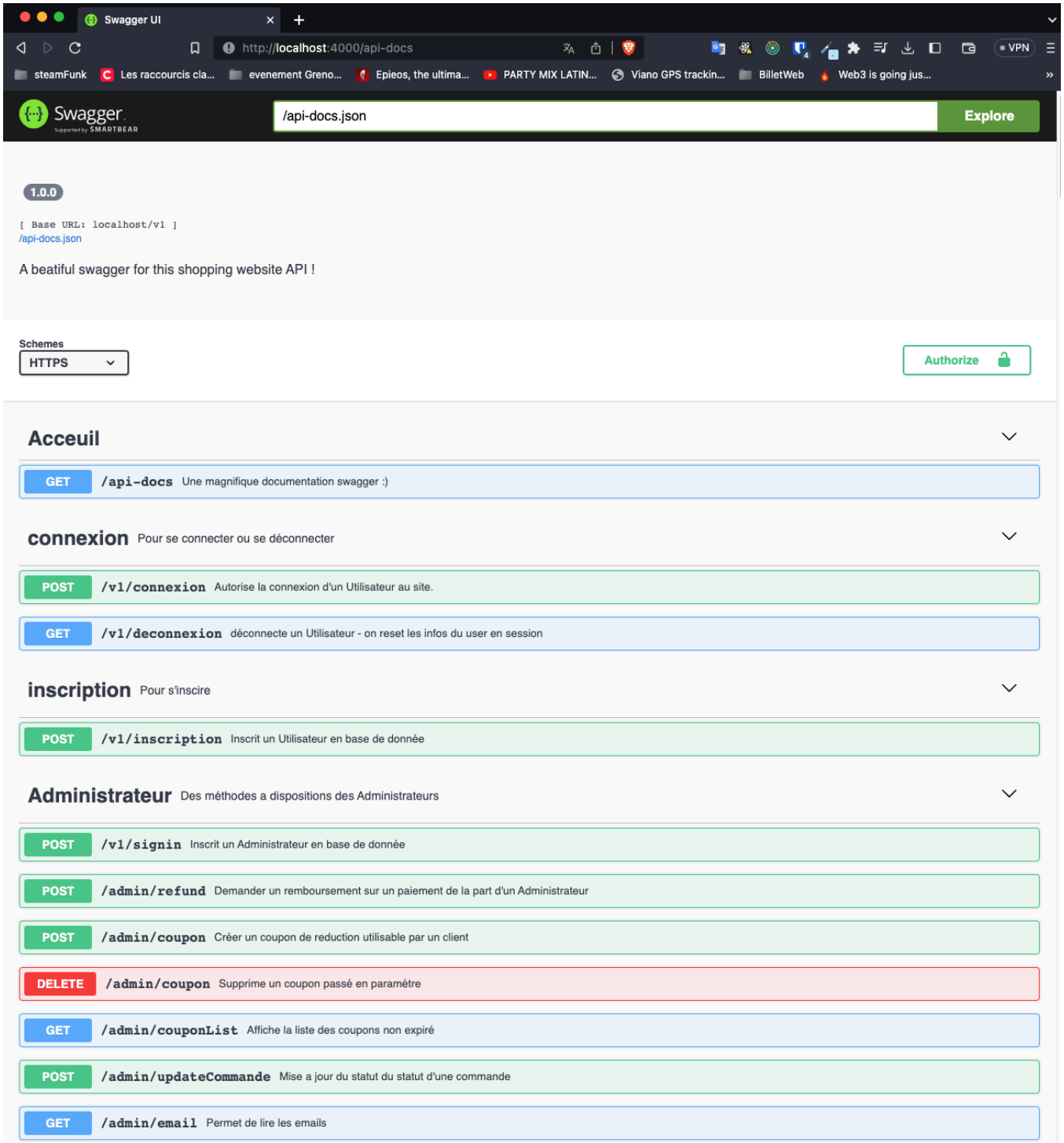
En tant que	Je veux
client non connecté	Rechercher un produit par son nom dans une barre de recherche
client non connecté	En bas de page, bandeau, Acceder sur les informations sur les livraisons
client non connecté	En bas de page, bandeau, Acceder sur les informations sur les différent moyen de paiement
client non connecté	En bas de page, bandeau, Accéder sur les informations sur les retours de produit
client non connecté	En bas de page, Accéder sur les avis données par Trustpilot sur le site
client non connecté	En bas de page, Accéder aux "Mentions légales"
client non connecté	En bas de page, Accéder aux "Conditions Générale de Vente"
client non connecté	En bas de page, Accéder a la page "Qui sommes nous"
client non connecté	En bas de page, Accéder a la page "Nous contacter"
client non connecté	En bas de page, Accéder a la page "Foire aux questions"
client non connecté	En bas de page, Accéder a la page "Nous retrouver sur les marchés"
client non connecté	En bas de page, Accéder a la page "plan du site"
client non connecté	En bas de page, Accéder a la page "suivez nous sur facebook"
client non connecté	En bas de page, Accéder a la page "RGPD"
----------	----------
client connecté	Pouvoir finaliser le processus de vente :
client connecté	* Choisir mon une adresse de livraison
client connecté	* Choisir une adresse de facturation identique a mon adresse de livraison
client connecté	* Pouvoir mettre a jour mes adresses de livraison et de facturation directement durant la commande

En tant que	Je veux
client connecté	* Choisir une option de livraison pour la commande en détaillant le nom du fournisseur, le type / gamme de livraison, le temps de livraison en lien avec la gamme, le prix TTC
client connecté	* Accepter les Conditions Générale de Ventes après le choix du mode de livraison
client connecté	* Pouvoir accéder aux Conditions Générale de Ventes via un lien en même temps que je peux y adhérer : "J'ai lu les conditions générales de vente et j'y adhère sans réserve. (Lire les Conditions générales de vente)
client connecté	* Pouvoir cliquer sur "commander" aprés avoir adheré aux C.G.V. et choisir mon moyen de paiement
client connecté	* Choisir mon moyen de paiement parmis : carte bancaire, paypal, virement bancaire
client connecté	* Rentrer mes informations bancaires et valider le paiement
client connecté	* Avoir un systéme de paiement sécurié via 3D secure avec un code a rentré via sms ou apli pour valider le paiement.
client connecté	* Recevoir une confirmation de paiement bien effectué aprés le paiement
client connecté	* Recevoir un mail m'indiquant que ma commande a bien été pris en compte
client connecté	* recevoir un email quand ma commande à bien été envoyé contenant, le numéro de suivi et un lien pour suivre le colis renvoyant chez le transporteur ou sur mon espace perso.
client connecté	Accéder a la liste de mes produits retournés
client connecté	Accés a l'historique et aux détail de mes commandes (*) :
client connecté	* Accéder au détail de ma commande, avec pour chaque produit commander : référence, nom produit, quantité, prix unitaire, prix total, total TTC, total frais de port, total
client connecté	* Télécharger mes factures au format PDF

En tant que	Je veux
client connecté	* accéder a mon adresse de facturation et mon adresse de livraison
client connecté	* Pouvoir recommander la même commande
client connecté	* Suivre ma commande à chaque état (paiement accepté/Préparation en cour/En cour de livraison / livré)
client connecté	* Connaitre le nom de mon transporteur, le numéro du suivi, date, poid et frais d'expédition
client connecté	* Ajouter un message en lien avec un produit préalablement commandé
client connecté	Accéder a mes adresses de livraison et de facturation
client connecté	Accéder a mes informations personnelles et pouvoir les modifier :
client connecté	* Pouvoir modifier ma civilité
client connecté	* Pouvoir modifier mon prénom
client connecté	* Pouvoir modifier mon nom
client connecté	* Pouvoir modifier mon email
client connecté	* Pouvoir modifier mon mot de passe
client connecté	* Pouvoir modifier ma date de naissance
client connecté	* Mettre a jour mon adresse, la supprimer, ou ajouter une nouvelle adresse
client connecté	Pouvoir supprimer mon compte aprés une confirmation écrite "oui je souhaite supprimer mon compte"
client connecté	Pouvoir effectuer un remboursement immédiat aprés achat, tant que le status de la commande reste "paiement validé" (comme Amazon)
----------	----------
administrateur du site	Pouvoir recevoir en temps réel, par mail ET par sms les commandes qui arrivent pour les traiter au plus vite

En tant que	Je veux
administrateur du site	Bénéficier d'un traitement automatiser aprés une commande, qui lors de chaque commande validé, met à jour les stocks, informe le client par mail que sa commande à bien été prise en compte, envoi un sms et un mail
(selon le choix a l'admin), d'une commande a faire, insére une commande et un paiement dans les tables en bdd, et enfin vire l'argent nouvellement dispo sur le compte STRIPE vers le compte bancaire	
administrateur du site	Pouvoir faire varier l'état d'avancement de l'envoi par sms, mail ou sur le site, dans le back office (en préparation/envoyé)
administrateur du site	Pouvoir déclencher un envoi d'email automatique quand j'update le status de la commande a "envoyée" en rentrant le numéro de suivi
administrateur du site	Pouvoir changer le prix de tous les articles a volonté
administrateur du site	Pouvour ajouter un article a la vente
administrateur du site	Pouvoir ajouter / supprimer une catégorie a la vente
administrateur du site	Pouvoir supprimer un article a la vente
administrateur du site	Pouvoir refuser une commmande en cas de produit non en stock et rembourser l'acheteur directement sur sa carte bancaire
administrateur du site	Pouvoir envoyer un message correspondant à une commande a l'acheteur
administrateur du site	Pouvoir ajouter, supprimer des photos relatif à un produit sur le site
administrateur du site	Voir le nombre total de commandes passées sur le site
administrateur du site	Voir le nombre total de compte client existant dans la BDD
administrateur du site	Voir le volume total de commandes passées en euro
administrateur du site	Changer les stocks rapidement pour chaque produit (faire le liens aprés chaque marché sur ce qu'il reste en stock)

Documentation Swagger API



GET	`/admin/email`	Permet de lire les emails
GET	`/admin/StartUpdateCommandeFromEmail`	Permet de démarrer le serveur qui lira les email et de mettra a jour le statut d'une commande selon le contenu d'un mail
GET	`/admin/stopUpdateCommandeFromEmail`	Permet de démarrer le serveur qui lira les email et de mettra a jour le statut d'une commande selon le contenu d'un mail
POST	`/sendEmailLink`	Prend un mail en entrée et renvoie un email si celui çi est présent en BDD. Cliquer sur le lien dans l'email envoyé enmenera sur la route /verifyemail
POST	`/admin/smsVerify`	Utilise l'API de Twillio. Permet de vérifier un numéro de téléphone
POST	`/admin/smsCheck`	Utilise l'API de Twillio. C'est le retour de la route smsVerify. Insert un téléphone vérifié d'un admin en BDD
POST	`/admin/smsChoice`	Permet d'insérer en BDD le choix de l'admin en matiére d'envoie de sms a chaque commande reçu
POST	`/admin/emailChoice`	Permet d'insérer en BDD le choix de l'admin en matiére d'envoie d'email a chaque commande reçu ou annulée.
GET	`/admin/smsSend`	Utilise l'API de Twillio. Permet d'envoyer un sms sur le numéro souhaité. Relié au numéro de l'admin du site.
GET	`/admin/user/all`	Renvoie un client selon son id
DELETE	`/admin/user/:id`	Supprime un client en BDD selon son id
DELETE	`/admin/user`	Supprime un client en BDD selon son email
GET	`/admin/user/adresse`	Renvoie toutes les adresses des clients en BDD
POST	`/admin/transporteur/new`	Met a jour un nouveau transporteur
DELETE	`/admin/transporteur/:id`	Supprime un transporteur
GET	`/admin/livraisons`	Renvoie toutes les livraisons en BDD
GET	`/admin/produitcommande`	Renvoie toutes les livraisons en BDD
POST	`/admin/livraison/new`	Insére une nouvelle livraison
DELETE	`/admin/livraison/:id`	Supprime une livraison

Utilisateur Des méthodes a dispositions des Utilisateurs

PATCH	`/user/update/:id`	Met a jour un Utilisateur en base de donnée. Un email est envoyé pour signaler les changements. Si changement d'email, un second email est également envoyé sur l'ancienne adresse pour signaler le changement.
POST	`/user/new_pwd`	Prend un mail en entrée et renvoie un email dessus si celui çi est présent en BDD. Cliquer sur le lien dans l'email l'enmenera sur la route /user/reset_pwd ou l'attent un formulaire

POST	/user /reset_pwd	Reset du mot de passe. prend en entrée, newPassword et passwordConfirm dans le body et userId et token en query: decode le token avec clé dynamique et modifit password (new hash + bdd) !
POST	/setcookie	Renvoie un cookie avec le nom 'cookieAccepted' si la valeur cookieAccepted est a 'true'
POST	/cgv	L'acceptation des Conditions Générale de Ventes est stocké en session
GET	/user/paiementCB	Prend en charge le paiement via STRIPE
GET	/user/paiementSEPA	Prend en charge le paiement via STRIPE
GET	/user/paiementkey	Permet de récupérer la clé client secret nécéssaire a STRIPE *** nécéssite d'être authentifié et d'avoir tenté d'effectuer un paiement.
GET	/user/paiementkeySEPA	Permet de récupérer la clé client secret nécéssaire a STRIPE *** nécéssite d'être authentifié et d'avoir tenté d'effectuer un paiement.
POST	/webhookpaiement	Prend en charge le webhook STRIPE apres un paiement validé CB et SEPA
POST	/webhookpaiementSEPA	Prend en charge le webhook STRIPE apres un paiement
GET	/balanceStripe	Connaitre la balance STRIPE du compte
POST	/client/refund	Demander un remboursement sur un paiement de la part d'un Admin
POST	/webhookRefundUpdate	Prend en charge le webhook STRIPE apres un échec ou une mise a jour d'une tentative de remboursement
POST	/webhookRefund	Prend en charge le webhook STRIPE apres un remboursement
POST	/user/coupon	Met a jour le panier d'un Utilisateur avec le montant du panier déduit du montant du coupon
GET	/user/cancelCoupon	Supprime la valeur d'un coupon passé par un Utilisateur et met a jour le panier
GET	/users/facture/:id	Permet de générer une facture PDF pour la commande voulue
GET	/users/readFacture	Permet de lire dans le navigateur une facture PDF pour la commande voulue
POST	/user/searchProduit	Permet la recherche d'un mot ou d'une phrase (une string) dans les produits.
GET	/client/adresses/:id	Renvoie toutes les adresse d'un client en BDD
DELETE	/client/adresses/:id	Supprime des adresses d'un même client
GET	/client/adresse/:id	Renvoie une seule adresse d'un client selon son client_adresse.id
PATCH	/client/adresse/:id	Met a jour une adresse ***néccésite un mot de passe
DELETE	/client/adresse/:id	Supprime une adresse

GET	`/client/adresseFacturation/:id`	Renvoie l'adresse de facturation d'un client selon son idClient
GET	`/client/adresseEnvoie/:id`	Renvoie l'adresse d'envoie d'un client selon son idClient
POST	`/client/adresse/new`	Insére une nouvelle adresse
POST	`/client/livraisonChoix`	Permet de déterminer le choix du transporteur fait par le client et de laisser un commentaire en session
GET	`/user/transporteurs`	Renvoie tous les transporteurs en BDD
GET	`/user/livraisons/:id`	Renvoie toutes les livraisons d'un client en BDD
GET	`/user/produitcommande/:id`	Renvoie tous les produit commandés / livrés pour un client en particulier
GET	`/user/produitLivreByCommande`	Renvoie tous les produits commandés / livré pour une commande particuliére
PATCH	`/user/choixAdresseEnvoi/:id`	Met a jour la nouvelle adresse de livraison
PATCH	`/user/choixAdresseFacturation/:id`	Met a jour la nouvelle adresse de facturation
GET	`/user/panier`	Affiche les articles d'un panier selon la session
GET	`/user/addPanier`	Ajoute un article dans le panier
DELETE	`/user/delPanier`	Supprime un article dans le panier
GET	`/user/lastconn/:id`	Affiche la derniere connexion valide d'un Utilisateur

Developpeur Twillio

GET	`/dev/smsBalance`	Utilise l'API de Twillio. Renvoie la balance du compte par sms au numéro souhaité. Relier au numéro du developpeur.
GET	`/dev/balanceTwillio`	Utilise l'API de Twillio. Renvoie la balance du compte.
POST	`/admin/smsRespond`	Utilise l'API de Twillio. Permet d'envoyer un sms selon le contenu d'un sms reçu. Relier au numéro du développeur
GET	`/dev/allTwillio`	Renvoie les informations de connexion au compte twillio
GET	`/dev/oneTwillio`	Renvoie une information de connexion lié a un compte twillio
POST	`/dev/newTwillio`	Insére une information de connexion lié a un compte twillio
PATCH	`/dev/newTwillio`	Met a jour une information de connexion lié a un compte twillio
DELETE	`/dev/newTwillio`	Supprime une information de connexion lié a un compte twillio


POST /dev/psd2Verify Permet de certifier un paiement, mesure de sécurité redondante si 3D secure... codé pour le plaisir ;)
POST /dev/psd2Check Permet de certifier un paiement, mesure de sécurité redondante si 3D secure... codé pour le plaisir ;)
PATCH /v1/updatePrivilege Transforme un client en Administrateur dans la base de donnée
GET /v1/dev/emailVerif Affiche tous les emails des admins existants, vérifié ou non
GET /v1/dev/emailVerif/:id Affiche un emails d'admin existant, vérifié ou non
GET /v1/dev/emailVerifByIdClient/:id Affiche un emails d'admin existant, vérifié ou non, selon son id Client
POST /v1/dev/newEmailVerif Insére un email d'admins existant, non vérifié
PATCH /v1/dev/updateEmailVerif Met a jour un email d'admins, non vérifié
DELETE /v1/dev/delEmailVerif Supprime un email d'admins
DELETE /v1/dev/delEmailVerifByIdClient Supprime un email d'admin selon son id Client
GET /v1/dev/adminPhone Affiche tous les adminPhone des admins
GET /v1/dev/Phone/:id Affiche un adminPhone d'un admin
GET /v1/dev/PhoneByIdClient/:id Affiche un adminPhone d'un admin selon son id Client
DELETE /v1/dev/delPhone Supprime un privilege
DELETE /v1/dev/delPhoneByIdClient Supprime un adminPhone d'admins selon son id Client
GET /v1/dev/privilege Affiche tous les privilege des admins
GET /v1/dev/privilege/:id Affiche un privilege
POST /v1/dev/newPrivilege Insére un privilege
Panier Gestion du panier en base de donnée
GET /admin/allPanier Affiche les paniers en BDD
GET /admin/panier/:id Affiche un panier en particulier
POST /admin/newPanier Insére un nouveau panier
PATCH /admin/updatePanier/:id Met à jour un nouveau panier

DELETE `/admin/delPanier` Supprimme un nouveau panier

DELETE `/admin/delPanierByIdClient` Supprimme un nouveau panier selon son Id client

Produit Gestion des produits

GET `/user/produit/:id` Affiche un article en particulier

GET `/user/produits` Affiche tous les articles

GET `/user/produitByCategorie/:id` Affiche des articles lié a une catégorie précise

POST `/admin/newProduit` Insére un nouveau produit

PATCH `/admin/updateProduit/:id` Met à jour un nouveau produit

DELETE `/admin/delProduit` Supprimme un nouveau produit

Categorie Gestion des categories

GET `/user/categorie/:id` Affiche toutes les catégories

POST `/admin/newCategorie` Insére une nouvelle catégorie

PATCH `/admin/updateCategorie` Met à jour une nouvelle catégorie

DELETE `/admin/delCategorie` Supprimme une nouvelle catégorie

Caracteristique Gestion des caracteristiques

GET `/admin/caracteristique/:id` Affiche toutes les caracteristiques

GET `/admin/caracteristiqueByIdProduit/:id` Affiche une caracteristique selon son id produit

POST `/admin/newCaracteristique` Insére une nouvelle caracteristique

PATCH `/admin/updateCaracteristique` Met à jour une nouvelle caracteristique

DELETE `/admin/delCaracteristique` Supprimme une nouvelle caracteristique

DELETE `/admin/delCaracteristiqueByIdProduit` Supprimme une nouvelle caracteristique selon son id produit

stock Gestion des stocks

GET `/admin/stock/:id` Affiche un stock

GET `/admin/allStock` Affiche tous les stocks

GET `/admin/stockByIdProduit/:id` Affiche un stock selon son id produit

POST `/admin/newStock` Insére un nouveau stock

PATCH `/admin/updateStock/:id` Met à jour un nouveau stock

DELETE `/admin/delStock` Supprimme un nouveau stock

DELETE `/admin/delStockByIdProduit` Supprimme un nouveau stock selon son id produit

GET `/admin/imageByIdProduit/:id` Affiche une image selon son id produit

Fournisseur Gestion des fourniseurs

GET `/admin/fournisseur/:id` Affiche un Fournisseur

GET `/admin/allFournisseur` Affiche tous les Fournisseurs

POST `/admin/newFournisseur` Insére un nouveau Fournisseur

PATCH `/admin/updateFournisseur/:id` Met à jour un nouveau Fournisseur

DELETE `/admin/delFournisseur` Supprimme un nouveau Fournisseur

Fournie Table de liaison avec Fournisseur

GET `/admin/fournie/:id` Affiche un Fournie

GET `/admin/allFournie` Affiche tous les Fournies

POST `/admin/newFournie` Insére un nouveau Fournie

PATCH `/admin/updateFournie/:id` Met à jour un nouveau Fournie

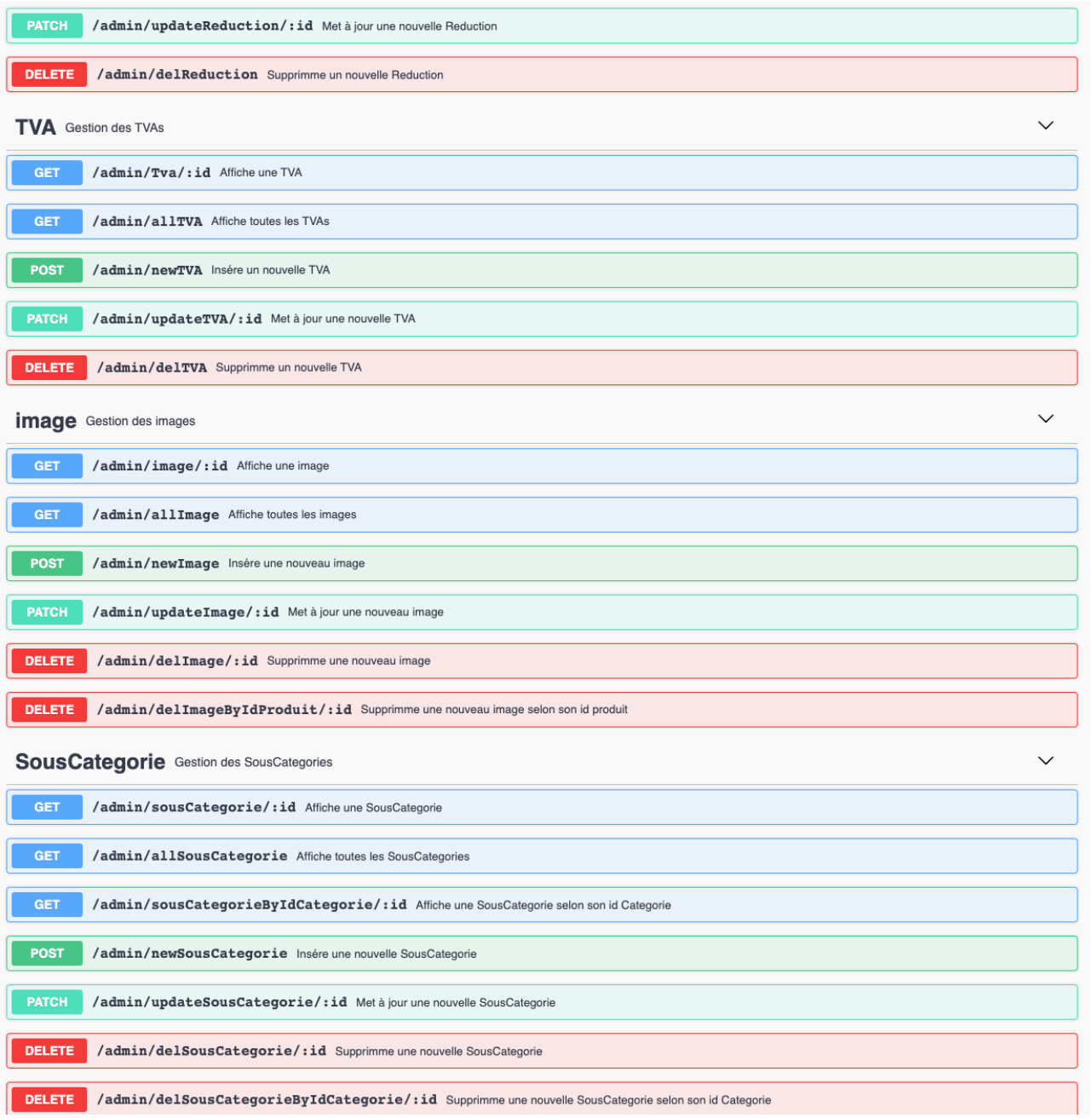
DELETE `/admin/delFournie` Supprimme un nouveau Fournie

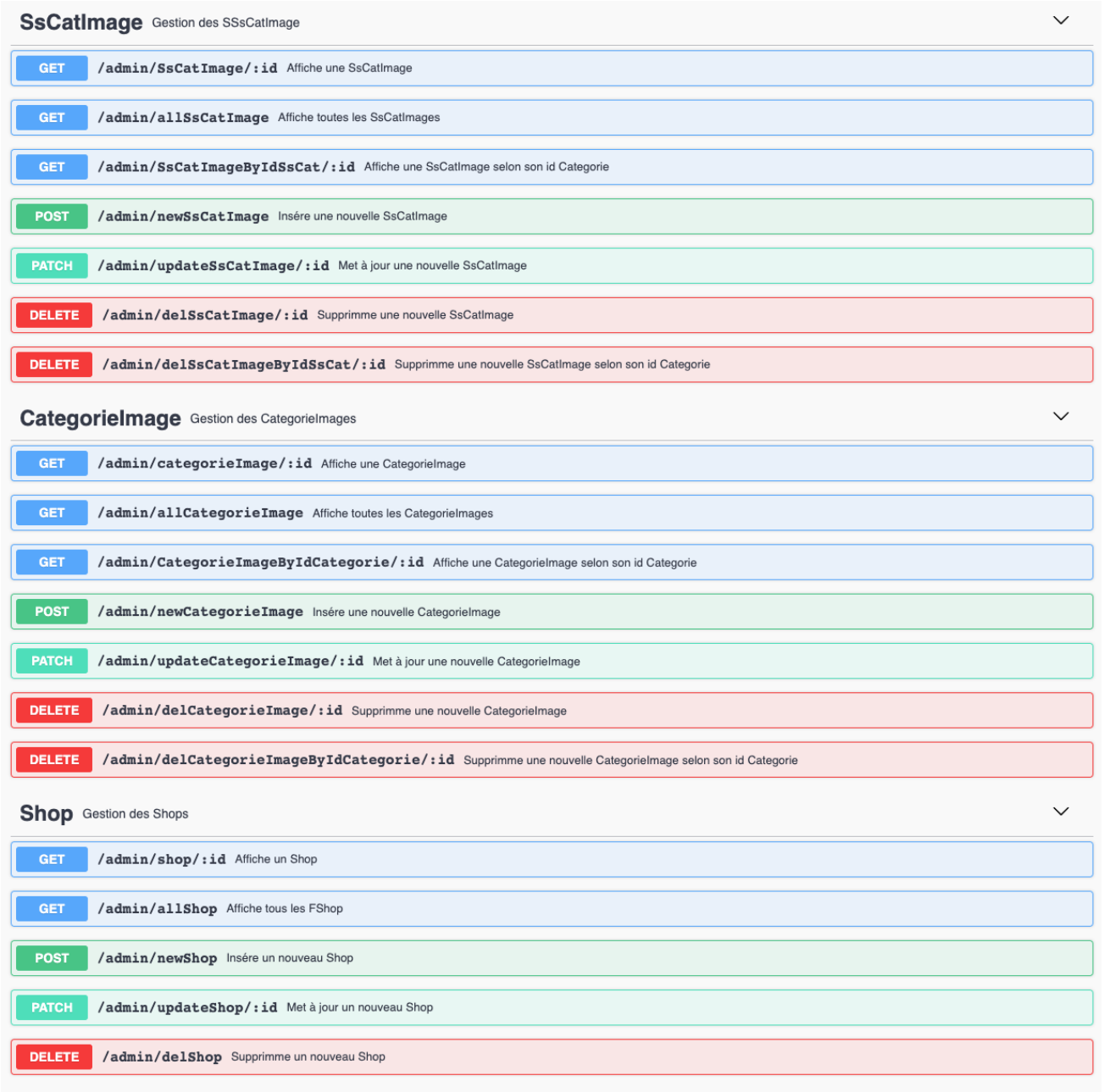
Reduction Gestion des Reductions

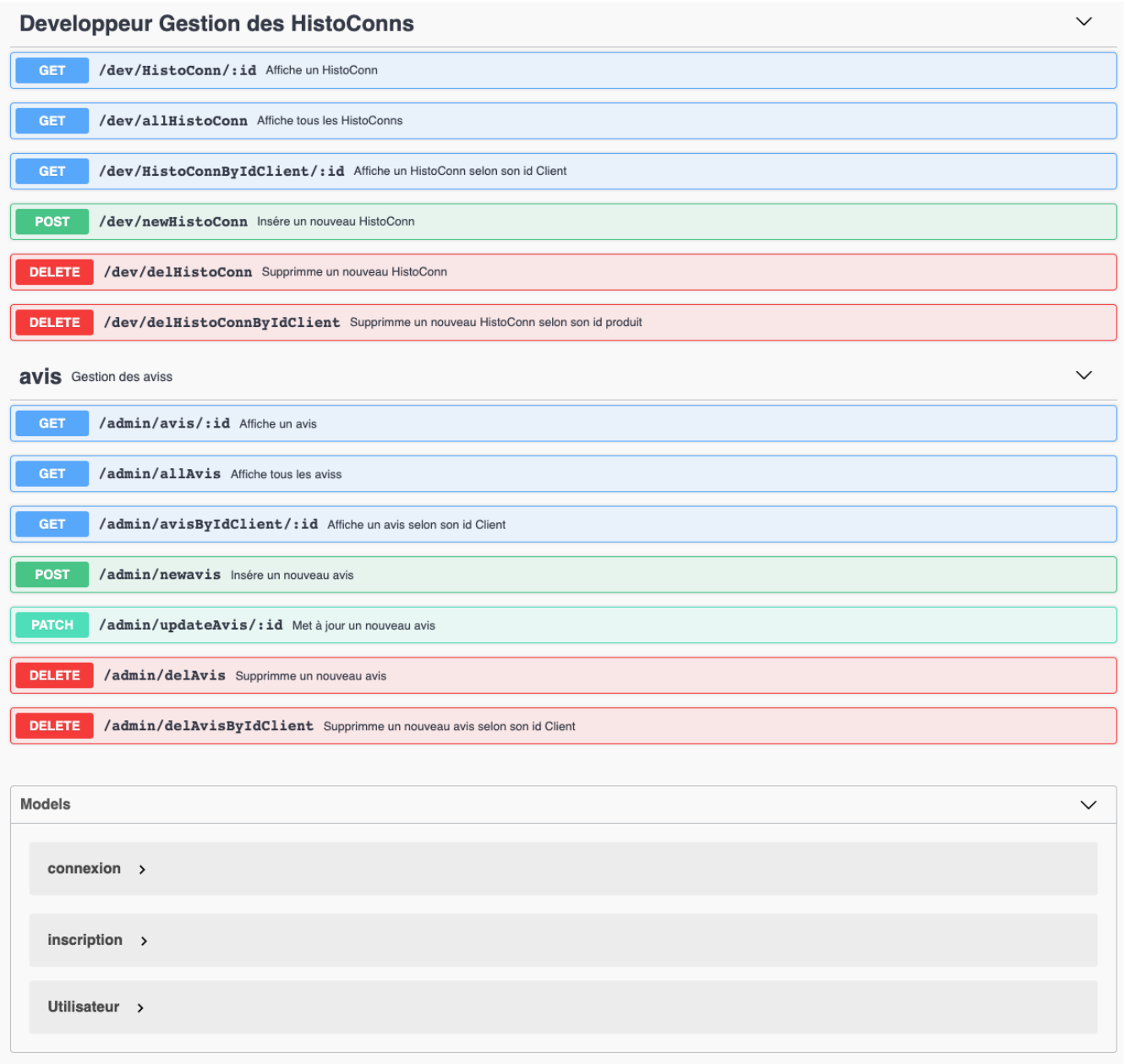
GET `/admin/reduction/:id` Affiche une Reduction

GET `/admin/allReduction` Affiche toutes les Reductions

POST `/admin/newReduction` Insére un nouvelle Reduction


PATCH /admin/updateReduction/:id Met à jour une nouvelle Reduction
DELETE /admin/delReduction Supprimme un nouvelle Reduction
TVA Gestion des TVAs
GET /admin/Tva/:id Affiche une TVA
GET /admin/allTVA Affiche toutes les TVAs
POST /admin/newTVA Insére un nouvelle TVA
PATCH /admin/updateTVA/:id Met à jour une nouvelle TVA
DELETE /admin/delTVA Supprimme un nouvelle TVA
image Gestion des images
GET /admin/image/:id Affiche une image
GET /admin/allImage Affiche toutes les images
POST /admin/newImage Insére une nouveau image
PATCH /admin/updateImage/:id Met à jour une nouveau image
DELETE /admin/delImage/:id Supprimme une nouveau image
DELETE /admin/delImageByIdProduit/:id Supprimme une nouveau image selon son id produit
SousCategorie Gestion des SousCategories
GET /admin/sousCategorie/:id Affiche une SousCategorie
GET /admin/allSousCategorie Affiche toutes les SousCategories
GET /admin/sousCategorieByIdCategorie/:id Affiche une SousCategorie selon son id Categorie
POST /admin/newSousCategorie Insére une nouvelle SousCategorie
PATCH /admin/updateSousCategorie/:id Met à jour une nouvelle SousCategorie
DELETE /admin/delSousCategorie/:id Supprimme une nouvelle SousCategorie
DELETE /admin/delSousCategorieByIdCategorie/:id Supprimme une nouvelle SousCategorie selon son id Categorie


SsCatImage Gestion des SSsCatImage
GET /admin/SsCatImage/:id Affiche une SsCatImage
GET /admin/allSsCatImage Affiche toutes les SsCatImages
GET /admin/SsCatImageByIdSsCat/:id Affiche une SsCatImage selon son id Categorie
POST /admin/newSsCatImage Insére une nouvelle SsCatImage
PATCH /admin/updateSsCatImage/:id Met à jour une nouvelle SsCatImage
DELETE /admin/delSsCatImage/:id Supprimme une nouvelle SsCatImage
DELETE /admin/delSsCatImageByIdSsCat/:id Supprimme une nouvelle SsCatImage selon son id Categorie
CategorieImage Gestion des CategorieImages
GET /admin/categorieImage/:id Affiche une CategorieImage
GET /admin/allCategorieImage Affiche toutes les CategorieImages
GET /admin/CategorieImageByIdCategorie/:id Affiche une CategorieImage selon son id Categorie
POST /admin/newCategorieImage Insére une nouvelle CategorieImage
PATCH /admin/updateCategorieImage/:id Met à jour une nouvelle CategorieImage
DELETE /admin/delCategorieImage/:id Supprimme une nouvelle CategorieImage
DELETE /admin/delCategorieImageByIdCategorie/:id Supprimme une nouvelle CategorieImage selon son id Categorie
Shop Gestion des Shops
GET /admin/shop/:id Affiche un Shop
GET /admin/allShop Affiche tous les FShop
POST /admin/newShop Insére un nouveau Shop
PATCH /admin/updateShop/:id Met à jour un nouveau Shop
DELETE /admin/delShop Supprimme un nouveau Shop


Developpeur Gestion des HistoConns
GET /dev/HistoConn/:id Affiche un HistoConn
GET /dev/allHistoConn Affiche tous les HistoConns
GET /dev/HistoConnByIdClient/:id Affiche un HistoConn selon son id Client
POST /dev/newHistoConn Insére un nouveau HistoConn
DELETE /dev/delHistoConn Supprimme un nouveau HistoConn
DELETE /dev/delHistoConnByIdClient Supprimme un nouveau HistoConn selon son id produit
avis Gestion des aviss
GET /admin/avis/:id Affiche un avis
GET /admin/allAvis Affiche tous les aviss
GET /admin/avisByIdClient/:id Affiche un avis selon son id Client
POST /admin/newavis Insére un nouveau avis
PATCH /admin/updateAvis/:id Met à jour un nouveau avis
DELETE /admin/delAvis Supprimme un nouveau avis
DELETE /admin/delAvisByIdClient Supprimme un nouveau avis selon son id Client
Models
connexion
inscription
Utilisateur

Récap des commandes git :

- git status : Affiche la liste des fichiers modifiés ainsi que les fichiers qui doivent encore être ajoutés ou validés
- git add : utilisée pour ajouter des fichiers à l'index
- git commit -m ""Nom de mon commit" : Valide les modifications apportées au HEAD
- git push : Envoie les modifications locales à la branche associée
- git checkout -b <nom-branche> // git branch <nom-branche> : Crée une branche puis git push --set-upstream origin <nom-branche> pour pousser la branche courante
- git checkout <nom-branche> : Change de branche
- git branch -d < nom-branch > : supprimer une branche localement puis pour la supprimer en remote : git push origin --delete <nom branch
- git branch : Liste les branches
- git log --oneline : Liste les commit avec une seul ligne par commit pour plus de lisibilité
- git checkout < code alphanumérique du commit > : Permet de revenir au commit sélectionné.
- gitk --all : interface graphique pour voir l'avancement des commits => vous avez fait un git checkout <numéro_de_commit> et vous ne voyez plus les commits en amont avec git log --oneline : vous retrouvez ainsi les derniers commits. SousVM et ubuntu : **sudo apt-get install -y gitk**
- git branch -d <nom-branche> : Supprime une branche
- git pull : fusionne toutes les modifications présentes sur le dépôt distant dans le répertoire de travail local
- git merge <nom-branche> : Fusionne une branche dans la branche active
- git diff --base <nom-fichier> : Permet de lister les conflits
- git diff <branche-source> <branche-cible> : Affiche les conflits entre les branches à fusionner avant de les fusionner
- git diff : Permet de lister les conflits actuels
- git stash : Retour au précédent commit //casse les modif effectuées
- git stash list
- git stash drop // git stash drop <nom_fichier_a_dropper>
- git rebase master : Réapplication des commits sur une autre branche
-
- git shortlog -s -n --all --no-merges Pour savoir le nombre de commit par personne

Aller plus loin avec un client git => https://www.sublimemerge.com/

avec le tuto qui va bien : https://blog.shevarezo.fr/post/2018/09/20/sublime-merge-client-git#:~:text=Il%20suffit%20d'ouvrir%20la,et%20filtrer%20ensuite%20par%20terme.