01/02/2021

Dossier de Synthèse

Développeur web et web mobile

Niveau III

Lisa Foret

Table des matières

I. 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

II. 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 envoyer 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 e-mail, ses adresses, ses commandes effectuées, où il pourra également modifier son adresse 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 chargé 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 des 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 destiné 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 récoltées et à quelles fins mais également ses droits concernant ses données.

6. Maquettes

Une maquette 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). Elle est également nécessaire lorsque l'on travaille avec un client pour qu'il ait rapidement une idée du projet. Ici, 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 réalisé les maquettes de mon site web pour le format téléphone (*Figure 1 & 2*). Ensuite, j'ai réalisé les maquettes pour la version desktop (*Figure 3*). Lors du développement de mon projet, mes maquettes m'ont grandement facilité la tâche en terme d'intégration web.

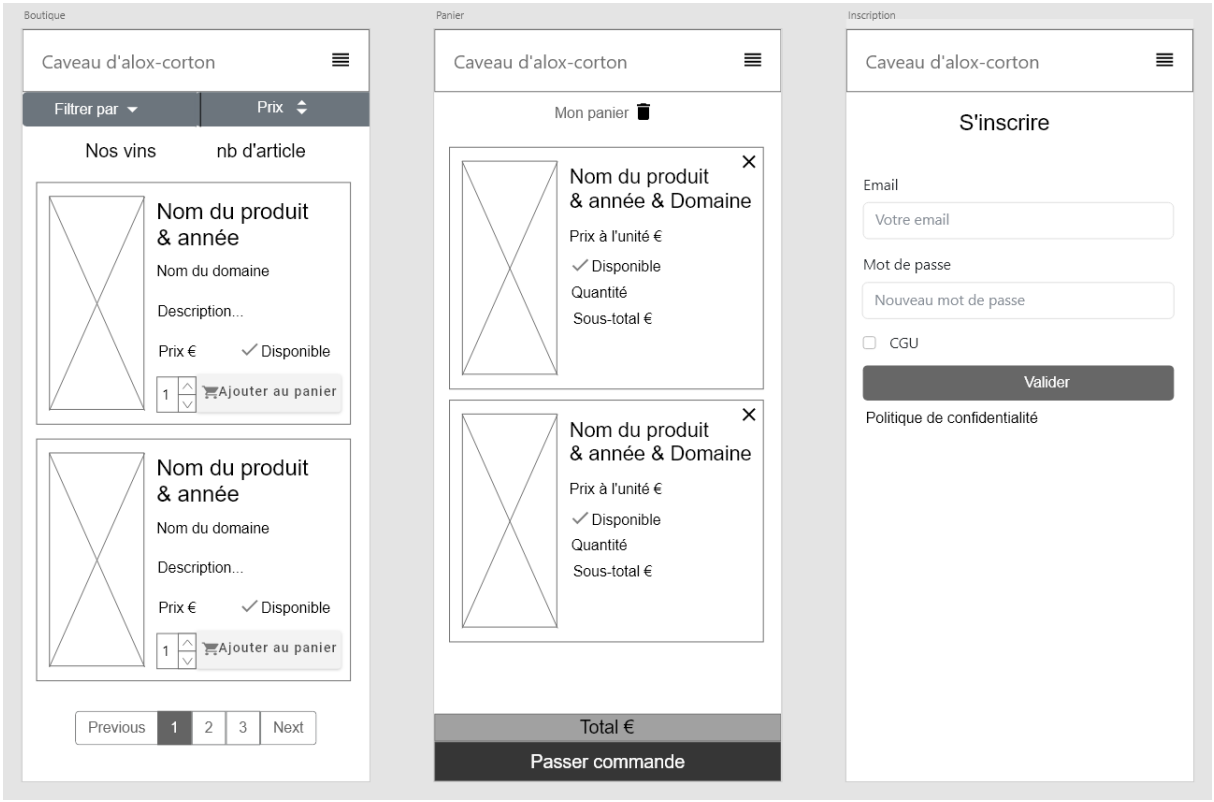


Figure 1 Maquette Exemple 1

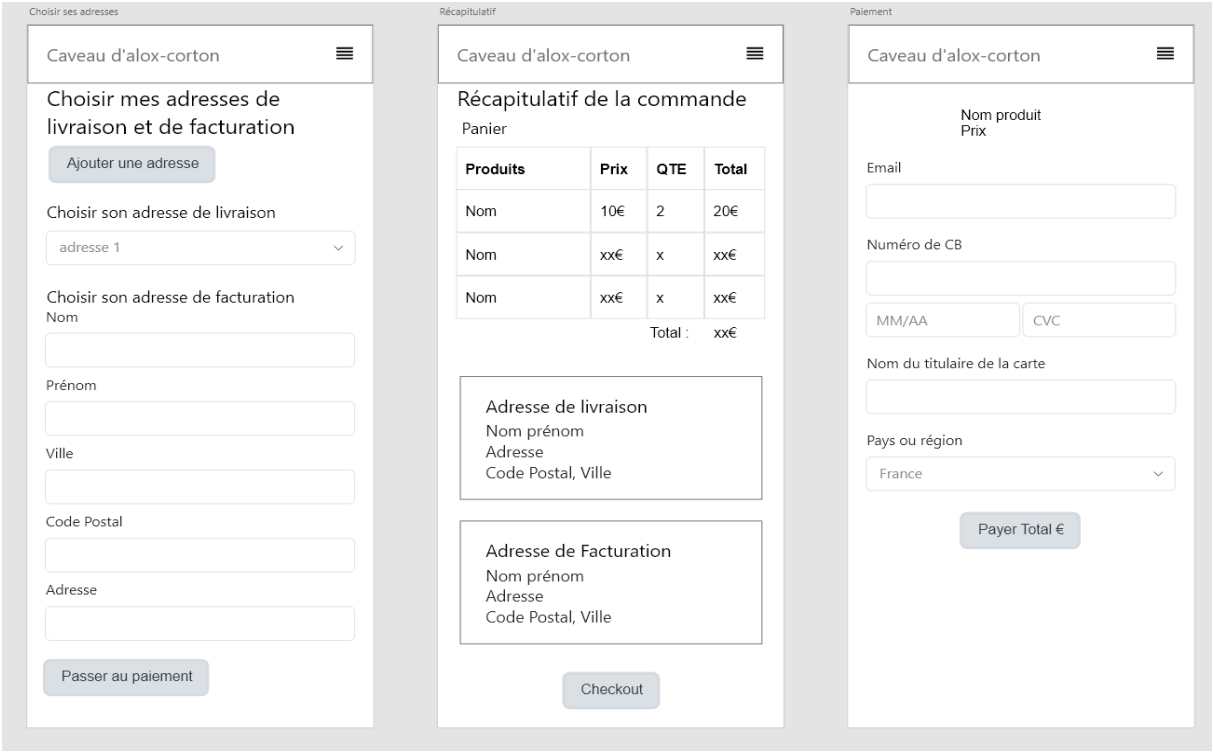


Figure 2 Maquette Exemple 2

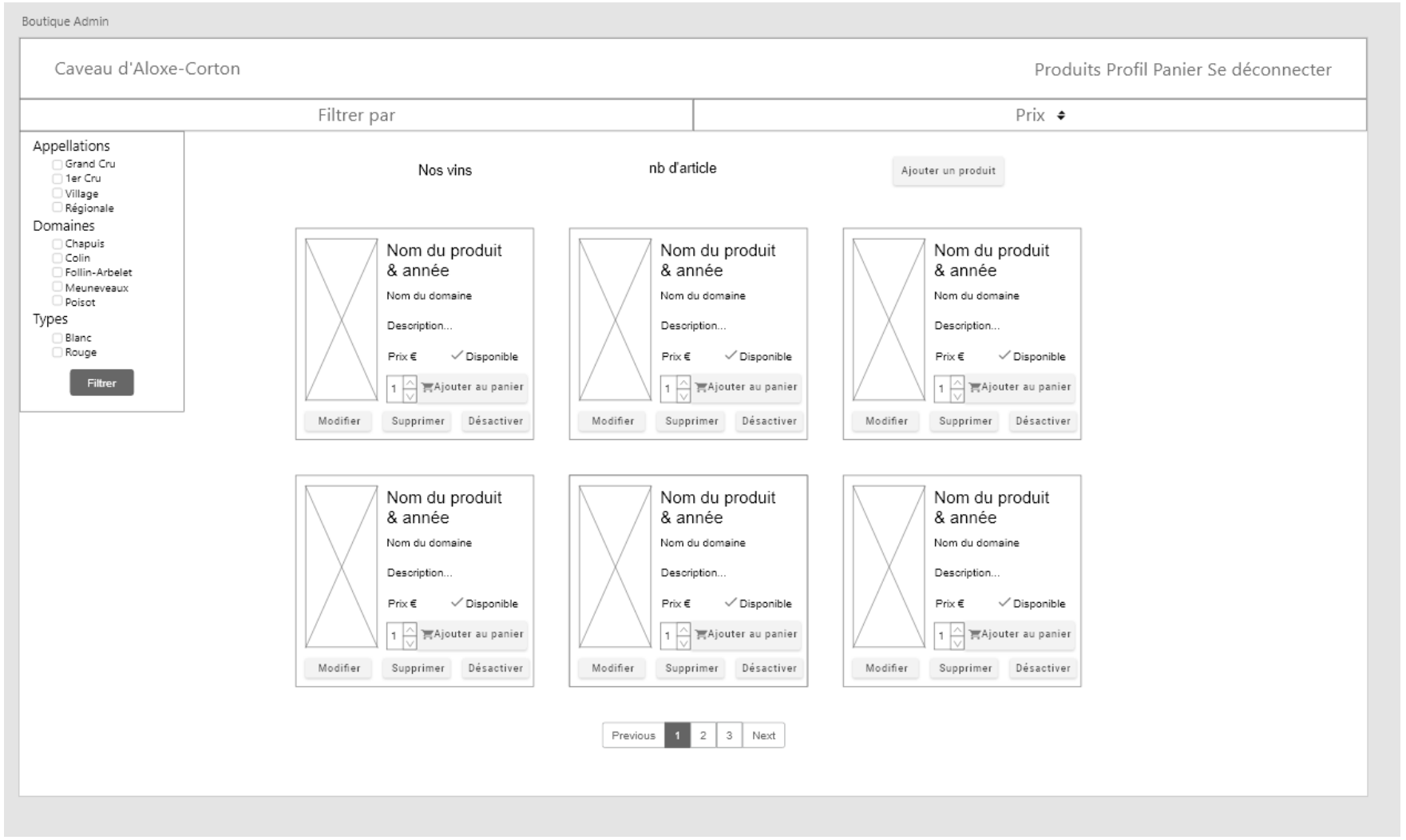


Figure 3 Maquette Desktop

III. Technologies utilisées

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)

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.

Visual Studio Code : est un éditeur de code extensible développé par Microsoft pour Windows.

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

Jmerise : c'est un logiciel dédié à la modélisation des modèles conceptuels de données, 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.

API Stripe : est une solution très simple d'API bancaire pour effectuer des paiements en lignes.

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)

JavaScript c'est un langage de programmation de scripts, principalement employé dans les pages web interactives.

Jquery est une bibliothèque JavaScript libre et multiplateforme créer pour faciliter l'écriture de scripts côté client.

Git : est un système de gestion de version décentralisé.

Bootstrap : est un Framework CSS.

IV. Conception & Développement

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 2*).

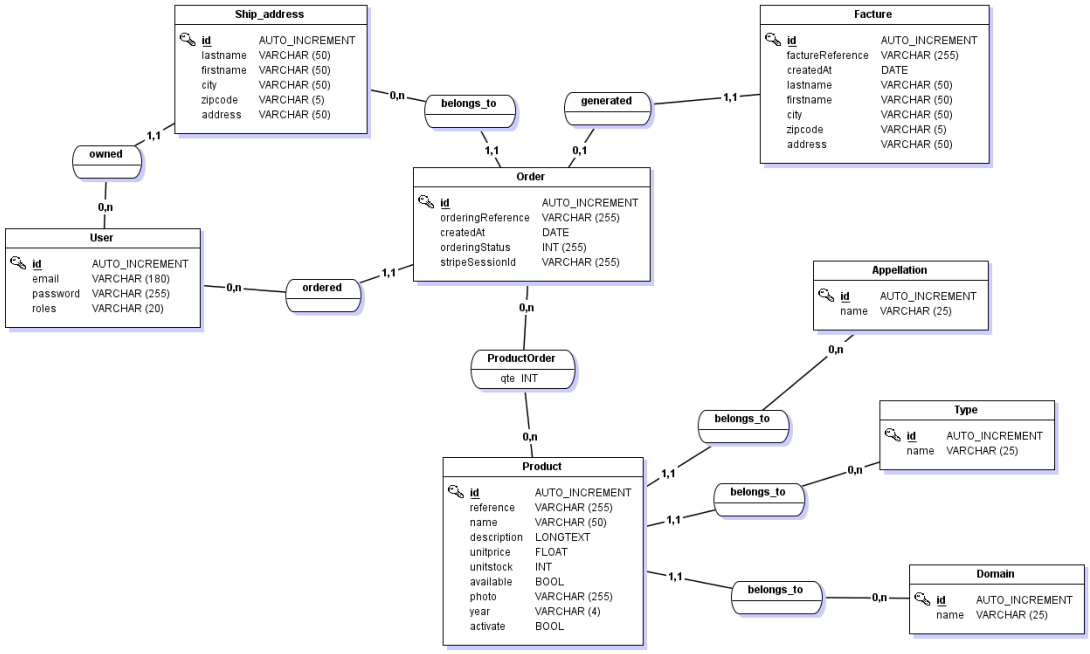


Figure 4: MCD

Chaque produit possède une appellation, un type et un domaine. L'utilisateur contient les données essentielles, l'e-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 3*) à partir du MCD, il met en évidence les relations existantes entres les entités et ajoute toute les clés étrangère pour chaque entité. Par exemple, il 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.

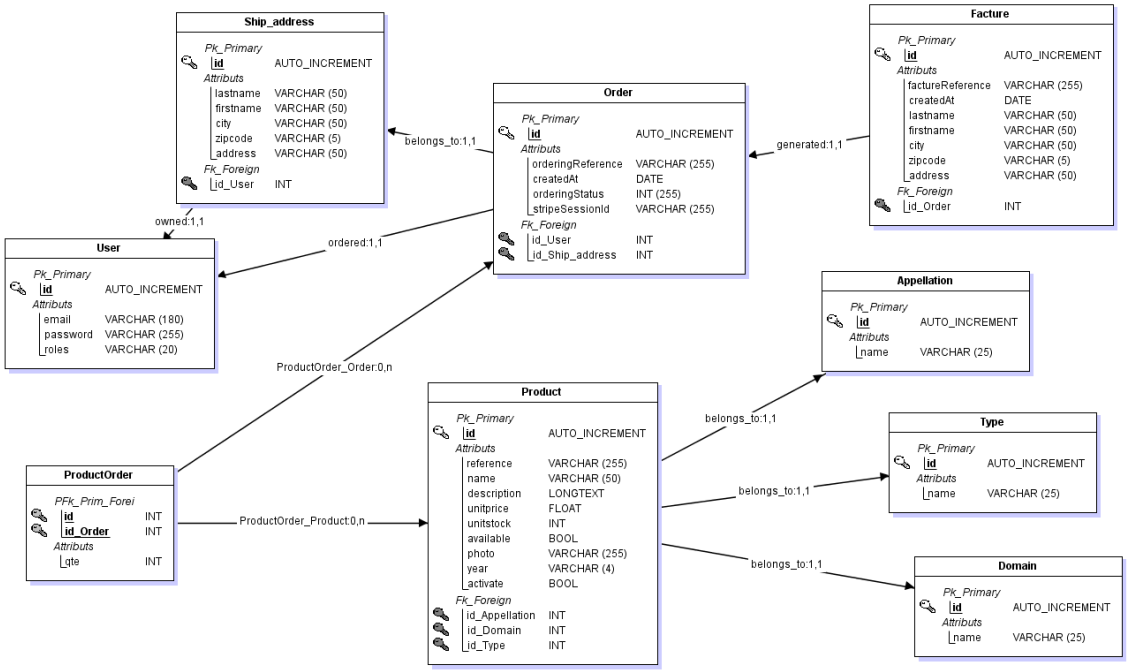


Figure 5: MLD

Le MCD et le MLD m'ont permis de bien comprendre les relations entre les entités, et ainsi de bien démarrer mon projet.

2. Symfony

J'ai choisi d'utiliser le framework Symfony pour réaliser mon projet afin de faciliter et donc 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.

a. Présentation du design pattern MVC & MVP

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 Modèle Vue 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/SQL.
- 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.

L'utilisateur interagit en envoyant une requête au contrôleur, c'est le point d'entrée de l'application. Le contrôleur demande les données au modèle qui traduit cette demande par une requête SQL, il récupère les données et les renvois au contrôleur. Le contrôleur transmet les données à la vue qui se charge d'afficher les données. La vue ici n'a aucune logique.

Cependant, Symfony est capable de créer n'importe quel type d'architecture. C'est pourquoi pour ce projet nous utiliserons plutôt l'architecture MVP qui découle de l'architecture MVC. Le P signifie ici Présentation qui correspond au contrôleur. Dans cette architecture, l'utilisateur envoie sa requête à la vue en cliquant sur un bouton par exemple, qui fait elle-même appel au Présenter (*Figure 6 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).

```php
class AdminController extends AbstractController
{
    /**
     * @Route("/admin", name="admin")
     *
     * Fonction permettant d'afficher toute les commandes pour l'admin
     */
    public function index(OrderingRepository $or, Request $request, PaginatorInterf
    {

        $donnees = $or->findByPayed();
        $orderings = $paginator->paginate(
            $donnees, // Requête qui contient les données
            $request->query->getInt('page', 1), // Numéro de la page en cours, pass
            5 // Nb de résultat par page
        );
        $productMostSold = $pr->findMostSold(); // [[]] on récupère le ou les produ
        $array = [];
        foreach($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)
            ];
        }
        // dd($array);
        return $this->render('admin/index.html.twig', [
            'orders' => $orderings,
            'productMostSold' => $array
        ]);
    }
```

Figure 6 Exemple de Presenter

Le Modèle (*Figure 7 ci-dessous*) traite la demande du contrôleur en récupérant les données, puis les retourne au contrôleur.

```php
<?php

namespace App\Repository;

use App\Entity\Ordering;
use Doctrine\Persistence\ManagerRegistry;

use Doctrine\Bundle\DoctrineBundle\Repository\ServiceEntityRepository;

/**
 * @method Ordering|null find($id, $lockMode = null, $lockVersion = nul
 * @method Ordering|null findOneBy(array $criteria, array $orderBy = nu
 * @method Ordering[]    findAll()
 * @method Ordering[]    findBy(array $criteria, array $orderBy = null,
 */
class OrderingRepository extends ServiceEntityRepository
{
    public function __construct(ManagerRegistry $registry)
    {
        parent::__construct($registry, Ordering::class);

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

    }

```

Figure 7 Exemple de Model

Le contrôleur retourne alors une vue (*Figure 8 prochaine page*) avec les données. Autrement dit ici, le point d'entrée de l'application c'est la vue, de plus, la vue et le modèle ne sont pas directement en relation.

```twig
{% extends 'base.html.twig' %}

{% block title %}Admin Panel{% endblock %}

{% block body %}
    {% block nav %}
        {{ parent() }}
    {% endblock %}
    <section id="adminOrdersSection">
        <h1 class="h3 mb-3 font-weight-normal text-center">Historique des commandes</h1>
        <div>
            <p>Le(s) produit(s) le(s) plus vendu(s) :
                <ul>
                    {% for product in productMostSold %}
                        <li>
                            {{ product.productName ~ ", domaine " ~ product.domain|title ~ " commandé: " ~ product.quantity ~ " fois. " }}
                        </li>
                    {% endfor %}
                </ul>
            </p>
        </div>
        <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>
```

Figure 8 Exemple de View

b. 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.

Ensuite avec les commandes « php bin/console make :migration » qui permet de créer un fichier de version de la base de données contenant le SQL pour mettre à jour la base de données, « php bin/console doctrine:migrations:migrate » qui permet d'exécuter la migration. Cette méthode permet d'obtenir des versions à chaque modification de la base de données.

On peut cependant remplacer cette méthode avec la commande « php bin/console doctrine:schema:update --force » qui en revanche ne conserve aucun historique. C'est cette méthode que j'ai utilisée, ***car ma base de données n'est pas très complexe.***

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

c. Gestion des utilisateurs

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

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 les données de l'utilisateur de la session.

La méthode de hachage des mots de passe des utilisateurs est définie dans le fichier security.yaml (*Figure 4*), dans notre cas l'encodage est laissé par défaut en « auto », car il permet de sélectionner le meilleur encodeur possible selon Symfony.

```
1  security:
2      encoders:
3          App\Entity\User:
4              algorithm: auto
```

Figure 9 : 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 sont 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.

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.

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 (Figure5), 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 10: 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.

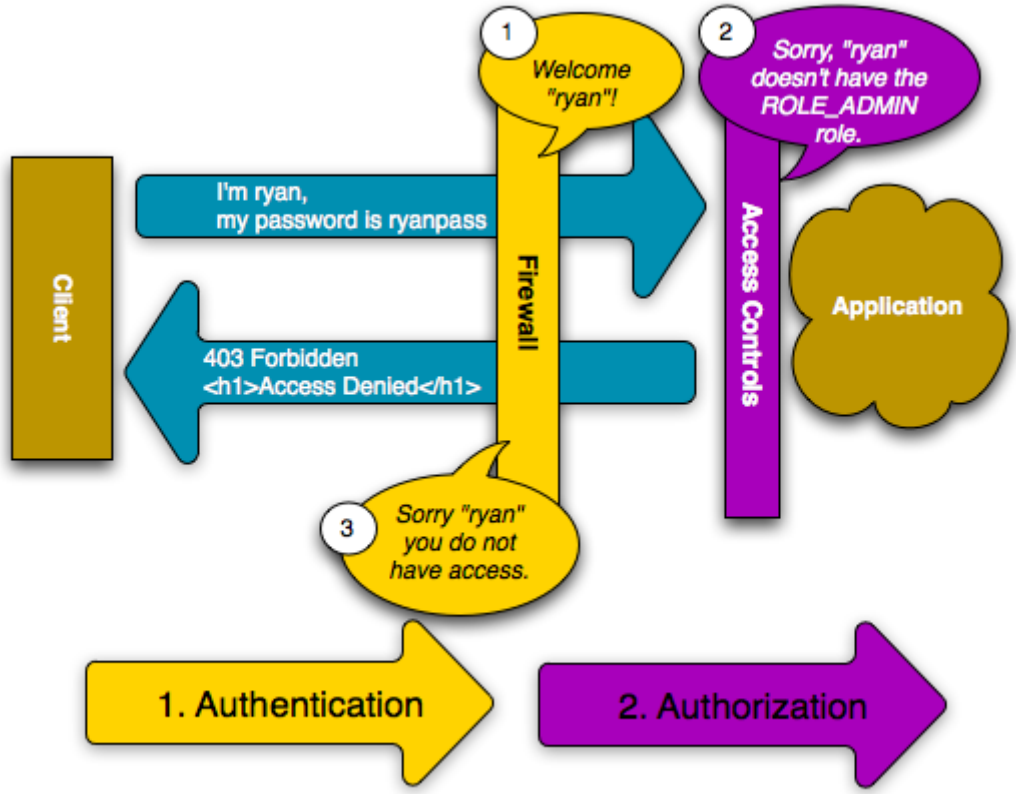
Autorisation

La dernière partie du fichier security.yaml (figure 6) 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.

```yaml
access_control:
    - { path: ^/admin, roles: ROLE_ADMIN }
    - { path: ^/profil, roles: ROLE_USER }
```

Figure 11: Fichier security.yaml section access_control

Ce schéma 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).



d. 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.

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.

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.

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 12 ci-dessous*), on génère une chaine de caractère 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.

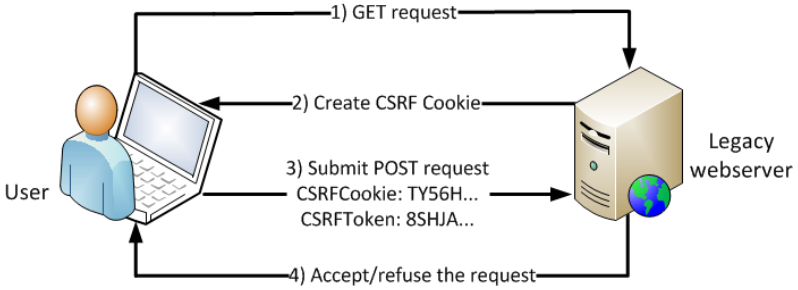


Figure 12 Schéma fonctionnement CSRFToken

V. 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 et 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 13 ci-dessous*), permet d'afficher les produits, en fonction des filtres que l'utilisateur choisi.

```php
class StoreController extends AbstractController
{
    /**
     * @Route("/products/", name="products_index")
     *
     * Fonction permettant d'afficher la liste des produits, et de les filtrer
     */
    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 13 Fonction showProducts()

J'ai mis en place une entité Filtre qui me permet de stocker et de récupérer les données des filtres, elle aura 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 14 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 tout 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 tout les produits, pour l'administrateur, ainsi que les données des produits activé, 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 14 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 en session et l'utilisateur. Ci-dessous (*Figure*) la fonction pour ajouter un produit au panier.

```php
/**
* @Route("/cart/add/{id}", name="cart_add")
*/
public function add(Request $request, Product $product = null, SessionInterface $session)
{
    // Vérifier que le produit existe
    if(!$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");
    }
}
```

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 ajouter et on vérifie quel est bien supérieur 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. A 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 toute 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 toute les méthodes dont il a besoin concernant le panier. La méthode add() du contrôleur utilise la fonction add() de l'entité Cart (figure ci-dessous).

```php
public function add(Product $product, $qtt){
    if(!array_key_exists($product->getId(), $this->incart)){
        $this->incart[$product->getId()] = [
            "product" => $product,
            "quantity" => $qtt
        ];
    }
    else{
        $this->incart[$product->getId()]["quantity"]+=$qtt;
    }
}
```

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, envoyer sur une page pour choisir ces adresses de livraison et de facturation. Il s'agit de l'une des fonctions les plus importantes (*Figure ci-dessous Partie 1*).

```php
/**
 * @Route("/chooseAdd", name="choose_address")
 */
public function chooseAddress(Request $request, EntityManagerInterface $manager, FactureRepository $fr, SessionInterface $session)
{
  $incart = [];
  $user = $this->getUser();
  if(!$user){
    return $this->redirectToRoute("app_login");
  }

  $newOrder = new Ordering();
  $newFacture = new Facture();
  $cart = $session->get('cart', new Cart());
  $newFacture->setUserId($this->getUser()->getId());
  $newOrder->setFacture($newFacture);
  //on récuppère la dernière facture
  $lastFacture = $fr->findLastFacture($this->getUser()->getId());
  // Pré rempli la nouvelle facture
  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());
  }
  foreach($cart->getFullCart() as $cartLine){
    $incart[] = [
      'product' => $cartLine['product'],
      'quantity' => $cartLine['quantity']
    ];
  }
```

Tout d'abord, on vérifie que l'utilisateur est bien identifié. C'est ici, que l'on créer la nouvelle commande et la nouvelle facture. Ensuite on récupère la dernière facture de l'utilisateur, si toutefois elle existe, et on pré remplie les champs concernant l'adresse de facturation à l'aide de cette dernière facture. On vérifie que le panier n'est pas vide et on récupère toutes ses informations afin de les envoyer sur la prochaine vue.

```php
    $total = $cart->getTotal($incart);
    $formOrder = $this->createForm(OrderType::class, $newOrder);
    $formOrder->handleRequest($request);
    if($formOrder->isSubmitted() && $formOrder->isValid()){
      $newOrder->setUser($this->getUser());
      $newOrder->getFacture()->setOrdering($newOrder);
      $manager->persist($newOrder);
      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();
      return $this->render('checkout/index.html.twig', [
        'items' => $incart,
        'total' => $total,
        'order' => $newOrder,
        'reference' => $newOrder->getOrderingReference(),
      ]);
    }
    return $this->render('cart/addresses.html.twig', [
      'formOrder' => $formOrder->createView(),
    ]);
  }
```

Dans la seconde partie de cette fonction (*figure ci-dessus Partie 2*), on créer un formulaire de commande qui contiendra simplement les informations des adresses de livraison et de facturation. Lorsque ce formulaire est soumis et valider, je remplace l'utilisateur dans ma commande mais également la commande dans ma facture, j'utilise la méthode persist() afin que doctrine gère l'objet. Ensuite, pour chaque ligne du panier, on instancie un nouveau « ProductOrdering » où l'on récupère le produit et sa quantité, j'utilise la méthode persist() afin que doctrine gère l'objet.

Pour finir, j'utilise la méthode flush(), ainsi Doctrine exécute une requête qui permettra d'ajouter tout ce qui a été persisté auparavant à la base de données. A ce moment-là, la facture et la commande sont créer dans la base de données, le statut de la commande est alors « en attente de paiement ». Je retourne une vue qui affichera un récapitulatif de la commande, je lui inclus les données du panier, la nouvelle commande, ainsi que la référence de la commande dont j'aurai besoin pour procéder au 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 (*Figure ci -dessous*), à l'aide de la documentation qu'il fournit (qui a été utilisé pour la traduction).

```php
/**
 * @Route("/create-checkout-session/{reference}", name="create-checkout-session")
 */
public function payment($reference, OrderingRepository $or, EntityManagerInterface $manager)
{
  $order = $or->findOneByOrderingReference($reference);


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

  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('sk_test_51HvgjELyEjuAwgbZtFkkq4UfxmsjafIAB10xIVuEjqHkQqVuHmrtBD4XvNGHPL
  $checkout_session = \Stripe\Checkout\Session::create([
    // 'customer_email' => $this->getUser()->getEmail(),
    '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]);
}
```

La seul difficulté que j'ai rencontré 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*).

```
<script type="text/javascript">
    // Create an instance of the Stripe object with your publishable API key
    var stripe = Stripe('pk_test_51HvgjELyEjuAwgbZePdTWvQMgmdhI1uAxfEmwx6zqvuS
    var checkoutButton = document.getElementById('checkout-button');


    checkoutButton.addEventListener('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>
```

Lorsque la vue du récapitulatif du commande s'affiche, un bouton « payer » se trouve sur la page. C'est ce bouton qui grâce au JavaScript que Stripe fournit, renvoie à mon contrôleur « Checkout », qui contient la méthode qui est appelé par le javascript, la référence de ma commande.

Grâce a ça, j'ai pu faire appel à mon repository pour qu'il récupère la commande concerné et ainsi afficher les données de la commande dans la vue de paiement de Stripe.

Pour finir avec cette méthode, je récupère et je stock dans ma commande l'identifiant de checkout de la session que Stripe fournit. Stripe permet de configurer les vues de succès et d'erreur qui s'afficheront à la suite du paiement, j'y intègre donc dans l'url l'identifiant de checkout de la session de Stripe.

Une fois le paiement effectué, Stripe renvoie, selon le résultat, sur la page de succès ou la page d'erreur préconfigurer dans la méthode juste avant. En cas de succès (*Figure*), je retrouve en paramètre d'URL l'identifiant de checkout de la session qui me permet de récupérer la commande intégralement. Il faut ensuite hydrater la commande avec l'utilisateur pour vérifier que l'utilisateur de la commande correspond bien à l'utilisateur connecté. Je récupère également toutes les données de la commandes afin de les afficher et de les présenter comme une facture. Niveau back-end, je remplace le statut de la commande qui était « en attente de paiement » par « payé », puis je retire la quantité de stock qui a été vendu. Et enfin je vide le panier

```php
/**
 * @Route("/success/{stripeSessionId}", name="success")
 */
public function success($stripeSessionId, OrderingRepository $or, UserRepository $ur, SessionInterface $session, EntityManagerInterface $manager)
{
  $order = $or->findOneByStripeSessionId($stripeSessionId);
  // on hydrate car order ne récup que l'id du user
  $user = $ur->findOneById($order->getUser()->getId());
  if(!$order || $user != $this->getUser()){
    return $this->redirectToRoute('home_index');
  }
  $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
    $cart->clear($cart->getFullCart());
    return $this->render('checkout/success.html.twig', [
      'order' => $order,
      'total' => $total,
      'quantityTotal' => $quantityTotal,
    ]);
  }
```

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

- Mettre en place un système de mailer pk ?
- Mettre en place du ajax panier/filtre
- Bouton supprimer pour l'admin
- DragNDrop photo
- Modale de sureté
- Voir pour l'adresse de l'utilisateur s'il peut la supprimer avec la commande. ? ou l'anonymisé
- Ajuster niveau design & css pour que ce soit optimal pour l'utilisateur
- Optimiser le code
- Version Anglais
- Disposition des produits
-

VIII. Conclusion

IX. Remerciement

1. Réalisations du candidat comportant les extraits de code les plus significatifs et en les argumentant, y compris pour la sécurité et le web mobile

2. Présentation du jeu d'essai élaboré par le candidat de la fonctionnalité la plus représentative (données entrée, données attendues, données obtenus)

3. Description de la veille, effectuée par le candidat durant le projet, sur les vulnérabilités de sécurité

4. Description d'une situation de travail ayant nécessité une recherche, effectuée par le candidat durant le projet, à partir d'un site anglophone

5. Extrait du site anglophone, utilisé dans le cadre de la recherche décrite précédemment, accompagné de la traduction en français effectuée par le candidat sans traducteur automatique, environ 750 signes

6. difficultés rencontrées ?

X. Résumé

200 à 250 mots

Ce projet consiste à créer et développer une boutique en ligne de vente de vin, pour une société. Cette société souhaite à l'avenir, mettre en place une boutique en ligne. Cette application doit avoir une partie présentation de la société ainsi qu'une partie boutique en ligne ayant les fonctions suivantes : création d'un compte utilisateur ainsi que sa gestion, afficher la liste des produits disponibles, commander et payer ces produits. Ce projet est réalisé avec le framework Symfony, ainsi que l'API Stripe pour réaliser les paiements bancaires.