

Dossier de soutenance

# Boutique de bières l'échoppe



Développeur web et web mobile

- Niveau 5 -

Session de formation : Mars 2023 – Novembre 2023





# Table des matières

---

<b>I.</b>	<b>Introduction</b>	<b>4</b>
<b>II.</b>	<b>Liste des compétences du référentiel couvertes par le projet</b>	<b>5</b>
<b>III.</b>	<b>Cahier des charges</b>	<b>6</b>
A.	<i>Contexte et motivations pour le projet</i>	6
B.	<i>Description du projet</i>	6
C.	<i>Méthode Agile et MVP</i>	7
D.	<i>Description fonctionnelle du site</i>	8
E.	<i>La réglementation</i>	10
1.	Règlement général du commerce en ligne	10
2.	Règlement général sur la protection des données	11
3.	Règlementation spécifique à la vente d'alcool en ligne	12
<b>IV.</b>	<b>Préparation au projet - spécificités technique</b>	<b>14</b>
A.	<i>Outil de gestion de projet – Trello</i>	14
B.	<i>Modèle conceptuel de donnée -le modèle MCD</i>	15
C.	<i>Charte Graphique et typographie</i>	20
D.	<i>Maquettages</i>	20
E.	<i>Langages et ressources logicielles</i>	23
1.	Les langages frontend	23
2.	Les langages backend	23
3.	Ressources logicielles	23
4.	Framework et Bundles	24
F.	<i>Architecture logicielle</i>	25
<b>V.</b>	<b>Réalisation du projet.</b>	<b>27</b>
A.	<i>Le Modèle</i>	27
1.	Les entités	27
2.	La Base de données	29
3.	Le Repository	30
B.	<i>Le Contrôleur</i>	33
C.	<i>La Vue</i>	36
D.	<i>Le responsive Design</i>	40
E.	<i>Le référencement</i>	41
<b>VI.</b>	<b>Présentation des fonctionnalités</b>	<b>44</b>
A.	<i>La création des filtres de produits</i>	44
1.	La construction du modèle	44
2.	La construction du formulaire	45
3.	Le rendu du formulaire et l'affichage des produits non filtrés	46
4.	Le Traitement du formulaire et l'envoi des données filtrées	49

B.	<i>Le panier</i>	51
1.	L'ajout au panier	51
2.	Affichage du panier	52
C.	<i>Le paiement par Stripe</i>	53
1.	La Session Checkout	54
2.	Validation du paiement et enregistrement de la commande	56
<b>VII.</b>	<b>Sécurité</b>	<b>60</b>
A.	<i>Sécurisation de la connexion</i>	60
B.	<i>Sécurisation des routes</i>	61
C.	<i>Protection contre les injections SQL</i>	62
D.	<i>Protection contre le Cross Site Scripting (faille XSS)</i>	63
E.	<i>Protection contre les attaques CSRF (Cross-site Request Forgery)</i>	63
<b>VIII.</b>	<b>Recherches anglophones</b>	<b>64</b>
A.	<i>Extrait de la documentation Symfony Sessions</i>	64
B.	<i>Version traduite</i>	65
<b>IX.</b>	<b>Difficultés rencontrées</b>	<b>67</b>
<b>X.</b>	<b>Perspectives d'améliorations</b>	<b>68</b>
<b>XI.</b>	<b>Conclusion</b>	<b>69</b>
<b>XII.</b>	<b>Annexes – les maquettes</b>	<b>70</b>

# I. Introduction

---

Cela fait plusieurs années que mûrit en moi l'envie de reprendre mes études afin d'obtenir un travail stimulant et me permettant de m'épanouir davantage.

Aillant déjà pu appréhender des notions d'informatique au cours de ma formation initiale en biologie, je me suis tourné naturellement vers les métiers de l'informatique. Avant de me lancer dans cette reconversion, j'ai repris les notions de base à l'aide d'ouvrages et de cours disponibles sur le web, ce qui m'a permis de me conforter dans mon choix.

Sur les conseils de mon entourage, je me suis inscrit chez Elan formation, dans le but d'obtenir un titre de développeur web et web mobile de niveau 5. Au cours de cette formation intensive, j'ai pu découvrir et développer mes compétences me permettant de développer un site web.

A travers ce dossier, je vais vous présenter les différentes étapes dans la conception de mon projet de soutenance, une boutique de bières artisanales, imaginée sur un modèle associatif. Le lieu sera géré par les membres de l'association, les produits proposés seront uniquement fournis par des brasseurs locaux et les ventes s'effectueront en click'n'collect.

Ce projet couvrira l'ensemble des compétences du référentiel du titre professionnel.

## II. Liste des compétences du référentiel couvertes par le projet

---

### AT 1 : Développer la partie front-end d'une application web ou web mobile en intégrant les recommandations de sécurité

CP 1 : Maquetter une application

CP 2 : Réaliser une interface utilisateur web statique et adaptable

CP 3 : Développer une interface utilisateur web dynamique

CP 4 : Réaliser une interface utilisateur avec une solution de gestion de contenu ou e-commerce

### AT 2 : Développer la partie back-end d'une application web ou web mobile en intégrant les recommandations de sécurité

CP 1 : Créer une base de données

CP 2 : Développer les composants d'accès aux données

CP 3 : Développer la partie back-end d'une application web ou web mobile

CP 4 : Elaborer et mettre en œuvre des composants dans une application de gestion de contenu ou e-commerce

## III. Cahier des charges

---

Avant de se lancer dans la réalisation d'un projet web, il est important de définir les besoins auquel notre application va répondre et avoir une vue d'ensemble des fonctionnalités qui vont être mises en place. Grâce à cette réflexion, il nous sera possible d'estimer la durée nécessaire à la réalisation du projet. De cette estimation, nous définirons le produit minimum viable (**Minimum Viable Product** en anglais) afin de rendre un produit fonctionnel dans le délai de livraison.

### A. Contexte et motivations pour le projet

Le brassage de bières artisanales, dites "Craft-Beer" est un phénomène en plein essor dans notre région comme partout ailleurs. Cependant, étant donné le nombre croissant de brasseries, il devient de plus en plus difficile de se faire connaître du grand public et de trouver des espaces de vente. Les microbrasseries sont peu exposées dans les grandes surfaces, leur production étant souvent trop faible. Les boutiques spécialisées ont un espace de vente très restreint et ne proposent que quelques productions locales à des prix souvent élevés. La vente en direct est une solution moins onéreuse pour le consommateur, mais il devient compliqué de s'approvisionner chez différents brasseurs de la région sans augmenter ses frais de déplacement.

Il existe une alternative que je souhaiterais développer à travers ce projet, le magasin coopératif.

### B. Description du projet

L'échoppe est un bar associatif et un magasin spécialisé dans la vente de bières artisanales. C'est un projet factice que je développe dans le cadre de ma formation et ne répond pas à la demande d'un client. Je te tiens également à préciser que malgré la présence de vrais produits, les prix affichés seront également factices.

Le site s'articulera sur 2 axes principaux :

- Une boutique de vente en click'n'collect dont le but est de centraliser les commandes, effectuer des achats groupés auprès de divers brasseurs de la région, proposant ainsi une large gamme de produits locaux à des prix accessibles.
- Une partie sera dédiée à l'organisation de la vie de l'association. Cela se traduira par la création d'un forum permettant aux membres actifs de l'association d'organiser, entre autres, les roulements des permanences. Ce forum disposera également d'une partie accessible à tous les membres inscrits sur le site, leur permettant ainsi d'échanger sur divers sujets autour de la bière, de l'organisation de l'association et partager des événements.  
Un autre outil sera également développé pour organiser la vie de l'association et du site : un agenda permettant de mettre en avant les événements particuliers organisés au sein de l'association (concerts, dégustations) et autour de la bière en général (événements type festival

de la bière, dégustations et portes ouvertes chez un brasseur partenaire de l'association, etc....).

Ce site vise avant tout à toucher une clientèle locale, amateur de bières artisanales. C'est une clientèle sensible aux enjeux environnementaux et qui souhaite réduire son impact en réduisant ses déplacements et en s'approvisionnant de manière locale.

## C. Méthode Agile et MVP

Avant de développer les différentes fonctionnalités du site, il est important de définir de bonnes pratiques dans la gestion de projet. Pour l'élaboration d'une application web, la méthode Agile correspond parfaitement à nos besoins. En effet, cette approche permet d'apporter souplesse et performance dans la gestion de projet. Etant donné le projet est réalisé par un seul développeur et en absence d'un client certaines méthodes Agile ne seront pas développées ici, notamment celles concernant la partie humaine et collaborative des projets.

Notons tout de même les 3 réponses concrètes aux problématiques de gestion de projets que nous utiliserons tout au long du processus de création de l'application :

- La réactivité : pour pallier le manque de temps dans l'intégration des solutions il est important de pouvoir cibler les fonctionnalités à mettre en place dans le temps imparti. C'est dans cette optique que nous définirons un MVP (Minimum Viable Product).
- La flexibilité : il s'agit ici d'identifier la diversité des supports d'utilisation de l'application afin de s'adapter aux usages les plus courants (Web et mobile). On parlera alors de responsive design pour adapter notre application à un maximum de support numérique.
- La puissance : il faudra garantir la performance du système dans la collecte et l'accès aux données. Notre application devra être conforme à la législation en vigueur (règlements e-commerce, RGPD et droits spécifiques à la vente d'alcool) et répondre aux standards de sécurité web.

Le **MVP** ou **Minimum Viable Produit** (Produit minimum viable) est une méthode Agile qui a pour but de sortir un produit uniquement avec la fonctionnalité la plus attendue et de le mettre en fonctionnement le plus rapidement possible afin de le confronter au marché. Cette technique permet notamment de diminuer le taux d'échec des projets et de minimiser les risques pour une entreprise lors de l'élaboration d'un projet. Cela nous permettra de livrer un produit fonctionnel tout en respectant l'échéance qui nous est imposée. Dans notre cas, la date de fin de formation. Des améliorations pourront être apportées par la suite.

C'est dans cet optique que nous nous concentrerons sur l'élaboration de la partie boutique du site, car c'est l'élément principal à mettre en place pour rendre le site fonctionnel. Les parties forum et agenda pourront être développées par la suite et ne sont pas nécessaires au bon fonctionnement du site. De plus, pour que ces outils aient une réelle utilité sur notre site, il faudrait que ce dernier présente déjà un certain trafic.



## D. Description fonctionnelle du site

Compte tenu des éléments décrits précédemment, nous pouvons à présent décrire les fonctionnalités à développer en priorité sur notre site.

### Le menu :

Il sera composé de 2 parties. A gauche, on devra retrouver des liens vers les différentes sections du site, La boutique ainsi que la page d'accueil. C'est dans cette partie que se greffera plus tard les liens vers le forum et l'agenda d'évènements. A droite, on retrouvera les sections concernant la session en cours. Le profil de l'utilisateur avec un sous-menu lui permettant de se connecter/déconnecter et d'accéder à sa page profil. Un lien vers le panier.

### La page d'accueil :

La page d'accueil est la page de présentation du site. Cette page présentera succinctement l'association, puis la liste des nouveautés sera mise en avant. Enfin un bandeau explicatif donnera une idée claire du fonctionnement de la boutique. Elle contiendra également les liens vers les pages suivantes :inscription, boutique.

### Inscription :

Un utilisateur doit pouvoir s'inscrire et avoir un compte utilisateur pour accéder à toutes les fonctionnalités du site. A travers un formulaire, il devra obligatoirement renseigner son nom, prénom, email, pseudo, date de naissance, adresse complète et son numéro de téléphone. Il devra renseigner un mot de passe dans 2 champs, permettant de valider ce dernier. Pour valider le formulaire, il devra également accepter les conditions générales d'utilisation.

Pour rendre son compte actif l'utilisateur nouvellement inscrit devra valider son compte par l'intermédiaire d'un email qui nous permettra de vérifier son adresse.

### Connexion :

Un utilisateur inscrit se connecte au site en utilisant la combinaison email/mot de passe. La connexion est nécessaire à partir du moment où l'utilisateur souhaite valider son panier pour passer au paiement. La connexion au site lui permettra également d'accéder à son compte. Une fonctionnalité de récupération de mot de passe sera disponible via cette page en cas d'oubli.

### Profil :

Depuis la page profil, un utilisateur connecté à accès à ses coordonnées personnelles. Il doit pouvoir modifier si besoin : nom, prénom, pseudo, adresse, numéro de téléphone. Un lien d'accès vers un second formulaire permettra à l'utilisateur de modifier son mot de passe. Il aura également la possibilité de supprimer son compte depuis cette vue. Le profil permettra à l'utilisateur d'accéder à sa liste de commandes : celles en cours de traitement et celles déjà traitées. Pour chaque commande un lien lui permettra d'être redirigé vers le détail de chaque commande.

### La boutique :

Une première vue permet de visualiser la **liste des produits** disponibles à la vente. Cette page est la plus importante du site et doit permettre à l'utilisateur de trouver facilement les produits qui l'intéressent. C'est pourquoi une fonctionnalité de recherche multicritère devra être mise en place pour faciliter la navigation dans le catalogue de produits. Compte tenu des différentes pratiques dans la production des bières un affichage clair devra permettre à l'utilisateur de repérer les brassins particuliers (éphémères, saisonniers). L'utilisateur doit pouvoir cliquer sur un article pour être redirigé vers une page de détail produit.

La page **détail produit** est divisée en 3 parties :

- La première permet de présenter l'article mise en vente à l'aide d'une photo, d'une description et du prix. C'est dans cette partie qu'on retrouvera le formulaire d'ajout au panier. Dans ce formulaire l'utilisateur pourra renseigner la quantité de produits souhaité, soit par l'intermédiaire d'un champ dédié à cet effet, soit par l'intermédiaire de boutons interactifs.
- La deuxième partie doit permettre à l'utilisateur de connaître précisément toutes les spécifications du produit : brasseur, liste des ingrédients, taux d'alcool, degré d'amertume, styles de la bière.
- La troisième partie listera les avis donnés par les utilisateurs sur le produit. Un lien permettra de rediriger l'utilisateur connecté vers la page création d'un nouvel avis.

### Les avis :

Chaque utilisateur inscrit pourra donner son avis sur les produits. Un avis sera constitué d'une note (de 1 à 5) et d'une description. Un utilisateur ne pourra donner qu'un seul avis par produit, mais il lui sera possible de modifier son ancien avis s'il le souhaite.

### Le panier :

Cette vue permettra de lister les articles que l'utilisateur a sélectionné. Depuis cette vue l'utilisateur pourra modifier la quantité de chaque article et les retirer du panier. Il lui sera également possible de supprimer l'intégralité du panier. Un lien sera présent pour passer à l'étape suivante. C'est sur cette vue qu'apparaîtra également la cotisation annuelle de l'utilisateur. Cette cotisation sera automatiquement ajoutée au panier si elle n'est pas à jour.

L'étape suivante sera uniquement accessible aux utilisateurs connectés et vérifiés. Ce sera la dernière étape avant le paiement. L'utilisateur pourra vérifier une dernière fois le contenu de sa commande et vérifier l'adresse de facturation qui apparaîtra sur la facture. Un lien permettra de rediriger l'utilisateur vers le service de paiement.

### Le paiement :

Le paiement s'effectuera sur une plateforme externe et sécurisée : Stripe. L'utilisateur renseignera ses coordonnées bancaires puis sera redirigé vers notre site. Une page de confirmation lui signalera la prise en compte de sa commande.

### Le back-office :

Une partie accessible uniquement à l'administrateur du site permettra de gérer l'intégralité de la boutique. Elle sera composée d'une interface simple d'utilisation et devra répondre aux fonctionnalités suivantes :

- Ajout/modification/suppression des éléments du catalogue. Cela concerne les brasseries et les références de bières.
- Ajout/modification/suppression des différents styles de bière et types de production.
- Modération des utilisateurs : Possibilité de voir le profil complet des utilisateurs pour les contacter en cas de litige, possibilité de modifier les profils (si la demande est faite par l'utilisateur présent en boutique). Bloquer l'accès aux commentaires à un utilisateur qui tiendrait des propos offensants ou injurieux.
- Modération des commentaires. Suppression des commentaires offensant, injurieux.
- Gestion des commandes : Affichage de la liste des commandes, possibilité de visualiser les commandes non préparées et modifier leur statut lorsque la commande est prête. Possibilité d'accéder au détail de la commande et de modifier cette dernière si besoin.
- Partie facturation : Affichage des factures éditées, afin de les transmettre à la comptabilité.

## E. La réglementation

### 1. Règlement général du commerce en ligne

Pour mettre en ligne un site de e-commerce certaines règles sont à respecter et des mentions obligatoires à afficher :

- **Les mentions légales** sont les informations permettant d'identifier notre association. Ces mentions contiendront les informations suivantes : identité de l'association, adresse du siège, numéro d'immatriculation (N° de siret), un email et un numéro de téléphone pour contacter l'association et l'identité de l'hébergeur du site.

Une association à but non lucratif n'étant pas assujettie à la TVA, le numéro d'identification à la TVA ne sera pas affiché dans notre cas.

- **Les conditions générales de vente** (CGV) encadrent les relations commerciales. Ces informations doivent permettre aux clients de connaître leurs droits et obligations lors de la vente de nos produits. C'est une obligation de transparence qui doit permettre de réduire le risque de litige entre le client et l'association. Les conditions générales de vente contiendront notamment les informations suivantes : les caractéristiques essentielles des biens, le prix TTC en euros, les modalités d'exécution du contrat, les modalités de paiement, le service après-vente, durée de l'abonnement et conditions de résiliation, modalités de règlement des litiges. Un lien vers la plateforme européenne de règlement en ligne des litiges (RLL) devra également apparaître.

Ces mentions seront présentes sur notre site, sur des pages dédiées. Des liens d'accès vers ces pages seront présents dans la partie footer (pied-de-page) et visible depuis n'importe quelle page du site.

Il existe également une autre réglementation qui est générale à tout site web, il s'agit du règlement général sur la protection des données (RGPD).

## 2. Règlement général sur la protection des données

Le règlement général sur la protection des données (RGPD) est une réglementation européenne relative à la protection des données à caractère personnel. En France, c'est la CNIL qui veille au respect de cette réglementation.

Une donnée personnelle constitue toute information permettant d'identifier directement (ex: nom, prénom) ou indirectement (ex : N° identifiant client, num. de téléphone...) une personne physique.

Dans le cadre de cette réglementation, nous avons plusieurs obligations vis-à-vis utilisateurs présents sur notre site.

Tout d'abord nous devons les **informer** de la manière dont nous allons récupérer et utiliser leurs données et justifier leurs utilisations. Nous devons également leur renseigner le lieu de stockage et la durée de sauvegarde de ces données. Ces informations seront visibles depuis une page dédiée nommée "protection des données et vie privée". Un lien d'accès sera présent dans le pied de page de notre site.

Nous devons également prévoir une bannière "pop-up" qui s'affichera dès l'arrivée de l'utilisateur sur notre site, pour l'informer avec transparence sur l'utilisation de ces données. Il aura alors le choix d'accepter ou non les cookies non essentiels. Seul le cookie de session sera utile dans notre cas.

Concernant les données enregistrées en Base de données, il est important de recueillir le consentement de l'utilisateur, cela se traduira par une case à cocher (décochée de base) présente en fin de formulaire d'inscription. Cette case sera précédée de la mention : "J'accepte les conditions générales d'utilisation du site et sa politique de confidentialité".

Donnée personnelle	Justifications
Nom, Prénom, adresse postale	Données utilisées pour la facturation
Email et mot de passe	Authentification sur le site et sécurisation du compte
Date de naissance	Confirmer la majorité de l'utilisateur. Sécurité pour l'association en cas de litige et l'accès à la vente d'alcool est interdite aux mineurs
Pseudo	Anonymisation des commentaires. Utilisation future pour le forum.
Facture (Numéro)	Document comptable obligatoire

Commande (identifiant, détails)	Nécessaire pour la préparation des commandes, pouvant être utilisé à des fins statistiques.
---------------------------------	---

On doit également respecter le principe de minimisation des données. Les données collectées doivent être adéquates, pertinentes et limitées à ce qui est nécessaire au regard des finalités pour lesquelles elles sont traitées ». (Source CNIL)

Voici la liste des informations présentes sur la page “protection des données et vie privée” :

- Durée de conservation des données : en absence d’activité sur le site durant 3 ans, les données seront retirées de la base de données, à l’exception des factures dont la durée de sauvegarde est de 10 ans.
- Lieu de stockage : serveur Européen.
- Accès aux données : administrateur(s) du site l’échoppe.

Il est également important de **sécuriser** les données présentes sur le site. Pour ce faire, certaines règles sont à respecter :

- L’ensemble du parcours de vente doit être en https.
- Il faut imposer aux clients un mot de passe complexe à la création de leurs comptes.
- Ne pas transmettre d’information personnel par email.
- Sécuriser la transaction bancaire. Dans notre cas c’est Stripe qui se charge de cette partie.

Dernier point à souligner concernant le RGPD, il est important de laisser au client un **droit de regard** sur ces données. Une adresse email doit être renseignée sur la page “vie privée” afin de permettre à l’utilisateur de contacter l’administration du site pour demander l’accès, la rectification ou l’effacement de ses données. On parle dans ce cas de **droit à l’oubli**. Sur notre site, la modification ou la suppression du compte pourra s’effectuer directement par l’utilisateur depuis sa page “profil”. Lors de la suppression d’un compte utilisateur, l’ensemble des commandes seront supprimées et les reviews de l’utilisateur seront anonymisées.

### 3. Règlementation spécifique à la vente d’alcool en ligne

La vente d’alcool en ligne est soumise à une réglementation particulière. Tout d’abord il est interdit de vendre de l’alcool aux mineurs. Un affichage clair doit être présent sur le site, en bas de page. Un rappel sur chaque formulaire de commande est également à prévoir. La présence d’un bandeau fourni par les services de l’Etat devra également apparaître sur le site.



Figure 1- bandeau réglementaire - code de la santé publique

Nous rajouterons également une fenêtre “pop-up” demandant à l'utilisateur de valider son âge avant de rentrer sur le site par l'intermédiaire d'un formulaire oui/non. C'est également dans le but d'éviter tout litige que l'on demandera à l'utilisateur de renseigner sa date de naissance lors de son inscription.

Autre action de sensibilisation aux dangers de l'alcool, un message d'avertissement “ L'abus d'alcool est dangereux pour la santé, à consommer avec modération” devra apparaître clairement sur le site.

Une association souhaitant vendre de l'alcool au public est, au même titre que les restaurants et débits de boissons en devoir de posséder une licence. Cependant, si cette vente s'effectue de manière privée entre membres de l'association, l'organisme peut se passer (sous certaines conditions) de cette licence. C'est pourquoi il sera impératif pour chaque utilisateur de s'acquitter d'une cotisation annuelle pour justifier son inscription à l'association.

## IV. Préparation au projet - spécificités technique

### A. Outil de gestion de projet – Trello

Trello est un outil de gestion de projet en ligne. Il repose sur un système de cartes représentant des tâches. A l'intérieur de chaque carte, les tâches peuvent être décrites de manière précise et découpées en sous-parties à l'aide de check-lists afin de nous guider dans le bon déroulement de ces dernières. Nous avons classé ces cartes dans 3 listes : à faire, en cours, terminé.

Pour respecter et mettre en pratique les techniques de gestion de projet présentées précédemment (AGILE et MVP) nous nous sommes basés sur la méthode MoSCoW. Sous cet acronyme se cache une technique permettant de définir l'importance des tâches à réaliser :

- M : Must have this. "doit être fait", en priorité. (Vital)
- S : Should have this if at all possible. Devrait être fait, dans la mesure du possible. (Essentiel)
- C : Could have this if it does not affect anything else. Pourrait être fait, dans la mesure du possible. Ce sont des petits plus, pouvant être ajoutés. (Confort)
- W : Won't have this time but would like in the future : Exclu du projet, mais sera fait plus tard. (Luxe)

Voici quelques exemples de tâches extraites de mon projet Trello. A noter que l'ajout des étiquettes Backend et Frontend sur les cartes est un choix personnel et ne font pas partie de la méthode MoSCoW. Ces étiquettes permettent de visualiser le type de tâches à réaliser pour chaque carte dans le but de varier le travail tout au long du projet.

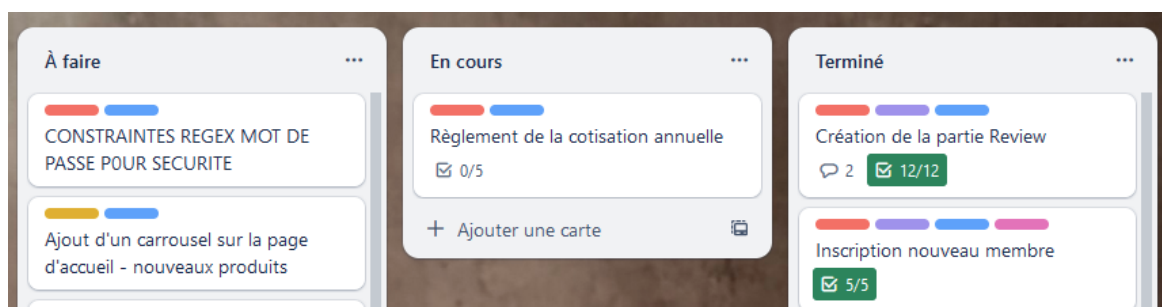


Figure 2-Exemples de cartes – projet "l'échoppe"

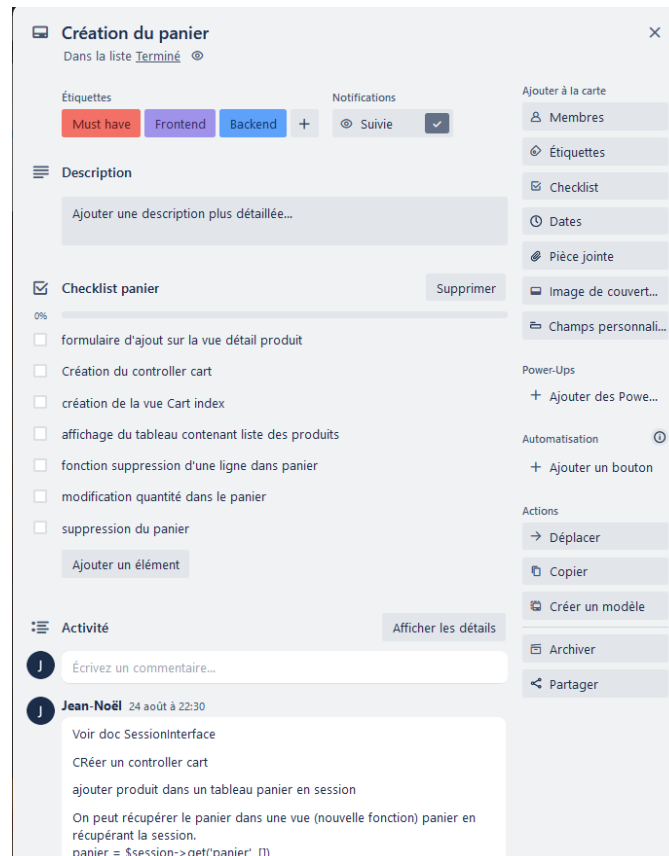


Figure 3- Détail d'une carte Trello - Projet "l'échoppe"

## B. Modèle conceptuel de donnée-le modèle MCD

Maintenant que le projet est clairement défini dans le cahier des charges, il faut réfléchir à l'organisation de son système d'information. La modélisation des données est le processus de représentation graphique des données. Elle va nous permettre d'organiser les données et comprendre comment elles interagissent entre elles dans le but de nous guider dans le processus de développement de l'application.

Pour ce faire, nous pouvons nous appuyer sur une méthode de conception française appelée la méthode MERISE, laquelle définit un certain nombre de schémas et diagrammes, dont le MCD (ou Modèle Conceptuel des Données).

Le MCD est un diagramme permettant une représentation schématique d'une base de données relationnelle et vise en outre à modéliser des tables et les relations existantes entre elles. Plutôt que de se focaliser sur les valeurs des données on modélisera la structure et les relations des entités.

Le MCD possède un vocabulaire qui lui est propre. On parlera d'entités (les tables en b.d.d.) et d'associations (les relations entre les tables).

- Une **entité** est constituée d'attributs. Un attribut souligné est appelé une clé primaire. Une clé primaire est un identifiant unique qui sera associé à chaque ligne de notre table. Les entités sont représentées par un rectangle



- Une **association** définit une relation existante entre plusieurs entités. Elle est définie par un nom (souvent un verbe, qui définit la relation entre 2 entités) et éventuellement des propriétés.

Lorsque 2 entités sont reliées par une association on définira également leurs **cardinalités** (0-1 ; 1-1 ; 0-n ; 1-n. )

Dans notre projet les entités sont définies comme suit :

- Un **produit** possède un identifiant, une date de création, une désignation, une description, un prix, une quantité (ou unité de vente), un volume (pour une unité), un stock, des ingrédients, un taux d'alcool, un taux d'amertume, une image, un slug (*voir partie référencement*), un statut disponible à la vente.
- Un **type de production** est défini par un identifiant et un nom (permanente, éphémère, saisonnière, etc...).
- Un **producteur** est défini par un identifiant, un nom, une adresse, un code postal, une ville, un site web(optionnel), un lien vers un réseau social (optionnel).
- Un **style de bière** est défini par un identifiant et un nom.
- Un **utilisateur** est défini par un identifiant, une adresse mail, un pseudo, un mot de passe, un nom, un prénom, une date de naissance, une adresse, un code postal, une ville, un numéro de téléphone, une date de création, une date de fin de cotisation, un rôle, un statut vérifié et un statut banni.
- Une **commande** est définie par un identifiant, une date de création, une référence, une référence Stripe, un statut payé, un statut préparé, un montant de cotisation.
- Un **avis** est défini par un identifiant, une note, une description, une date de création, une date de modification.

Nous pouvons à présent définir les relations entre ces entités ainsi que les cardinalités :

- Un produit est défini par 1 seul type de production (relation 1-1) et un type de production peut définir 0 ou plusieurs produits (relation 0-N).
- Un produit est produit par 1 seul producteur (relation 1-1) et un producteur peut produire 0 ou plusieurs produits (relation 0-N).
- Un produit est défini par 1 ou plusieurs types de bières (relation 1-n) et un type de bière peut définir 0 ou plusieurs produits (relation 0-N).
- Un produit peut est contenu dans 0 ou plusieurs commandes (relation 0-N) et une commande peut contenir 1 ou plusieurs produits (relation 1-N).

- Un produit peut recevoir 0 ou plusieurs avis (relation 0-N) et un avis ne peut être donné qu'à un seul produit (relation 1-1).
- Un avis ne peut être donné que par 1 utilisateur (relation 1-1) et un utilisateur peut donner 0 ou plusieurs avis (relation 0-N).
- Un utilisateur peut passer 0 ou plusieurs commandes (relation 0-N) et une commande ne peut être passée que par un utilisateur (relation 1-1).

On obtient alors un diagramme complet que l'on représentera ci-dessous. A noter que ce modèle est une représentation globale du projet. Seules les entités de couleur bleue font partie du MVP et sont implémentées dans le projet.

Un deuxième diagramme découlant du MCD est également présenté : le MLD (Modèle Logique des Données). Ce diagramme apporte des spécifications et propriétés plus détaillées. Il se rapproche davantage de la modélisation telle qu'elle sera représentée en base de données. On se focalise sur la structure des données en affichant le type de chaque attribut et les clés étrangères présentes dans chaque entité.

Dans notre base de données les entités seront représentées par une table et leurs relations par des clés primaires.

Une **table** est un tableau regroupant un ensemble de données représenté par des **colonnes** correspondant à des catégories de données (ex : nom, prénom...). Chaque entrée dans une table est représentée par une **ligne** et contient les valeurs correspondantes aux catégories (ex : Dupont, Jean). Chaque enregistrement est défini par un identifiant unique qu'on appelle **clé primaire**.

Dans une table, une clé étrangère est une contrainte permettant de garantir l'intégrité référentielle entre 2 tables. La structure de la clé étrangère doit correspondre à la structure de la clé primaire de la table avec laquelle elle noue une relation.

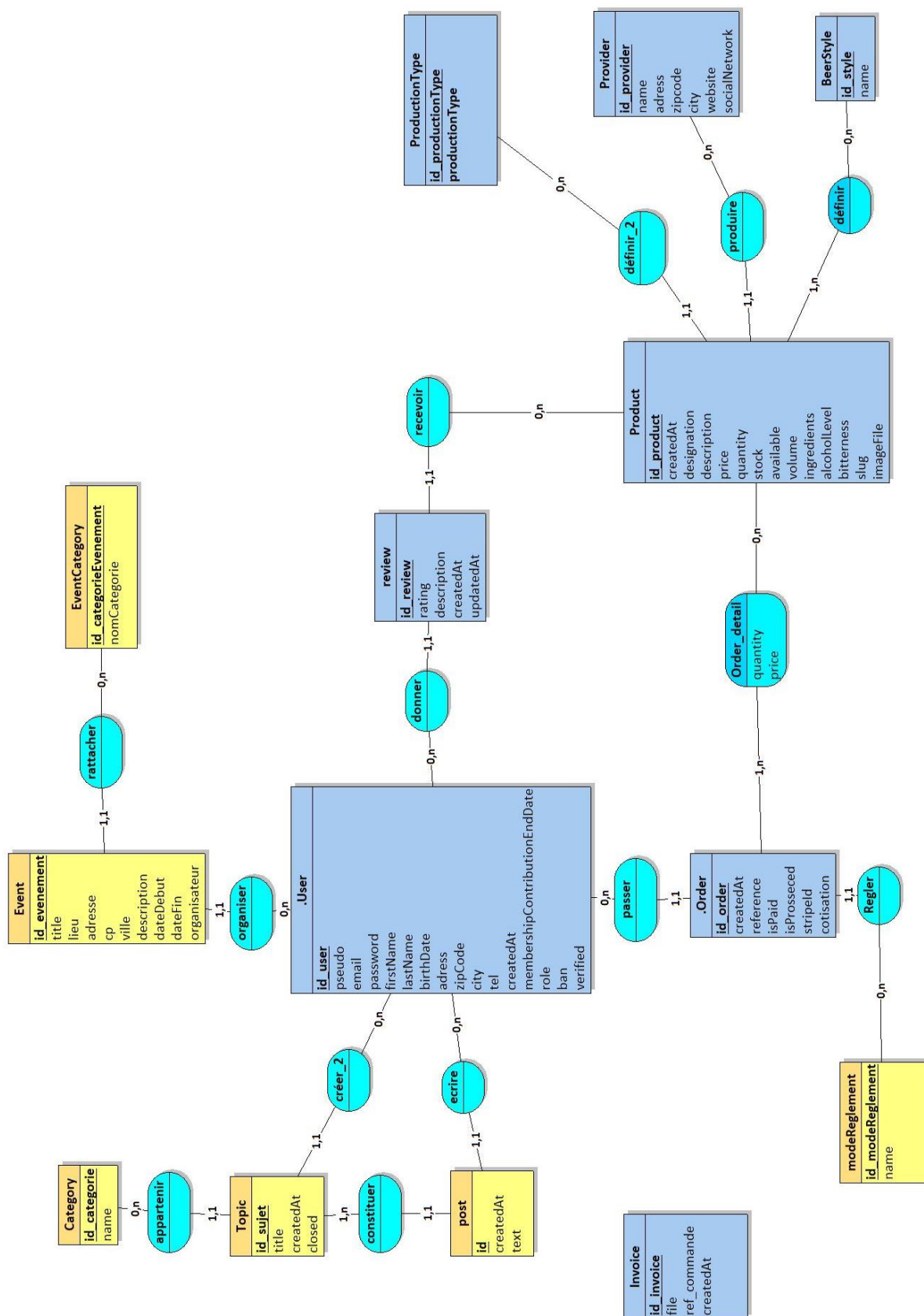


Figure 4- Diagramme MCD (Modèle Conceptuel des données) - Projet "l'échoppe". Réalisé avec le logiciel looping

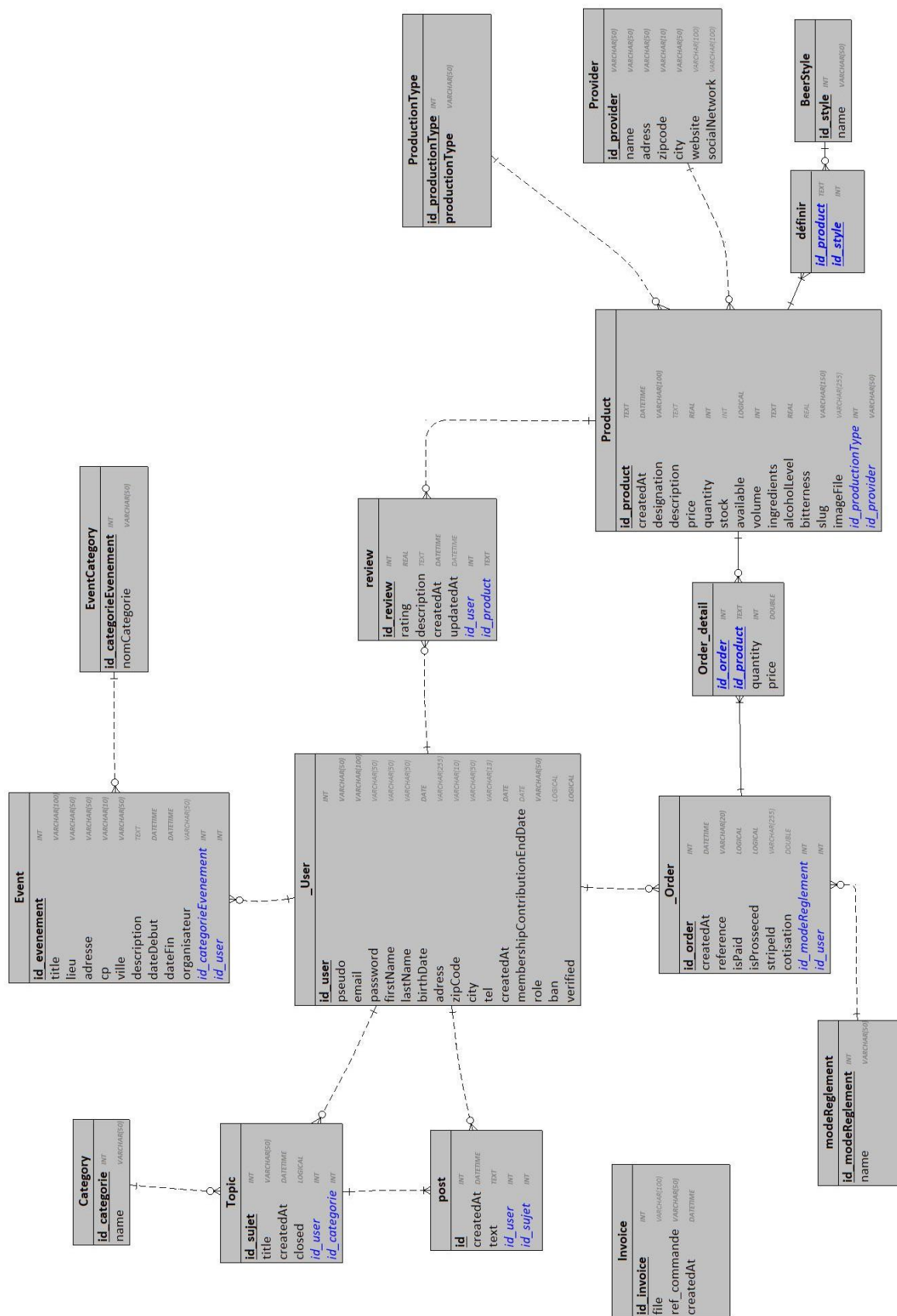


Figure 5-Diagramme MLD (Modèle Conceptuel des données) - Projet "l'échoppe". Visualisation des clés étrangères des entités

## C. Charte Graphique et typographie






Figure 6- Logo du site "l'échoppe"

N'ayant pas de compétences de graphiste, la création du logo a été confiée à un outil de conception proposé par Renderforest. Une IA permet de guider l'utilisateur dans la conception de son logo. Le but était de créer un logo faisant apparaître une chope de bière pour mettre en évidence le jeu de mot contenu dans le nom de l'association ainsi les éléments qui constituent ce breuvage (le houblon et les céréales).

Concernant les couleurs, le choix de la couleur principale s'est tourné vers une couleur chaude, un jaune tirant sur l'orange pouvant rappeler la couleur de la bière. Afin d'apporter de la clarté au site, les 2 autres couleurs sont des complémentaires à la première. Tout d'abord, une couleur froide, le bleu foncé sera utilisé pour mettre en avant les éléments d'interaction de l'utilisateur ainsi que le prix des produits. C'est une couleur apaisante et rassurante qui doit mettre en confiance l'utilisateur.

Pour de petits détails un brun a été ajouté à la palette de couleurs, complémentaire à la couleur principale.

Code hexadécimal	Aperçu
#F2A03D	
#0872A6	
#A66619	

La typographie **Poppins** fait partie de la bibliothèque de polices de Google Fonts. C'est une police très répandue, simple et lisible au style sans Serif.

## D. Maquettages

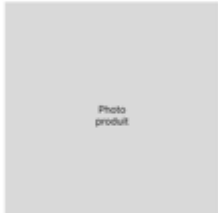
Le maquettage est un élément important dans la réalisation d'un site web. En effet, avant de se lancer dans le développement du site il est nécessaire de structurer ses idées, et de définir l'agencement des différentes pages de l'interface utilisateur.

Dans notre cas l'objectif n'est pas de réaliser une maquette "pixel perfect", mais de construire les pages en y intégrant toutes les fonctionnalités présentes pour chacune d'entre-elles et de définir l'agencement des différents éléments graphiques. Cette conception nous permet également d'anticiper la manière dont ces éléments seront construits pour s'adapter aux différents supports. Nous avons construit ces maquettes en se focalisant sur l'adaptabilité de l'application dans sa version mobile, chaque élément sera construit pour s'adapter facilement en fonction de la largeur de l'écran.

Ci-dessous, quelques extraits du maquette version Wireframe et du maquettage final:



Retour boutique



## Désignation du produit

Description du produit Sed ut perspiciatis unde omnis iste natus error sit voluptatem accusantium doloremque laudantium, totam rem aperiam, eaque ipsa quae ab illo inventore veritatis et quasi architecto beatae

3,95 €  
(soit 11,96 € le litre)

Quantité  
▼ 5 ▲

Ajouter au panier

### Informations détaillées

Brasserie	Brasserie 1
Ingédients	Malt d'orge (Pils, Munich 20), houblons (Pils, Mistral, Triskel), levures (Saflma Un-05)
Taux d'alcool	5 °C
Amertume	56 I.B.U
Type de Bière	I.P.A.

### Avis

Ajouter un avis

5 reviews

Note : 4.2 / 5

Utilisateur 56

Note : 5 / 5

Description du produit Sed ut perspiciatis unde omnis iste natus error sit voluptatem accusantium doloremque laudantium, totam rem aperiam, eaque ipsa quae ab illo inventore veritatis et quasi architecto beatae

Utilisateur 5

Note : 5 / 5

Description du produit Sed ut perspiciatis unde omnis iste natus error.

Utilisateur 78

Note : 5 / 5

Description du produit Sed ut perspiciatis unde omnis iste natus error sit voluptatem accusantium doloremque laudantium.

Voir plus

L'abus d'alcool est dangereux pour la santé, à consommer avec modération.

#### Notre association

- > Mentions légales
- > Conditions générales de vente
- > Conditions d'utilisation
- > Données personnelles

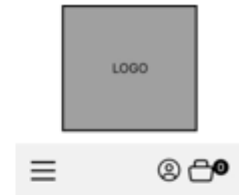
#### Nous contacter



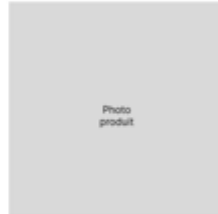
#### Informations

01, rue du Paradis du Houblon  
67000 Strasbourg  
lechoppe.asso@exemple.com

Copyright 2023 - Association l'échoppe



Retour boutique



## Désignation du produit

Description du produit Sed ut perspiciatis unde omnis iste natus error sit voluptatem accusantium doloremque laudantium, totam rem aperiam, eaque ipsa quae ab illo inventore veritatis et quasi architecto beatae

Quantité  
▼ 5 ▲

Ajouter au panier

### Informations détaillées

Brasserie 1
Malt d'orge (Pils, Munich 20), houblons (Pils, Mistral, Triskel), levures (Saflma Un-05)
5 °C
56 I.B.U
I.P.A.

### Avis

Ajouter un avis

5 reviews

Note : 4.2 / 5

Utilisateur 56

Note : 5 / 5

Description du produit Sed ut perspiciatis unde omnis iste natus error sit voluptatem accusantium doloremque laudantium, totam rem aperiam, eaque ipsa quae ab illo inventore veritatis et quasi architecto beatae

Utilisateur 5

Note : 3 / 5

Description du produit Sed ut perspiciatis unde omnis iste natus error.

Utilisateur 78

Note : 4 / 5

Description du produit Sed ut perspiciatis unde omnis iste natus error sit voluptatem accusantium doloremque laudantium.

Voir plus

L'abus d'alcool est dangereux pour la santé, à consommer avec modération.

#### Notre association

- > Mentions légales
- > Conditions générales de vente
- > Conditions d'utilisation
- > Données personnelles

#### Nous contacter



#### Informations

01, rue du Paradis du Houblon  
67000 Strasbourg  
lechoppe.asso@exemple.com

Copyright 2023 - Association l'échoppe

Figure 7- Wireframe de la vue détail produit, version Desktop (à gauche), version mobile (à droite) - site "l'échoppe" - Réalisé avec l'application web Figma.

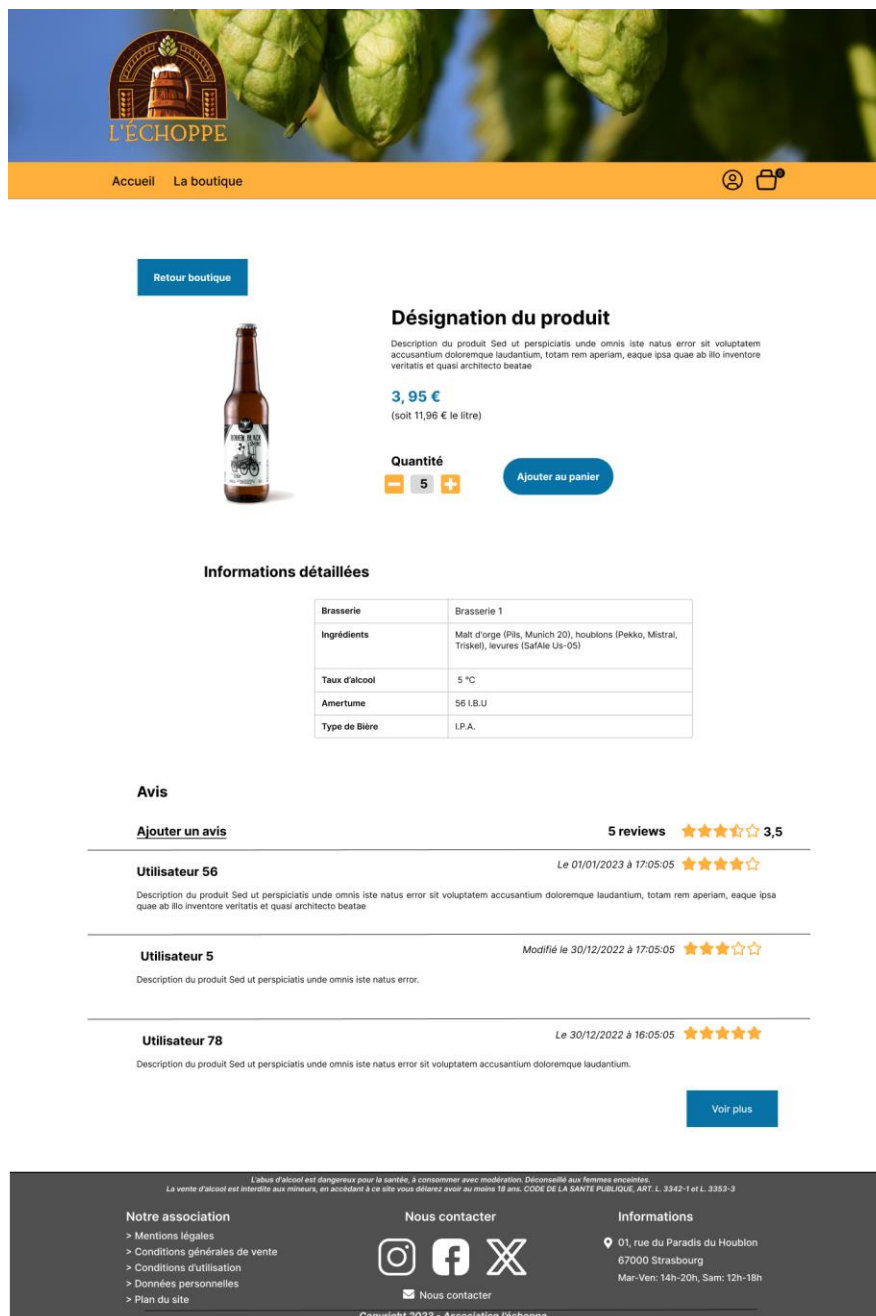


Figure 8- Maquette de la vue détail produit, version Desktop (à gauche), version mobile (à droite) - site "l'échoppe" - Réalisé avec l'application web Figma.



## E. Langages et ressources logicielles

Dernière étape avant de se lancer dans la réalisation pratique d'une application web, le développeur doit faire des choix importants concernant les langages et outils adéquats pour développer son application.

### 1. Les langages frontend



**HTML 5** (ou Hypertext Markup Language) est un langage de balise incontournable dans le domaine web. Il permet de construire et structurer le squelette d'une page web.

**CSS3** (ou Cascading style Sheets, feuille de style en français) est un langage indissociable de HTML. Le CSS permet de mettre en forme les pages web en appliquant des règles aux différentes balises HTML.



**Javascript** permet d'implémenter des mécanismes plus complexes à une page web. Il apporte de l'interactivité et rend nos pages dynamiques. Dans ce projet JavaScript est utilisé sans Framework, on parle alors de Vanilla JS.

### 2. Les langages backend

**PHP 8.1.** (ou PHP : HyperText Preprocessor) est un langage de script s'exécutant côté serveur. Ce sera le langage le plus utilisé dans notre application. Ce langage pouvant être facilement intégré à du HTML, il permet notamment de réaliser des pages dynamiques, de gérer les cookies et traiter les données envoyées au serveur. C'est à la fois un langage procédural et un langage orienté objet.



**DQL** (ou Doctrine Query language) est un langage de requête de base de données et plus spécifiquement un langage de requête orienté objet. Contrairement au langage SQL, le langage DQL ne communique pas avec la base de données en utilisant les relations entre les tables et colonnes qui la constitue mais en utilisant notre modèle d'objet. Ce langage est lié à l'ORM (Objet Relationnal Mapping) Doctrine du Framework Symfony utilisé pour ce projet. Nous détaillerons ces points plus tard.

### 3. Ressources logicielles

**Visual Studio Code** est un éditeur de code et un IDE (Environnement de Développement intégré) créé par Microsoft. Il supporte de nombreux langages et est gratuit. De nombreuses extensions sont disponibles et téléchargeables directement à l'intérieur de l'application pour faciliter la vie du développeur.







**Git** est un logiciel de gestion de version permettant de sauvegarder l'état du projet au fil du temps. Un système de branche permet de développer de nouvelles fonctionnalités au projet, sans toucher à l'intégrité de ce dernier en cas de problème et de développer plusieurs fonctionnalités en parallèle. C'est un outil très utile pour travailler en équipe, mais ce système de branche s'avère également pratique lorsqu'on développe seul une application.

**GitHub** est une plateforme qui peut contenir des dépôts de code dans un stockage dans le cloud.



**Laragon** est un environnement de développement web qui regroupe les serveurs suivants :

- Apache HTTP serveur permettant d'interpréter PHP.
- MySQL, serveur de base de données relationnelles.

Dans notre cas, nous utiliserons uniquement le serveur MySQL fourni par Laragon car le Framework Symfony (voir ci-dessous) contient son propre serveur PHP.

**Heidy SQL** est un outil d'administration de base de données fourni avec un éditeur SQL.



**Composer** est un logiciel gestionnaire de dépendances libres. Il permet d'installer, mettre à jour et supprimer les bibliothèques nécessaires au projet, dont Symfony. La liste complète des dépendances installées sera stockée dans un fichier `composer.json` visible dans notre projet.

**Mailhog** est une petite application permettant de tester l'envoi d'emails dans un environnement de développement local.

## 4. Framework et Bundles



Symfony

**Symfony 6.3.3** est un Framework PHP orienté objets. Il regroupe de nombreux composants qui facilite et accélère grandement le développement d'une application. On peut y ajouter des bundles (modules) permettant d'augmenter encore ses fonctionnalités. Nous décrivons le fonctionnement de ce Framework plus en détail dans ce dossier, mais notons déjà que pour ce projet nous avons installé Symfony dans sa version complète, c'est-à-dire avec l'ensemble des composants de base. Pour un projet plus léger il est tout à fait envisageable d'installer qu'une partie des composants du Framework, en fonction des besoins.

**Doctrine** est un ORM (Object Relational Mapping) pour PHP. C'est une couche indépendante qui se base sur le modèle Object de l'application pour établir une connexion entre ce dernier et la base de données. Pour communiquer avec la base de données nous utiliserons le langage de requête qui lui est associé : le DQL décrit précédemment.





**Twig** est un moteur de Template. C'est un outil qui permet de générer des vues (le modèle des pages) de manière efficace et rapide. Il nous permettra d'intégrer facilement du PHP à l'intérieur de nos pages HTML. C'est un bundle installé par défaut sur Symfony.

**Panigator** est un bundle Symfony permettant de créer la pagination de notre liste de produits. On s'en servira également pour trier la liste des articles (par ordre de prix ou notes).

**EasyAdmin** est un bundle permettant la gestion du Back Office pour les applications Symfony. Il est rapide à mettre en place et customisable.

**TinyMCE** est un éditeur de HTML de type WYSIWYG, il nous sera utile dans la gestion de certains formulaires.

**Stripe** est une plateforme de traitement de paiements. Pour tester son fonctionnement dans notre environnement de développement nous aurons besoin d'installer certains outils comme un interface de ligne commande Stripe et le SDK-PHP (Software Development Kit) nous permettant d'accéder à l'API Stripe.

**KarserRecaptcha 3** est un bundle permettant d'intégrer facilement l'API reCAPTCHA V3 de google au Framework Symfony. Ce bundle permet de sécuriser nos formulaires en s'assurant qu'ils n'ont pas été soumis par un robot.

**Dompdf** est un bundle permettant de convertir un document HTML au format pdf. Nous l'utiliserons pour générer les factures.

## F. Architecture logicielle

Le Design Pattern (ou Patron de conception) est un modèle de référence qui sert à concevoir l'architecture d'un logiciel informatique en sous éléments plus simple. Il présente de nombreux avantages.

- C'est une méthodologie professionnelle reconnue.
- Le code est facilement réutilisable, car la structure est identique d'un projet à l'autre. (Standardisation)
- Plus forte maintenabilité du code, car les éléments sont séparés. On peut modifier des parties sans impacter sur le reste du projet. C'est donc également un avantage lors d'un travail en équipe.
- Facilite la lisibilité et la compréhension du code pour une personne ayant des connaissances dans le Design pattern utilisé.

Comme tous les Frameworks orientés objets, Symfony implémente un Design Pattern, le **MVP** ou **Modèle-Vue-Présentateur**, qui est un dérivé du MVC ou Modèle-Vue-Contrôleur avec lequel il est fréquemment confondu.

Le MVC est un concept de programmation qui consiste à séparer les différentes parties d'un projet web dans 3 sections bien distinctes : le modèle, la vue et le contrôleur. C'est le premier patron à avoir été créé dès 1979.

Le **contrôleur** reçoit l'interaction de l'utilisateur, également appelé requête. Il se charge de transmettre ces informations une fois contrôlées au modèle.

Le **modèle** est la couche métier de l'application. C'est cette couche qui s'occupe de récupérer les données envoyées depuis le contrôleur, d'interroger le système de gestion de base de données et de préparer les données reçues en réponse pour qu'elles soient facilement manipulables par la vue.

La **vue** apporte la réponse du modèle à l'utilisateur. Elle ne se préoccupe que de l'affichage des informations pas de son traitement. C'est également la vue qui permet à l'utilisateur d'interagir avec l'application en envoyant des requêtes au contrôleur.

Cependant cette pratique présente un problème logique, c'est le modèle qui doit préparer, formater et envoyer les données à la vue alors que cette tâche devrait être confié au contrôleur, d'où l'apparition du modèle MVP.

Le **présentateur** possède à peu de chose près les mêmes fonctionnalités que le contrôleur. C'est lui qui interroge le modèle. Le modèle lui retransmet les informations en réponses et le présentateur se charge de les contrôler et les formater pour les transmettre à la vue.

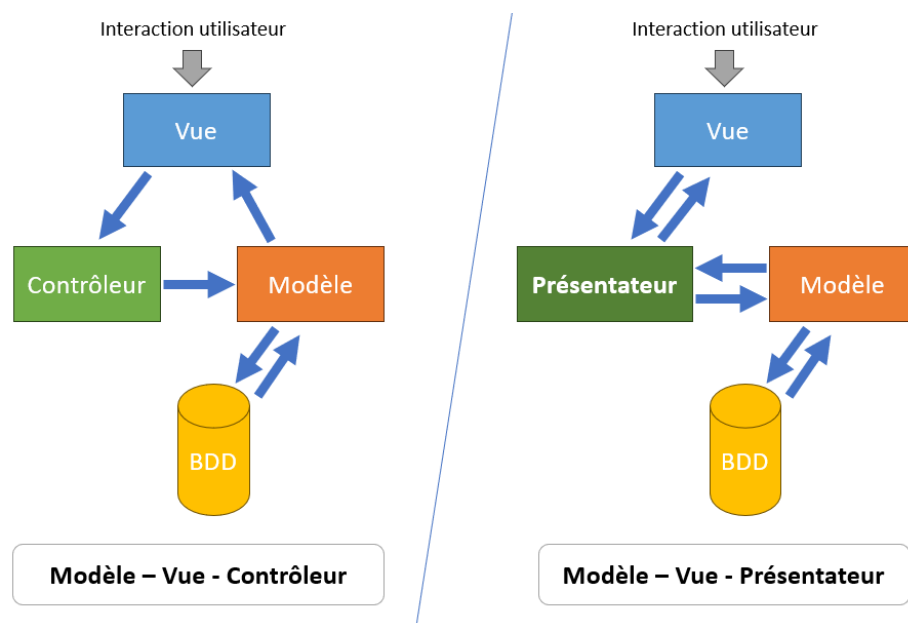


Figure 9 - Schéma représentant les différences entre le MVC et le MVP utilisé par Symfony

Maintenant que tous les éléments nécessaires à la réalisation du projet ont été défini, nous allons pouvoir passer à la partie concrète du projet et la mise en place des différentes couches de notre application en suivant l'architecture MVP décrite dans cette partie.

## V. Réalisation du projet.

---

### A. Le Modèle

Pour débiter notre projet Symfony nous avons commencé par mettre en place la couche modèle de l'application. Cette couche se retrouve sous la forme d'entités et d'un ORM. Chaque entité est une classe PHP qui peut être connectée à la base de données en passant par l'ORM Doctrine. Si tel est le cas les entités seront associées à un fichier Repository. Ce fichier contiendra les requêtes destinées à communiquer avec la base de données. C'est par l'intermédiaire du bundle maker de Symfony que nous avons pu mettre en place facilement les différents composants de notre application en respectant l'architecture MVP.

#### 1. Les entités

La première étape du projet a été la mise en place des entités en s'appuyant sur le modèle conceptuel de données décrit précédemment. Cette étape peut être découpé en 2 parties :

##### *a) Mise en place des attributs de classes :*

Prenons l'exemple de l'entité Product. Nous avons lancé la commande de création d'entité dans une console :

```
$ symfony console make:entity Product
```

Puis, nous avons renseigné chaque attribut de la classe en spécifiant leur type et les contraintes liées au type. Par exemple l'attribut désignation d'un produit est de type « string » avec une taille maximum définie à 100 caractères. C'est un attribut qui ne peut être nul.

```
New property name (press <return> to stop adding fields):  
> designation  
  
Field type (enter ? to see all types) [string]:  
> string  
string  
  
Field length [255]:  
> 100  
  
Can this field be null in the database (nullable) (yes/no) [no]:  
> no  
  
updated: src/Entity/Product.php
```

Nous avons répété l'opération pour tous les attributs de la classe et pour chaque classe du projet. Notons que l'attribut identifiant a été créé automatiquement lors de la création de l'entité.

Lorsque le formulaire est rempli, on obtient donc une entité constituée de ses attributs privés mais également ses méthodes publiques de base, permettant l'accès aux données (**Getter**) et leur affectation en base de données (**Setter**). On respecte ainsi le principe d'encapsulation de la Programmation Orienté Objet. Un fichier Repository est créé automatiquement lors de la création de chaque entité avec la commande `make:entity`, nous reviendrons plus tard sur ce point.

## b) Mise en place des relations entre entités :

Lorsque 2 entités sont liées entre elles par des liens d'association il est nécessaire de les renseigner dans nos entités pour que doctrine puisse créer les clés étrangères en base de données. Il existe 3 types de relations : `OneToMany`/`ManyToOne`, `ManyToMany` et `OneToOne`.

Prenons l'exemple de la relation entre l'entité `Product` et `Provider`. Nous avons décrit dans le MCD que :

« Un produit est produit par 1 seul producteur (relation 1-1) et un producteur peut produire 0 ou plusieurs produits (relation 0-N). »

Dans Doctrine cela se traduit par une relation `ManyToOne` en se plaçant dans `product` ou une relation `OneToMany` depuis `Provider`. Encore une fois, le bundle maker nous guide dans la création de cette relation :

Type	Description
<code>ManyToOne</code>	Each <code>Product</code> relates to (has) one <code>Provider</code> . Each <code>Provider</code> can relate to (can have) many <code>Product</code> objects.

La classe `Product` contiendra un objet `Provider` et `Provider` une collection d'objets `Product`. On parle alors de relation bidirectionnelle.

```
#[ORM\ManyToOne(inversedBy: 'products')]  
#[ORM\JoinColumn(nullable: false)]  
private ?Provider $provider = null;
```

```
#[ORM\OneToMany(  
    mappedBy: 'provider',  
    targetEntity: Product::class,  
    orphanRemoval: true  
)]  
private Collection $products;  
  
public function __construct()  
{  
    $this->products = new ArrayCollection();  
}
```

Figure 10 - Extrait du code des entités `Product` (Gauche) et `Provider` (Droite) décrivant leurs relations d'association.

Dans le code décrit en figure 10, 2 champs sont définis : « inverted By » et « mapped By ». Dans une relation entre 2 entités, il y a toujours une entité dite propriétaire et une dite inverse. L'entité propriétaire est celle qui contient la référence de l'autre entité. Dans notre cas c'est Product qui est propriétaire car elle contient l'identifiant Provider et donc provider est inverse.

Dans le cas d'une relation bidirectionnelle il faut expliciter la relation dans l'entité inverse pour faciliter la recherche d'éléments en partant de l'inverse (donc récupérer les produits fournis par un producteur).

Lors de la création d'une nouvelle instance de classe Provider un tableau contenant une collection de produit sera créé via le constructeur.

C'est le 3eme élément à décrire dans nos classes. Le constructeur n'est pas créé obligatoirement par Symfony, sauf dans le cas précédemment cité. Il permet de définir certains paramètres par défaut à mettre en place lors de la création de nouveaux objets. On l'utilise notamment dans toutes nos classes possédant un attribut « createdAt » afin d'attribuer la date et l'heure actuelle à l'objet au moment de sa création.

## 2. La Base de données

Nos entités sont maintenant créées et leurs relations établies, mais elles n'existent pas en base de données. Avant de pouvoir effectuer la migration 2 étapes sont à réaliser.

Tout d'abord, nous avons paramétré la connexion avec base de données. Pour ce faire nous avons dû modifier le chemin d'accès à la B.D.D. dans le fichier .env pour accéder au serveur MySQL de Laragon.

```
DATABASE_URL="mysql://root@127.0.0.1:3306/lehoppe?serverVersion=8.0.30"
```

Maintenant que le chemin d'accès est défini on peut faire appel à l'ORM pour créer la base de données à l'aide de la commande suivante :

```
$ php bin/console doctrine:database:create
```

Cette commande a pour but de créer le nom de la base de données, mais pour l'instant la base de données reste vide. Il faut donc migrer schéma des entités vers la base et permettre à l'ORM de transposer nos entités et attribues en tables et colonnes. Nous avons lancé les commandes suivantes :

```
$ symfony console make:migration
```

```
$ symfony console doctrine:migrations:migrate
```

La première commande permet de créer le fichier de migration qui contient des requêtes au format sql. Ces requêtes permettent de définir les tables et les colonnes à créer ou modifier ainsi que les paramètres qui leur sont liés. En effet, lorsque nous avons créé nos entités, Symfony s'est chargé de créer sur chaque attribut « mappé » des annotations #[ORM/...] permettant de définir la manière dont Doctrine doit gérer ces attribues et les paramétrer en base de données. On a pu voir par exemple sur la figure 10, un paramètre « nullable : false » qui signifie que la colonne « idProvider » ne pourra être nulle lors de la création d'un produit.

Ces 2 commandes nous ont suivis tout au long de la création du projet. Toutes les fonctionnalités n'ayant pas été créées d'un seul bloc, il nous a fallu mettre à jour la base de données en fonction de nos besoins. Le fichier de migration présente l'avantage de garder une trace des modifications apportées à la B.D.D. et la possibilité d'annuler les modifications erronées car le fichier intègre les requêtes permettant un retour à l'état précédent la migration.

### 3. Le Repository

La base de données étant en place, il est maintenant possible de manipuler cette dernière. On va pouvoir ajouter, modifier, supprimer et accéder aux données par l'intermédiaire de Doctrine. On parle des fonctions de base du CRUD (acronyme pour Create, Read, Update, Delete).

Dans ce projet les entités présentes dans le catalogue ont été gérées dans le Back Office par l'intermédiaire du bundle EasyAdmin. Il nous facilite la gestion du CRUD de ces entités, ceci dans le but de gagner du temps de développement. Les exemples décrits ci-dessous seront extraits du CRUD des reviews provenant de l'interface utilisateur du site.

#### *a) L'insertion de données :*

L'insertion de nouvelles données s'effectue par l'intermédiaire de méthodes que l'on intègre dans le Contrôleur (nous détaillerons son fonctionnement dans la prochaine partie). Prenons l'exemple la création d'une review (ou critique) d'un produit par un utilisateur. Pour réaliser cette opération, dans la fonction `addReview`, on peut commencer par récupérer l'utilisateur connecté en session ainsi que le produit à noter. On initialise également un nouvel objet `Review` dans lequel on va pouvoir assigner des valeurs à ses attributs. On parle d'hydratation un objet. On insère ensuite les données (Produit et Utilisateur) à la review par l'intermédiaire des fonctions `Set` (les `Setters`) de l'entité `Review`. Il nous reste encore à récupérer les données soumises par l'utilisateur dans le formulaire par l'intermédiaire de la fonction `handleRequest` et d'hydrater notre objet avec ces données.

Notre objet est enfin créé mais il n'est pas encore envoyé en base de données. Pour ce faire nous avons fait appel au gestionnaire d'entités de doctrine, l'`EntityManagerInterface`, qui contient 2 fonctions qui nous intéressent ici. La fonction **`persist`** permet de signaler à Doctrine que l'objet doit être suivi et persister en b.d.d. et la fonction **`flush`** envoie ensuite les données à la base de données.

```

// On initialise un nouvel objet Review
$review = new Review();
//On insère Le user et Le produit dans La nouvelle instance de review.
$review->setUser($userSession);
$review->setProduct($product);

$formReview = $this->createForm(ReviewType::class, $review);
//On récupère Les données reçu par L'intermédiaire du formulaire
$formReview->handleRequest($request);

if ($formReview->isSubmitted() && $formReview->isValid()) {
    // On insère Les données du formulaire à notre Review
    $review = $formReview->getData();
    //On signale à Doctrine que L'objet doit être enregistré en b.d.d.
    $entityManager->persist($review);
    // On envoie Les données à persister en b.d.d.
    $entityManager->flush();

    return $this->redirectToRoute('detail_product', ['slug' => $product->getSlug()]);
}

```

Figure 11-Extrait de la fonction addReview

## b) La modification et suppression des données

La modification s'effectue sensiblement de la même manière que l'insertion et peut s'effectuer à l'intérieur de la même fonction. Lorsqu'on souhaite modifier une review on récupère cette dernière en b.d.d. Dans ce cas de figure, nous n'avons pas besoin de créer une nouvelle instance de Review mais simplement de récupérer le formulaire avec les informations à modifier. On insère également le champ updatedAt pour mettre à jour sa date de modification.

```

//Si La review n'existe pas en b.d.d.
if (!$review) {
    // On initialise un nouvel objet Review
    $review = new Review();
    //On insère Le user et Le produit dans La nouvelle instance de review.
    $review->setUser($userSession);
    $review->setProduct($product);
    //Sinon on modifie sa date de mise à jour
} else {
    $review->setUpdatedAt(new \DateTimeImmutable());
}

```

Figure 12-Modifications apportés à la fonction addReview pour permettre la modification d'une Review.

Pour supprimer une entité, on crée une nouvelle fonction, mais le principe reste également similaire. On remplacera juste la fonction persist(\$review) par remove(\$review), pour signaler à doctrine notre intention de supprimer une entité, avant de faire appel à nouveau à la fonction flush().



### c) La recherche d'entités

Doctrine, par l'intermédiaire du ServiceEntityRepository propose des requêtes prédéfinies qui correspondent aux usages les plus courants. On peut faire appel à ces requêtes dans nos contrôleurs par l'intermédiaire de l'entityManagerInterface ou en appelant directement le repository qui nous intéresse.

Voici quelques exemples d'utilisations de ces requêtes dans notre projet :

- **find(\$id)** : Renvoie un élément de l'entité (ou null) à l'aide de l'identifiant de ce dernier.

*Exemple :* On récupère les informations de l'utilisateur enregistré en session pour afficher ses informations personnelles dans la vue du profil utilisateur.

```
$userId = $this->getUser()->getId();  
$user = $entityManager->getRepository(User::class)->find($userId);
```

- **findAll()** : Renvoie toutes les entrées d'une entité.

*Exemple :* On récupère la liste complète des produits.

```
$products = $entityManager->getRepository(Product::class)->findAll();
```

- **findOneBy (array \$criterias)** ou **findOneByX()** : Renvoie un objet (ou null) en prenant pour paramètre une liste de critères. FindOneByX est une requête magique où X correspond à n'importe quel champ défini de notre entité.

*Exemple :* On récupère un produit spécifique à partir de son champs slug.

```
$product = $entityManager->getRepository(Product::class)->findOneBySlug($slug);
```

- **findBy(array \$criteria)** ou **findByX()** : Renvoie un tableau d'objets en prenant pour paramètre une liste de critères. findByX() est une méthode magique où X correspond à n'importe quel champs défini dans notre entité.

*Exemple :* On récupère la liste des commandes payées et traitées de l'utilisateur connecté pour les afficher sur sa page profil.

```
$ordersArchives = $orderRepository->findBy(
    ['user'=> $userId,
     'isPaid'=>true,
     'isProcessed'=> true],
    ['createdAt' => 'DESC']);
```

Si aucune des requêtes prédéfinies ne répondent à nos besoins, nous pouvons également créer nos propres fonctions dans le repository. Les requêtes à l'intérieur de ces fonctions seront écrites en DQL, le langage propre à Doctrine.

*Exemple :* On souhaite récupérer la moyenne des notes de chaque produit, pour afficher cette dernière dans la vue liste des produits.

```
//recherche les produits en fonction des critères renseignés dans le formulaire produits
public function findByCriteria(Filters $filters): PaginationInterface
{
    //On cherche à récupérer les produits et la moyenne de leur note si celle-ci existe
    $query = $this->createQueryBuilder('p')
        ->select('p', 'AVG(r.rating) AS averageRating')
        ->leftjoin('p.reviews', 'r')
        ->groupBy('p.id');

    //...
    // ici d'autres requêtes de filtres. Voir partie V - Présentation d'une fonctionnalité
    //...

    $query = $query->getQuery();

    return $this->paginator->paginate(
        $query,
        $filters->page,
        12
    );
}
```

Cette fonction sera présentée plus en détail dans la partie V – Présentation d'une fonctionnalité : La création des filtres de produits.

## B. Le Contrôleur

Le contrôleur représente l'intermédiaire entre le modèle et la vue. Il reçoit les requêtes envoyées depuis le navigateur à l'aide du protocole http, Il contrôle et sécurise les données reçues, communique avec le modèle au travers de l'ORM et donc l'entityManagerInterface. Il renvoie ensuite une réponse vers la vue. Un contrôleur est une classe PHP contenant des méthodes appelées actions ou méthodes d'actions.

Pour commencer à mettre en place les contrôleurs dans notre projet nous avons fait appel au bundle maker comme précédemment :

```
$ symfony console make:controller

Choose a name for your controller class (e.g. TinyElephantController):
> ProductController

created: src/Controller/ProductController.php
created: templates/product/index.html.twig

Success!
```

Cette commande a créé pour nous le fichier de base du contrôleur, à l'intérieur duquel on retrouve une première action, ainsi qu'un fichier contenant le Template qui est rattaché à cette action.

Chaque méthode d'action d'un contrôleur est associée à une route. Depuis la version 6 de Symfony on utilise les Attributs de route PHP pour les décrire. Anciennement, elles étaient écrites sous forme d'annotations.

Une route est constituée d'un nom (name) et d'un chemin (path). Elle permet de diriger une url vers une action de contrôleur. L'url sera renseignée dans le chemin de notre route. Dans notre projet nous retrouvons 2 types de routes, **constantes** ou **dynamiques**. Les routes dynamiques contiennent dans leur url des variables contenues entre accolades {}. On pourra récupérer ces variables et les utiliser dans notre méthode.

*Exemple de route constante : La vue profil utilisateur. L'url ne contient pas d'attribut car on récupère l'utilisateur par l'intermédiaire de la session.*

```
#[Route('/user', name: 'app_userProfile')]
```

*Exemple de route dynamique : On accède à la vue détail produit à partir du slug renseigné dans l'url. Cette variable entourée par des accolades pourra être récupérée pour être utilisée dans notre méthode afin de rechercher le produit correspondant.*

```
#[Route('/products/{slug}', name: 'detail_product')]
public function show(
    EntityManagerInterface $entityManager,
    string $slug = null,
    Request $request
): Response {

    $product = $entityManager->getRepository(Product::class)->findOneBySlug($slug);
```

Une action d'un Controller retourne toujours une Response. Cette réponse est un objet qui contient la propriété header de php, ce qui signifie qu'une action retournera une redirection via un protocole http. Il existe plusieurs types de réponses, voici celles que nous avons utilisé au cours du projet :

- **redirectToRoute(string \$route, array \$parameters = [], int \$status = 302)**: redirige vers une autre méthode d'un contrôleur et prend en argument le nom de la méthode ainsi que les paramètres liés au chemin de cette méthode.

*Exemple : Redirection de l'utilisateur vers la vue détail produit après l'ajout d'une review.*

```
return $this->redirectToRoute('detail_product', ['slug' => $product->getSlug()]);
```

- **render(string \$view, array \$parameters = [], Response \$response = null)** : Récupère une vue twig et affiche cette dernière à l'aide des variables (critères) envoyés.

*Exemple : On récupère le Template de la liste des produits, dans lequel on injecte les produits, le formulaire de filtres et les paramètres liés à la boutique (ici, l'image par défaut pour les produits dont l'image n'est pas renseignée)*

```
return $this->render('product/index.html.twig', [
    'products' => $products,
    'parameters' => $parameters,
    'formFilter' => $formFilter,
]);
```

- **RedirectResponse(string \$url, int \$status = 302, array \$headers = [])**: Redirige vers une autre url.

*Exemple : On génère l'url de la session Stripe pour rediriger l'utilisateur vers la page de paiement. \$checkout\_session contient toutes les informations nécessaires au paiement. Un tableau de la liste des articles du panier, l'email de l'utilisateur et le mode de paiement.*

```
return new RedirectResponse($checkout_session->:url);
```

- **generateUrl(string \$route, array \$parameters = [], int \$referenceType)** : Génère une URL à partir d'une route.

*Exemple : On génère l'url de redirection lorsque le paiement Stripe est validé.*

```
'success_url' => $this
    ->generateUrl('add_order', [], UrlGeneratorInterface::ABSOLUTE_URL)
    . '?session_id={CHECKOUT_SESSION_ID}',
```

## C. La Vue

La vue ou Template permet d'afficher des données envoyées depuis le contrôleur à l'utilisateur et sert d'interface graphique avec laquelle l'utilisateur peut interagir. L'ensemble des vues sont regroupées dans le dossier Template de notre projet Symfony. Une vue est constituée de HTML pour structurer les éléments à afficher et de CSS pour styliser la vue. Cependant, ces langages seuls ne permettent d'afficher que des pages uniques et statiques. Heureusement, le Framework Symfony intègre un moteur de Template permettant d'intégrer facilement du PHP dans nos vues et les rendre dynamiques : TWIG.

TWIG est un langage à part entière permettant d'intégrer des boucles, conditions, filtres et fonctions dans nos vues. Il permet également d'afficher du contenu et est composé d'une syntaxe simple que l'on peut résumer en 3 points :

- L'affichage de contenu se fait à l'intérieur des doubles accolades : `{{ }}`
- Pour intégrer de la logique à nos vues (boucles, conditions) on l'encadre dans des accolades simples additionnées au symbole pourcentage : `{% %}`
- Pour afficher des commentaires, on utilise des accolades simples et le symbole dièse : `{# #}`



Figure 13- Exemple d'article, vue liste des produits

Prenons par exemple la vue « liste des produits ». Chaque produit est contenu dans une balise article, à l'intérieur de laquelle on va retrouver toutes les informations de base du produit : Une image de présentation, un titre contenant le nom de la brasserie, la désignation du produit et sa contenance, son prix, une note moyenne et un nombre de notes. (figure 13)

Cette structure étant identique à chaque produit on peut générer la liste complète des articles en faisant appel à une **boucle « for »** sur la liste des produits. Pour rappel, ces informations ont été envoyées par le contrôleur vers la vue à l'aide de la fonction *render* (*Voir l'extrait de code de la fonction render, partie IV.3. Le contrôleur*). On intègre à l'intérieur de la boucle les différents éléments du produit à afficher.

```

{% for product in products %} Boucle sur la liste des produits

    {% if product[0].available %}
        <article class="article product"> Génération de l'url à l'aide de la fonction path
            <a href="{{ path('detail_product', {'slug' : product[0].slug}) }}">
                <div class="img_product_article">
                    {% if product[0].productionType is defined and product[0].productionType.productionType != 'Permanente' %}
                        <div class="production type"> Affichage du contenu : "permanente", "saisonnière", ...
                            <span> {{ product[0].productionType.productionType }} </span>
                        </div>
                    {% endif %}

                    <div class="more info">plus d'infos</div> Condition d'affichage de l'image produit
                    {% if product[0].imageFile is defined and product[0].imageFile is not null %}
                        
                    {% else %}
                        
                    {% endif %}
                </div>
            </a>
            <div class="product_info">

```

Figure 14- Extrait du Template TWIG - liste des produits.

### Filtrer par

Rechercher


**Brasseries**

- ☐ Brasserie test
- ☐ Lorem Ipsum Brewer
- ☒ Roue libre


**Styles de bières**

- ☐ Ambrée
- ☐ Belgians
- ☐ Black IPA
- ☐ Blanche
- ☐ Blonde
- ☐ Gose
- ☐ I.P.A.
- ☐ Impérial
- ☐ Kölsch
- ☐ Old School IPA
- ☐ Pale Ale
- ☐ Pils
- ☐ Red Ale
- ☐ Scotch Ale
- ☐ Smoked Oatmeal
- ☐ Stout
- ☐ Triple
- ☐ Wee Heavy
- ☐ Weissbier


Trier par ▼




Roue libre - Auguste Friville - 33cl  
2,90 € T.T.C.  
★★★★★ (1)




Roue libre - The HopRider - 33cl  
3,30 € T.T.C.




Roue libre - Calico - 33cl  
3,80 € T.T.C.



Roue libre - Cabane à Suc' - 33cl  
4,50 € T.T.C.  
★★★★☆ (1)



Roue libre - Rétropédalage - 33cl  
3,10 € T.T.C.  
★★★★★ (1)



Roue libre - Le Pot Belge - 33cl  
4,10 € T.T.C.

Figure 15-Extrait de la page liste des produits. Interface utilisateur.

On peut ajouter des conditions d’affichage, à l’aide des **conditions « if »**, comme indiqué en figure 14. Par exemple, on peut choisir d’afficher l’image du produit si celle-ci est bien définie et non null, sinon on affichera l’image produit par défaut contenue dans les paramètres du site.

TWIG intègre également des fonctions spécifiques à Symfony. Par exemple la **fonction path** permet de générer les chemins relatifs des urls à partir du nom de l’action du contrôleur ciblé et des paramètres contenus dans la route si celle-ci est dynamique.

```
<a href="{{ path('products_index') }}" class="button_base button_links">La boutique</a>
```

Il existe également de **nombreux filtres** qui nous ont permis de formater nos données, on peut citer quelques exemples comme le formatage des dates, ou l’affichage de la longueur d’un tableau. Dans l’exemple ci-dessous on affiche la date de création d’un avis et le nombre total d’avis pour un produit.

```
{{ review.createdAt|format_time('short', locale='fr', timezone='Europe/Paris') }}
```

```
<p>{{ product.reviews|length }} avis</p>
```

On peut également créer nos propres filtres Twig. Dans notre projet, l’affichage des notes pour chaque review, ainsi que la moyenne des notes sont gérés à l’aide d’un **filtre Twig personnalisé**.

Pour créer ce filtre nous avons créé une nouvelle class FilterExtension qui étend la classe AbstractExtension de Twig. Cette classe est composée de 2 fonctions :

- La fonction **getfilters** définit les filtres à créer. Elle retourne un tableau d’objets TwigFilter qui sont définis par un nom ainsi que la méthode utilisée lors de l’appel du filtre. Dans notre cas on rajoute un 3eme paramètre pour indiquer à Twig que notre fonction retournera du code html sain. En effet, par sécurité Twig échappe les balises html pour se prémunir des failles XSS (vulnérabilité par injection de code, nous traiterons plus en détail cette faille dans la partie Sécurité).

```
namespace App\Twig;

use Twig\Extension\AbstractExtension;
use Twig\TwigFilter;

You, 3 weeks ago | 1 author (You)
class FiltersExtension extends AbstractExtension
{
    public function getFilters(): array
    {
        return [
            new TwigFilter('stars', [$this, 'stars'], ['is_safe' => ['html']]),
        ];
    }
}
```

- La fonction **stars** contient la logique de notre filtre et retourne une chaîne de caractères composés de balises HTML contenant des icônes « star » Font Awesome. Le principe de la fonction est d'afficher 5 étoiles représentant une note contenue entre 1 et 5. La partie décimale de la note est arrondie au demi-point inférieur.

```
public function stars(?float $note): string
{
    $html = '';

    if (!$note) {
        return $html;
    } else {
        $entier = floor($note);
        $decimal = $note - $entier;
        $count = 0;

        for ($i = 0; $i < $entier; $i++) {
            $html .= '<i class="fa-solid fa-star"></i>';
            $count++;
        }
        if ($decimal > 0 && $decimal < 0.5) {
            $html .= '<i class="fa-regular fa-star"></i>';
            $count++;
        } elseif ($decimal >= 0.5) {
            $html .= '<i class="fa-solid fa-star-half-stroke"></i>';
            $count++;
        }
        for ($i = 0; $i < 5 - $count; $i++) {
            $html .= '<i class="fa-regular fa-star"></i>';
        }

        return $html;
    }
}
```

Pour finir, il nous suffit d'intégrer notre filtre dans la vue comme n'importe quel autre filtre, en l'appelant avec le nom renseigné dans la fonction `getFilters`.

```
<p>{{ review.rating | stars }}</p>
```



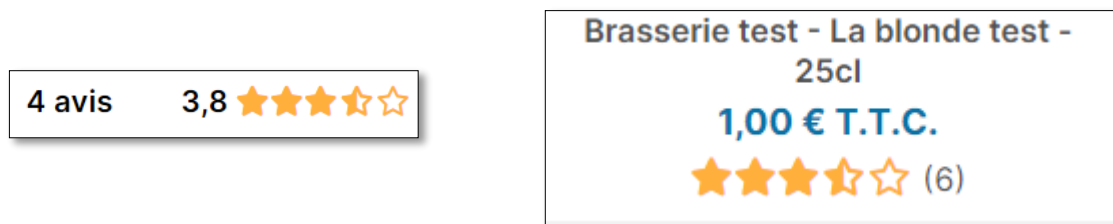


Figure 16- Exemples d'intégration du filtre stars dans différentes vues

## D. Le responsive Design

Le Responsive design est un élément important à prendre en compte lors de la création d'un site Web. En effet, les appareils mobiles représentent 57% de la part de marché contre 43% pour les ordinateurs (chiffres 2020) et le trafic de données sur les réseaux mobiles ne cesse d'augmenter. Il faut donc prendre en compte la diversité des écrans et créer notre site pour répondre aux besoins des utilisateurs.

Pour respecter la définition du Responsive Design il faut suivre certaines règles : « Le même code HTML doit être affiché sur la même URL quel que soit l'appareil utilisé. Toutefois, le contenu peut être affiché différemment selon la taille de l'écran » (source : *Google Search Central*). C'est également un élément très important pour l'indexation de notre site sur les moteurs de recherche, Google utilise principalement la version mobile des sites pour réaliser son référencement, on parle alors de d'indexation orientée mobile.

Pour réaliser le Responsive design de notre projet nous avons fait appel aux Media Queries (ou Requêtes média) du langage CSS. Ces requêtes permettent de cibler des éléments de nos pages en leur appliquant certains styles de façon conditionnelle grâce à des règles (ici la largeur de l'écran).

Prenons par exemple la vue « détail d'un produit ». Nous avons appliqué des règles spécifiques en fonction de la taille de l'écran en définissant des break-points à l'intérieur desquels nous avons spécifié les nouvelles règles à appliquer à nos éléments. Dans le premier exemple de code, on modifie la présentation du produit lorsque la résolution de l'écran passe sous la barre des 735px. On affiche alors le titre et la description du produit sous la photo et non plus à côté. On ajoute également une marge pour plus de clarté. Dans l'exemple 2, on modifie la disposition du formulaire d'ajout au panier en suivant le même principe lorsque la largeur de l'écran est inférieure à 440px.

```
@media screen and (max-width: 735px) {
  .product_presentation {
    flex-direction: column;
    align-items: center;
  }

  .product_informations {
    margin-top: 40px;
    text-align: center;
  }
}
```

Figure 18-Exemple 1 de Media Query - Vue détail produit.

```
@media screen and (max-width: 440px) {
  .form_detail_product {
    flex-direction: column;
  }

  .form_select_qte {
    padding-bottom: 30px;
  }

  .reviews_summary a {
    font-size: 1rem;
  }

  .review_description p {
    font-size: 1rem;
  }
}
```

Figure 17- Exemple 2 de Media Query - Vue détail produit.

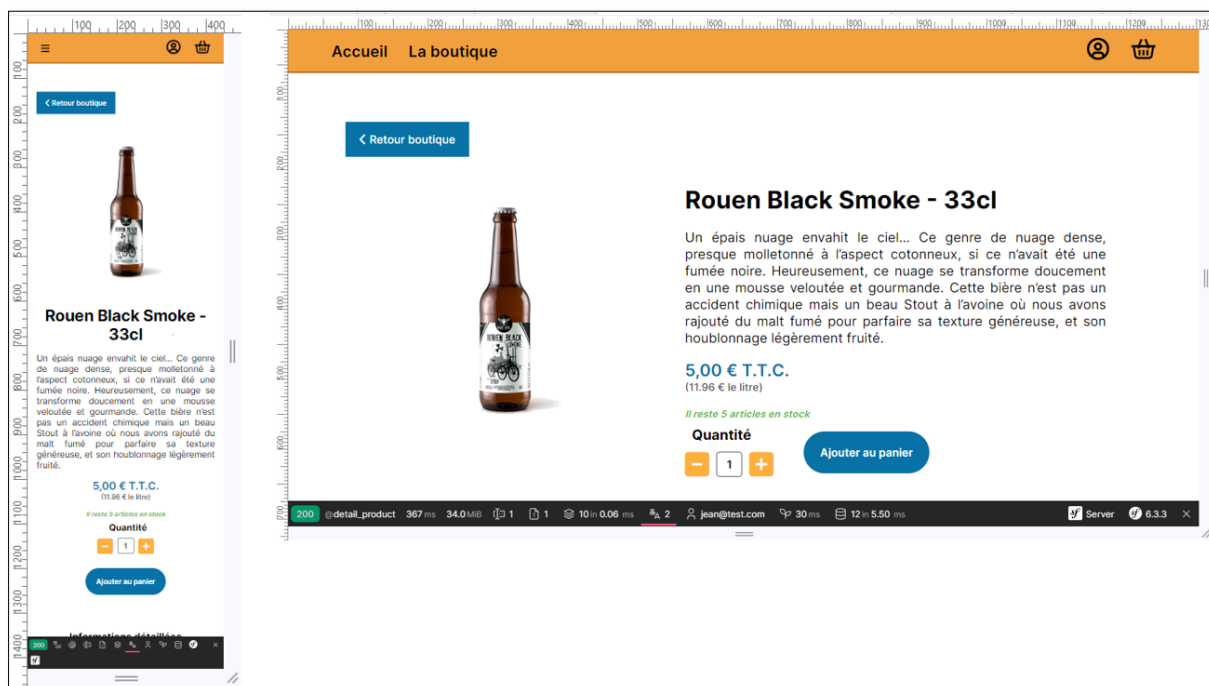


Figure 19-Aperçu du Responsive Design - Vue détail produit. A gauche largeur d'écran 440px, à droite 1300px.

## E. Le référencement

Le référencement naturel, également appelé **SEO** (Search Engine Optimisation), est l'ensemble des techniques mises en place pour améliorer la position du site dans les moteurs de recherche. Cela permet d'augmenter la visibilité du site, il faut donc prêter attention à différents détails qui additionnés ont un impact notable sur la visibilité du site.

Dans notre projet plusieurs techniques de référencement ont été mise en place :

### La Balise <title>

Balise présente dans la partie Head d'un document html, cette balise est l'élément principal scanné par les robots. C'est le titre affiché dans les résultats des moteurs de recherche et le titre visible dans les onglets. Dans notre projet ce titre sera intégré dans un block twig title, comme le montre l'exemple suivant :

```
{% block title %}Association l'echoppe - Bar associatif et vente de bières locales{% endblock %}
```

### Balisage sémantique (H1-H6)

Les niveaux de titres ont leur importance dans le référencement des sites. La balise H1 étant la plus importante. Une seule balise h1 est présente par page et contient le mot-clé principal de la page affichée.

### La Balise <strong>

C'est également une balise sémantique permettant de délimiter du texte (mots-clés et expressions) que l'on souhaite mettre en valeur pour augmenter son importance. Ce texte doit avoir un rapport direct avec le sujet abordé dans la page web et resté dans le champ sémantique du mot-clé principal. A utilisé avec parcimonie.

### Les balises <meta>

Les balises meta sont des balises permettant de fournir des informations supplémentaires sur une page au moteur de recherche.

On peut utiliser par exemple la balise **meta description** pour fournir au moteur de recherche une description de la page affichée dans les résultats de recherche. Cela n'a pas une incidence directe sur le référencement mais une description pertinente permettra d'augmenter le nombre de cliques.

La balise **meta robots** permet d'indiquer aux moteurs de recherche si on souhaite ou non indexer une page web. On utilise essentiellement les valeurs "noindex" et "nofollow" pour indiquer aux moteurs de recherche de ne pas indexer certaines pages dont le contenu n'est pas important, car les moteurs de recherche n'apprécient pas les contenus de faible valeur.

### Les URLs

Les moteurs de recherche prennent en compte la structure des URLs, ils ne doivent pas contenir d'ID trop longs et les termes utilisés doivent être compréhensibles. C'est pourquoi chaque produit contient un attribut slug qui est défini par le nom du produit et sa contenance, chaque mot étant séparé par des traits d'union (-). L'accentuation des caractères ainsi que les caractères spéciaux sont supprimés. (Ex : "Cabane à Suc' 33 cl" donnera "/cabane-a-suc-33")

### Le responsive design

Les moteurs accordent une grande importance au responsive design et à l'indexation orientée mobile.

### Les images

Pour optimiser le référencement des images il est important de respecter plusieurs règles :

- Le **nom du fichier** doit être clair. Dans notre cas le nom est constitué de la désignation du produit et de sa contenance.
- L'**attribut alt** d'une image permet de décrire le contenu de cette dernière. Il s'affichera également si un problème survient lors du chargement de l'image et que celle-ci n'apparaît pas. C'est également ce texte qui sera lu par les lecteurs d'écran.
- Le **poids et la taille** des images : Elles ont été compressées avant d'être ajoutées au site.

### Performances et bonnes pratiques

Afin de pouvoir réaliser un audit SEO, Google a mis à disposition des développeurs un outil permettant d'évaluer son site : lighthouse.

4 points importants sont évalués :

- **La performance** du site, se traduit par la vitesse de chargement d'une page. La taille et le poids des images influent beaucoup sur ce résultat.
- **L'accessibilité** correspond à de bonnes pratiques pour permettre au site d'être accessible aux malvoyants. Cela se traduit par des tailles de police suffisamment importantes, des balises alt sur nos images permettant au lecteur d'écran de décrire correctement l'image, des contrastes importants, etc...
- **Les bonnes pratiques** couvrent un ensemble de paramètres concernant notamment la sécurité du site (protection contre les attaques XSS), la structure de la page, l'affichage correct des images, etc...
- **Le SEO** vérifie la présence des balises précédemment citées (title, meta, etc...) ainsi que l'absence de liens non cliquables.

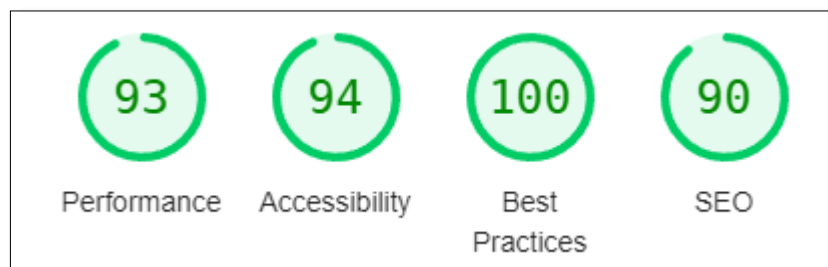


Figure 20-Résultat de l'audit obtenu sur la page principale du site (Google lighthouse)

## VI. Présentation des fonctionnalités

---

### A. La création des filtres de produits

Afin d'améliorer l'expérience utilisateur, nous avons mis en place un système de filtres sur la vue « liste des produits ». L'utilisateur peut combiner ces filtres comme il le souhaite pour obtenir une sélection d'articles correspondant au mieux à ses attentes. Nous avons également mis en place des fonctionnalités de tri et une pagination par l'intermédiaire du bundle Symfony knpPaginator.

#### 1. La construction du modèle

Construit sur une structure semblable à nos entités, le modèle de filtre est une classe composée d'attributs correspondant aux paramètres de nos recherches. A la différence des entités, le modèle de filtre n'est pas relié à l'ORM Doctrine et par conséquent il n'est pas relié à une table. On parlera plutôt de ce modèle comme d'un service, c'est-à-dire une classe assurant un rôle précis. Dans notre cas, contenir les données de nos recherches.

Cette méthode nous permet d'attribuer à chaque variable de filtre un type et de leur fixer des contraintes. Cela nous sera très utile lors de la validation du formulaire.

```
<?php

namespace App\Model;

    You, last month * Mise en place du formulaire filtres
use Symfony\Component\Validator\Constraints as Assert;

    You, 14 hours ago | 1 author (You)
class Filters
{
    #[Assert\Length(
        max: 30,
        maxMessage: 'Votre recherche ne peut pas contenir plus de {{ limit }} caractères',
    )]
    public ?string $searchProduct = '';

    public ?array $providers = [];

    public ?array $beerTypes = [];

    #[Assert\PositiveOrZero(
        message: 'Veuillez selectionner un prix minimum supérieur à 0€'
    )]
    public ?float $min;

    #[Assert\PositiveOrZero(
        message: 'Veuillez selectionner un prix max supérieur à 0€'
    )]
    public ?float $max;
```

Figure 21 - Extrait de la classe Filters contenant les paramètres de la recherche

## 2. La construction du formulaire

On construit notre formulaire en suivant la même logique que pour les autres formulaires Symfony. La création des formulaires est réalisée dans une classe en dehors du contrôleur afin de minimiser la logique présente dans ce dernier. Nous avons donc créé une classe `FormType` nommée `FiltersType`. Cette classe est constituée de 2 fonctions :

- **configureOptions** : cette méthode nous permet de configurer les paramètres de base de notre formulaire. On définit la classe à laquelle le formulaire est rattaché, on y ajoute la méthode d'envoi « GET » et nous pouvons également retirer la protection CSRF proposée par défaut car ce formulaire ne présente pas de risque d'attaque CSRF.

```
public function configureOptions(OptionsResolver $resolver): void
{
    $resolver->setDefaults([
        'data_class' => Filters::class,
        'method' => 'GET',
        'csrf_protection' => false,
    ]);
}
```

- **buildForm** : cette méthode permet de construire le formulaire à l'aide du `FormBuilderInterface`. Elle contient les différents champs du formulaire. Pour chaque champ, on définit le type de formulaire à mettre en place ainsi que les options rattachées à ces différents champs. On peut noter, par exemple, la présence de l'option `'required' => 'false'` dans l'intégralité des champs, car aucun n'est requis lors de l'envoi du formulaire.

```
public function buildForm(FormBuilderInterface $builder, array $options): void
{
    $builder
        ->add('searchProduct', SearchType::class, [
            'required' => false,
            'label' => false,
            'attr' => [
                'placeholder' => 'Rechercher',
                'class' => 'searchProducts',
                'maxlength' => '30',
                'error_bubbling' => true,
            ],
            'sanitize_html' => true,
        ])
        // You, last month • Mise en place du formulaire filtres
        ->add('providers', EntityType::class, [
            'class' => Provider::class,
            'label' => false,
            'choice_label' => 'name',
            'expanded' => true,
            'multiple' => true,
            'required' => false,
            'attr' => [
                'class' => 'filters_checkbox'
            ],
            'error_bubbling' => true,
        ])
}
```

### 3. Le rendu du formulaire et l'affichage des produits non filtrés

#### a) *Intégration du formulaire dans la vue*

- On intègre notre formulaire dans l'action index du productController, action dont le rôle est de gérer la vue Liste des produits.
- On crée une nouvelle instance de l'objet filter, auquel on attribue par défaut la valeur 1, puis on construit le formulaire à partir de cette instance.

```
#[Route('/products', name: 'products_index')]
public function index(
    EntityManagerInterface $entityManager,
    Request $request,
): Response {

    $filters = new Filters();
    $filters->page = $request->query->get('page', 1);
    $formFilter = $this ->createForm(FiltersType::class, $filters);
```

- On ajoute ensuite ce formulaire à la méthode render pour l'intégrer à notre vue. La fonction render va appeler automatiquement la méthode createView pour créer une vue de notre formulaire par l'intermédiaire des services du FormInterface.

```
return $this->render('product/index.html.twig', [
    'products' => $products,
    'parameters' => $parameters,
    'formFilter' => $formFilter,
]);
```

- Dans notre fichier contenant le Template de l'index des produits, on peut alors intégrer notre formulaire à l'aide des fonctions form proposées par Twig.

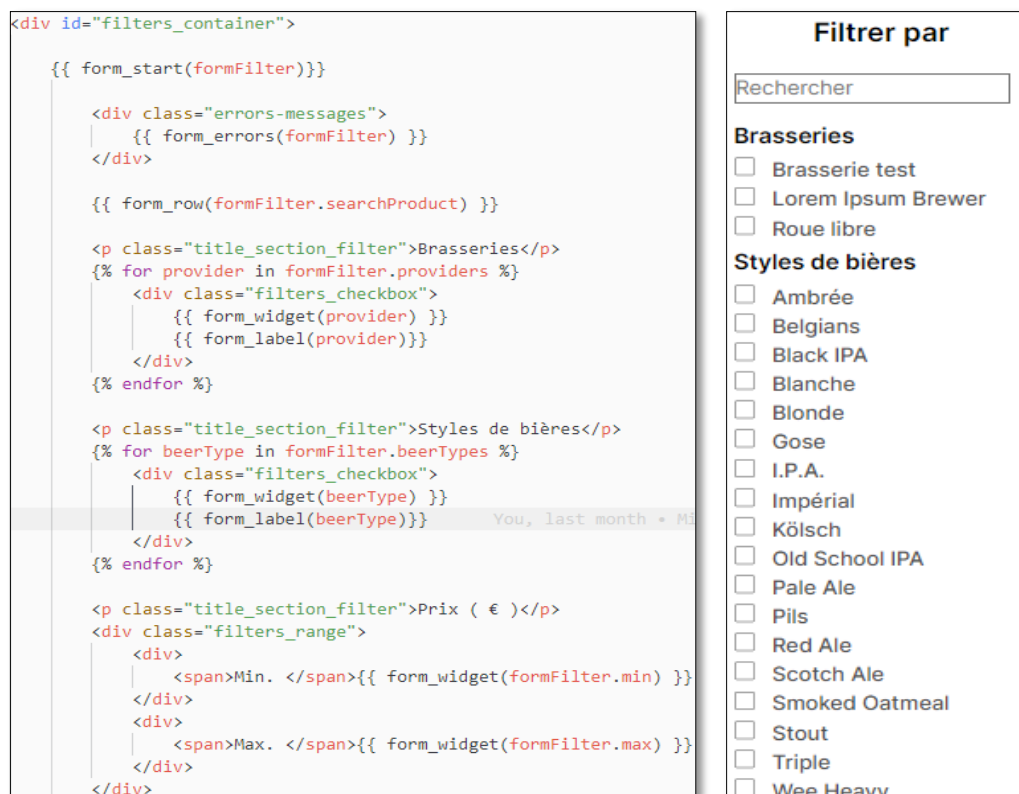


Figure 22-Extrait du code d'intégration du formulaire dans la vue « liste des produits » et son rendu final.

## b) Recherche de la liste complète des produits et intégration de la pagination.

Pour récupérer la liste complète des produits et intégrer la pagination à notre vue, nous avons créé dans le repository une requête personnalisée. Il est possible d'intégrer la pagination directement depuis le contrôleur, mais il est préférable de limiter la logique dans ce dernier et l'ajouter au repository.

Depuis le contrôleur nous appelons la fonction personnalisée « findWithoutCriteria » du repository :

```
$products = $entityManager->getRepository(Product::class)->findWithoutCriteria($filters->page);
```

Contrairement à la fonction findAll présente dans le Repository, notre fonction ne retournera pas un simple tableau de produits, mais un objet de type PaginationInterface. Cet objet contiendra le résultat de notre requête ainsi que le numéro de page à afficher et le nombre d'articles à afficher par page.

Notons que dans cette requête, nous avons sélectionné les produits, la moyenne de leurs notes et le nombre de note par produit. Ce regroupement nous a permis de limiter le nombre de requêtes envoyées vers la base de données lors de la création de la vue. L'alias « averageRating » nous sera également nécessaire lors de l'implémentation de la fonction tri par notes. Nous y reviendrons dans la prochaine partie.



```

public function findWithoutCriteria($page):PaginationInterface
{
    $query = $this->createQueryBuilder('p')
        ->select('p', 'AVG(r.rating) AS averageRating', 'COUNT(r) AS reviewCount' )
        ->leftjoin('p.reviews', 'r')
        ->groupBy('p.id')
        ->orderBy('p.designation', 'ASC')
        ->getQuery();

    return $this->paginator->paginate(
        $query,
        $page,
        12
    );
}

```

Figure 23 - Fonction de recherche de la liste complète des produits en intégrant la pagination

L'intégration de la pagination à notre vue se fait très simplement, à l'aide de la fonction `KnppaginatorRender` installée avec le bundle et qui prend en argument la pagination. Nous avons ensuite amélioré le rendu de la pagination avec du CSS.

```

<div class="paginator">
    {{ knp_pagination_render(products) }}
</div>

```

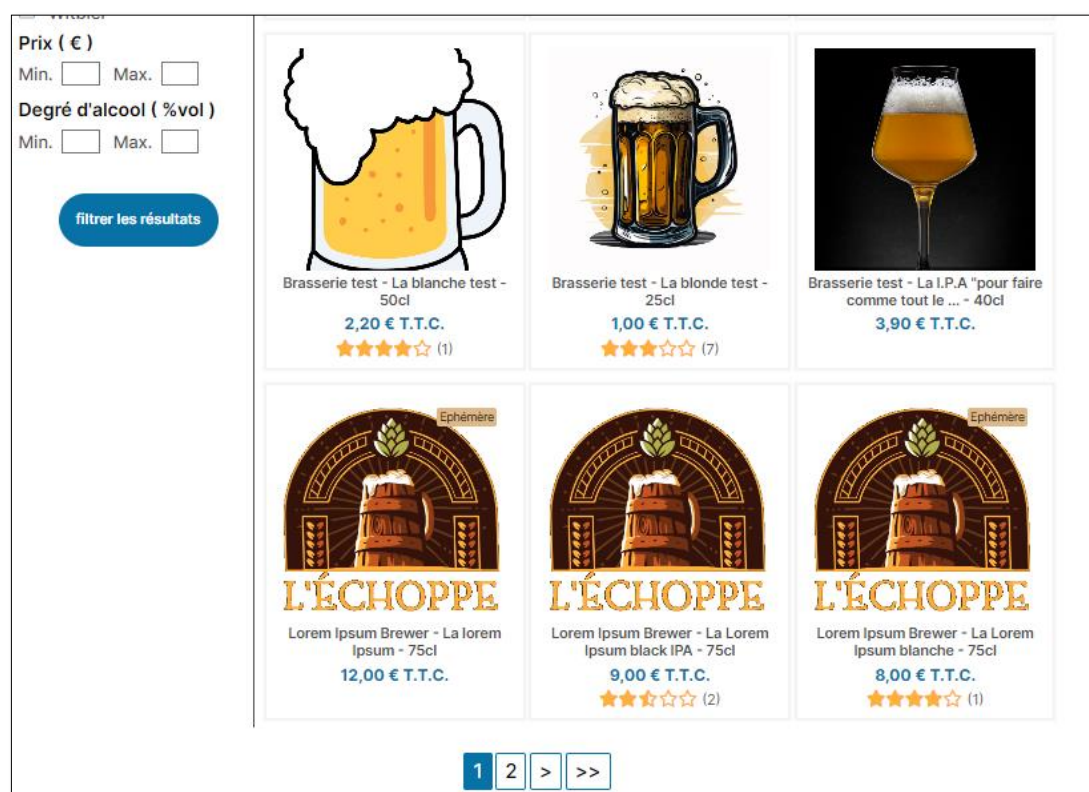


Figure 24 - Extrait de la vue "liste des produits" avec l'intégration de la pagination

## 4. Le Traitement du formulaire et l'envoi des données filtrées

### a) *Traitement du formulaire*

Maintenant que le formulaire est en place dans notre vue, l'utilisateur peut sélectionner et additionner les filtres comme il le souhaite, mais aucun changement ne sera visible à l'écran. Il faut pour cela récupérer les données dans notre contrôleur et interroger la base de données avant de renvoyer le nouveau jeu de données vers la vue.

On retourne dans l'action index du contrôleur et on récupère les données envoyées par la requête du formulaire à l'aide de la méthode `handleRequest()`.

Avant de lancer la recherche par critères on vérifie que le formulaire a bien été soumis et que ces informations sont valides. La fonction `isValid()` se base sur les contraintes présentes dans le modèle ainsi que les `formTypes` renseignées dans le `formbuilder` pour valider le formulaire et éviter l'envoi de données aberrantes.

Nous pouvons ensuite récupérer la liste des produits filtrés en appelant la fonction « `findByCriteria` » du repository, qui prend en argument un objet filtre avant de renvoyer le nouveau résultat à la vue.

```
$formFilter->handleRequest($request);

if ($formFilter->isSubmitted() && $formFilter->isValid()) {
    $products = $entityManager
        ->getRepository(Product::class)
        ->findByCriteria($filters);
}
```

### b) *Recherche des données dans la b.d.d.*

La requête présente dans notre fonction « `findByCriteria` » se décompose comme suit :

- On crée tout d'abord une nouvelle instance de `QueryBuilder`, nous permettant de créer notre requête sur l'entité produit.
- On sélectionne les produits, leur note moyenne et le nombre de notes de chaque produit.
- On joint à la requête l'entité `Provider` car présente dans plusieurs recherches.
- On regroupe le résultat global par produit
- On renvoie un résultat trié par ordre alphabétique de désignation de produit.

```
//Cherche Les produits en fonction des critères renseignés dans Le formulaire produits
public function findByCriteria(Filters $filters): PaginationInterface
{
    //On cherche à récupérer Les produits et La moyenne de Leur note
    //si celle-ci existe ainsi que Le nombre de note par produit
    $query = $this->createQueryBuilder('p')
        ->select('p', 'AVG(r.rating) AS averageRating', 'COUNT(r) AS reviewCount')
        ->join('p.provider', 'pr')
        ->leftjoin('p.reviews', 'r')
        ->groupBy('p.id')
        ->orderBy('p.designation', 'ASC');
```

- On vérifie pour chaque attribut de filtre si ce dernier est défini dans l'instance du filtre. Si celui-ci existe, on le rajoute à la requête.  
Par exemple, si une chaîne de caractère a été renseignée dans la barre de recherche, on vérifie si la désignation d'un produit ou le nom d'une brasserie contiennent cette chaîne de caractères et on récupère les produits correspondants.  
On parle de recherche par modèle. Les caractères % sont des caractères « joker » et remplacent tout autre caractère dans notre recherche.

```
if (!empty($filters->searchProduct)) {
    $query = $query
        ->andWhere('p.designation LIKE :searchProduct')
        ->orWhere('pr.name LIKE :searchProduct')
        ->setParameter('searchProduct', "%{$filters->searchProduct}%");
}
```

- On notera également que, lorsqu'un attribut contient plusieurs arguments, Doctrine se chargera d'ajouter chacun de ces arguments à la requête. Nous n'avons pas besoin de créer une boucle sur la liste des brasseurs, ni sur la liste des styles de bière.

```
if (!empty($filters->providers)) {
    $query = $query
        ->andWhere('pr.id IN (:provider)')
        ->setParameter('provider', $filters->providers);
}

if (!empty($filters->beerTypes)) {
    $query = $query
        ->join('p.beerTypes', 'bt')
        ->andWhere('bt.id IN (:beerTypes)')
        ->setParameter('beerTypes', $filters->beerTypes);
}
```

Comme pour la fonction `findWithoutCriteria`, notre fonction retourne un objet de type `PaginatorInterface` contenant la requête, le numéro de page et le nombre d'articles par page.

### c) Ajout de la fonction tri



Avec le bundle Paginator, l'ajout de la fonction tri se fait directement dans la vue détail produit. On utilise la fonction `knp_pagination_sortable` qui prend en argument la pagination, le label à afficher, le champ utilisé pour le tri et la direction du tri.

Le champ « note moyenne » n'étant pas un attribut de produit, nous avons dû créer un alias « `averageRating` » dans notre requête pour trier notre liste.

```
<div class="dropdown_sortBy_content" id="dropdown_blop" style="display:none">
  {{ knp_pagination_sortable(products, 'Prix croissants', 'p.price', {}, {'direction': 'asc'}) }}
  {{ knp_pagination_sortable(products, 'Prix décroissants', 'p.price', {}, {'direction': 'desc'}) }}

  {{ knp_pagination_sortable(products, 'Notes croissantes', 'averageRating', {}, {'direction': 'asc'}) }}
  {{ knp_pagination_sortable(products, 'Notes décroissantes', 'averageRating', {}, {'direction': 'desc'}) }}
</div>
```

## B. Le panier

### 1. L'ajout au panier

Nous avons fait le choix de créer le panier de l'utilisateur en nous servant de la Superglobal `$_SESSION`.

Les variables superglobales sont des variables fournies par l'environnement PHP. Elles sont accessibles dans l'intégralité de notre application. Il en existe 9, dont `$_GET` et `$_POST` que nous avons utilisé pour l'envoi des différents formulaires.

Celle qui va nous intéresser ici est la variable `$_SESSION`. Elle nous permet de stocker des données différentes pour chaque utilisateur en utilisant un identifiant de session unique. A la différence des cookies la variable session n'est pas gardée en mémoire sur l'ordinateur de l'utilisateur, elle est supprimée à la fermeture du navigateur.

Toutes les actions concernant le panier ont été regroupées dans un contrôleur dédié, nommé `CartController`. Lorsqu'un utilisateur ajoute un nouveau produit dans son panier depuis le formulaire d'ajout, les données sont transmises au contrôleur dans l'action « `add_cart` ».

Cette action ajoute un nouvel élément dans un tableau nommé panier que l'on insère dans la variable session. Ce tableau associatif contiendra uniquement l'identifiant de chaque produit ainsi que sa quantité correspondante.

Il reste 7 articles en stock

**Quantité**

- 5 +

Ajouter au panier

```
// On récupère La session
$session = $request->getSession();

// On récupère Le panier. si on entre dans la boutique,
// Le panier n'existe pas, on en récupère un en créant un tableau vide
$panier = $session->get('panier', []);

if ($form->isSubmitted() && $form->isValid()) {
    // On récupère La quantité soumise dans le formulaire
    $quantity = $form->getData()['quantity'];
    // On ajoute La quantité du produit au panier
    if (!empty($panier[$id])) {
        $panier[$id] = $panier[$id] + $quantity;
    } else {
        $panier[$id] = $quantity;
    }
    $message = '<span style="font-weight:600";>' . $quantity .
        '</span> ' . $product->getDesignation() .
        ' - ' . $product->getVolume() . 'cl <br> ajouté au panier';
    $this->addFlash('add_cart', $message);
}

// On sauvegarde Le panier en session pour continuer nos achats en boutique
$session->set('panier', $panier);

return new RedirectResponse($this->generateUrl('detail_product', ['slug' => $slug]));
```

## 2. Affichage du panier

```
//On récupère Le panier avec Les produits.
foreach ($panier as $id => $quantity) {
    $product = $entityManager->getRepository(Product::class)->find($id);
    $ssTotal = $product->getPrice() * $quantity;
    $stockTemp = $product->getStock() - $quantity;
    $elements[] = [
        'product' => $product,
        'quantity' => $quantity,
        'stockTemp' => $stockTemp,
        'ssTotal' => $ssTotal
    ];

    $total += $ssTotal;
}

/***** COTISATION *****/
$cotisation = $membershipContribution->checkContribution();
$total += $cotisation['price'];
```

Dans une nouvelle action du contrôleur CartController nommée « cart\_index », on récupère les éléments du panier simplement en créant une boucle sur le tableau panier.

Après avoir récupéré les éléments du panier, on vérifie également la date de validité de la cotisation de l'utilisateur en faisant appel à la fonction checkContribution. Elle provient de la classe service MembershipContributionService que nous avons créé dans le but de

ne pas surcharger le contrôleur. Cette technique présente également l'avantage de pouvoir accéder aux fonctionnalités du service à plusieurs endroits dans notre application sans réécrire la logique de la fonction. Cela rend notre code plus facilement maintenable. Nous en aurons notamment besoin lors du passage au paiement avec la session Stripe.














Mon panier			
Désignation du produit	Quantité	Prix	Sous-total
Calico 33cl	 5 	3,80 €	19,00 € 
La Lorem Ipsum blanche 75cl	 2 	8,00 €	16,00 € 
Rupture de stock : le délais de préparation de votre commande peut-être ralongé de quelques jours			
Bière brune test 33cl	 3 	3,30 €	9,90 € 
Y'a plus d'saison 25cl	 2 	3,70 €	7,40 € 
Cotisation annuelle valable jusqu'au 25/10/2024		10,00 €	10,00 €
TOTAL			62,30 €
<div>  Supprimer le panier </div> <div>Valider mon panier</div>			

Figure 25-Aperçu du panier

## C. Le paiement par Stripe

La plateforme Stripe propose 3 systèmes de paiement :

- Le système de lien de paiement. Utile lorsqu'on ne dispose pas de site, pour vendre un produit unique ou un abonnement. Stripe crée un lien de redirection vers le paiement du produit.
- Le système de Checkout. Permet de vendre plusieurs produits. L'utilisateur est redirigé vers un flot de paiement géré par Stripe.
- La solution Element. La plus complexe à mettre en place, elle permet d'intégrer directement le flux de paiement sur notre site, l'utilisateur ne sera donc pas redirigé.

Dans ce projet nous avons fait le choix d'utiliser la méthode de Checkout, car c'est la plus adaptée à notre situation. Elle présente l'avantage d'être moins complexe à mettre en place que la solution Element, tout en étant suffisamment paramétrable. L'utilisateur sera redirigé vers un flux de paiement sécurisé directement géré par Stripe.

La solution Checkout s'effectue en 3 étapes :

- La session Checkout créée à partir du panier est envoyée à Stripe pour que la plateforme puisse créer un lien de paiement.
- Redirection du client vers l'url créé par la session Stripe. Le client effectue le paiement, puis il est redirigé vers le site.
- Validation du paiement grâce aux Webhooks.

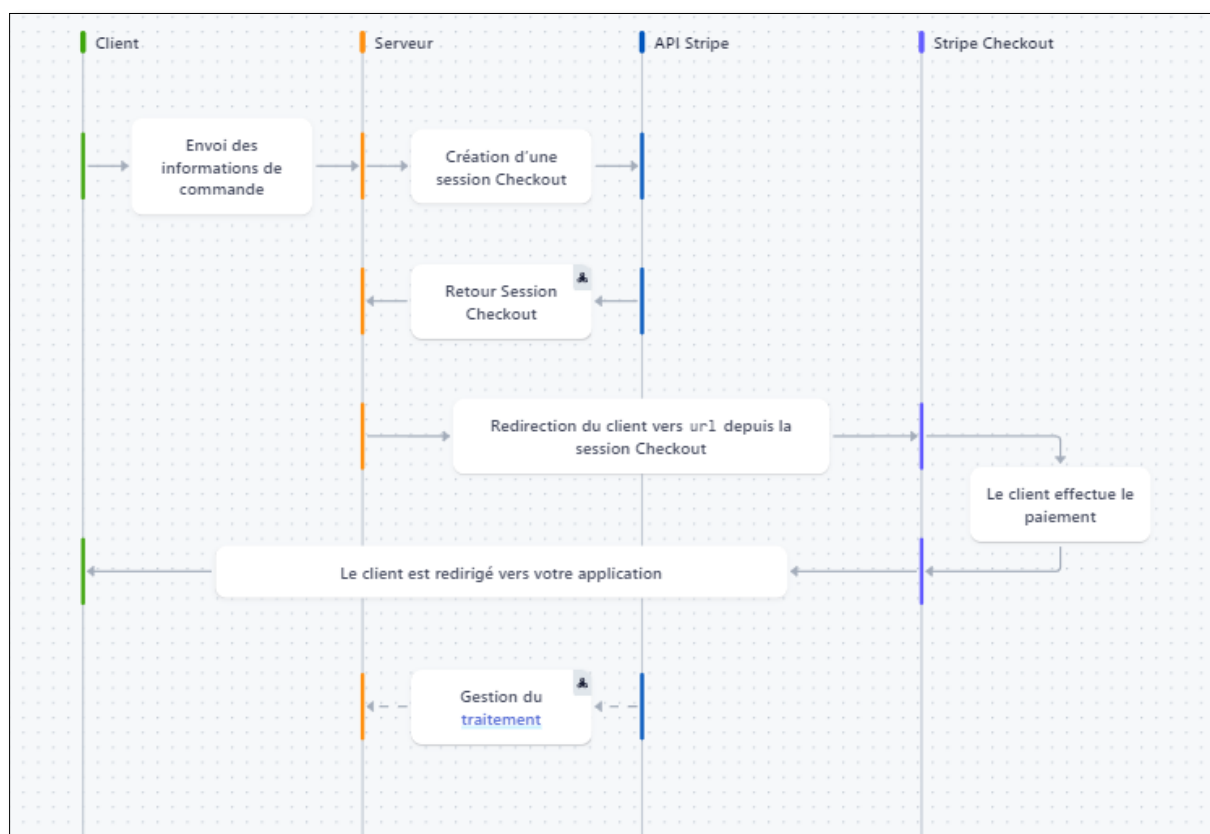


Figure 26 - Schéma du fonctionnement de Stripe checkout (Source : Documentation Stripe)

## 1. La Session Checkout

Pour mettre en place la session de paiement, la première étape consiste à installer l'API Stripe à l'aide de composer.

Une API (Application Programming Interface) est une interface de programmation permettant de connecter un service avec un autre logiciel ou service afin d'échanger des données. Elle permet notamment d'apporter de nouvelles fonctionnalités à notre application de manière simplifiée, sans avoir à connaître le détail de leur mise en œuvre.

Afin de nous identifier et authentifier nos requêtes, la plateforme Stripe nous fournit une clé d'API. Nous avons renseigné cette clé dans notre fichier de configuration `env.local` pour plus de sécurité. Ce fichier n'étant pas pris en compte par notre gestionnaire de version, la clé ne se retrouvera pas visible sur notre dépôt github.

Pour mettre en place le checkout, nous avons inséré la logique dans un nouveau contrôleur « `StripeController` » en suivant les étapes suivantes :

- Tout d'abord on renseigne la clé d'API dans un constructeur pour qu'elle soit appelée depuis le fichier `.env` à chaque nouvelle instance de la session.

```
public function __construct()
{
    Stripe::setApiKey($_ENV["STRIPE_SECRET"]);
}
```

- On crée une nouvelle méthode dans le contrôleur pour gérer la session.
- On récupère la liste des produits de manière similaire à la méthode d’affichage du panier. On récupère le panier présent en session et on crée une boucle sur ce dernier, en respectant le format attendu par Stripe.

```
// On récupère la liste des produits du panier
foreach ($panier as $id => $quantity) {
    $product = $em->getRepository(Product::class)->findOneById($id);

    $productStripe[] = [
        'price_data' => [
            'currency' => 'eur',
            'unit_amount' => $product->getPrice(),
            'product_data' => [
                'name' => $product->getDesignation()
            ],
        ],
        'quantity' => $quantity,
    ];
}
```

- On fait de même avec la cotisation.

```
//On ajoute la cotisation à la liste des produits
$cotisation = $contributionService->checkContribution();
if ($cotisation['price'] > 0) {
    $productStripe[] = [
        'price_data' => [
            'currency' => 'eur',
            'unit_amount' => $cotisation['price'],
            'product_data' => [
                'name' => 'Cotisation valable jusqu\'au ' . $cotisation['endDate']->format('d-m-Y')
            ],
        ],
        'quantity' => 1,
    ];
}
```

- Afin de pouvoir communiquer avec l’API nous devons lui fournir des informations dans un format précis. On lui renseigne toutes les informations nécessaires au paiement : l’email de l’utilisateur, la liste de nos produits, le type de transaction (paiement ou abonnement), 2 liens de redirection en cas d’échec ou validation du paiement.



```
$checkout_session = \Stripe\Checkout\Session::create([
    'customer_email' => $this->getUser()->getEmail(),
    'line_items' => [[
        $productStripe
    ]],
    'mode' => 'payment',
    //aide construction lien:
    //https://stackoverflow.com/questions/67555522/symfony-5-stripe-v3-cant-find-checkout-session-id
    'success_url' => $this
        ->generateUrl('add_order', [], UrlGeneratorInterface::ABSOLUTE_URL)
        . '?session_id={CHECKOUT_SESSION_ID}',
    'cancel_url' => $this->generateUrl('stripe_cancel', [], UrlGeneratorInterface::ABSOLUTE_URL),
]);
```

- Ces informations sont envoyées à l'API, qui génère une URL de redirection permettant à l'utilisateur d'être redirigé vers la plateforme de paiement.

```
return new RedirectResponse($checkout_session->url);
```

## 2. Validation du paiement et enregistrement de la commande

**À la suite des difficultés rencontrées pour faire communiquer notre application avec l'API Stripe depuis un serveur local et compte tenu du fait que cette partie dépasse le cadre de la formation, notre projet n'intégrera pas les webhooks.**

Nous nous sommes tournés vers une solution plus simple consistant à créer la commande au moment de la redirection vers la page « success » sans pour autant attendre une réponse des webhooks. Pour sécuriser cette étape on réalise tout de même 2 vérifications importantes.

### a) Authentification de la session

- Tout d'abord on récupère l'identifiant de la session Checkout en l'intégrant à notre URL de redirection « success ». Le chemin de redirection ressemblera alors au schéma ci-dessous :

```
/order/add_order?session_id=cs_test_a1xtlgDZUG4XGIFQVJcz0EtL36wOM0sN0dazcjvVB0eRUe2xlt0ewkSM4i
```

- Dans la méthode d'ajout de commande (OrdersController), on utilise cet identifiant pour interroger l'API Stripe et vérifier si cette session a bien été enregistrée sur notre compte Stripe. Si la session existe, cela signifie que l'utilisateur est passé par le tunnel de paiement avant d'arriver sur cette page. Si cette session n'existe pas, le serveur renverra une erreur 500 et la commande ne sera pas créée. Cette première vérification empêche un utilisateur malintentionné de valider une commande en entrant simplement le chemin /order/add\_order depuis son navigateur sans passer par le paiement.

```
'success_url' => $this
->generateUrl('add_order', [], UrlGeneratorInterface::ABSOLUTE_URL)
. '?session_id={CHECKOUT_SESSION_ID}';
```

```
$stripe = new StripeClient($ENV["STRIPE_SECRET"]);
$session_id = $request->query->get('session_id');
$session_stripe = $stripe->checkout->sessions->retrieve($session_id);
```

Figure 27- A gauche, on intègre l'id de la session Checkout (StripeController). A droite, on récupère l'identifiant renseigné dans l'url et on interroge l'API Stripe (OrdersController)

Cependant, cette sécurité ne couvre pas tous les cas de figures. Elle nous permet juste de vérifier que le Checkout existe bien mais ne garantit pas que l'utilisateur n'a pas déjà passé cette commande.

## b) Validité de la commande

Prenons un exemple : un utilisateur passe une première commande qu'il paie et finit par être redirigé vers cette page « succès ». Il ouvre un deuxième onglet, dans lequel il remplit à nouveau son panier. Il lui suffirait de revenir sur son premier onglet et d'actualiser la page pour valider sa deuxième commande sans passer par le paiement, la session renseignée dans l'url étant valide.

Pour corriger cette faille, nous sauvegardons la référence du paiement Stripe dans l'entité commande.

```
$order->setStripeId($session_stripe["payment_intent"]);
```

A présent, lorsqu'un utilisateur est redirigé vers la page « success » on vérifie également si l'identifiant de paiement de la session checkout correspond à un identifiant présent dans notre base de données.

```
$orderVerification = $em
->getRepository(Order::class)
->findOneBy(["stripeId" => $session_stripe["payment_intent"]]);
```

On utilise une condition « if » pour vérifier que notre résultat renvoie bien null (résultat vide) et que le panier n'est pas vide avant de pouvoir créer la commande et l'envoyer en base de données.

```

if (!$orderVerification) {
    if ($panier !== []) {
        //Création objet commande et insertion des données
    }
}

```

### c) Sauvegarde de la commande

Une fois les informations de la commande récupérée et les vérifications d'usage faites, nous pouvons créer une nouvelle instance de commande. On y insère les données la concernant : l'utilisateur, la référence et l'identifiant de paiement Stripe.

```

//Création objet commande et insertion des données
$order = new Order();
//On créer la référence au format : id de l'utilisateur,date,heure,minutes,secondes
$time = new \DateTimeImmutable();
$reference = $user->getId() . $time->format('Ymdhis');

$order->setUser($user);
$order->setReference($reference);
$order->setStripeId($session_stripe["payment_intent"]);

```

**Dans cette phase de test, on considère que le paiement a bien été validé sans attendre une réponse de la part de l'API Stripe (Partie « traitement du paiement », figure 24). Cependant, il existe des cas où l'application Stripe redirige l'utilisateur vers la page succès, sans que la validation du paiement ne soit confirmée par les services de Stripe. La réponse arrive sur notre serveur après la redirection et elle peut être négative. Si on souhaite mettre notre site en production il faudra mettre en place le déclenchement des actions avec les webhooks.**

```

/*
 * //! Attention, la validation du paiement est automatisée pour les tests en local,
 * mais elle devra être supprimée lorsque les webhooks seront fonctionnels.
 */
$order->setIsPaid(true);

```

```

//Création du détail de commande insertion des données
foreach ($panier as $id => $qte) {
    $orderDetails = new OrderDetails();
    //On récupère le produit
    $product = $productRepository->find($id);
    if (!$product) {
        throw $this->createNotFoundException(
            'No product found for id ' . $id
        );
    }
    $price = $product->getPrice();
    //On remplit orderDetails
    $orderDetails->setProduct($product);
    $orderDetails->setPrice($price);
    $orderDetails->setQuantity($qte);
    //On rajoute OrderDetails à la commande
    $order->addOrderDetail($orderDetails);
    //On retire du stock la quantité commandée.
    $newStock = $product->getStock() - $qte;
    $product->setStock($newStock);
}

```

On peut à présent créer une boucle sur le panier et créer une nouvelle instance de « orderDetails » pour chaque élément du panier et y insérer les données. Durant cette étape, on actualise également nos stocks.

Dernière étape avant de persister nos données et de les transférer à la base de données, on vérifie si l'utilisateur ne possède pas encore de date de fin de cotisation ou si cette date est différente de celle calculée par le service « MembershipContributionService ». Si une des 2 conditions est validée cela signifie que l'utilisateur a réglé sa cotisation lors du paiement. On renseigne alors dans la commande le prix de cotisation et on insère la nouvelle date de fin de cotisation dans l'entité User.

```
//Ajout cotisation à la commande et mise à jour nouvelle date à l'utilisateur
$endDate = $user->getMembershipContributionEndDate();
if (!isset($endDate) || $cotisation['endDate']->format('Y-m-d') != $endDate->format('Y-m-d')) {
    $order->setContribution($cotisation['price']);
    $user->setMembershipContributionEndDate($cotisation['endDate']);
}
```

## VII. Sécurité

---

Installé de base dans la version complète de Symfony, le bundle Security contient de nombreux outils nous facilitant la mise en place des composants de sécurité. Le paramétrage de ces outils est en partie centralisé dans le fichier `security.yaml` présent dans le dossier `package` de notre application. Nous allons détailler dans cette partie la mise en place des composants de sécurité présents dans ce projet.

### A. Sécurisation de la connexion

Élément important pour sécuriser les données des utilisateurs et pour l'application en général, les mots de passes doivent être de plus en plus robustes. En effet, les attaques par force brute ou par dictionnaire restent simple à mettre en place et largement utilisées.

La robustesse d'un mot de passe est définie par l'entropie qui est définie par la quantité de hasard pour un mot de passe ou une clé cryptographique. Cela correspond à son degré d'imprédictibilité théorique et donc son attaque par force brute. (Source CNIL) On mesure l'entropie en bits. Pour un site de e-commerce la CNIL préconise un mode de passe ayant une entropie minimum de 50bits accompagnée de mesures complémentaires.

- Nous avons mis en place un mot de passe à l'aide d'une regex composée 8 caractères minimum contenant au minimum 1 lettre majuscule, 1 lettre minuscule, 1 chiffre et 1 caractère spécial, correspondant à une entropie de 50 bits.
- Pour compléter la sécurisation, nous avons également limité le nombre de tentatives de connexions pour une même adresse IP pour un compte donné. Cela permet de limiter les tentatives de connexion par force brute. Symfony dispose d'un composant complémentaire « `rate limiter` » paramétrable depuis le fichier `security.yaml`. Nous avons limité à 3 tentatives de connexion par minute.
- 3<sup>ème</sup> élément important, nous avons fait appel au service reCAPTCHA (V3) proposé par Google, permettant de vérifier que seul un humain puisse valider un formulaire. Ce test CAPTCHA a également été ajouté aux formulaires de modification de mot de passe et dans le formulaire d'ajout de review.

Autre élément indispensable pour protéger les mots de passe est le hachage. Le hachage permet de stocker une empreinte du mot de passe et non pas le mot de passe en lui-même dans la base de données. En cas de vol de données cela rendra très difficile, voire impossible la lecture du mot de passe par une personne malveillante. C'est pourquoi Symfony va hacher les mots de passe par l'intermédiaire du composant `password Hasher`. Par défaut Symfony sélectionne l'algorithme le plus sûr actuellement disponible.

2 éléments sont à prendre en compte pour que le hachage soit efficace :

- Le temps de traitement, car plus la puissance de traitement requise est élevée, plus il faudra de temps pour casser le mot de passe.
- Le grain de sel ou `salts`, est un élément aléatoire ajouté durant le processus de hachage pour éliminer la possibilité d'attaque par dictionnaire. En effet, sans grain de sel, un mot de passe serait toujours haché de manière identique pour un algorithme donné. Par exemple le mot de

passer « parapluie » pourrait être retrouvé facilement par un hacker ayant accès des dictionnaires de correspondances mot de passe / hash. Le grain de sel permet de palier à ce problème.

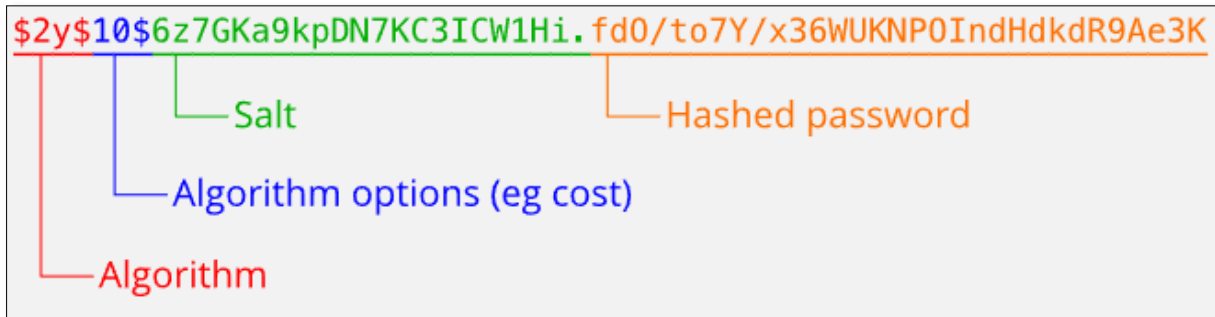


Figure 28- exemple de mot de passe haché (source : php.net )

## B. Sécurisation des routes

Il est important de définir différents droits d'accès à l'application afin de la sécuriser. En effet, s'il est tout à fait possible pour un utilisateur non connecté d'accéder à la boutique et aux différents produits, on ne souhaite pas qu'il puisse passer commande. De même le back office du site, sans restriction particulière n'importe quel utilisateur pourrait accéder au panneau d'administration simplement en accédant l'URL/admin.

Lors de la création de notre entité User, Symfony a intégré automatiquement un attribut rôles défini par un tableau et donne un rôle par défaut « Role\_User » à chaque utilisateur inscrit.

```
/**
 * @see UserInterface
 */
public function getRoles(): array
{
    $roles = $this->roles;
    // guarantee every user at least has ROLE_USER
    $roles[] = 'ROLE_USER';

    return array_unique($roles);
}
```

2 méthodes sont à notre disposition pour bloquer l'accès à certaines routes :

- Dans le fichier security.yaml on peut sécuriser facilement des sections entières du site grâce au paramètre « access\_control ». On définit les routes et les rôles associés à ces dernières.

```
# Easy way to control access for large sections of your site
# Note: Only the *first* access control that matches will be used
access_control:
    - { path: ^/admin, roles: ROLE_ADMIN }
    - { path: ^/user, roles: ROLE_USER }
    - { path: ^/order, roles: ROLE_USER }
```

- Lorsqu'il s'agit de restreindre l'accès à des vues spécifiques, on peut paramétrer l'action d'un contrôleur (ou l'intégralité d'un contrôleur) à l'aide de l'attribut `IsGranted`.

*Exemple : un utilisateur non connecté peut remplir son panier, mais il ne pourra pas accéder à la page de vérification de l'adresse de facturation s'il n'est pas connecté.*

```
#[Route('/cart/check_address', name: 'checkBillingAddress')]
#[IsGranted('ROLE_USER')]
public function checkBillingAddress()
```

A présent, lorsqu'un utilisateur tente d'accéder à une page dont il n'a pas les accès, il est automatiquement redirigé vers la page de connexion.

## C. Protection contre les injections SQL

Lorsqu'un utilisateur envoie des requêtes, l'application interroge la base de données afin de construire une réponse. Une attaque par injection SQL intervient lorsqu'un utilisateur malintentionné arrive à altérer la requête en l'utilisant pour injecter d'autres requêtes.

Pour contrer les injections SQL, Symfony propose des moyens protéger notre application :

- Tout d'abord on doit mettre en place la validation des inputs. Par exemple, si on attend qu'un utilisateur rentre un nombre dans un formulaire, on va vérifier que la donnée transmise est effectivement un nombre. Cela va se traduire par la mise en place de contraintes de validation (ou « Validation constraints ») au niveau de nos entités, à l'aide des attributs `#[Assert\...]`. Dans nos formulaires, on précisera également le type d'input attendu (ex : `NumberType`). C'est un moyen simple d'éviter des injections SQL, mais il faut bien penser à filtrer tous les inputs !
- Une deuxième méthode complémentaire consiste à préparer les requêtes au lieu de les transmettre directement à la base de données. Ces différents paramètres sont injectés dans des espaces réservés, qui sont échappés avant d'être intégrés à la requête. En utilisant les fonctions de base proposée par Doctrine notre application est protégée lors des requêtes classiques. Lorsque nous utilisons des requêtes DQL personnalisées, les données entrées sont paramétrées et donc sécurisées grâce à la fonction `setParameters()`.

## D. Protection contre le Cross Site Scripting (faille XSS)

Une attaque par Cross site Scripting (XSS) consiste à injecter un code malveillant sur un site web qui sera ensuite chargé par le navigateur de l'utilisateur du site infecté. Ces scripts peuvent permettre à l'attaquant de rediriger l'utilisateur vers un site malveillant, de récupérer les cookies de session, diffuser un Malware, etc...

Pour se prémunir des attaques XSS il faut respecter 2 règles :

- On filtre les inputs, ce qui est réalisé sur Symfony à l'aide des contraintes de type dans les formulaires et les asserts dans les entités, pour s'assurer que nos formulaires renvoient les bons types de données. Cependant pour les champs plus complexes (type `textArea`) pouvant contenir des balises et autres caractères spéciaux on doit utiliser des `data Transformers`. Dans notre projet, l'éditeur de texte (`tinyMCE`) présent dans le formulaire d'ajout d'un commentaire est configuré de base avec une sécurité contre les failles XSS. Il est équipé de `DOMPurify`, un XSS sanitizer pour HTML.
- On échappe les données en sortie. Le moteur de template Twig est configuré de base pour échapper les données en sortie.

Le headers Content Security Policy (CSP) implémenté dans Symfony avec le composant HTTPFoundation indique au navigateur les sources autorisées à charger du contenu sur notre site ce qui empêche l'intrusion de sources inconnues.

## E. Protection contre les attaques CSRF (Cross-site Request Forgery)

Une faille CSRF (Cross-site Request Forgery) ou falsification de requêtes inter-site permet à un attaquant de forcer un utilisateur authentifié d'exécuter des actions à son insu. Cette attaque peut permettre à l'attaquant de rentrer des données dans un formulaire sans que l'utilisateur s'en aperçoive, par exemple en modifiant son mot de passe.

Pour se prémunir de ce type d'attaque Symfony implémente des jetons CSRF (ou CSRF token). Ce sont des valeurs uniques hachées générées aléatoirement côté serveur et envoyées au client, afin de vérifier que les requêtes proviennent bien de l'utilisateur.

A chaque connexion l'utilisateur reçoit un jeton enregistré en session et chaque formulaire créé à partir de FormBuilder génère automatiquement un jeton. Lorsqu'un formulaire est soumis, le jeton côté serveur est comparé au jeton côté client permettant ainsi de vérifier que la requête a bien été envoyée depuis le site auquel l'utilisateur est connectée et non depuis un site tiers.



## VIII. Recherches anglophones

---

Au cours de ce projet, de nombreuses recherches ont été menées sur des sites anglophones, que ce soit de la documentation pour comprendre le fonctionnement des différents bundles et composants de Symfony, ainsi que des recherches sur des forums pour débloquer une situation.

Prenons l'exemple de la documentation Symfony concernant l'utilisation de la Session, superglobale qui nous a suivis tout au long de la construction de la boutique.

### A. Extrait de la documentation Symfony Sessions

The Symfony HttpFoundation component has a very powerful and flexible session subsystem which is designed to provide session management that you can use to store information about the user between requests through a clear object-oriented interface using a variety of session storage drivers.

Symfony sessions are designed to replace the usage of the `$_SESSION` super global and native PHP functions related to manipulating the session like `session_start()`, `session_regenerate_id()`, `session_id()`, `session_name()`, and `session_destroy()`.

*Sessions are only started if you read or write from it.*

#### Installation

You need to install the HttpFoundation component to handle sessions: « `composer require symfony/http-foundation` »

#### Basic Usage

The session is available through the Request object and the RequestStack service. Symfony injects the `request_stack` service in services and controllers if you type-hint an argument with `RequestStack`:

```
use Symfony\Component\HttpFoundation\RequestStack;

class SomeService
{
    public function __construct(
        private RequestStack $requestStack,
    ) {
        // Accessing the session in the constructor is *NOT* recommended, since
        // it might not be accessible yet or lead to unwanted side-effects
        // $this->session = $requestStack->getSession();
    }

    public function someMethod(): void
    {
        $session = $this->requestStack->getSession();
    }
}
```

#### Session Attributes

PHP's session management requires the use of the `$_SESSION` super-global. However, this interferes with code testability and encapsulation in an OOP paradigm. To help overcome this, Symfony uses *session bags* linked to the session to encapsulate a specific dataset of **attributes**.

This approach mitigates namespace pollution within the `$_SESSION` super-global because each bag stores all its data under a unique namespace. This allows Symfony to peacefully co-exist with other applications or libraries that might use the `$_SESSION` super-global and all data remains completely compatible with Symfony's session management.

A session bag is a PHP object that acts like an array:

```
1 // stores an attribute for reuse during a later user request
2 $session->set('attribute-name', 'attribute-value');
3
4 // gets an attribute by name
5 $foo = $session->get('foo');
6
7 // the second argument is the value returned when the attribute doesn't exist
8 $filters = $session->get('filters', []);
```

Stored attributes remain in the session for the remainder of that user's session. By default, session attributes are key-value pairs managed with the `AttributeBag` class.

## B. Version traduite

Le composant Symfony HttpFoundation contient un sous-système de session puissant et flexible qui est conçu pour fournir une gestion de session que vous pouvez utiliser pour stocker des informations sur l'utilisateur entre les requêtes au travers d'un interface orienté objets, en utilisant une variété diverse de pilotes de stockage de session.

Les sessions Symfony sont conçues pour remplacer l'utilisation de la superglobale `$_Session` ainsi que les fonctions natives à php et relatives à la manipulation de la session: `session_start()`, `session_regenerate_id()`, `session_id()`, `session_name()` et `session_destroy()`.

*Note : les sessions ne démarrent que si vous y écrivez ou lisez des informations.*

### Installation

Vous devez installer le composant HttpFoundation pour pouvoir manipuler les sessions :

« `composer require symfony/http-foundation` »

### Utilisation

La session est disponible à travers l'objet « Request » et du service « RequestStack ». Symfony injecte le service « request\_stack » dans les contrôleurs et services si vous utilisez une variable de type RequestStack.

*Commentaire : accéder à la session dans un constructeur n'est pas recommandé, car elle pourrait ne pas être encore accessible ou provoquer des effets indésirables.*

### Attributs de sessions

Gérer les sessions en PHP nécessite d'utiliser la superglobale `$_session`. Cependant, cette pratique interfère avec la testabilité du code et l'encapsulation dans un paradigme de programmation orienté

objet (POO). Pour pallier ce problème, Symfony utilise des « conteneurs de sessions » (ou « sacs de sessions ») liés à la session pour encapsuler un ensemble spécifique d'attributs.

Cette approche déminue la pollution des espaces de noms de la variable `$_SESSION` car chaque conteneur (ou sac) stocke ses données dans un unique espace de nom. Cela permet à Symfony de co-exister avec d'autres applications et librairies utilisant la superglobal `$_SESSION` et toutes ses données restent compatibles avec le système de gestion de session de Symfony.

Un conteneur de session (ou sac de session) est un objet PHP qui fonctionne comme un tableau :

```
// Stocker un attribut pour le réutiliser plus tard lors d'une requête de l'utilisateur.  
$session->set('attribute-name', 'attribute-value');
```

```
//Récupérer un attribut à partir de son nom  
$foo = $session->get('foo');
```

```
//Le deuxième argument est la valeur à retourner si l'attribut n'existe pas.  
$filters = $session->get('filters', []);
```

Les attributs sont stockés en session pour le reste de la session de l'utilisateur. Par défaut, les attributs de sessions sont des paires clés-valeurs gérés par la classe `AttributeBag`.

## IX. Difficultés rencontrées

---

**Travailler seul :** le fait de travailler seul et sans client réel a augmenté la difficulté dans la réalisation du projet. Travailler avec un client implique de respecter ses exigences, mais permet d'avoir un point de vue extérieur sur son travail et d'avoir une vision concrète des objectifs auxquelles l'application doit répondre. Le fait de travailler seul m'a obligé à penser le projet dans son intégralité, à imaginer le contexte de sa réalisation et tous les cas de figures sans pouvoir confronter mes idées avec quelqu'un. Cela a été source d'erreurs et d'oublis et donc de retards dans mon avancement, malgré la préparation en amont. Cependant je suis conscient que n'importe quel projet peut être amené être modifié en cours de route, même en travaillant au sein d'une équipe, mais le risque d'erreur et d'oublis sera moins élevé. Ce genre d'oubli peut être également lié au manque d'expérience.

**L'expérience :** le manque d'expérience rend la gestion du temps de travail plus complexe. Apprendre à maîtriser une nouvelle technologie n'est jamais une chose facile, et le temps nécessaire à l'apprentissage est difficile à estimer. Après 2 mois de travail sur ce projet, je me suis rendu compte que j'étais capable de mettre en place des fonctionnalités beaucoup plus rapidement que dans les premières semaines. Je connais mieux mon code et je maîtrise mieux le Design Pattern.

**Système de paiement :** Implémenter le système de paiement Stripe a été une difficulté supplémentaire durant la réalisation du projet. La mise en place du checkout n'était pas une difficulté en soit, mais la mise en place des webhooks m'a fait perdre beaucoup de temps. J'ai fait plusieurs tentatives pour implémenter cette fonctionnalité, par obstination et envie de présenter un projet conforme à mes attentes. N'étant pas primordial dans le cadre de ma formation, j'ai dû me résoudre à mettre cette fonctionnalité de côté pour me concentrer sur d'autres tâches importantes. Cela m'a fait perdre du temps dans la réalisation du projet.

## X. Perspectives d'améliorations

---

- Implémenter les fonctionnalités du cahier des charges qui n'ont pas pu être mises en place par manque de temps. L'agenda des événements et le forum.
- Améliorer l'expérience utilisateur avec du contenu dynamique en mettant en place de l'Ajap afin d'actualiser uniquement la zone d'affichage des produits sur la page index produit et les avis clients dans la vue détails produits.
- Mettre en place l'API adresse disponible sur data.gouv pour améliorer le formulaire d'inscription de l'utilisateur.
- Refactoriser le code. Il est encore possible d'améliorer l'écriture de certaines fonctions, notamment celles réalisées en début de projet afin d'améliorer la maintenabilité du code. On pourrait créer un service spécifique au panier afin de diminuer la logique présente dans le contrôleur.
- Améliorer le back office :
  - Augmenter le nombre de paramètres de la boutique modifiables par l'administrateur du site (ex : horaires d'ouverture, la couleur des mises en avant de produits visibles sur les articles dans l'index)
  - Override les Templates de certaines vues sur EasyAdmin, notamment pour améliorer l'affichage du détail des commandes.
- Ajouter un carrousel sur la page d'accueil pour affichage dynamique des nouveautés.

## XI. Conclusion

---

Ce projet m'a permis de mettre en pratique mes connaissances acquises au cours de ma formation et mon stage, tout en me permettant d'en acquérir de nouvelles. A travers ce projet, j'ai pu adopter une démarche professionnelle dans sa réalisation et me confronter aux contraintes de mon futur métier.

Par exemple, la contrainte de temps m'a forcé à organiser mon travail et à prendre des décisions quant aux différentes parties à développer. Se lancer dans un tel projet pour la première fois m'a permis de me rendre compte de mes capacités à surmonter les problèmes, que ce soit en implémentant de nouvelles fonctionnalités ou en résolvant de bugs. Ce projet m'a fait évoluer et m'a permis d'acquérir de l'expérience.

Je suis à présent conforté dans mon choix de reconversion. J'espère pouvoir mettre à profits très prochainement mes nouvelles compétences dans des projets professionnels au sein d'une équipe en entreprise. J'ai hâte pouvoir découvrir de nouveaux projets et continuer à améliorer mes compétences en développant de nouveaux projets qu'ils soient personnels ou professionnels.

## XII. Annexes – les maquettes



### Bienvenue sur le site de l'association l'échoppe - Bar associatif et vente de bières locales -

#### Présentation



"Sed ut perspiciatis unde omnis iste natus error sit voluptatem accusantium doloremque laudantium, totam rem aperiam, eaque ipsa quae ab illo inventore veritatis et quasi architecto beatae vitae dicta sunt explicabo. Nemo enim ipsam voluptatem quia voluptas sit aspernatur aut odit aut fugit, sed quia consequuntur magni dolores eos qui ratione voluptatem sequi nesciunt.

Neque porro quisquam est, qui dolorem ipsum quia dolor sit amet, consectetur, adipisci velit, sed quia non numquam eius modi tempora incidunt ut labore et dolore magnam aliquam quaerat voluptatem. Ut enim ad minima veniam, quis nostrum exercitationem ullam corporis suscipit laboriosam, nisi ut aliquid ex ea commodi consequatur? Quis autem vel eum iure reprehenderit qui in ea voluptate velit esse quam nihil molestiae consequatur, vel illum qui dolorem eum fugiat quo voluptas nulla pariatur?"

[En savoir plus...](#)

[REJOIGNEZ-NOUS](#)

#### Les nouveautés

 Plus d'infos	 Plus d'infos	 Plus d'infos	 Plus d'infos
Désignation du produit - Style Bière 2.95 € T.T.C.	Désignation du produit - Style Bière 2.95 € T.T.C.	Désignation du produit - Style Bière 2.95 € T.T.C.	Désignation du produit - Style Bière 2.95 € T.T.C.

#### Le fonctionnement



L'abus d'alcool est dangereux pour la santé, à consommer avec modération. Déconseillé aux femmes enceintes.  
La vente d'alcool est interdite aux mineurs, en accordant à ce site vous déclarez avoir au moins 18 ans. CODE DE LA SANTE PUBLIQUE, ART. L. 3342-1 et L. 3353-3

#### Notre association

- > Mentions légales
- > Conditions générales de vente
- > Conditions d'utilisation
- > Données personnelles
- > Plan du site

#### Nous contacter



[Nous contacter](#)

#### Informations

01, rue du Paradis du Houblon  
67000 Strasbourg  
Mar-Ven: 14h-20h, Sam: 12h-18h

Copyright 2023 - Association l'échoppe

## Filtrer par

### Brasserie

- ☒Brasserie 1
- ☒Brasserie avec un nom long
- ☒Brasserie test 5
- ☐Brasserie 10
- ☐Brasserie 2
- ☒nom court

### Type de bière

- ☐ I.P.A
- ☐ Pils
- ☐ Lager
- ☒ Gose
- ☐ Stout
- ☐ Neipa
- ☐ Weisse
- ☐ Neipa
- ☐ Pale Ale
- ☐ Sours
- ☐ WitBier
- ☐ Ambrée
- ☐ Amber Ale
- ☐ Irish/Scottish

### Amertume

- ☐ Faible
- ☐ Moyenne
- ☐ Importante

### Taux d'alcool

- ☐ 0°C
- ☐ + 0°C à 3°C
- ☐ + 3°C à 5°C
- ☐ + 5°C à 8°C
- ☐ + 8°C à 10°C
- ☐ + de 10°C

### Type de production

- ☐ Permanente
- ☐ Ephémère

### Note

- ☐ de 1 à 2
- ☐ de 2 à 3
- ☐ de 3 à 4
- ☐ 4 et +

Filtrer les résultats

## Trier par ▼

Prix croissant

Prix décroissant

Notes croissante

Notes décroissante



Nouveauté

Plus d'infos

Désignation du produit -  
contenance  
2.95 € T.T.C

★★★★☆ (8)



Plus d'infos

Désignation du produit -  
contenance  
2.95 € T.T.C

★★★★☆ (8)



Nouveauté

Plus d'infos

Désignation du produit -  
contenance  
2.95 € T.T.C

★★★★☆ (8)



Plus d'infos

Désignation du produit -  
contenance  
2.95 € T.T.C

★★★★☆ (8)



Plus d'infos

Désignation du produit -  
contenance  
2.95 € T.T.C

★★★★☆ (8)



Nouveauté

Plus d'infos

Désignation du produit -  
contenance  
2.95 € T.T.C

★★★★☆ (8)



Plus d'infos

Désignation du produit -  
contenance  
2.95 € T.T.C

★★★★☆ (8)



Nouveauté

Plus d'infos

Désignation du produit -  
contenance  
2.95 € T.T.C

★★★★☆ (8)



Nouveauté

Plus d'infos

Désignation du produit -  
contenance  
2.95 € T.T.C

★★★★☆ (8)



Nouveauté

Plus d'infos

Désignation du produit -  
contenance  
2.95 € T.T.C

★★★★☆ (8)



Plus d'infos

Désignation du produit -  
contenance  
2.95 € T.T.C

★★★★☆ (8)



Nouveauté

Plus d'infos

Désignation du produit -  
contenance  
2.95 € T.T.C

★★★★☆ (8)

## Notre association

- > Mentions légales
- > Conditions générales de vente
- > Conditions d'utilisation
- > Données personnelles
- > Plan du site

## Nous contacter



Nous contacter

## Informations

01, rue du Paradis du Houblon  
 67000 Strasbourg  
 Mar-Ven: 14h-20h, Sam: 12h-18h



[Retour boutique](#)

### Désignation du produit

Description du produit Sed ut perspiciatis unde omnis iste natus error sit voluptatem accusantium doloremque laudantium, totam rem aperiam, eaque ipsa quae ab illo inventore veritatis et quasi architecto beatae

3,95 €  
(soit 11,96 € le litre)

Quantité  
- 5 +

[Ajouter au panier](#)

### Informations détaillées

Brasserie	Brasserie 1
Ingrédients	Malt d'orge (Pils, Munich 20), houblons (Pekko, Mistral, Triskel), levures (SafAle Us-05)
Taux d'alcool	5 °C
Amertume	56 I.B.U
Type de Bière	I.P.A.

### Avis

[Ajouter un avis](#)

5 reviews  3,5

Utilisateur 56

Le 01/01/2023 à 17:05:05 

Description du produit Sed ut perspiciatis unde omnis iste natus error sit voluptatem accusantium doloremque laudantium, totam rem aperiam, eaque ipsa quae ab illo inventore veritatis et quasi architecto beatae

Utilisateur 5

Modifié le 30/12/2022 à 17:05:05 

Description du produit Sed ut perspiciatis unde omnis iste natus error.

Utilisateur 78

Le 30/12/2022 à 16:05:05 

Description du produit Sed ut perspiciatis unde omnis iste natus error sit voluptatem accusantium doloremque laudantium.

[Voir plus](#)


L'abus d'alcool est dangereux pour la santé, à consommer avec modération. Déconseillé aux femmes enceintes.  
La vente d'alcool est interdite aux mineurs, en accédant à ce site vous déclarez avoir au moins 18 ans. CODE DE LA SANTE PUBLIQUE, ART. L. 3342-1 et L. 3353-3

#### Notre association

- > Mentions légales
- > Conditions générales de vente
- > Conditions d'utilisation
- > Données personnelles
- > Plan du site

#### Nous contacter



 Nous contacter

#### Informations

 01, rue du Paradis du Houblon  
67000 Strasbourg  
Mar-Ven: 14h-20h, Sam: 12h-18h

Copyright 2023 - Association l'échoppe



Bienvenue sur le  
site de  
l'association  
l'échoppe  
- Bar associatif et vente  
de bières locales -

#### Présentation



Plus qu'un simple lieu de rencontre, L'ÉCHOPPE est un véritable lieu de vie. C'est ici que se retrouvent les amateurs de bières locales, pour partager leur passion et découvrir les nouvelles créations de nos producteurs locaux.

Notre bar est ouvert tous les jours, de 16h à 22h. Nous proposons une carte de bières locales, ainsi qu'une sélection de vins et de plats. Nous sommes également à votre service pour organiser des événements et des soirées thématiques.

En savoir plus...

#### REJOIGNEZ-NOUS

#### Les nouveautés



Désignation du produit - Blue Style  
2,85 € T.T.C.



Désignation du produit - Plus Style  
2,85 € T.T.C.



Désignation du produit - Plus Style  
2,85 € T.T.C.



Désignation du produit - Plus Style  
2,85 € T.T.C.

#### LA BOUQUIN

#### Le fonctionnement



Notre association  
Mentions légales  
Conditions générales de vente  
Conditions d'utilisation  
Données personnelles  
Plan du site

Nous contacter  
Instagram Facebook Twitter  
Nous contacter

Informations  
01, rue du Paradis du Houblon  
67000 Strasbourg  
Mar-Ven: 14h-20h, Sam: 12h-18h

Copyright 2023 - Association l'échoppe



Mon panier 0 0

Mon panier 0 0

Mon panier 0 0

Mon panier 0 0

Mon panier 0 0

Mon panier 0 0

Mon panier 0 0

Mon panier 0 0

Mon panier 0 0

Mon panier 0 0

Mon panier 0 0

Mon panier 0 0

Mon panier 0 0

Mon panier 0 0

Mon panier 0 0

Mon panier 0 0

Mon panier 0 0

Mon panier 0 0

Mon panier 0 0

Mon panier 0 0

Mon panier 0 0

Mon panier 0 0

Mon panier 0 0

Mon panier 0 0

Mon panier 0 0

Mon panier 0 0

Mon panier 0 0

Mon panier 0 0

Mon panier 0 0

Mon panier 0 0

Mon panier 0 0

Mon panier 0 0

Mon panier 0 0

Mon panier 0 0

Mon panier 0 0

Mon panier 0 0

Mon panier 0 0

Mon panier 0 0

Mon panier 0 0

Mon panier 0 0

Mon panier 0 0

Mon panier 0 0

Mon panier 0 0

Mon panier 0 0

Mon panier 0 0

Mon panier 0 0

Mon panier 0 0

Mon panier 0 0

Mon panier 0 0

Mon panier 0 0

Mon panier 0 0

Mon panier 0 0

Mon panier 0 0

Mon panier 0 0

Mon panier 0 0

Mon panier 0 0

Mon panier 0 0

Mon panier 0 0

Mon panier 0 0

Mon panier 0 0

Mon panier 0 0

Mon panier 0 0

Mon panier 0 0

Mon panier 0 0

Mon panier 0 0

Mon panier 0 0

Mon panier 0 0

Mon panier 0 0

Mon panier 0 0

Mon panier 0 0

Mon panier 0 0

Mon panier 0 0

Mon panier 0 0

Mon panier 0 0

Mon panier 0 0

Mon panier 0 0

Mon panier 0 0

Mon panier 0 0

Mon panier 0 0

Mon panier 0 0

Mon panier 0 0

Mon panier 0 0

Mon panier 0 0

Mon panier 0 0

Mon panier 0 0

Mon panier 0 0



Retour boutique



#### Désignation du produit

Description du produit Sed ut perspiciatis unde omnis iste natus error sit voluptatem accusantium doloremque laudantium, totam rem aperiam, eaque ipsa quae ab illo inventore veritatis et quasi architecto beate

3,95 €

(soit 11,96 € le litre)

Quantité

- 5 +

Ajouter au panier

#### Informations détaillées

Brasserie 1
Malt d'orge (Pils, Munich 20), houblons (Pekko, Mistral, Triskel), levures (SaAle Us-05)
5 °C
56 l.B.U
I.P.A.

#### Avis

Ajouter un avis

5 reviews ★★★★★ 3,5

Utilisateur 56 ★★★★★ Le 01/01/2023 à 17:05:05

Description du produit Sed ut perspiciatis unde omnis iste natus error sit voluptatem accusantium doloremque laudantium, totam rem aperiam, eaque ipsa quae ab illo inventore veritatis et quasi architecto beate

Utilisateur 5 ★★★★★ Modifié le 30/12/2022 à 17:05:05

Description du produit Sed ut perspiciatis unde omnis iste natus error.

Utilisateur 78 ★★★★★ Le 30/12/2022 à 16:05:05

Description du produit Sed ut perspiciatis unde omnis iste natus error sit voluptatem accusantium doloremque laudantium

Voir plus

L'abus d'alcool est dangereux pour la santé, à consommer avec modération. Déconseillé aux femmes enceintes. La vente d'alcool est interdite aux mineurs, en procédant à ce site vous déclarez avoir au moins 18 ans. CODE DE LA SANTÉ PUBLIQUE, ART. L. 3342-1 et L. 3353-3

#### Notre association

Mentions légales  
Conditions générales de vente  
Conditions d'utilisation  
Données personnelles  
Plan du site

#### Nous contacter

Instagram Facebook Twitter  
Nous contacter

#### Informations

01, rue du Paradis du Houblon  
67000 Strasbourg  
Mar-Ven: 14h-20h, Sam: 12h-18h

Copyright 2023 - Association l'échoppe