01/02/2021

Dossier de Synthèse

Développeur web et web mobile

Niveau III

Lisa Foret

Table des matières

Liste des compétences du référentiels couverte par le projet

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

- Maquetter une application
- Réaliser une interface utilisateur web statique et adaptable
- Développer une interface utilisateur web dynamique

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

- Développer les composants d'accès aux données
- Développer la partie back-end d'une application web ou web mobile
- Elaborer et mettre en œuvre des composants dans une application de gestion de contenu ou e-commerce

I. Introduction

1. Présentation personnelle

Je m'appelle Lisa FORET, 23 ans.

Après avoir réalisé des études en biotechnologies et m'être spécialisée en chimie, je réalise, suite à mes expériences professionnelles, que ce domaine ne correspond plus à mes attentes.

Ayant toujours été intéressée par l'informatique et plus particulièrement par le développement web, je décide de me lancer dans une reconversion professionnelle. Après une période d'auto-formation sur Openclassroom, je décide de trouver une formation afin d'accélérer ma reconversion. C'est ainsi que j'ai été acceptée au sein d'Elan Formation pour le titre de Développeur web et web mobile de niveau III.

2. Présentation de Elan

Elan formation est un organisme de formation local dans les domaines de la bureautique, la PAO, le multimédia, d'internet, des techniques de secrétariat. L'organisme propose des formations sur mesures et individualisées. Ils disposent de locaux à Strasbourg, Sélestat, Haguenau, Saverne, Colmar, Mulhouse, Metz et Nancy.

3. Présentation du Projet

Le but de mon projet est de créer une boutique en ligne. Pour cela, je me suis appuyée sur la société dans laquelle j'ai effectué mon stage : le Caveau d'Aloxe-Corton. Il est situé au cœur de la Bourgogne dans un petit village nommé Aloxe-Corton.

Le caveau est né en 1975 de la volonté de plusieurs vignerons de se regrouper afin d'avoir un lieu commun au centre du village. Aujourd'hui, ce caveau représente 5 domaines sur le village d'Aloxe-Corton : le domaine Chapuis, le domaine Meuneveaux, le domaine Colin, le domaine Follin-Arbelet et le domaine Poisot.

Le gérant du caveau, Denis Priest a pour projet, dans les années à venir, de créer un site web afin de pouvoir expédier du vin à ses clients en France et à l'internationale et garder contact avec eux. Je souhaite donc mettre au point une boutique en ligne afin de lui apporter une ébauche pour ses futurs projets.

Actuellement, le statut de l'entreprise ne permet pas d'afficher les prix sur internet, c'est pourquoi dans ce projet tous les prix seront factices. En revanche, toutes les autres informations seront réelles. À noter que le caveau d'Aloxe-Corton n'est pas considéré comme mon client lors de ce projet.

Le caveau d'Aloxe-Corton possédant déjà un site web statique présentant l'entreprise, je me focaliserai essentiellement sur le développement de la boutique.

4. Objectifs

L'objectif de ce projet est de réaliser une boutique en ligne viable. Pour cela, j'ai établi un cahier des charges avec les fonctionnalités essentielles :

Le site web doit avoir une interface utilisateur où celui-ci peut :

- S'inscrire, se connecter et se déconnecter

- Obtenir la liste des produits, ainsi que ses caractéristiques
- Accéder à un panier et pouvoir y ajouter les produits en quantité souhaité
- Accéder à son profil avec ses informations, notamment son mail, ses adresses, ses commandes effectuées. Il pourra également modifier son mail et son mot de passe
- Avoir la possibilité de passer une commande et de la payer, et par la suite, obtenir une facture en PDF

L'utilisateur a besoin de se connecter uniquement pour accéder à son profil et passer une commande.

Le site web doit avoir une partie administrateur où celui-ci peut :

- Se connecter et se déconnecter
- Ajouter, modifier, désactiver et supprimer un produit
- Obtenir l'historique de toutes les commandes effectuées
- Gérer les commandes

L'administrateur a également accès à tout ce que l'utilisateur a accès.

Voici ci-dessous les points non-essentiels du cahier des charges, qui s'ils ne sont pas respectés, peuvent être poussés en axe d'amélioration :

- Un système de filtre des produits, par type, domaine ou appellation
- Un système de tri des produits par prix
- Un système de mail qui demande une vérification à l'inscription et envoie les factures ainsi que les confirmations de commande
- Une version anglaise du site
- Un système de disposition des produits (en colonne ou en tableau)

De plus, le site posséderait une partie site vitrine concernant la présentation de la société.

5. RGPD

La protection des données personnelles est un sujet essentiel car il concerne tout individu. Le Règlement Général sur la Protection des Données encadre le traitement des données personnelles sur le territoire de l'Union Européenne. Applicable depuis 2018, il encadre la mise en œuvre des traitements de données à caractère personnel. Il fixe les conditions dans lesquelles de telles données peuvent être légalement collectées, conservées et exploitées par les organismes.

Le RGPD s'articule autour de trois axes majeurs :

- Le renforcement quantitatif et qualitatif des droits des personnes,
- Une nouvelle logique de responsabilisation de l'ensemble des acteurs des traitements de données,
- Le renforcement des pouvoirs de sanction des CNIL (Commission Nationale de l'Informatique et des Libertés, elle est chargée de s'assurer que le développement de l'informatique se fera toujours dans le respect de la vie privé et des libertés individuelle.) européennes.

Le RGPD s'applique à tous les organismes publics et privés : les entreprises, les administrations, les collectivités, les associations, établies sur le territoire de l'UE ou dont l'activité cible des personnes qui se trouvent sur le territoire de l'UE.

Les 8 principes de la protection des données :

- Licéité du traitement
- Finalité du traitement
- Minimisation des données
- Protection particulière de certaines données
- Conservation limitée des données
- Obligation de sécurité
- Transparence
- Droits des personnes

Afin de suivre ces principes dans mon projet, je récupère uniquement les informations essentielles de l'utilisateur destinées à accomplir le service que je propose. Aucune donnée sensible ne sera collectée. Lors de la conception de ce projet, il a été prévu que l'utilisateur puisse supprimer son compte s'il le souhaite, seule la facture sera conservée à des fins commerciales.

De plus, lors de l'inscription l'utilisateur doit accepter les conditions générales d'utilisation. Ces conditions rappelleront à l'utilisateur pourquoi ses données sont collectées et à quelles fins, mais également ses droits concernant ses données.

II. Technologies utilisées

1. Langages

Php (HyperText Preprocessor): c'est un langage de script(interprété) open source, coté serveur. Il est généralement utilisé pour le développement de site/page web dynamique.

HTML5 CSS3 : Hypertext Markup Language est un langage de balisage conçu pour représenter des pages web. Cascading Style Sheet ou les feuilles de style en cascade sont un langage informatique qui décrit la présentation des documents HTML et XML(eXtensible Markup Language : langage de balisage extensible, conçu pour structurer des fichiers).

JavaScript c'est un langage de programmation de scripts, principalement employé dans les pages web interactives. **// à garder ? coder en Jquery**

2. Outils supplémentaires

Visual Studio Code : est un éditeur de code extensible développé par Microsoft pour Windows, il dispose d'énormément d'extensions.

Git : est un système de gestion de version décentralisé, qui m'a permis notamment de travailler sur plusieurs postes, et de conserver et d'exploiter l'historique du code de mon application.

Jmerise : c'est un logiciel dédié à la modélisation des modèles conceptuels de données (MCD), il permet la généralisation et la spécialisation des entités, la création des relations et des cardinalités ainsi que la généralisation des modèles logiques de données (MLD) et des script SQL.

Adobe XD : est un outil de conception d'expérience utilisateur basé sur le vecteur pour les applications web et les applications mobiles. Utilisé pour concevoir les maquettes.

Laragon : est un environnement de développement web dédié au système d'exploitation Windows. Il est composé de différentes technologies : Apache (serveur web), PHP (langage interprété coté serveur), MySQL (base de données).

3. API, Librairies & Framework

a. API

Une API, Application Programming Interface ou Interface de programmation en français. Elle permet de rendre disponibles certaines données ou fonctionnalités d'une application existante. C'est l'intermédiaire qui permet à deux systèmes informatiques indépendants d'interagir de manière automatique.

Stripe est une plateforme de traitement de paiement qui permet de transférer l'argent du compte bancaire d'un client vers le compte de l'entreprise au moyen d'une transaction par carte de crédit, sans frais d'installation ni frais mensuels.

Elle propose plusieurs solutions de paiements : par carte bancaire, via un portefeuille numérique (ex : Apple Pay et Google Pay), par paiement automatique ou encore par prélèvements et virements bancaires. De plus, il est possible de l'utiliser pour des transactions à l'internationale, sans frais supplémentaires pour le client.

N'ayant jamais mis en place ce type de fonctionnalité, j'ai choisi Stripe car c'est une solution de paiement fiable avec un niveau de sécurité élevé qui, surtout est facile et rapide à mettre en place.

b. Librairies

Une librairie est un ensemble de fonctions et de classes déjà codées dans un langage spécifique mis à disposition, ainsi un développeur peut y récupérer les fonctions qu'il souhaite et les intégrer dans son projet.

Jquery est une bibliothèque JavaScript libre et multiplateforme créer pour faciliter l'écriture de scripts côté client. Je l'ai utilisé pour créer une galerie photo, ainsi que pour mettre en place l'API de Stripe.

c. Framework

Un Framework est un cadre de travail, une infrastructure qui suit un schéma de fonctionnement et qui souvent inclut des bibliothèques. Son utilisation permet de gagner un gain de temps considérable.

Bootstrap est un framework CSS, il m'a permis de créer et d'organiser, facilement et rapidement, la partie front-end de mon projet. J'ai choisi ce Framework CSS, car il est d'une part très populaire, de l'autre il met à disposition beaucoup de compostants. De plus, Symfony permet d'intégrer des thèmes Bootstrap pour les formulaires.

Symfony : est un ensemble de composant PHP ainsi qu'un framework MVC libre écrit en PHP. Il fournit des fonctionnalités modulable et adaptables qui permettent de faciliter et d'accélérer le développement d'un site web.

Twig : est un moteur de template souple, rapide et sécurisé pour le langage de programmation PHP, il est utilisé par défaut par le framework Symfony.

Doctrine : il s'agit d'un Object Relational Mapping. Il est utilisé par défaut par le framework Symfony.

Composer : est un logiciel de gestionnaire de dépendances libre écrit en PHP. Il permet de gérer les dépendances d'un projet. Il m'a beaucoup été utile notamment pour installer et mettre à jour des bundles tel que knp-paginator.

J'ai choisi d'utiliser le framework Symfony pour réaliser mon projet, principalement car nous l'avons utilisé au cours de la formation, ce qui m'a permis de gagner du temps dans le développement de mon projet. Ce framework possède de nombreux avantages notamment la sécurité native à Symfony ainsi qu'une facilité d'utilisation grâce à sa documentation complète et le soutien de sa communauté.

III. Principe de fonctionnement de Symfony

1. Présentation du design pattern MVC

Un design pattern, patron de conception en français, est la conception d'une idée abstraite de résolution d'un problème récurrent. Il permet d'accélérer le processus de développement et d'améliorer la lisibilité du code.

Symfony se présente comme framework possédant une architecture Model View Controller (MVC). Cela nous permet de séparer 3 points essentiels :

- La couche Model, son rôle est de récupérer et gérer les données brutes de la base de données à l'aide de requêtes DQL.
- La couche Controller, qui gère la logique relative aux traitements des demandes, sert également à effectuer des vérifications et des autorisations.
- La couche View, correspond à l'affichage côté client.

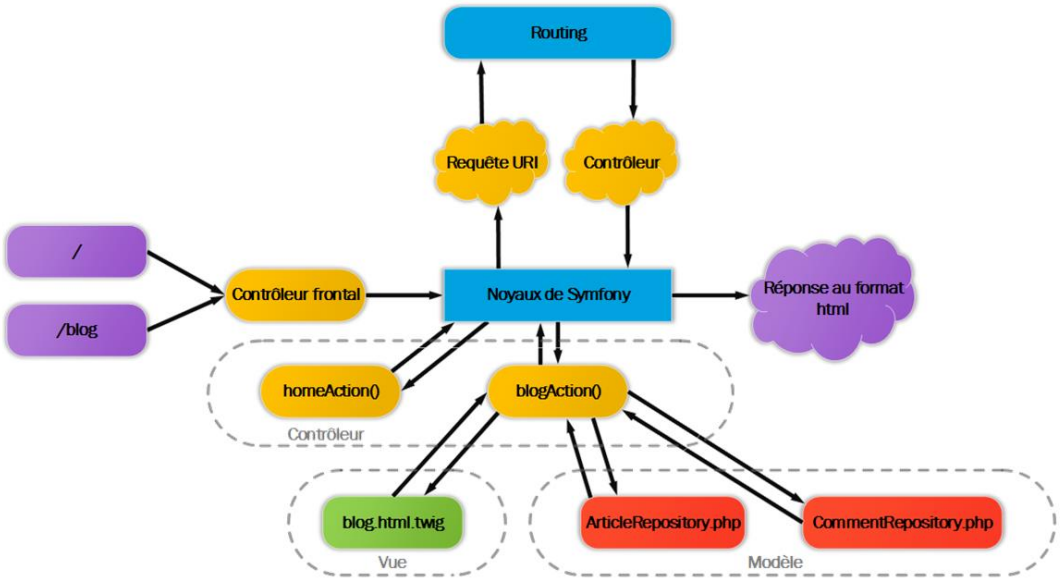


Figure 1 Structure de Symfony

Voici de manière plus précise et schématique la structure de Symfony (*Figure 1 ci-dessus*). Sur demande de l'utilisateur, le client envoie une requête http au serveur, elle est traitée par le contrôleur frontal de Symfony (index.php) avant d'être envoyée au noyaux. Lorsque celui-ci reçoit la demande, il fait appel au service de routing. Ce dernier indique au noyau quel contrôleur il faut appeler pour répondre à la demande. Le noyau fait donc appel au bon contrôleur. Ce contrôleur demande les données au modèle qui traduit cette demande par une requête DQL, il récupère les données et les renvois au contrôleur. Le contrôleur génère la vue, puis il retourne le résultat au noyaux de Symfony qui transmettra la réponse HTML au client.

Cependant, il serait plus logique de parler d'une structure MVP (Model View Presenter), ce design pattern est une variante du MVC, il ne met pas en relation la vue avec le modèle alors que dans un MVC c'est normalement le cas.

Voici un exemple plus concret du MVC dans mon projet. Le client envoie une requête http qui fait appel à cette méthode du contrôleur (*Figure 2 ci-dessous*) pour traiter la demande de l'utilisateur. Dans cet exemple, l'utilisateur en l'occurrence l'administrateur, souhaite avoir accès à toutes les commandes qui ont été payées. Le Contrôleur fait alors appel au « OrderingRepository » ($or) qui correspond au Modèle

```php
public function index(OrderingRepository $or, Request $request,
{

    $donnees = $or->findByPaid();
    $orderings = $paginator->paginate(
        $donnees, // Requête qui contient les données
        $request->query->getInt('page', 1), // Numéro de la pag
        5 // Nb de résultat par page
    );
    $productMostSold = $pr->findMostSold(); // [[]] on récupère
    $array = [];
    foreach($productMostSold as $productLine){
        //on récupère le produit par son nom
        $product = $pr->findOneByName($productLine['name']);
        // on récupère l'id du domaine
        $domainId = $product->getDomain()->getId();
        // et récupère le nom du domaine
        $domain = $dr->findOneById($domainId);

        $array[] = [
            'productName' => $productLine['name'],
            'domain' => $domain->getName(),
            'quantity' => array_pop($productLine)
        ];
    }
    return $this->render('admin/index.html.twig', [
        'orders' => $orderings,
        'productMostSold' => $array
    ]);
}
```

Figure 2 Exemple de Controller

Le Modèle (*Figure 3 ci-dessous*) traite la demande du contrôleur en récupérant les données, puis les retourne au contrôleur.

```php
    public function findByPaid(){
        return $this->createQueryBuilder('o')
        ->where('o.orderingStatus != :id')
        ->setParameter('id', '0')
        ->orderBy('o.createdAt', 'DESC')
        ->getQuery()
        ->getResult()
        ;

    }
```

Figure 3 Exemple de Model

Le Contrôleur retourne alors une vue (*Figure 4 page suivante*) avec les données.

```
<table class="table table-responsive table-striped">
    <thead>
        <tr>
            <td>Référence</td>
            <td>Utilisateur</td>
            <td>Adresse de livraison</td>
            <td>Adresse de facturation</td>
            <td>Statut de la commande</td>
            <td>Produits commandés</td>
            <td></td>
        </tr>
    </thead>
    <tbody>
        {% for order in orders %}
            {% if order.facture is not null %}
                <tr>
                    <td>Commande : {{ order.orderingReference }} Facture : {{ order.facture.factureReference }}</td>
                    <td>{{ order.user.email }}</td>
                    <td>{{ order.shipAddress }}</td>
                    <td>{{ order.facture }}</td>
                    <td>
                        {% if order.orderingStatus == 1 %}
                            <a href="{{ path('statusToSend', {'id':order.id}) }}">{{ order.getStatusOrder }}</a>
                        {% else %}
                            {{ order.getStatusOrder }}
                        {% endif %}
                    </td>
                    <td>
                        {% for productOrder in order.productOrderings %}
                            {{ productOrder.product.reference ~ " " ~ productOrder.product.name ~
                            " domaine " ~ productOrder.product.domain.name ~ " " ~ productOrder.product.year ~
                            " x" ~ productOrder.quantity }} <hr>
                        {% endfor %}
                        Total: {{ order.getTotal|number_format(2,',') }}€
                    </td>
                    <td>
                        <a href="{{ path('dl_facture', {'id': order.id}) }}">Télécharger la facture</a>
                    </td>
```

Figure 4 Exemple de View

2. Gestion des utilisateurs

Symfony nous permet de créer rapidement et simplement un système de gestion des utilisateurs.

a. Utilisateur

Grâce au MakerBundle, nous pouvons utiliser la commande « php/bin console make :user » qui nous permet de créer une entité User, elle possédera les propriétés suivantes par défaut un identifiant unique tel que l'e-mail, un mot de passe ainsi qu'un rôle. Lorsqu'une entité est créée avec la console, le repository de l'entité est également créer (il correspond au Modèle (de l'architecture MVP)).

Par défaut, cette commande créera une classe « User Provider », qui permettra notamment de recharger l'utilisateur à partir de la session.

La méthode de hachage des mots de passe des utilisateurs est définie dans le fichier security.yaml (*Figure 5 ci-dessous*), dans notre cas l'encodage est laissé par défaut en « auto », car il permet de sélectionner le meilleur encodeur possible selon Symfony.

```
security:
    encoders:
        App\Entity\User:
            algorithm: auto
```

Figure 5 : Fichier security.yaml section Encoders

Actuellement, l'algorithme par défaut utilise Argon2id pour hacher les mots de passe. Il s'agit d'une méthode de hachage récente et sécurisée. Lorsque l'on récupère le mot de passe en clair de l'utilisateur, on le hache avant de le stocker en base de données. Lorsque l'utilisateur souhaite se connecter, on récupère son mot de passe en clair on l'encode à nouveau puis on compare les deux mots de passe hacher pour voir s'ils correspondent. La méthode de hachage a été conçu pour qu'une chaine de caractère, une fois hacher, correspondent toujours à la même valeur, mais qu'il soit impossible de pouvoir récupérer la chaine de caractère à partir d'un mot de passer hacher.

b. Inscription

Pour créer un formulaire d'inscription Symfony fournit également, grâce au MakerBundle, la commande « php bin/console make :registration-form » permettant de générer un contrôleur et un formulaire d'inscription.

c. Authentification

Symfony nous permet de générer un formulaire d'authentification facilement avec la commande « php bin/console make:auth » dans la console. Cette commande génère un « SecurityController » qui contient les méthodes pour se connecter et se déconnecter, ainsi qu'une vue qui inclut un formulaire HTML basique.

La section « Firewall » du fichier security.yaml (*Figure 6 ci-dessous*), permet de définir comment les utilisateurs s'authentifieront.

```yaml
firewalls:
    dev:
        pattern: ^/(_(profiler|wdt)|css|images|js)/
        security: false
    main:
        anonymous: true
        lazy: true
        provider: app_user_provider
        guard:
            authenticators:
                - App\Security\LoginFormAuthenticator
        logout:
            path: app_logout
            # where to redirect after logout
            target: app_login
```

Figure 6: Fichier security.yaml section Firewall

« Anonymous : true » définit que l'utilisateur peut accéder au site de façon anonyme sans se connecter. « Guard » définit le mécanisme d'authentification utilisé, ici nous utilisons un formulaire de connexion. « logout » permet de choisir quel chemin déclenchera la déconnexion et vers quel chemin l'utilisateur sera rediriger.

d. Autorisation

La dernière partie du fichier security.yaml (*figure 7* ci-dessous) concerne les autorisations. Elle définit une protection de sécurité des URL de mon application. Autrement dit pour une route qui commence par « /admin » uniquement les utilisateurs authentifiés et possédant le rôle admin pourront y avoir accès, les autres utilisateurs auront l'accès refusé. Il en va de même pour les routes qui commence par « /profil », seuls les utilisateurs ayant le rôle User pourront y accéder, l'accès sera refusé aux autres utilisateurs anonymes.

```
34      access_control:
35          - { path: ^/admin, roles: ROLE_ADMIN }
36          - { path: ^/profil, roles: ROLE_USER }
37
```

Figure 7: Fichier security.yaml section access_control

Ce schéma (*Figure 8 ci-dessous*) résume très bien le fonctionnement de l'authentification et de l'autorisation. Dans un premier temps, on cherche à savoir qui (authentification) puis on cherche à savoir quel rôle il possède (autorisation).

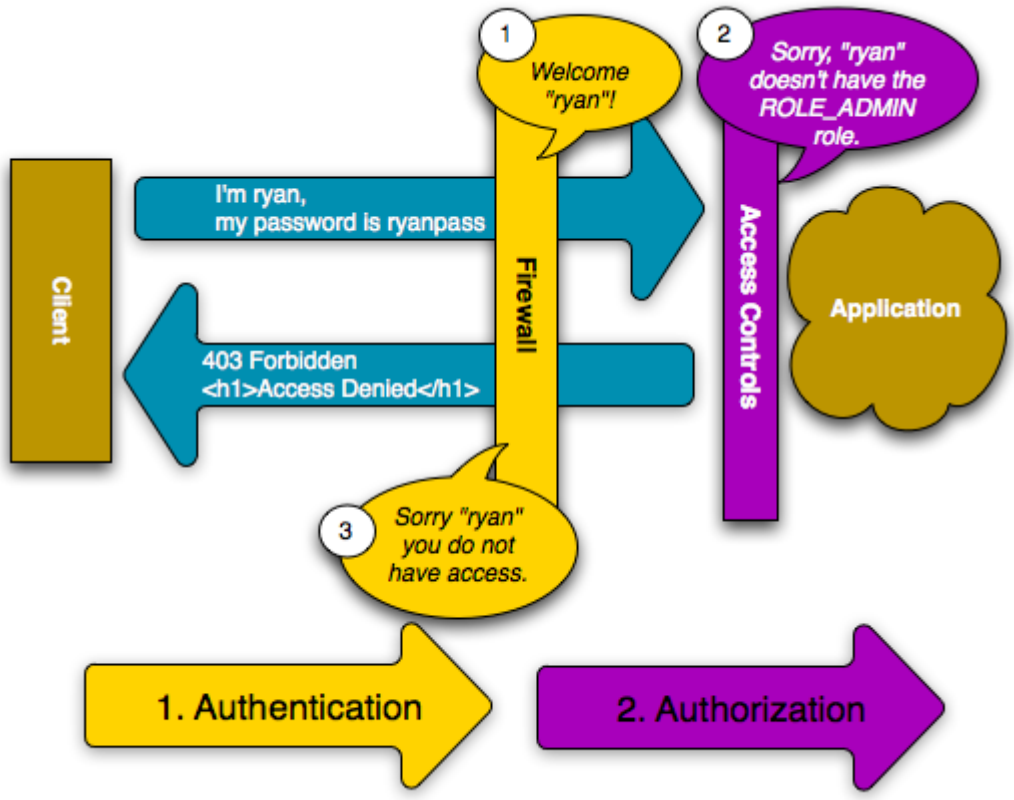


Figure 8 Schéma Firewall

3. Sécurité Native

Symfony intègre des fonctionnalités de sécurité, ce qui le rend moins vulnérable faces aux failles et attaques. Voici trois failles que Symfony gère nativement.

a. Faille XSS

Le Cross Site Scripting, c'est une méthode permettant d'injecter du contenu dans une page. Cette faille peut être évitée en utilisant les contraintes de validation dans les

formulaires. (Cependant, pour les données complexes qui peuvent accepter certains caractères spéciaux il faut utiliser data transformer). Symfony utilise par défaut l'échappement des données en sortie du moteur de template Twig. En dehors de Symfony, on peut se protéger avec les fonctions php htmlspecialchars() et htmlentities() qui filtre respectivement les chevrons et les entités html.

b. Injection SQL

L'Injection SQL est une méthode permettant d'injecter du code SQL dans un formulaire par exemple. Entre autres, elle peut permettre à un utilisateur malveillant de récupérer des données ou d'exécuter du code SQL afin de supprimer toute la base de données. L'échappement des données avec Twig et les contraintes de validation de Symfony permettent de filtrer les champs des formulaires. Elles sont à mettre en place pour chaque champ de formulaire. Grâce à Doctrine, lors de requêtes personnalisées, il faut utiliser « setParameter », elles deviennent alors des requêtes paramétrées et évitent les injections SQL. En dehors de Symfony, on utilise les fonctions de PHP htmlentities() et htmlspecialchars() qui permettent de filtrer les données. PDO (PHP Data Objects), constitue une couche d'asbtraction qui intervient entre l'application PHP et un système de gestion de base de données, applique un filtre pour vérifier le type du paramètre et échappe les caractères spéciaux.

c. Faille CSRF

Le CSRF (Cross Site Request Forgery), il s'agit d'une attaque où un utilisateur malveillant effectue une action visant un site ou une page précise en utilisant l'utilisateur comme déclencheur, à son insu.

Symfony se prémunit de cette attaque en intégrant dans tous ses formulaires un CSRFToken qui sera masquer pour l'utilisateur. Ce token contient une valeur qui est vérifié lors de la soumission de ce formulaire.

Plus en détail (*Figure 9 ci-dessous*), on génère une chaine de caractère hacher aléatoire et unique qui correspond à notre Token, il est stocké dans la session de l'utilisateur, et dans un champ caché du formulaire.

Lorsque l'utilisateur soumet le formulaire, on vérifie que le jeton de l'utilisateur correspond au jeton du formulaire. Cela permet de vérifier que l'utilisateur qui tente d'exécuter la page est bien passé par le formulaire avant, où on lui a délivré le jeton.

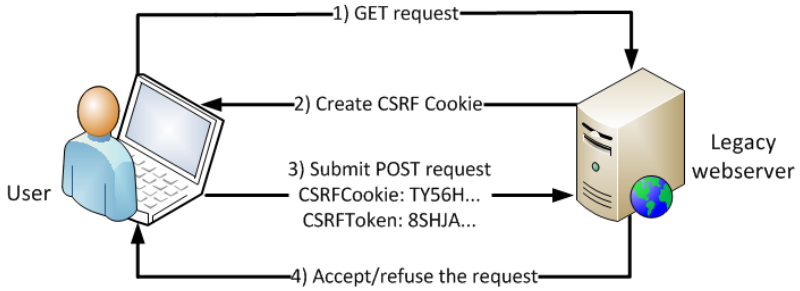


Figure 9 Schéma fonctionnement CSRFToken

IV. Gestion ou Organisation du projet

La première étape du projet a été de définir le cahier des charges. Autrement dit, lister toutes les fonctionnalités que mon applications pourrait et devrait avoir. En suivant le concept de Minimum Viable Product, j'ai trié les fonctionnalités essentielles et non essentielles de mon application. Ensuite, j'ai réalisé le MCD afin de concevoir une base de données. Avant de commencer à coder, j'ai conçu les maquettes de mon application.

En termes d'organisation, je me suis appuyé sur un outil qu'il m'est arrivée d'utiliser auparavant, « un cahier de laboratoire ». En chimie il est utilisé lors de projets, il doit contenir chaque jours ce qui a été réalisé, et les résultats obtenus. J'ai réadapté le concept en tenant un cahier en notant chaque jour l'avancée du projet, les erreurs rencontrées, comment elles ont été résolues, les idées que j'ai eu lors du développement, les tâches qui me restait à accomplir parfois en les numérotant afin de prioriser les tâches, etc…

J'ai préféré utiliser le format papier au profit de Trello car je me sentais plus à l'aise avec pour suivre l'avancée du projet. De plus il m'a vraiment permis de me vider l'esprit en notant tout sur ce cahier afin de travailler efficacement.

Donc, j'ai commencé par coder les parties essentielles de mon application, soit la partie concernant les utilisateurs, les produits, le panier et le mode de paiement.

V. Conception & Développement

Pour concevoir ma base de données, j'ai utilisé la méthode Merise, il s'agit d'une méthode d'analyse, de conception et de gestion de projet informatique. Cette méthode m'a permis de réaliser un MCD.

1. MCD/MLD

Un Modèle Conceptuel de Données est la représentation la plus abstraite des données d'un système d'information. Les données sont représentées sous forme d'entités et d'associations entre entités.

J'ai réalisé mon MCD à l'aide de Jmerise (*Figure 20 en annexe page 36*). Chaque produit possède une appellation, un type et un domaine. L'utilisateur contient les données essentielles, le mail comme moyen d'authentification unique, un mot de passe ainsi qu'un rôle afin de différencier les utilisateurs de l'administrateur.

L'utilisateur doit être en mesure d'ajouter des produits dans son panier qui se trouve en session. Il peut avoir plusieurs adresses de livraison. L'utilisateur peut commander plusieurs produits en choisissant leurs quantités, la facture sera générée lorsque l'utilisateur aura payé sa commande.

Le Modèle Logique de Données est la représentation des données d'un système d'information. Jmerise génère le MLD (*Figure 21 en annexe page 37*) à partir du MCD, il met en évidence les relations existantes entres les entités et ajoute toutes les clés étrangère pour chaque entité.

La génération du MLD nous permet de mieux concevoir la relation entre les entités « Order » et « Product », la relation est devenue une table de jointure et correspond donc à une nouvelle entité. On peut également voir une relation de type 1,1 entre les entités « Order » et « Facture », en effet une facture n'appartient qu'à une commande et inversement, une commande ne possède qu'une facture.

Le MCD et le MLD m'ont permis de modéliser les données en vue de préparer la base de données et de vérifier que le projet était viable et ainsi de bien démarrer mon projet. En outre cela m'a permis de bien comprendre les relations entre mes entités.

2. Création de la base de données

Le Framework Symfony utilise l'ORM Doctrine par défaut. Un Object Relational Mapping est une technique de programmation faisant le lien entre la base de données et la programmation orienté objet. Les entités peuvent être créées par Doctrine à partir de la base de données et inversement, la base de données peut être créée par Doctrine à partir des entités existantes. Pour ce projet, c'est ce dernier point que j'ai utilisé pour créer ma base de données.

Pour créer la base de données il faut avant tout créer les entités avec la commande « php bin/console make:entity ». Cette commande pose plusieurs questions concernant l'entité, il est ainsi possible de la configurer comme on le souhaite, et faire des relations entre nos entités.

Il ne faut pas oublier de configurer la base de données dans le fichier .env de notre dossier Symfony.

Doctrine permet également d'écrire des requêtes simplifiées avec le « query builder » en DQL(Doctrine Query Language).

3. Maquettes

Une maquette est une représentation graphique d'un site web, présenté sous forme statique à un client, elle permet d'avoir un premier rendu d'un site web en indiquant l'emplacement des blocs afin de concevoir la structure d'un site. Elle permet également d'avoir une première idée de l'interface utilisateur (UI) et de l'expérience utilisateur (UX).

Pour ce projet, j'ai choisi de réaliser mes maquettes avec Adobe XD, il s'agit d'un logiciel gratuit et complet qui permet de réaliser des maquettes pour tout type de formats, très simplement et rapidement.

Tout d'abord , j'ai mis en place le concept de Mobile First, qui consiste à concevoir un site en mettant la priorité sur la version mobile et en adaptant progressivement le design pour les écrans plus large. J'ai donc réalisé les maquettes de mon site web pour le format mobile (*Figure 22 & 23 en annexe page 38*). J'ai essayé de concevoir mes maquettes de la façon la plus ergonomique et la plus intuitive possible pour l'utilisateur. Ensuite, j'ai réalisé les maquettes pour la version desktop (*Figure 24 en annexe page 39*).

Lors du développement de mon projet, mes maquettes m'ont grandement facilité la tâche en termes d'intégration web. Le site web final est entièrement responsive et adaptable sur toutes les plateformes. Lors de la mise en place du responsive, j'ai notamment utilisé les Media Queries, ce sont des spécifications de CSS3 qui permettent d'attribuer des propriétés CSS en fonction des largeur d'écran .

4. Description détaillée du code ou extrait de code

Les fonctionnalités les plus importantes de mon application sont celle qui sont utilisés lorsqu'un utilisateur effectue un achat.

Tout d'abord, l'utilisateur authentifié ou anonyme, a accès à la boutique où l'on retrouve tous les produits. J'ai donc créé un « StoreController » qui contiendra toute les méthodes relatives à la boutique. La fonction showProducts() (*Figure 10 ci-dessous*), permet d'afficher les produits, en fonction des filtres que l'utilisateur choisi.

```php
/**
 * @Route("/products/", name="products_index")
 *
 * Fonction permettant d'afficher la liste des produits, et de les filtrer
 *
 * @return Response
 */
public function showProducts(Request $request, ProductRepository $pr): Response
{
    $filter = New Filter();
    $filter->page = $request->get('page', 1);
    $formFilter = $this->createForm(FilterType::class, $filter);
    $formFilter->handleRequest($request);
    if($formFilter->isSubmitted() && $formFilter->isValid()){
        // Produit pour l'utilisateur
        $productsActive = $pr->findByFilterAndActivate($filter);
        // Produit pour l'administrateur
        $allProducts = $pr->findByFilter($filter);
    }
    $productsActive = $pr->findByFilterAndActivate($filter);
    $allProducts = $pr->findByFilter($filter);

    return $this->render('store/index.html.twig', [
        'allProducts' => $allProducts,
        'productsActive' => $productsActive,
        'formFilter' => $formFilter->createView()
    ]);
}
```

Figure 10 Fonction showProducts()

J'ai mis en place une entité « Filter » qui me permet de stocker et de récupérer les données des filtres, elle a les propriétés suivantes : appellations, types, domaines, min, max, page. Ainsi lorsque l'utilisateur soumet le formulaire de filtre, on récupère les produits avec la requête suivantes (*Figure 11 page suivante*).Cette requête écrite en DQL, récupère les produits activé, on joint les appellation, le domaine et le type. Ensuite, si le filtre n'est pas vide, on fait une requête où l'on récupère les ou l'appellation(s) contenu dans notre filtre. On procède ainsi pour tous les autres paramètres, afin d'obtenir des produits en fonction des filtres.

Finalement, on retourne la requête dans la méthode paginate() du bundle knp-paginator, qui nous permet de gérer facilement et simplement la pagination. Dans cette fonction, on récupère dans un premier temps les données, donc ici la requête. Ensuite, le numéro de la page, puis le nombre de produit par page.

Pour finir, le contrôleur renvoi les données de tous les produits, pour l'administrateur, ainsi que les données des produits activés, pour l'utilisateur.

```php
public function findByFilterAndActivate($filter)
{
    $query =  $this->createQueryBuilder('p')
        ->select('a', 'p', 'd', 't')
        ->andWhere('p.activate = :activate')
        ->setParameter('activate', 1)
        ->join('p.appellation', 'a')
        ->join('p.domain', 'd')
        ->join('p.type', 't')
    ;

    if(!empty($filter->appellations)){
        $query = $query
            ->andWhere('a.id IN (:appellations)')
            ->setParameter('appellations', $filter->appellations);
    }

    if(!empty($filter->domains)){
        $query = $query
            ->andWhere('d.id IN (:domains)')
            ->setParameter('domains', $filter->domains);
    }

    if(!empty($filter->types)){
        $query = $query
            ->andWhere('t.id IN (:types)')
            ->setParameter('types', $filter->types);
    }
    if(!empty($filter->min)){
        $query = $query
            ->andWhere('p.unitPrice >= :min')
            ->setParameter('min', $filter->min);
    }
    if(!empty($filter->max)){
        $query = $query
            ->andWhere('p.unitPrice <= :max')
            ->setParameter('max', $filter->max);
    }

    $query = $query->getQuery();
    return $this->paginator->paginate(
        $query,
        $filter->page,
        9
    );

}
```

Figure 11 Requête des produits en fonction des filtres

En même temps que l'utilisateur visionne les produits, il peut ajouter un ou plusieurs produits au panier et les retirer s'il le souhaite. J'ai donc créé un CartController qui gère les interactions entre le panier stocké en session (qui est définit dans le constructeur de mon contrôleur) et l'utilisateur. Ci-dessous (*Figure 12 ci-dessous*) la fonction pour ajouter un produit au panier.

```php
    /**
     * @Route("/cart/add/{id}", name="cart_add")
     *
     * Fonction permettant d'ajouter un produit au panier
     *
     * @param Request $request
     * @param Product $product
     * @param SessionInterface $session
     *
     * @return Response
     */
    public function add(Request $request, Product $product = null, SessionInterface
    {
        // Vérifier que le produit existe
        if(!$product){
            $this->addFlash('warning', 'Le produit n\'existe pas');
            return $this->redirectToRoute("products_index");
        }
        $qtt = $request->request->get("quantity");
        // Vérifier que la quantité de produit de dépasse pas la quantité en
        //stock et qu'elle est supérieur à 0

        if($qtt <= $product->getUnitStock() && $qtt > 0){

            $this->cart->add($product, $qtt);
            $session->set('cart', $this->cart);

            $this->addFlash('success', 'Le produit a été ajouté au panier');

            return $this->redirectToRoute("products_index");
        } else {
            $this->addFlash('warning', 'La quantité de produit que vous souhaitez
            ajouter au panier est insuffisante par rapport au stock');
            return $this->redirectToRoute("products_index");
        }
    }
```

Figure 12 fonction add() du CartController

La méthode add() contient l'identifiant du produit, ainsi le contrôleur sait quel produit ajouter au panier. Cette méthode vérifie que le produit existe. On récupère la quantité de produit ajouté et on vérifie quelle est bien supérieure à zéro mais également qu'elle est inférieure ou égale à la quantité de ce même produit en stock. Si ces deux points ne sont pas vérifiés on redirige l' utilisateur vers la liste des produits en indiquant l'erreur à l'aide d'un message flash. À l'inverse, on ajoute le produit dans le panier avec la quantité demandé.

Afin d'optimiser mon travail, j'ai créé une entité Cart qui me permet de gérer toutes les fonctionnalités de base d'un panier, tel que l'ajout d'un produit ou la suppression d'un produit. L'entité Cart fournit au contrôleur toutes les méthodes dont il a besoin concernant le panier. La méthode add() du contrôleur utilise la fonction add() de l'entité Cart (Figure *13 ci-dessous*).

```
25      public function add(Product $product, $qtt){
26          if(!array_key_exists($product->getId(), $this->incart)){
27              $this->incart[$product->getId()] = [
28                  "product" => $product,
29                  "quantity" => $qtt
30              ];
31          }
32          else{
33              $this->incart[$product->getId()]["quantity"]+=$qtt;
34          }
35      }
```

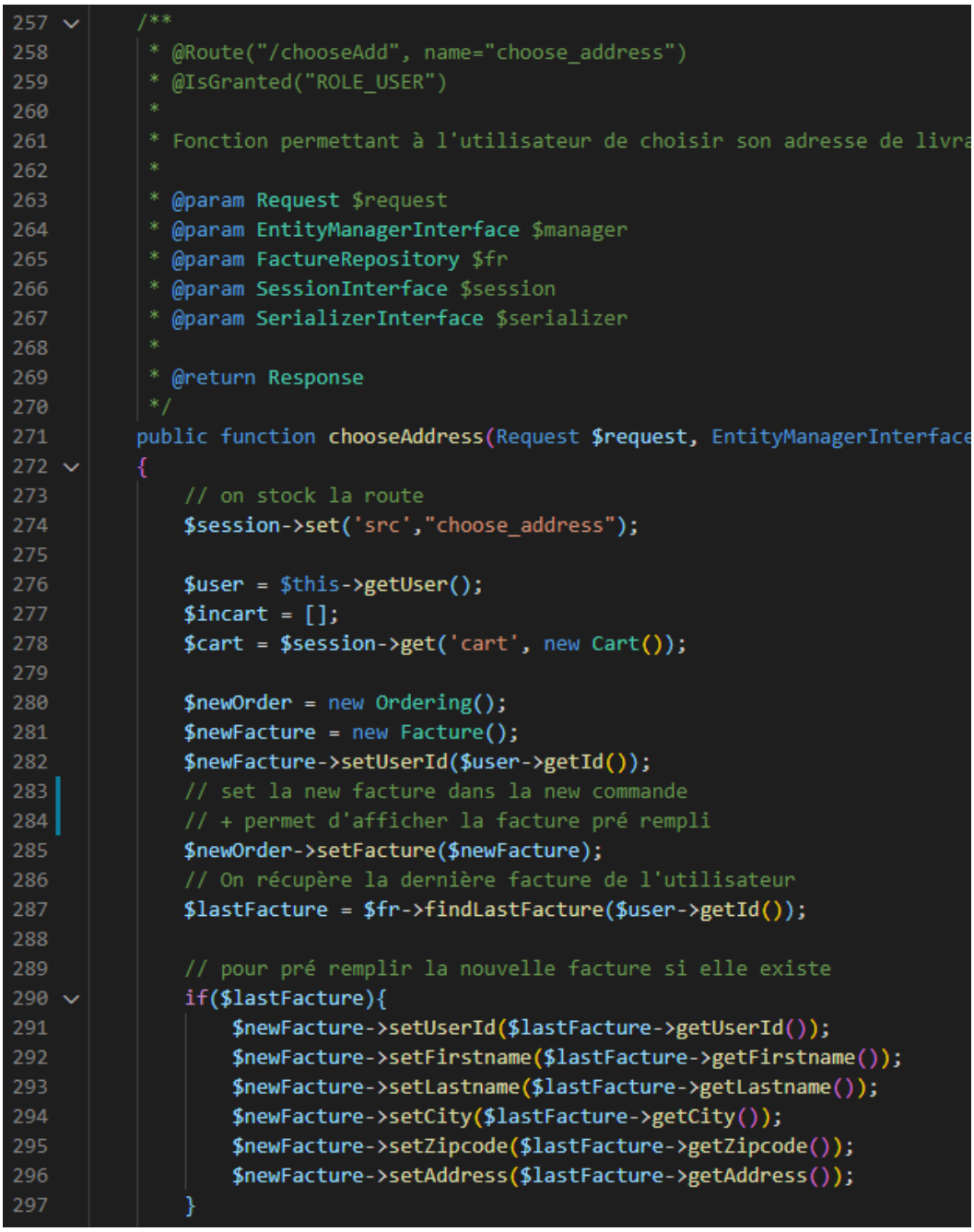
Figure 13 fonction add() de l'entité Cart

La méthode add() de l'entité Cart prend en paramètre le produit et la quantité qui sont fourni par le contrôleur. On vérifie que l'Id du produit existe dans le panier. Si ce n'est pas le cas on ajoute le produit. En revanche, s'il existe on rajoute la quantité souhaité à la quantité existante.

Lorsque l'utilisateur a choisi ses articles et qu'il souhaite procéder au paiement, il est, depuis le panier, envoyé sur une page pour choisir ces adresses de livraison et de facturation. Il s'agit de l'une des fonctions les plus importantes (*Figure 14 page suivante*).

Tout d'abord, on remarque l'annotation @IsGranted(''ROLE_USER'') sur cette méthode, cela permet uniquement aux utilisateurs authentifié d'avoir accès à cette page. C'est cette méthode qui nous permettra de créer une nouvelle commande et une nouvelle facture qui seront hydrater puis par la suite stockés en base de données.

Dans un premier temps, on stock la route sur laquelle on se trouve. Cela nous permet, par la suite, si l'utilisateur souhaite ajouter une nouvelle adresse avec la fonction addAddress(), de redirigé l'utilisateur d'où il vient. Car cette fonction peut être appelé ici, ainsi que dans le profil de l'utilisateur.



```php
/**
 * @Route("/chooseAdd", name="choose_address")
 * @IsGranted("ROLE_USER")
 *
 * Fonction permettant à l'utilisateur de choisir son adresse de livra
 *
 * @param Request $request
 * @param EntityManagerInterface $manager
 * @param FactureRepository $fr
 * @param SessionInterface $session
 * @param SerializerInterface $serializer
 *
 * @return Response
 */
public function chooseAddress(Request $request, EntityManagerInterface
{
    // on stock la route
    $session->set('src',"choose_address");

    $user = $this->getUser();
    $incart = [];
    $cart = $session->get('cart', new Cart());

    $newOrder = new Ordering();
    $newFacture = new Facture();
    $newFacture->setUserId($user->getId());
    // set la new facture dans la new commande
    // + permet d'afficher la facture pré rempli
    $newOrder->setFacture($newFacture);
    // On récupère la dernière facture de l'utilisateur
    $lastFacture = $fr->findLastFacture($user->getId());

    // pour pré remplir la nouvelle facture si elle existe
    if($lastFacture){
        $newFacture->setUserId($lastFacture->getUserId());
        $newFacture->setFirstname($lastFacture->getFirstname());
        $newFacture->setLastname($lastFacture->getLastname());
        $newFacture->setCity($lastFacture->getCity());
        $newFacture->setZipcode($lastFacture->getZipcode());
        $newFacture->setAddress($lastFacture->getAddress());
    }
```

Figure 14 Fonction chooseAddress() Partie 1

On récupère le panier en session, on instancie une nouvelle commande et une nouvelle facture. On hydrate la facture avec l'utilisateur connecté. On hydrate la nouvelle commande avec la nouvelle facture

On récupère la dernière facture de l'utilisateur et si elle existe, on hydrate la nouvelle facture avec les informations de l'adresse de l'ancienne facture , ce qui permettra par la suite de préremplir les champs du formulaire de l'adresse de facturation.

Ensuite (*Figure 15 page suivante*), on créer le formulaire relatif à l'entité Order où je demande les informations relatives aux adresses de livraison et de facturation. Si le formulaire est soumis et validé, on encode la nouvelle facture avec la méthode serialize() du SerializerInterface, il s'agit d'un composant de Symfony, qu'il a fallu installer à l'aide de Composer.

```
        $formOrder = $this->createForm(OrderType::class, $newOrder);
        $formOrder->handleRequest($request);

        if($formOrder->isSubmitted() && $formOrder->isValid()){
            // on encode la facture
            $jsonFacture = $serializer->serialize($newFacture, 'json',[AbstractNormalizer::IGNORED_ATTRIBUTES => ['id']]);
            // on stock la facture en session
            $session->set('facture', $jsonFacture);
            //on met la facture de la nouvelle commande à null
            $newOrder->setFacture(null);
            // on récupère l'user qu'on stock dans la facture
            $newOrder->setUser($this->getUser());

            // on persist à ce moment pour pouvoir addPorudctOrdering sur qq chose de "réel"
            $manager->persist($newOrder);
            // on récupère ce qu'il y a dans le panier
            // pour l'ajouter à la commande
            foreach($cart->getFullCart() as $cartLine){
                $newProductOrder = new ProductOrdering();
                $product = $this->getDoctrine()->getRepository(Product::class)->find($cartLine['product']->getId());
                $newProductOrder->setProduct($product);
                $newProductOrder->setQuantity($cartLine['quantity']);
                $newOrder->addProductOrdering($newProductOrder);
                $manager->persist($newProductOrder);
            }
            $manager->flush();
```

Figure 15 Fonction chooseAddress() Partie 2

Cette méthode permet donc à partir d'un objet de sérialiser c'est-à-dire de transformer un objet en un format spécifique. Ici j'ai choisi le format Json (JavaScript Object Notation), il s'agit d'un format de données textuelles et permet de représenter l'information structurée.

On stock la nouvelle facture au format Json dans la session, cela me permettra de la récupérer lorsque le paiement sera accepté et ainsi générer la facture au bon moment. De ce fait, je remplace la facture hydratée plutôt dans la nouvelle commande par null. J'hydrate l'utilisateur connecté dans la nouvelle commande et puis je persist la commande.

Ensuite pour chaque ligne de produit du panier, j'instancie un nouveau ProductOrder, et je l'hydrate pour chaque ligne du panier avec le produit et la quantité. J'hydrate la nouvelle commande avec chaque ligne de produit du panier, puis je persist chaque nouveau ProductOrder. Et pour finir je flush, à ce moment, je récupère en base de données, la nouvelle commande ainsi que les ProductOrdering.

```
324             // on récupère le panier pour l'afficher
325             foreach($cart->getFullCart() as $cartLine){
326                 $incart[] = [
327                     'product' => $cartLine['product'],
328                     'quantity' => $cartLine['quantity']
329                 ];
330             }
331             if(empty($incart)){
332                 return $this->render('bundles/TwigBundle/Exception/error404.html.twig');
333             }
334             $total = $cart->getTotal($incart);
335             return $this->render('checkout/index.html.twig', [
336                 'items' => $incart,
337                 'total' => $total,
338                 'order' => $newOrder,
339                 'facture' => $newFacture,
340                 'reference' => $newOrder->getOrderingReference(),
341             ]);
342         }
343         return $this->render('user/chooseAddresses.html.twig', [
344             'formOrder' => $formOrder->createView(),
345         ]);
346     }
```

Figure 16 Fonction chooseAddress() Partie 3

Pour finir avec cette fonction (*Figure 16 ci-dessus*), on récupère le panier, le total du panier et on vérifie qu'il n'est pas vide, si c'est le cas on renvoie sur la page d'erreur 404. On renvoie les données du panier, du total du panier, de la commande, de la facture afin de les afficher sur la prochaine page qui contiendra les informations récapitulatives de la commande avant de passer au paiement. On remarque sur ce dernier screen que l'on envoie la référence de la commande, cette information nous sera utile pour récupérer les informations de la commande notamment le nom du produit et la quantité du produit afin que ces données soit afficher lors du paiement avec Stripe.

La mise en place d'un mode de paiement me semblait indispensable pour une boutique en ligne. N'ayant jamais mis en place ce type de fonctionnalité, j'ai choisi Stripe pour ses recommandation en termes de facilité d'intégration et pour ses multiples moyens de paiement. J'ai mis en place Stripe à l'aide de la documentation qu'il fournit (la documentation a été utilisé pour la traduction Page 29).

La seule difficulté que j'ai rencontrée pour mettre en place Stripe, a été de récupérer les données de la commande afin de les récupérer dans cette méthodes. La solution a été de récupérer la référence de la commande dans la vue du récapitulatif de la commande et de la faire passer en paramètre (*Figure 17 page suivante*).

```
<script type="text/javascript">
    // Create an instance of the Stripe object with your publishable API key
    var stripe = Stripe('{{ public_key }}');
    var checkoutButton = $('#checkout-button');


    checkoutButton.on('click', function() {
        // Create a new Checkout Session using the server-side endpoint you
        // created in step 3.
        fetch("/create-checkout-session/{{ reference }}", {
            method: 'POST',
        })
        .then(function(response) {
            return response.json();
        })
        .then(function(session) {
            return stripe.redirectToCheckout({ sessionId: session.id });
        })
        .then(function(result) {
            // If `redirectToCheckout` fails due to a browser or network
            // error, you should display the localized error message to your
            // customer using `error.message`.
            if (result.error) {
            alert(result.error.message);
            }
        })
        .catch(function(error) {
            console.error('Error:', error);
        });
    });
</script>
```

Figure 17 Script Stripe

Lorsque la page du récapitulatif de commande s'affiche, un bouton « payer » se trouve en bas de la page. Lorsqu'un utilisateur click sur le bouton qui possède l'id « check-button » en l'occurrence le bouton payer, la méthode fetch effectue une requête AJAX(Asynchronous JavaScript XML) sur la route « /create-checkout-session/{{ reference }} » qui correspond à la méthode payment() de mon CheckoutController. C'est Stripe qui fournit dans sa documentation cette partie de Script.

Dans ce script, on a la promesse que la méthode fetch() nous retourne une réponse, cette réponse est retournée au format Json, cela permet de vérifier que c'est le bon utilisateur. Ensuite, cette réponse devient une session qui vérifie que la page de paiement est disponible. Pour finir, la fonction result permet de rediriger vers la page de paiement. Si cela échoue à cause du navigateur ou d'une erreur de serveur, la fonction renvoie une erreur à l'utilisateur.

La fonction catch, peut intervenir à n'importe quel moment à la suite de la méthode fetch(). La page de paiement ne s'affichera pas s'il y a la moindre erreur.

La méthode payment() (*Figure 18 ci-dessous*) permet notamment de configurer la page de paiement de Stripe. Dans un premier temps on récupère la commande avec la référence de la commande qui est passée par la route. On définit chaque ligne de produit avec 'price_data' qui appartient à Stripe. Ensuite on définit la session de Stripe avec le type de méthode de paiement, les lignes de produit, le mode de paiement et enfin les pages de succès et d'annulation où seront renvoyé les utilisateurs à la suite de leur paiement.

```php
public function payment($reference, OrderingRepository $or, EntityManager
{
    $order = $or->findOneByOrderingReference($reference);

    $YOUR_DOMAIN = 'http://127.0.0.1:8000';
    $productsForStripe = [];

    // Pour chaque ligne de produit dans le panier
    // On récuppère le nom du produit et son prix
    // Et la quantité de produit
    foreach($order->getProductOrderings()->getValues() as $cartLine){
        $productsForStripe[] = [
        'price_data' => [
            'currency' => 'eur',
            'product_data' => [
            'name' => $cartLine->getProduct()->getName(),
            ],
            'unit_amount' => $cartLine->getProduct()->getUnitPrice()*100,
        ],
        'quantity' => $cartLine->getQuantity(),
        ];
    }

    Stripe::setApiKey( $this->getParameter('app.secretKey'));
    $checkout_session = \Stripe\Checkout\Session::create([
    'payment_method_types' => ['card'],
    'line_items' => [[
        $productsForStripe
    ]],
    'mode' => 'payment',
    'success_url' => $YOUR_DOMAIN.'/success/{CHECKOUT_SESSION_ID}',
    'cancel_url' => $YOUR_DOMAIN.'/error/{CHECKOUT_SESSION_ID}',
    ]);

    $order->setStripeSessionId($checkout_session->id);
    $manager->flush();
    return new JsonResponse(['id' => $checkout_session->id]);
}
```

Figure 18 Fonction payment()

Pour terminer, on stock l'identifiant de la session de Stripe en base donnée dans l'entité Order. Cet identifiant passera dans l'URL de succès ou d'annulation à la suite du paiement. Cette méthode retourne l'identifiant de session de Stripe au format Json.

Dans le cas d'un paiement effectué avec succès, Stripe renvoie sur l'URL définit plus tôt. Cette route contenant l'identifiant de session de Stripe renvoie à la méthode success() (*Figure 19 ci-dessous*).

Dans cette méthode, on récupère les informations de la commande, grâce à l'identifiant de session de Stripe. On vérifie que l'utilisateur authentifié correspond à l'utilisateur de la commande, sinon on le redirige vers la page des produits. Dans un premier temps, on récupère la facture stockée en session, et on la désérialise en objet Facture. Ensuite on hydrate la facture avec la commande et inversement puis on persist les deux et on flush pour stocker en base de données.

```php
public function success($stripeSessionId, OrderingRepository $or,  SessionInte
{

  $order = $or->findOneByStripeSessionId($stripeSessionId);
  // On vérifie que l'utilisateur est le bon
  if($this->getUser() != $order->getUser()){
    return $this->redirectToRoute('products_index');
  }
  $factureJson = $session->get('facture');
  $facture = $serializer->deserialize($factureJson, Facture::class, 'json');
  $facture->setOrdering($order);
  $order->setFacture($facture);
  $manager->persist($facture);
  $manager->persist($order);
  $manager->flush();

  $total = $order->getTotal();
  $quantityTotal = $order->getQuantityTotal();

  //je passe le status de la commande à payé

  if($order->getOrderingStatus() == 0){
    $order->setOrderingStatus(1);
    $manager->flush();

    //je retire des stock la qte de produit qui a été vendu
    foreach($order->getProductOrderings() as $productLine){
      $product = $productLine->getProduct();
      $quantity = $productLine->getQuantity();
      $value = $product->getUnitStock() - $quantity;
      $product->setUnitStock($value);
    }
    $manager->flush();

    $cart = $session->get('cart', new Cart());
    //je vide le panier
    $cart->clear();
    return $this->render('checkout/success.html.twig', [
```

Figure 19 Fonction success()

On récupère le total de la commande et la quantité total de produit dans la commande.

Dans un second temps, on vérifie que le statut de la commande correspond à « en attente de paiement », si c'est le cas on le modifie en le passant à 1 qui correspond à « payée », cette valeur est stockée en base de données.

Ensuite, on retire des stock la quantité de produit qui a été vendu, c'est-à-dire que pour chaque ligne de produit de la commande, on récupère le produit et sa quantité, et on soustrait le stock du produit à la quantité du produit, on réhydrate la nouvelle valeur en base de données.

Pour finir, on récupère le panier et on le vide. Puis on renvoie la vue qui contient le récapitulatif de la commande avec les données suivantes : commande, total et quantité total de produit.

Cette page contiendra entre autres un bouton « télécharger la facture », qui permettra à l'utilisateur de télécharger sa facture s'il le souhaite. Si toutefois il oubli, il pourra à nouveau y accéder dans son profil.

En cas d'erreur lors du paiement de Stripe, l'API Stripe redirige l'utilisateur vers l'URL que je lui ai fourni. Cette route renvoie vers la méthode error() qui passe le statut de la commande à « erreur lors du paiement » et renvoie une vue. Cette page contient l'erreur afin d'informer l'utilisateur, ainsi qu'un bouton qui le redirige vers son panier dans le but de renouveler sa commande.

VI. Traduction d'un Extrait Anglophone

Stripe/Documentation/checkout/integration-builder -- https://stripe.com/docs/checkout/integration-builder

Accept a payment

Set up the server

Install the Stripe PHP library

Install the library with composer and initialize with your secret API key. Alternatively, if you are starting from scratch and need a composer.json file, download the files using the Download link in the code editor.

Install the library:

composer require stripe/stripe-php

Accepter un paiement

Installer le serveur

Installer la bibliothèque PHP de Stripe

Installer la bibliothèque avec composer et initialisez là avec votre clé API privée. Sinon, si vous utilisez scratch et avez besoin d'un fichier composer.json, téléchargez les documents avec le lien de téléchargement dans l'éditeur de code.

Create a Checkout Session

Add an endpoint on your server that creates a Checkout Session. A Checkout Session controls what your customer sees in the Stripe-hosted payment page such as line items, the order amount and currency, and acceptable payment methods. Return the Checkout Session's ID in the response to reference the Session on the client.

Créer une session de paiement

Ajoutez une route sur votre serveur qui crée une session de paiement. Une session de paiement contrôle ce que votre client voit sur la page de paiement hébergée par Stripe, tel qu'une ligne de produits, la quantité et la devise monétaire de la commande, et une méthode de paiement acceptable. Retournez l'ID de la session de paiement dans la réponse pour référencer la session du client.

Specify payment methods

Checkout supports several payment methods beyond cards. If multiple payment methods are passed, Checkout dynamically reorders them to prioritize the most relevant payment methods based on the customer's location and other characteristics. If you accept cards as a payment method, Apple Pay and Google Pay are displayed in Stripe Checkout when applicable.

Définir les méthodes de paiement

Le paiement avec Stripe supporte plusieurs méthodes de paiement en plus de celle par carte bancaire. Si de multiples méthodes de paiement peu utilisées, le paiement avec Stripe les réorganise dynamiquement afin de prioriser la méthode de paiement la plus pertinente basée sur l'emplacement du client ou d'autres caractéristiques. Si vous acceptez les cartes bancaires

comme une méthode de paiement, Apple Pay et Google Pay sont affichés dans la vérification de Stripe lorsque cela est possible.

Define the line items

Always keep sensitive information about your product inventory, like price and availability, on your server to prevent customer manipulation from the client. Define product information when you create the Checkout Session with price_data or alternatively use pre-defined prices and pass their IDs.

Définir la ligne de produits

Conserver toujours les informations sensibles sur vos produits, comme le prix et la disponibilité, sur votre serveur afin d'éviter que le client les manipule. Définissez les informations du produit quand vous créez la session de paiement avec price_data ou alors utilisez les prix prédéfinis et passez leurs ID.

Choose the mode

Checkout has three modes: payment, subscription, or setup. Use payment mode for one time purchases. Learn more about subscription and setup modes in the docs.

Choisir le mode

Il existe trois modes de paiement : le paiement, l'abonnement ou l'installation. Utilisez le mode paiement pour un achat unique. Apprenez en plus sur les modes abonnement et installation dans les documents.

Supply the redirect URLs

Specify URLs for success and cancel pages—make sure they are publicly accessible so Stripe can redirect customers to them. You can also handle both the success and canceled states with the same URL.

Fournir les URL de redirection

Fournir les URL pour les pages de confirmation et d'annulation –assurez-vous qu'elles sont publiques pour que Stripe puisse rediriger le client vers-elles. Vous pouvez également manipuler les deux états (confirmation et annulation) avec le même URL.

Build your checkout

Add a success page

Create a success page for the URL you provided as the Checkout Session success_url to display order confirmation messaging or order details to your customer. Stripe redirects to this page after the customer successfully completes the checkout.

Construit ton paiement

Ajouter une page de confirmation

Créer une page de confirmation pour l'URL que vous avez fourni en tant que session de paiement success_url pour afficher le message de confirmation de commande ou les détails de la commande à votre client. Stripe redirige vers cette page une fois que le client a terminé le paiement avec succès.

Add a canceled page

Add another page for cancel_url. Stripe redirects to this page when the customer clicks the back button in Checkout.

Ajouter une page d'annulation

Ajoutez une autre page pour cancel_url. Stripe redirige sur cette page lorsque l'utilisateur clique sur le bouton retour sur la page de paiement.

Add an order preview page

Finally, add a page to show a preview of the customer's order. Allow the customer to review or modify their order—once a customer is sent to the Checkout page, the order is final and cannot be modified without creating a new Checkout Session.

Ajouter une page aperçue de commande

Enfin, ajoutez une page pour afficher l'aperçu de la commande du client. Cela permet au client de revoir ou modifier sa commande—une fois que le client est envoyé sur la page de paiement, la commande est définitive et ne peut être modifiée sans créer une nouvelle session de paiement.

Load Stripe.js

Stripe Checkout relies on Stripe.js, Stripe's foundational JavaScript library for collecting sensitive payment information and advanced fraud detection. Always load Stripe.js from js.stripe.com to remain compliant. Do not include the script in a bundle or host it yourself.

Charger stripe.js

Le paiement avec Stripe repose sur stripe.js, la bibliothèque JavaScript de du fondement de Stripe pour collecter les informations sensibles de paiement et détecter les fraudes avancées. Toujours charger stripe.js depuis js.stripe.com pour rester conforme. Ne pas inclure le script dans un paquet ou l'héberger vous-même.

Add a checkout button

Add a button to your order preview page. When your customer clicks this button, they're redirected to the Stripe-hosted payment page.

Ajouter un bouton de paiement

Ajoutez un bouton à votre page aperçu de commande. Lorsque votre client cliquera sur ce bouton, il sera redirigé vers la page de paiement hébergée par Stripe.

Initialize Stripe.js

Initialize Stripe.js with your publishable API key.

Initialiser Stripe.js

Initialiser Stripe.js avec votre clé API public.

Fetch a Checkout Session

Make a request to the endpoint on your server to create a new Checkout Session when your customer clicks on the checkout button.

Chercher une session de paiement

Faites une requête vers la route de votre serveur pour créer une nouvelle session de paiement lorsque le client clique sur le bouton paiement.

Redirect to Checkout

Call stripe.redirectToCheckout with the ID of the Checkout Session to redirect the customer to the Stripe Checkout page.

Rediriger vers le paiement

Appelez stripe.redirectToCheckout avec l'ID de la session de paiement pour rediriger le client vers la page de paiement de Stripe.

VII. Axes d'améliorations

Cette application a atteint son objectif de Minimum Viable Product et même plus puisque j'ai pu mettre en place des fonctionnalités non essentielles. Néanmoins, elle peut encore être améliorée.

Fonctionnalités supplémentaires :

- Mettre en place un système de mailer avec Swift Mailer, afin de vérifier l'inscription et la connexion, d'envoyer la facture par mail, de pouvoir mettre en place un système de newsletter et/ou prévenir lorsqu'un produit est à nouveau disponible.
- Mettre en place un formulaire de contact
- Mettre en place de l'ajax sur le filtre et sur le panier, afin que ces actions soient plus fluides pour l'utilisateur
- Mettre en place un fil d'ariane afin de pouvoir revenir en arrière, notamment sur les pages du panier au paiement.
- Mettre en place des modales de confirmation, lorsque l'utilisateur est amené à supprimer son compte, son adresses ou un produit dans le panier.
-
- Mettre en place un système de disposition des produits, en ligne ou en colonne
- Mettre en place le système de Drag & Drop afin que l'administrateur puisse ajouter des photos aux nouveaux produits.

Modification d'interface :

- Choisir et modifier la quantité de produit dans le panier
- Optimisation du code et du design afin que celui soit optimal pour l'utilisateur
- Une version en anglais du site

Sécurité :

- Ne pas afficher les boutons qui ne permettent pas d'effectuer l'action par l'utilisateur : les boutons supprimer et modifier d'un produit qui appartient déjà à une commande n'est pas possible.
- *Anonymiser les données pour que l'utilisateur puisse supprimer ses adresses et la commande ou plut*
- *Voir pour l'adresse de l'utilisateur s'il peut la supprimer avec la commande. ? ou l'anonymisé*

VIII. Conclusion

Remerciement

Annexes

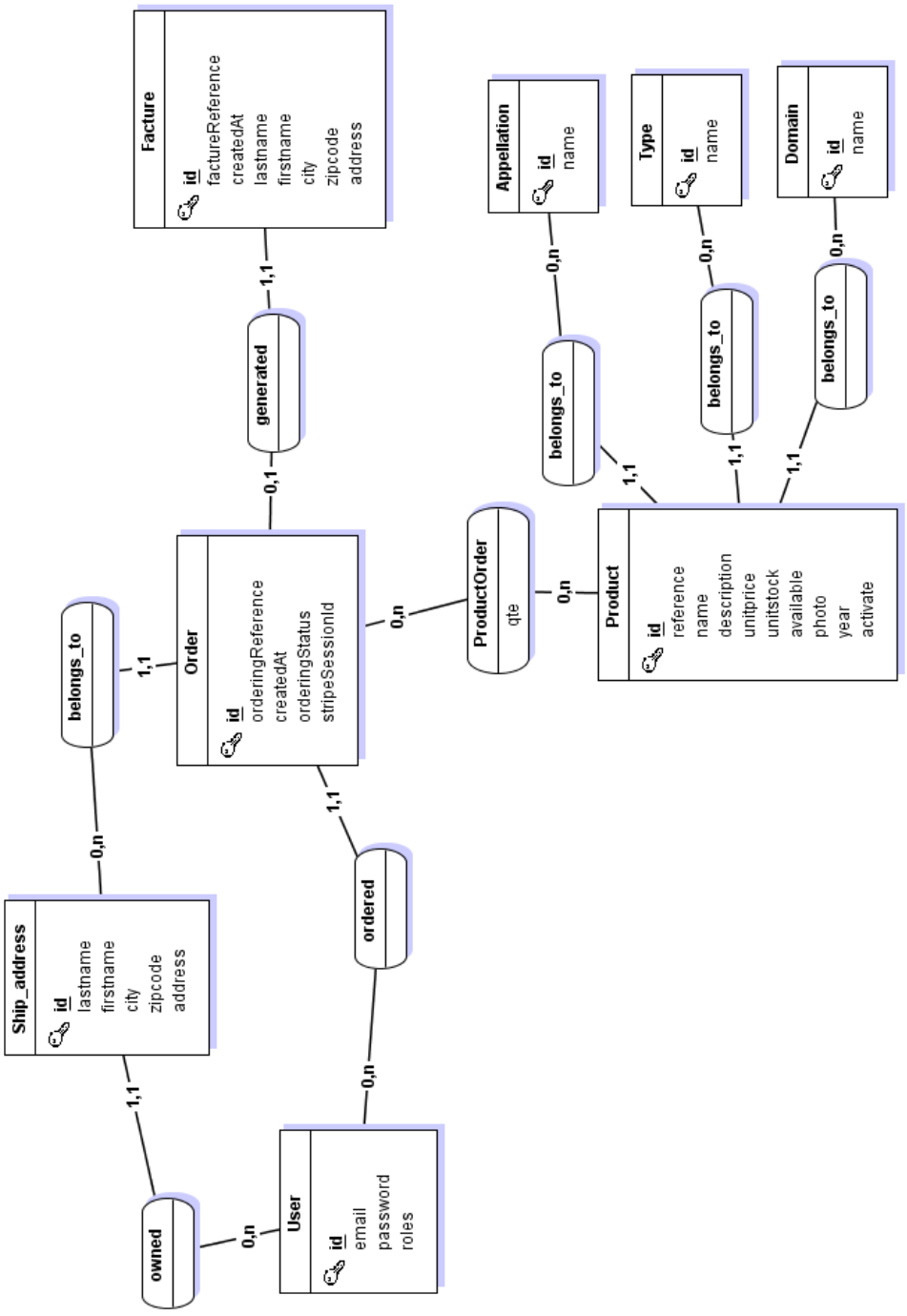


Figure 20 MCD

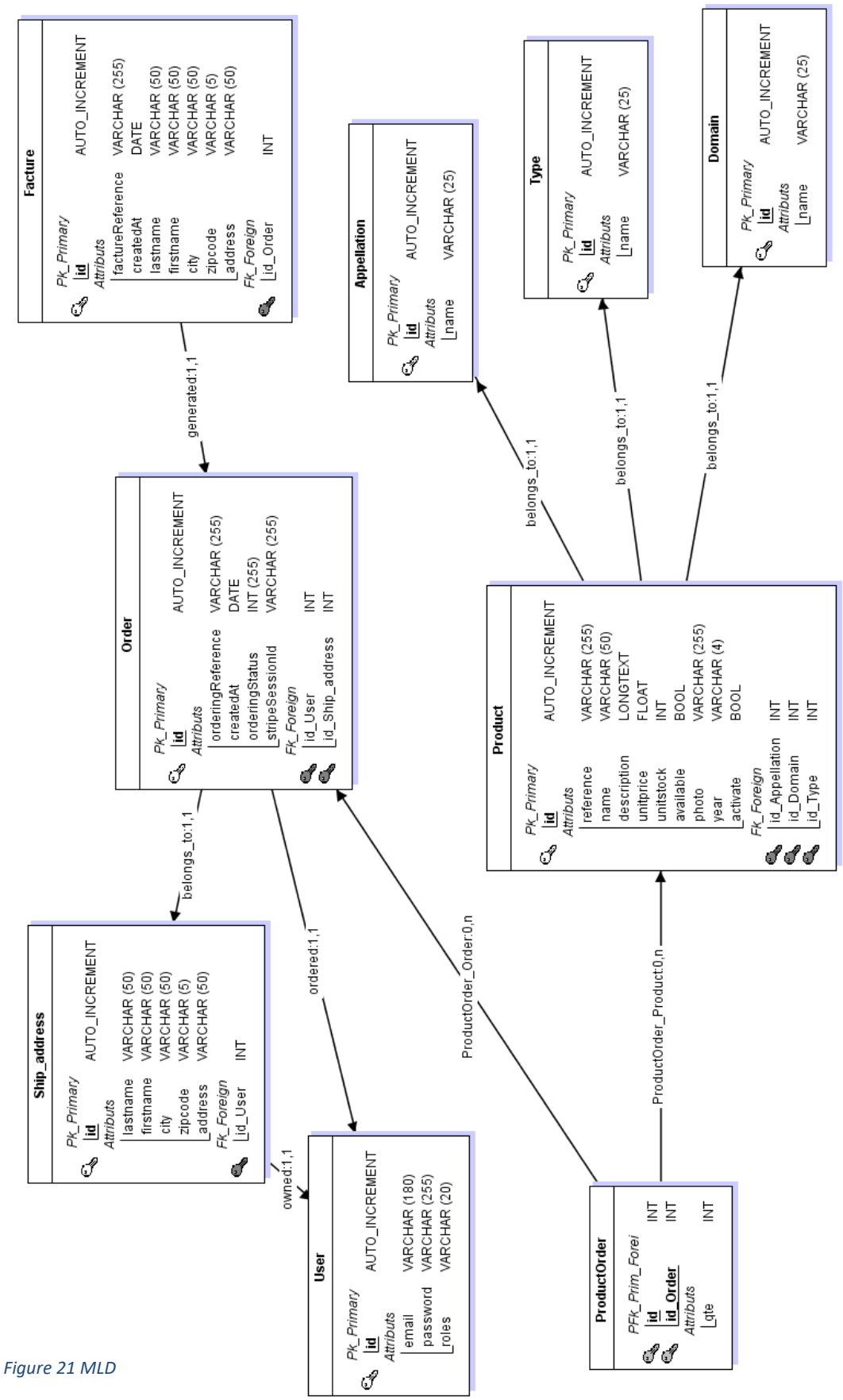


Figure 21 MLD

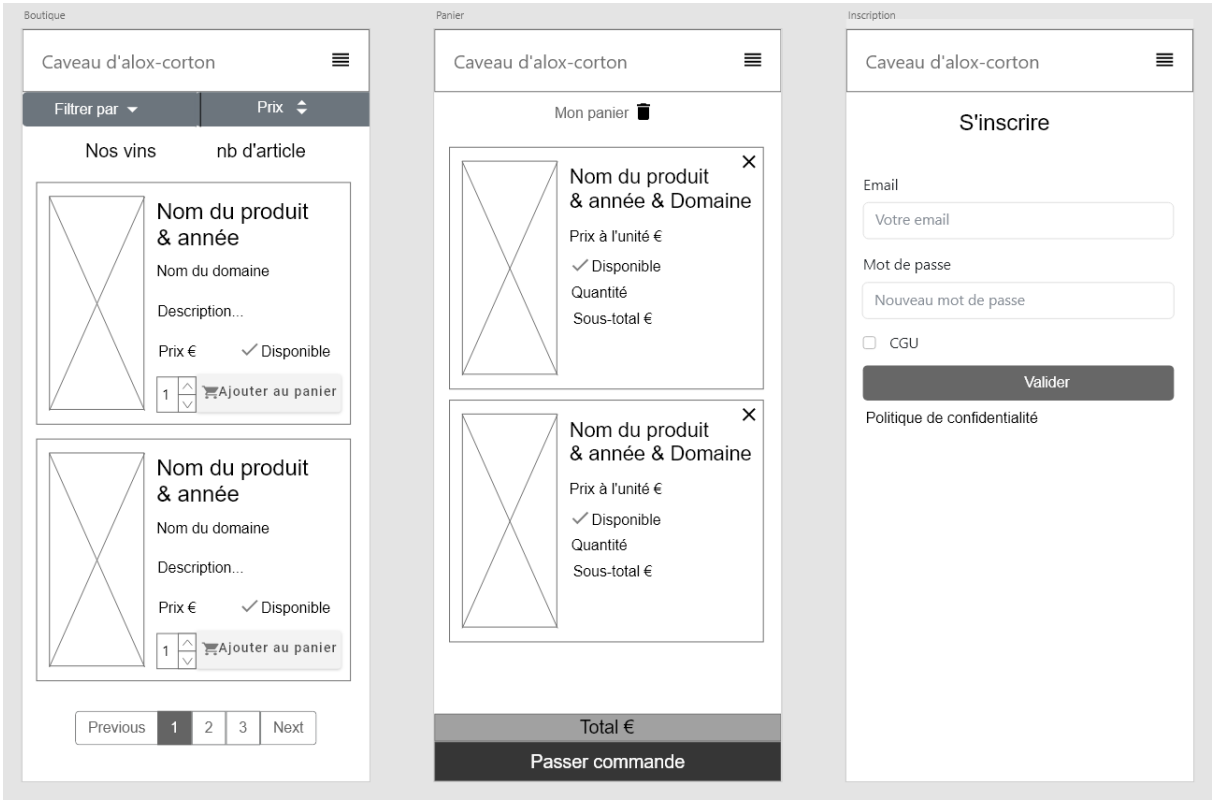


Figure 22 Maquette Exemple 1

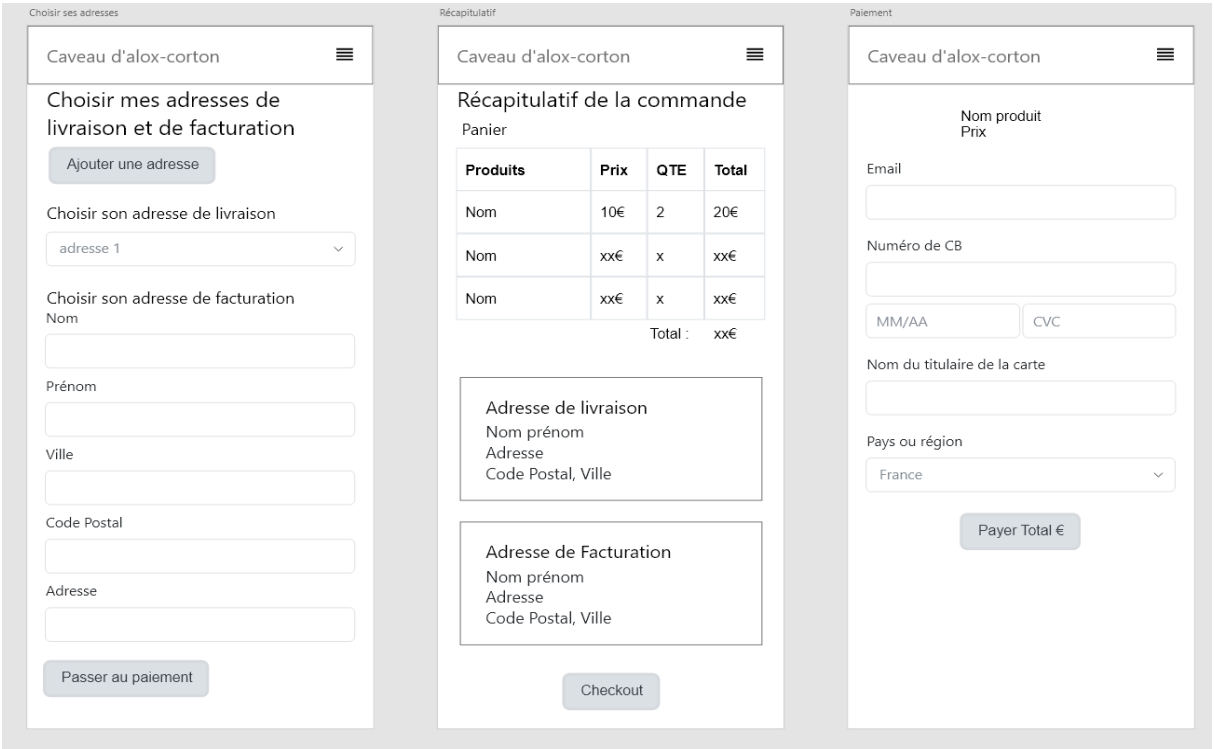


Figure 23 Maquette Exemple 2

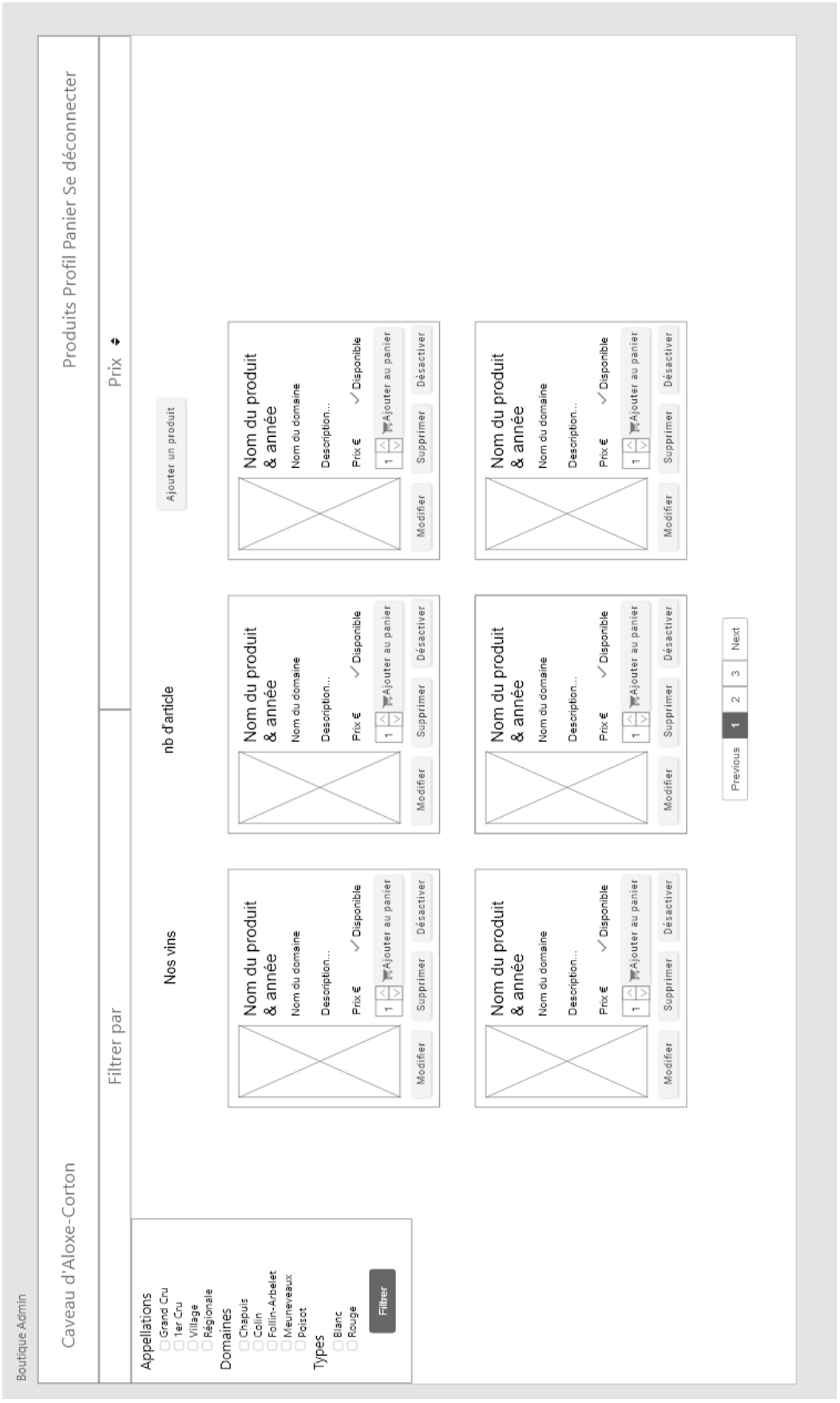


Figure 24 Maquette Desktop