L3 CMI Informatique

HLIN601 – TERL3

CourtCircuit

RAPPORT #2

Bachar Rima
Othmane Farajallah
Wissem Soussi

Responsable CMI Informatique :
Anne-Elisabeth Baert

Encadrant :
Eric Bourreau

5 mai 2018

Table des matières

Remerciements

Nous tenons à remercier toutes les personnes ayant contribué de près ou de loin à la réussite de ce projet, en donnant des conseils ou en répondant à nos questionnements.

Nous souhaitons également remercier en particulier M. Eric *Bourreau* pour l'ensemble de ses efforts au cours de cette année, sachant répondre à l'ensemble de nos interrogations et également pour nous avoir donné l'opportunité de rencontrer des professionnels qui ont pu nous exposer leurs besoins ainsi qu'attentes envers notre projet en tant qu'utilisateurs potentiels.

Chapitre 1

Introduction

1.1 Contexte du projet

Dans ce rapport, nous nous consacrons à la description détaillée de la phase d'implémentation du projet intitulé `CourCircuit`[1] effectué au sein du **LIRMM** (**L**aboratoire d'**I**nformatique, de **R**obotique et de **M**icroélectronique de **M**ontpellier) dans le cadre du module **HLIN601** – **TER** de la 3e année de licence en informatique **CMI** (**C**ursus **M**aster **I**ngénierie).

Le projet se déroule sous l'encadrement de Mme Anne-Elisabeth Baert, enseignante/chercheuse au sein du LIRMM dans l'équipe **MAORE** (**M**éthodes **A**lgorithmes pour l'**O**rdonnancement et les **Ré**seaux), en tant que responsable de la formation CMI informatique et M. Eric Bourreau, enseignant/chercheur au sein du LIRMM dans la même équipe **MAORE**, en tant que responsable pédagogique et encadrant du projet.

Le sujet du projet consiste à créer un site web représentant une interface de communication entre fournisseurs de produits locaux et leurs clients. Il est inspiré du site « La Ruche Qui Dit Oui »traitant les mêmes problématiques, cependant adoptant une approche différente, surtout au niveau de la logistique et de l'architecture du site, afin de fournir une gestion plus optimisée des interactions directes entre clients et fournisseurs.

Nous commencerons ce rapport en faisant un récapitulatif du contexte du projet, des problématiques traitées, de la méthodologie et des outils de conception. De plus, nous proposerons une courte description des outils d'implémentation choisis. Après, nous procéderons à la description détaillée de ces outils et à tout en présentant les résultats obtenus. Enfin, nous conclurons en discutant les difficultés survenues lors du développement et les perspectives du projet.

1. précédemment nommé **LaRuche**

Chapitre 2

Problème, Méthodologie, et Outils

2.1 Problème

Les consommateurs cherchent de *plus en plus* à acheter des produits frais minimisant les étapes de *processing*, alors que les producteurs cherchent à se libérer des centres d'achat et des intermédiaires de distribution.

Dans l'esprit du site francais LaRucheQuiDitOui, on souhaite implémenter une interface sous forme d'un **site web** d'*e-commerce* permettant aux producteurs de vendre des sélections diversifiées de produits aux consommateurs en se regroupant dans des endroits précis de collecte.

Le site web offrira ainsi **tout ce dont il est nécessaire** aux consommateurs pour effectuer des commandes prépayées et précisera ensuite les points de collecte de produits les plus proches. D'autre part, il mettra à la disposition des fournisseurs la possibilité d'organiser ces points, de gérer la mise à jour des stocks et la mise en vente/prise de commandes par les clients, en se basant sur des algorithmes d'optimisation se portant aussi bien sur la gestion de la logistique, la préparation/facturation des commandes, que sur la redistribution/partage des produits entre les différents fournisseurs voisins.

2.2 Méthodologie

Dans le but d'assurer une bonne gestion des ressources et d'offrir le plus de fonctionnalités possibles aux utilisateurs, en les implémentant itérativement dans des versions fonctionnelles et testées du site, nous avons opté pour une approche basée sur les **méthodes agiles** de développement, en particulier

sur **XP** [1].

En effet, la méthodologie de développement proposée par les méthodes agiles étant de plus en plus prépondérante en génie logiciel, nous avons décidé de l'utiliser pour la modélisation et l'implémentation de notre site web afin d'assurer le plus d'extensibilité et de flexibilité possibles.

2.3 Outils de conception

Pour garantir une bonne modélisation du projet, en cohérence avec l'approche de la méthodologie discutée précédemment, nous avons explicité les spécifications fonctionnelles et organisationnelles de notre projet en se servant des outils suivants :

user stories des requis fournis par les utilisateurs décrivant en langage naturel les fonctionnalités qu'ils souhaitent avoir dans le site.

diagrammes de cas d'usage des diagrammes dynamiques, souvent utilisés en **UML** pour décrire en haut niveau les fonctionnalités d'un système, en se servant de notions telles que **acteurs**, **cas d'usage**, **systèmes** et les **relations** entre chacune de ces entités.

modèle EA un modèle **conceptuel** utilisé pour décrire les entités du projet ainsi que les associations décrivant leurs relations et comportements.

schéma de base de données schéma en modèle **relationnel** composé des schémas des relations et des contraintes d'intégrité sur l'ensemble des relations, traduit généralement à partir du **modèle EA** et servant comme **modèle logique** lors de l'implémentation de la **base de données**.

mockup storyboard document de haut niveau offrant un moyen pour schématiser l'utilisation d'un projet, en positionant les différents éléments le composant, sans rentrer dans les détails de leur fonctionnement.

2.4 Outils d'implémentation

Afin de réaliser les impératifs de la partie conception du projet, nous avons prévu, pendant la partie de modélisation, d'utiliser principalement *React.js*, *JQuery*, *Boostrap*, *PHP* et *MySQL*, ainsi que d'autres **API** pour l'implémentation de certaines parties du projet. Toutefois, après avoir compléter

1. *eXtreme Programming*

notre soutenance et suite à des recherches supplémentaires, nous avons décidé d'abandonner quelques technologies et de les remplacer par d'autres plus pertinentes afin d'obtenir un écosystème de travail plus cohérent et plus efficace.

Ce choix fût alimenté par plusieurs facteurs que nous discuterons plus en détail dans le chapitre 3 de ce rapport. Pour cette section, nous nous contentons simplement de fournir une courte description de l'ensemble de l'écosystème qui, en l'occurrence, est composé de :

Front-end

React.js : une bibliothèque **JavaScript** créée par *Facebook* et sortie en 2013 pour la création des interfaces graphiques.

JSX : (*JavaScript XML*) une extension syntaxique du **JavaScript** permettant de simplifier la création des composants *React* à travers une syntaxe à la **XML**.

Bootstrap : un *framework* **CSS** permettant la gestion du *responsive web* du site.

Back-end

Node.js : un environnement d'exécution **JavaScript** côté serveur gratuit et *open-source* utilisant le moteur **JavaScript V8** de *Google Chrome*.

Express : un framework gratuit et *open-source* pour construire des applications web et des **API**s sur **Node.js**.

MariaDB : un serveur de base de données gratuit et *open-source* remplaçant **MySQL** tout en restant compatible avec ce dernier.

Par conséquent, l'architecture des technologies choisies pour l'implémentation de notre projet est illustrée dans la figure 2.1.

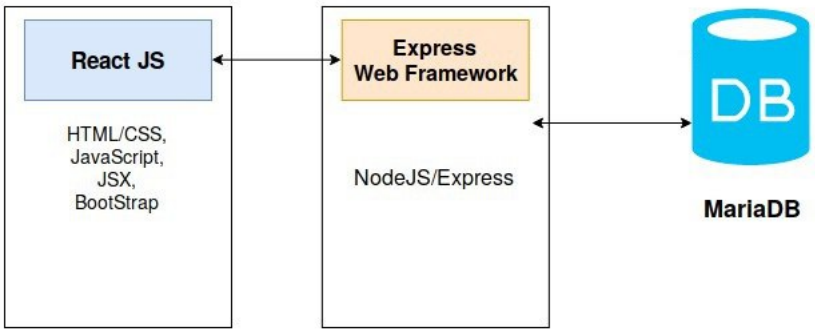

FIGURE 2.1 – L'architecture des technologies utilisées en CourtCircuit

Editeur de texte

Pour la rédaction du code de notre projet, nous avons choisi unanimement d'utiliser l'éditeur de texte gratuit, *open-source* et multiplateforme **Atom** pour son architecture basée sur **Node.js**. En effet, cette architecture permet de personnaliser son utilisation et de le rendre plus souple via des modules téléchargeables par le biais de son gestionnaire de packages **APM** (*Atom Package Manager*).

Chapitre 3

Implémentation et Résultats

3.1 Application web monopage

Avant l'introduction d'**AJAX**[1], un application web classique était composée de plusieurs fichiers **HTML** incorporant la structure de ses pages, avec des feuilles de styles **CSS** pour la mise-en-forme et des scripts **JavaScript** *front-end* classiques manipulant le **DOM** pour rendre les pages dynamiques via l'**API DOM** ou des bibliothèques facilitant cette manipulation telles que **JQuery**.

Après l'introduction d'**AJAX**, qui a permis de changer une page dynamiquement sans besoin de la recharger, une nouvelle structuration des applications webs a commencé à émerger, notamment la structuration monopage (**SPA** ou *Single Page Application*).

Une application web est dite monopage quand elle est composée d'une unique page HTML associée à des ressources Javascript et CSS permettant de changer son contenu dynamiquement. Cette action met en place un effet illusoire de transition entre pages bien qu'il s'agit en réalité d'une seule page rechargée.

Avec une application web classique, le rôle d'un navigateur est, grosso modo, de transmettre au serveur les actions et données saisies par le client et d'afficher les pages HTML générées par le serveur au retour.

Avec une application web monopage, le rôle du navigateur devient plus important, consistant à interagir avec les actions de l'utilisateur et faire une première validation des informations saisies. Le serveur sera libéré ainsi pour les traitements des données reçues et à transmettre tout en effectuant une seconde validation des informations pour assurer leur intégrité.

Une application web monopage devient alors une véritable « applica-

1. *Asynchronous JavaScript and XML*

tion »qui tourne dans le navigateur du client pour améliorer son expérience et éviter de surcharger le serveur avec la génération de pages entières après chaque requête envoyée.

Pour ces raisons nous avons ainsi décidé d'adopter une modélisation monopage pour notre application web tout en expérimentant avec les technologies associées permettant sa mise en œuvre.

3.2 Front-end

Pour la partie *front-end*, nous avons décidé d'utiliser **React.js**/**JSX** pour les interfaces graphiques et **Bootstrap** pour le *responsive web*. Nous détaillerons les raisons pour ce choix et l'utilisation de ces outils dans les sections suivantes.

3.2.1 React.js et JSX

React est une bibliothèque **JavaScript** libre développée par **Facebook** en 2013. Elle permet de créer des interfaces hautement personnalisables et interactives pour des applications monopages.

Nous avons choisi **React** au vu des nombreux points forts que possède le paradigme de la programmation réactive. En effet c'est un style de programmation qui se prête très bien aux interfaces graphiques puisqu'il est généralement possible de définir une représentation statique et plusieurs états de son interface d'une manière asynchrone.

D'autre part, **React** est connue pour sa performance. Les applications développées en **React** peuvent gérer des mises à jour complexes tout en restant rapides et réactives. **React** est aussi modulaire ; au lieu d'écrire de gros fichiers de code denses on peut écrire beaucoup de petits fichiers réutilisables.

Le DOM virtuel

Une page **HTML** classique est représentée sous forme d'un arbre **DOM** dans le navigateur. Quand un changement est appliqué à un ou plusieurs éléments préalablement chargés dans le **DOM**, le navigateur doit effectuer de nombreuses opérations impliquant un coût mémoire important, notamment en rechargeant et repositionnant l'ensemble des nœuds.

Pour minimiser ce coût, **React** agit sur l'affichage dans le navigateur en utilisant un système de **DOM virtuel**. Ce dernier est une représentation du **DOM** en **JavaScript** qui est plus rapide en écriture.

Quand un changement est dû, **React** calcule les changements à opérer et recharge uniquement les parties du **DOM** impactées. Les opérations coûteuses sont ainsi évitées. Cette approche permet ainsi de fournir des performances exceptionnelles.

La modularité

FIGURE 3.1 – Barre de navigation du site fait avec **Bootstrap** et représentant un composant de type *Component*

Une application **React** peut être représentée par un ensemble de composants imbriqués les uns dans les autres. L'intérêt majeur de cette architecture est qu'elle permet la réutilisabilité de « morceaux de code »pour économiser son écriture et améliorer sa lisibilité mais également afin de faciliter la maintenance.

Cette fonctionnalité est un gage de productivité permettant de facilement définir et manipuler l'ensemble des éléments constituant notre interface graphique. Pour implémenter ce concept de composants, nous avons adopté une organisation inspirée du patron de conception *Container Component*. Dans cette organisation, nous avons séparé nos composants en deux catégories :

1. une partie *Components* incluant les éléments qui s'occupent uniquement de présenter les informations au **DOM**. Ce sont donc des éléments indépendants prenant en charge l'aspect visuel de notre application, comme par exemple les éléments *header, footer, navigation bar* (cf. Figure 3.1)
2. une partie *Containers* qui inclut les éléments responsables de gérer les changements de *state* en correspondant chaque état avec un comportement particulier. Ce sont donc des composants de haut niveau qui fournissent aux *Components* les données qu'il faut afficher.

Cette séparation permet d'optimiser la réutilisabilité des composants, et de faciliter la relecture du code en les distinguant selon leurs rôles. Quelques éléments de l'interface graphique sont consultables dans les annexes de ce rapport.

JSX : *JavaScript XML*

React peut s'écrire avec une syntaxe **JavaScript pure**. Toutefois, ce processus devient fort répétitif et illisible. Pour gagner en lisibilité et économie

d'écriture, **JSX** a été introduit comme extension syntaxique du **JavaScript**, d'où notre choix de l'utiliser dans notre projet.

JSX rend le code beaucoup plus souple à lire et écrire, et donc facilite la compréhension du code et le rend plus déclarative. Cependant, **JSX** ne pouvant être interprété par le navigateur, **Babel** a proposé un *preset* permettant de transpiler le code **JSX** en code **JavaScript** classique et interprétable par le navigateur.

3.2.2 Bootstrap et Font Awesome

Bootstrap est un *framework open-source* et gratuit pour le développement *front-end* d'un site web. Il est pricipalement un *framework* **CSS** bien qu'il embarque aussi des composants **JavaScript** et **HTML**.

En utilisant un système de grille pour gérer la mise en forme d'une page, **Bootstrap** est capable d'adapter dynamiquement les éléments d'une page par rapport à la taille de la fenêtre du navigateur, permettant ainsi d'avoir un site web *responsive* utilisable sur toute sorte d'écrans (*smartphone, tablet, moniteur, téléviseur, ...*).

D'autre part, **Bootstrap** offre un ensemble de classes **CSS**, des éléments prêts à l'usage (*boutons, formulaires, accordéons, menus de navigation, ...*), et des animations implémentées en **JQuery**, tout en rendant possible leur personnalisation. Ceci permet alors un développement rapide d'une application web avec peu de lignes de code.

FIGURE 3.2 – *Footer* du site fait avec les icônes de **Font Awesome**

Dans notre projet, nous avons alors eu recours à **Bootstrap** pour éviter la redéfinition de l'aspect esthétique de chaque élément HTML à partir du zéro.

D'autre part, nous nous sommes servis aussi des outils fournis par **Font Awesome**. **Font Awesome** a été créé pour être utilisé avec **Bootstrap** en rendant les icônes utilisables en tant que polices pouvant subir des changement de styles via CSS, SASS et LESS.

Ainsi en se servant de ces deux technologies, chaque composante **React** du site a été esthétiquement personnalisée, par exemple :

— le *header* du site (cf. Figure 3.1) contient une barre de navigation *responsive* pouvant être pliée et dépliée ;
— le *footer* du site (cf. Figure 3.2) contient des icônes fournis par **Font Awesome** comme liens vers d'autres pages.

3.2.3 Charte graphique

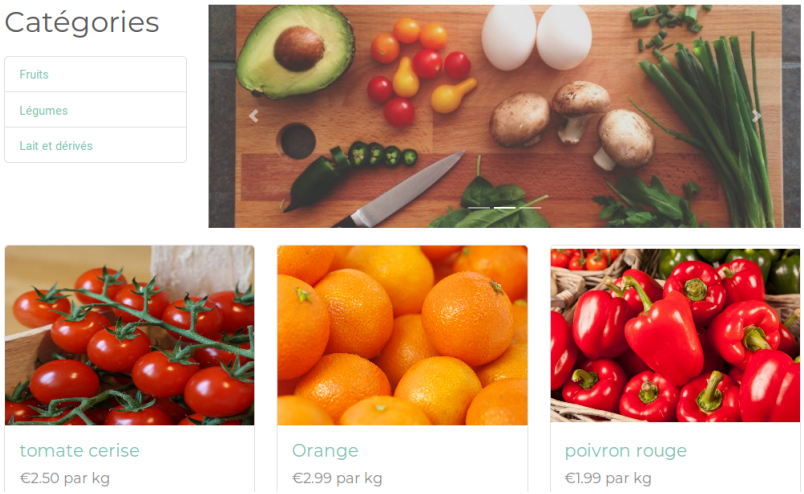

FIGURE 3.3 – Page d'accueil du site web

Le site web, étant principalement un site *e-commerce* de fruits et de légumes [2] (cf. Figure 3.3), nous avons pensé à utiliser pour les éléments du site des couleurs telles que le vert et l'orange. Ces couleurs reflètent un aspect naturel et associe aux produits affichés la qualité de fraîcheur. De plus, les polices, couleurs et logos sont sombres et simples.

3.3 Back-end

Pour la partie *back-end*, nous avons décidé d'utiliser **Node.js**/**Express.js** pour le serveur et **MariaDB** pour la base de données. Nous détaillerons les raisons pour ce choix et l'utilisation de ces outils dans les sections suivantes.

FIGURE 3.4 – le logo de **Node.js**

3.3.1 Node.js

Node.js est un environnement d'exécution **JavaScript** côté serveur utilisant le moteur **JavaScript V8** de *Google Chrome*. Il est libre, gratuit et doté d'une efficacité et d'une légèreté provenant de sa modélisation événementielle, monothread et non-bloquante et de son architecture modulaire. Sa facilité d'usage et son extensibilité est dûe aussi à son gestionnaire de paquets **NPM** (*Node Package Manager*) considéré comme l'un des plus larges écosystèmes de librairies *open-source* au monde.

Node.js est une technologie récente qui est de plus en plus utilisée, surtout en développement web avec les architectures **MERN** (*MongoDB Express React Node*) et **MEAN** (*MongoDB Express Angular Node*). Par suite, ceci nous a motivé à l'intégrer dans notre projet au vu de ses nombreuses avantages.

En premier, notre choix de **Node.js** découle du fait qu'il permet d'écrire du code **JavaScript** du côté serveur. Cette particularité nous a permis d'unifier la rédaction du code en un seul langage, pour les côtés client et serveur. En effet, l'unification du code a rendu possible à l'ensemble du groupe de participer au débogage du projet entier sans aucune restriction liée à la maîtrise d'un langage ou l'autre.

Outre l'unification du code, étant donné que nous aurons besoin de gérer un ensemble important de données dans le cadre de notre projet, l'aspect monothread et non-bloquant de **Node.js** assure une performance fluide que nous ne trouverons pas forcément avec d'autres technologies *back-end*, utilisant généralement des modèles multi-processus ou multithread.

Enfin, **Node.js** est le choix par défaut pour une architecture utilisant **React.js**[3], ce qui nous a encouragé davantage à s'en servir.

Dans le cadre de notre projet, nous avons principalement utilisé **Node.js** pour la création d'une **API** côté serveur permettant d'interfacer avec la base de données. Ceci fût concrétisé par la création des objets pour les différentes entités manipulés (*Client, Utilisateur, Produit, ...*), en s'appuyant sur des concepts de la programmation orienté-objet. Par conséquent, nous avons pu

2. entre autres
3. bien que pas le seul

factoriser le code pour éviter les redondances et améliorer sa lisibilité lors de l'exécution des requêtes sur la base de données.

D'autre part, nous nous sommes également servi de **Node.js** pour l'utilisation du framework **Express** que nous détaillerons dans la section suivante.

3.3.2 Express

FIGURE 3.5 – Le logo de **Express**

Express est un framework minimaliste gratuit et *open-source*, devenu standard pour construire des applications web et des **API**s côte serveur sur **Node.js**. Il s'appuie sur des concepts **MVC** (*Model View Controller*) et offre plusieurs services notamment l'utilisation de moteurs de templates (*EJS (Embedded JavaScript), Handlebars, et Jade*), l'utilisation de *middleware*, la gestion des routes et des formulaires, ...

Dans le cadre de notre projet nous nous sommes principalement servi d'**Express** pour la gestion côté serveur des routes implémentées par **React** en implémentant les opérations **CRUD** (*Create, Read, Update, Delete*) correspondantes.

3.3.3 MariaBD

FIGURE 3.6 – Le logo de **MariaDB**

En général, le serveur de base de données **MongoDB** est le choix par défaut pour une architecture incluant **React**, **Node.js** et **Express**. Ceci est dû principalement à sa modélisation **NoSQL** basée sur des documents au format **JSON** facilitant la récupération et le stockage des données à travers une technologie *back-end* utilisant **JavaScript** telle que **Node.js**.

Toutefois, vu que nous avions déjà traduit notre **modèle EA** (cf. Figure 3.7) en **modèle relationnel**, et vu que le serveur de la faculté qui nous a été attribué pour le déploiement de notre site n'hébérgeait pas un serveur **MongoDB** mais hébérgeait bien son contrepartie **MySQL**, nous avons opté pour un **SGBD** relationnel classique, notamment **MariaDB**.

MariaDB est un serveur de base de données gratuit et *open-source*, assurant l'interopérabilité avec **MySQL**, qui fût introduit par les créateurs de **MySQL** suite à l'achat de ce dernier par **Oracle**. Il s'agit en réalité d'un fork communautaire de **MySQL** qui est mis à jour plus souvent que **MySQL**.

Pour toutes ces raisons, nous avons ainsi décidé d'adopter **MariaDB** comme **SGBD** de notre base de données.

Pour interfacer **Node.js** et **MariaDB** nous avons eu recours au *package* « `mysql` »que nous avions téléchargé via `npm`. Le package contient l'ensemble des fonctionnalités permettant de se connecter à la base de données et d'exécuter des requêtes **SQL** là-dessus.

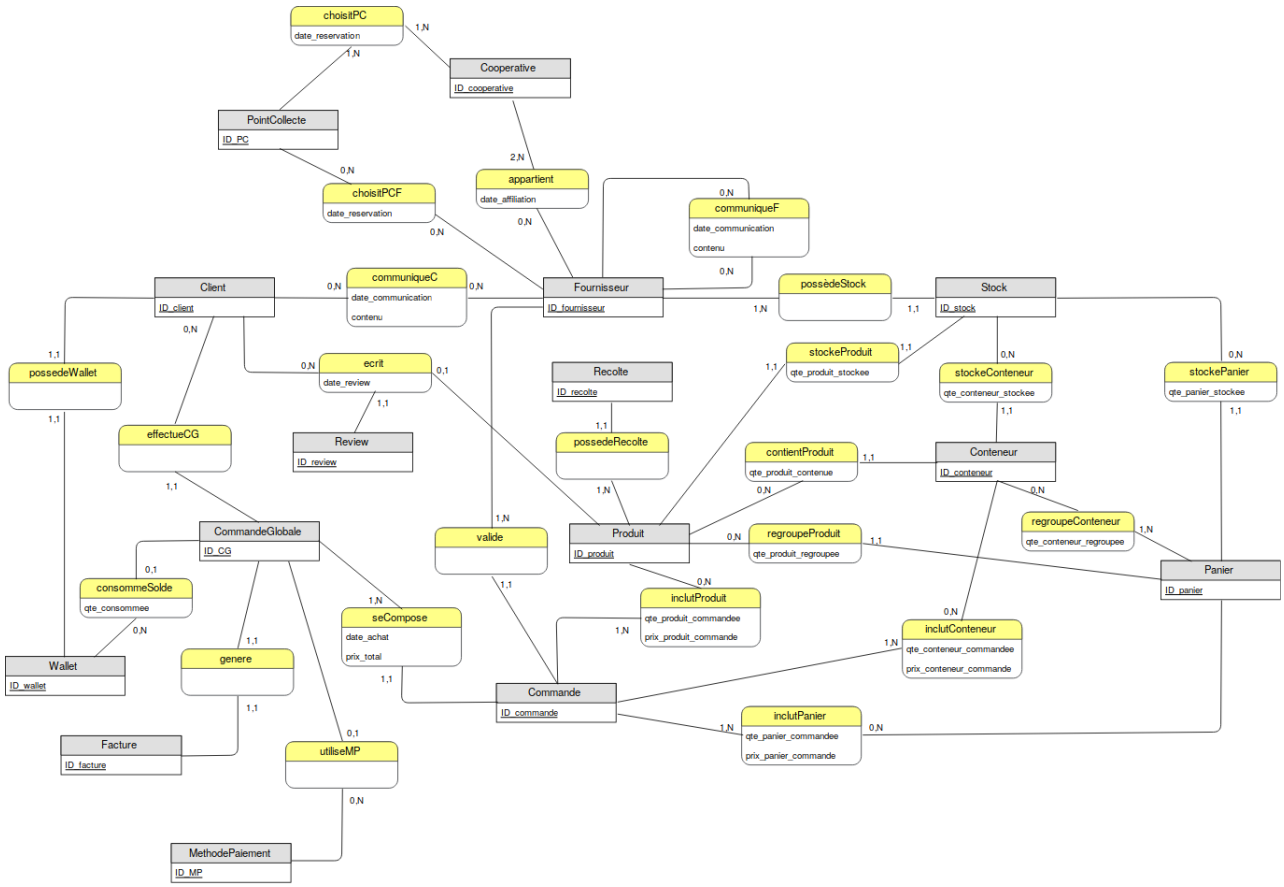

FIGURE 3.7 – Modèle Entité-Association

Chapitre 4

Conclusion

4.1 Difficultés survenues

Nous avons rencontré plusieurs difficultés pendant le développement du projet, notamment des difficultés liées à la maîtrise des outils nécessaires pour son développement, et que nous n'avons jamais utilisés auparavant (*React, JSX, Express* ...).

D'autre part, le temps que nous avions prévu d'accorder à la partie modélisation s'est révelé insuffisant. Nous avons donc été obligés d'y consacrer plus de temps pour avoir un résultat cohérent. Ceci a eu des répercussions négatives sur notre livrable final.

De plus, nous avons eu un souci avec le serveur d'hébergement du département informatique. En effet, le serveur hébérgea des versions anciennes de **Node** et **NPM** et nous n'avons pas accès à une base de données dediée à notre projet comme c'était indiqué sur l'espace projets du **SIF**. Ainsi, n'ayant pas les droits d'accès nécessaires pour rectifier ces problèmes, nous n'avons pas pu héberger notre site en ligne. Par conséquent, nous étions obligés de travailler sur notre application et de tester l'ensemble de ses fonctionnalités localement.

4.2 Perspectives

Outre les fonctionnalités que nous avons jugées essentielles au projet, nous avons pensé à ajouter des fonctionnalités supplémentaires une fois celles de base établies. En effet, plusieurs extensions sont possibles afin de proposer aux utilisateurs un choix plus diversifié et plus complet.

Dans ce cadre, nous avons prévu de donner aux clients la possibilité de régler des achats à l'aide d'une carte bancaire mais également à l'aide de

services de paiements en ligne tels que Paypal. En addition, nous pourrions également internationaliser le site et son concept en proposant des versions en plusieurs.

Enfin, nous nous sommes aperçus de plus en plus que le projet est devenu une source d'intérêt personnel, au point où nous souhaiterons fort probablement le poursuivre au niveau individuel au delà le cadre de ce module.

4.3 Apports personnels

Tout au long du projet, chaque difficulté rencontrée était une nouvelle expérience enrichissante. L'apprentissage de nouveaux concepts de programmation web et la maîtrise de nouvelles technologies associées, que nous n'avions guère manipulés au préalable, était un défi profondément apprécié. En particulier, nous avons eu la chance de profiter des richesses offertes par JavaScript dans différents contextes, ce qui a radicalement changé notre perception du langage.

Nous sommes convaincus que, jusqu'à présent, ce projet fût le plus intéressant, quoique le plus exigeant, parmi les projets auxquels nous avions participé. De plus, nous sommes bien ravis par les connaissances acquises qui se prouverons certainement utiles ultérieurement dans nos formations futures mais également dans nos démarches professionnelles.

Annexes

Annexe A

Snapshots du site

Dans ce chapitre, nous illustrons quelques snapshots de notre application web.

FIGURE A.1 – La page d'inscription

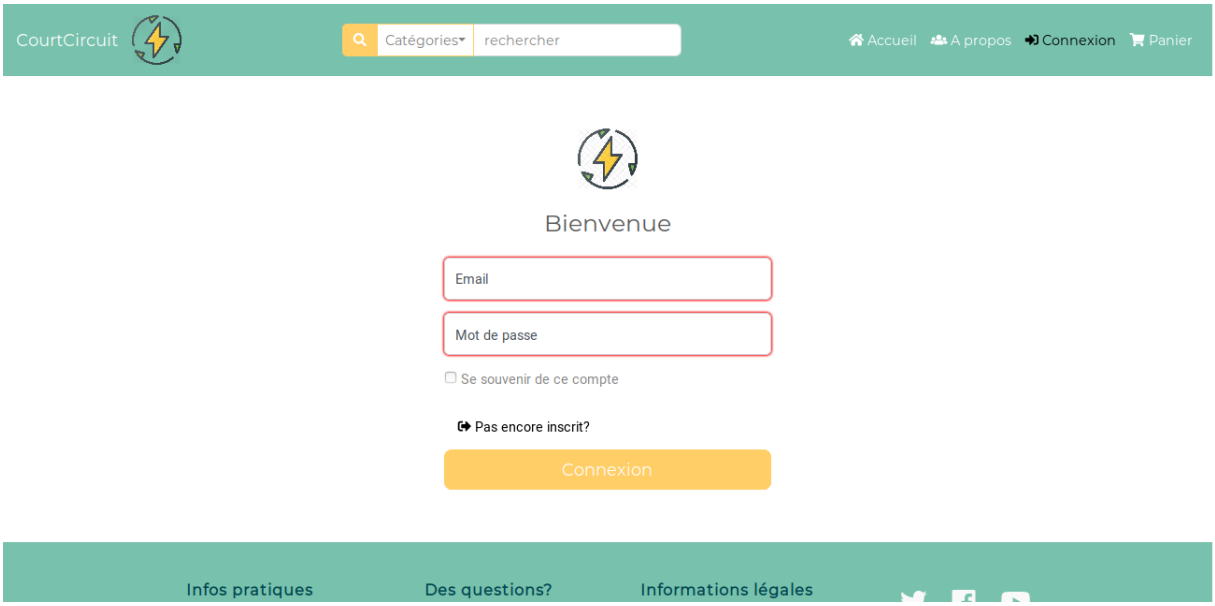

FIGURE A.2 – La page de connexion

FIGURE A.3 – La page d'un produit

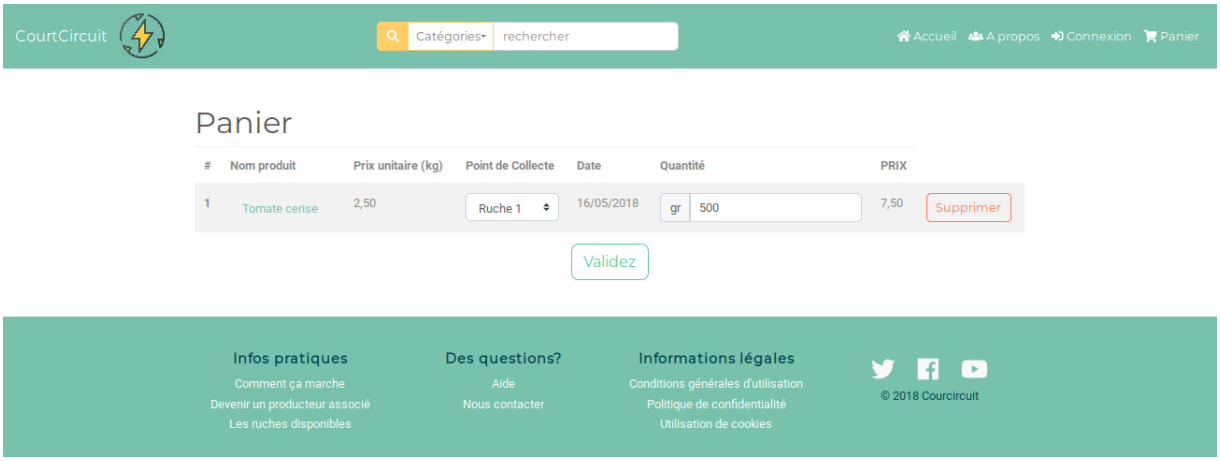

FIGURE A.4 – La page du chariot

FIGURE A.5 – La page d'un vendeur