

Alexis Boulangé
Élan Formation
Développeur web & web mobile



ZiCommerce

E-commerce de produits informatiques et électroniques

Dossier de synthèse

Période : Du 04/12/2020 au 14/10/2021

Table des matières

| | |
|--|----|
| Remerciements | 4 |
| I. Introduction | 5 |
| Présentation | 5 |
| Élan formation | 5 |
| Compétences couvertes..... | 6 |
| II. Présentation du projet..... | 7 |
| Histoire du projet | 7 |
| Cahier des charges | 7 |
| III. Conception du projet..... | 9 |
| Gestion du projet..... | 9 |
| Méthodologie appliquée | 10 |
| Choix technologiques | 12 |
| Modélisation des données..... | 15 |
| Maquettes de l'application | 17 |
| IV. Spécifications techniques | 20 |
| Responsive Design | 20 |
| UI, UX et référencement..... | 20 |
| RGPD..... | 22 |
| V. Le projet..... | 24 |
| Introduction de Laravel..... | 24 |
| Design Pattern | 24 |
| Composants Laravel | 25 |
| Création et structure de l'application | 26 |
| Sécurité..... | 28 |
| Jetstream | 33 |
| Multi Authentication..... | 34 |
| VI. Réalisation de l'application | 39 |
| CRUD..... | 39 |
| Panier en session..... | 42 |
| Vérification | 45 |
| Paiement avec l'API Stripe..... | 47 |
| VII. Rendu final..... | 51 |
| Axes d'améliorations | 51 |
| Conclusion | 51 |

Remerciements

Je tiens d'abord à remercier notre formateur Stéphane SMAIL pour sa pédagogie, sa patience et sa bienveillance qui nous a appris et suivi durant toute la formation, ainsi que les autres formateurs au sein d'Élan : Gilles MUESS, Mickaël MURMANN et Virgile GIBELLO.

Un remerciement aux intervenants professionnels qui nous ont apportés des conseils et appris de nouvelles choses dans le cadre de la formation.

Je souhaiterais ensuite remercier les cadres d'Applipro qui m'ont accueilli au sein de leur entreprise pour mon stage ainsi que M. Vrajolli pour ses conseils et son apprentissage durant la durée de celui-ci.

Enfin, je remercie mes camarades de formation pour la bonne ambiance, l'entraide, les conseils durant la formation qui nous a permis de se motiver les uns et les autres.

I. Introduction

Présentation

Alexis Boulangé, 25 ans, actuellement en formation de développeur web et web mobile au sein d'Élan Formation.

Détenteur d'un Bac S (Sciences de l'Ingénieur) j'ai poursuivi mes études en Faculté de Sciences en Maths, informatique, électronique et robotique. Puis j'ai travaillé (essentiellement en intérim) avant de rejoindre la formation.

Mon objectif pour la suite est d'obtenir le diplôme de développeur web et web mobile afin de pouvoir continuer mes études dans le domaine du développement.

Élan formation



Élan formation, c'est plus de 25 ans d'expérience dans les domaines de la Bureautique, la PAO, le Multimédia, d'internet, des Techniques de secrétariat. Ce sont des formations sur mesure et totalement individualisées. Possibilité de monter un dossier en partenariat avec des OPCA (Organismes Paritaires Collecteurs Agréés) et optimiser ainsi les recherches de financement. A ce titre, il dispose de locaux à Strasbourg, Sélestat, Haguenau, Saverne, Colmar, Mulhouse, Metz et Nancy.

La méthode pédagogique regroupe les points ci-mentionnés :

- Écouter et comprendre la demande précise de l'entreprise ou du stagiaire.
- Adapter une formation qui prenne en compte la singularité de l'apprenant.
- Guider le stagiaire en permanence grâce à un formateur.
- Anticiper ses attentes.
- Suivre l'évolution des acquis tout en avançant à son rythme.
- Valoriser la formation et certifier les compétences acquises.

Compétences couvertes

| N° Fiche AT | Activités types | N° Fiche CP | Compétences professionnelles |
|-------------------|---|-------------------|---|
| 1 | Développer la partie front-end d'une application web ou web mobile en intégrant les recommandations de sécurité | 1 | Maquetter une application |
| | | 2 | Réaliser une interface utilisateur web statique et adaptable |
| | | 3 | Développer une interface utilisateur web dynamique |
| | | 4 | Réaliser une interface utilisateur avec une solution de gestion de contenu ou e-commerce |
| 2 | Développer la partie back-end d'une application web ou web mobile en intégrant les recommandations de sécurité | 5 | Créer une base de données |
| | | 6 | Développer les composants d'accès aux données |
| | | 7 | Développer la partie back-end d'une application web ou web mobile |
| | | 8 | Elaborer et mettre en œuvre des composants dans une application de gestion de contenu ou e-commerce |

II. Présentation du projet

Histoire du projet

Ce projet d'e-commerce d'informatique et d'électronique m'a été proposé par mon entreprise de stage qui voulait me mettre au défi. L'entreprise avec laquelle j'ai effectué mon stage est une entreprise qui crée des applications principalement pour les commerces. Pour ce qui est du thème, ou du moins des produits proposés sur l'application j'avais le choix. J'ai choisi l'informatique et l'électronique car ce sont le type de produits que je connais le mieux ainsi que ceux que je « consomme » le plus. J'avais donc en tête plusieurs références pour mon travail de veille.

De plus, ayant envie d'apprendre de nouvelles choses, n'ayant jamais fait un e-commerce et étant curieux d'apprendre un nouveau Framework, j'ai décidé de faire mon projet en Laravel.

Après un travail de veille et de recherche, j'ai pu établir un cahier des charges pour une telle application.

Cahier des charges

Un cahier des charges a été mis en place pour avoir un suivi des fonctionnalités à mettre en place. Nous avons donc trois parties, une partie publique accessible à tous, une partie utilisateur en session et une partie administrateur. Pour réaliser cette application j'ai donc décidé de faire une « Multiple Authentication¹ » afin de bien séparer ma partie « frontend » (partie visiteur/utilisateur en session) de ma partie « backend » (partie administrateur).

Partie publique, afin de donner envie au client de s'inscrire et d'acheter :

- Avoir la possibilité de naviguer à travers l'application (sur ordinateur comme sur téléphone de manière totalement responsive) afin de pouvoir prendre connaissance des produits proposés.
- Trouver facilement un produit répertorié à l'aide de différentes catégories, de tags, de prix ou de couleurs.
- Être capable d'ajouter un produit à son panier en session.
- Pouvoir créer un compte utilisateur sécurisé et respectant le RGPD.

¹ Une « Multiple Authentication » ou « Multi Auth » permet à un administrateur et à un utilisateur de se connecter séparément ; nous avons deux tables séparées dans notre base de données, notre application contient deux formulaires de connexion différents ainsi que deux interfaces différentes.

Partie utilisateur :

- Création d'un tableau de bord, afin que celui-ci puisse accéder à ses données facilement afin de les modifier ou les supprimer.
- Possibilité d'ajouter des produits à sa liste de souhait.
- Permettre à l'utilisateur d'acheter les produits mis dans son panier.
- Pouvoir commenter et noter un produit acheté.
- Avoir la possibilité de renvoyer un produit.

Partie administrateur :

- Création d'un tableau de bord administrateur, afin que celui-ci puisse accéder et modifier ses informations.
- Possibilité de créer, lire, modifier ou supprimer (CRUD) les différents éléments de l'application tel que : les catégories, sous-catégories, sous sous-catégories et les produits.
- Être capable de mettre en avant les produits ; produit populaire, offre spéciale, occasion spéciale ou bonne affaire.
- Définir l'état du produit : c'est-à-dire définir si celui-ci sera afficher ou non (actif ou inactif).
- Pouvoir personnaliser les différents éléments de l'application, tel que le slider ou les images mis en avant.

Ainsi un administrateur aura la possibilité de créer, modifier, supprimer et voir lui-même les éléments (catégorie, sous-catégorie, sous sous-catégorie, produits, slider...) qu'il souhaite pour son application d'e-commerce. Il pourra aussi définir un statut actif/inactif s'il souhaite retirer un produit temporairement. Il pourra également appliquer un rabais ou mettre en avant un ou plusieurs produits particuliers.

L'expérience d'utilisateur quant à elle se veut simple et claire. Celui-ci doit pouvoir accéder aux différents produits rapidement et facilement afin de l'inciter à utiliser notre application et à commander (tel est le but premier d'un e-commerce). De la même façon l'application doit se montrer attractive, donc un niveau de design et d'ergonomie minimum est requis. Elle doit être responsive pour une meilleure accessibilité afin d'attirer plus de clients.

III. Conception du projet

Gestion du projet

Pour mon projet j'ai d'abord commencé par effectuer un **benchmark**, aussi appelé benchmarking, qui est un terme anglais pouvant être traduit par les mots **référence**, **étalon** ou **repère**. Elle vise à **étudier** et **analyser** d'autres entreprises pour optimiser ses propres techniques de gestion et modes d'organisation. En d'autres termes, « benchmarker » consiste à étudier ce que fait la concurrence. Ce peut-être un produit ou un service qui est à l'étude. L'objectif du benchmark est de **s'améliorer** et de **rivaliser** avec les meilleurs sur le marché.

L'idée du benchmark est en effet de **s'inspirer** d'autres pratiques pour rendre celles de l'entreprise plus efficaces. C'est également un outil très utilisé dans les domaines à forte innovation, qui nécessitent une **veille concurrentielle**² pointue.

La réalisation d'un benchmark permet ainsi :

- de définir les critères qui favorisent la performance et la réussite.
- d'identifier les meilleures entreprises dans votre domaine d'activité.
- d'avoir une vision claire du marché et de ses tendances.
- de maîtriser l'univers concurrentiel.
- de connaître le positionnement, les points forts et les points faibles de chacun de vos concurrents.
- d'effectuer une synthèse des bonnes pratiques.
- et de mettre en place un plan d'action au niveau de l'entreprise, pour s'approprier ces « *Best practices* » et optimiser la performance de l'application.

J'ai ensuite commencé une première application « test » afin d'apprendre et comprendre le fonctionnement du Framework Laravel. Pour cela j'ai utilisé plusieurs ressources telles que **Laracasts**, **Stack Overflow** et surtout la **documentation officielle de Laravel**. Puis j'ai commencé à établir mon **cahier des charges** à l'aide des notes que j'ai pris au fur et à mesure de mon benchmark. J'ai établi mon **MCD** (Modèle Conceptuel des Données) et le **maquettage** de mon application.

Enfin, j'ai établi un ordre de priorité des tâches à effectuer, avec une estimation du temps de conception pour ces tâches avant de commencer à réaliser mon application.

² La veille concurrentielle peut se définir comme la recherche, la collecte et l'analyse de diverses données issues du web concernant l'activité des autres entreprises présentes sur le même marché.

Méthodologie appliquée

Pour la gestion de ce projet, j'ai décidé d'appliquer une **méthode Agile**. Une méthode Agile est une **méthode de développement informatique** permettant de concevoir des logiciels en **impliquant au maximum le demandeur** (mon entreprise de stage), ce qui permet une grande réactivité à ses demandes. Les méthodes Agiles se veulent plus pragmatiques que les méthodes traditionnelles. Elles visent la satisfaction réelle du **besoin du client**, et non d'un contrat établi préalablement. La notion de méthode Agile est née à travers un manifeste signé par 17 personnalités (parmi lesquelles Ward Cunningham, l'inventeur du Wiki), créateurs de méthodes ou dirigeants de sociétés.

La méthode Agile appliquée pour la réalisation de mon projet est la méthode **MoSCoW**. La méthode MoSCoW est une technique de **priorisation** qui permet de prioriser les besoins et les exigences d'un projet. MoSCoW est l'acronyme des mots suivants :

- **M** pour « **must have if** » ; ce qui doit être fait.
- **S** pour « **should have this if at all possible** » ; ce qui serait bien d'être fait si c'est possible.
- **C** pour « **could have this if it does not affect anything else** » ; ce qu'il faudrait faire s'il n'y a pas d'impact avec d'autres demandes.
- **W** pour « **won't have this time but would like in the future** » ; à faire si nous avons le temps un jour.

Les « o » ayant été rajoutés à l'acronyme afin de le rendre prononçable.

De ce fait, j'ai défini quelles fonctionnalités étaient prioritaires pour mon application :

- Permettre à un **utilisateur** de *s'inscrire*, de *se connecter* et de *modifier* ou *supprimer* ses informations personnelles.
- Permettre à l'**utilisateur** de naviguer à travers les **différents produits**.
- Permettre à l'**utilisateur** d'*ajouter* un produit à son **panier**, effectuer un **paiement** et pouvoir donner son **avis**.

Ce qui devrait être bien de faire si cela est possible :

- Permettre à l'**utilisateur** d'*ajouter* des produits à sa **liste de souhait**.
- Améliorer l'**interface** et l'**expérience** de l'**utilisateur**.

Ce qu'il faudrait faire s'il n'y a pas d'impact avec d'autres demandes :

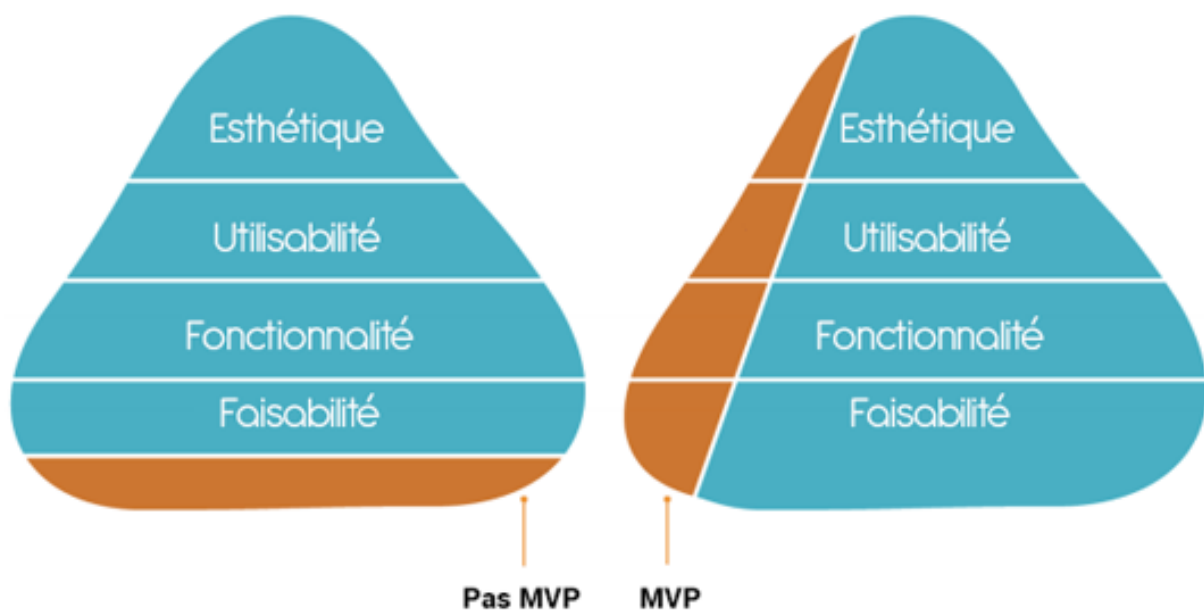
- Ajouter de l'ajax afin d'empêcher le rafraîchissement de certaines pages.

Enfin, ce qu'il serait à faire si nous avons le temps un jour :

- Ajouter du React sur les filtres (par exemple) afin de rendre notre **application** plus **dynamique**.
- Ajouter la **possibilité de zoomer** sur nos produits.
- Pouvoir avoir un aperçu de notre produit lorsque l'on change la couleur.

En plus de la méthode MoSCoW et toujours dans le cadre de la conception de mon application j'ai eu recours au concept de **MVP (Minimum Viable Product)**. Dans le cycle de développement d'un produit, le Minimum Viable Product correspond au lancement d'un **produit « test »** auprès de sa cible afin de **recueillir un maximum de retours** utilisateurs de façon à **valider** le projet et le produit avant de commencer un développement long et coûteux. Il permet à l'entreprise de s'assurer que le produit **répond à un besoin réel**. Le Minimum Viable Product n'est ni une esquisse ni une maquette, mais un produit **utilisable** et **fonctionnel**. Concentré sur sa ou ses **fonctionnalités essentielles**, le MVP doit fournir suffisamment de valeur aux utilisateurs, de sorte qu'il soit « viable ».

C'est une stratégie de **vérification** qui permet de **minimiser les risques** en s'attachant en premier lieu au **besoin utilisateur**. Voici un schéma illustrant le MVP :



Un exemple dans le cadre de mon projet de l'application du MVP :

- **Création** de l'espace utilisateur/administrateur.
- **Mise en place** du CRUD pour mes catégories (Create Read Update Delete) côté administrateur.
- **Affichage** dynamique des données créées par l'administrateur sur la partie utilisateur.

Choix technologiques

Base de données



Looping est un logiciel permettant la création de MCD et la génération du script SQL si l'on souhaite créer la base de données. J'ai utilisé Looping pour mon projet afin de créer mon MCD et permettre d'avoir une meilleure visibilité de mes données et ma base de données.



Laragon est un environnement de développement web complet sous Windows qui a pour atout d'être simple d'utilisation et ergonomique



MySQL (5.7) est un système de gestion de bases de données relationnelles. Il permet de stocker, manipuler, gérer les informations dans une base de données en garantissant la qualité, la pérennité et la confidentialité des informations.

Back-end



PHP : Hypertext Preprocessor (7.4), plus connu sous son sigle PHP est un langage de programmation libre. Il est utilisé côté serveur afin de produire des pages Web dynamiques. PHP est un langage impératif orienté objet³.



Laravel 8 est un Framework web open-source écrit en PHP respectant le principe MVC (Modèle-Vue-Controller) et est entièrement développé en POO. Nous utiliserons également son moteur de template : Blade ainsi que son ORM (Object Relational Mapping) : Eloquent.

³ La programmation orientée objet (POO), est un paradigme de programmation informatique. Elle consiste en la définition et l'interaction de briques logicielles appelées objets ; un objet représente un concept, une idée ou toute entité du monde physique (exemple : une voiture, une personne). Il possède une structure interne et un comportement, et il sait interagir avec ses pairs. Il s'agit donc de représenter ces objets et leurs relations ; l'interaction entre les objets via leurs relations permet de concevoir et réaliser les fonctionnalités attendues, de mieux résoudre le ou les problèmes. Dès lors, l'étape de modélisation revêt une importance majeure et nécessaire pour la POO. C'est elle qui permet de transcrire les éléments du réel sous forme virtuelle.

Front-end



HTML 5, pour HyperText Markup Language 5 est un langage de balisage et de sémantique. Il sert principalement à structurer son contenu logiquement grâce à des balises qui ont une signification sémantique. C'est un langage statique.



CSS 3, pour Cascading Style Sheets est, un langage de style qui sert au placement et au design d'une application. Il permet également de faire des animations.



JavaScript ES6 est un langage de programmation exécuté principalement côté client par le navigateur web (il peut néanmoins être exécuté côté serveur, en back-end, grâce à certaines technologies, notamment Node.js). Il n'est pas orienté objet à proprement parler mais plutôt orienté objet à prototype. La différence entre les deux est la façon dont nos objets sont construits et manipulés. Javascript sert principalement à rendre nos pages HTML/CSS dynamiques, et permet également de faire des animations. Il intègre également AJAX (Asynchronous JavaScript And XML), qui sert à traiter des requêtes http, de version asynchrone, et permet de récupérer des données et de les afficher sans rafraichissement de la page internet.



Ajax, pour Asynchronous JavaScript and XML permet de réaliser de rapides mises à jour du contenu d'une page Web, sans qu'elles nécessitent le moindre rechargement visible par l'utilisateur de la page Web.



jQuery est une librairie JavaScript libre et multiplateforme créée pour faciliter l'écriture de scripts JavaScript.



Bootstrap 4 est un Framework d'interface mettant à disposition une collection d'outils utiles à la création du design d'applications web qui contient des codes HTML, CSS, des formulaires, boutons, outils de navigation et autres éléments interactifs, ainsi que des extensions JavaScript en option.

Autres outils



Composer est un logiciel gestionnaire de dépendances libre écrit en PHP. Il permet à ses utilisateurs de déclarer et d'installer les bibliothèques dont le projet principal a besoin.



Laravel Jetstream est un « starter kit » proposé pour Laravel, celui-ci implémente un système d'Authentification complet.



GitHub est un service web d'hébergement et de gestion de développement de logiciels, utilisant le logiciel de gestion de versions Git. Le site assure un contrôle d'accès et des fonctionnalités destinées à la collaboration comme le suivi des bugs, les demandes de fonctionnalités, la gestion de tâches et un wiki pour chaque projet.



Draw.io est une application Web qui permet de faire des schémas et du dessin vectoriel mais aussi des maquettes de site Web.

Modélisation des données

Mon cahier des charges étant mis en place je me suis attelé à la **modélisation des données**.

La **modélisation des données** (data modeling en anglais) est un processus de **description de la structure**, des **associations**, des **relations** et des **contraintes** relatives aux données disponibles. Elle sert à établir des **normes**, à coder des **règles de gestion** (modèles) et des **data** dans l'organisation.

Pour mon projet, j'ai décidé d'utiliser la méthode **MERISE**. Le but de cette méthode est d'arriver à concevoir un système d'information. La méthode **MERISE** est basée sur la **séparation des données** et des **traitements** à effectuer en plusieurs **modèles conceptuels** et **physiques**. La **séparation des données** et des **traitements** assure une longévité au modèle. En effet, l'**agencement des données** n'a pas à être souvent remanié, tandis que les traitements le sont plus fréquemment.

Le **modèle conceptuel des données (MCD)**, qui est un élément de la méthode **MERISE** a été créé avec **Looping**.

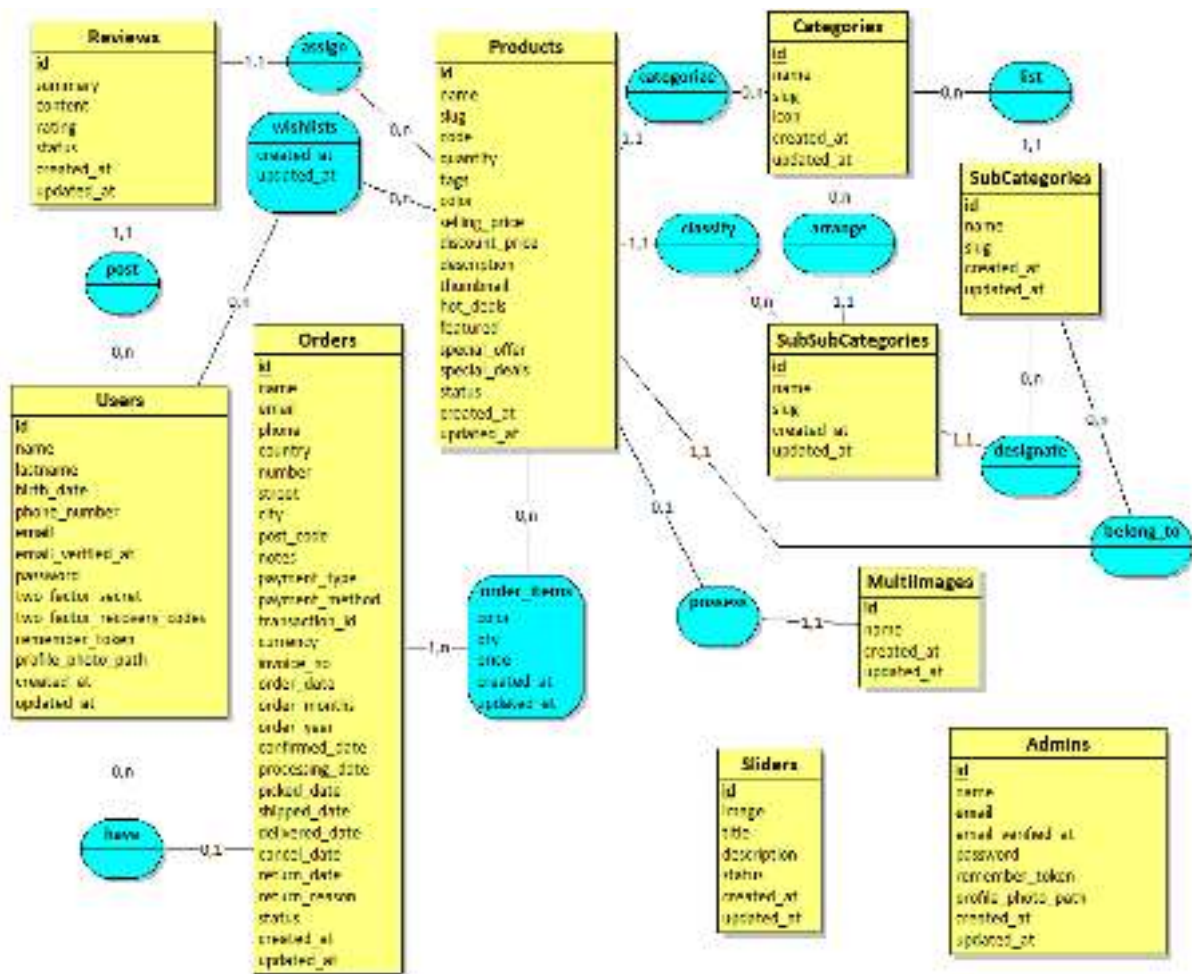
Le **MCD** permet d'établir une **représentation claire** des données du système d'information⁴. Il s'agit donc d'une **représentation des données**, facilement **compréhensible**, permettant de **décrire** le système d'information à l'aide d'**entités**. Les éléments utilisés pour la formalisation d'un MCD sont les suivants :

- **Entité** : définition d'entité (objet physique ou abstraits) ayant des caractéristiques comparables.
- **Relation** : définition d'une association liant plusieurs entités. Signification d'un lien entre deux ou plusieurs types d'objets.
- **Propriété** : définition d'une caractéristique d'un objet ou d'une association. Une propriété est elle-même caractérisé par un type (String, Integer...) et une longueur. L'ensemble des propriétés du MCD compose le **dictionnaire des données**⁵.
- **Identifiant** : Propriété ou concaténation de propriétés permettant de distinguer une entité parmi toutes les autres dans une entité.
- **Cardinalité** : Couple de nombres qui exprime la participation des occurrences d'une entité à une association.

Looping permet de **schématiser** la **structure des données**, de la couche la plus abstraite jusqu'au modèle physique : en générant le script contenant les requêtes pour créer les **tables** dans la base de données. Pour ce projet, la **base de données relationnelles** de l'application a été créée à partir des entités, à l'aide de l'**ORM Eloquent**. En jaune, nos entités. En bleu, nos associations.

⁴ Le système d'information (SI) est un ensemble organisé de ressources qui permet de collecter, stocker, traiter et distribuer de l'information.

⁵ Un dictionnaire des données est une collection de métadonnées ou de données de référence nécessaire à la conception d'une base de données relationnelle.



Modèle Conceptuel des Données

D'après ce schéma, nous pouvons voir que l'entité « **Products** » qui est notre produit est au centre de l'application. Ce produit appartient à une sous sous-catégorie, une sous-catégorie et à une catégorie. Le produit possède un « **multimages** », qui lui permet de posséder plusieurs images. Nous pouvons également observer qu'un produit peut se faire juger (entité « **reviews** ») par les différents utilisateurs en session. Un utilisateur authentifié peut ajouter un produit à sa liste de souhait (association « **wishlists** »). Enfin, un utilisateur peut passer une commande (entité « **orders** »), qui peut contenir un ou plusieurs produits et un produit quant à lui peut faire partie de plusieurs commandes différentes (association « **order_items** »).

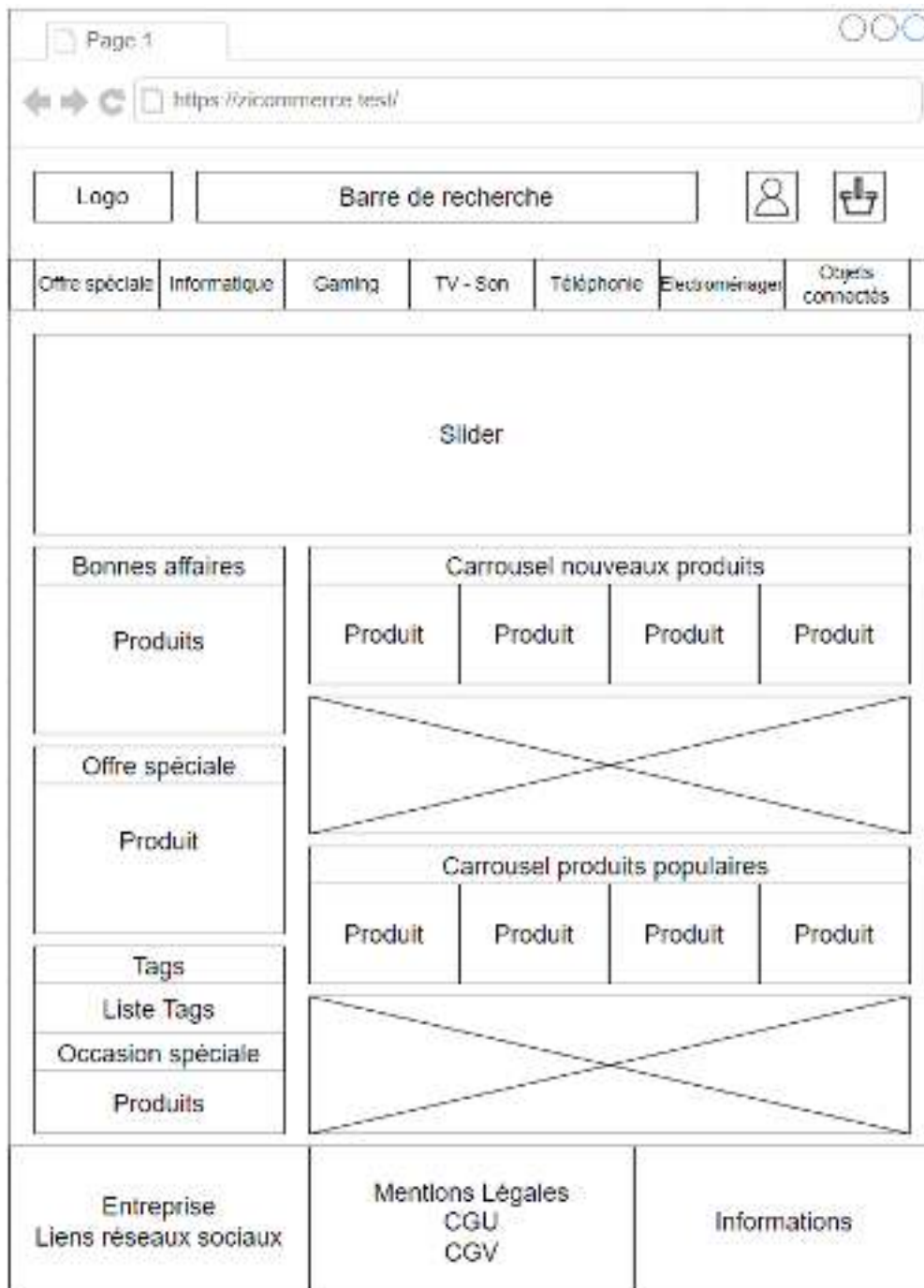
Maquettes de l'application

Après avoir modéliser mon application, je me suis lancé sur le « **maquettage** ». Une **maquette** de site web correspond à une **esquisse**, voire un **prototype** d'un site internet en création. Le processus de **création graphique** qui donne lieu à la création de maquettes graphiques est appelé « **maquettage** ».

Créer une maquette graphique permet de travailler à partir d'une **charte graphique** précise, une intégration des contenus rédigés de manière ergonomique et fluide. Elle est efficace pour **visualiser** le projet plus **facilement** et procéder à des corrections de manière progressive. Il existe différents types de maquette :

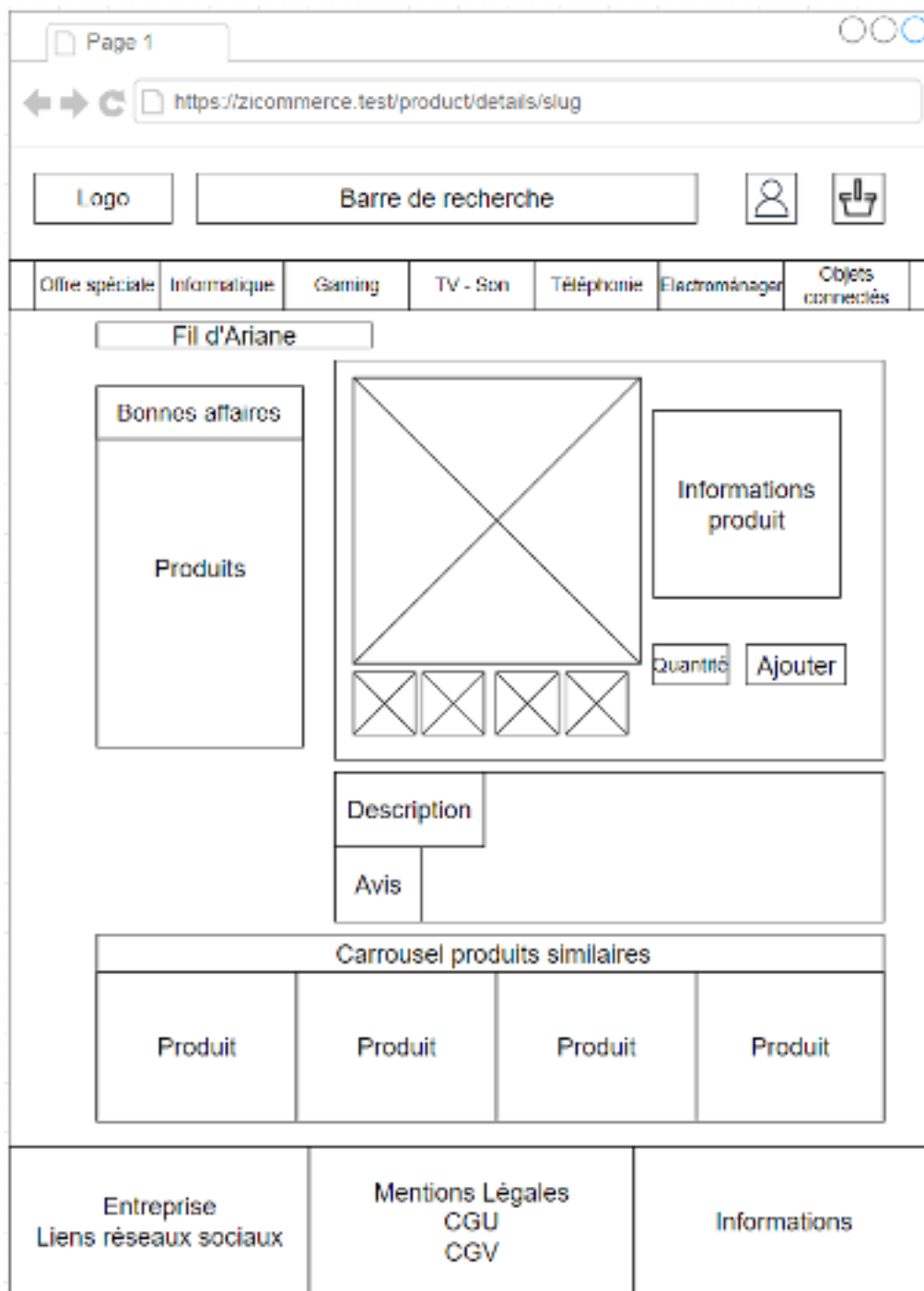
- **Zoning** : technique consistant à **schématiser** une page Web à l'aide de **blocs** ou boîtes, dans le but de **montrer** les grandes **fonctionnalités** et les **zones principales** du contenu.
- **Wireframe** : technique qui s'appuie sur le zoning réalisé auparavant, et permet d'**indiquer** le contenu présent dans chaque bloc de la page Web et de **structurer** l'interface. Aucun design n'est fait sur cette étape de « wireframing », son objectif étant avant tout fonctionnel.
- **Maquette** : permet d'intégrer la **dimension interactive**. Ainsi, des liens peuvent être faits afin de montrer la navigation entre les différentes pages (ex : **simuler** la connexion à un compte utilisateur, simuler un carrousel d'image...).
- **Prototype : application fonctionnelle** qui se focalise sur le fond et permet de déterminer avec quelles technologies les informations seront affichées.
-

Voici donc, quelques exemples de « zoning » de mon application réalisés sur draw.io.



Page d'accueil de l'application

Situé tout en haut, nous avons notre **<header>** avec notre logo, notre barre de recherche, le profil utilisateur et le panier. Nous retrouvons ensuite une **<nav>** affichant nos catégories, un slider composé de « flyers ». Occupant majoritairement la place, nous avons le **<main>** qui contient des carrousels et des images. A gauche de celui-ci nous avons un **<aside>** regroupant plusieurs **<section>**. Enfin en bas de page, le **<footer>**, séparé en 3 parties. Avec nos informations et les liens vers les réseaux sociaux de l'entreprise à gauche. Les mentions légales, les conditions générales d'utilisation et les conditions générales de vente au centre. Et diverses informations concernant l'application à droite.



Page détail d'un produit

Dans cette maquette nous retrouvons notre **<header>**, notre **<nav>** ainsi que notre **<footer>**. Le contenu de la page quant à lui change en grande partie. Nous retrouvons toujours une partie **<main>** contenant notre produit ainsi que ses informations, la possibilité de l'ajouter au panier et de voir ou de rédiger un avis. Nous retrouvons également, à gauche, un **<aside>** présentant les « bonnes affaires ». Enfin, en bas nous avons une **<section>** contenant un carrousel ayant pour but d'afficher des produits similaires.

IV. Spécifications techniques

Responsive Design

Le **Responsive Design** est une technique de conception d'interface digitale qui fait en sorte que l'affichage d'une quelconque page d'un site s'**adapte** de façon automatique à la **taille de l'écran** du terminal qui le lit. Pour ce faire, nous utilisons un module de CSS3 appelé « **media queries** ».

Les **media queries** permettent de modifier l'apparence de l'application en fonction du **type d'appareil** (impression ou écran) et de ses caractéristiques (la **résolution d'écran** ou la largeur de la zone d'affichage). Par exemple, dans une application nous pouvons **changer la disposition d'éléments** en « ligne » : en « colonne », nous pouvons également changer la forme de l'élément ou simplement le masquer de façon à avoir un rendu plus propre.

L'utilisation optimale sur mobile est inscrite au cahier des charges, pour une partie de l'application : dans un premier temps la priorité a été donnée à la partie de l'**interface utilisateur** pour l'affichage des produits, sa navigation sur l'application ou son inscription.

UI, UX et référencement

UI est l'acronyme d'**User Interface** (Interface Utilisateur en français). Une interface utilisateur permet à un humain d'interagir avec une machine. Dans le cas d'un ordinateur, l'interface utilisateur est composée des éléments qui permettent l'**interaction**, à savoir : la souris, le clavier, l'écran... Dans le cas d'une application web, l'interface utilisateur (UI) est une **interface web** matérialisée par une **interface graphique** que l'on peut manipuler avec un navigateur web.

Le **design de l'UI** prend en compte l'**identité visuelle**, elle est constituée de tous les éléments graphiques contenus dans la charte graphique d'une entreprise ou d'une marque (logo, couleur, typographie, pictogrammes, ...).

L'**UI** est donc axée sur le **design** d'une interface graphique en s'appuyant sur des normes techniques permettant d'organiser les éléments textuels et graphiques d'une page web. Elle conditionne également la manière dont l'application ou le site internet sollicite les interactions des utilisateurs. Seulement, l'UI seule ne permet pas de répondre à toutes les attentes et problématiques des internautes, c'est là qu'intervient l'**expérience utilisateur (UX)**.

Le design d'une interface web (UI) doit prendre en compte l'expérience utilisateur (UX) afin de proposer la **meilleure expérience** possible en termes de navigation.

Le terme **UX** vient d'**user experience** (expérience utilisateur). Le travail de l'UX designer consiste donc à concevoir une **interface accessible** et **facile à prendre en main** pour **tout type de support**. L'UX se charge donc de l'**architecture** et de l'**ergonomie** de l'application. Celui-ci prend en compte l'UI et concerne : les contenus textes et visuels, l'analyse des données, les tests utilisateurs, les fonctionnalités, l'architecture du site web et la satisfaction client.

Pour une expérience utilisateur optimale, un site internet doit être :

- Visible sur les moteurs de recherche grâce à une conception **SEO Friendly**⁶.
- **Compatible** pour un affichage optimal sur toutes les tailles d'écran.
- **Conforme** à l'image de l'entreprise et **rassurant**.
- **Efficace** en facilitant la navigation et la recherche d'information.
- **Disponible** 24 heures/24 et sans erreurs.

Dans l'univers du **SEO**, le terme **référencement** regroupe les différentes techniques utilisées pour **améliorer** la position d'un site internet dans les pages de résultats affichées par les moteurs de recherche en réponse aux requêtes des internautes.

Pour cela nous avons deux possibilités :

- Faire de la publicité sur les moteurs de recherche (aussi appelé « **SEA** » pour **Search Engine Advertising**) afin d'être mis en avant. Cette technique répond surtout à des enjeux sur le **court terme** et est **payante**.
- Optimiser le **SEO (Search Engine Optimization)** de l'application, c'est-à-dire le **référencement naturel** de notre application, ce qui renforce la **notoriété** de la marque et la **confiance** des utilisateurs.

Afin d'améliorer le référencement naturel de mon application (qui est essentiel pour un e-commerce) j'ai mis en place plusieurs solutions :

- Les **balises META**, tel que la balise **<title>**, le contenu de cette balise s'affiche dans l'onglet de notre page et permet à l'utilisateur d'avoir une idée du contenu de la page. Elles servent également à insérer dans la partie **HEAD** des **informations** sur la page Web, ce contenu n'est pas visible dans le navigateur mais est lu par les moteurs de recherche.
- L'utilisation de **balises sémantiques** (**<header>**, **<main>**, **<aside>**, **<footer>**... ou encore les balises de titres comme **<h1>**, **<h2>**...).
- Mise en valeur des **mots clés** et **mise en forme des caractères** avec des balises comme **<h1>**, ****, ****...
- Mots clés dans les textes d'ancres des liens
- **Interconnexion** des pages, c'est-à-dire connexion de toutes les pages entre elles de façon que les robots des moteurs puissent toutes les indexer en suivant les liens trouvés dans le code HTML.
- Mise en place d'un « **Fil d'Ariane** », qui est un **chemin de navigation** qui permet à l'utilisateur de se repérer plus facilement.
- L'**optimisation des images**, avec des **mots-clés** dans le nom du fichier et la mise en place d'un texte alternatif avec la balise **<alt>**.
- Mise en place de **slugs**⁷ lors de la création d'une catégorie ou d'un produit.

⁶ L'expression SEO Friendly caractérise un site internet naturellement optimisé pour les moteurs de recherche. Un site internet SEO Friendly est un site conçu pour répondre aux préconisations des moteurs de recherche. Tant au niveau de sa structure que de ses contenus.

⁷ Le terme « slug » se réfère à la partie finale d'une URL, son rôle permet d'identifier un article/une page en particulier ainsi que d'indiquer aux moteurs de recherche le contenu de celle-ci.

RGPD

Le **Règlement Général sur la Protection des Données (RGPD)** a été mis en place afin d'encadrer la mise en œuvre des **traitements des données à caractère personnel** sur le territoire européen. Le respect du RGPD est **obligatoire**, et le développeur web s'expose aux sanctions de la CNIL⁸ au même titre que son client et les sous-traitants de celui-ci. La notion de données « personnelles » est à comprendre au sens large, comme « ***toute information se rapportant à une personne physique identifiée ou identifiable*** », que la personne puisse être identifiée à l'aide d'une seule donnée ou en croisant plusieurs informations. Les grands principes des règles de protection des données personnelles sont les suivants :

- **Finalité** : les données des personnes physiques ne peuvent être enregistrées et utilisées que dans un **but bien précis, légal et légitime**.

- **Proportionnalité et pertinence** : les données stockées et utilisées doivent être **pertinentes et strictement nécessaires**.

- **Durée de conservation limitée** : la durée de conservation doit être fixée, en fonction du type des données et de la finalité du fichier, de manière raisonnable.

- **Sécurité et de confidentialité** : le responsable du fichier garantit la sécurité et la confidentialité des informations, et doit veiller à ce que seules les personnes autorisées aient accès à ces informations.

- **Droit des personnes** : l'utilisateur a le **droit de consulter, éditer et supprimer ses données**. La structure qui récolte et traite les données est tenue de désigner une personne chargée de la protection des données.

Lors de son enregistrement l'utilisateur entre ses données personnelles tel que son nom, son prénom, son numéro de téléphone, sa date de naissance (pour vérifier sa majorité), son adresse électronique... Toutes ces informations sont utilisées afin de pouvoir effectuer des commandes. De plus lors de sa commande l'utilisateur doit entrer une adresse d'habitation afin de recevoir sa marchandise. Ces informations sont donc utilisées dans le cadre d'une application e-commerce.

⁸ Commission Nationale de l'Informatique et des Libertés

Mentions d'information

Informations personnelles de l'utilisateur inscrit stockées en base de données.

| | |
|----------------------------|---|
| Adresse électronique | Permet à l'utilisateur inscrit de se connecter à l'application, et à l'application de distinguer les réinscriptions des nouvelles inscriptions |
| Mot de passe | Le mot de passe est hashé , et celui-ci permet à l'utilisateur inscrit de se connecter |
| Prénom/Nom | Permet d' identifier la personne lors d'une livraison de commande |
| Date de naissance | Permet d' empêcher une personne mineure de s'inscrire sur le site et d'ainsi effectué un paiement contre le gré de ses parents |
| Numéro de téléphone | Permet à l'administrateur de contacter l'utilisateur en cas de problème. Peut également servir pour une double authentification |
| Adresse, code postal, pays | Permet de connaître l'adresse de livraison pour la commande passée par l'utilisateur |

Affichage des données dans l'application

Le prénom, la première lettre du nom de famille ainsi que l'image servant de photo de profil⁹ seront affichés pour identifier un utilisateur lorsque celui-ci créera une « **review** » d'un produit. Cette review comporte un titre, un commentaire, une note sur 5 et une date de création ou de modification. En plus de ne **pas être obligatoire**, l'utilisateur aura la possibilité de modifier ou supprimer sa review.

Droits des utilisateurs

L'utilisateur a le **droit** de **pouvoir vérifier**, **corriger** et **supprimer** ses données. Pour ce faire, il est prévu d'afficher pour un utilisateur toutes les données qui le concernent dans une page de profil avec un formulaire, lui permettant ainsi d'avoir le **contrôle sur ses données personnelles**.

⁹ Totalelement facultative.

V. Le projet

Introduction de Laravel

Comme énoncé dans « choix technologiques » : **Laravel** est un Framework web open-source écrit en PHP respectant le principe MVC (Modèle-Vue-Controller) et est entièrement développé en POO.

Un **Framework** (cadre de travail en français) désigne un **ensemble cohérent de composants logiciels structurels**, qui sert à créer les **fondations** ainsi que les grandes lignes de tout ou d'une partie d'une application. Son but est de **simplifier** le travail des développeurs, en leur offrant une **architecture** « *prête à l'emploi* » et qui permet de ne pas repartir de zéro à chaque nouveau projet.

L'un des avantages les plus importants du choix de Laravel pour le développement d'une application Web réside dans ses capacités à fournir une **sécurité de haut niveau**. De plus Laravel dispose d'un énorme **choix de composants et de bibliothèques** rendant la création d'une application plus **rapide** et plus **simple**.

Design Pattern

Dans la conception d'applications orientées objet, les **design patterns** (en français modèles, motifs ou encore patrons de conception) constituent un **ensemble de bonnes pratiques**, partagées comme des **schémas de conception**. Il s'agit de définir des modèles de **composants fonctionnels, élaborés** à partir de cas typiques. Lorsque la solution à un problème récurrent est suffisamment éprouvée, on peut la partager comme un modèle. Le design pattern est aussi utile au développeur qu'au designer ; en architecture logicielle le modèle sert à décomposer le logiciel en **éléments plus simples**, de manière **organisationnelle**.

L'architecture de Laravel se base sur le **Modèle-Vue-Contrôleur** (ou MVC ou Model-View-Controller) qui est un patron d'architecture logicielle. Il est utilisé pour **diviser de manière logique** les éléments de l'application en trois groupes de composants et leur interaction : **modèle**, **vue** et **contrôleur**.

- Le **modèle** contient la structure des données de l'application et la logique en rapport avec ces données (lecture, validation et enregistrement).
- La **vue** représente l'interface utilisateur, ce avec quoi il interagit. Elle n'effectue aucun traitement, elle se contente d'afficher les données que lui fournit le contrôleur. Il peut tout à fait y avoir plusieurs vues qui présentent les données d'un même modèle.
- Le **contrôleur** gère les actions de l'utilisateur envoyées par la vue : il reçoit la demande, récupère les données auprès du modèle, qu'il traite et éventuellement qu'il modifie avant de les renvoyer à la vue.

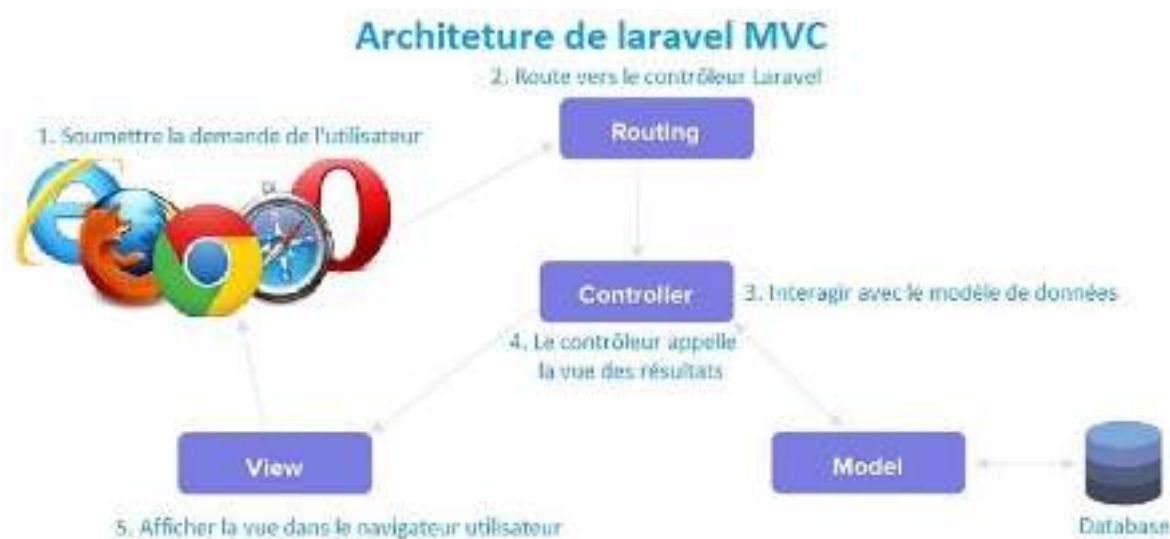


Schéma illustrant le système de MVC avec le routing de Laravel

Composants Laravel

Laravel est un Framework qui utilise des composants d'autre source, dont Symfony. A l'intérieur il est possible de trouver :

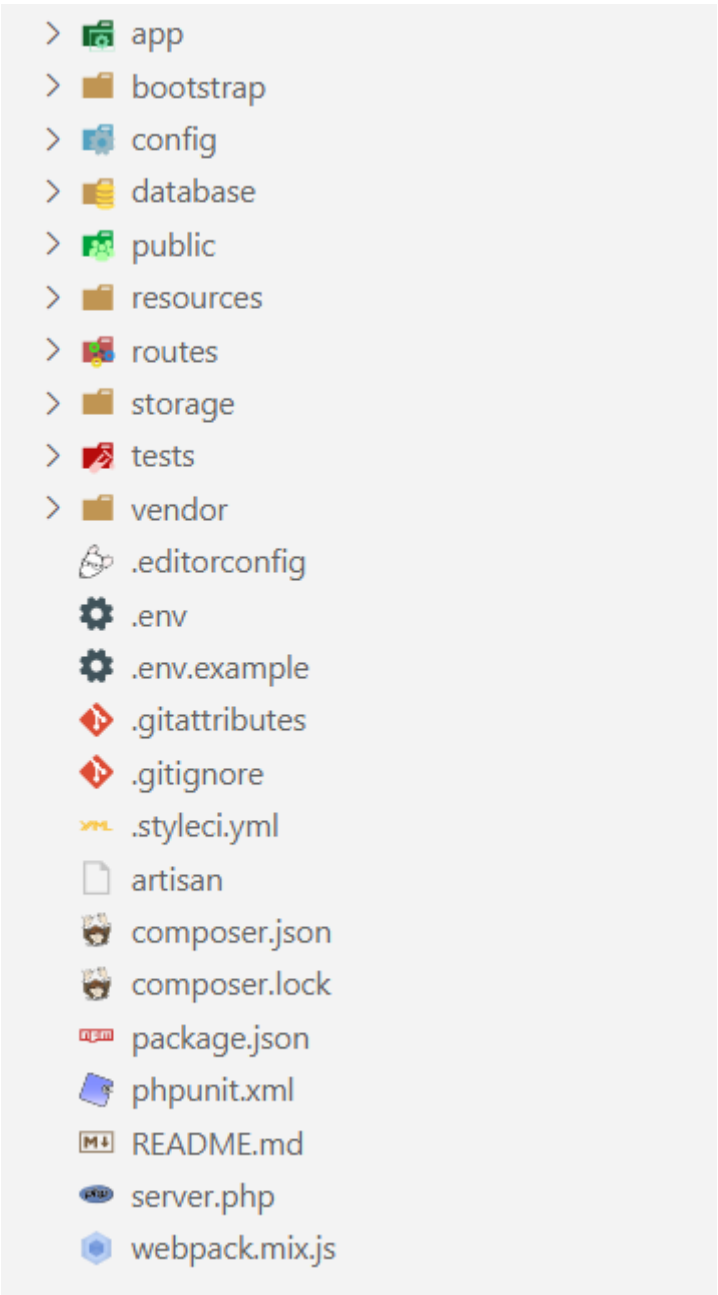
- Un **système de routage** perfectionné (RESTFUL et ressources).
- Un **créateur de requêtes SQL** et un **ORM** performants (Eloquent).
- Un **moteur de template** efficace (Blade).
- Un **système d'authentification** pour les connexions.
- Un **système de validation**.
- Un **système de pagination**.
- Un **système de migration** pour les bases de données.
- Un **système d'envoi de courriels**.
- Un **système de cache**.
- Un **système d'autorisations** (middleware)
- Une **gestion des sessions**...
- Et bien d'autres choses...

Création et structure de l'application

Il existe plusieurs façons d'installer Laravel, la plus courante est de passer par **Composer**. **Composer** est un **gestionnaire de dépendance** écrit en PHP, celui-ci se travail en ligne de commande. L'un des avantages de Composer et des gestionnaires de dépendance est de pouvoir gérer les mises à jour des composants. Pour la création de mon projet j'ai donc utilisé Composer.

```
composer create-project laravel/laravel ZiCommerce
```

Par cette simple ligne de commande entrée dans ma console, l'architecture de mon projet Laravel était créée. Une fois notre projet créé, l'organisation des dossiers se visualise ainsi :



- > app
- > bootstrap
- > config
- > database
- > public
- > resources
- > routes
- > storage
- > tests
- > vendor
- .editorconfig
- .env
- .env.example
- .gitattributes
- .gitignore
- .styleci.yml
- artisan
- composer.json
- composer.lock
- package.json
- phpunit.xml
- README.md
- server.php
- webpack.mix.js

Le dossier App, dans lequel on travaillera le plus souvent, contient cinq dossiers :

- **Console** : toutes les commandes en mode console.
- **Exceptions** : si besoin de créer des exceptions personnalisées.
- **Http** : tout ce qui concerne la communication : contrôleurs, middlewares (il y a 8 middlewares de base qui servent à filtrer les requêtes HTTP) et le kernel.
- **Providers** : tous les fournisseurs de services (providers), il y en a déjà 5 au départ. Les providers servent à initialiser les composants.
- **Models** : avec un modèle qui concerne les utilisateurs pour la base de données.

On retrouve ensuite :

- **Bootstrap** : scripts d'initialisation de Laravel pour le chargement automatique des classes, la fixation de l'environnement et des chemins, et pour le démarrage de l'application.
- **Config** : toutes les configurations : applications, authentications, caches, bases de données, espaces de noms, courriels, systèmes de fichier, session...
- **Database** : migrations et populations.
- **Public** : tout ce qui doit apparaître dans le dossier public du site : images, CSS, scripts...
- **Ressources** : vues, fichiers de langage et assets.
- **Routes** : la gestion des URLs d'entrée de l'application.
- **Storage** : données temporaires de l'application : vues compilées, caches, clés de session...
- **Tests** : fichiers de tests,
- **Vendor** : tous les composants de Laravel et de ses dépendances (créés par Composer),

Il y a un certain nombre de fichiers dans la racine dont voici les principaux :

- **Artisan** : outil en ligne de Laravel pour des tâches de gestion,
- **Composer.json** : fichier de référence de composer,
- **Package.json** : fichier de référence de npm pour les assets,
- **Phpunit.xml** : fichier de configuration de phpunit (pour les tests unitaires),
- **.env** : fichier pour spécifier l'environnement d'exécution.

Sécurité

Middleware

Les **middlewares** sont chargés de **filtrer** les requêtes HTTP¹⁰ qui arrivent dans l'application, ainsi que celles qui en partent (beaucoup moins utilisé). Le cas le plus classique est celui qui concerne la vérification de l'authentification d'un utilisateur pour qu'il puisse **accéder** à certaines ressources. On peut aussi utiliser un **middleware** par exemple pour **démarrer la gestion des sessions**.

On peut avoir en fait plusieurs **middlewares**, chacun effectue son traitement et transmet la requête ou la réponse au suivant. Donc dès qu'il y a un traitement à faire à l'arrivée des requêtes (ou à leur départ) un middleware est tout indiqué.

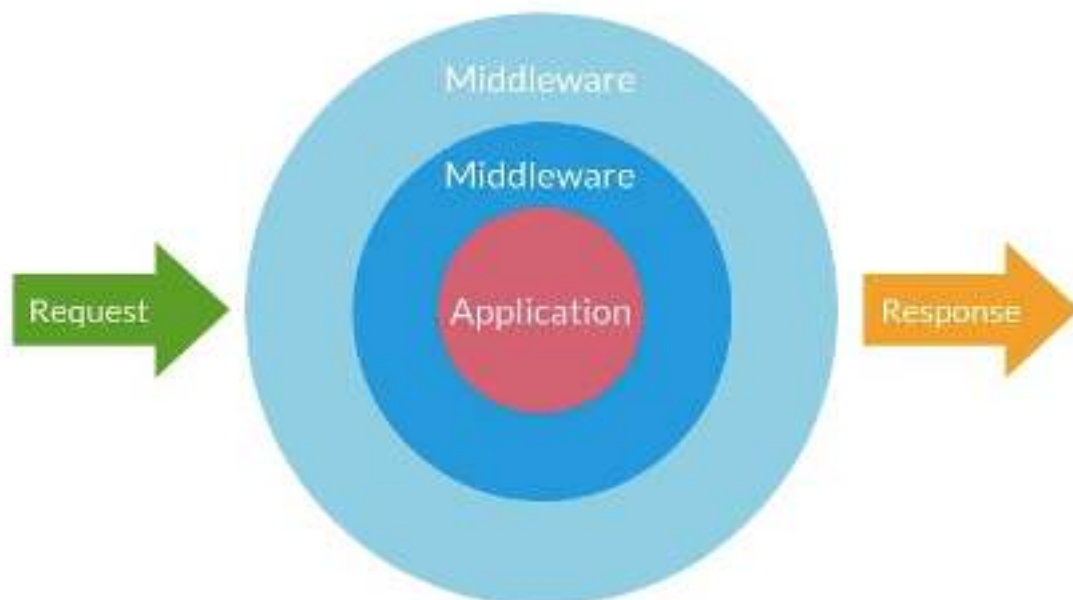


Schéma illustrant le fonctionnement des middlewares

Dans le cadre de mon application, j'ai créé/modifié des middlewares afin de **sécuriser** l'accès à certaines de mes routes et à rediriger tout utilisateur n'ayant pas les droits d'accéder à cette route.

¹⁰ HyperText Transfer Protocol est un protocole de communication entre un client et un serveur pour le « World Wide Web ».

Faible CSRF

La **faible CSRF** (Cross-Site Request Forgery) est une **attaque** qui consiste à faire exécuter à un utilisateur une requête http à son insu sans qu'il ne s'en aperçoive. Elle peut être réalisée en ajoutant des paramètres dans des URLs, par exemple sur une image qui ne s'affichera pas mais exécutera un script malveillant.

Parmi les moyens de prévention, on peut demander à l'utilisateur de confirmer les actions critiques, demander l'ancien mot de passe au moment de changer celui-ci ou l'adresse mél de connexion, ou encore utiliser des **jetons de validité** dans les formulaires.

Pour s'en prémunir Laravel utilise ceci (source documentation Laravel) :

Preventing CSRF Requests

Laravel automatically generates a CSRF "token" for each active user session managed by the application. This token is used to verify that the authenticated user is the person actually making the requests to the application. Since this token is stored in the user's session and changes each time the session is regenerated, a malicious application is unable to access it.

The current session's CSRF token can be accessed via the request's session or via the csrf_token helper function:

```
use Illuminate\Http\Request;

Route::get('/token', function (Request $request) {
    $token = $request->session()->token();

    $token = csrf_token();

    // ...
});
```

Anytime you define a "POST", "PUT", "PATCH", or "DELETE" HTML form in your application, you should include a hidden CSRF _token field in the form so that the CSRF protection middleware can validate the request. For convenience, you may use the @csrf Blade directive to generate the hidden token input field:

« Laravel génère automatiquement un jeton CSRF pour chaque utilisateur actif en session géré par l'application. Ce jeton est utilisé afin de vérifier si l'utilisateur authentifié est bien celui qui fait la requête à l'application. Ce jeton est stocké dans la session de l'utilisateur et change chaque fois qu'une nouvelle session est générée, une application malveillante est incapable d'y accéder. »

« Le jeton CSRF de la session en cours peut être consulté via la requête de session ou via la fonction `csrf_token` du helper »

« Lorsque l'on définit un formulaire HTML dans notre application avec l'une des méthodes « POST », « PUT », « PATCH » ou « DELETE » vous devez inclure un champ jeton CSRF caché dans le formulaire afin que la protection CSRF de notre middleware valide la requête. Pour plus de commodité, vous pouvez utiliser `@csrf` de blade pour générer le champ caché du jeton : »

```
<form method="post" action="{{ route('product.store') }}" enctype="multipart/form-data">
    @csrf
```

Exemple depuis le code de mon application

`@csrf` reviendrait à faire ça :

```
<input type="hidden" name="_token" value="{{ csrf_token() }}" />
```

The `App\Http\Middleware\VerifyCsrfToken` middleware, which is included in the `web` middleware group by default, will automatically verify that the token in the request input matches the token stored in the session. When these two tokens match, we know that the authenticated user is the one initiating the request.

« Le fichier « `VerifyCsrfToken` » de notre dossier middleware, qui est inclus par défaut va automatiquement vérifier que le jeton dans notre champ de requête correspond au jeton stocké en session. Quand les deux jetons correspondent, nous savons que l'utilisateur authentifié est celui qui a initié la requête. »

Si le jeton CSRF est invalide ou absent lors de la soumission de notre formulaire, le navigateur nous renvoie sur une erreur 419, nous expliquant que notre session a expiré. Je rajouterai (car cela m'a posé un problème) que nous devons également ajouter un jeton CSRF lors d'une requête AJAX en méthode POST, sans cela notre console nous renvoie une erreur 500 (Internal Server Error).

245

The best way to solve this problem "X-CSRF-TOKEN" is to add the following code to your main layout, and continue making your ajax calls normally:

In header



```
<meta name="csrf-token" content="{{ csrf_token() }}" />
```

In script

```
<script type="text/javascript">
$.ajaxSetup({
  headers: {
    'X-CSRF-TOKEN': $('meta[name="csrf-token"]').attr('content')
  }
});
</script>
```

Réponse de Stack Overflow m'ayant permis de résoudre mon problème

Faible XSS

Le **Cross-site scripting** (abrégé XSS), est un type de faille de sécurité des sites web permettant d'**injecter du contenu dans une page**, provoquant ainsi des actions sur les navigateurs web visitant la page. Les possibilités des XSS sont très larges puisque l'attaquant peut utiliser tous les langages pris en charge par le navigateur (JavaScript, Java, Flash...) et de nouvelles possibilités sont régulièrement découvertes notamment avec l'arrivée de HTML 5. Il est possible par exemple, de rediriger vers un autre site pour de l'Hameçonnage ou encore de voler la session en récupérant les cookies.

Néanmoins **Blade**, le moteur de template de Laravel offre une **protection contre les attaques XSS** avec la syntaxe `{{...}}` qui « **échappe** » les données à l'affichage.

Injection SQL

L'**injection SQL** consiste à injecter dans une requête, un morceau de requête indésirable. Selon la méthode et le degré de vulnérabilité du serveur, elle peut être utilisée pour récupérer des données ou pour altérer les données stockées en base, voire pour vider ou supprimer les tables d'une base de données. **Eloquent**, l'ORM de Laravel prend en charge cette faille en utilisant « **PDO bindings**¹¹ » qui protège des injections SQL.

¹¹ Prépare une requête à l'exécution et retourne un objet, associe une valeur à un paramètre et exécute la requête préparée.

Assignation de masse

L'**assignation de masse** (ou mass assignment) est lorsqu'un formulaire est utilisé de manière abusive pour modifier des éléments de données auxquels l'utilisateur ne devrait normalement **pas être autorisé à accéder** tel que les autorisations accordées ou le rôle administrateur. Pour s'en prémunir il nous faut alors définir quels champs pourront être remplis dans un modèle.

```
class Category extends Model
{
    use HasFactory;

    protected $fillable = [
        'name',
        'slug',
        'icon',
    ];
}
```

Par exemple, dans ma catégorie, seuls les champs dans la méthode **\$fillable** pourront être remplis. La méthode **\$guarded** (avec la logique inverse) existe et peut également être utilisée.

Autre sécurité

Voici une liste des autres sécurités mis en place par Laravel :

- Le **throttle** est une **dissuasion** pour tout individu voulant **pirater** le compte d'un autre utilisateur en usant de l'**attaque par force brute**. En cas d'un certain nombre de connexions avortées (5 par défaut), on bloque l'accès (Erreur 429) pour le couple « identifiant/adresse IP » pendant une minute.
- Le **hashage** en **bcrypt** (par défaut) qui est utilisé pour **hasher** le mot de passe d'un utilisateur.
- La **double authentification** avec le kit de démarrage Jetstream, celle-ci permet d'accroître la protection du compte d'un utilisateur dans le cas où le pirate aurait les identifiants de celui-ci.

Jetstream

Comme énoncé précédemment, pour l'enregistrement et la connexion d'un utilisateur j'ai décidé d'utiliser le starter kit de Laravel Jetstream.

Pour ce faire, j'ai utilisé **composer** et est entré la commande suivante dans ma console

```
composer require laravel/jetstream
```

J'ai ensuite choisi d'installer **livewire** pour le design.

```
php artisan jetstream:install livewire
```

J'ai installé et « *build* » le gestionnaire de paquets de Node.js comme demandé.

```
npm install npm run dev
```

Et avant de faire ma migration, j'ai changé les données de mes utilisateurs afin de les faire correspondre à mon MCD.

```
class CreateUsersTable extends Migration
{
    /**
     * Run the migrations.
     *
     * @return void
     */
    public function up()
    {
        Schema::create('users', function (Blueprint $table) {
            $table->id();
            $table->string('name');
            $table->string('lastname');
            $table->dateTime('birth_date');
            $table->string('phone_number');
            $table->string('email')->unique();
            $table->timestamp('email_verified_at')->nullable();
            $table->string('password');
            $table->rememberToken();
            $table->string('profile_photo_path', 2048)->nullable();
            $table->timestamps();
        });
    }
}
```

```
php artisan migrate
```

Cette commande permet de créer et de mettre à jour un schéma de base de données.

Une fois Jetstream installé, la partie authentification est alimentée par Laravel **Fortify**. **Fortify** définit les routes et les contrôleurs pour **implémenter la fonctionnalité d'authentification** de l'application tant dis que Jetstream s'occupe des requêtes pour ces routes.

Ayant ajouté des champs aux données d'un utilisateur, il fallait que j'ajoute ceux-ci dans mon Fortify.

```
public function create(array $input)
{
    validator::make($input, [ //Conditions de mes inputs
        'name' => ['required', 'string', 'max:255'],
        'lastname' => ['required', 'string', 'max:255'],
        'birth_date' => ['required', 'date'],
        'phone_number' => ['required', 'string', 'max:255'],
        'email' => ['required', 'string', 'email', 'max:255', 'unique:users'],
        'password' => $this->passwordRules(),
        'terms' => Jetstream::hasTermsAndPrivacyPolicyFeature() ? ['required', 'accepted'] : '',
    ]->validate();

    return User::create([ //Données ajoutées en base de données si les champs sont validés
        'name' => $input['name'], //Colonne 'name' en DB prend la valeur de l'input 'name' du form
        'lastname' => $input['lastname'],
        'birth_date' => $input['birth_date'],
        'phone_number' => $input['phone_number'],
        'email' => $input['email'],
        'password' => Hash::make($input['password']),
    ]);
}
```

Multi Authentication

J'ai décidé de faire un « **multi auth** » pour mon application afin de rajouter une couche de **sécurité** et de bien **distinguer** la **partie admin** et la **partie utilisateur**.

Pour ce faire, j'ai créé un contrôleur Admin

```
php artisan make:controller AdminController
```

Puis j'ai créé mon model, ainsi qu'une « migration » de base de données

```
php artisan make:model Admin -m
```

« -m » est la commande abrégée de « --migration »

Comme pour mes utilisateurs, je change les données que je veux pour un administrateur.

```

public function up()
{
    Schema::create('admins', function (Blueprint $table) {
        $table->id();
        $table->string('name');
        $table->string('email')->unique();
        $table->timestamp('email_verified_at')->nullable();
        $table->string('password');
        $table->rememberToken();
        $table->string('profile_photo_path', 2048)->nullable();
        $table->timestamps();
    });
}

```

Schéma de la création de ma table admin en base de données

Afin de ne pas avoir de système d'enregistrement pour un admin j'ai utilisé le système de modèle « **factory** » et de « **seed data** » d'Eloquent afin d'insérer un admin dans ma base de données. Pour ce faire j'ai créé un AdminFactory :

```
php artisan make:factory AdminFactory
```

Puis en reprenant le code de l'exemple existant dans le dossier Factory j'ai créé mon admin :

```

public function definition()
{
    return [
        'name' => 'Admin',
        'email' => 'admin@gmail.com',
        'email_verified_at' => now(),
        'password' => '$2y$10$92IXUNpkj08rQ0Q5byMi.Ye4oKoEa3Ro9llC/.og/at2.uheWG/igi', // password
        'remember_token' => Str::random(10),
    ];
}

```

En suivant la documentation de Laravel, j'ai ajouté ceci dans mon Seeder, cela permet de récupérer les données de mon AdminFactory afin de peupler ma base de données, puis j'ai exécuté la commande permettant de migrer mes « **seed** ».

```

public function run()
{
    \App\Models\Admin::factory()->create();
}

php artisan migrate --seed

```

Garde d'authentification

Un **garde** est un moyen de fournir la logique permettant **d'identifier les utilisateurs authentifiés**. Laravel fournit **différents gardes** comme des **sessions** et des **jetons**. Le gardien de session maintient l'état de l'utilisateur dans chaque demande par le biais de **cookies**. D'autre part, le gardien de jeton authentifie l'utilisateur en **vérifiant** un jeton valide **dans chaque demande**. Le service de garde définit la logique de l'authentification et il n'est pas nécessaire qu'il traite toujours de cette information en récupérant des informations d'identification valides du serveur principal. On peut implémenter une protection qui vérifie simplement la présence d'une chose spécifique dans les en-têtes de demande et authentifie les utilisateurs en fonction de cela.

Le garde d'authentification se situe dans le fichier **auth.php** dans le dossier **config**. En reprenant le garde d'un utilisateur, je créer le miens pour l'administrateur :

```
'guards' => [  
    'web' => [  
        'driver' => 'session',  
        'provider' => 'users',  
    ],  
    'admin' => [  
        'driver' => 'session',  
        'provider' => 'admins',  
    ],  
]
```

Ensuite je dois créer mon provider, encore une fois je reprends ce qui a déjà été fait un utilisateur et je remplace ce dont j'ai besoin (le nom et le modèle ici) :

```
'providers' => [  
    'users' => [  
        'driver' => 'eloquent',  
        'model' => App\Models\User::class,  
    ],  
    'admins' => [  
        'driver' => 'eloquent',  
        'model' => App\Models\Admin::class,  
    ],  
]
```

Et enfin je fais la même chose pour la partie « **passwords** ».

En suivant la documentation, j'ajoute ce code à la fonction register() :

```
$this->app->when(AdminController::class, AttemptToAuthenticate::class, RedirectIfTwoFactorAuthenticatable::class)
->needs(StatefulGuard::class)
->give(function() {
    return Auth::guard('admin');
});
```

Je crée un dossier **Guards** avec le fichier **AdminStatefulGuard** avec le fichier de base **StatefulGuard** situé dans mon **vendor**. Je fais de même pour mon **AdminController** avec **AuthenticatedSessionController**.

Je crée une nouvelle fonction afin de récupérer mon garde admin :

```
public function loginForm(){
//retourne sur la vue admin_login dans auth et notre guard sera admin
return view('auth.admin_login', ['guard' => 'admin']);
}
```

Je crée mes routes :

```
//Connexion
Route::group(['prefix' => 'admin', 'middleware' => ['admin:admin']], function(){
    //Méthode('url'), [Controller::class, 'propriété dans le controller'] non dans la vue
    Route::get('/login', [AdminController::class, 'loginForm']);
    Route::post('/login', [AdminController::class, 'store'])->name('admin.login');
});
```

La route en méthode « **get** » permet de récupérer le formulaire admin et la route en méthode « **post** » permet de se connecter.

```
{!-- Si guard = admin alors l'url est (admin/login) sinon on utilise la route user --}
<form method="POST" action="{ { isset($guard) ? url($guard.'/login') : route('login') } }">
    @csrf
```

Dans mon **RouteServiceProvider** j'ajoute cette fonction :

```
public static function redirectTo($guard){
    return $guard.'/dashboard';
}
```

Je crée un nouveau **Middleware de redirection** ainsi que sa route si mon admin est authentifié (tout en l'ajoutant à mon fichier **Kernel**) :

```
Route::middleware(['auth:admin'])->group(function(){
    Route::middleware(['auth:sanctum,admin', 'verified'])->get('/admin/dashboard', function () {
        return view('admin.index');
    })->name('dashboard')->middleware('auth:admin');
```

```

public function handle(Request $request, Closure $next, ...$guards)
{
    $guards = empty($guards) ? [null] : $guards;

    foreach ($guards as $guard) {
        if (Auth::guard($guard)->check()) {
            return redirect($guard.'/dashboard');
        }
    }

    return $next($request);
}

```

Et enfin pour la mise en place du Logout qui est également différent de celui de notre utilisateur :

```

//Déconnexion
Route::get('admin/logout', [AdminController::class, 'destroy']->name('admin.logout'));

```

Route de déconnexion dans notre web.php

```

public function destroy(Request $request): LogoutResponse
{
    $this->guard->logout();

    $request->session()->invalidate();

    $request->session()->regenerateToken();

    return app(LogoutResponse::class);
}

```

Fonction de déconnexion dans l'AdminController

VI. Réalisation de l'application

CRUD

La première tâche réalisée après la mise en place de mon multi auth a été la création de mon **CRUD**¹² dans la partie « **backend** » de mon application. Pour ce qui est de la partie visuelle de cette partie j'ai eu recours à un **template HTML/CSS** afin de gagner du temps et montrer que j'étais capable de me servir d'un outil de ce genre.

Pour la création de mes classes, j'ai commencé par créer les modèles ainsi que leurs migrations :

```
php artisan make:model Category -m
```

Commande pour créer le modèle et la migration.

```
public function up()
{
    Schema::create('categories', function (Blueprint $table) {
        $table->id();
        $table->string('name');
        $table->string('slug');
        $table->string('icon');
        $table->timestamps();
    });
}
```

Fonction qui permet de créer ma table dans la base de données.

```
protected $fillable = [
    'name',
    'slug',
    'icon',
];
```

On indique les champs que l'on peut remplir afin de se prémunir de l'assignation de masse.

¹² Create Read Update Delete pour Créer, lire, modifier et supprimer

Ensuite, on crée notre contrôleur, ici notre contrôleur sera dans un dossier appelé « Backend » afin de garder une structure ordonnée.

php artisan make:controller Backend/CategoryController

Commande pour créer un contrôleur.

Une fois cela fait, il nous faut créer nos méthodes au sein de notre contrôleur.

```
public function view(){
    //latest() permet d'obtenir les dernières catégories (en fonction de created_at) en premier
    $category = Category::latest()->get();
    //On retourne les données dans la vue située dans le dossier backend, puis category
    //compact('') permet de créer un tableau à partir de variables et de leur valeur
    return view('backend.category.category_view', compact('category'));
}
```

Méthode view, permet d'afficher la liste des catégories.

```
public function store(Request $request){
    //On récupère les données des inputs si nos champs sont remplis
    $request->validate([
        'name' => 'required',
        'icon' => 'required',
    ], [
        'name.required' => 'Entrez le nom de la catégorie' //Message d'erreur si le champ est vide
    ]);
    //On insère les données de nos inputs dans la BDD
    Category::insert([
        'name' => $request->name, // 'name' est le nom de la colonne en BDD et $request->name celle du champ
        'icon' => $request->icon,
        //Pour le slug on met tous les caractères en minuscule et on remplace les espaces par des -
        'slug' => strtolower(str_replace(' ', '-', $request->name)),
    ]);
    //On établit le message de notification que la catégorie est créée
    $notification = [
        'message' => 'Catégorie créée avec succès',
        'alert-type' => 'success'
    ];
    //On redirige vers la page précédente en affichant le message
    return redirect()->back()->with($notification);
}
```

Méthode store, permet d'ajouter une nouvelle ligne en base de données.

```
public function edit($id){
    //On prend l'id d'une catégorie et on retourne un seul élément
    //Si il n'y a pas d'élément qui correspond ça renvoie une erreur
    $category = Category::findOrFail($id);
    //Si on trouve l'élément correspondant on renvoie les données
    return view('backend.category.category_edit', compact('category'));
}
```

Méthode edit, permet de récupérer les données de la catégorie à modifier.


```

public function update(Request $request, $id){
    //On cherche la catégorie par id, si on la trouve,
    //On met à jour les champs name et icon avec les données des inputs
    Category::findOrFail($id)->update([
        'name' => $request->name,
        'icon' => $request->icon,
        //Pour le slug on met tous les caractères en minuscule
        //On remplace les espaces par des -
        'slug' => strtolower(str_replace(' ', '-', $request->name)),
    ]);
    //On établit le message de notification que la catégorie est créée
    $notification = [
        'message' => 'Catégorie modifiée avec succès',
        'alert-type' => 'success'
    ];
    //On redirige vers la liste des catégories en affichant le message
    return redirect()->route('categories')->with($notification);
}

```

Méthode update, permet de « poster » les données de la catégorie à modifier.

```

public function delete($id){
    //On cherche la catégorie par id, si on la trouve,
    //On supprime la catégorie
    Category::findOrFail($id)->delete();
    $notification = [
        'message' => 'La catégorie a été supprimée avec succès',
        'alert-type' => 'success'
    ];
    //On redirige vers la liste des catégories en affichant le message
    return redirect()->route('categories')->with($notification);
}

```

Méthode delete, supprime la ligne en base de données de la catégorie ciblée.

J'ai ainsi créé, avec la même logique de CRUD mes éléments tels que mes sous-catégories, mes sous sous-catégories et mes produits. Une fois ceci fait, je me suis attelé à la création de mon **panier**.

Panier en session

Pour la réalisation de mon panier en **session** j'ai eu recours au bundle « **bumbummen99/shoppingcart** » afin d'être ancré le plus possible dans le réel et dans le professionnel.

Une **session** correspond à une façon de **stocker** des données différentes pour chaque utilisateur en utilisant un **identifiant de session unique**. Les identifiants de session vont généralement être envoyés au navigateur via des **cookies de session** et vont être utilisés pour récupérer les données existantes de la session.

Un des grands intérêts des sessions est qu'on va pouvoir **conserver des informations** pour un utilisateur lorsqu'il navigue d'une page à une autre dans le but d'éviter de lui redemander les mêmes informations. De plus, les informations de session ne vont cette fois-ci pas être **stockées** sur les ordinateurs des visiteurs à la différence des cookies mais plutôt **côté serveur** ce qui fait que les sessions vont pouvoir être beaucoup **plus sûres** que les cookies.

Le but des sessions n'est pas de conserver des informations indéfiniment mais simplement durant une « **session** ». Une session démarre dès que la fonction **session_start()** est appelée et se termine en général dès que la fenêtre courante du navigateur est fermée (à moins qu'on appelle une fonction pour terminer la session de manière anticipée ou qu'un cookie de session avec une durée de vie plus longues ait été défini). Il existe également une **superglobale**¹³ **\$_SESSION** qui est un **tableau associatif**¹⁴ qui va contenir toutes les données de session une fois la session démarrée.

Laravel prend en charge nativement la session, c'est-à-dire que lorsqu'un visiteur entre sur l'application, Laravel démarre la fonction **session_start()**. La session d'un utilisateur a une durée définie, celle-ci peut être modifiée dans notre fichier « **.env** » ainsi que dans notre fichier « **session.php** » se trouvant dans le dossier config. Elle peut également être fermée lorsque l'on quitte notre navigateur.

```
'lifetime' => env('SESSION_LIFETIME', 120),  
  
'expire_on_close' => true,  
  
SESSION_LIFETIME=120
```

¹³ Variable prédéfinie en PHP, ce qui signifie qu'elles sont toujours disponibles quel que soit le contexte du script.

¹⁴ Type de données associant à un ensemble de clefs, un ensemble correspondant de valeurs.

Laravel permet également d'avoir une « trace » des utilisateurs actuellement sur l'application avec un **id hashé** grâce à une **table « sessions »** automatiquement pris en charge par le framework. La « **table sessions** » se présente ainsi :

| # | Nom | Type de données | Taille/Ensem... | Non signé | NULL autorisé | ZEROFILL | Par défaut |
|---|---------------|-----------------|-----------------|-------------------------------------|-------------------------------------|--------------------------|---------------|
| 1 | id | VARCHAR | 255 | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | Pas de défaut |
| 2 | user_id | BIGINT | 20 | <input checked="" type="checkbox"/> | <input checked="" type="checkbox"/> | <input type="checkbox"/> | NULL |
| 3 | ip_address | VARCHAR | 45 | <input type="checkbox"/> | <input checked="" type="checkbox"/> | <input type="checkbox"/> | NULL |
| 4 | user_agent | TEXT | | <input type="checkbox"/> | <input checked="" type="checkbox"/> | <input type="checkbox"/> | Pas de défaut |
| 5 | payload | TEXT | | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | Pas de défaut |
| 6 | last_activity | INT | 11 | <input checked="" type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | Pas de défaut |

Cette table nous donne accès aux informations suivantes :

- **id** : identifiant hashé permettant d'identifier un utilisateur.
- **user_id** : identifiant permettant de voir que l'utilisateur est authentifié (cette valeur est NULL si celui-ci ne l'est pas).
- **ip_address** : son adresse IP.
- **user_agent** : le navigateur qu'il utilise.
- **payload** : les données de sa session.
- **last_activity** : timestamp de sa dernière activité sur l'application.

Le **panier** d'un e-commerce est en **session** car nous ne voulons **pas enregistrer** les produits ajoutés du panier dans une **base de données** puisque souvent les paniers ne sont pas payés (les utilisateurs ne finalisent pas toujours la commande). De plus, avec un panier en session, l'utilisateur n'a **pas besoin d'être authentifié** pour ajouter un produit à celui-ci (il devra s'inscrire ou s'authentifier pour procéder au paiement). Cela évitera donc de « **polluer** » la base de données. En revanche, (si le panier et le paiement sont validés) la **commande sera enregistrée dans la base de données**. De la même façon, l'application propose à l'utilisateur **authentifié** de pouvoir ajouter un ou plusieurs produits dans sa **liste de souhait**, qui elle, sera **stockée** en base de données.

Le bundle « **shoppingcart** » met à disposition différentes **méthodes**, que nous pouvons utiliser lorsque nous **instancions** ¹⁵ la classe « **Cart** ». Nous avons également la possibilité de « configurer » notre panier à l'aide de ce bundle.

```
//Permet de supprimer le panier lorsque l'utilisateur se déconnecte,
//Ceci pour des raisons de sécurité,
//Si celui-ci se connecte dans un endroit public par exemple
'destroy_on_logout' => true,
```

¹⁵ Une instance désigne une occurrence d'une classe. Cette dernière est elle-même définie comme la structure d'un objet, c'est-à-dire comme l'ensemble des fonctionnalités et des comportements qui caractérisent cet objet.

```

public function addToCart(Request $request, $id){
    //On cherche notre produit dans la BDD
    $product = Product::findOrFail($id);
    //Si le produit n'est pas au rabais
    if ($product->discount_price == NULL) {
        Cart::add([ //On appelle la méthode pour ajouter au panier
            'id' => $id, //id produit
            'name' => $request->name, //nom du produit
            'qty' => $request->quantity, //quantité
            'price' => $product->selling_price, //prix de vente
            'weight' => 1, //donnée par défaut du bundle
            'options' => [ //autres données que celle prise en charge
                'image' => $product->thumbnail, //vignette
                'color' => $request->color, //couleur
            ],
        ]);
    }
    //on retourne et on affiche une alerte
    return response()->json(['success' => 'Ajouté au panier']);
}
else{ //Sinon le prix est au rabais
    Cart::add([
        'id' => $id,
        'name' => $request->name,
        'qty' => $request->quantity,
        'price' => $product->discount_price, //prix au rabais
        'weight' => 1,
        'options' => [
            'image' => $product->thumbnail,
            'color' => $request->color,
        ],
    ]);
    return response()->json(['success' => 'Ajouté au panier']);
}
}

```

Cette fonction a été construite en suivant la documentation, elle nous permet d'ajouter un produit à notre panier. Sans l'utilisation de ce bundle, nous aurions dû **instancier** notre classe « **Session** », native de Laravel, **recupérer** les informations du produit à ajouter au panier, récupérer notre panier en session et « **put** » les valeurs du produit dans celui-ci.

Nous avons également d'autres fonctionnalités comme un compteur, permettant de compter le nombre de produit, ou un calculateur permettant de calculer la somme totale du panier (en prenant en compte le nombre d'articles). Nous pouvons afficher, modifier ou supprimer un ou plusieurs produits.

Vérification

Avant de procéder au paiement des produits mis dans le panier en session, on doit procéder à une **étape de vérification** « **checkout** », qui va permettre à l'utilisateur en session de vérifier que les produits sont bien ceux qu'il veut commander ainsi que de définir les informations concernant le destinataire (lui par défaut). C'est également à ce moment-là que l'utilisateur devra entrer une adresse de livraison.

```
public function checkoutCreate(){
    //On vérifie que l'utilisateur est authentifié
    if (Auth::check()) {
        //On vérifie que l'utilisateur a au moins un produit dans son panier
        if (Cart::total() > 0) {
            //On récupère les données du panier
            $carts = Cart::content(); //Contenu du panier
            $cartQty = Cart::count(); //Quantité de produits du panier
            $cartTotal = Cart::total(); //Coût total du panier
            //On retourne les données du panier dans notre vu checkout
            //compact() permet de créer un tableau à partir de variables et de leur valeur
            return view('frontend.checkout.checkout_view', compact('carts', 'cartQty', 'cartTotal'));
        } else //Sinon on signale à l'utilisateur qu'il doit ajouter un produit au panier
        {
            $notification = [
                'message' => 'Vous devez ajouter un produit à votre panier',
                'alert-type' => 'error'
            ];
            //On redirige vers l'accueil avec le message de notification
            return redirect()->to('/')->with($notification);
        }
    } else //Sinon on signale à l'utilisateur de se connecter
    {
        $notification = [
            'message' => 'Veuillez vous connecter',
            'alert type' => 'error'
        ];
        //On redirige vers la page de connexion avec le message de notification
        return redirect()->route('login')->with($notification);
    }
}
```

Méthode nous permettant de récupérer les données du panier en session afin de les afficher à notre utilisateur.


```

public function checkoutStore(Request $request){
    //On définit la variable $data comme étant un tableau
    $data = [];
    //Pour chaque clé de notre tableau data, on attribut une valeur
    //Ces valeurs sont les données entrées dans notre formulaire
    $data['name'] = $request->name;
    $data['email'] = $request->email;
    $data['phone'] = $request->phone;
    $data['country'] = $request->country;
    $data['number'] = $request->number;
    $data['street'] = $request->street;
    $data['city'] = $request->city;
    $data['post_code'] = $request->post_code;
    $data['notes'] = $request->notes;
    //On récupère la somme totale de notre panier
    $cartTotal = Cart::total();
    //Si l'utilisateur à choisi comme moyen de paiement Stripe
    if ($request->payment_method == 'stripe') {
        //On retourne à notre vue notre tableau data ainsi que notre total
        return view('frontend.payment.stripe',compact('data','cartTotal'));
    }else{
        $notification = [
            'message' => 'Moyen de paiement à venir',
            'alert-type' => 'info'
        ];
        //On redirige vers la page de connexion avec le message de notification
        return redirect()->route('checkout')->with($notification);
    }
}

```

Méthode permettant de récupérer temporairement les données entrées dans le formulaire ainsi que les données de notre panier. (Ces données ne sont pas encore ajoutées à notre base de données dans le cas où notre utilisateur en session ne finaliserait pas sa commande).

Une fois la vérification faite, notre formulaire de commande et notre moyen de paiement choisi, l'utilisateur peut alors effectuer son paiement.

Paielement avec l'API Stripe

Dans le but d'effectuer un paiement **sécurisé**, j'ai décidé d'avoir recours à l'**API** Stripe. Une **API** (Application Programming Interface ou interface de programmation d'application en français) est un ensemble de définitions et de protocoles qui facilite la création et l'intégration de logiciels d'applications.

Une **API** permet à l'application de **communiquer** avec d'autres produits et services sans connaître les détails de leur mise en œuvre. Elle **simplifie** le développement de l'application et fait ainsi **gagner du temps**. Une **API** offre plus de **flexibilité**, **simplifie** la conception, l'administration et l'utilisation, et donne les **moyens d'innover**.

Étant la première fois que j'utilisais une API de ce genre j'ai choisi Stripe pour ses recommandations en termes de facilité d'intégration ainsi que pour sa documentation claire et complète.

Afin de pouvoir utiliser cette API et avant l'installation de celle-ci j'ai dû augmenter la sécurité de mon application avec le **protocole SSL** que j'ai pu mettre en place en localhost à l'aide de mon environnement de développement web **Laragon**.

Le **protocole SSL** (Secure Sockets Layer) est un **protocole cryptographique** utilisé pour assurer la **sécurité des communications** sur Internet. Le SSL crée un canal sécurisé entre deux machines ou appareils communiquant sur Internet ou un réseau interne. Son usage le plus courant est la **sécurisation de la communication entre un navigateur web et un serveur web**. L'adresse URL passe alors de HTTP à **HTTPS**¹⁶, le « S » signifiant « **Secure** ».

Le **certificat SSL** est un **fichier** de données qui lie une **clé cryptographique** aux informations de l'application. Installé sur un serveur, le **certificat active le cadenas et le protocole « https »**, afin d'**assurer** une **connexion sécurisée** entre le serveur web et le navigateur. Le **SSL** est généralement utilisé pour **sécuriser les transactions bancaires**, le **transfert de données** et les **informations de connexions**.

Une fois mon protocole SSL mis en place et mon application sécurisée, j'ai suivi la documentation de Stripe et j'ai installé l'API via composer.

J'ai ensuite procédé à mon inscription sur l'application Stripe afin de récupérer ma « **clé d'API** ¹⁷ » permettant de faire fonctionner l'API. J'ai ajouté le script de l'application, ce **script** a été intégré dans le **<HEAD>** de mon application, celui-ci doit être présent dans toutes les pages pour des raisons de **sécurité**. Chaque utilisateur reçoit un **jeton unique** durant sa session en provenance de l'API qui sera comparé à un jeton identique lors du paiement. Si ce jeton ne correspond pas, alors l'API renvoie une erreur. Toujours en suivant la documentation, j'ai alors mis en place différents éléments dans mon script de la vue de paiement « **stripe** » afin d'obtenir le résultat souhaité.

¹⁶ Version sécurisée du protocole de communication http.

¹⁷ Une clé d'API est un identifiant unique utilisé pour authentifier un utilisateur, un développeur ou un programme appelant auprès d'une API.

```

<script type="text/javascript">
  //Variable stripe avec ma clé publique d'API
  var stripe = Stripe('pk_test_51Jc34MBomdm5Uc1pnEFHMrxeIdrc6C7hhfnwWkCz');
  //Création de l'instance éléments
  var elements = stripe.elements();
  //Style mis en place par le template prit de la documentation
  var style = {
    base: {
      color: '#32325d',
      fontFamily: '"Helvetica Neue", Helvetica, sans-serif',
      fontSmoothing: 'antialiased',
      fontSize: '16px',
      '::placeholder': {
        color: '#aab7c4'
      }
    },
    invalid: {
      color: '#fa755a',
      iconColor: '#fa755a'
    }
  };
  //Création de l'instance "card"
  var card = elements.create('card', { style: style });
  //Ajout d'une instance "card Element" dans "card-element" <div>.
  card.mount('#card-element');
  //Gère nos erreurs.
  card.on('change', function(event) {
    var displayError = document.getElementById('card-errors');
    if (event.error) {
      displayError.textContent = event.error.message;
    } else {
      displayError.textContent = '';
    }
  });
  // Gère l'envoi du formulaire
  var form = document.getElementById('payment-form');
  form.addEventListener('submit', function(event) {
    event.preventDefault();
    stripe.createToken(card).then(function(result) {
      if (result.error) {
        // Informe l'utilisateur en cas d'erreur
        var errorElement = document.getElementById('card-errors');
        errorElement.textContent = result.error.message;
      } else {
        // Envoie le token
        stripeTokenHandler(result.token);
      }
    });
  });
  // Envoie le formulaire avec l'id du token
  function stripeTokenHandler(token) {
    //Insère le jeton dans le form afin d'être soumis au serveur
    var form = document.getElementById('payment-form');
    var hiddenInput = document.createElement('input');
    hiddenInput.setAttribute('type', 'hidden');
    hiddenInput.setAttribute('name', 'stripeToken');
    hiddenInput.setAttribute('value', token.id);
    form.appendChild(hiddenInput);
    // Envoie le formulaire
    form.submit();
  }
}
</script>

```



```

public function stripeOrder(Request $request){
    //Somme totale de notre panier
    $total_amount = round(Cart::total());
    //Clé privée personnelle de l'API
    \Stripe\Stripe::setApiKey('sk_test_51Jc34MBomdm5Uc1pB6YxKP6Gwp3nyATHm7opWM');
    //Post jeton Stripe
    $token = $_POST['stripeToken'];
    $charge = \Stripe\Charge::create([ //Méthode de l'API Stripe
        'amount' => $total_amount*100,
        'currency' => 'eur',
        'description' => 'ZiCommerce',
        'source' => $token,
        'metadata' => ['order_id' => uniqid()],
    ]);
    //On enregistre la commande de notre utilisateur en BDD
    //insertGetId permet de d'insérer et de récupérer l'id
    $order_id = Order::insertGetId([
        'user_id' => Auth::id(), //id de l'utilisateur Authentifié
        //inputs remplis dans notre checkout, on récupère ceux-ci
        //grâce au form en méthode POST de notre vue avec des champs hidden
        //ayant pour valeur les données de notre variable $data
        'country' => $request->country,
        'number' => $request->number,
        'street' => $request->street,
        'city' => $request->city,
        'name' => $request->name,
        'email' => $request->email,
        'phone' => $request->phone,
        'post_code' => $request->post_code,
        'notes' => $request->notes,
        //Ces champs ont été rajoutés en suivant la documentation
        'payment_type' => 'Stripe',
        'payment_method' => 'Stripe',
        'payment_type' => $charge->payment_method,
        'transaction_id' => $charge->balance_transaction,
        'currency' => $charge->currency,
        'amount' => $total_amount,
        'order_number' => $charge->metadata->order_id,
        'invoice_no' => 'EOS'.mt_rand(10000000,99999999),
        'order_date' => Carbon::now()->format('d F Y'),
        'order_month' => Carbon::now()->format('F'),
        'order_year' => Carbon::now()->format('Y'),
        'status' => 'Pending',
        'created_at' => Carbon::now(),
    ]);
}

```

Première partie de la fonction `stripeOrder` dans notre « **StripeController** ».

```

//On récupère tous les produits de notre panier
$scarts = Cart::content();
//Pour chacun de ces produits
foreach ($scarts as $scart) {
    //On insère en BDD
    OrderItem::insert([
        'order_id' => $order_id, //id de la commande en cours
        'product_id' => $scart->id, //id du produit
        'color' => $scart->options->color,
        'size' => $scart->options->size,
        'qty' => $scart->qty,
        'price' => $scart->price,
        'created_at' => Carbon::now(),
    ]);
}
//Une fois la commande passée, on vide notre panier
Cart::destroy();
$notification = [
    'message' => 'Votre commande va être prise en charge',
    'alert-type' => 'success'
];
return redirect()->route('dashboard')->with($notification);
}

```

Deuxième partie de la fonction *stripeOrder* dans notre « **StripeController** ».

En ayant suivis la documentation, j'ai créé la méthode me permettant d'ajouter en base de données la commande de l'utilisateur en session, dans les tables « **orders** » et « **order_items** ». Ainsi nous avons donc une trace de ce que l'utilisateur a commandé.

VII. Rendu final

Axes d'améliorations

- Améliorer le design global de l'application.
- Ajouter du React aux filtres lorsque nous avons une liste de produits, afin de rendre le tout plus dynamique.
- Choix de l'ordre des produits.
- Ajouter un système de coupon et de code promotionnel.
- Ajouter la possibilité de zoomer sur l'image d'un produit.
- Ajouter un système permettant de changer les images du produit en fonction de la couleur de celui-ci.
- Mettre en place un système de mailer afin qu'un utilisateur puisse recevoir une confirmation de commande.
- Ajouter un aperçu lorsqu'on cherche un produit dans la barre de recherche.
- Ajouter une version d'une autre langue (dans le cas où l'entreprise serait à l'international).
- Ajouter un système permettant de comparer des produits entre eux.
- Ajouter un fil d'actualité/blog de manière à tenir les utilisateurs informés des dernières nouveautés.

Conclusion

La conception et le développement de cette application se sont avérés être une excellente expérience pour apprendre à utiliser le Framework Laravel dont je n'avais absolument aucune connaissance. J'ai également pu apprendre différentes choses à propos du marketing et de ce qui entoure les e-commerces de manière générale. La réalisation de ce projet m'a également permis de renforcer les compétences acquises lors de la formation.

L'application peut être utilisable en l'état mais il reste de nombreuses modifications à faire ainsi que d'ajouts de fonctionnalités dans le but d'améliorer l'expérience de l'utilisateur.

Ce projet, le stage et la formation ont été une expérience très enrichissante et j'espère pouvoir à l'avenir continuer à apprendre dans ce domaine.