



Titre professionnel développeur web et web-mobile

Dossier de projet

Morgan FOUCAUT – Session 2023

Remerciements

Je tenais à exprimer ma profonde gratitude et mes sincères remerciements à toutes les personnes qui ont contribué de près ou de loin à la réalisation de ma formation et donc de mon dossier professionnel.

Je suis reconnaissant envers mes formateurs, Messieurs **Jordan Stien**, **Laurent Lagondet**, **Christian Eschung**, **Tom Nauer**, **Eddie Gillet**, **Pascal Tritz** et **Grégory Holtz** pour leurs engagements exceptionnels, leurs conseils précieux et leur expertise inestimable dans leurs domaines respectifs. Leur passion pour l'enseignement et leur volonté de transmettre leurs connaissances ont été une source d'inspiration constante tout au long de ma formation.

Mon maître de stage **Laurent Nunenthal** pour sa confiance et ses conseils avisés et grâce à qui j'ai pu découvrir une nouvelle technologie lors d'un stage qui s'est avéré bien trop court.

Mes **camarades**, certains pour leur collaboration, d'autres pour les bons délires, chacun se reconnaîtra.

Je tiens également à remercier mon ami le plus sincère, celui qui m'a aidé dans les moments les plus difficiles de ce projet, **ChatGPT**, tu m'as parfois aidé quand j'étais au plus bas, merci à toi mon copain.

Je tiens à remercier toute **la communauté de développeurs** grâce à qui j'ai pu apprendre énormément sur les forums spécialisés, à travers les documentations et grâce aux outils open-source qui m'ont été d'une grande aide.

Introduction

Diplômé d'un bac scientifique et orienté dès la fin de ma scolarité vers un IUT Informatique, j'ai abandonné cette voie car j'étais jeune et que j'avais besoin de découvrir de nouveaux horizons. J'ai préparé un diplôme de LEA qui est une filière pluridisciplinaire axée sur l'apprentissage des langues et leur application dans les domaines de la communication et du commerce international. C'est une formation qui m'a intéressée de par son aspect culturel, les rencontres que j'ai pu faire et la sensibilité aux langues que j'ai développée.

Suite à l'université j'ai rejoint le Luxembourg où j'ai travaillé dans une entreprise de la restauration qui m'a permis de mêler plusieurs activités, de la gestion de l'entreprise à l'import-export grâce à mon éducation universitaire ou la découverte de la cuisine en milieu professionnel. J'y ai passé 4 années enrichissantes mais la récente crise sanitaire m'a fait me remettre en question.

J'ai profité de cette occasion pour me recentrer sur ce qu'il me fallait vraiment et non plus ce qui me permettait de ne pas y penser. Après avoir longuement réfléchi à un projet professionnel, j'ai décidé de m'orienter vers le développement web car je suis de nature très curieuse, j'ai à cœur d'apprendre constamment, d'intégrer un milieu dynamique et en constante évolution. L'aspect créatif et collaboratif ainsi que la grande part de réflexion m'ont convaincu de retourner à ma première idée et j'ai intégré la formation de Développeur Web et Web Mobile.

Ce dossier met en lumière le site web que j'ai réalisé dans le cadre de ma formation. Il se distingue par son thème amusant et "ludique". Il représente un projet ambitieux qui m'a permis d'explorer et de mettre en pratique l'ensemble des compétences acquises tout au long de ma formation. Dans ce dossier je présenterai les étapes de sa création, de sa genèse aux technologies employées afin de réaliser une application front-end et back-end.

Table des matières

Remerciements	1
Introduction	2
Front-end.....	6
Maquetter une application.....	6
Réaliser une interface utilisateur web statique et adaptable	6
Développer une interface utilisateur web dynamique	6
Réaliser une interface utilisateur avec une solution de gestion de contenu ou e-commerce	6
Back-end.....	7
Créer une base de données.....	7
Développer les composants d'accès aux données	7
Développer la partie back-end d'une application web ou web mobile	7
Elaborer et mettre en œuvre des composants dans une application de gestion de contenu ou e-commerce	7
Introduction	8
Le contexte	8
Le sujet	8
Genèse	9
Un thème intéressant.....	9
Le client et le nom du site	9
Veille concurrentielle.....	9
Inspirations	10
Le cahier des charges	11
Logiciels et systèmes employés.....	11
Maquettage	11
Environnement de développement intégré	11
Modélisation conceptuelle de données	11
Versioning	11
Hébergement.....	11
La charte graphique.....	12
Le choix des couleurs.....	12

Le logo.....	13
Les polices d'écritures.....	14
Exigences du projet.....	14
La maquette	16
Le mapping	16
Zoning / Wireframe	16
Le mockup	16
Méthode AGILE	17
Langages employés	18
HTML5 & CSS3	18
Bootstrap 5	18
Javascript jQuery	18
PHP8.2	18
SQL.....	18
Symfony.....	19
MVC	20
Le modèle	20
La vue.....	21
Le contrôleur.....	21
Les principales dépendances utilisées	22
La base de données.....	23
MERISE	23
Doctrine.....	24
Création des entités.....	24
Front-End.....	26
Twig, templates et héritage.....	26
Navbar et sidebar	28
Stimulus	29
La sidebar.....	29
La page produit et la sélection des variants.....	33
L'affichage	33
Le contrôleur Stimulus (Code significatif).....	35
L'algorithme	37

L'ajout au panier	39
Back-end.....	43
Le contrôleur, le 'C' de 'MVC'	43
Rôle du contrôleur	43
Les routes.....	44
L'injection de dépendances	44
Les Repositories.....	45
Leur importance	45
Exemple de la fonction de recherche	46
Le contrôleur qui construit le fichier JSON des variants pour la page produit (Code significatif)	48
Les comptes utilisateur.....	51
Security Bundle et création de l'entité	51
Inscription, Connexion.....	51
Les rôles et les permissions	57
Le back-office.....	60
Introduction et rôle.....	60
Gérer le CRUD dans le back-office	61
Les tests.....	77
Pourquoi c'est important ?	77
Types de tests	77
Le Framework de tests PHPUnit.....	78
L'installation	78
Réalisation de test unitaires	78
Usage de l'anglais et résolution de problèmes	83
Veille informatique en matière de sécurité	85
Injections	85
Exposition de données sensibles.....	86
Server Side Request Forgery (SSRF)	86
Conclusion.....	88
Annexes	89

Front-end

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

Maquetter une application

Suite à l'élaboration d'un cahier des charges détaillant les fonctionnalités et les exigences attendues, j'ai réalisé le zoning ainsi que la maquette fonctionnelle, ergonomique et mobile-first de mon site en utilisant le logiciel Adobe XD.

Réaliser une interface utilisateur web statique et adaptable

Grâce au moteur de rendu du Framework¹ Symfony et des langages HTML, CSS et Javascript, j'ai développé la partie statique de mon site. L'utilisation de la bibliothèque Bootstrap et de son système de grille ainsi que de jQuery m'a permis de mettre en place une interface responsive et adaptable.

Développer une interface utilisateur web dynamique

Grâce à Javascript et plus particulièrement de l'AJAX (Asynchronous Javascript and XML), j'ai pu développer une application interactive dans laquelle les actions de l'utilisateur déclenchent des mises à jour de la page sans rechargement. L'expérience de l'utilisateur est alors plus fluide.

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

J'ai créé une interface de gestion de contenu (back-office) permettant aux utilisateurs de modifier ou de mettre à jour le contenu du site. Grâce à Symfony et à Bootstrap j'ai pu proposer une interface simple et intuitive.

¹ Ensemble d'outils et de composants permettant de simplifier le développement.

Back-end

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

Créer une base de données

J'ai réfléchi à la structure logique qu'allaient devoir prendre mes données. Grâce à différents modèles visuels j'ai pu définir l'organisation des tables, de leurs champs et des relations entre elles. J'ai ensuite implémenté ma logique dans une base de données MySQL grâce à l'ORM² Doctrine. L'application web open-source PhpMyAdmin m'a par ailleurs permis d'accéder directement à cette base de données.

Développer les composants d'accès aux données

J'ai développé des classes PHP permettant d'accéder à la base de données et d'effectuer les opérations de création, de lecture, de mise à jour et de suppression des données. Ces classes interagissent avec la base grâce à l'ORM Doctrine et à des requêtes SQL personnalisées.

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

J'ai conçu et mis en œuvre la logique de l'application côté serveur, la logique métier. Grâce à Symfony et à PHP j'ai mis en œuvre la création dynamique de pages web, la gestion des utilisateurs et d'une façon plus générale le code responsable de l'affichage, de la collecte et du traitement des données.

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

J'ai créé un back-office complet destiné aux administrateurs et permettant d'effectuer toutes les opérations de mise à jour du contenu du site. De la gestion de la bannière du site aux opérations liées au contenu du catalogues ou aux utilisateurs.

² Object-Relational Mapping, permet de lier des objets (ou entités) à des tables.

Introduction

Le contexte

Lors de ma formation au titre professionnel de Développeur web et web-mobile, j'ai été amené à réaliser un site web e-commerce dit « fil-rouge ».

Ce projet ayant l'avantage de valider l'intégralité des points acquis pendant la formation et devant être restitués lors de l'examen final, il est un support idéal de démonstration des compétences. En effet, la conduite de cet exercice tout au long de la formation a pour objectif la livraison d'une application web complète et sécurisée à un client fictif.

- Ce projet sera réalisé avec une approche mobile-first³ pour offrir une expérience optimale sur tous les appareils, qu'il s'agisse des smartphones, des tablettes ou bien des ordinateurs du bureau traditionnels. Cette approche est centrée sur l'ergonomie et l'utilisation intuitive de l'application tant par le client que par les utilisateurs finaux.
- Il permettra au client de gérer son activité avec une autonomie maximale en lui donnant le contrôle total de la gestion du contenu. Il pourra ainsi développer son activité en ligne et gérer sa boutique sans intervention extérieure.
- Il intégrera les mesures de sécurité adéquates afin de garantir l'intégrité de la base de données, de protéger les données personnelles des utilisateurs ainsi que de l'entreprise contre les potentielles attaques malveillantes.

En début de formation, mes camarades et moi-même avons été invités à choisir librement un thème autour duquel développer notre projet et le mien fût pour le moins atypique.

Le sujet ...

« Ces dernières années, le bien-être sexuel a été l'un des secteurs de vente ayant connu la plus forte croissance au niveau mondial. La croissance annuelle du secteur est incroyable, 6.7%, son chiffre d'affaires devrait atteindre les 122 milliards de dollars en 2026 ... »⁴

A la genèse du projet, l'humour et l'impulsion m'ont mené à spontanément répondre « sexshop » lorsque l'on m'a demandé sur quel sujet allait porter mon site. Dans les jours qui ont suivi, j'ai mené quelques recherches qui m'ont permis de comprendre l'importance véritable de ce secteur en matière de e-commerce et la transformation de ce qui était auparavant une niche commerciale en secteur d'activité perdant petit à petit ses tabous. Pour moi c'était clair, si il existe une demande, il existe une offre et s'il existe une offre, des clients ont besoin de développeurs web pour mettre en avant leur activité.

³ Concept de création de site web visant à focaliser le développement sur la version mobile pour l'adapter ensuite aux plus grands écrans.

⁴ Barrica, A. (2019, Mars). *How Sextech Pioneers Are Outsmarting Conservative Gatekeepers*. Forbes. Consulté en Octobre 2022, <https://www.forbes.com/sites/andreabarrica/2019/03/19/how-sextech-pioneers-are-outsmarting-conservative-gatekeepers/?sh=62b2bd0e96d1>

Genèse

Un thème intéressant

J'ai abordé ce sujet sans réelles connaissances du secteur au-delà de celles sur lesquelles je ne m'étendrai pas dans ce dossier. Au-delà de la curiosité suscitée par mes recherches sur ce marché en plein développement, j'ai pu me mettre dans la peau d'un développeur s'initiant à un projet nouveau et devant comprendre au mieux les besoins et les attentes des utilisateurs finaux ainsi que les spécificités du domaine d'activité concerné.

Le client et le nom du site

J'ai décidé de nommer mon site feuocu⁵. Ce projet ayant été initié par une blague, j'ai décidé de la pousser jusqu'au bout en choisissant un nom amusant, percutant et facile à retenir. C'est aussi un nom de marque facile à promouvoir et dont le nom est naturellement associé à l'activité commerciale et à l'objectif premier du site. Par ailleurs, ce nom de domaine est disponible avec toutes les extensions imaginables, ôtant dès le début un problème qui pourrait à l'avenir engendrer beaucoup de frustration.

www. feuocu	Rechercher
Extensions pour ce domaine	
VOIR LES 200 EXTENSIONS	
feuocu.fr	✓ Disponible 1 an 19,90 €/an Ajouter
feuocu.com	✓ Disponible 1 an 19,90 €/an Ajouter
feuocu.paris	✓ Disponible 1 an 49,90 €/an Ajouter
feuocu.net	✓ Disponible 1 an 19,90 €/an Ajouter
feuocu.eu	✓ Disponible 1 an 19,90 €/an Ajouter

Veille concurrentielle

Grâce aux enseignements reçus lors de mon cursus universitaire et d'une partie de mon ancienne activité professionnelle, mon premier réflexe a été d'effectuer une veille concurrentielle pour savoir ce qui se faisait chez les autres acteurs du secteur. Ce business étant en plein essor, le nombre d'acteurs est astronomique et cela m'a permis de déceler de nombreuses tendances. Celles-ci m'ont notamment aidées dans l'établissement du cahier des

⁵ Prononcer « feu au cul »

charges tant quant au contenu proposé, notamment en matière de charte graphique, qu'aux pratiques et technologies mises en avant.

Inspirations

« On dit merci qui ? »

- <https://www.dorcelstore.com/fr/>
- <https://www.jacqueetmichelstore.com/>
- <https://www.espaceplaisir.fr/>
- <https://www.lelo.com/fr>
- <https://www.adameteve.fr/>
- <https://www.fleshlight.eu/?locale=fr>
- <https://www.loveandvibes.fr/>
- <https://www.avenue-privee.com/29-sex-toys>
- etc.

Cette liste non exhaustive présente quelques-unes des références que j'ai utilisées lorsque j'ai commencé à réfléchir à mon site. Il s'agit tout autant de sites de fournisseurs que de sites de revendeurs tels que le client fictif pour lequel je vais réaliser mon application. Ces références m'ont rapidement donné une idée de ce à quoi allait ressembler la maquette plus tard, ce qui m'a permis d'apprivoiser l'intégralité du problème en amont de la réalisation du cahier des charges.

Le cahier des charges

Logiciels et systèmes employés

Maquettage

Adobe XD est un logiciel de la suite Adobe qui permet de concevoir l'interface et l'expérience utilisateur d'une application. Du wireframe⁶ à la maquette interactive, le logiciel offre la possibilité de créer, tôt dans le processus de conception, un aperçu au plus proche du résultat final. C'est non seulement bénéfique pour le client qui peut facilement se projeter avec un visuel concret et interactif mais cela permet aussi un gain de temps non négligeable lors de la phase de développement.

Environnement de développement intégré

Ayant bénéficié d'une licence PhpStorm destinée à l'apprentissage, j'ai utilisé l'IDE⁷ de JetBrains spécialement destiné aux développeurs PHP. Son auto-complétion intelligente permet d'accélérer le codage et de réduire la probabilité de commettre des erreurs d'inattention pouvant limiter fortement la productivité. Il propose une option de refactorisation qui permet de renommer facilement des variables ou de modifier rapidement la structure du code pour le rendre plus lisible et donc maintenable. Par ailleurs, il est compatible avec les outils de versioning tels que Git.

Modélisation conceptuelle de données

J'ai utilisé le logiciel gratuit Looping pour la modélisation conceptuelle de données qui est l'étape de la conception de la base de données permettant de représenter les données sans se soucier de la façon dont les données seront stockées. Elle présente les objets et les objets de la base, leurs attributs et les relations entre elles.

Versioning

Git est le système que j'ai utilisé pour gérer le versioning, il permet de conserver un historique des modifications apportées au code et de revenir à une version antérieure si besoin. Les versions seront hébergées sur GitLab qui à la différence de Github est open-source⁸.

Hébergement

Le site pourra être hébergé auprès de l'un des leaders du marché tels que Ionos ou OVH qui proposent des solutions d'hébergement fiables, sécurisées et avec un support technique réactif ou accessible. Avec le choix et la réservation du nom de domaine, la sélection d'une solution d'hébergement est cruciale dans le processus de création d'une application web car la fiabilité et la qualité de l'hébergement auront une répercussion sur la qualité du produit fini et donc de la satisfaction du client. Il convient donc de régler ces aspects au plus tôt afin d'éviter les mauvaises surprises plus tard lors de l'avancée du développement.

⁶ Maquette structurelle de l'application.

⁷ Environnement de développement intégré, ensemble d'outils qui aident à développer plus efficacement.

⁸ Utilisable, modifiable et diffusable librement par le public.

La charte graphique

Le choix des couleurs

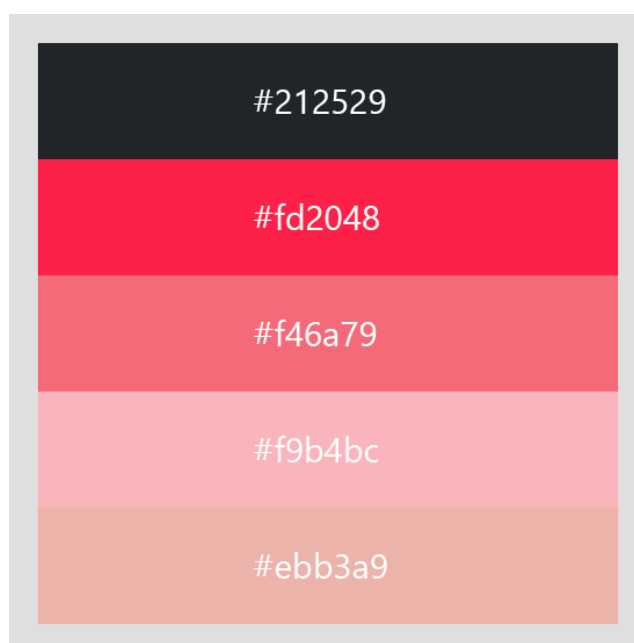
S'agissant du développement d'un sexshop en ligne, il est impossible de faire l'impasse sur une couleur essentielle : le rose ! Couleur représentative de l'amour par excellence, on la retrouve partout dans l'industrie des sexshops.

« Quand il me prend dans ses bras, qu'il me parle tout bas, je vois la vie en rose... »⁹

C'est une couleur associée à l'amour et son nom est celui d'une fleur qui l'évoque elle aussi. Dans le langage des fleurs, offrir une rose de couleur rose est un symbole de tendresse et d'attachement. Pour de nombreuses cultures, il est de coutume d'associer cette couleur à la douceur, au romantisme, à la passion, la séduction et à la beauté féminine. Etrangement ce n'est que depuis les années 1940 et la célèbre chanson d'Edith Piaf que cette couleur est réellement associée à la féminité.

Ces thèmes et émotions sont sensiblement les mêmes que celles que la psychologie prête à cette couleur. C'est une couleur dont les déclinaisons pâles évoquent l'affection et le calme quand ses teintes plus éclatantes tendent plutôt à instaurer une ambiance dynamique et passionnelle, à la limite de l'excitation évoquées par le rouge. On notera que la couleur évoque aussi les bonbons, l'enfance et l'innocence.

Souvent utilisée pour cibler la gent féminine dans le marketing, cette couleur est particulièrement importante dans l'industrie des sextoys, c'est pourquoi il est presque impossible de faire l'impasse dessus. Elle servira donc de base de travail au développement du visuel du site.



⁹ Edith Piaf. (1946). *La vie en rose*

Le gris foncé #212529 permettra d'instaurer une forte présence visuelle sur le site sans l'effet cassant qu'aurait pu apporter du noir pur. Il se mariera plus facilement avec les couleurs tout en apportant une certaine élégance que n'aurait pas le noir.

#fd2048 est la première couleur que j'ai sélectionnée pour le nuancier, c'est de celle-ci que j'ai déduit toutes les autres. Je cherchais un rose évoquant la passion, créant une forte impression tout en étant moins agressif que le rouge vif. Ce sera la base de mon identité visuelle et cette couleur sera utilisée pour attirer l'attention de l'utilisateur. Par ailleurs elle sera au cœur du logo.

Les autres déclinaisons seront destinées à harmoniser l'interface en créant du contraste tout en restant dans cet univers à dominante rose. Elles seront utilisées pour colorer des polices, adoucir l'écran et générer des effets visuels notamment lors d'interaction avec des boutons ou des liens.

Le logo

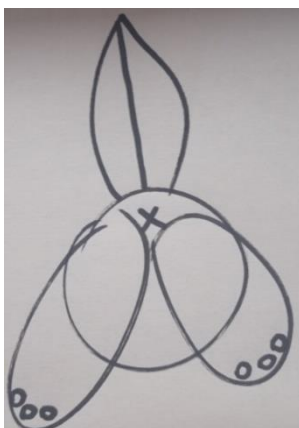
Lorsque j'ai choisi le thème, j'ai tout de suite trouvé l'inspiration concernant le logo.

« Le Rabbit, icône de la pop CULTure »¹⁰

Difficile aujourd'hui de parler de sextoys sans penser à cet animal incontournable. Parfois considéré comme un porte-bonheur, sa réputation de symbole sexuel le précède. Qui n'a jamais vu le célèbre logo du magazine Playboy¹¹ ou entendu l'expression « chaud comme un lapin » en référence à ses capacités de reproductions hors-normes ? L'animal est associé à de nombreux fantasmes et depuis les années 80 il est surtout le nom donné à un célèbre sextoy destiné aux femmes dont la forme rappelle celle des oreilles du petit mammifère.



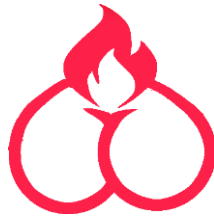
Postérieur de l'un de nos amis aux grandes oreilles.



La photographie ci-dessus est la première image que j'ai eu en tête lorsque j'ai commencé à donner forme au projet. Dessinateur à mes heures, j'ai tenté à de nombreuses reprises d'essayer de faire fonctionner cette idée en vain. Je souhaitais un logo minimaliste et je n'arrivais tout simplement pas à garder un niveau de détails suffisant pour que l'on comprenne au premier coup d'œil de quoi il s'agit sans ajouter de complexité. C'est ainsi que j'ai changé d'approche tout en gardant une partie de cette inspiration dans le résultat final. M'étant imposé le nom feuocu, j'ai décidé de garder la position du « cul » de notre ami le lapin et j'ai pensé au fait qu'à défaut de représenter ses oreilles, je pourrais peut-être représenter sa queue, lui donner un aspect rappelant les flammes et j'y étais, mon logo était né.

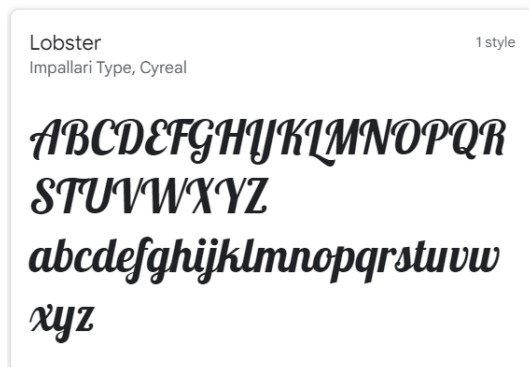
¹⁰ Pizzuto, A. *Le rabbit, icône de la pop CULTure*. Marie claire. Consulté en Avril 2023, <https://www.marieclaire.fr/histoire-rabbit-sextoy,1332014.asp>

¹¹ Magazine de presse masculine fondé en 1953 par Hugh Hefner et édité dans le monde entier.



Grâce à l'outil de tracé vectoriel de Adobe Photoshop, j'ai réalisé un premier dessin qui correspondait plus ou moins à mon idée de base, je me suis débarrassé des pattes du lapin, représenté sa queue sous l'aspect d'une flamme ardente et j'avais exactement ce qu'il me fallait, ni plus, ni moins le logo de feuocu serait donc un « cul » en feu !

Les polices d'écritures



Lobster est une police cursive élégante qui s'accordera parfaitement au thème du site en apportant une part de glamour aux titres et aux éléments importants. Raleway quant à elle est une police sans serif élégante qui sera utilisée pour le corps du site.

Exigences du projet

Fonctionnalités front-end

En tant qu'application mobile-first, l'accent devra être mis sur l'ergonomie. La navigation devra être claire et intuitive pour l'utilisateur final. Au-delà de l'aspect purement graphique de l'interface (UI), l'utilisateur devra retrouver les éléments clés là où il a l'habitude de les retrouver. Cela permettra d'améliorer son expérience d'utilisateur (UX).

Le site devra proposer une barre de recherche afin de pouvoir retrouver des produits grâce à leur nom ou leur marque.

Les fiches de produits devront être simples et visuellement attractives, comprendre un carrousel dynamique si de multiples photographies sont attachées au produit, présenter de façon claire les différentes variations du produit en question si elles existent.

Le panier des achats devra lui aussi être facile à utiliser, il devra afficher de façon claire et détaillée toutes les informations concernant les produits qu'il contient, prix hors taxe, prix TTC, montant total des taxes, frais de port, etc. Par ailleurs, l'utilisateur devra être en mesure de modifier le contenu de son panier depuis l'interface dédiée. Il pourra changer la quantité de produits et supprimer des éléments intuitivement.

Une interface dédiée au changement des informations personnelles du compte de l'utilisateur est indispensable. Il pourra y modifier son adresse de livraison, son adresse email, son mot de passe etc.

Fonctionnalités back-end

L'administrateur du site pourra apporter des modifications au contenu du site en modifiant à sa guise la bannière présente sur la page d'accueil du site web.

Véritable centre de gestion, l'application back-end permettra aux utilisateurs autorisés de gérer en temps réel tout le contenu de la boutique en ajoutant, supprimant ou modifiant des :

- Catégories de produits
- Marques de produits
- Produits ainsi que toutes les variations possibles pour ces produits, taille, couleur, etc.
- Comptes utilisateurs

Par ailleurs l'administrateur pourra gérer les comptes des utilisateurs et leur donner des autorisations particulières si la situation le demande. L'administrateur principal pourra par exemple accorder des droits d'accès au panel d'administration à d'autres utilisateurs dans le but de leur déléguer des tâches.

Sécurité

Le site devra implémenter un système de gestion des accès afin d'empêcher les utilisateurs non autorisés d'accéder à des informations sensibles. Les informations sensibles telles que les mots de passe des utilisateurs devront être cryptées afin d'empêcher un utilisateur malveillant d'y accéder au cas où l'accès à la base de données aurait été violé. On prendra par ailleurs le soin de protéger la base de données contre les injections qui peuvent compromettre la sécurité du site.

La maquette

Le mapping¹²

Le mapping est une étape qui permet de planifier l'organisation du site web. L'objectif est de créer une architecture permettant de visualiser l'arborescence du site, d'identifier les pages ainsi que les liens entre celles-ci. Cette étape permet par ailleurs de découper le contenu en zones accessibles à différents types d'utilisateurs.

Pour la réalisation des diagrammes, j'ai utilisé l'outil open-source draw.io. Le résultat est visible en annexes :

Annexe 1 - Mapping du site

Annexe 2 - Mapping du back-office

Zoning / Wireframe

L'ensemble de la maquette du site web a été réalisée en utilisant le logiciel de conception Adobe XD. Le zoning est l'étape préalable à la conception de la maquette finale, elle consiste à structurer l'interface visualisant l'emplacement dédié à chaque bloc (ou élément) de la page. Je les ai organisés de manière logique en pensant à la fois à l'aspect UI (relatif à l'interface) et UX (relatif à l'expérience de l'utilisateur).

Annexe 3 - Zoning et wireframe de la page d'accueil

Annexe 4 - Zoning et wireframe d'une page catégorie

Annexe 5 - Zoning et wireframe du login

Le mockup¹³

Suite à la réalisation du wireframe du site, j'ai pu développer une maquette fonctionnelle grâce au logiciel Adobe XD. Le mockup est très proche du résultat final attendu. Elle présente l'avantage d'intégrer des couleurs, des images ainsi que les éléments visuels du site. Il permet au client de se projeter avec une maquette interactive et au travers de laquelle on peut naviguer comme si l'on utilisait le véritable site. Les liens entre les pages sont actifs et les animations déclenchées par les interactions de l'utilisateur le sont aussi.

Annexe 6 - Maquette page d'accueil et catégories mobile

Annexe 7 - Maquette Page d'accueil et catégories desktop

¹² Mappage ou cartographie du site web

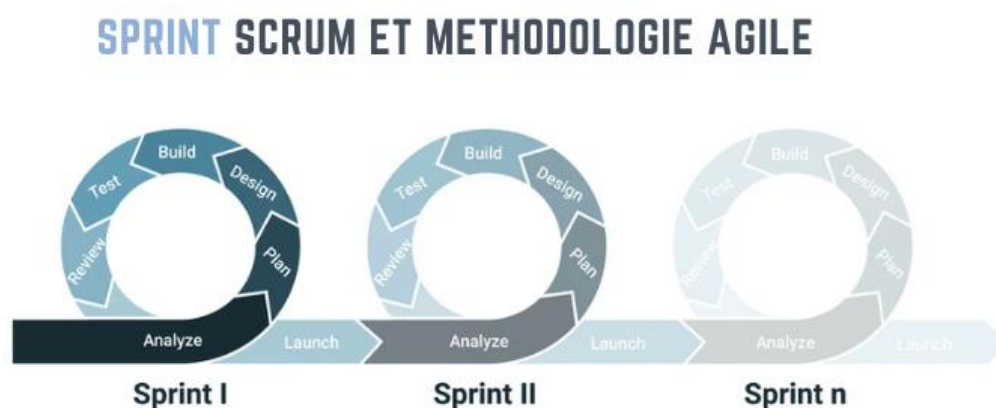
¹³ Maquette fonctionnelle

Méthode AGILE

Ce concept repose sur un postulat simple. Lors du développement d'un projet informatique, la planification totale en amont est une perte de temps. En effet, il est rare que la situation se déroule exactement comme prévu et des aléas viennent régulièrement bouleverser la planification initiale.

La méthodologie AGILE préconise donc une approche différente. Elle recommande de fixer des objectifs à court terme et de diviser le projet en sous-projets ayant chacun leurs objectifs et échéances. La finalisation d'une étape entraînant ainsi le démarrage de la suivante. C'est une méthode qui permet de faire face aux imprévus en laissant plus de place aux imprévus et plus de flexibilité pour faire face aux changements. Il s'agit ici de mettre le client au cœur du projet en instaurant une relation privilégiée entre celui-ci et l'équipe responsable du développement. Le dialogue et la réactivité sont donc indispensables pour proposer des ajustements en adéquations avec le changement potentiel de ses attentes.

SCRUM est l'approche AGILE la plus utilisée. Elle consiste en l'organisation de cycles de travail courts appelés sprints qui visent chacun à l'obtention d'un objectif. L'équipe responsable du développement communique régulièrement sur l'avancée du projet et à chaque fin de sprint, elle révisé le travail accompli puis planifie le sprint suivant.



Langages employés

HTML5 & CSS3

Ce sont les versions les plus récentes des langages utilisés pour la création de pages web. HTML5 permet de structurer le contenu tandis que CSS3 est utilisé pour définir le style et la mise en forme. Les deux technologies sont interconnectées.

Bootstrap 5

C'est un Framework CSS open-source qui fournit des classes prédéfinies pour faciliter le développement grâce à des composants prêts à l'emploi et qui offrent une cohérence visuelle entre les éléments. Elle permet de substituer une partie importante du CSS « pur ».

De plus elle est compatible avec tous les navigateurs et intègre notamment un concept de Grid pour faciliter la responsivité.

Javascript jQuery

Javascript est le langage qui permet de rendre les pages web interactives et dynamiques. Il est pris en charge par tous les navigateurs et permet d'ajouter une couche de logique sur la partie du site accessible aux utilisateurs.

jQuery est une bibliothèque Javascript qui permet entre autres de manipuler le DOM grâce à une syntaxe simplifiée. J'ai utilisé ce langage principalement pour le contrôle de composants externes que j'ai ajouté à mon application.

PHP8.2

C'est le langage de scripts côté serveur le plus utilisé pour le développement web. Il permet de créer des sites dynamiques en se connectant à une base de données, de gérer des fichiers, des formulaires ou encore des sessions d'utilisateur. J'utilise donc le Framework Symfony qui fournit des bibliothèques pour faciliter le développement de mon application.

SQL

C'est le langage utilisé pour manipuler et gérer les données stockées en base de données.

Symfony

Symfony est un Framework open-source utilisant une architecture MVC¹⁴ sur laquelle je reviendrai plus tard. Il simplifie énormément le travail du développeur en intégrant un système de gestion des routes, un moteur de rendu intégré visant à générer des templates. Il embarque un CLI¹⁵ un système de traitement des formulaires, s'occupe d'une grande partie de la sécurité du projet et intègre bien d'autres fonctionnalités. Symfony repose sur l'injection de dépendances afin d'assurer la modularité des composants et utilise le gestionnaire de dépendances libre Composer pour gérer cet aspect.

L'injection de dépendances consiste à passer en paramètre les objets dont une classe dépend plutôt que de les instancier directement. Elle permet de séparer la création et la gestion des dépendances d'une classe de son code métier, ce qui permet leur remplacement ou modification ainsi que leur maintenabilité et test sans affecter le code de la classe en question. Par ailleurs, la complexité du code est réduite car on évite la création de classes imbriquées.

Sans injection:

```
use App\MaDependance;

class MaClasse
{
    private $maDependance;

    public function __construct()
    {
        $this->maDependance = new MaDependance();
    }
}
```

Avec injection:

```
use App\MaDependance;

class MaClasse
{
    private $maDependance;

    public function __construct(MaDependance $maDependance)
    {
        $this->maDependance = $maDependance;
    }
}
```

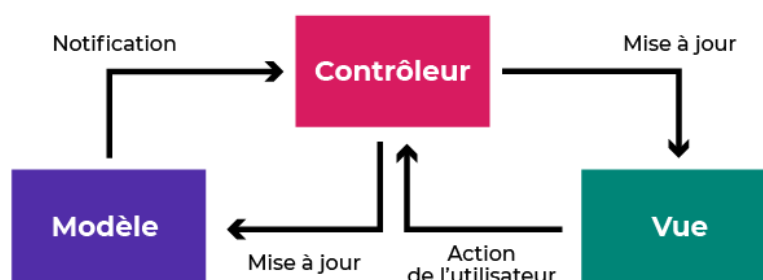
J'ai choisi de développer mon projet en utilisant la version la plus récente de Symfony, 6.2, les versions antérieures n'étant plus maintenues actuellement. Durant toute la réalisation de mon projet, il a été essentiel pour moi de m'appuyer sur la documentation en ligne du Framework afin d'en comprendre assez pour mener à bien mon projet.

¹⁴ Modèle-Vue-Contrôleur

¹⁵ Interface de Ligne de Commandes

MVC

Le Modèle-Vue-Contrôleur est un design pattern (ou motif d'architecture logicielle) conçu en 1978 et qui permet de séparer le code d'une application en 3 parties ayant chacune leurs responsabilités propres.



Le modèle

C'est lui qui contient la logique métier de l'application. Il est responsable de la gestion et de la persistance des données, comprendre qu'il doit être en mesure de les lire, de les valider et de les enregistrer. Dans Symfony et dans la plupart des Frameworks utilisant le design MVC, on manipulera ces données représentées sous forme d'objets PHP.

- Des classes dédiées nommées Entités permettront grâce aux annotations Doctrine de définir des propriétés ainsi que des relations avec d'autres classes. Elles contiendront également des annotations relatives à la validation de ces propriétés.
- Chaque entité possède son Repository qui a pour charge d'interagir directement avec la base de données. Ils effectuent les opérations de base dites CRUD (Create, Read, Update, Delete)¹⁶. On les utilisera aussi pour effectuer toutes les requêtes plus complexes à la base en utilisant le QueryBuilder¹⁷ embarqué par Doctrine ou des requêtes SQL brutes.
- Les services quant à eux sont des classes qui auront pour responsabilité la réalisation de tâches spécifiques, dans mon cas, la création de nouvelles entités, leurs mises à jour, l'upload d'images etc.

¹⁶ Créer, Lire, Mettre à jour, Supprimer

¹⁷ Concepteur de requêtes

La vue

Elle comprend toute la partie de l'application relative à l'affichage. C'est l'interface graphique de l'application qui permet à l'utilisateur d'interagir avec elle. La vue comprend toute la logique lui permettant d'afficher les informations provenant du modèle. Elle met en œuvre les langages HTML, CSS et Javascript. Dans le cas de Symfony, j'utiliserai des templates développés grâce au moteur Twig qui est le moteur de templates par défaut de Symfony.

Le contrôleur

C'est lui qui est chargé de traiter les interactions de l'utilisateur. C'est l'intermédiaire entre le modèle et la vue. Il gère les requêtes entrantes, fait appel aux services et sélectionne la vue appropriée à l'affichage des données demandées par l'utilisateur afin de les renvoyer à son client. Ils peuvent recevoir des paramètres et renvoyer des réponses sous diverses formes, pages HTML, JSON etc.

Les principales dépendances utilisées

Toutes les dépendances seront gérées depuis un fichier `composer.json` placé à la racine du projet.

Annexe 8 - Extrait du fichier `composer.json` de mon projet

```
"doctrine/orm": "^2.14",
```

C'est un ORM (Object-Relational Mapping) pour le langage PHP. Il permet de mapper¹⁸ les entités de la base de données à des objets PHP. Doctrine permet de simplifier grandement la création et la gestion de la base de données et propose des fonctionnalités telles que la migration (comprendre la mise à jour effective de la base en fonction des modifications apportées au code), un queryBuilder qui est un système de requêtes personnalisées.

```
"doctrine/doctrine-fixtures-bundle": "^3.4",
```

Fixtures-bundle est une dépendance qui permet de créer des données d'exemple afin de peupler la base après la création de celle-ci. Ces données permettent ensuite de faciliter le développement en testant en temps réel l'application.

```
"symfony/webpack-encore-bundle": "^1.16",
```

C'est un outil open-source qui permet de gérer et compiler les ressources de l'application (assets). Ceux-ci comprennent les fichiers CSS, les fichiers Javascript, les images, etc. de l'application. Il va me permettre de transformer les fichiers sources en bundles plus optimisés pour le navigateur de l'utilisateur.

```
"symfony/security-bundle": "6.2.*",
```

C'est un composant qui permet de gérer l'authentification des utilisateurs, leurs autorisations et la sécurité dans l'application.

```
"twig/twig": "^2.12|^3.0"
```

C'est le moteur de templates par défaut du Framework. Il va me permettre de concevoir l'interface graphique, d'y intégrer la logique qui conditionne l'affichage des données ainsi que la saisie des entrées par les utilisateurs. Il utilise un héritage de templates, définit des blocs, permet l'utilisation de variables en plus d'offrir un degré de sécurité intégré contre les attaques XSS¹⁹ en échappant automatiquement les données potentiellement compromises avant de les afficher sur la page.

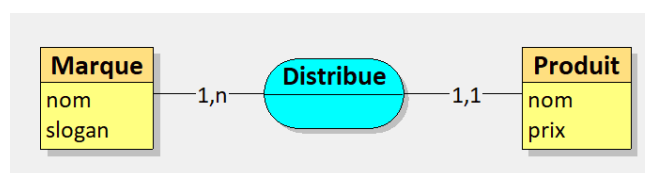
¹⁸ Etablir une correspondance

¹⁹ Cross-Site Scripting

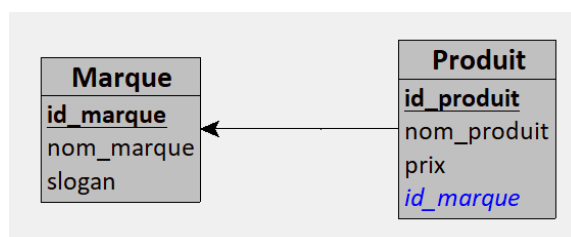
La base de données

MERISE

Merise est une méthode d'analyse et de conception des systèmes d'informations qui propose une approche globale et structurée. La modélisation des données est un élément clé de cette méthode et elle implique la création d'un modèle conceptuel de données (MCD) pour décrire les entités et les relations entre ces entités de manière abstraite. On établit un dictionnaire des données qui seront essentielles pour manipuler ces entités et on décrit donc nos entités indépendamment de tout système d'information. L'association est caractérisée par des cardinalités. Une marque distribue au minimum 1 produit et la relation est sans limites. De l'autre côté, un même produit est distribué par une seule marque.



A partir du MCD on peut générer un MLD (modèle logique de données) dans lequel les entités deviennent des tables. Dans le cas de la relation entre le produit et la marque, l'association disparaît pour devenir une clé étrangère dans la table produit qui pointe vers la table marque.



Lorsque le MLD est prêt, on peut passer à la dernière étape avant l'implantation de la base de données dans le SGBD²⁰, le MPD (Modèle Physique de Données). Cette dernière étape va permettre de décrire le système d'informations en prenant en compte les aspects techniques spécifiques au SGBD utilisé, on va ainsi typer les données.

```
CREATE TABLE Marque (
    idMarque INT PRIMARY KEY,
    nomMarque VARCHAR(50),
    slogan VARCHAR(255)
);

CREATE TABLE Produit (
    idProduit INT PRIMARY KEY,
    nomProduit VARCHAR(50),
    prix DECIMAL(5,2)
);
```

Annexe 9 - Diagramme UML de la base de données

²⁰ Système de Gestion de Bases de Données

Doctrine

Comme expliqué précédemment, j'utilise le Framework Symfony et plus particulièrement Doctrine pour créer ma base de données.

J'ai commencé par lier mon application Symfony à la base de données grâce au fichier `.env` placé à la racine du projet.

```
DATABASE_URL="mysql://root:@localhost:3306/feuocu"
```

Cette ligne contient les codes d'accès à la base de données, dans mon cas avec un utilisateur 'root' et sans mot de passe. Le fichier est placé à la racine du projet et en dehors du dossier public.

Une fois ma base de données liée à mon application, je peux exécuter la commande suivante qui génère la requête SQL de création d'une nouvelle base nommée feuocu.

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

L'ORM Doctrine permet donc de faire le lien entre des objets du code PHP et les tables dans la base de données. Il utilise l'extension PHP PDO (PHP Data Objects) qui permet d'accéder aux données de la base en les interfaçant directement avec des objets. Un autre avantage de PDO est qu'il offre une couche d'abstraction qui protège des injections SQL en utilisant des requêtes paramétrées sur lesquelles je reviendrai dans la partie sécurité.

Création des entités

Pour créer les tables avec Doctrine, j'ai commencé par créer des classes PHP d'entités correspondant à la table et j'ai mappé cette entité grâce à des annotations.

```
php bin/console make:entity Product
```

La commande `make:entity` du bundle Maker de Symfony m'a permis de gagner du temps dans la création de mes entités en m'assistant dans la création des propriétés, des relations entre les entités et la génération de getters et setters associés. Par ailleurs, le code de l'entité permet d'ajouter des contraintes sur les propriétés qui permettront d'ajouter un degré de validation côté serveur lorsque des données seront ajoutées à la base.

Annexe 10 - Extrait du code de l'entité Produit

Lorsque l'entité est créée, je peux exécuter la commande suivante pour créer la table en tenant compte de toutes les annotations doctrines indiquées.

```
php bin/console doctrine:schema:update --force
```

Annexe 11 - Structure de la table Product

J'ai créé toutes les entités nécessaires à mon projet puis j'ai créé les tables correspondantes toujours en suivant la même méthode.

Suite à cela, j'ai ajouté quelques fixtures dans ma base à l'aide du package Doctrine-Fixtures-Bundle

```
"doctrine/doctrine-fixtures-bundle": "^3.4",
```

Ces fixtures sont des données de test qui m'ont aidé lors du développement. Grâce aux Setters des entités qui sont des méthodes permettant de définir la valeur de l'attribut d'une classe, j'ai créé des objets fictifs dans une classe dédiée (marques, produits, images, utilisateurs) que j'ai ensuite simplement persistés dans la base de données grâce à la commande :

```
php bin/console doctrine:fixtures:load
```

Annexe 12 - Exemple de fixture permettant d'ajouter un utilisateur disposant des droits d'administrateur

Cela me permet de peupler ma base de données avec des données initiales qui me permettent non seulement d'avoir un aperçu du rendu en temps réel mais aussi de tester des comportements et des fonctionnalités.

Une fois toutes mes entités et les relations entre elles créées, j'ai pu commencer à développer la partie front de mon application en utilisant la maquette réalisée précédemment.

Front-End

Twig, templates et héritage

Grâce à son moteur de templates Twig et plus particulièrement la fonction `include`, il est possible de concevoir le front-end de l'application comme une imbrication de templates. Cela permet de structurer le code de manière modulaire en créant des templates indépendants et réutilisables. La fonction `'include'` de Twig permet d'imbriquer les templates les uns dans les autres.

```
{% include 'components/header/_header.html.twig' %}
```

Il devient alors possible de créer un template de base contenant la structure globale et les éléments communs à chaque page puis d'inclure des composants spécifiques selon les besoins de chaque page. Dans ce fichier de base, j'inclue toutes les feuilles de style et scripts nécessaires aux pages qui en hériteront (dans mon cas, principalement des liens vers des CDN puisque le webpack Encore s'occupe de la compilation de mes assets). Cela me permet de simplifier le code, de faciliter sa lecture et son entretien.

Annexe 12 - Exemple de fixture permettant d'ajouter un utilisateur

```
<?php
namespace App\DataFixtures;

use App\Entity\User;
use Doctrine\Bundle\FixturesBundle\Fixture;
use Doctrine\Persistence\ObjectManager;

class AppFixtures extends Fixture
{
    public function load(ObjectManager $manager): void
    {
        // USERS
        $user1 = new User();
        $user1->setEmail('foucaut.morgan@gmail.com')
            ->setPassword('$2a$12$WmzLEY/fMamOUP1cW2w5OeZh8gWtqpx6MTDK1QJ2Bnqzm0joW0LXy'
        )
            ->setRoles(['ROLE_SUPER_ADMIN'])
            ->setFirstName('Morgan')
            ->setLastName('Foucaut')
            ->setBirth(new \DateTime('1990-01-01'))
            ->setAddress('18 rue Paul Vaillant Couturier')
            ->setZipcode('57300')
            ->setCity('Hagondange')
            ->setPhone('0767603690')
            ->setCreatedAt(new \DateTimeImmutable())
            ->setIsValidated(true)
            ->setValidationKey(12);
        $manager->persist($user1);

        $manager->flush();
    }
}
```

Il est aussi possible de créer une hiérarchie d'héritage entre templates en définissant des blocs de contenu dans les templates parents.

```
{% block body %}{% endblock %}
```

Ces blocs seront définis dans des templates en utilisant les mêmes balises pour définir le contenu qui remplacera le bloc du parent.

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

{% block title %}feuOcu !{% endblock %}

{% block body %}
    {% if hero %}
        {% include 'components/_hero.html.twig' with {'hero': hero} %}
    {% endif %}
    {% include 'components/_infos.html.twig' %}
{% endblock %}
```

Ci-dessus le code du fichier home.html.twig. C'est cette page qui est responsable du contenu de la page d'accueil du site. La page hérite de base.html.twig, elle redéfinit le titre de la page grâce à la balise {% block title %} et redéfinit le contenu du bloc body de son parent grâce à {% block body %}.

En tant que moteur de templates intégré dans le modèle MVC (Modèle-Vue-Contrôleur), Twig permet de manipuler et d'afficher des données en provenance du contrôleur. Il offre la possibilité d'afficher ces données de manière conditionnelle grâce à des instructions simples telles que if, else, for, foreach, etc.

Dans l'exemple précédent, j'intègre 2 composants personnalisés à ma page d'accueil :

- Un Hero _hero.html.twig qui aura besoin des données contenues dans la variable 'hero' et transmises par le contrôleur.
- Un bloc _infos.html.twig qui contenant du code HTML simple affichant des informations et des icônes.

Annexe 13 - base.html.twig

Annexe 14 – Composants de la page d'accueil

Navbar et sidebar



Aperçu de la barre de navigation sur desktop

La barre de navigation située sur toutes les pages est un composant Twig indépendant.

```
<nav class="navbar navbar-expand-md" id="nav">
  <div class="container-fluid">
    <div class="collapse navbar-collapse" id="navbarNav">
      <ul class="navbar-nav">
        <li class="nav-item">
          <a class="nav-link" href="#"
            data-action="click->toggle-sidebar-categories#toggle">
            <i class="bi bi-list fa-lg me-3 fw-bolder"></i>Tous les
produits
          </a>
        </li>
        {% for category in categories %}
          <li class="nav-item">
            <a class="nav-link" href="{{ path('show_category',
              {'slug': category.slug}) }}">{{ category.name }}</a>
          </li>
        {% endfor %}
      </ul>
    </div>
  </div>
</nav>
```

J'ai utilisé la librairie Bootstrap pour créer ce composant qui aura la particularité de n'être visible que sur les écrans dont la largeur est supérieure à 768 pixels grâce à la classe :

```
navbar-expand-md
```

- La liste contenant les éléments de la barre de navigation contient un élément personnalisé qui contient une icône de type 'burger' cliquable.

```
data-action="click->toggle-sidebar-categories#toggle"
```

Cet attribut de l'élément HTML indique qu'un comportement spécifique sera déclenché lorsque l'élément sera cliqué. Plus spécifiquement, le navigateur de l'utilisateur fera appel à un contrôleur du Framework Javascript que j'utilise pour rendre mon application dynamique : Stimulus. J'y reviendrai dans la partie suivante.

- La seconde partie de la barre de navigation comprend une génération dynamique des catégories de produits de ma base de données. Le contrôleur chargé d'afficher la barre de navigation lui transmet alors des données reçues d'un modèle et Twig se charge de générer dynamiquement une barre indépendamment du nombre de catégories existantes sur le site, tout est automatique !

Stimulus

Stimulus est un Framework Javascript très léger qui présente l'avantage de décomplexifier le code javascript nécessaire à la dynamisation d'une page HTML. Il permet plus particulièrement de manipuler avec aisance des composants existants en utilisant des attributs HTML spécifiques :

- `data-controller` : définit le contrôleur stimulus en charge d'un élément du DOM
- `data-target` : permet de donner à un nom à un élément important pour le manipuler plus facilement en l'appelant par son nom
- `data-action` : définit des événements et les actions appropriées associées à ces éléments

Lorsque ces attributs personnalisés sont définis, on peut alors créer des contrôleurs Stimulus qui sont des classes Javascript contenant généralement une méthode **connect()** qui sera appelée lorsque Stimulus détectera un élément du DOM associé au contrôleur. Lorsque la page se charge et que Stimulus détecte un élément disposant d'un attribut `data-controller`, il instancie automatiquement le contrôleur adéquat et appelle sa méthode **connect**. D'autres méthodes peuvent être ajoutées selon les besoins.

Si je reviens à mon exemple précédent, l'élément de la barre de navigation contenant le burger cliquable détenait cet attribut :

```
data-action="click->toggle-sidebar-categories#toggle"
```

Cela signifie que lorsque l'élément sera la cible d'un événement 'clic', le navigateur appellera la méthode 'toggle' du contrôleur stimulus 'toggle-sidebar-categories' dont le rôle sera de gérer l'affichage d'une barre de menu latérale contenant des liens vers des catégories et leurs sous-catégories.

J'ai défini que l'élément du DOM le plus 'élevé' dans la hiérarchie et allant être amené à interagir avec la sidebar était le body. J'ai donc décidé de donner cet attribut au body dans mon fichier `base.html.twig`.

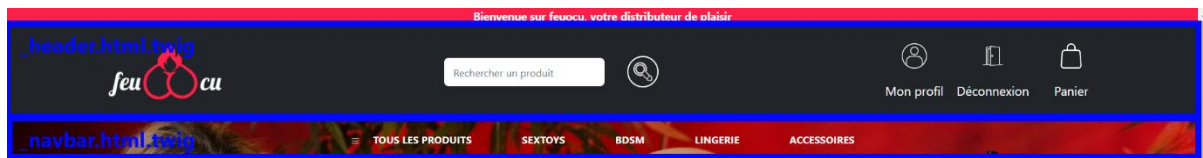
```
<body data-controller="toggle-sidebar-categories">
```

Ainsi, lorsque le navigateur charge la page, il décèle immédiatement que toute la page va être sujette à des interactions qui vont nécessiter l'utilisation du contrôleur `toggle-sidebar-categories`.

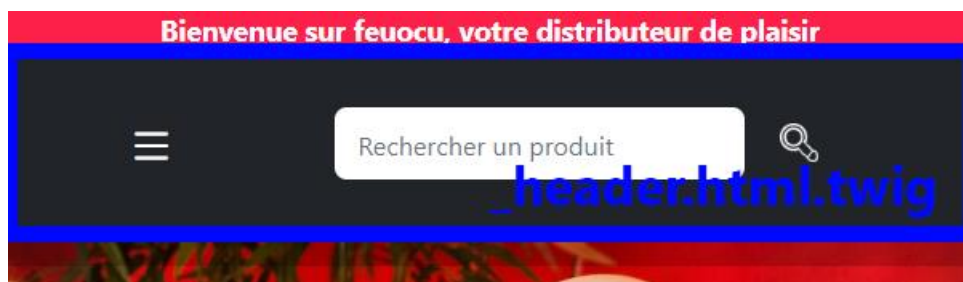
La sidebar

Grâce aux classes de la librairie Bootstrap, mon site est responsive et adaptable. Dans le cas de mon interface utilisateur et dans un souci d'ergonomie, le haut de ma page se divise en deux parties, un header et une barre de navigation comme vu précédemment. Le header contenant beaucoup de sous-éléments, notamment une barre de recherche, un accès au compte, une icône pour se connecter ou se déconnecter ainsi qu'un accès au panier, il devient difficile d'afficher le tout en mode mobile. Il en va de même pour la barre de navigation horizontale sous-jacente.

C'est pourquoi grâce à Bootstrap, je fais disparaître la barre de navigation ainsi qu'une grande partie du contenu du header lorsque l'écran est de petite ou moyenne taille et je les remplace par une icône burger qui va permettre d'afficher toutes les informations manquantes dans la barre latérale dont j'ai parlé précédemment.

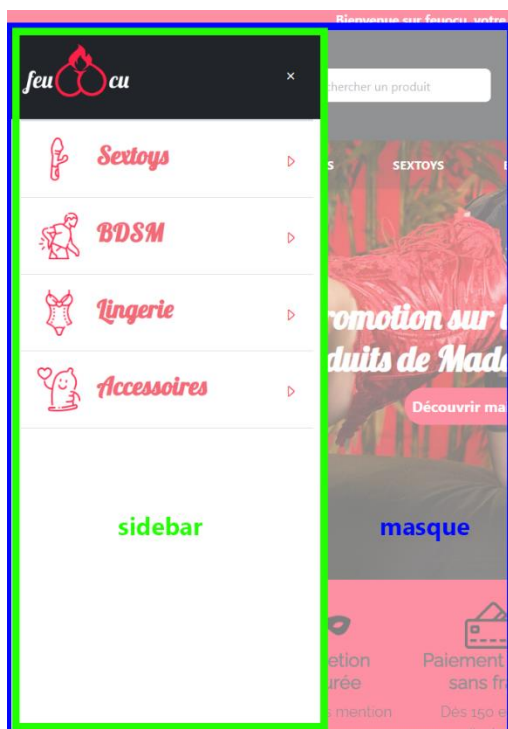


Header et barre de navigation en mode desktop

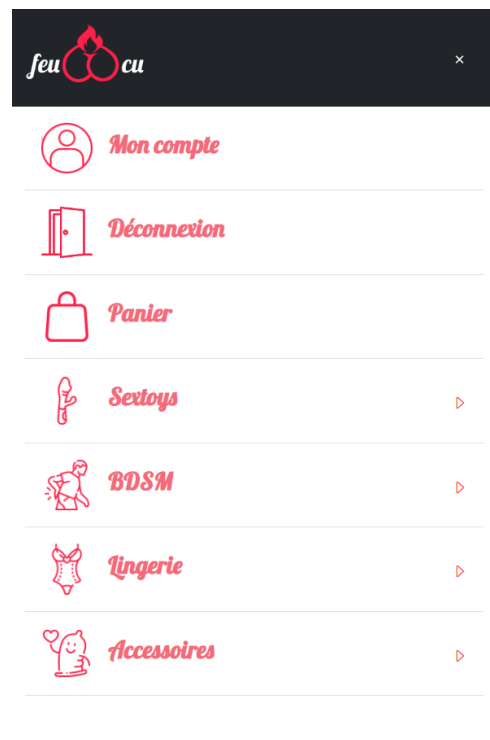


Header en mode mobile, une partie du contenu et la barre de navigation ont disparu

Le nouveau burger apparaissant en mode mobile est donc lié au même contrôleur stimulus qui permet d'afficher ou de faire disparaître la barre latérale et Bootstrap ainsi que l'affichage conditionnel de Twig me permettent d'y ajouter du contenu lorsque l'utilisateur utilise un petit écran.



La sidebar et son masque blanc en mode desktop



La sidebar en mode mobile

La barre latérale est un composant Twig intégré au template de base de mon application. Il est ainsi présent dans toutes les pages du site. Tout comme pour la barre de navigation horizontale, son contenu est généré automatiquement grâce à des données reçues du modèle par le contrôleur et transférées à ma vue pour être traitées par Twig.

La partie liée au parcours du catalogue contient deux éléments principaux contenus dans une div parente:

- une div contenant une liste intégrant les principales catégories de produits du site
- un masque blanc et disposant d'un degré d'opacité qui recouvre toute la page. Il est inséré entre la barre latérale et le reste de la page afin de créer une séparation pour l'utilisateur lorsque celui-ci active le menu latéral, il aura une autre utilité.

Par ailleurs, grâce à Twig, je suis en mesure de générer pour chaque catégorie une div indépendante et invisible contenant toutes les sous-catégories de la catégorie en question.

```
import {Controller} from "@hotwired/stimulus";

export default class extends Controller {
  sidebar;

  connect() {
    this.sidebar = document.getElementById('sidebar-wrap');

    const mainCategories = document.getElementsByClassName('main-category');
    Array.from(mainCategories).forEach((category) => {
      category.addEventListener('click', (event) => {
        document.getElementById(category.getAttribute('data-target')).classList.toggle('d-none');
      })
    })
  }

  toggle() {
    this.sidebar.classList.toggle('d-none');
  }

  hide() {
    const childCategories = document.getElementsByClassName('child-category');
    Array.from(childCategories).forEach((category) => {
      category.classList.add('d-none');
    })
  }
}
```

Fichier `toggle_sidebar_categories_controller.js`

Mon contrôleur possède donc 3 méthodes.

- La méthode connect() qui consiste au lancement de la page à associer à l'attribut de classe l'élément contenant le menu principal ainsi que le masque. Par ailleurs elle attribue un eventListener de type 'click' à chaque élément de la classe 'main-category' qui est la classe attribuée à chaque élément de la liste de catégories de la sidebar. Un clic sur une catégorie permettra d'afficher la div contenant toutes ses sous-catégories.
- Une méthode toggle() qui permet d'afficher ou de faire disparaître la sidebar.
- Une méthode hide() qui permet de faire disparaître les div de sous-catégories si besoin.

Grâce à l'attribut spécial data-action, je peux contrôler l'affichage et la disparition de toutes les div liées au menu.

Lorsque l'utilisateur clique sur un burger, le menu apparaît, lorsqu'il clique sur une catégorie, un sous-menu prend sa place. Un clic sur l'icône de fermeture ou sur toute partie du masque sous-jacent déclenche la fermeture de l'ensemble des éléments du menu.

```
data-action="click->toggle-sidebar-categories#toggle click->toggle-sidebar-categories#hide"
```

Lorsque l'utilisateur clique sur un lien de sous-catégorie, il est amené sur une nouvelle page dans laquelle tous les menus sont cachés par défaut. La boucle est bouclée.

En somme j'ai développé l'équivalent d'une sidebar Bootstrap dynamique et correspondant davantage à mes besoins que ceux proposés par la bibliothèque de styles.

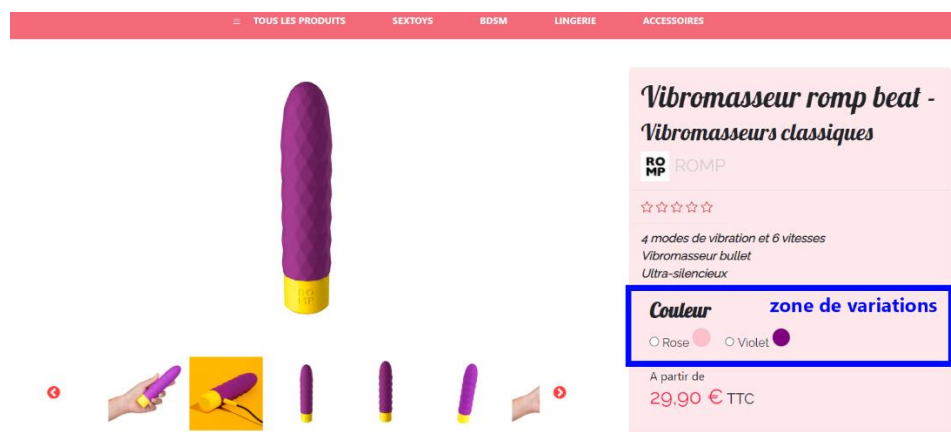
La page produit et la sélection des variants

L'affichage

Ma page d'affichage des produits est un élément essentiel de ma boutique en ligne. Pour l'aspect général je me suis inspiré des templates de l'outil de création de plateformes e-commerce Prestashop. Mon projet proposant des produits pouvant disposer de variations, il était impératif de développer un composant intuitif permettant à l'utilisateur de pouvoir sélectionner le variant adéquat afin de pouvoir l'ajouter à son panier.

La sélection des variants devant être dynamique, je me suis tourné vers l'approche AJAX (Asynchronous Javascript + XML) permettant grâce à Javascript d'envoyer des requêtes au serveur en arrière-plan et récupérer des données formatées sans avoir besoin de recharger la page. L'utilisation d'AJAX permet d'offrir une meilleure expérience utilisateur.

Dans mon cas, j'ai développé un contrôleur Stimulus spécifique, lié à chaque page produit et permettant de récupérer toutes les informations concernant les variations disponibles pour un produit.



Exemple d'une page produit présentant la zone dédiée aux variations.

Dans la capture ci-dessus, le produit présente des variations sur la couleur et sur le matériau, l'affichage des options disponibles est dynamique en fonction des disponibilités de chaque variant et si différents variants présentent un prix différent, alors il doit être mis à jour en temps réel en conséquence.



Sélection des variations et mise à jour de la page en conséquence

La partie de la fiche produit contenant les variations si elles existent ainsi que l'input de quantité et le bouton d'ajout au panier est un formulaire. Grâce à twig je peux conditionner l'affichage de ce formulaire à la disponibilité d'au minimum un variant de ce produit. Si aucun stock n'est disponible, alors je l'indique à l'utilisateur.



Ce produit n'est pas disponible

```
{% set variantsInStock = false %}
{% for variant in product.productVariants %}
    {% if variant.stock > 0 %}
        {% set variantsInStock = true %}
    {% endif %}
{% endfor %}
{% if product.productVariants|length > 0 and variantsInStock %}
    // Affichage du formulaire de variations
{% else %}
    // Indisponible
{% endif %}
```

Extrait de la vue `_product.html.twig` qui conditionne l'affichage des variations disponibles

Dans twig, je déclare une variable initialisée à false et pour chaque variant du produit reçu dans la vue, je cherche s'il est en stock. Dès que je trouve au moins un variant en stock, cela signifie que le produit est disponible et l'affichage qui suit créera un formulaire adapté.

Le contrôleur Stimulus (Code significatif)

```
export default class extends Controller {

  static targets = ['spinner'];
  data = {};
  tempData = {};
  selectedValues = [];
  depth = 0;
  numberOfOptions = -1;

  connect() {
    const productId = this.element.dataset.productId;
    this.spinnerTarget.style.display = 'block';
    fetch(`/shop/product/${productId}/options`, {
      method: 'GET'
    })
      .then(response => {
        if (!response.ok) {
          throw new Error('Fail : ' + response.status);
        }
        return response.json();
      })
      .then(data => {
        this.data = data;
        this.tempData = this.data.variants;
        this.numberOfOptions =
Object.keys(this.data.options).length;

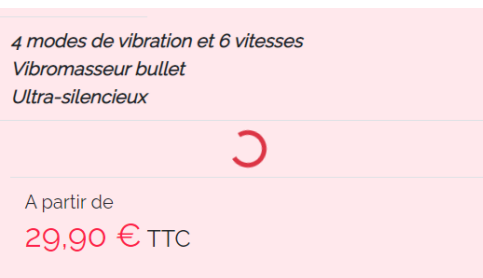
        this.spinnerTarget.style.display = 'none';

        if (this.tempData.hasOwnProperty('id')) {
          this.enableSubmit(this.tempData);
        }

        // On fournit les valeurs au premier input
        this.populateDomOption();
      })
  }
}
```

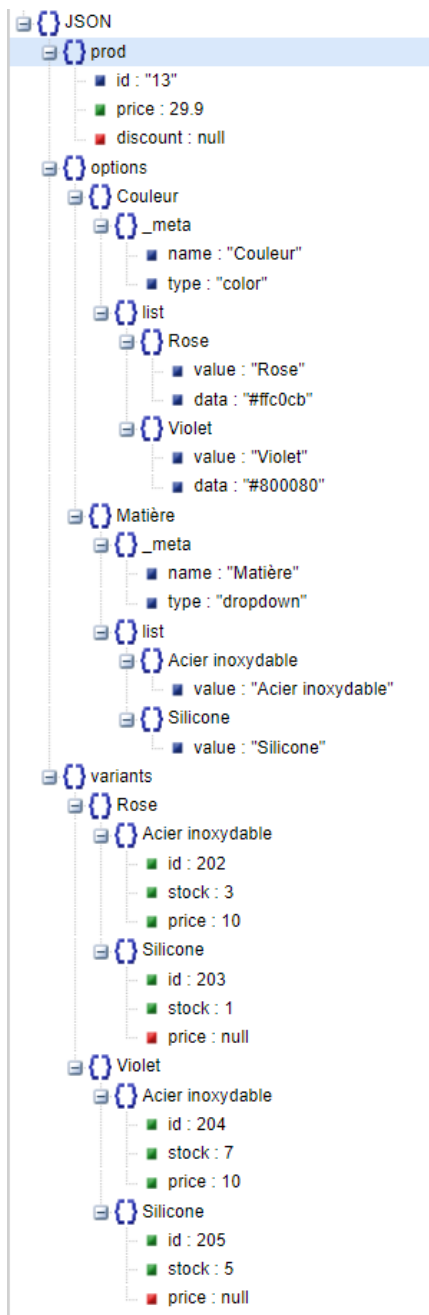
Méthode connect() du contrôleur options_controller

Lors de l'instanciation du contrôleur, je récupère l'identifiant du produit en cours depuis un attribut data-product-id de l'élément. Celui-ci va me permettre d'envoyer une requête asynchrone au serveur afin d'obtenir toutes les informations dont j'aurais besoin pour générer mon formulaire de variants dynamique. J'active également une div cachée qui contient une icône de chargement animée. Cette div sera active jusqu'à ce que le contrôleur Stimulus exécute une requête et reçoive une réponse attendue.



Une icône de chargement en attendant la réponse JSON

La méthode envoie donc une requête asynchrone vers `/shop/product/${productId}/options` en remplaçant `${productId}` par l'identifiant du produit récupéré plus tôt. Le contrôleur PHP va se charger de construire une réponse au format JSON et la renvoyer. Pendant que le contrôleur est en attente



La réponse JSON se présente comme tel. JSON associe des données en créant des paires clé-valeur. Dans mon exemple, la réponse contient 3 objets principaux.

- **prod** contient des informations directement liées au produit, son identifiant qui nous servira lors de l'ajout au panier, son prix de base ainsi que l'existence d'une réduction sur l'article.

- **options** va contenir les informations sur toutes les options présentes parmi les variants disponibles. Les objets JSON sont imbriqués et chaque clé a pour valeur un autre objet JSON. Chaque option aura dans ses valeur un objet **_meta** qui contient les informations nécessaires à l'affichage des options dans le formulaire. Il contient non seulement le nom de la valeur mais aussi son type, par exemple une option de type « color » sera représentée par des boutons radio associés à une gomme de la couleur en question tandis qu'une option de type « dropdown » sera représentée par un menu déroulant. A l'intérieur de l'objet **list**, je récupère pour chaque valeur d'option son nom ainsi qu'un code hexadécimal dans le cas des options de type couleur. C'est lui qui me permettra de créer les gommes dynamiquement.

- **variants** est une imbrication d'objets JSON structurée. Le parcourir permet de trouver en fin de branche chaque variant unique pour chaque objet. Dans l'exemple ci-contre, si j'ouvre l'objet dont la clé est Rose, puis l'objet dont la clé est Acier Inoxydable, je trouve à l'intérieur les informations relatives au variant Rose et en Acier Inoxydable. Cela me permet de retrouver l'identifiant du variant en question pour pouvoir ajouter le bon au panier, de connaître son stock et son éventuel surcoût. Dans

l'exemple pré-cité, le variant aura un stock de 3 et un surcoût de 10 €, ce qui me permettra de modifier dynamiquement la limite maximale de l'input de sélection des quantités ainsi que l'affichage du prix sur la page.

L'algorithme

Lorsque je reçois la réponse du serveur. J'enregistre les données reçues dans le contrôleur Stimulus grâce à l'attribut de classe `data`, ce qui me permettra de les réemployer à volonté pour mettre à jour la page. Je n'aurai plus besoin de demander d'informations au serveur à chaque fois que l'utilisateur interagira avec le formulaire puisque toutes les données sont déjà à disposition et ce jusqu'à ce que l'utilisateur quitte la page.

Dans la variable `tempData`, je place uniquement le sous-objet contenant l'imbrication d'objets JSON permettant d'atteindre un variant.

Grâce à `numberOfOptions` et en comptant le nombre de clé dans l'objet **options**, je peux savoir combien d'options sont à sélectionner pour pouvoir définir un variant.

`selectedValues = []` est un tableau vide qui contiendra les valeurs d'options successives sélectionnées par l'utilisateur.

`depth = 0` est une variable qui va me permettre de connaître la « profondeur » dans le processus de sélection des variations pour que le contrôleur Stimulus puisse savoir en temps réelle où récupérer toutes les informations dont il a besoin dans le fichier JSON. En association avec `numberOfOptions` il sera possible de savoir si l'utilisateur a terminé sa sélection des options et donc de savoir quand activer l'input des quantités ainsi que le bouton d'ajout au panier.

Je désactive à nouveau la div de chargement et je commence à modifier le DOM avec les informations disponibles.

```
if (this.tempData.hasOwnProperty('id')) {  
  this.enableSubmit(this.tempData);  
}
```

Si `tempData` a une propriété, cela signifie que le produit dispose d'un seul variant, aucun formulaire ne sera donc nécessaire et j'afficherai directement l'input de sélection des quantités ainsi que le bouton d'ajout au panier. Dans le cas contraire, j'affiche les valeurs disponibles pour la première option grâce à ma méthode créée pour peupler le DOM.

```
this.populateDomOption();
```

```
populateDomOption() {  
  const optionList =  
this.data.options[Object.keys(this.data.options)[this.depth]].list;  
  const optionName =  
this.data.options[Object.keys(this.data.options)[this.depth]]._meta.name;  
  const optionType =  
this.data.options[Object.keys(this.data.options)[this.depth]].meta.type;  
}
```

Extrait de la méthode populateDomOption()

C'est elle qui va pour chaque option choisir quel type d'input devra être affiché en fonction du type contenu dans le fichier JSON, quelles valeurs seront disponibles, etc.

Annexe 15 - extrait de code de la méthode populateDom() de options_controller.js

Dans l'**Annexe 15**, l'extrait présente la construction d'un input du DOM pour une option de type « color ». Je crée une div et je parcours l'objet `tempData`.

```
for (const key in this.tempData)
```

Pour chaque couple clé -> valeur, je récupère la valeur grâce à

```
const optionData = optionList[key]
```

Puis je crée un bouton radio auquel je lie le nom de la valeur `optionData.value` puis je crée un span de forme ronde en lui donnant pour background la couleur hexadécimale contenue dans `optionData.data`.

Une fois mon input créé, je l'ajoute au DOM grâce à :

```
this.element.appendChild(optionContainer);
```

puis je lui ajoute un eventListener qui va gérer l'interaction avec l'utilisateur :

```
optionField.addEventListener('change', (event) => {  
  this.handleInput(event, optionField);  
});
```

L'évènement survient à chaque changement du DOM et appelle la méthode :

```
handleInput(event, field) {  
  const currentPos = parseInt(field.getAttribute('data-pos'));  
  const selectedValue = event.target.value;  
  
  if (currentPos < this.selectedValues.length) {  
    this.depth = currentPos;  
    this.selectedValues = this.selectedValues.slice(0, this.depth);  
    this.tempData = this.data.variants;  
    for (let i = 0; i < currentPos ; i++) {  
      this.tempData = this.tempData[this.selectedValues[i]];  
      console.log(this.tempData);  
    }  
    this.clearDom();  
    this.disableSubmit();  
  }  
  this.selectedValues.push(selectedValue);  
  
  if (this.depth+1 < this.numberOfOptions) {  
    this.tempData = this.tempData[selectedValue];  
    this.depth++;  
    this.populateDomOption();  
  } else {  
    const variant = this.tempData[selectedValue];  
    this.enableSubmit(variant);  
  }  
}
```

C'est probablement la méthode la plus importante de mon contrôleur Stimulus car c'est elle qui va décider du comportement de ma page. L'attribut 'data-pos' est un attribut HTML donné par la méthode `populateDomOption()` à chaque input généré. Il est défini grâce à l'attribut de classe `depth` et permet à chaque interaction de l'utilisateur de savoir s'il a sélectionné l'option la plus « récente » ou s'il a voulu modifier une ancienne option. Le code

doit prendre en compte cette possibilité et le DOM doit être remis à zéro ou adapté en conséquence afin de proposer à l'utilisateur les variants effectivement disponibles pour la nouvelle sélection.

Toujours en comparant l'avancée de l'utilisateur dans le processus de sélection grâce à une comparaison simple entre `depth` et `numberOfOptions`, je suis en temps-réel capable de savoir si je dois continuer à générer des inputs pour l'utilisateur ou si je dois activer la sélection des quantités et le bouton d'ajout au panier.

Pour pouvoir afficher un nouvel input dans mon formulaire ou pour mettre à jour les input déjà existants. Je me sers de la variable `tempData` qui me sert à matérialiser les ouvertures successives des objets JSON qui se trouvent dans **variants**. C'est grâce à cette variable que je peux, dans le cas où l'utilisateur sélectionne une « ancienne » option, revenir à l'état initial.

Lorsque j'active la partie du formulaire réservée au Submit, je mets donc à jour la quantité maximale que l'utilisateur peut sélectionner, mets à jour l'affichage du prix en tenant compte du prix de base, de l'éventuel surcoût lié à une option et de toute réduction qui pourrait être associée au produit. Je profite de l'occasion pour transmettre l'identifiant du variant au bouton.

```
buttonElem.setAttribute('data-variant', variant.id);
```

C'est grâce à cet identifiant que je vais pouvoir ajouter le variant au panier.

L'ajout au panier

Le bouton d'ajout au panier dispose lui aussi d'un data-action qui permet d'appeler un contrôleur Stimulus lors du clic.

```
<button
  class="btn bg-fd2048 text-white btn-sm"
  data-mdb-ripple-duration = "0"
  type="submit"
  value="Ajouter au panier"
  data-add-cart-target="submitButton"
  data-action="add-cart#addToCart">
  Ajouter au panier
</button>
```

Lors de l'ajout, l'unique méthode `addToCart()` du contrôleur est donc appelée. Elle va elle aussi effectuer une requête asynchrone au serveur grâce à la fonction Javascript `fetch()`.

```
fetch(`/cart/add`, {
  method: 'POST',
  headers: {
    'Content-type': 'application/json'
  },
  body: JSON.stringify({variantId: variantId, quantity:quantity})
})
```

Dans le corps de la requête, je transmets l'identifiant du variant ainsi que la quantité sélectionnée par l'utilisateur au format JSON. Une fois la requête envoyée, le contrôleur Stimulus se place en attente le temps de recevoir la réponse. A la réception de celle-ci, j'utilise la méthode `dispatch` qui déclenche un élément personnalisé nommé « `added` » dans Stimulus. Celle-ci permet de notifier d'autres composants qu'un évènement spécifique s'est produit :

```
.then(data => {
  if (data.success) {
    this.dispatch('added', {
      target: document.documentElement,
    });
  }
})
```

Dans le cas d'un ajout effectué avec succès, j'informe l'utilisateur grâce à un message Flash de Bootstrap. De même lorsqu'une erreur survient. Le contrôleur Stimulus se chargera d'insérer un message flash Bootstrap dans une div dédiée et présente sur de nombreuses pages de mon site.

```
<div class="row" id = "flash-messages">
  {% for label, messages in app.flashes %}
    {% for message in messages %}
      {% if label == 'success' %}
        <div class="alert alert-success">
      {% elseif label == 'error'%}
        <div class="alert alert-danger">
      {% else %}
        <div class="alert alert-info">
      {% endif %}
        {{ message }}
      </div>
    {% endfor %}
  {% endfor %}
</div>
```

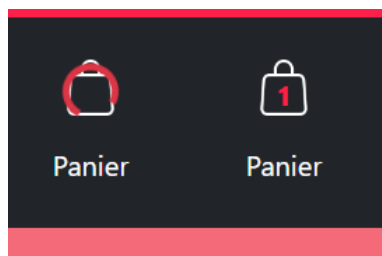
Composant Twig servant à afficher les erreurs. C'est dans cette même div que j'intègre en plus des messages personnalisés depuis mes contrôleurs Javascript.

```
const flashContainer = document.getElementById('flash-messages');
flashContainer.innerHTML = '';
[...];
const flashMessage = document.createElement('div');
flashMessage.classList.add('alert', 'alert-success');
flashMessage.textContent = 'Produit ajouté au panier avec succès';
flashContainer.appendChild(flashMessage);
```



En parallèle de l'ajout au panier, l'évènement « added » propagé plus tôt est intercepté par un autre composant qui s'occupe de l'affichage du nombre d'objets dans le panier au sein de mon header.

Lorsque j'ajoute un nouvel article au panier, une icône de chargement se superpose puis la quantité se met à jour lorsque l'objet est effectivement ajouté.



Avant - Après

```
<div
  id="cart-count"
  class="text-fd2048 fw-bold position-absolute top-50 start-50 translate-
middle mt-1"
  data-controller="cart-counter"
  data-action="
    add-cart:added@window->cart-counter#update
    cart-quantity:updated@window->cart-counter#update
    cart-delete:removed@window->cart-counter#update">
    <div data-cart-counter-target="count"></div>
    <div id="spinner-cart-counter-div" class="position-relative" data-cart-
counter-target="spinner">
      <div class="spinner-border text-danger" role="status"></div>
    </div>
</div>
```

Dans le code de la vue ci-dessus qui est responsable de l’affichage de l’icône panier ainsi que du compteur dans une div dédiée. L’attribut

`data-action="add-cart:added@window->cart-counter#update"` indique à l’instance du contrôleur Stimulus `data-controller="cart-counter"` qu’il doit être à l’écoute de l’évènement « added » et que lorsqu’il se produit il doit appeler sa méthode `update`. Celle-ci effectuera une requête au serveur pour avoir le nouveau nombre réel d’objets contenus dans le panier, il fera disparaître le spinner et affichera le nouveau compte dans la div dédiée.

On notera que cet élément du DOM est à l’écoute de 2 autres évènements qui sont propagés par d’autres contrôleurs, à savoir celui qui dans l’interface du panier se charge de mettre à jour la quantité d’un article et celui qui se charge de supprimer un article. Tous les contrôleurs communiquent les uns avec les autres afin qu’une modification effectuée par l’un soit répercutée dans ses « frères ».

Maintenant que j’ai expliqué comment fonctionnaient les templates Twig, l’affichage logique des données et les fonctionnalités interactives permises par Javascript, je vais pouvoir passer à la partie back-end de l’application, l’authentification, la logique métier et le fonctionnement du back-office.

Back-end

Le contrôleur, le 'C' de 'MVC'

Rôle du contrôleur

Pour faire simple, le rôle du contrôleur est de recevoir les requêtes des utilisateurs, de les traiter en utilisant la logique adaptée puis de retourner une réponse au client.

Les requêtes peuvent contenir des paramètres, des données contenues dans un formulaire envoyé ou toute autre information nécessaire à l'exécution d'une action spécifique.

C'est le contrôleur qui va organiser le déroulement des différentes actions à effectuer pour être en mesure de répondre à la requête. Il va interagir avec les Modèles (les entités dont j'ai parlé lors de la création de la base de données), utiliser les Repositories qui sont des classes intermédiaires entre les contrôleurs et les modèles pour pouvoir communiquer avec la base de données, utiliser des services pour effectuer des tâches particulières, etc.

C'est le contrôleur qui va exécuter la logique métier relative à la requête de l'utilisateur. Il va valider des données et/ou les modifier, prendre des décisions et préparer une réponse. Cette réponse peut consister à générer une vue, rediriger vers une autre page, envoyer des données JSON ...

```
<?php
namespace App\Controller;

class ShopController extends AbstractController
{
    #[Route('/shop/product/{slug?}', name: 'show_product')]
    public function showProduct(
        $slug,
        ProductRepository $productRepository,
        CategoryRepository $categoryRepository
    ): Response
    {
        $categories = $categoryRepository->findMotherCategories();

        $product = $productRepository->findOneBy(['slug' => $slug]);

        if (!$product) {
            throw $this->createNotFoundException('Ce produit n\'existe pas');
        }

        return $this->render('shop/show_product.html.twig', [
            'categories' => $categories,
            'product' => $product
        ]);
    }
}
```

Extrait du contrôleur en charge des pages de la boutique. La méthode est celle qui génère une route spécifique vers une page produit.

Dans l'exemple ci-dessus, le contrôleur retrouve toutes les catégories mères dans la base grâce à une méthode du Repository des catégories. Il retrouve l'objet produit correspondant au slug de l'URL et les transfère à la vue qui s'occupera de l'affichage adéquat.

Les routes

Les routes sont les URL accessibles aux utilisateurs, et elles permettent d'associer une URL à une action, à une méthode du contrôleur qui sera exécutée lors de l'accès à la dite-route.

Dans Symfony, les routes possèdent un chemin

```
#[Route('/shop/product/{slug?}', name: 'show_product')]
```

Dans Symfony, les routes possèdent :

- un chemin : `/shop/product/{slug?}`
`{slug?}` est un paramètre dynamique qui prendra la valeur du slug²¹ d'un produit auquel l'utilisateur essaye d'accéder. Le point d'interrogation `?` permet d'accepter une valeur nulle.
- Un nom : `'show_product'` qui est un identifiant unique pour la route et permettra de la mentionner de générer des routes plus simplement que via son chemin.
Par ailleurs, toute modification future du chemin de la route permettra de ne pas avoir à modifier chaque occurrence du chemin dans le code puisque le nom lui restera le même.
- Il est possible de spécifier les méthodes autorisées pour une route, (GET, POST, DELETE, etc.), par défaut, Symfony n'accepte que GET.

Lorsqu'un utilisateur essaye d'accéder à une URL, Symfony dispose d'un système de routage intégré qui va rechercher une route correspondante. Si un contrôleur est trouvé alors la méthode associée est appelée et la requête est traitée.

L'injection de dépendances

Symfony encourage à injecter des dépendances, cela permet de rendre le code plus modulable. Dans l'extrait de code ci-dessus, on constate que les Dépendances nécessaires sont passées en paramètre de la méthode du contrôleur. Il n'est pas nécessaire de les instancier explicitement. Symfonyinstanciera automatiquement `ProductRepository` de `ProductVariantRepository` lorsque la méthode sera appelée. Cela permet de séparer la logique métier du contrôleur. Le code devient plus facile à réutiliser, à entretenir et maintenir car les requêtes sont interchangeables.

²¹ Chaîne de caractères unique qui permet d'identifier une ressource

Les Repositories

Leur importance

Les repositories jouent un rôle important dans le fonctionnement des applications Symfony ou de toute autre architecture reposant sur ce pattern.

Ils permettent de gérer l'accès aux données de façon organisée et efficace et jouent le rôle d'interface entre le code et la base de données.

- Ils fournissent des méthodes prédéfinies pour récupérer des objets, rechercher et filtrer les données et permettent la création de méthodes personnalisées en utilisant le queryBuilder fourni par Doctrine qui facilite la création de requêtes dynamiques grâce à une syntaxe orientée objet.

```
public function findCategoryAndSubCategories(Category $category): array
{
    $products = [];
    $products = array_merge($products, $this->findBy(['category' =>
$category]));
    foreach ($category->getChildren() as $schild)
    {
        $products = array_merge($products, $this-
>findCategoryAndSubCategories($schild));
    }

    return array_filter($products, function ($product) {
        return count($product->getProductVariants()) > 0;
    });
}
```

Méthode de ProductRepository qui recherche tous les produits d'une catégorie ainsi que les produits de ses sous-catégories et retourne tous les produits qui ont des variants.

- Il est aussi possible d'écrire des requêtes SQL pur dans les méthodes.

```
public function findMotherCategories(): array
{
    $entityManager = $this->getEntityManager();

    $query = $entityManager->createQuery(
        'SELECT c, children
        FROM App\Entity\Category c
        LEFT JOIN c.children children
        WHERE c.parent IS NULL'
    );

    return $query->getResult();
}
```

Méthode findMotherCategories() de CategoryRepository.php, elle permet de récupérer toutes les catégories de produit qui n'ont pas de parent, elle est utilisée pour remplir la barre de navigation.

C'est principalement grâce aux repositories que l'application va pouvoir manipuler des données.

Exemple de la fonction de recherche

La barre de recherche sur le header de mon site est un formulaire simple contenant un input nommé 'searchInput' qui envoie une requête GET à la route 'search_products' lorsqu'il est soumis.

```
#[Route('/shop/search', name:'search_products')]
public function searchProducts(
    CategoryRepository $categoryRepository,
    ProductRepository $productRepository,
    Request $request
): Response
{
    $input = $request->query->get('searchInput', '');

    try {
        $products = $productRepository->search($input);
    }
}
```

Le contrôleur qui prend en charge les inputs des utilisateurs dans la barre de recherche.

Lorsqu'un utilisateur recherche un produit dans la barre de recherche, le contrôleur extrait la saisie de l'utilisateur de la requête puis appelle une méthode du ProductRepository afin de récupérer les produits correspondant dans la base de données.

```
public function search(string $query): array
{
    $queryBuilder = $this->getEntityManager()->createQueryBuilder();
    $queryBuilder->select('p')
        ->from(Product::class, 'p')
        ->leftJoin(Category::class, 'c', 'WITH', 'c.id = p.category')
        ->leftJoin(Brand::class, 'b', 'WITH', 'b.id = p.brand')
        ->where($queryBuilder->expr()->orX(
            $queryBuilder->expr()->like('p.name', ':query'),
            $queryBuilder->expr()->like('c.name', ':query'),
            $queryBuilder->expr()->like('b.name', ':query')
        ))
        ->setParameter('query', '%' . $query . '%');

    return $queryBuilder->getQuery()->getResult();
}
```

La fonction search() du ProductRepository

Je crée un objet QueryBuilder. Je sélectionne les entités Product grâce à `select('p')` en spécifiant que la table Product est la source de ces entités (`Product::class, 'p'`).

En utilisant la même syntaxe j'effectue des jointures avec les tables Category et Brand.

J'effectue ensuite la recherche grâce à la condition `->where`.

`expr()->orX` me permet de construire une expression de recherche avec une condition OR où le nom du produit ressemble au paramètre (`like('p.name', ':query')`).

Je lie ensuite la valeur du paramètre `:query` à la variable `$query` et j'autorise une recherche partielle, soit une recherche sur tous les résultats qui contiennent `$query`.

Les produits retrouvés par le Repository sont ensuite transformés avant d'être envoyés à la vue pour affichage.

Bonjour tout le monde

Rechercher un produit

1 Résultat



Vibromasseur usb pinga

MY FIRST

☆☆☆☆☆

Vibrant • Hypoallergénique • Waterproof • Rechargeable

Lisse et fin, ce vibromasseur servira aussi de vibromasse...

A partir de 29€

Voir plus

J'effectue une recherche PINGA qui correspond à un seul nom de produit

Bonjour tout le monde

Rechercher un produit

8 Résultats



Vibromasseur usb pinga

MY FIRST

☆☆☆☆☆

Vibrant • Hypoallergénique • Waterproof • Rechargeable

Lisse et fin, ce vibromasseur servira aussi de vibromasse...

A partir de 29€

Voir plus

J'effectue une recherche sur FIRST qui est contenu dans le nom de la marque de 8 produits.

Le contrôleur qui construit le fichier JSON des variants pour la page produit (Code significatif)

Annexe 22 - Méthode `getOptions()` du contrôleur `ShopController.php` qui génère un fichier JSON contenant les données relatives aux variations d'un produit

Pour illustrer le travail de la route appelée par le contrôleur Stimulus responsable de la construction d'un fichier JSON contenant toutes les informations sur les variations d'un produit, je vais expliquer son fonctionnement.

```
#[Route('/shop/product/{id}/options', name: 'get_product_options')]
public function getOptions(
    $id,
    ProductRepository $productRepository
): JsonResponse
```

La route ci-dessus est la route vers laquelle le contrôleur Stimulus évoqué précédemment et responsable de l'affichage des variants d'un produit envoie une requête. Comme indiqué il renvoie une réponse JSON.

Grâce à l'identifiant du produit en paramètre de la route `{id}`, je récupère le produit en question via le Repository de la classe et je vérifie qu'il existe et lève une exception le cas échéant.

```
$product = $productRepository->find($id);
if (!$product) {
    throw $this->createNotFoundException('Ce produit n\'existe pas');
}
```

Je récupère ensuite les variants de ce produit qui vont me servir à construire la réponse JSON.

```
$productVariants = $productRepository->getAvailableVariants($product);
```

Je construis un premier tableau qui contient les données relatives au produit et qui seront utilisées dans la logique d'ajout des variants au panier.

```
$productData = [
    'id' => $id,
    'price' => $product->getPrice(),
    'discount' => $product->getDiscount() ? [
        'type' => $product->getDiscount()->getType(),
        'val' => $product->getDiscount()->getValue()
    ] : null
];
```

Celui-ci contient uniquement des données retrouvées directement depuis l'entité produit

Ensuite j'initialise 2 tableaux vides qui vont être peuplés lors de l'itération sur les variants du produit :

```
$options = [];
$variants = [];
```

Si le produit possède des variants je démarre une boucle sur les variants :

```
if (count($productVariants) > 0) {
```

Pour chaque variant du produit, je récupère toutes ses options

```
foreach ($productVariants as $productVariant) {  
    $productVariantOptions = $productVariant->getProductVariantOptions();  
    $variantData = ['id' => $productVariant->getId()];
```

Puis le nom et le type de chaque option sont extraits pour initialiser une entrée du tableau \$options.

```
foreach ($productVariantOptions as $productVariantOption) {  
    $productOptionValue = $productVariantOption->getValue();  
    $productOption = $productOptionValue->getOption();  
    $optionName = $productOption->getName();  
    $optionType = $productOption->getType();  
  
    if (!isset($options[$optionName])) {  
        $options[$optionName] = [  
            '_meta' => [  
                'name' => $optionName,  
                'type' => $optionType,  
            ],  
            'list' => [],  
        ];  
    }  
}
```

Suite à quoi j'ajoute la valeur de l'option dans le tableau 'list' si celle-ci n'y est pas déjà présente. Si l'option est de type couleur, j'ajoute également le code hexadécimal de la couleur en question.

```
$optionvalue = $productOptionValue->getValue();  
  
if (!isset($options[$optionName]['list'][$optionvalue])) {  
    $options[$optionName]['list'][$optionvalue] = [  
        'value' => $optionvalue,  
    ];  
  
    if ($optionType === 'color') {  
        $options[$optionName]['list'][$optionvalue]['data'] =  
$productOptionValue->getHexacode();  
    }  
}  
$variantData[$optionName] = $optionvalue;  
}  
$temp = &$variants;  
foreach ($variantData as $key => $value) {  
    if ($key === 'id') {  
        continue;  
    }  
    if (!isset($temp[$value])) {  
        $temp[$value] = [];  
    }  
    $temp = &$temp[$value];
```

```

}

$temp = [
  'id' => $variantData['id'],
  'stock' => $productVariant->getStock(),
  'price' => $productVariant->getPrice() ? $productVariant->getPrice() :
null
];

```

Toujours dans la même boucle sur les variants du produit, j'initialise une variable `$temp` avec l'opérateur `&` qui me permet de pointer vers le contenu de `$variants`.

Je démarre ensuite une boucle sur chaque paire clé-valeur dans `$variantData`. Si la clé est `'id'` la boucle continue sans exécuter le code qui suit.

Si la clé n'est pas `'id'` alors je recherche l'existence de la valeur dans `$temp`. Si elle n'existe pas alors un tableau vide est créé à cet emplacement avec la valeur comme clé.

`$temp` est alors mise à jour avec une référence au tableau créé pour pouvoir se déplacer plus "profondément" dans le tableau lors des prochaines itérations.

A la fin de la boucle, `$temp` est mis à jour avec un tableau qui contient les informations sur le variant actuel, id, stock et prix.

A la fin de l'exécution de cette partie du code, je dispose d'un tableau `$options` qui contient pour chaque Option disponible un tableau `'list'` contenant la liste des valeurs disponibles et un autre tableau `'_meta'` contenant pour chacune des options son nom et son type qui seront utiles pour la construction du formulaire côté Front. Je dispose également d'un tableau `$variants` qui contient une structure imbriquée d'options menant à tous les variants disponibles.

Je renvoie donc une réponse JSON construite à partir des 3 tableaux obtenus.

```

return new JsonResponse(['prod' => $productData, 'options' => $options,
'variants' => $variants]);

```

C'est cet objet JSON qui sera récupéré par le contrôleur Stimulus responsable de l'affichage du formulaire de sélection côté front.

Les comptes utilisateur

Dans une application web et plus particulièrement e-commerce, il est essentiel d'implémenter des comptes utilisateurs, de permettre aux visiteurs de s'inscrire, de se connecter et d'accéder aux informations relatives à leurs comptes, le panier notamment. Symfony permet à travers le Security Bundle de mettre en place un système de gestion des comptes utilisateurs couvrant toutes ces étapes. Il permet par ailleurs d'attribuer des rôles et des permissions et sécuriser des routes pour limiter leur accès aux comptes autorisés.

Grâce à la documentation complète du Framework, j'ai pu comprendre les bases du fonctionnement de ce bundle et implémenter une gestion des comptes efficace.

Security Bundle et création de l'entité

```
composer require symfony/security-bundle
```

La première étape pour pouvoir me servir de ce bundle dans mon projet a été de l'installer grâce au gestionnaire de dépendances Composer.

L'installation du Security-Bundle m'a créé un fichier de configuration security.yaml permettant de configurer tous les paramètres relatifs à la sécurité de l'application tels que l'authentification, les autorisations ainsi que la gestion des rôles utilisateurs et la hiérarchie, les mécanismes de hachage de mots de passe, etc.

Les permissions sont obligatoirement liées à un objet utilisateur. Afin de sécuriser l'accès à des espaces de l'application, il convient donc de créer une classe utilisateur. Dans mon cas j'avais déjà créé une classe User lorsque j'ai créé ma base de données. J'ai alors apporté des modifications au fichier de configuration security.yaml en lui indiquant que mon entité `App\Entity\User` serait celle qui lui fournirait des utilisateurs et que l'email servirait de propriété unique afin de les identifier. J'ai modifié mon entité pour qu'elle implémente la classe Symfony `UserInterface` qui est nécessaire pour que Symfony puisse interagir avec des objets User dans le cadre de l'authentification et de la vérification des autorisations.

Annexe 17 - Extrait de code de l'entité User

Inscription, Connexion

Le formulaire d'inscription

Grâce au MakerBundle de Symfony j'ai pu créer facilement une classe de formulaire avec la commande

```
make:registration-form
```

Elle génère automatiquement une classe de formulaire d'inscription basique `RegisterFormType.php` placé dans le dossier Form de mon projet Symfony. Celle-ci contient tous les champs basiques nécessaires à la création d'un utilisateur.

Cette classe de formulaire implémente la méthode `buildForm()` de l'interface `FormBuilderInterface` de Symfony pour définir les champs, les options, des contraintes de validation, etc.


```

    $builder
        ->add('email', EmailType::class, [
            'attr' => [
                'class' => 'form-control border'
            ],
            'label' => 'Adresse email valide',
        ])
    ]
}

```

Exemple de définition du champ email dans la classe de formulaire RegisterForm.php

Par la suite, j'ai créé un contrôleur dédié à l'inscription d'un nouvel utilisateur.

Annexe 18 - RegistrationController qui gère l'inscription des utilisateurs

```

$user = new User();
$form = $this->createForm(RegisterFormType::class, $user);
$form->handleRequest($request);

```

```

return $this->render('registration/index.html.twig', [
    'form' => $form,
]);

```

C'est dans le contrôleur que je peux utiliser la classe de formulaire et la lier à une entité User afin de pouvoir récupérer les données saisies par l'utilisateur souhaitant créer un compte, vérifier qu'elles sont valides avant de les enregistrer dans ma base de données. Lorsque le contrôleur effectue le rendu du template Twig, il lui transmet le formulaire `$form` ainsi créé et je pourrai personnaliser le rendu dans Twig.

```

{{ form_start(form) }}

<div class="row">
    <div class="col-12 col-md-6">
        {{ form_label(form.email) }}
        {{ form_widget(form.email) }}
    </div>
    <div class="col-12 col-md-6">
        {{ form_label(form.emailConfirm) }}
        {{ form_widget(form.emailConfirm) }}
    </div>
</div>

```

La validation des données

Il convient de valider les données à la fois côté front-end et côté back-end pour garantir la sécurité de la base. Dans mon code, je dispose de 3 niveaux de validation :

- Côté front-end grâce au Form Builder qui permet d'ajouter des règles de validation qui seront effectuées par le navigateur avant même la soumission des données au serveur.

Les validations côté client sont les plus faciles à outrepasser, leur importance est donc moindre.

```
->add('email', EmailType::class, [
    'attr' => [
        'class' => 'form-control border'
    ],
    'label' => 'Adresse email valide',
    'constraints' => [
        new Length([
            'min' => 6,
            'max' => 180,
            'minMessage' => 'Le mail est trop court, 6 caractères minimum',
            'maxMessage' => 'Le mail est trop long, 180 caractères maximum'
        ])
    ]
])
```

Grâce à `EmailType::class` je m'assure que l'utilisateur entre bien une adresse mail et avec la contrainte `Length`, je lui impose une adresse dont la taille est comprise entre 6 et 50 caractères.

- Côté back-end avec les annotations de validation au sein des entités. Leur non-validation lors de la soumission des données engendrera une erreur.

```
#[ORM\Column(length: 180, unique: true)]
#[Length(
    min: 6, max: 180, minMessage: 'Le mail est trop court, 6 caractères minimum', maxMessage: 'Le mail est trop long, 180 caractères maximum'
)]
#[Email(
    message: 'L\'adresse email n\'est pas valide.'
)]
private ?string $email = null;
```

Dans l'entité `User`, on retrouve les mêmes règles de validation

`unique: true` permet de s'assurer de l'unicité des adresses email dans la base puisqu'elles servent d'identifiant aux utilisateurs et sont donc un élément essentiel de la sécurité. L'annotation `#[Email()]` garantit que la saisie est bien un email et `#[Length()]` vérifie que la taille de la chaîne de caractères soumise correspond bien aux attentes.

- Côté back-end dans la logique métier de l'application.

J'ai créé un service `UserService.php` qui permet d'effectuer toutes les opérations liées aux comptes utilisateurs, inscription, mise à jour de données, vérification de l'âge, etc. C'est ce service qui est appelé dans mon contrôleur lorsque l'utilisateur soumet le formulaire d'inscription.

```
if ($form->isSubmitted() && $form->isValid()) {
    try {
```

```

        $userService->registerUser(
            $user, $form['email']->getData(), $form['emailConfirm']->
            >getData(),
            $form['password']->getData(), $form['passwordConfirm']->
            >getData(), $form['birth']->getData());

```

```

public function registerUser(User $user, string $email, string
$emailConfirm, ?string $password, ?string $passwordConfirm, \DateTime
$birth): void
{
    if ($email !== $emailConfirm) {
        throw new \Exception('Les adresses email ne correspondent pas');
    }
    if ($password !== $passwordConfirm) {
        throw new \Exception('Les mots de passe ne correspondent pas.');
```

La fonction registerUser() de mon UserService.php

La fonction `registerUser()` vérifie que l'utilisateur a bien entré deux fois la même adresse email et deux fois le même mot de passe avant de poursuivre la création d'un compte.

```

public function createUser(User $user, ?string $password, \DateTime
$birth): void
{
    if (!$this->isAdult($birth)) {
        throw new \Exception('Il a perdu sa maman ? Il reviendra quand il
sera plus grand !');
```

La fonction createUser() de UserService.php vérifie quant à elle que l'utilisateur est bien majeur en faisant appel à une autre fonction du service.

En cumulant les niveaux de validation des données, je peux garantir leur intégrité avant de les insérer en base de données, ce qui contribue grandement à la sécurité et à la fiabilité de mon site.

Hachage du mot de passe

Afin de garantir la sécurité des comptes utilisateurs, le hachage des mots de passe est essentiel. C'est une technique qui utilise un algorithme mathématique pour transformer une entrée (dans notre cas une chaîne de caractères) en sortie qui est une séquence unique de caractères alphanumériques. C'est un processus unidirectionnel qui rend presque impossible de retrouver l'entrée originale à partir du hachage. Une entrée donnera toujours le même hachage.

Ce processus permet de stocker les hachages des mots de passe plutôt que de stocker les mots de passe directement en texte brut dans la base de données. Lorsqu'un utilisateur essaye de se connecter au site, on compare alors le hachage de la saisie de l'utilisateur avec le hachage présent dans la base. Cela garantit la sécurité des informations sensibles.

C'est la méthode personnalisée `hashPassword()` de mon `UserService.php` qui s'occupe de hacher les mots de passe lors de la création ou modification d'un compte.

```
public function hashPassword(?string $passwordToHash, User $user): string
{
    if (empty($passwordToHash)) {
        throw new InvalidArgumentException('Le mot de passe ne peut pas être vide.');
```

Elle utilise un objet de `UserPasswordHasherInterface` qui est une interface fournie par le Security Bundle de Symfony et permet d'effectuer des opérations de hachage et de vérification des mots de passe des utilisateurs.

Elle utilise l'algorithme de hachage BCrypt par défaut car il est considéré comme l'un des plus sûrs à l'heure actuelle.

```
« password » =>
$2y$10$a9f5flNvVHkBwRodpqkWRu4pQq6ZDQraIkfVxSwTbxtEmoP.Ym6SS
```

Exemple de hachage BCrypt avec pour entrée la chaîne de caractère « password »

Connexion

Dans mon application, j'utilise une authentification par formulaire via email et mot de passe. Comme indiqué dans la documentation de Symfony, j'ai créé un contrôleur spécifique dont la tâche est de rendre un formulaire de saisie.

Annexe 19 - Extrait du template de connexion de `index/login.html.twig`

```
#[Route('/login', name: 'app_login')]
public function index(
    AuthenticationUtils $authenticationUtils): Response
{
    if ($this->isGranted('IS_AUTHENTICATED_FULLY')) {
        return $this->redirectToRoute('app_home');
    }

    $error = $authenticationUtils->getLastAuthenticationError();
    $lastUsername = $authenticationUtils->getLastUsername();
    $form = $this->createForm(LoginFormType::class);

    return $this->render('login/index.html.twig', [
        'error' => $error,
        'last_username' => $lastUsername,
        'form' => $form
    ]);
}
```

Le contrôleur Symfony chargé de gérer les connexions des utilisateurs

Dans le fichier `security.yaml`, j'active le formulaire grâce aux réglages du firewall.

```
firewalls:
  main:
    form_login:
      login_path: app_login
      check_path: app_login
      username_parameter: "login_form[email]"
      password_parameter: "login_form[password]"
      enable_csrf: true
      access_denied_handler: App\Security\AccessDeniedHandler
```

Il indique à Symfony que la route `'app_login'` est la route par défaut à utiliser dans l'application lorsqu'une connexion est nécessaire. Il indique aussi que c'est cette même route qui sera utilisée pour valider les informations d'identification fournies par l'utilisateur. J'informe le firewall des champs du formulaire qui serviront d'identifiants et de mot de passe. Et j'en profite pour activer l'option `enable_csrf` grâce à laquelle Symfony automatisera la gestion d'un jeton CSRF dans le formulaire de connexion.

```
<div class="row">
  <input type="hidden" name="_csrf_token" value="{{
csrf_token('authenticate') }}">
</div>
```

C'est un input caché et généré par le Framework dans le template de chaque formulaire généré. Lorsque celui-ci est soumis par l'utilisateur, l'application vérifie que le jeton envoyé correspond bien à celui qui est attendu. Cela participe à la sécurité de l'application en garantissant que la session en cours est bien employée par l'utilisateur qui le prétend.

Le processus d'identification :

- Si l'utilisateur n'est pas authentifié et qu'il souhaite accéder à une zone protégée de l'application, Symfony le redirige vers le formulaire de connexion.
- L'utilisateur soumet le formulaire de connexion et les données du formulaire sont envoyées au serveur.
- Security Bundle de Symfony récupère les informations et recherche un utilisateur correspondant dans la base de données.
- Si l'utilisateur est authentifié avec succès, la classe User est instanciée pour stocker les données de l'utilisateur. Cet utilisateur authentifié est stocké dans le Token Storage, une zone de mémoire qui permet à l'application d'accéder rapidement à l'utilisateur si besoin.
- Un jeton d'authentification est créé pour représenter l'utilisateur nouvellement authentifié. Ce jeton sera réutilisé dans toutes les étapes du processus de sécurité.
- Pour déterminer si l'utilisateur a le droit d'accéder à la ressource demandée, Symfony utilise les rôles et les permissions puis autorise l'accès si c'est possible, dans le cas échéant, une exception de type AccessDenied est levée.

Déconnexion

Pour la gestion de la déconnexion avec Security Bundle, c'est très simple. La documentation de Symfony impose seulement la création d'activer le paramètre `logout` de `security.yaml` dans la partie dédiée au Firewall.

```
logout:
    path: app_logout
    target: app_login
```

Ici j'ai indiqué que la route `app_logout` est celle destinée à désauthentifier un utilisateur et que par défaut celui-ci sera ensuite redirigé vers `app_login`.

J'ai ensuite créé un contrôleur pour la route `app_logout` :

```
#[Route('/logout', name: 'app_logout', methods: ['GET'])]
public function logout(Request $request)
{}
```

Celui-ci est totalement vide, c'est Symfony qui se chargera de désauthentifier l'utilisateur en supprimant son jeton d'authentification et en nettoyant sa session.

Les rôles et les permissions

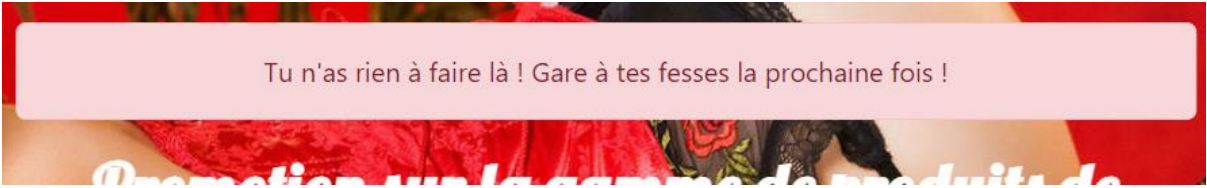
Dans le cas d'un accès refusé, j'ai spécifié dans le Firewall dans `security.yaml` d'accorder la gestion de l'erreur à une classe dédiée :

```
class AccessDeniedHandler implements AccessDeniedHandlerInterface
{
    public function __construct(private UrlGeneratorInterface
$urlGenerator) {
    }

    public function handle(Request $request, AccessDeniedException
$accessDeniedException): ?Response
    {
        $homepageRoute = $this->urlGenerator->generate('app_home', [],
UrlGeneratorInterface::ABSOLUTE_URL);
        $request->getSession()->getFlashBag()->add('error', 'Bouges, tu
n\'as rien à faire là ! Gare à tes fesses la prochaine fois !');
        return new RedirectResponse($homepageRoute);
    }
}
```

Lorsqu'une erreur de type `AccessDenied` est levée, c'est donc cette classe qui recevra la requête ayant mené à l'erreur ainsi que l'erreur en question. Le but de sa méthode `handle` est uniquement d'ajouter un message flash d'erreur à la session puis de rediriger l'utilisateur vers la page d'accueil du site. Le template de la page d'accueil intègre une bannière et celui-ci contient lui-même un template permettant de prendre en charge l'affichage des messages d'erreurs.

Dans l'exemple ci-dessous, j'essaie d'accéder au panneau d'administration avec un utilisateur ne disposant d'aucun droit.



Tu n'as rien à faire là ! Gare à tes fesses la prochaine fois !

L'utilisateur est redirigé vers la page d'accueil qui lui affiche le message d'erreur.

Les rôles

Symfony propose de base des rôles par défaut auxquels il est tout à fait possible d'ajouter des rôles personnalisés. Dans le cadre de mon site, j'ai eu besoin de 3 rôles uniquement.

- `ROLE_USER` qui est le rôle attribué à tous les utilisateurs ayant un compte et souhaitant s'authentifier.
- `ROLE_ADMIN` dont les autorisations sont étendues et vont en particulier pouvoir accéder à certaines parties du panneau d'administration afin de mettre à jour le contenu du site.
- `ROLE_SUPER_ADMIN` qui accorde les pleins pouvoirs à son détenteur.

Dans le fichier `security.yaml`, j'ai défini une hiérarchie des rôles qui permet d'indiquer à l'application que des rôles « supérieurs » incluent les droits des rôles « inférieurs ».

```
role_hierarchy:
    ROLE_ADMIN: ROLE_USER
    ROLE_SUPER_ADMIN: [ROLE_ADMIN, ROLE_ALLOWED_TO_SWITCH]
```

Les permissions

Dans une application Symfony, le système d'autorisations du SecurityBundle utilise les Rôles pour déterminer les droits d'accès d'un utilisateur à un contenu.

```
#[Route('/profile')]
#[IsGranted('ROLE_USER')]
class ProfileController extends AbstractController
{
    #[Route('/', name: 'app_profile')]
    public function index(
        CategoryRepository $categoryRepository
    ): Response
    {
        $this->denyAccessUnlessGranted('IS_AUTHENTICATED_FULLY');

        /** @var User $user */
        $user = $this->getUser();
        $categories = $categoryRepository->findMotherCategories();

        return $this->render('profile/index.html.twig', [
            'categories' => $categories,
            'user' => $user
        ]);
    }
}
```

Contrôleur permettant d'accéder à la route `/profile` qui permet à un utilisateur connecté de modifier les informations de son compte.

```
#[IsGranted('ROLE_USER')]
```

Cette annotation est utilisée pour bloquer l'accès à la route `/profile` aux utilisateurs ne disposant pas du rôle `ROLE_USER`. Puisque ce rôle est le rôle par défaut de tous les utilisateurs, l'accès à la route ne pourra pas être possible pour un utilisateur non authentifié. Lorsqu'un utilisateur essaiera d'accéder à une méthode du contrôleur `ProfileController`, Symfony vérifiera qu'il possède bien le rôle attendu. Dans le cas contraire une exception de type `AccessDeniedException` sera levée et l'accès sera refusé. Security Bundle propose également la méthode `isGranted()` qui prend en paramètre la chaîne de caractères représentant un rôle pour vérifier qu'un utilisateur en dispose avant d'exécuter une séquence de code.

```
$this->denyAccessUnlessGranted('IS_AUTHENTICATED_FULLY');
```

Dans l'exemple ci-dessus, cette méthode est utilisée pour vérifier que l'utilisateur est complètement authentifié avant de lui permettre d'accéder à une ressource.

Il existe plusieurs niveaux d'authentification dans Symfony qui représentent chacun un état.

- `IS_AUTHENTICATED_ANONYMOUSLY` : c'est l'état d'un utilisateur anonyme et sans compte qui visite le site.
- `IS_AUTHENTICATED_REMEMBERED` : c'est le cas d'un utilisateur qui se serait identifié par le passé avec l'utilisation d'une fonctionnalité de type « se souvenir de moi ». Il est identifié grâce à un cookie mais ne s'est pas « connecté ».
- `IS_AUTHENTICATED_FULLY` : l'utilisateur a fourni des identifiants valides et a été identifié entièrement par l'application.

L'emploi de la méthode `denyAccessUnlessGranted()` permet ici de s'assurer que lorsqu'un utilisateur tente d'accéder à sa page de profil, il s'est bien authentifié et a satisfait à toutes les exigences de sécurité en se connectant grâce au formulaire et en transmettant un jeton CSRF valide, apportant une couche de sécurité à l'application.

Le back-office

Introduction et rôle

Définition

Le back-office du site web fait référence à la partie réservée aux administrateurs ou utilisateurs autorisés. C'est une interface permettant de gérer le contenu, les fonctionnalités et les paramètres de l'application.

Dans le cadre de mon projet, il permettra principalement aux utilisateurs habilités de :

- Interagir avec les entités de la base en ajoutant de nouvelles marques, de nouveaux produits en configurant leurs variations, de créer de nouvelles options, etc.
- Configurer la bannière du site en définissant une nouvelle image, un message promotionnel ainsi qu'un lien vers la page appropriée si besoin.
- Manipuler les utilisateurs.

Accès sécurisé

Il est capital de protéger l'accès au back-office pour protéger les fonctionnalités administratives de l'application afin de ne permettre qu'aux seuls utilisateurs autorisés d'accéder et de manipuler des données sensibles.

Le Security Bundle offre la possibilité grâce au bloc `access_control` de `security.yaml` de spécifier des règles d'accès en fonction de chemins spécifiques d'URL.

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

Grâce à cette simple ligne, Symfony vérifiera le rôle de tous les utilisateurs tentant d'accéder aux routes débutant par `/admin`. Si l'utilisateur ne possède pas le rôle `ROLE_ADMIN` ou tout rôle englobant les mêmes accès, une exception de type `AccessDeniedException` sera levée.

Pour plus de sécurité, j'utilise à nouveau l'annotation `"IsGranted()"` et `"IS_AUTHENTICATED_FULLY"` dans le contrôleur de la route du back-office.

```
#[IsGranted('ROLE_ADMIN')]
class AdminController extends AbstractController
{
    #[Route('/admin', name: 'app_admin')]
    public function index(): Response
    {
        $this->denyAccessUnlessGranted('IS_AUTHENTICATED_FULLY');

        return $this->render('admin/index.html.twig', [
        ]);
    }
}
```

Le contrôleur chargé de rendre la vue de la page d'accueil du back-office

Gérer le CRUD dans le back-office

Explication du CRUD

Une des fonctionnalités essentielles du back-office est la Création, le Lecture, la Mise à jour et la Suppression des entités, aussi appelée CRUD (acronyme de Create, Read, Update, Delete). Pour chaque entité accessible aux utilisateurs du back-office, j'ai implémenté grâce au Maker Bundle des contrôleurs disposant de routes spécifiques pour :

- Créer des entités (Create)

Cette route contient toute la logique nécessaire à la création d'une nouvelle entité. Cette route traite les données reçues d'un formulaire et persiste une nouvelle instance de l'entité dans la base de données grâce à Doctrine. Comme vu précédemment pour la classe User, faire correspondre le contrôleur avec son formulaire de classe permettant de lier intrinsèquement les données saisies et l'entité à persister permet de faciliter la validation et la manipulation des données.

Pour les entités dont la logique de création d'entité est plus complexe, je passerai par l'utilisation d'un Service qui contiendra des méthodes permettant de scinder cette logique en blocs d'instructions, par exemple : upload une image, slugifier un nom de produit, etc .

- Lire des entités (Read)

Grâce aux Repositories de Doctrine, il est possible de définir des méthodes d'accès personnalisées aux données afin de récupérer des entités depuis la base de données en fonctions de critères de recherches précis. Ces entités peuvent ensuite être affichées dans des templates Twig en utilisant des instructions pour afficher les données souhaitées.

- Mettre à jour des entités (Update)

J'utilise une route qui permet de récupérer une entité existante et de l'injecter dans un formulaire de classe qui sera donc pré-rempli avec les données actuelles de l'entité. Lors de la soumission du formulaire, le contrôleur traite les nouvelles données, en faisant appel aux mêmes Services que pour la création d'entité si nécessaire puis le gestionnaire d'entité de Doctrine met à jour l'entité.

- Supprimer une entité (Delete)

Cette route utilise les repositories pour supprimer une entité de la base de données. Avec Symfony il est simple de d'ajouter des liens de suppression dans les vues Twig pour permettre aux administrateurs de supprimer les entités en 1 clic.

La commande `make:crud`

Grâce au Maker Bundle , il est possible de générer un CRUD basique pour une entité.

```
php bin/console make:crud Product
```

Cette commande va rapidement générer plusieurs fichiers de code qui serviront de point de départ à la création du CRUD pour un l'entité `Product` :

- **src/Controller/ProductController.php**, le contrôleur pour l'entité qui va inclure les routes nécessaires
- **src/Form/ProductType.php**, un formulaire de classe de base pour l'entité, il implémente la fonction `buildForm` pour créer un formulaire auquel il sera nécessaire d'ajouter des contraintes de validations et/ou de la logique de récupération de données.
- **templates/Product/_delete_form.html.twig**, un formulaire simple sous la forme d'un bouton de suppression intégrant une modale de confirmation ainsi qu'un token CSRF pour sécuriser l'opération destructive d'une entité. C'est ce formulaire qui pourra être facilement intégré dans un template Twig.
- **templates/Product/_form.html.twig**, une vue basique permettant de rendre le formulaire de classe **ProductType.php**
- **templates/Product/index.html.twig**, un template qui servira de page d'accueil pour le CRUD d'une entité. Il contiendra un listing des entités de la base ainsi que des liens vers les principales opérations pour chaque entité.
- **templates/Product/new.html.twig**, une vue incluant **_form.html.twig**.
- **templates/Product/edit.html.twig**, une vue similaire pour l'édition.
- **Template/Product/show.html.twig**, une vue permettant d'afficher toutes les informations d'un produit existant.

Créer une nouvelle entité (Produit)

Comme évoqué précédemment, le Maker Bundle ne génère qu'un squelette de base qui n'intègre aucune logique particulière dans sa conception du CRUD.

```
#[Route('/new', name: 'app_product_new', methods: ['GET', 'POST'])]
public function new(Request $request, ImageRepository $imageRepository):
Response
{
    $product = new Product();
    $form = $this->createForm(ProductType::class, $product);
    $form->handleRequest($request);

    if ($form->isSubmitted() && !$form->isValid()) {
        foreach ($form->getErrors(true) as $error) {
            $this->addFlash('error', $error->getMessage());
        }
    }

    if ($form->isSubmitted() && $form->isValid()) {
        try {
            $imageFiles = $form['images']->getData();
            $this->productService->createProduct($product,
            $imageRepository, $imageFiles);
            $this->addFlash('success', 'Produit ajouté avec succès');
        } catch (UniqueConstraintViolationException $e) {
            $this->addFlash('error', 'Un produit avec ce nom existe déjà');
            return $this->redirectToRoute('app_product_new');
        } catch (\Exception $e) {
            $this->addFlash('error', $e->getMessage());
        }

        return $this->redirectToRoute('app_product_index', [],
Response::HTTP_SEE_OTHER);
    }

    return $this->renderForm('product/new.html.twig', [
        'product' => $product,
        'form' => $form,
    ]);
}
```

Route responsable de la création d'un nouveau produit

Lorsque cette route est appelée, elle associe donc un formulaire de classe à une nouvelle entité vide. Il effectue 2 actions distinctes :

- Il injecte l'entité produit ainsi que le formulaire dans la vue adéquate.
- Il traite le formulaire dans le cas où celui-ci est soumis ET valide les contraintes qu'il intègre

Dans le cas où le formulaire correspond aux attentes de l'application, il fait appel à mon Service personnalisé ProductService.php qui est injecté comme dépendance dans le constructeur de mon contrôleur.

```
public function __construct(ProductService $productService)
{
    $this->productService = $productService;
}
```

Interface d'ajout d'un nouveau produit sur mobile

Certains attributs d'un produit ne sont pas saisis par l'utilisateur, et certaines actions ne sont pas effectuées automatiquement lors de la soumission du formulaire. C'est pourquoi la logique contenue dans `ProductService.php` est essentielle au bon fonctionnement de l'application.

Le contrôleur fait donc appel à la méthode `createProduct()` de la classe `ProductService` et lui passe en paramètre l'entité `$product`, les fichiers uploadés par l'utilisateur à la soumission du formulaire ainsi qu'une instance de `ImageRepository` pour traiter les images.

```
public function createProduct(Product $product, ImageRepository
$imageRepository, ?array $imageFiles): void
{
    $slug = $this->slugifyName($product);
    $product->setSlug($slug)
        ->setAdded(new \DateTimeImmutable())
        ->setEAN($this->generateRandomEAN())
        ->setSimpleProduct(true);

    if ($imageFiles) {
        $this->handleImages($product, $imageRepository, $imageFiles);
    }
}
```

```

    $this->productRepository->save($product, true);
}

```

Methode createProduct() du service ProductService

Cette méthode est responsable de la création et de la persistance d'un nouveau Produit. Pour mener à bien cette tâche, elle appelle elle aussi d'autres méthodes du Service, ce qui permet de découper le code et de le rendre plus lisible.

Ainsi elle appelle une méthode `slugifyName()` qui permet de transformer le nom de l'article en slug, supprimant ses majuscules, ses caractères spéciaux, ses espaces et y concatène un identifiant unique grâce à la fonction `uniqid()`, qui permet d'assurer l'unicité de chaque slug.

La méthode `generateRandomEAN()`, génère un code à 13 chiffres pour chaque produit.

`handleImages()`, quant à elle s'occupe de toute la logique relative à la création d'entités pour les images ainsi qu'à l'upload des fichiers et à leur rangement dans le dossier contenant les Images.

```

public function handleImages(Product $product, ImageRepository
$imageRepository, array $imageFiles): void
{
    $productName = $product->getName();
    foreach ($imageFiles as $imageFile) {
        $image = ImageService::saveImage($imageFile, $productName,
$imageRepository, 'products');
        $product->addImage($image);
        $directory = $this->parameterBag->get('images_directory') .
'/products/' . $productName;
        $fileName = $image->getFileName();
        ImageService::uploadImage($imageFile, $directory, $fileName);
    }
}





```

Méthode handleImage qui traite l'upload des images si l'utilisateur en a upload pour un produit.

Pour chaque fichier contenu dans le tableau de fichiers `$imageFiles` passés en paramètres, cette fonction va se charger d'appeler les méthodes **statiques** d'un autre service `ImageService` qui est lui chargé de gérer les opérations relatives aux images dans toute l'application et pas seulement pour les Produits.

Ce service contient des méthodes permettant de sauvegarder une entité Image dans la base de données, d'upload une image sur le serveur, de supprimer une image du serveur et de supprimer un dossier si celui-ci ne contient plus d'images.

Revenons-en à notre produit, si celui-ci a été créé sans problème, alors l'utilisateur est redirigé par le contrôleur vers la page d'index du CRUD de l'entité et un message lui indiquant que le produit a été ajouté avec succès sera affiché, le cas échéant, c'est un message d'erreur qui lui sera affiché sur la page en cours.

Administration

Produit ajouté avec succès




Produits

[Ajouter un nouveau produit](#)

Produit ajouté avec succès

Ajouter un produit c'est bien, ajouter des variants c'est mieux (Code significatif).

Je vais reprendre l'exemple du produit développé dans la partie relative à l'affichage des variants de produits via un contrôleur Stimulus dans la partie Front-end.

Vibromasseur ROMP Beat	Vibromasseurs classiques	ROMP		29,90 €	pas de note	6265440013823	Voir Modifier SUPPRIMER Variants du produit
Vibromasseur réaliste 16,5 cm Midi Girthy	Vibromasseurs réalistes	Lovehoney		34,90 €	pas de note	1525008975622	Voir Modifier SUPPRIMER Variants du produit
Vibromasseur ROMP Hype	Vibromasseurs classiques	ROMP		39,90 €	pas de note	7719315945188	Voir Modifier SUPPRIMER Variants du produit

Exemple de 3 produits visibles depuis l'index du CRUD des Produits

Je viens tout juste d'ajouter un produit dans ma base de données. Celui-ci existe dans la base mais ne dispose d'aucune variation. Or ce sont les variations de produits que j'ai décidé de rendre disponibles et donc visibles aux clients. Pour pouvoir rendre un produit disponible à la vente, il faut donc lui ajouter des variants et du stock.

Dans l'exemple ci-dessus, on retrouve 3 produits différents. Pour simplifier la visibilité des produits depuis l'index, j'ai choisi d'utiliser un code couleur sur la cellule contenant le nom du produit grâce à la puissance de l'affichage conditionnel de Twig.

```
{% if product.productVariants|length == 0 %}
<td style="background-color: rgba(255, 0, 0, 0.5);">
  {{ product.name }}</td>
{% elseif product.isInStock %}
  <td style="background-color: rgba(0, 128, 0, 0.5);">
    {{ product.name }}</td>
{% else %}
  <td style="background-color: rgba(255, 255, 0, 0.5);">
    {{ product.name }}</td>
{% endif %}
```


- Rouge : Le produit ne dispose pas de variants pour le moment
- Vert : Le produit a des variants configurés et au moins l'un d'eux est en stock
- Jaune : Le produit a des variants configurés mais ils sont en rupture

C'est dans le contrôleur 'app_product_index', avant de rendre la page que je vérifie si chaque produit est en stock et que je lui donne un attribut booléen correspondant.


```
'products' => array_map(static function (Product $product) {
    $isInStock = false;

    foreach($product->getProductVariants() as $productVariant) {
        $isInStock = $isInStock || $productVariant->getStock() > 0;
    }
    $product->isInStock = $isInStock;

    return $product;
}, $products),
```

Vibromasseur ROMP Beat	Vibromasseurs classiques	ROMP		29,90 €	pas de note	6265440013823	Voir Modifier SUPPRIMER Variants du produit
---------------------------	-----------------------------	------	---	---------	-------------------	---------------	---

J'ai donc ajouté un lien "Variants du produit" menant à une vue personnalisée gérée par une route du contrôleur ProductController.php.

```
#[Route('/{id}/variants', name: 'app_product_variants', methods: ['GET', 'POST'])]
public function variants(
    Request $request,
    Product $product): Response
{
    $formBuilder = $this->createFormBuilder();
    $this->productService->createVariantsForm($formBuilder, $product);
    $form = $formBuilder->getForm();
    $form->handleRequest($request);
```

Extrait de la route responsable de l'affichage et de la gestion des variants d'un produit.

La route fait appel à la méthode `createVariantsForm()` du contrôleur ProductService.php.

Annexe 20 - Méthode `createVariantsForm()` du Service ProductService.php

Cette méthode modifie le `$formBuilder` et ajoute un champ de selection multiples, pour chaque option de la base de données retrouvée grâce au Repository des options et à sa méthode `findAll()`.

Par ailleurs, la méthode ajoute également un bouton Checkbox `'no_variant'` pour gérer le cas où un produit ne dispose que d'un seul et unique variant (comprendre sans options). Ce bouton est lié à l'attribut `isSimpleProduct` de l'entité Produit et dispose d'un attribut `'data-action'` pour lier le clic à une action d'un contrôleur Stimulus.

Ce formulaire est ensuite intégré dans le template Twig correspondant qui contient un tableau contenant tous les variants et les informations à leur sujet, code, surcoût, stock, etc.

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

{% block title %}Variants{% endblock %}

{% block body %}
    <div class="row py-3 border-bottom">
        <h1>Variants de {{ product.name }}</h1>
    </div>
{% endblock %}
```

```

</div>
<div class="row py-3 border-bottom">
  <a href="{ { path('app_product_index') } }">Retour à la liste</a>
</div>
<div class="row py-3">
  <table class="table"...>
</div>
<div class="row" data-controller="admin-product-variant-options">
  {{ form_start(form) }}
  {{ form_widget(form) }}
  <button class="btn">{{ button_label|default('Ajouter /
Réinitialiser les variants') }}</button>
  {{ form_end(form) }}
</div>
{% endblock %}

```

J'ai décidé d'utiliser grandement le Javascript et Stimulus pour gérer certaines fonctionnalités de cette page. Le formulaire est ainsi intégré dans une div contrôlée par le contrôleur Stimulus `admin_product_variant_options_controller.js`.

```

export default class extends Controller {
  static targets = ['noVariant'];
  options;
  connect() {
    this.options = this.element.querySelectorAll('.variant-option');
    if (this.noVariantTarget.checked) {
      this.options.forEach((option) => {
        option.classList.add('d-none');
      })
    }
  }
  toggle() {
    this.options.forEach((option) => {
      option.classList.toggle('d-none');
    })
  }
}

```

Le contrôleur Stimulus qui gère l'affichage et le masquage des options de création de variants.

Lorsque la page est chargée, la méthode `connect()` cache la partie du formulaire relative aux options. Lorsque la checkbox est décochée, elle les réaffiche grâce au `'data-action'` et l'appel à la méthode `toggle()`. Cela permet de faire comprendre à l'utilisateur qu'il ne peut créer un produit sans variations et créer des variations pour ce même produit en même temps (cela même si le code du contrôleur l'en empêchera de toute façon).

Lorsque l'utilisateur soumet le formulaire de création de variants, le contrôleur va tout d'abord vérifier si la cellule "no-variant" est cochée. Dans ce cas il fera appel à la méthode `createUniqueVariant()` du `ProductService` puis l'utilisateur sera directement redirigé. Si la cellule est décochée, une méthode `deleteUniqueVariant()` permettant de supprimer le variant unique du produit s'il existe sera exécutée.

```

public function createUniqueVariant(Product $product): void
{
    // On vérifie que le variant unique n'existe pas déjà, s'il existe on
    // ne fait rien
    if (!$this->productVariantRepository->findOneBy(['code' => $product-
    >getName()]))) {
        $product->setSimpleProduct(true);
        $this->productRepository->save($product, true);
        // On supprime les variants existants pour ce produit
        $variants = $this->productVariantRepository->findBy(['product' =>
        $product]);
        foreach ($variants as $variant) {
            $this->productVariantRepository->remove($variant, true);
        }
        // On crée le variant sans options pour le produit
        $variant = new ProductVariant();
        $variant->setProduct($product);
        $code = $product->getName();
        $variant->setCode($code);
        $this->productVariantRepository->save($variant, true);
    }
}

```

La fonction `createUniqueVariant()` vérifie tout d'abord si le variant unique existe, si tel est le cas, la fonction ne fait rien. Sinon la fonction met à jour l'attribut `simpleProduct` de l'entité `Produit` puis supprime tous les potentiels variants existants afin de les remplacer par le nouveau variant unique.

La partie la plus significative du code concerne le traitement des données du formulaire dans le cas où des variants avec options doivent être créés. Encore dans un souci d'optimisation du code et de lisibilité, j'ai découpé la logique de création des variants grâce à la création de méthodes dans le `ProductService.php`.

Ces méthodes seront appelées une à une.

```

$filterData = $this->productService->processOptionsDataOnly($data);
$allEmpty = $this->productService->areOptionsSelected($product,
$filterData);
if (!$allEmpty) {
    $dataToCombine = $this->productService-
    >generateDataToCombine($filterData);
    $combinations = $this->productService-
    >generateCombinations($dataToCombine);
    foreach ($combinations as $combination) {
        $this->productService->createVariant($product, $combination);
    }
}
$this->addFlash('success', 'Variants ajoutés avec succès.');
```

Logique de création des variants en fonction des options sélectionnées par l'utilisateur.

Le contrôleur transmet les données du formulaire à une première méthode.

```
public function processOptionsDataOnly(array $data): array
{
    $filteredData = [];
    foreach($data as $optionName => $selectedValues) {
        if (str_starts_with($optionName, 'Option')) {
            $filteredData[] = $selectedValues;
        }
    }
    return $filteredData;
}
```

Méthode processOptionsDataOnly() de ProductService.php

La mission de cette première méthode est de filtrer les données du formulaire afin d'exclure les données relatives à l'input 'no-variant'. Il ne retourne que les données relatives aux options.

```
public function areOptionsSelected(Product $product, array
$sortedOptionValues): bool
{
    foreach ($sortedOptionValues as $values) {
        if (!empty($values)) {
            return false;
        }
    }
    return true;
}
```

Méthode areOptionsSelected() de ProductService.php

Cette méthode vérifie si parmi les données renvoyées par la première méthode, des options ont effectivement été sélectionnées. Si ce n'est pas le cas, il est inutile de continuer l'exécution du code puisqu'aucun variant ne sera créé.

Si des options ont effectivement été sélectionnées, une autre méthode est appelée et consiste à ne retourner que les données sélectionnées et qui seront utilisées pour générer des combinaisons.

```
public function generateDataToCombine(array $data): array
{
    $dataToCombine = [];
    foreach ($data as $optionValues) {
        if (!empty($optionValues)) {
            $dataToCombine[] = $optionValues;
        }
    }
    return $dataToCombine;
}
```

Méthode generateDataToCombine() de ProductService.php

Cette méthode retourne un tableau de tableaux qui contient uniquement les éléments non-vides du tableau `$data` qui sont eux-même des tableaux.

```

public function generateCombinations(array $dataToCombine): array
{
    $combinations = [[]];
    foreach ($dataToCombine as $optionValues) {
        $tmp = [];
        foreach ($combinations as $combination) {
            foreach ($optionValues as $value) {
                $tmp[] = array_merge($combination, [$value->getValue()]);
            }
        }
        $combinations = $tmp;
    }
    return $combinations;
}

```

Méthode generateCombinations() de ProductService.php

Dans cette méthode suivante :

- Je déclare un tableau de tableaux dont le premier élément est vide et qui va accueillir les combinaisons.
- Pour chaque élément du tableau de tableaux `$dataToCombine` renvoyé par la méthode précédente que je nomme `$optionValues`
- Je déclare une variable temporaire de type array `$tmp`.
- Puis pour chaque élément du tableau de combinaisons,
- Je parcours les éléments de `$optionValues` sous le nom `$value` et je crée une nouvelle combinaison en fusionnant la valeur actuelle de `$combination` avec la valeur de `$value` grâce à la méthode `array_merge`. Cette combinaison est ajoutée à `$tmp`.
- Lorsque toutes les valeurs de `$optionValues` ont été parcourues, `$tmp` contient toutes les nouvelles combinaisons pour cette itération. `$combinations` est mis à jour avec les nouvelles combinaisons de `$tmp`.
- Après avoir effectué toutes les itérations de `$dataToCombine`, `$combinations` contient toutes les combinaisons qu'il est possible de générer depuis les données contenues dans `$dataToCombine`.

Je termine la génération de variants en utilisant chaque combinaison pour générer un variant pour le produit.

```

public function createVariant(Product $product, array $options): void
{
    $variant = new ProductVariant();
    $variant->setProduct($product);
    $code = $product->getName() . ':' . implode(':', $options);
    $variant->setCode($code);
    if (!$this->productVariantRepository->findBy(['code' => $code])) {
        $this->productVariantRepository->save($variant, true);

        for ($i = 0; $i < count($options); $i++) {
            $newProductVariantOption = new ProductVariantOption();
            $newProductOptionValue = $this->productOptionValueRepository-
>findOneBy(['value' => $options[$i]]);
            $productVariantOptionCode = $variant->getCode() . '//' .
            $newProductOptionValue->getValue();

```

```

        $newProductVariantOption->setVariant($variant)
        ->setCode($productVariantOptionCode)
        ->setOption($newProductOptionValue->getOption())
        ->setValue($newProductOptionValue);

        $this->productVariantOptionRepository-
>save($newProductVariantOption, true);
    }
}

```

Méthode createVariant() du service ProductService

Celle-ci génère un code à partir du nom du produit ainsi que des dénominatifs des options. Elle vérifie qu'un variant avec ce code n'existe pas déjà. Si ce n'est pas le cas alors je crée un variant que je sauvegarde dans la base de données.

Pour chaque option, je crée une instance de `ProductVariantOption` liée au Variant, au type d'Option et à la valeur de l'option puis je la persiste dans la base de données.

Variants de Vibromasseur ROMP Beat

[Retour à la liste](#)

Id	Code	Impact sur le prix	Prix Total	Quantité	Actions
Produit sans variants <input type="checkbox"/> Taille * <input type="checkbox"/> L <input type="checkbox"/> XL Couleur * <input type="checkbox"/> Noir <input type="checkbox"/> Blanc <input checked="" type="checkbox"/> Rose <input type="checkbox"/> Rouge <input checked="" type="checkbox"/> Violet <input type="checkbox"/> Bleu <input type="checkbox"/> Vert <input type="checkbox"/> Jaune <input type="checkbox"/> Orange <input type="checkbox"/> Argent <input type="checkbox"/> Or Matière * <input type="checkbox"/> Coton <input type="checkbox"/> Plastique <input checked="" type="checkbox"/> Acier inoxydable <input checked="" type="checkbox"/> Silicone <input type="button" value="AJOUTER / RÉINITIALISER LES VARIANTS"/>					

L'utilisateur sélectionne les variations et soumet le formulaire

Variants ajoutés avec succès.

Variants de Vibromasseur ROMP Beat

[Retour à la liste](#)

Id	Code	Impact sur le prix	Prix Total	Quantité	Actions
210	Vibromasseur ROMP Beat:Rose,Acier inoxydable	<input type="text"/> €	29,90 €	<input type="text" value="0"/>	SUPPRIMER
211	Vibromasseur ROMP Beat:Rose,Silicone	<input type="text"/> €	29,90 €	<input type="text" value="0"/>	SUPPRIMER
212	Vibromasseur ROMP Beat:Violet,Acier inoxydable	<input type="text"/> €	29,90 €	<input type="text" value="0"/>	SUPPRIMER
213	Vibromasseur ROMP Beat:Violet,Silicone	<input type="text"/> €	29,90 €	<input type="text" value="0"/>	SUPPRIMER

Les variants sont créés avec succès

Comme je l'ai expliqué précédemment, j'ai choisi de me reposer sur Stimulus pour gérer les prix ainsi que les quantités de chaque variant afin de permettre à l'utilisateur de modifier les valeurs sans avoir besoin de recharger la page.

Chaque ligne du tableau représentant les variants dispose de champs liés à une instance de contrôleur Stimulus pour mettre à jour les valeurs. Celui-ci ajoute un EventListener de type 'change' sur le champ qui déclenche une requête asynchrone pour modifier la quantité.

Annexe 21 - Contrôleur Stimulus qui modifie la quantité d'un variant

```
fetch('/admin/product/variant/update/stock', {
  method: 'POST',
  headers: {
    'Content-type' : 'application/json'
  },
  body: JSON.stringify({variantId: variantId, quantity: quantity})
})
```

Extrait de la méthode connect() de admin_product_variant_controller.js

La méthode connect() du contrôleur Stimulus envoie une requête asynchrone vers la route '/admin/product/variant/update/stock'. Elle transmet des données au format JSON dans le header de la requête, ces données contiennent l'identifiant du variant à modifier ainsi que la nouvelle quantité.

```
#[Route('/variant/update/stock', name:
'app_product_update_variant_quantity', methods: ['POST'])]
public function updateQuantityVariant(
    Request $request,
    ProductVariantRepository $productVariantRepository
): JsonResponse
{
    $data = json_decode($request->getContent(), true);
    $variantId = (int) $data['variantId'];
    $quantity = (int) $data['quantity'];

    $productVariant = $productVariantRepository->find($variantId);
    if ($productVariant) {
        if ($quantity >= 0) {
            $productVariantRepository->setQuantity($productVariant,
            $quantity, true);
            return new JsonResponse(['success' => true]);
        } else {
            return new JsonResponse(['success' => false, 'message' =>
            'Veuillez entrer une quantité positive']);
        }
    } else {
        return new JsonResponse(['success' => false, 'message' => 'Variant
non trouvé.']);
    }
}
```

Route 'app_product_update_variant_quantity' de ProductController.php

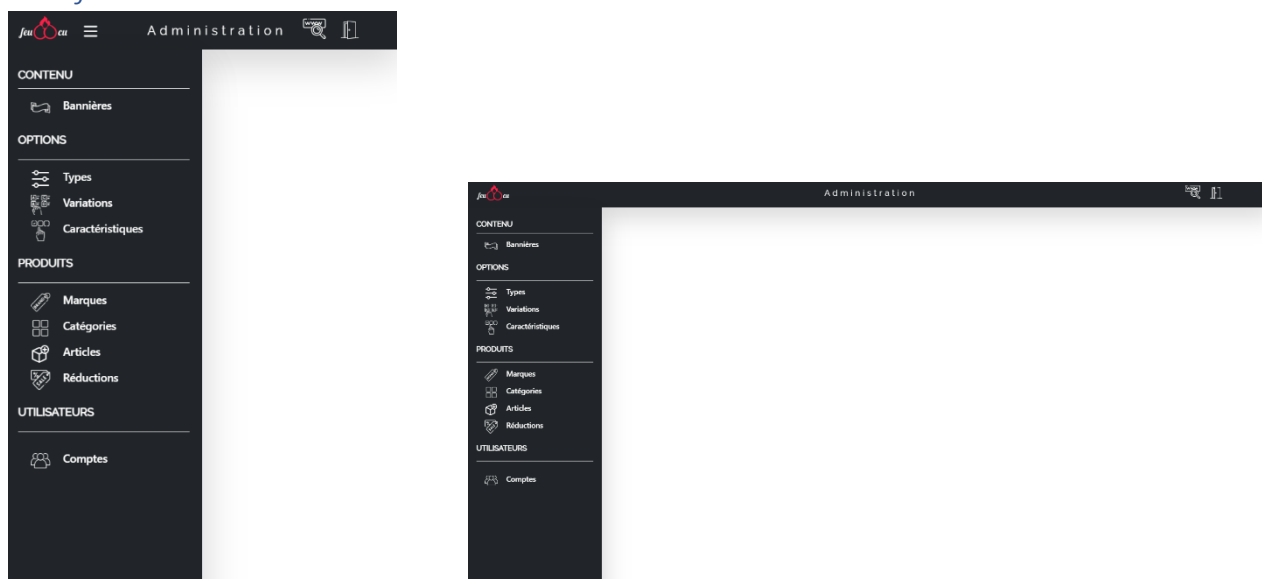
La méthode récupère les données transmises dans le header de la requête et décode le JSON afin d'en extraire les 2 variables.

Elle utilise ensuite `ProductVariantRepository` pour récupérer le variant, si tout se passe bien et que la quantité est positive, elle met à jour sa quantité et renvoie une réponse JSON indiquant que tout s'est déroulé correctement. Dans le cas contraire, elle retourne une réponse indiquant l'échec associée à un message d'erreur expliquant le problème.

Prix mis à jour					
Variants de Vibromasseur ROMP Beat					
Retour à la liste					
Id	Code	Impact sur le prix	Prix Total	Quantité	Actions
210	Vibromasseur ROMP Beat:Rose,Acier inoxydable	<input type="text" value="20"/> €	49.90 €	<input type="text" value="8"/>	<button>SUPPRIMER</button>
211	Vibromasseur ROMP Beat:Rose,Silicone	<input type="text"/> €	29.90 €	<input type="text" value="0"/>	<button>SUPPRIMER</button>
212	Vibromasseur ROMP Beat:Violet,Acier inoxydable	<input type="text"/> €	29.90 €	<input type="text" value="0"/>	<button>SUPPRIMER</button>
213	Vibromasseur ROMP Beat:Violet,Silicone	<input type="text"/> €	29.90 €	<input type="text" value="0"/>	<button>SUPPRIMER</button>

Le surcoût et la quantité du variant 210 ont été mis à jour avec succès

Interface



Back-office mobile et desktop

Tout comme pour le front-end de mon site, grâce à la librairie Bootstrap et au Grid, j'ai pu développer une interface adaptable et responsive. Même si il n'est pas forcément pratique d'utiliser le back-office depuis un support mobile, c'est une fonctionnalité de plus en plus demandée par les clients et c'est pourquoi il était important de permettre le double emploi à l'application.

Conclusion

Le back-office joue donc un rôle crucial dans le développement d'une application web. Il permet donc aux administrateurs autorisés de gérer toutes les données contenues sur le site. Cette partie de l'application est sécurisée grâce aux fonctionnalités d'authentification de Symfony.

La configuration du fichier `security.yaml` et l'utilisation des annotations `#[IsGranted()]` et de la fonction `denyAccessUnlessGranted()` permet de contrôler les accès ainsi que les permissions des utilisateurs y accédant.

La gestion du CRUD est le cœur de l'application car elle permet donc de configurer le contenu de manière efficace et sécurisée. Le back-office offre un environnement sécurisé et convivial pour le client.

Les tests

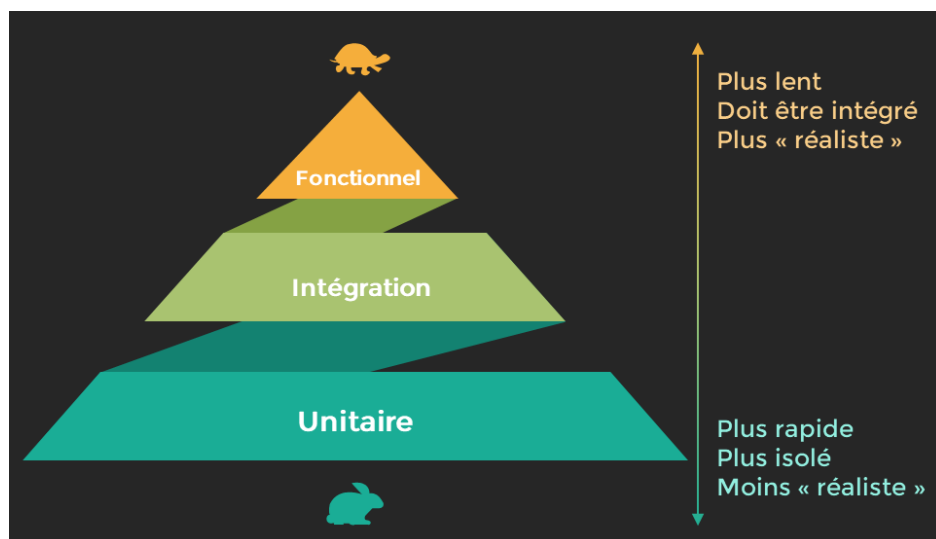
Pourquoi c'est important ?

Les tests sont une étape importante du processus de développement. Ils sont utilisés pour vérifier la qualité et la performance d'un logiciel. Ils consistent à exécuter le code avec des entrées spécifiques et à vérifier que les résultats obtenus en sortie correspondent bien aux attentes.

Ils permettent principalement de :

- Détecter des erreurs ou des comportements indésirables dans le code. C'est un moyen d'améliorer la fiabilité de l'application en identifiant les problèmes dans un environnement de test avant qu'ils ne surviennent en production.
- Ils permettent de vérifier que des fonctionnalités fonctionnent correctement et répondent aux exigences définies et que l'application se comporte comme prévu.
- Ils permettent de maintenir le code en détectant rapidement les changements de comportement lorsque des modifications sont apportées au code.
- Ils permettent de réduire les coûts car en détectant les erreurs tôt dans le processus de développement, les tests évitent des coûts ultérieurs de correction.
- D'une manière générale, ils réduisent les risques liés à l'utilisation de l'application en offrant qualité et stabilité.

Types de tests



Pyramide des tests

La pyramide des tests est un concept qui présente la répartition idéale des types de tests en fonction de leur coût et de leur vitesse d'exécution.

On trouve généralement 3 niveaux qui représentent chacun un des principaux types de tests :

- Les tests unitaires : Ce sont des tests qui ont pour objectif de vérifier le bon fonctionnement d'unités de code telles que des fonctions, des classes, des méthodes ... Ils constituent la base de la pyramide car ils sont très rapides à mettre en place, peu coûteux. Ce sont les tests les plus nombreux.
- Les tests d'intégration : Ils vérifient la bonne communication entre différents éléments de l'application, par exemple entre des services ou des composants. Ils peuvent impliquer plus de ressources et de temps que les tests unitaires.
- Les tests fonctionnels : Ils simulent le parcours de l'utilisateur au sein de l'application en reproduisant des scénarios réels. Ils s'assurent que l'intégralité de l'application fonctionne harmonieusement et répondent à toutes les exigences fonctionnelles. Ce sont les tests qui sont les plus longs et donc les plus coûteux car complexes à mettre en place.

Le Framework de tests PHPUnit

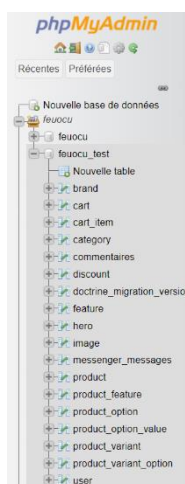
Dans le cadre de mon projet Symfony, j'ai utilisé le Framework de test open-source PHPUnit. C'est le Framework PHP le plus utilisé pour les tests unitaires et il est très largement utilisé pour tester des applications Symfony.

Il est parfaitement intégré à l'écosystème de Symfony et il est pris en charge par les outils de test (commandes) du système. Il dispose d'une excellente documentation qui présente sa syntaxe simple et explicite. En plus de sa documentation il dispose d'une vaste communauté d'utilisateurs, ce qui permet de trouver énormément de contenu et de faciliter l'apprentissage.

Ce Framework est donc un choix naturel pour les tests dans Symfony

L'installation

J'ai dû installer le package grâce à composer.



```
composer require --dev symfony/test-pack
```

Puis j'ai dû configurer un environnement de test pour effectuer les tests. J'ai donc ajouté DATABASE_URL à un fichier .env.test.local et j'ai relancé la création d'une base de données en spécifiant à Doctrine que cette fois la commande serait à exécuter pour l'environnement de test.

```
0
```

Par la suite j'ai lancé une migration vers la base de données nouvellement créée.

```
php bin/console doctrine:migrations:migrate --env=test
```

Cela m'a permis d'obtenir une copie de ma base de données qui ne serait utilisée que pour les tests.

Copie conforme de la
base de données pour
l'environnement de test

Réalisation de test unitaires

```
public function hashPassword(?string $passwordToHash, User $user):  
string
```

```

{
    if (empty($passwordToHash)) {
        throw new InvalidArgumentException('Le mot de passe ne peut pas être vide.');
```

Méthode hashPassword() de UserService.php

Je voudrais tester le fonctionnement de la méthode ci-dessous.

Grâce à la console de Symfony j'exécute la commande suivante qui ouvre l'utilitaire de création de Test.

```

PS C:\Users\cylb\Desktop\DEV\feu0cul> php bin/console make:test

Which test type would you like?:
[TestCase] basic PHPUnit tests
[KernelTestCase] basic tests that have access to Symfony services
[WebTestCase] to run browser-like scenarios, but that don't execute JavaScript code
[ApiTestCase] to run API-oriented scenarios
[PantherTestCase] to run e2e scenarios, using a real-browser or HTTP client and a real web server
```

Je sélectionne l'option TestCase et choisis de nommer ma classe de Test HashPasswordTest.

Un fichier de test HashPasswordTest.php est donc créé

```

<?php

namespace App\Tests;

use PHPUnit\Framework\TestCase;

class HashPasswordTest extends TestCase
{
    public function testSomething(): void
    {
        $this->assertTrue(true);
    }
}
```

Puisque je veux tester si ma méthode fonctionne, j'ai configuré des constantes contenant des mots de passe valides et invalides pour tester tous les scénarios. J'ai aussi créé des constantes contenant les messages d'erreur des exceptions levées dans la méthode afin de pouvoir comparer lors de mes tests si les messages correspondent.

```

private const VALID_PASSWORD = '1234!a';
private const EMPTY_PASSWORD = '';
private const EMPTY_MESSAGE = 'Le mot de passe ne peut pas être vide.';
```

```
private const TOO_SHORT_PASSWORD = '1a!';
private const TOO_SHORT_MESSAGE = 'Le mot de passe doit contenir au minimum
6 caractères';
private const REGEX_FAIL_PASSWORD = 'azert1';
private const REGEX_FAIL_MESSAGE = 'Le mot de passe doit contenir au moins
une lettre, un chiffre et un caractère spécial.';
```

Puis j'ai décidé de tester 4 scénarios possibles en créant des méthodes adaptées.

Le cas où tout va bien

```
public function testValidPassword(): void
{
    $user = new User();

    $passwordHashMock=$this->createMock(UserPasswordHasherInterface::class);
    $passwordHashMock->expects($this->once())
        ->method('hashPassword')
        ->with($user, self::VALID_PASSWORD)
        ->willReturn('success');
    $userRepositoryMock = $this->createMock(UserRepository::class);
    $validatorMock = $this->createMock(ValidatorInterface::class);
    $userService = new UserService($passwordHashMock, $userRepositoryMock,
    $validatorMock);

    $hashedPassword = $userService->hashPassword(self::VALID_PASSWORD,
    $user);
    $this->assertEquals('success', $hashedPassword);
}
```

Pour tester mon code, j'utilise des Mocks qui sont des objets reproduisant le comportement d'objets réels en environnement de test. Ils permettent d'isoler les dépendances externes et de se concentrer uniquement sur le comportement de l'unité de code testée. Ils permettent également de simuler des conditions spécifiques.

La méthode originale que je souhaite tester utilise une dépendance de la classe `UserPasswordHasherInterface` pour pouvoir accéder à la méthode `hashPassword` qui prend en paramètre un utilisateur ainsi qu'un mot de passe à hacher et retourne le hash en utilisant les algorithmes de hachage prédéfinis par Symfony.

```
$passwordHashMock = $this->createMock(UserPasswordHasherInterface::class);
$passwordHashMock->expects($this->once())
    ->method('hashPassword')
    ->with($user, self::VALID_PASSWORD)
    ->willReturn('success');
```

Ce morceau de code me permet de définir un objet qui va simuler le comportement original de la fonction de hachage. Au lieu de calculer puis de retourner un hash, lorsque cet objet sera appelé, il retournera la chaîne de caractère 'success'.

Après avoir créé différents mocks, je peux créer une instance de la dépendance `UserService` en lui passant les mocks comme paramètres de son constructeur.

```
$hashedPassword = $userService->hashPassword(self::VALID_PASSWORD, $user);
$this->assertEquals('success', $hashedPassword);
```

J'appelle ensuite la fonction du service pour hacher le mot de passe valide que j'avais défini lors de la création de mes constantes. Comme je sais à l'avance que ce mot de passe est valide, je m'attends à ce que le hachage renvoie la chaîne 'success' et j'utilise la fonction `assertEqual` fournie par PHPUnit pour comparer 2 valeurs. Le test sera validé si le hash créé est bien égal à 'success'.

Les cas où tout va mal

```
public function testTooShortPassword(): void
{
    $user = new User();

    $passwordHashMock = $this->createMock(UserPasswordHasherInterface::class);
    $userRepositoryMock = $this->createMock(UserRepository::class);
    $validatorMock = $this->createMock(ValidatorInterface::class);
    $userService = new UserService($passwordHashMock, $userRepositoryMock, $validatorMock);

    $this->expectException(\InvalidArgumentException::class);
    $this->expectExceptionMessage(self::TOO_SHORT_MESSAGE);
    $userService->hashPassword(self::TOO_SHORT_PASSWORD, $user);
}
```

Comme pour le cas précédent, j'ai créé une méthode pour tester le comportement de mon unité de code lorsque le mot de passe transmis est trop court.

```
$this->expectException(\InvalidArgumentException::class);
$this->expectExceptionMessage(self::TOO_SHORT_MESSAGE);
```

Ces 2 lignes me permettent d'indiquer respectivement que

- J'attends la levée d'une exception de type `\InvalidArgumentException` qui est l'exception normalement levée lorsque le mot de passe passé en paramètre est trop court.
- Je m'attends à ce que le message lié à cette exception soit celui que j'ai défini en constante de classe.

Dans ces deux cas, le test sera considéré comme réussi si l'exécution de la fonction échoue.

Je réalise des méthodes similaires pour tester le cas où le mot de passe est vide et où celui-ci enfreint la contrainte Regex.

J'exécute ensuite la commande qui lance l'exécution de mes tests. Grâce à l'option `--testdoxx` je dispose d'un rapport de test plus visible.

```
php bin/phpunit tests/HashPasswordTest.php --testdoxx
```

```
PS C:\Users\cylb\Desktop\DEV\feu0cu1> php bin/phpunit tests/HashPasswordTest.php --testdoxx
✓ Valid password
✓ Empty password
✓ Too short password
✓ Regex fail password

Time: 00:00.048, Memory: 8.00 MB

OK (4 tests, 8 assertions)
PS C:\Users\cylb\Desktop\DEV\feu0cu1> 
```

Les 4 tests sont réussis

Test	Attendu	Résultat
La fonction tente de hacher un mot de passe valide.	La fonction hache sans problème et le hash attendu est "success".	OK
Tentative de hachage d'un mot de passe vide.	L'exécution s'arrête car une exception de type <code>\InvalidArgumentException</code> est levée.	OK
	Le message de l'exception est "Le mot de passe ne peut pas être vide."	OK
Tentative de hachage d'un mot de passe trop court	L'exécution s'arrête car une exception de type <code>\InvalidArgumentException</code> est levée.	OK
	Le message de l'exception est "Le mot de passe doit contenir au minimum 6 caractères"	OK
Tentative de hachage d'un mot de passe qui enfreint la règle Regex.	L'exécution s'arrête car une exception de type <code>\InvalidArgumentException</code> est levée.	OK
	Le message de l'exception est "Le mot de passe doit contenir au moins une lettre, un chiffre et un caractère spécial."	OK

Usage de l'anglais et résolution de problèmes

While developing my controllers I figured out that there were a few exceptions that were common to many routes. One of those exceptions was the `\NotFoundException` occurring when the user tries to access a non-existent route, or when he tries to access a route providing a wrong argument in the URL.

For example,

/shop/product/vibromasseur-romp-beat6454e9903f1f0 is a route that should take the user to an existing product based on its slug.

If the user tries to access the following, with a non-existing slug, then a `\NotFoundException` will be thrown and Symfony will display an error message.

/shop/product/non-existing-product


The same goes for any non-existing route like **/nowhere/free**.

The problem I was facing was that sometimes there was no way for me to catch that one exception to handle it from inside my controller. Moreover, I was duplicating a lot of code handling every single `\NotFoundException` the same way by redirecting the user to the homepage and displaying a flash message indicating why he's been redirected.

So I went in search for a way to be able to handle `\NotFoundException` on a larger scale within my application as I was duplicating a lot of code

That is when I learned about EventListeners while browsing stackoverflow.com which is a popular online community that provides a question-and-answer platform for developers, I read a lot of posts and got redirected directly to Symfony's documentation.

Events and Event Listeners

 [Edit this page](#)

During the execution of a Symfony application, lots of event notifications are triggered. Your application can listen to these notifications and respond to them by executing any piece of code.

Symfony triggers several [events related to the kernel](#) while processing the HTTP Request. Third-party bundles may also dispatch events, and you can even dispatch [custom events](#) from your own code.

All the examples shown in this article use the same `KernelEvents::EXCEPTION` event for consistency purposes. In your own application, you can use any event and even mix several of them in the same subscriber.


As explained in this paragraph, Event Listeners are special classes designed to listen for event notifications and respond to them accordingly.


Following the documentation, I've been able to design an Event Listener for `\NotFoundException` but I was faced with another problem. To set up a custom flash message, I needed to access the Session as that's where flash messages are stored.

But to my knowledge, the only way to access the Session was from a Request Object I didn't have access to from my custom Event Listener Class.

How should one get Session object in a controller on Symfony 5.3?

Asked 1 year, 11 months ago Modified 1 year, 11 months ago Viewed 2k times

**Codez. Expérimentez. Créez.**
Développez continuellement de nouvelles compétences et expérimentez avec Azure



[Report this ad](#)

I'm a little confused as to the "right" way to retrieve the current session object within a controller in Symfony 5.3. The [current documentation](#) says to typehint an argument as `SessionInterface`. However, this [Symfony blog post](#) says `SessionInterface` has been deprecated in favor of using `RequestStack`.

In services it's clear that I should inject `RequestStack` and call `$requestStack->getSession()`. However, in most controller methods I'm already injecting the `Request` object which also has a `getSession()` method that seems to work.

Is it okay to get the session from the `Request` object, or should I inject `RequestStack` in addition to `Request` in my controller methods (this feels like duplication almost)

The O

Ké
m

Bl
(E

Featur

W
ar

AI
Di

Te

TH

St

I found a post on stackoverflow again that didn't really matched my issue at all as it was about a user dealing with a controller issue on an older Symfony version. What is interesting is that I discovered RequestStack and by looking for that key word in the official documentation, I learned that it is a Symfony Service that can provide a direct access to the Session object which totally solved my issue.

```
public function onKernelException(ExceptionEvent $event): void
{
    $exception = $event->getThrowable();
    if ($exception instanceof NotFoundException) {
        $homepageRoute = $this->urlGenerator->generate('app_home', [],
        UrlGeneratorInterface::ABSOLUTE_URL);
        $this->requestStack->getSession()->getFlashBag()->add('error',
        'Cette page n\'existe pas');
        $response = new RedirectResponse($homepageRoute);

        $event->setResponse($response);
    }
}
```

After defining my new Listener in my services.yaml, every single time a `\NotFoundException` occurs on my website, the user gets redirected to the homepage and is given the right message.

Veille informatique en matière de sécurité

Lors du développement d'un site web, il est très important de prendre en compte l'aspect sécuritaire de l'application que nous développons. Il en va de la protection des utilisateurs ainsi que des données sensibles. En effet aucun système n'est infallible et il est donc indispensable de se prémunir au maximum contre de potentielles attaques d'individus malintentionnés.

Pour se tenir à jour en matière de failles de sécurité, il convient de suivre l'actualité des organismes de sécurité tels que les CERT (Computer Emergency Response Team) qui sont des centres d'alerte et de réaction aux attaques informatiques dont les informations sont accessibles à tous. En France, le CERT-FR est l'organisme gouvernemental qui dépend de l'ANSSI (Agence Nationale de la Sécurité des Systèmes d'Information) occupe ce rôle.

Par ailleurs, l'OWASP (Open Worldwide Application Security Project) est une organisation à but non-lucratif qui œuvre à l'amélioration globale de la sécurité des logiciels. Dans ce but elle répertorie et publie un registre des failles critiques de sécurité ainsi que les mesures préventives pour s'en prémunir.

Injectons

Les injections SQL et les injections XSS représentent une faille de sécurité très courante. La première consiste à injecter du code malveillant dans une requête SQL. L'utilisateur malveillant manipule alors la requête pour qu'elle renvoie des données selon son désir ou qu'elle effectue une action non-autorisée voire malveillante. L'injection XSS quant à elle consiste à injecter du code dans une page web qui sera exécuté lorsque la page en question sera interprétée par le navigateur d'un utilisateur souhaitant la consulter.

Dans les deux cas, il convient de valider les input²² en filtrant toutes les données entrées par les utilisateurs avant de les utiliser. Les contraintes de validations du composant Form de Symfony permettent d'arriver à cette fin. Cette opération vise à garantir que ces données sont sûres et n'entraîneront aucune opération malveillante lors de leur utilisation. Les requêtes SQL qui utiliseront ces données devront être paramétrées, cela signifie que les données de l'utilisateur seront traitées comme des paramètres de la requête et ne seront pas directement concaténés dans celle-ci, empêchant ainsi d'injecter du code malveillant.

L'ORM Doctrine protège des injections en utilisant des requêtes paramétrées, ce qui protège des injections SQL.

Pour se protéger des injections XSS, le moteur de templates Twig permet d'échapper les caractères spéciaux grâce à des filtres et évite ainsi l'injection de code malveillant dans les pages.

²² Données entrées par l'utilisateur

Exemple d'injection SQL

Imaginons que le modèle de mes produits dispose d'une méthode qui exécute une requête SQL non paramétrée ressemblant à :

```
SELECT * FROM products WHERE product.name = '$input'
```

 Et dans laquelle \$input correspond à la saisie de l'utilisateur.

Un utilisateur mal intentionné pourrait remplacer \$input par du SQL afin de manipuler ma requête.

S'il saisit une chaîne telle que ' OR '1=1' ;-- qui est une des approches les plus connues, alors celle-ci remplace \$input et la requête exécutée devient :

```
SELECT * FROM products WHERE product.name = " OR '1=1' ;--".
```

La condition '1=1' est toujours vraie, ; permet de terminer la requête et -- permet de commenter le reste de la requête. On comprend facilement qu'il devient aisé d'ajouter d'autres instructions au code.

Exposition de données sensibles

Cette faille se réfère aux erreurs de conception liées au chiffrement. Ce type d'erreurs peut mener à une divulgation de données sensibles. Une des causes les plus courantes est une mauvaise utilisation des algorithmes de cryptographie, notamment à travers l'emploi d'algorithmes désuets. Il est indispensable de ne pas stocker en clair des données sensibles.

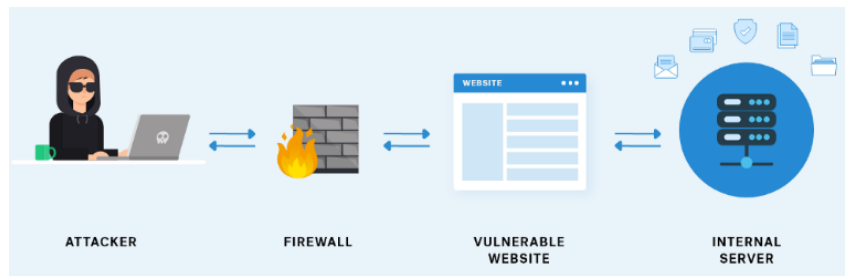
Dans le cas de mon application Symfony, le hachage des mots de passe est un des moyens mis en place pour lutter contre l'exposition aux données sensibles.

L'obtention d'un certificat SSL/TLS installé sur le serveur web sur lequel est hébergé le site permet l'utilisation de requêtes HTTPS qui sont chiffrées contrairement aux requêtes HTTP et offrent donc une sécurité lors des échanges de données entre un serveur et un client.

Server Side Request Forgery (SSRF)

Elle se traduit par falsification de requêtes côté serveur. C'est une attaque de sécurité durant laquelle un pirate peut manipuler ou exploiter les requêtes d'un serveur vulnérable en utilisant des protocoles réseau (HTTP, DNS...)

Elle permet au pirate de contourner les mesures de sécurité pour effectuer des requêtes indésirables vers d'autres ressources au nom du serveur vulnérable, il peut alors accéder à des services internes ou récupérer des données sensibles.



Pour s'en prémunir, il est possible de :

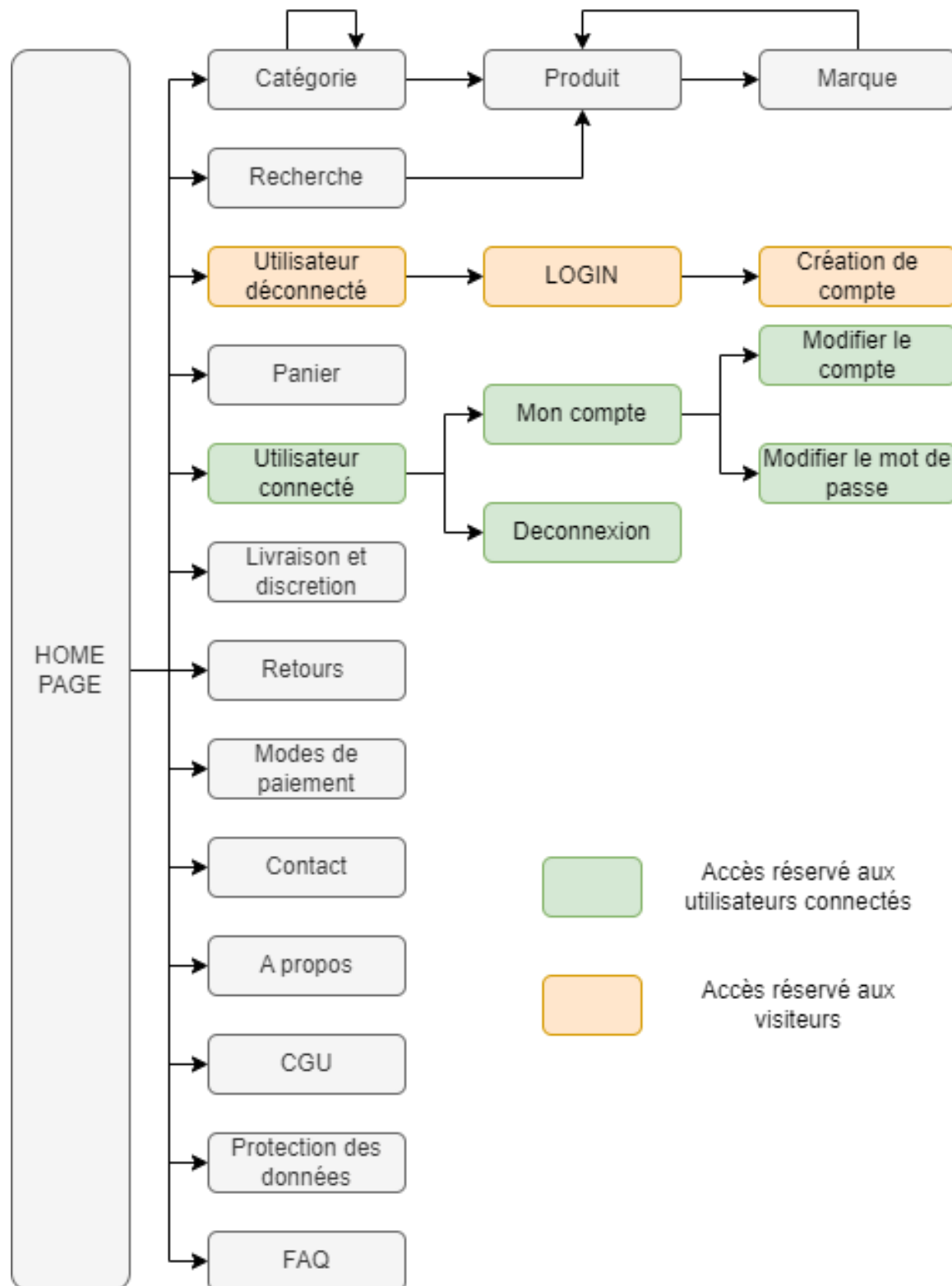
- Effectuer une validation stricte des URL en vérifiant que celles fournies par les utilisateurs sont valides et autorisées.
- Restreindre les ports non-autorisés
- Utiliser une liste blanche d'IP autorisées à accéder aux services internes
- Utiliser des tokens d'accès
- Sécuriser les serveurs en appliquant les correctifs de sécurité et en gardant à jour les services et les applications
- Utiliser des logs pour surveiller les activités suspects, détecter les requêtes sortantes et tentatives d'attaque SSRF

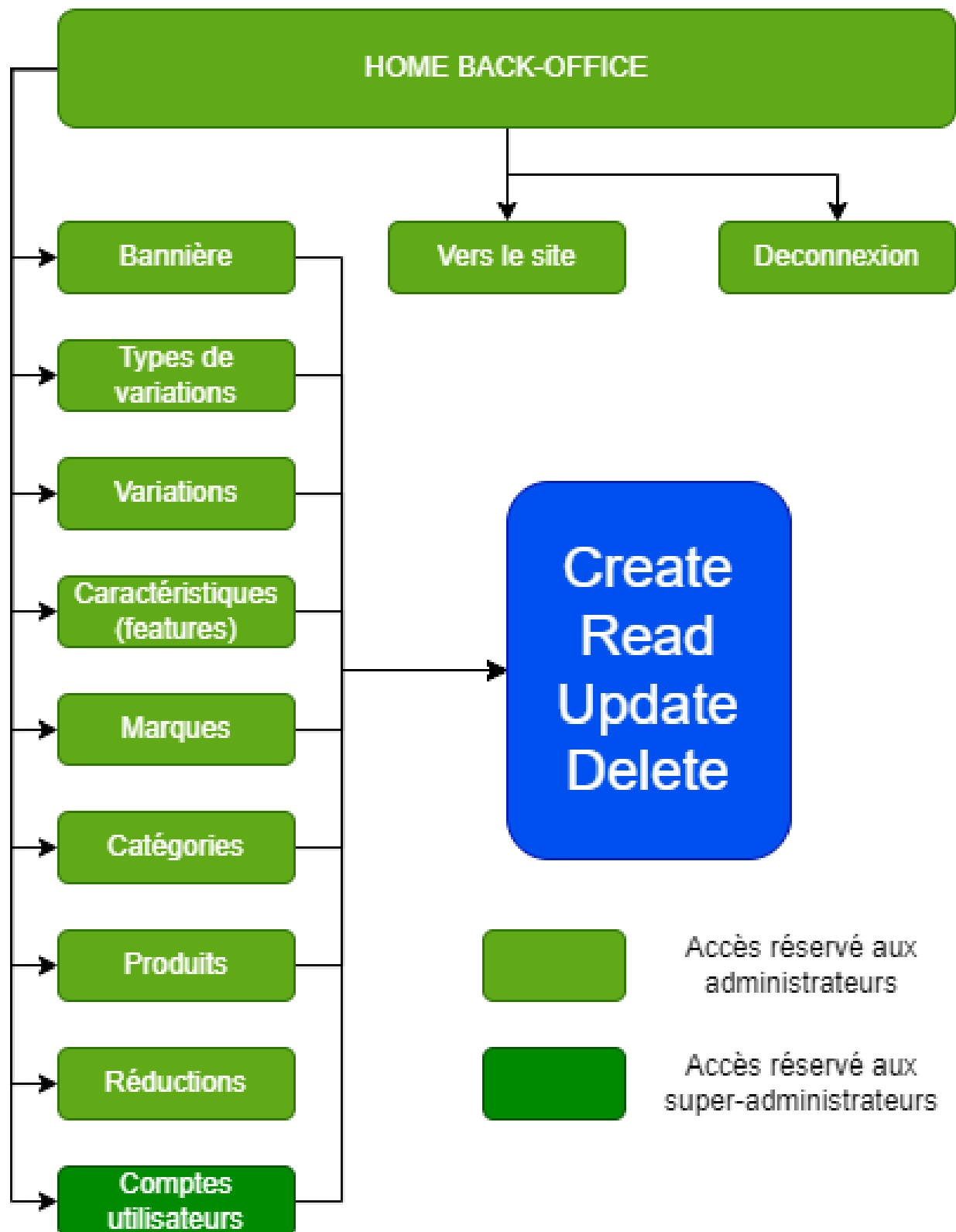
Conclusion

La réalisation de ce projet dans le cadre de ma formation a été une expérience enrichissante et motivante. Tout au long de la conception de ce site, j'ai dû faire preuve de persévérance et de détermination pour concevoir un site fonctionnel. Cette expérience m'a conforté dans l'idée que j'avais trouvé une voie me correspondant et que j'aimerais transformer cette passion nouvelle en opportunité professionnelle. J'ai fait face à de nombreuses difficultés qui ont su satisfaire mon appétit de connaissances et me stimuler intellectuellement grâce au travail de recherche constant nécessaire à la progression dans l'emploi des langages et technologies diverses. Je suis convaincu que cette voie est celle qui me correspond le mieux, que ce titre professionnel n'est que le début d'une aventure grâce à laquelle j'ai hâte de pouvoir continuer à me perfectionner.

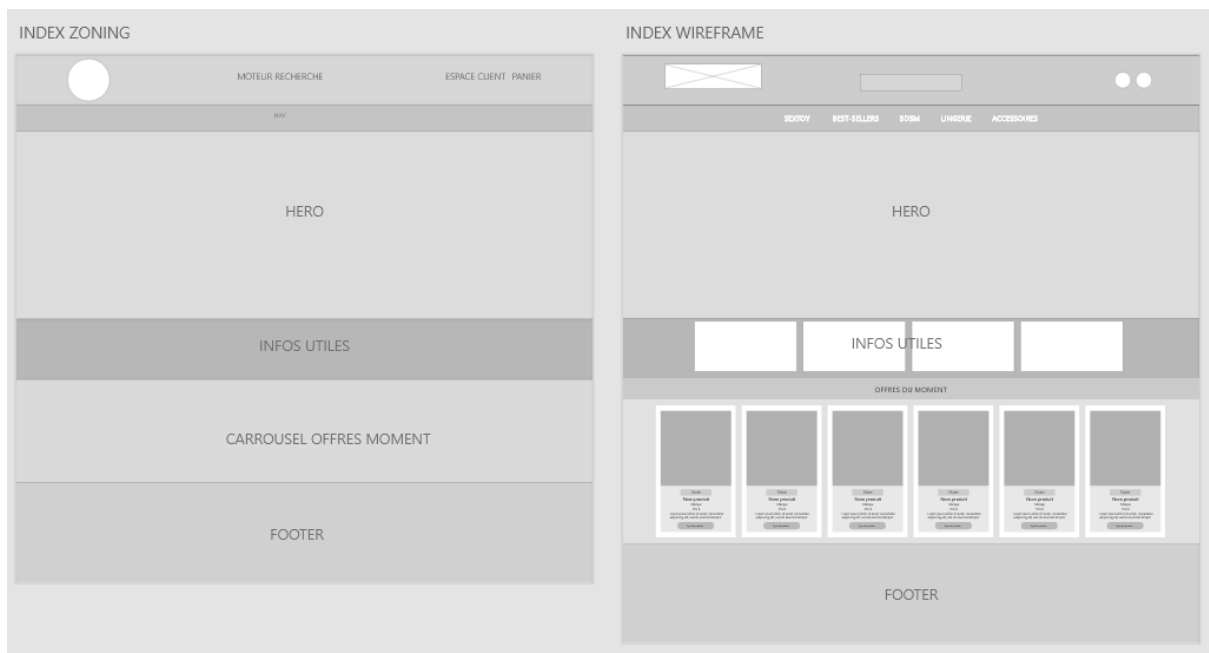
Annexes

Annexe 1 - Mapping du site	90
Annexe 2 - Mapping du back-office	91
Annexe 3 - Zoning et wireframe de la page d'accueil	92
Annexe 4 - Zoning et wireframe d'une page catégorie.....	93
Annexe 5 - Zoning et wireframe du login.....	94
Annexe 6 - Maquette page d'accueil et catégories mobile.....	95
Annexe 7 - Maquette Page d'accueil et catégories desktop	96
Annexe 8 - Extrait du fichier composer.json de mon projet	97
Annexe 9 - Diagramme UML de la base de données	98
Annexe 10 - Extrait du code de l'entité Produit	99
Annexe 11 - Structure de la table Product.....	100
Annexe 12 - Exemple de fixture permettant d'ajouter un utilisateur.....	101
Annexe 13 - base.html.twig	102
Annexe 14 - Composants de la page d'accueil.....	103
Annexe 15 - extrait de code de la méthode populateDom() de options_controller.js	104
Annexe 16 - code du contrôleur Stimulus add_cart_controller.....	105
Annexe 17 - Extrait de code de l'entité User	106
Annexe 18 - RegistrationController qui gère l'inscription des utilisateurs	107
Annexe 19 - Extrait du template de connexion de index/login.html.twig	109
Annexe 20 - Méthode createVariantsForm() du Service ProductService.php	110
Annexe 21 - Contrôleur Stimulus qui modifie la quantité d'un variant	111
Annexe 22 - Méthode getOptions() du contrôleur ShopController.php qui génère un fichier JSON contenant les données relatives aux variations d'un produit	112

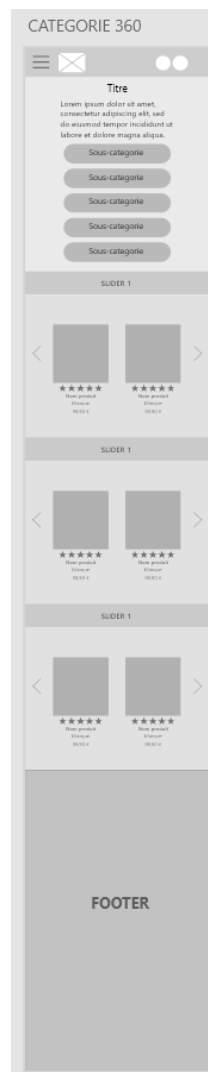
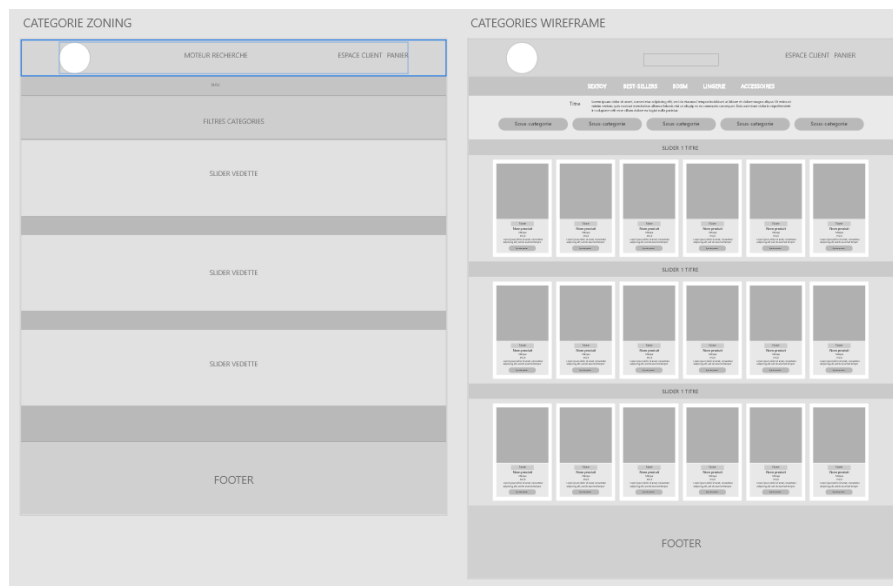




Annexe 3 - Zoning et wireframe de la page d'accueil



Annexe 4 - Zoning et wireframe d'une page catégorie

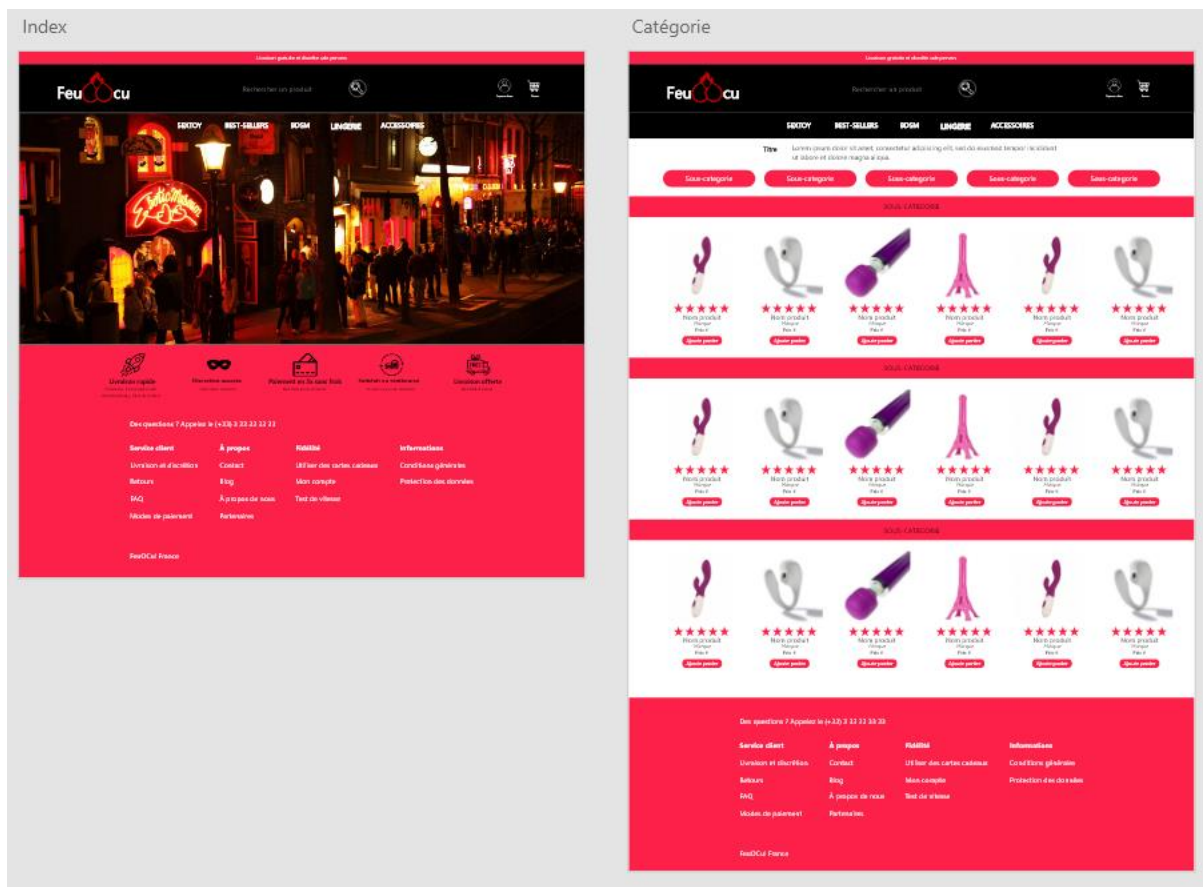


Annexe 5 - Zoning et wireframe du login

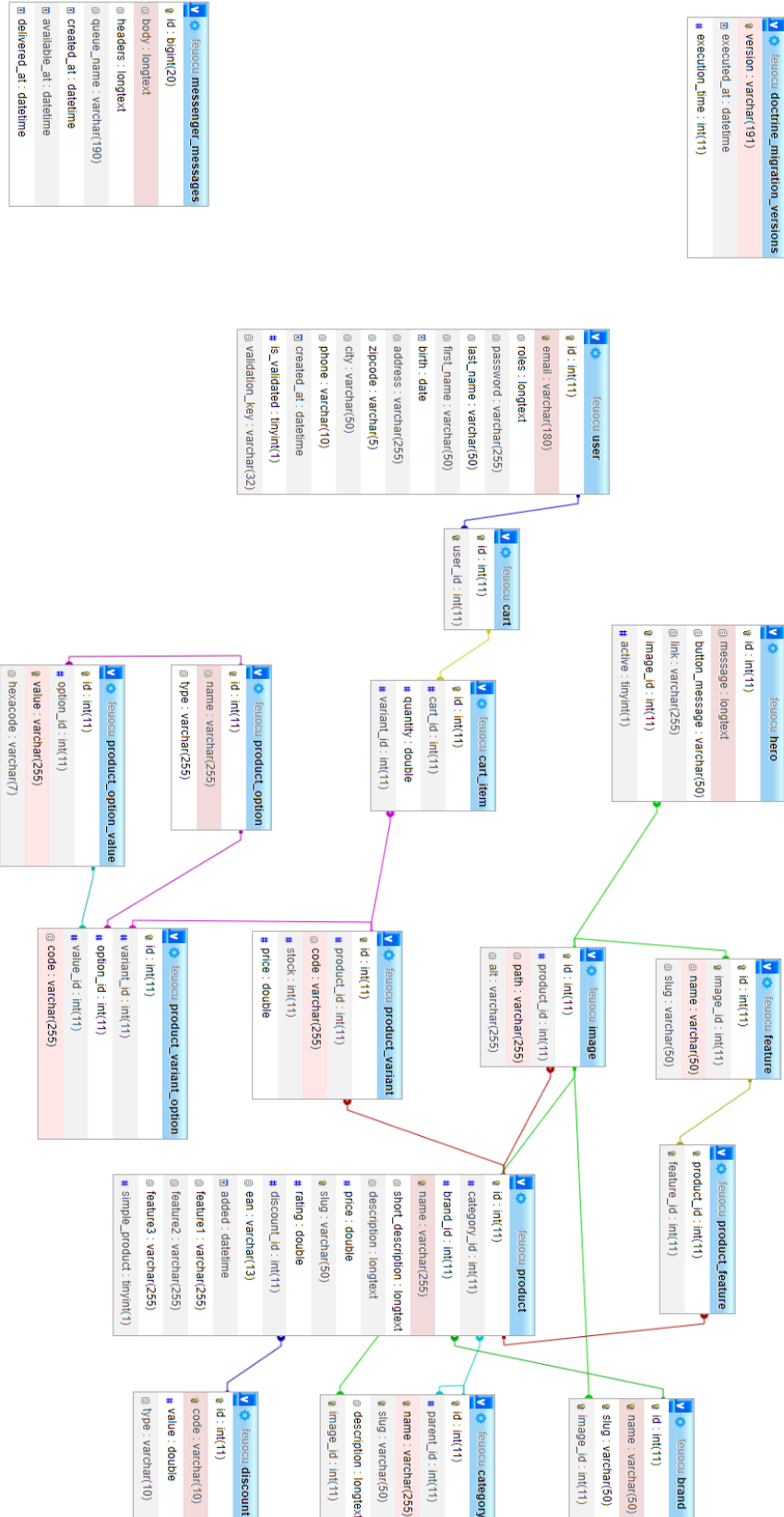




Annexe 7 - Maquette Page d'accueil et catégories desktop



```
{
  "type": "project",
  "license": "proprietary",
  "minimum-stability": "stable",
  "prefer-stable": true,
  "require": {
    "php": ">=8.1",
    "ext-ctype": "*",
    "ext-iconv": "*",
    "cocur/slugify": "^4.3",
    "doctrine/annotations": "^2.0",
    "doctrine/doctrine-bundle": "^2.8",
    "doctrine/doctrine-migrations-bundle": "^3.2",
    "doctrine/orm": "^2.14",
    "easycorp/easyadmin-bundle": "^4",
    "liip/imagine-bundle": "^2.10",
    "phpdocumentor/reflection-docblock": "^5.3",
    "phpstan/phpdoc-parser": "^1.16",
    "sensio/framework-extra-bundle": "^6.1",
    "symfony/asset": "6.2.*",
    "symfony/console": "6.2.*",
    "symfony/doctrine-messenger": "6.2.*",
    "symfony/dotenv": "6.2.*",
    "symfony/expression-language": "6.2.*",
    "symfony/flex": "^2",
    "symfony/form": "6.2.*",
    "symfony/framework-bundle": "6.2.*",
    "symfony/html-sanitizer": "6.2.*",
    "symfony/http-client": "6.2.*",
    "symfony/intl": "6.2.*",
    "symfony/mailer": "6.2.*",
    "symfony/mime": "6.2.*",
    "symfony/monolog-bundle": "^3.0",
    "symfony/notifier": "6.2.*",
    "symfony/password-hasher": "6.2.*",
    "symfony/process": "6.2.*",
    "symfony/property-access": "6.2.*",
    "symfony/property-info": "6.2.*",
    "symfony/runtime": "6.2.*",
    "symfony/security-bundle": "6.2.*",
    "symfony/serializer": "6.2.*",
    "symfony/string": "6.2.*",
    "symfony/translation": "6.2.*",
    "symfony/twig-bundle": "6.2.*",
    "symfony/validator": "6.2.*",
    "symfony/web-link": "6.2.*",
    "symfony/webpack-encore-bundle": "^1.16",
    "symfony/yaml": "6.2.*",
    "symfonycasts/verify-email-bundle": "^1.13",
    "twig/extra-bundle": "^2.12|^3.0",
    "twig/intl-extra": "^3.6",
    "twig/twig": "^2.12|^3.0"
  },
}
```



```

<?php

namespace App\Entity;

use App\Repository\ProductRepository;
use Doctrine\Common\Collections\ArrayCollection;
use Doctrine\Common\Collections\Collection;
use Doctrine\DBAL\Types\Types;
use Doctrine\ORM\Mapping as ORM;
use Symfony\Component\Validator\Constraints\Length;
use Symfony\Component\Validator\Constraints\NotBlank;
use Symfony\Component\Validator\Constraints\NotNull;
use Symfony\Component\Validator\Constraints\Positive;
use Symfony\Component\Validator\Constraints\Range;
use Symfony\Component\Validator\Constraints\Regex;

#[ORM\Entity(repositoryClass: ProductRepository::class)]
class Product
{
    #[ORM\Id]
    #[ORM\GeneratedValue]
    #[ORM\Column]
    private ?int $id = null;

    #[ORM\Column(length: 255, unique: true)]
    #[NotBlank(
        message: 'Le nom ne peut pas être vide.'
    )]
    #[Length(
        min: 3, max: 255, minMessage: 'Le nom est trop court, 3 caractères
minimum', maxMessage: 'Le nom est trop long, 255 caractères maximum'
    )]
    #[Regex(
        pattern: '/^[A-Za-z0-9\sÀ-ÿ]+$/',
        message: 'Le nom ne peut contenir que des lettres, des chiffres et
des espaces.'
    )]
    private ?string $name = null;

    #[ORM\Column(length: 50, unique: true)]
    private ?string $slug = null;

    #[ORM\OneToMany(mappedBy: 'product', targetEntity: Image::class,
cascade: ['persist', 'remove'])]
    private Collection $images;

    #[ORM\Column(type: Types::TEXT, nullable: true)]
    private ?string $short_description = null;

    #[ORM\Column(type: Types::TEXT, nullable: true)]
    private ?string $description = null;

    #[ORM\Column]
    #[NotNull]
    #[Positive]
    private ?float $price = null;

```


Annexe 11 - Structure de la table Product

<div> <div>Parcourir</div> <div>Structure</div> <div>SQL</div> <div>Rechercher</div> <div>Insérer</div> <div>Exporter</div> <div>Importer</div> <div>Privilèges</div> <div>Opérations</div> <div>Suivi</div> <div>Déclencheurs</div> </div>										
<div> <div>Structure de table</div> <div>Vue relationnelle</div> </div>										
	#	Nom	Type	Interclassement	Attributs	Null	Valeur par défaut	Commentaires	Extra	Action
<input type="checkbox"/>	1	id	int(11)			Non	Aucun(e)		AUTO_INCREMENT	Modifier Supprimer Plus
<input type="checkbox"/>	2	category_id	int(11)			Non	Aucun(e)			Modifier Supprimer Plus
<input type="checkbox"/>	3	brand_id	int(11)			Non	Aucun(e)			Modifier Supprimer Plus
<input type="checkbox"/>	4	name	varchar(255)	utf8mb4_unicode_ci		Non	Aucun(e)			Modifier Supprimer Plus
<input type="checkbox"/>	5	short_description	longtext	utf8mb4_unicode_ci		Oui	NULL			Modifier Supprimer Plus
<input type="checkbox"/>	6	description	longtext	utf8mb4_unicode_ci		Oui	NULL			Modifier Supprimer Plus
<input type="checkbox"/>	7	price	double			Non	Aucun(e)			Modifier Supprimer Plus
<input type="checkbox"/>	8	slug	varchar(50)	utf8mb4_unicode_ci		Non	Aucun(e)			Modifier Supprimer Plus
<input type="checkbox"/>	9	rating	double			Oui	NULL			Modifier Supprimer Plus
<input type="checkbox"/>	10	discount_id	int(11)			Oui	NULL			Modifier Supprimer Plus
<input type="checkbox"/>	11	ean	varchar(13)	utf8mb4_unicode_ci		Non	Aucun(e)			Modifier Supprimer Plus
<input type="checkbox"/>	12	added	datetime			Non	Aucun(e)	(DC2Type:datetime_immutable)		Modifier Supprimer Plus
<input type="checkbox"/>	13	feature1	varchar(255)	utf8mb4_unicode_ci		Oui	NULL			Modifier Supprimer Plus
<input type="checkbox"/>	14	feature2	varchar(255)	utf8mb4_unicode_ci		Oui	NULL			Modifier Supprimer Plus
<input type="checkbox"/>	15	feature3	varchar(255)	utf8mb4_unicode_ci		Oui	NULL			Modifier Supprimer Plus
<input type="checkbox"/>	16	simple_product	tinyint(1)			Non	Aucun(e)			Modifier Supprimer Plus

```
<?php

namespace App\DataFixtures;

use App\Entity\User;
use Doctrine\Bundle\FixturesBundle\Fixture;
use Doctrine\Persistence\ObjectManager;

class AppFixtures extends Fixture
{
    public function load(ObjectManager $manager): void
    {
        // USERS
        $user1 = new User();
        $user1->setEmail('foucaut.morgan@gmail.com')
        -
        >setPassword('$2a$12$WmzLEY/fMamOUP1cW2w5OeZh8gWtqpx6MTDKlQJ2Bnqzm0joW0LXy'
        )
            ->setRoles(['ROLE_SUPER_ADMIN'])
            ->setFirstName('Morgan')
            ->setLastName('Foucaut')
            ->setBirth(new \DateTime('1990-01-01'))
            ->setAddress('18 rue Paul Vaillant Couturier')
            ->setZipcode('57300')
            ->setCity('Hagondange')
            ->setPhone('0767603690')
            ->setCreatedAt(new \DateTimeImmutable())
            ->setIsValidated(true)
            ->setValidationKey(12);
        $manager->persist($user1);

        $manager->flush();
    }
}
```

```

<!DOCTYPE html>
<html>
    <head>
        <meta charset="UTF-8">
        <title>{% block title %}Bienvenue sur feuocu !{% endblock
%}</title>
        <link rel="shortcut icon" href="{{ asset('favicon-32x32.png') }}">
        <!-- Bootstrap Icons -->
        <link rel="stylesheet"
href="https://cdn.jsdelivr.net/npm/bootstrap-icons@1.3.0/font/bootstrap-
icons.css">
        <!-- Google Fonts -->
        <link rel="preconnect" href="https://fonts.googleapis.com">
        <link rel="preconnect" href="https://fonts.gstatic.com"
crossorigin>
        <link
href="https://fonts.googleapis.com/css2?family=Lobster&family=Playfair+Disp
lay&family=Playfair+Display+SC:ital,wght@1,700&family=Raleway&display=swap"
rel="stylesheet">
        <!-- SLICK -->
        <link rel="stylesheet" type="text/css"
href="//cdn.jsdelivr.net/npm/slick-carousel@1.8.1/slick/slick.css">
        <link rel="stylesheet" type="text/css"
href="//cdn.jsdelivr.net/npm/slick-carousel@1.8.1/slick/slick-theme.css">

        {% block stylesheets %}
            {{ encore_entry_link_tags('app') }}
        {% endblock %}

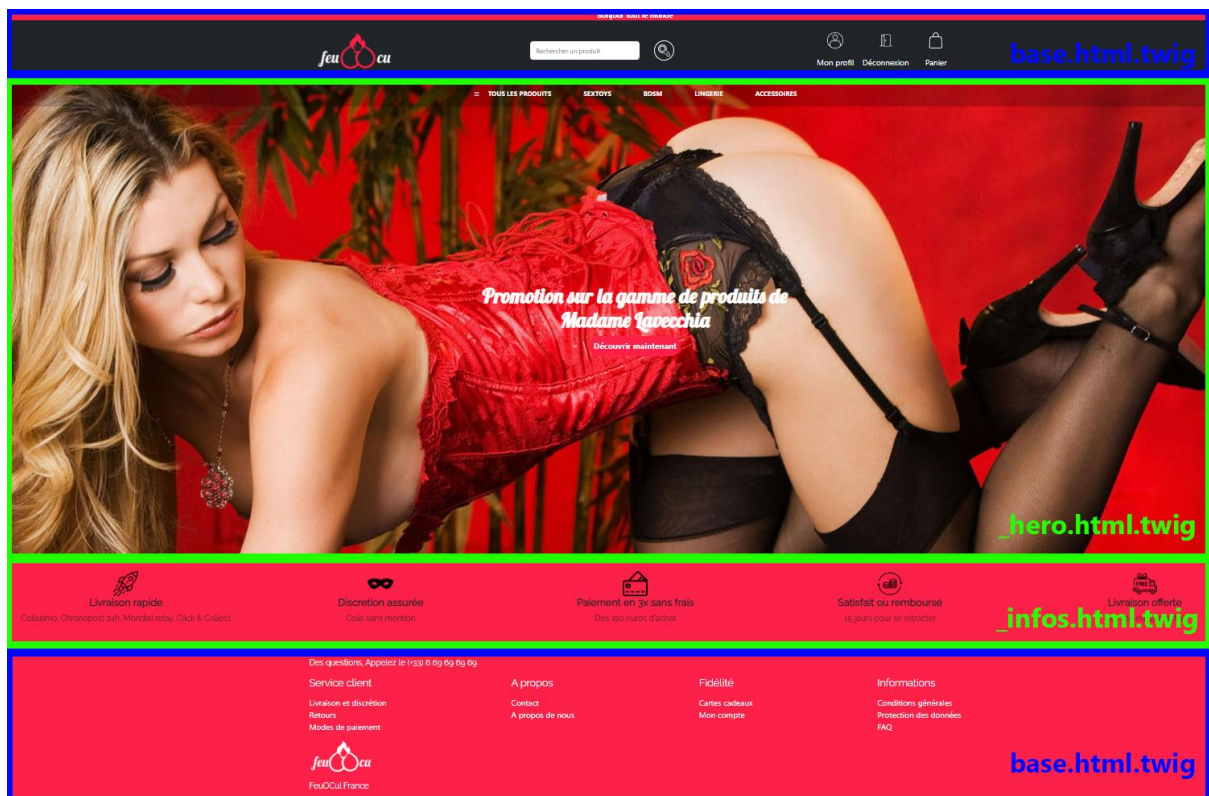
        {% block javascripts %}
            {{ encore_entry_script_tags('app') }}
        {% endblock %}
    </head>
    <body data-controller="toggle-sidebar-categories">
        {% set bcgColor = '#FD2048' %}
        {% set fontColor = 'white' %}
        {% include 'components/_textband.html.twig' with {'background':
bcgColor, 'color': fontColor, 'text': 'Bienvenue sur feuocu, votre
distributeur de plaisir'} %}
        {% include 'components/_sidebarcategories.html.twig' %}
        {% include 'components/header/_header.html.twig' %}

        {% block body %}{% endblock %}

        {% include 'components/_footer.html.twig' %}
        <!-- JQUERY -->
        <script
src="https://cdnjs.cloudflare.com/ajax/libs/jquery/3.6.3/jquery.min.js"></s
cript>
        <!-- SLICK -->
        <script type="text/javascript" src="//cdn.jsdelivr.net/npm/slick-
carousel@1.8.1/slick/slick.min.js"></script>
    </body>
</html>

```

Annexe 14 - Composants de la page d'accueil



```
if (optionType === 'color') {

  let inputColors = document.createElement('div');
  inputColors.setAttribute('id', `input_${optionName}s`);
  inputColors.classList.add('d-flex', 'flex-wrap');

  for (const key in this.tempData) {
    const optionData = optionList[key];

    let label = document.createElement('label');
    label.classList.add('me-3', 'gap-1');

    let input = document.createElement('input');
    input.setAttribute('type', 'radio');
    input.setAttribute('name', `input_${optionName}`);
    input.classList.add('me-1');
    input.setAttribute('value', optionData.value);
    input.setAttribute('required', true);

    let value = document.createElement('span');
    value.classList.add('me-1');
    value.innerHTML = optionData.value;

    let colorSpan = document.createElement('span');
    colorSpan.classList.add('d-inline-block', 'rounded-circle', 'me-1');

    colorSpan.style.backgroundColor = optionData.data;
    colorSpan.style.width = '25px';
    colorSpan.style.height = '25px';
    colorSpan.style.border = '1px solid lightgrey';

    label.appendChild(input);
    label.appendChild(value);
    label.appendChild(colorSpan);

    inputColors.appendChild(label);
  }
  optionField.appendChild(inputColors);
}
```

```

import {Controller} from "@hotwired/stimulus";

export default class extends Controller {
  static targets = ["quantityInput", "submitButton"];
  connect() {
  }

  addToCart() {
    event.preventDefault();
    const variantId = this.submitButtonTarget.dataset.variant;
    const quantity = this.quantityInputTarget.value;

    const flashContainer = document.getElementById('flash-messages');
    flashContainer.innerHTML = '';

    fetch(`/cart/add`, {
      method: 'POST',
      headers: {
        'Content-type': 'application/json'
      },
      body: JSON.stringify({variantId: variantId, quantity: quantity})
    })
    .then(response => {
      if(!response.ok) {
        throw new Error('Erreur ' + response.status);
      }
      return response.json();
    })
    .then(data => {
      if (data.success) {
        this.dispatch('added', {
          target: document.documentElement,
        });
      }
    })
    .catch(error => {
      const flashMessage = document.createElement('div');
      flashMessage.classList.add('alert', 'alert-danger');
      flashMessage.textContent = 'Erreur pendant l\'ajout. ' +
error.message;
      flashContainer.appendChild(flashMessage);
    });
  }
}

```

```

#[ORM\Entity(repositoryClass: UserRepository::class)]
#[UniqueEntity(fields: ['email'], message: 'Un compte avec cet email existe déjà.')]
class User implements UserInterface, PasswordAuthenticatedUserInterface
{
    #[ORM\Id]
    #[ORM\GeneratedValue]
    #[ORM\Column]
    private ?int $id = null;

    #[ORM\Column(length: 180, unique: true)]
    #[Length(
        min: 6, max: 50, minMessage: 'Le mail est trop court, 6 caractères minimum', maxMessage: 'Le mail est trop long, 180 caractères maximum'
    )]
    private ?string $email = null;

    #[ORM\Column]
    private array $roles = [];

    /**
     * @var string The hashed password
     */
    #[ORM\Column]
    #[NotBlank(
        message: 'Le mot de passe ne peut pas être vide.', allowNull: true
    )]
    private ?string $password = null;

    #[ORM\Column(length: 50)]
    #[NotBlank(
        message: 'T\'as pas de prénom ?'
    )]
    #[Length(
        min: 2, max: 50, minMessage: 'Ton prénom c\'est lettre ? Sérieux ?', maxMessage: 'C\'est pas un prénom ça, c\'est un roman, 50 caractères max'
    )]
    #[Regex(
        pattern: '/^[\\p{L}\\-]+(?!-)$/',
        message: 'Dans un prénom il n\'y a que des lettres et des fois un tiret pour séparer 2 lettres ...'
    )]
    private ?string $first_name = null;

```

```

class RegistrationController extends AbstractController
{
    private CategoryRepository $categoryRepository;
    private MailerInterface $mailer;
    private VerifyEmailHelperInterface $verifyEmailHelper;

    public function __construct(
        CategoryRepository $categoryRepository,
        MailerInterface $mailer,
        VerifyEmailHelperInterface $verifyEmailHelper
    )
    {
        $this->categoryRepository = $categoryRepository;
        $this->mailer = $mailer;
        $this->verifyEmailHelper = $verifyEmailHelper;
    }
    #[Route('/registration', name: 'app_registration')]
    public function register(
        Request $request,
        UserService $userService
    ): Response
    {
        if ($this->isGranted('IS_AUTHENTICATED_FULLY')) {
            $this->addFlash('error', 'On a plein de doubles-trucs ici mais
les double-comptes on en fait pas.');
```

les double-comptes on en fait pas.');

```

            return $this->redirectToRoute('app_home');
        }

        $user = new User();
        $categories = $this->categoryRepository->findMotherCategories();
        $form = $this->createForm(RegisterFormType::class, $user);
        $form->handleRequest($request);

        if ($form->isSubmitted() && !$form->isValid()) {
            foreach ($form->getErrors(true) as $error) {
                $this->addFlash('error', $error->getMessage());
            }
        }

        if ($form->isSubmitted() && $form->isValid()) {
            try {
                $userService->registerUser(
                    $user, $form['email']->getData(),
                    $form['emailConfirm']->getData(),
                    $form['password']->getData(), $form['passwordConfirm']->
                    >getData(), $form['birth']->getData());

                $signatureComponents = $this->verifyEmailHelper->
                >generateSignature(
                    'app_verify_user',
                    $user->getId(),
                    $user->getEmail()
                );
                $email = new TemplatedEmail();
                $email->from('morgan.foucaut@gmail.com');
                $email->to($user->getEmail());
                $email->htmlTemplate('registration/confirmation_email.html.twig');
                $email->context(['signedUrl' => $signatureComponents-

```



```

>getSignedUrl()]);
    $this->mailer->send($email);

    $this->addFlash('success', 'Votre compte a été créé avec
succès ! Nous vous avons envoyé un e-mail de confirmation avec un lien pour
activer votre compte. Veuillez vérifier votre boîte de réception et votre
dossier spam si vous ne trouvez pas l\'e-mail dans votre boîte de
réception.');
```

} catch (UniqueConstraintViolationException \$e) {
 \$this->addFlash('error', 'Cet email est déjà utilisé.');

return \$this->redirectToRoute('app_registration');

} catch (\Exception \$e) {
 \$this->addFlash('error', \$e->getMessage());
 return \$this->redirectToRoute('app_registration');

} catch (TransportExceptionInterface \$e) {
 \$this->addFlash('error', \$e->getMessage());
 return \$this->redirectToRoute('app_registration');

}

return \$this->redirectToRoute('app_login', [],
Response::HTTP_SEE_OTHER);

}

return \$this->render('registration/index.html.twig', [
 'categories' => \$categories,
 'form' => \$form,
]);

}

```
{{ form_start(form) }}

<div class="row">
  <div class="col-12 col-md-6 mx-auto">
    {{ form_label(form.email) }}
    {{ form_widget(form.email) }}
  </div>
</div>
<div class="row">
  <div class="col-12 col-md-6 mx-auto">
    {{ form_label(form.password) }}
    {{ form_widget(form.password) }}
  </div>
</div>
<div class="row">
  <a class="col-12 col-md-6 mt-2 mx-auto" href="#">Mot de passe oublié
?</a>
</div>
<div class="row">
  <input type="hidden" name="_csrf_token" value="{{
csrf_token('authenticate') }}">
</div>
<div class="row">
  {{ form_widget(form.submit) }}
</div>

{{ form_end(form) }}
```

```

public function createVariantsForm(FormBuilderInterface $formBuilder,
Product $product): void
{
    // Ajout d'une checkbox pour gérer les produits sans variants
    $formBuilder->add('no_variant', CheckboxType::class, [
        'label' => 'Produit sans variants',
        'required' => false,
        'data' => $product->isSimpleProduct(),
        'attr' => [
            'data-admin-product-variant-options-target' => 'noVariant',
            'data-action' => 'admin-product-variant-options#toggle'
        ]
    ]);
    // Création des choiceType pour tous les types d'options et leurs
options respectives
    $options = $this->productOptionRepository->findAll();
    foreach ($options as $option)
    {
        $optionValues = $this->productOptionValueRepository->findBy([
            'option' => $option->getId()
        ]);

        if (!empty($optionValues)) {
            $formBuilder
                ->add('Option' . $option->getId(), ChoiceType::class, [
                    'label' => $option->getName(),
                    'choices' => $optionValues,
                    'choice_label' => function ($optionValue) {
                        return $optionValue->getValue();
                    },
                    'row_attr' => [
                        'class' => 'variant-option'
                    ],
                    'multiple' => true,
                    'expanded' => true,
                ]
            );
        }
    }
}

```

```

import {Controller} from "@hotwired/stimulus";

export default class extends Controller {
  connect() {
    this.element.addEventListener('change', event => {
      const variantId = this.element.dataset.variantId;
      const quantity = parseInt(this.element.value);

      const flashContainer = document.getElementById('flash-
messages');
      flashContainer.innerHTML = '';

      fetch('/admin/product/variant/update/stock', {
        method: 'POST',
        headers: {
          'Content-type' : 'application/json'
        },
        body: JSON.stringify({variantId: variantId, quantity:
quantity})
      })
        .then(response => {
          if (!response.ok) {
            throw new Error('Fail: ' + response.status);
          }
          return response.json();
        })
        .then(data => {
          if (data.success) {
            const flashMessage = document.createElement('div');
            flashMessage.classList.add('alert', 'alert-
success');

            flashMessage.textContent = 'Quantité mise à jour';
            flashContainer.appendChild(flashMessage);
          } else {
            const message = data.message;
            const flashMessage = document.createElement('div');
            flashMessage.classList.add('alert', 'alert-
danger');

            flashMessage.textContent = message;
            flashContainer.appendChild(flashMessage);
          }
        })
        .catch(error => {
          console.error('Erreur: ', error);
        })
    });
  }
}

```

Annexe 22 - Méthode `getOptions()` du contrôleur `ShopController.php` qui génère un fichier JSON contenant les données relatives aux variations d'un produit

```
#[Route('/shop/product/{id}/options', name: 'get_product_options')]
public function getOptions(
    $id,
    ProductRepository $productRepository,
    ProductVariantRepository $productVariantRepository
): JsonResponse
{
    $product = $productRepository->find($id);
    if (!$product) {
        throw $this->createNotFoundException('Ce produit n\'existe pas');
    }

    $productVariants = $productRepository->getAvailableVariants($product);

    $productData = [
        'id' => $id,
        'price' => $product->getPrice(),
        'discount' => $product->getDiscount() ? [
            'type' => $product->getDiscount()->getType(),
            'val' => $product->getDiscount()->getValue()
        ] : null
    ];
    $options = [];
    $variants = [];

    if (count($productVariants) > 1) {
        foreach ($productVariants as $productVariant) {
            $productVariantOptions = $productVariant->
>getProductVariantOptions();
            $variantData = ['id' => $productVariant->getId()];
            foreach ($productVariantOptions as $productVariantOption) {
                $productOptionValue = $productVariantOption->getValue();
                $productOption = $productOptionValue->getOption();
                $optionName = $productOption->getName();
                $optionType = $productOption->getType();

                if (!isset($options[$optionName])) {
                    $options[$optionName] = [
                        '_meta' => [
                            'name' => $optionName,
                            'type' => $optionType,
                        ],
                        'list' => [],
                    ];
                }
                $optionvalue = $productOptionValue->getValue();

                if (!isset($options[$optionName]['list'][$optionvalue])) {
                    $options[$optionName]['list'][$optionvalue] = [
                        'value' => $optionvalue,
                    ];

                    if ($optionType === 'color') {
                        $options[$optionName]['list'][$optionvalue]['data']
= $productOptionValue->getHexacode();
                    }
                }
            }
        }
    }
}
```

```

        }
        $variantData[$optionName] = $optionvalue;
    }

    $temp = &$variants;
    foreach ($variantData as $key => $value) {
        if ($key === 'id') {
            continue;
        }

        if (!isset($temp[$value])) {
            $temp[$value] = [];
        }

        $temp = &$temp[$value];
    }

    $temp = [
        'id' => $variantData['id'],
        'stock' => $productVariant->getStock(),
        'price' => $productVariant->getPrice() ? $productVariant-
>getPrice() : null
    ];
}

}

if (count($productVariants) === 1) {
    $variants = [
        'id' => $productVariants[0]->getId(),
        'stock' => $productVariants[0]->getStock(),
        'price' => $productVariants[0]->getPrice() ?
$productVariants[0]->getPrice : null
    ];
}

return new JsonResponse(['prod' => $productData, 'options' => $options,
'variants' => $variants]);
}

```