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 a mes expériences professionnelles, que ce domaine ne correspond pas 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. A 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 static 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 qu'il souhaite

- Accéder à son profil avec ses informations, notamment son email, ses adresses, ses commandes effectués, 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 n'a besoin de se connecter uniquement pour accéder à son profil et passer une commande.

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

- Se connecter et se déconnecter
- Ajouter, modifier 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 la partie non essentielle du cahier des charges, qui si elle n'est pas respectée peut être poussé en axe d'amélioration :

- Un système de filtre des produits, par type, domaine ou appellation
- Un système de tri des produits, par ordre alphabétique ou 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 possèderait une partie site vitrine concernant la présentation de la société.

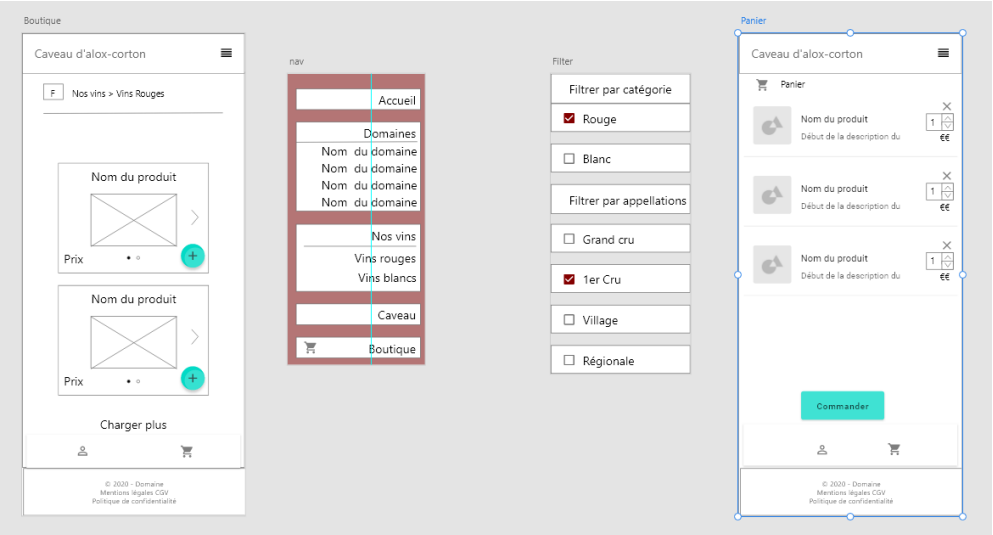
5. RGPD

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 réalisé 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.

J'ai d'abord réalisé les maquettes de mon site web pour le format téléphone (*Figure1*). J'ai passé plusieurs heures à optimiser les maquettes afin que l'expérience utilisateur soit la meilleur possible. Ensuite, j'ai réalisé les maquettes pour la version desktop.

Figure 1 : Exemple de maquette



Lors du développement de mon projet, mes maquettes initiales m'ont permis de me rendre compte que certains points n'étaient pas nécessaires ou trop précis. De ce fait, le rendu final de mon projet est assez différent de mes maquettes.

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 (ORM : Technique de programmation faisant le lien entre la base de données et la programmation orienté objet) 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 facilité l'écriture de scripts côté client.~~

Git : est un système de gestion de version décentralisé.

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. Il permet de représenter les données d'un système d'information.

J'ai réalisé mon MDC à l'aide de Jmerise (*Figure 2*).

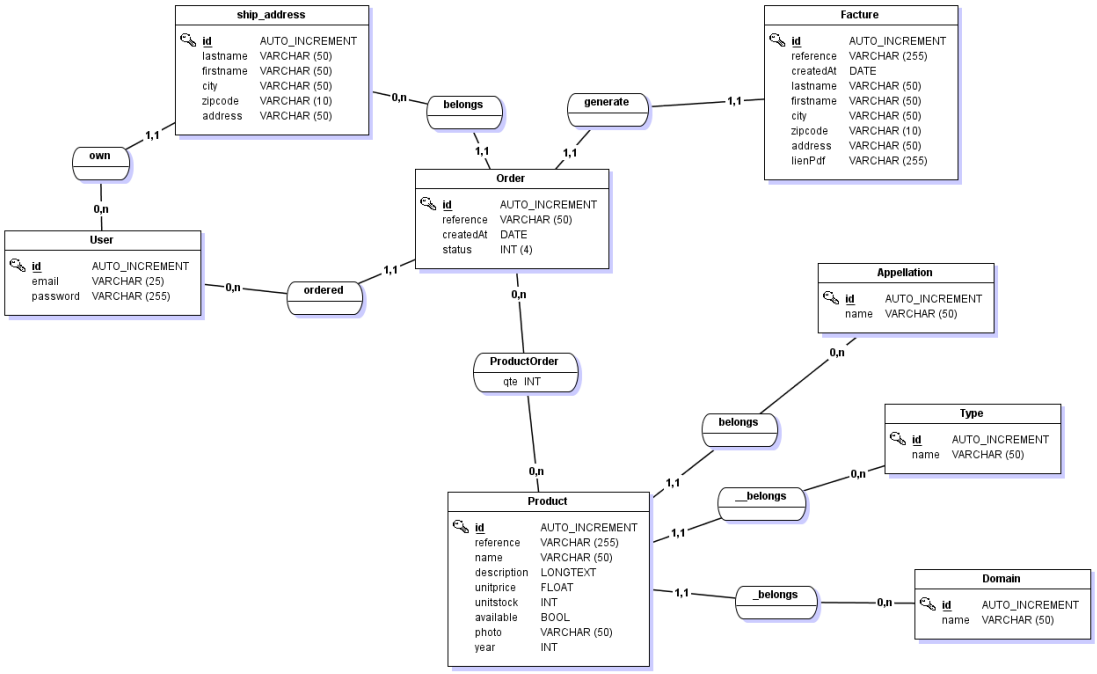


Figure 2: MCD

L'entité « User » contient l'essentiel : l'email comme moyen d'authentification, un mot de passe, ainsi qu'un rôle. Elle est reliée à l'entité « Ship_address » avec une relation qui permet qu'un utilisateur puissent avoir plusieurs adresses de livraisons mais qu'une adresse de livraison ne puisse appartenir qu'a un seul utilisateur. L'entité « Ship_address » contient toutes les informations relatives à une adresse, elle est reliée à l'entité « Order » de manière que l'adresse de livraison puisse correspondre avec plusieurs commandes mais qu'une commande ne soit reliée qu'a une seule adresse de livraison.

L'entité « Order » contient une référence de commande, une date de création, un statut qui correspondra à l'état de la commande (en cours de traitement, payé, annulé), elle est également reliée à l'utilisateur, un utilisateur peut avoir plusieurs commandes, mais une commande n'est faite que par un seul utilisateur.

L'entité « Order » est en relation avec l'entité facture qui elle contiendra sa propre référence et adresse de facturation, ainsi lorsque l'utilisateur souhaite supprimer ses données, je pourrais les supprimer sans problème en gardant uniquement le nécessaire qui sera inclut dans l'entité Facture. Cette entité possède également une propriété « lienPDF » qui sera générer lorsque l'utilisateur achèvera sa commande.

L'entité « Order » est en relation avec l'entité Product qui elle contient toutes les informations relatives à un produit : la référence, le nom, la description, le prix à l'unité, la quantité en stock,

la disponibilité du produit et la photo. Cette relation est de type : 0,n ce qui inclut donc une table de jointure que j'appellerai « ProductOrder », elle correspondra à l'association d'un ou plusieurs produits à une ou plusieurs commandes. J'y inclus la quantité afin d'obtenir une entité que me permettra d'avoir une commande, un produit, et la quantité de ce produit.
L'entité « Product » est reliée à trois entités qui sont des catégories différentes : Appellation, Type, Domain. Un produit ne possède qu'une appellation, qu'un type et qu'un domaine exemple : Corton Charlemagne étant le nom du produit, il a l'appellation Grand Cru, de type vin rouge et du domaine Chapuis.

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. Par exemple il nous permet de mieux concevoir la relation entre les entités « Order » et « Product », la relation est bien 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.

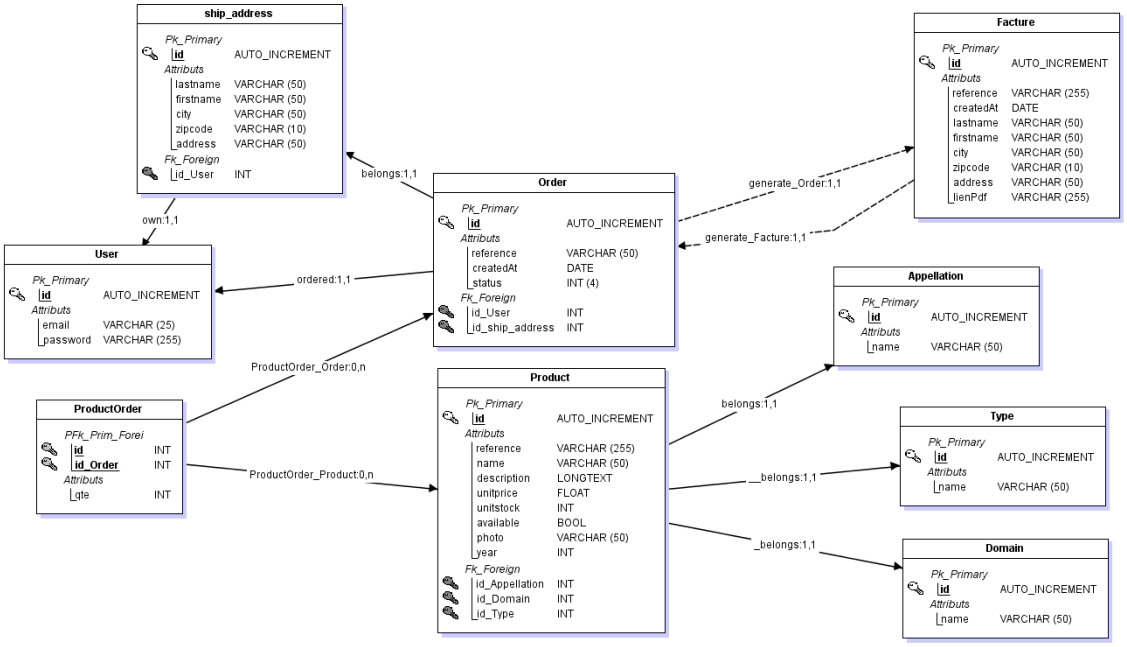


Figure 3: MLD

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 c' est quoi

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 qui fait elle-même appel au Presenter pour traiter la demande de l'utilisateur. Le Présenter fait appel au modèle qui traite la demande du contrôleur et lui renvoies des données. Le contrôleur retourne alors une vue avec les données. Autrement dit ici, la vue et le modèle ne sont pas directement en relation. Et le point d'entrée de l'application c'est la vue.

Exemple de Vue

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

{% block title %}Mes commandes{% endblock %}

{% block body %}
    {% block nav %}
        {{ parent() }}
    {% endblock %}
    <h2>Mes commandes</h2>
    {% if orders|length > 0 %}
        {% for order in orders %}
        <p>Date de la commande : {{ order.createdAt|date('d/m/Y') }}</p>
        <p>Référence de la commande : {{ order.orderingReference }}</p>
        <table class='table'>
            <thead>
                <tr>
                    <td>Produit</td>
                    <td>Quantité</td>
                    <td>Prix unitaire</td>
                    <td>Total</td>
                </tr>
            </thead>
            <tbody>
                {% for productOrder in order.productOrderings %}
                <tr>
                    <td>{{ productOrder.product.name }}</td>
                    <td>{{ productOrder.quantity }}</td>
                    <td>{{ productOrder.product.unitPrice }}</td>
                    <td>{{ productOrder.quantity * productOrder.product.unitPrice }}</td>
                </tr>

                {% endfor %}
            </tbody>
        </table>
        <div>
            <p>Total : {{ order.getTotal }}</p>
            <p>Nombre de produit total : {{ order.getQuantityTotal }}</p>
        </div>
        {% endfor %}
    {% else %}
        <p>Vous n'avez pas de commande</p>
    {% endif %}

    {% block footer %}
        {{ parent() }}
    {% endblock %}
{% endblock %}
```

Exemple de Présentation

```php
/**
 * @Route("/profil/orders", name="profil_orders")
 */
public function ordersUser(){
    $user = $this->getUser();
    if($user){
        $orders = $user->getOrderings();
        return $this->render('profil/orders.html.twig', [
            'orders' => $orders,
        ]);
    }
    return $this->redirectToRoute('app_login');
}
```

Exemple de modèle

b. ORM

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.

3. Création de la 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.

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

4. Gestion des utilisateurs

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

a. User

Grâce au MakerBundle, nous pouvons utiliser la commande « php/bin console make :user » qui nous permet de créer une entité User, elle possèdera les propriétés suivantes par défaut un identifiant unique tel que l'email, un mot de passe ainsi qu'un rôle. Lorsqu'une entité est créée avec la console, le repository de l'entité est également créer, (elle 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. (+ remember me & impersonation).

Les mots de passe des utilisateurs sont encodés dans le fichier security.yaml, dans notre cas l'encodage est laissé par défaut en « auto » car il permet de sélectionner le meilleur encodeur possible selon Symfony.

b. 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.

c. Registration

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.

5. Sécurité Native

Fichier security.yaml

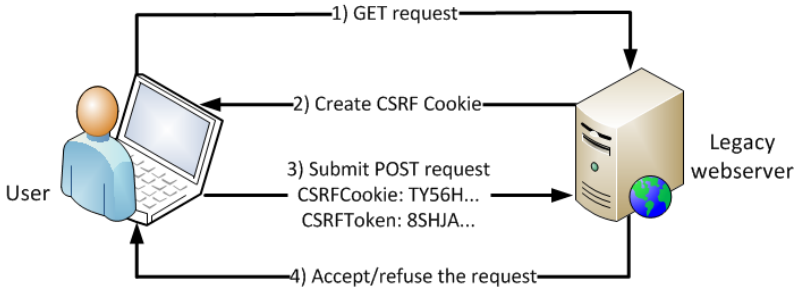
Firewall

Rôle

Faille XSS Cross Site Scripting c'est une méthode permettant d'injecter du contenu HTML ou JavaScript dans une page. Cette faille peut être évité 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.

Injection SQL c'est une méthode permettant d'injecter du code SQL dans un formulaire par exemple. 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 par exemple. Pour contrer cette faille, Symfony dispose de contrainte de validation qui 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.

CSRF (Cross Site Request Forgery) il s'agit d'une attaque où un utilisateur malveillant tente de faire envoyer à un utilisateur une requête vers un site pour y déclencher une action, sans que l'utilisateur en ai conscience. 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.



V. Extrait de code

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*

Extrait de code du panier (ajout au panier ?) jusqu'au paiement.

Passer une commande

```php
    /**
     * @Route("/chooseAdd", name="choose_address")
     */
    public function buy(Request $request, EntityManagerInterface $manager, FactureRepository $fr)
    {
        $user = $this->getUser();
        if(!$user){
            return $this->redirectToRoute("app_login");
        }
        $newOrder = new Ordering();
        $newFacture = new Facture();
        $newFacture->setUserId($this->getUser()->getId());
        $newOrder->setFacture($newFacture);
        $lastFacture = $fr->findLastFacture($this->getUser()->getId());
        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($this->cart->getFullCart() as $cartLine){
            $incart[] = [
                'product' => $cartLine['product'],
                'quantity' => $cartLine['quantity']
            ];
        }
        $total = $this->cart->getTotal($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($this->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(),
        ]);
    }
```

Passer commande est l'une des fonctionnalités les plus importantes de mon application. Lorsque mes produits se trouvent dans mon panier je peux « passer commande ». Cette fonction permet à l'utilisateur de choisir son adresse de facturation et son adresse de livraison, des informations essentielles pour passer une commande.

Dans cette fonction, on vérifie dans un premier temps que l'utilisateur est bien connecté, sinon on le renvoie sur la page du formulaire de connexion.

Ensuite on créer un nouvel objet Facture et Ordering afin de les instancier plus tard lorsque le formulaire sera soumit et validé.

Dans un second temps je vérifie si l'utilisateur à déjà passer une commande et possède donc une facture. Si c'est le cas je récupère l'adresse de la derniere facture, afin de préremplir les champs de la futur facture. Un gain de temps pour l'utilisateur, il peut toujours les modifier à sa guise.

Ensuite je récupère le contenu du panier ainsi que le total. Je créer le formulaire qui contient une liste déroulante des adresses de livraison, et un formulaire concernant l'adresse de facturation (pré rempli s'il existe une ancienne facture).

Si le formulaire est soumis et validé, j'ajoute l'utilisateur authentifié à ma nouvelle commande, je récupère la facture de cette commande et remplace la commande de la facture par cette nouvelle commande. Je persist la nouvelle commande.

Ensuite pour chaque ligne du panier, je créer un « newProductOrdering », je récupère l'ID du produit dans le panier et l'ajoute à newProductOrdering, et j'ajoute également la quantité. J'ajoute chaque productOrdering à ma nouvelle commande et je persist.

Ensuite je flush. Je renvoie sur une vue qui fait le récapitulatif de commande, juste avant de passer au paiement. Je renvoie également la référence de la commande afin de la récupérer lors du paiement avec stripe.

VI. Extrait Anglophone

VII. Axes d’améliorations

VIII. Conclusion

IX. Remerciement

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.