www.isl.be

INSTITUT SAINT-LAURENT
enseignement de promotion sociale

rue Saint-Laurent 33 - 4000 LIEGE
04 223 11 31

Travail de fin d'études

Soap-Opera

Cahier des charges pour le travail de fin d'études présenté par

Niessen Alain

En vue de l'obtention du brevet de l'enseignement supérieur de

WebDeveloper

Enseignement supérieur économique de promotion sociale et de type court

Année académique 2021-2022

Table de matière

1 Introduction

1.1 Présentation du projet

Le projet **Soap-Opera** a été créé par Sarah et Julia. Non seulement leur amitié les uni, mais aussi la passion pour l'artisanat.

Le but du projet est de développer pour la peau et les cheveux un nouveau produit, qui soit, non seulement, durable et respectueux de l'environnement, mais qui favorise également la régénération naturelle de la peau et des cheveux.

Après de nombreuses tentatives et beaucoup d'efforts, Sarah et Julia sont maintenant fières de présenter leurs produits.

Dans la production, Sarah et Julia utilisent une sélection d'eau florale de qualité supérieure, d'huiles végétales vierges, d'herbes aromatiques et également des huiles essentielles qui aident à rétablir l'équilibre naturel de la peau et des cheveux. La promesse : tous les produits sont durables, respectueux de l'environnement, artisanaux et certifiés. La plupart des produits sont vegan et étiquetés en conséquence.

Sarah et Julia fabriquent leurs savons à froid à partir de produits frais et de qualité. En petite quantité et en utilisant un procédé de fabrication à l'ancienne, très élaboré, elles veulent obtenir les meilleurs résultats. Elles proposent des savons avec différents taux de surgraissement. Ce dernier va limiter le dessèchement de la peau et facilitera la reconstitution du film hydrolipidique pour protéger ainsi la peau. Le savon est donc nourrissant.

Les produits de soins diversifiés sont, également, fabriqués avec des huiles et des graisses de première qualité. Dans la production, la qualité et la transparence sont mises en avant. La promesse: ingrédients naturels qui sont un pur bienfait pour le corps et les cheveux.

1.2 Contexte du travail

Le contexte du travail est la création d'un site avec une demande réelle, qui cible toutes les personnes qui ont un intérêt pour l'artisanat, des produits naturels et respectueux de l'environnement.

2 Objectifs généraux

2.1 Introduction

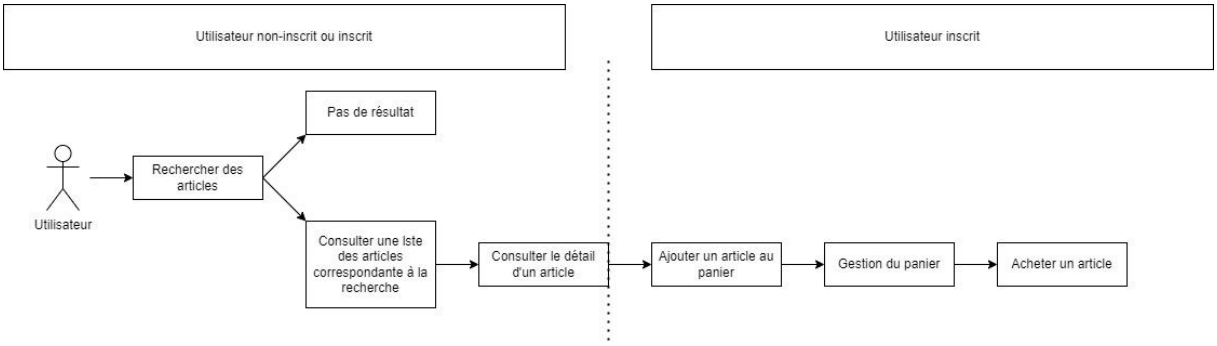
En collaboration avec le client, j'ai essayé de découper le projet en plusieurs objectifs généraux, eux-mêmes découpés en différents cas d'utilisation.

Les cas d'utilisation sont décrits en détail dans le chapitre suivant, c'est-à-dire l'idée derrière (le principe) et mon approche (approche pratique).

Au total nous avons défini 4 objectifs : la vente des articles, la présentation du projet, l'expérience d'utilisateur et une interface d'administrateur.

2.2 La vente des articles

Ce point est l'objectif principal du projet. Sur la base de divers cas d'utilisation, j'ai essayé de définir un processus d'achat complet par lequel passe un utilisateur pour acheter des articles.



Comme on peut le voir sur le graphique, un utilisateur passe par plusieurs étapes dans le processus d'achat.

Il peut d'abord rechercher des articles. Si les critères de recherche sont concluants, les articles associés s'affichent sous forme de liste. Sinon, un message s'affiche indiquant que la recherche selon les critères n'a pas abouti.

Un article est brièvement présenté dans les listes et un bouton permet d'accéder au détail de cet article.

Pour acheter un article, un utilisateur doit avoir un compte et être connecté.

Tant que ces critères sont remplis, un utilisateur peut ajouter des articles à un panier. En consultant ce panier, il peut à nouveau ajuster la quantité des articles avant de procéder à l'achat.

Tout ce processus s'accompagne de petits points supplémentaires. Par exemple, l'utilisateur recevra un e-mail de confirmation automatique après l'achat avec un aperçu de son achat et un fichier PDF généré dynamiquement en pièce jointe, qui représente la facture.

2.3 La présentation du projet

La présentation du projet contient tous les points pertinents sur ce projet et vise à informer l'utilisateur sur tout aussi complètement que possible.

Plusieurs points jouent un rôle important dans cet objectif : la présentation des articles, la présentation de la philosophie du projet ou encore la présentation des différents points de vente.

2.3.1 La présentation des articles

Dès que l'on consulte le détail d'un article, l'article est présenté dans son intégralité: son titre, une description détaillée, la catégorie associée, une description de la catégorie et une liste avec tous les ingrédients liés à cet article.

De plus, un utilisateur peut également découvrir comment d'autres utilisateurs ont perçu cet article. Il peut lire les commentaires publiés sur cet article ou voir à quelle fréquence l'article a été noté et quelle est la note moyenne d'un article.

La combinaison de ces deux points doit donner à l'utilisateur une image aussi complète que possible d'un article.

2.3.2 La présentation de la philosophie

Il s'agit d'un "à-propos" classique destiné à informer l'utilisateur en général sur le projet et sa philosophie. Cette section est facilement accessible via le menu de navigation et est accompagnée d'un diaporama qui montre le travail quotidien.

2.3.3 La présentation des points de vente

Certains des articles de Soap-Opera sont en fait proposés dans les magasins. L'utilisateur en sera également informé. Ici aussi, il peut consulter confortablement les points de vente via le menu de navigation. Les différents points de vente sont alors présentés de manière générale sur la page et chaque point de vente se voit attribuer une carte afin de retrouver rapidement et facilement l'adresse.

2.4 L'expérience d'utilisateur

Tous les points qui contribuent à une meilleure expérience d'utilisateur relèvent de cet objectif.

Certains de ces points sont définis comme des cas d'utilisation indépendants, tandis que d'autres sont plus intégrés comme des points supplémentaires d'un cas d'utilisation.

Les éléments suivants peuvent être considérés comme un cas d'utilisation indépendant :

- Choisir une langue (généralement ou via le formulaire d'inscription)
- Choisir une catégorie de Newsletter via le formulaire d'inscription
- Commenter un article
- Évaluer un article
- Ajouter un article comme favori
- Consulter un profil
- Prendre contact avec Soap-Opera via un formulaire de contact

Les éléments suivants peuvent être considérés comme faisant partie d'un cas d'utilisation:

- Présentation et accessibilité des informations
- L'envoi de mails automatiques à plusieurs endroits (par exemple, un deuxième mail automatique est envoyé après le processus d'achat lorsque Soap-Opera envoie la marchandise. De cette façon, l'utilisateur est directement informé que sa marchandise sera bientôt livrée)
- Affichage de différents messages à différents endroits (par exemple, si des données personnelles sont modifiées, un message s'affiche indiquant que les modifications ont également été acceptées) afin d'informer l'utilisateur de manière transparente à tout moment.
- Si l'utilisateur effectue un achat supérieur à 100 euros, les frais de livraison ne s'appliquent pas.
- Adapter ou modifier ses données personnelles dans le profil
- Affichage de toutes les articles dans le profil qui font partie d'une action effectuée (des articles commentés, évalués, ajoutés comme favori ou achetés) pour y effectuer des nouvelles actions (ajouter à nouveau au panier, supprimer, modifier un commentaire, modifier une évaluation)
- …

2.5 L'interface d'administrateur

Ce dernier objectif vise à installer une interface d'administration simple et claire pour gérer le site.

Il existe deux accès à cette interface, un en tant que "Super-admin" et un en tant que "Finance-admin". Le premier a accès à tout, le second uniquement à la comptabilité.

Un menu détaillé est proposé, qui conduit l'administrateur rapidement et facilement aux entités respectives afin de créer de nouvelles entités ou de modifier celles existantes.

J'ai également créé de nouvelles actions pour diverses entités, ce qui donne plus de contrôle à l'administrateur. Il peut par exemple envoyer une newsletter directement via l'interface, accepter la publication d'un commentaire ou modifier le statut de livraison d'une facture.

Toutes ces actions ont à leur tour un impact sur l'expérience de l'utilisateur. Par exemple, si l'administrateur crée une newsletter dans une certaine catégorie, cette newsletter est automatiquement envoyée à tous les utilisateurs qui ont sélectionné cette catégorie lors du processus d'inscription.

De plus, diverses statistiques sont affichées sur la page d'accueil de l'interface d'administration.

3 Méthode de travail

Afin de pouvoir gérer l'ensemble du projet, j'ai dû travailler dès le départ sur une méthode de travail qui me permettrait d'implémenter toutes les fonctionnalités que j'avais imaginées par petites étapes logiques et compréhensibles.

J'ai essayé de diviser le projet en plusieurs phases.

3.1 La préparation du projet

La phase préparatoire consiste par exemple à installer et configurer un nouveau projet Symfony ou à créer la base de données et à la lier au projet.

Dans cette phase j'ai aussi essayé de créer une interface d'administration fonctionnelle pour pouvoir directement créer et tester certaines entités (articles, utilisateurs,...). J'ai choisi cette méthode plutôt que de travailler avec des "Fixtures" afin de pouvoir travailler rapidement et de manière productive avec des exemples réels.

3.2 Les cas d'utilisation

Le but de cette méthode est de subdiviser le projet en plusieurs cas d'utilisation ou fonctionnalités gérables et de les parcourir point par point.

Je me suis inspiré du cours "Projet Web Dynamique" et j'ai ensuite documenté en détail toutes les fonctionnalités et priorités. Cette documentation devrait aider à créer ce rapport aussi détaillé et précis que possible. Un autre avantage de la documentation est que je peux créer un petit manuel pour des fonctionnalités individuelles que je n'ai jamais créées auparavant.

3.3 Le déploiement

Après la phase de développement, l'étape suivante consistait à mettre le projet en ligne et à tester à nouveau tous les cas d'utilisation et fonctionnalités dans les nouvelles conditions.

3.4 Le rapport

Dans mon rapport, j'ai essayé de décrire avec le plus de détails possible toutes les différentes étapes et approches, ainsi que de décrire certaines des difficultés rencontrées et les solutions apportées.

4 La préparation

4.1 Installation projet Symfony

En troisième année de formation nous avons abordé le Framework PHP Symfony. J'ai tellement aimé le travail que nous avions à faire dans le cadre du cours "Projet web dynamique" que j'ai aussi choisi Symfony comme outil pour le travail final.

Un vif intérêt à explorer les possibilités de Symfony a renforcé cette décision.

Afin d'installer Symfony et d'implémenter ultérieurement certaines fonctionnalités, la documentation de Symfony m'a été d'une grande aide.[1]

Après avoir installé un nouveau projet de Symfony, j'ai lancé le serveur locale ("symfony server:start") dans la console à la racine du projet.

C'était un petit test pour voir si je pouvais obtenir un résultat. En conséquence, la page d'accueil de Symfony s'est affichée dans le navigateur. C'est ainsi que ma base de travail a été créée.

4.2 Création répertoire Github

Après avoir créé un nouveau projet sous Symfony, j'ai créé un répertoire pour ce projet sur mon compte Github.[2]

Le répertoire sert d'une part à me fournir une copie toujours disponible et fonctionnelle du projet et d'autre part à travailler avec des branches. Lorsqu'une branche était terminée (généralement liée à une fonctionnalité ou à un cas d'utilisation), je pouvais fusionner la branche dans la branche principale et recréer une nouvelle branche pour la prochaine étape de travail. Cela m'a toujours donné la sécurité d'avoir une branche principale entièrement créée et fonctionnelle d'une part et une branche secondaire d'autre part où je pouvais tester des choses sans changer la branche principale.

[1] Symfony Contributors (s. d.) Symfony Documentation. https://symfony.com/doc/current/index.html

[2] Github Contributors (s. d.) Creating a new repository. https://docs.github.com/en/repositories/creating-and-managing-repositories/creating-a-new-repository

4.3 Front-End

4.3.1 TWIG-Bundle

Afin de gérer les vues, j'ai travaillé avec le bundle TWIG qui crée par défaut un répertoire de "modèles" (Templates) et un modèle de base "base.html.twig".

Pour configurer TWIG, je me suis référé à la documentation.[3]

4.3.2 Webpack Encore

Pour gérer le CSS, le SCSS et le JAVASCRIPT, j'ai décidé d'utiliser la bibliothèque JAVASCRIPT Webpack Encore.[4]

Pour configurer Webpack Encore, je me suis également référé à la documentation.[5]

L'essentiel de la configuration pour Webpack Encore se fait dans le fichier "webpack.config.js", où j'ai pu définir le "chemin" (path) et les "points d'entrée" (entrypoints) et activer le SCSS (via enableSassLoader).

Pour construire les "assets", j'ai toujours travaillé avec "yarn run encore dev --watch", où "—watch" me permettais de reconstruire les "assets" à chaque fois que j'enregistrais un changement et de ne pas entrer la même commande à chaque changement.

4.4 Installation composants style de base

Les éléments de style ont été élaborés en collaboration avec la graphiste Charlène Counson, qui est également impliquée dans le projet.

Depuis qu'elle a fait un flyer pour le projet, l'idée est venue de faire le site web dans un style similaire (famille de police, couleurs,...).

La famille de polices est "AvenirNext LT pro" et la palette de couleurs est définie comme une liste de variables dans le fichier SCSS "_var.scss".

---

[3] Symfony-Twig Contributors (s. d.) Twig Documentation. https://twig.symfony.com/doc/3.x/intro.html

[4] Symfony Contributors (s. d.) Symfony Documentation. https://symfony.com/doc/current/frontend.html

[5] Symfony Contributors (s. d.) Symfony Documentation. https://symfony.com/doc/current/frontend/encore/simple-example.html

4.5 Tests d'affichage

Maintenant que j'ai installé et configuré tous les composants de base, je voulais tester si tout fonctionnait correctement.

4.5.1 TWIG

Dans cette étape, j'ai d'abord créé un contrôleur en utilisant la commande "php bin/console make:controller", que j'ai nommée "HomeController". Ce contrôleur a pour but d'afficher la page d'accueil.

La première étape consistait à changer la route en "/" et à lui donner le nom "home".

J'ai ensuite défini une variable dans la fonction associée ($test = "Hello World") puis ai envoyé cette variable en réponse au fichier TWIG qui a été automatiquement créé lors de la création du contrôleur.

Dans le fichier TWIG, j'ai ensuite créé un titre (h2) et défini le contenu de la variable comme contenu de ce titre. Le but de ce test était de voir si le transfert de variables vers un fichier TWIG fonctionne et si le contenu est également affiché.

4.5.2 SCSS - CSS

Après avoir vérifié que "Hello World" était affiché comme titre H2 sur la page d'accueil, j'ai voulu tester si les fichiers SCSS sont correctement compilés en CSS et si les styles définis sont appliqués à la page web.

Dans le fichier TWIG "base.html.twig", j'ai créé la connexion dans la zone "Head" avec le fichier CSS compilé.

Étant donné que tous les autres fichiers TWIG héritent du fichier de base, le fichier CSS sera également disponible pour tous les fichiers TWIG créés ultérieurement.

Pour ce faire, j'ai utilisé une fonction d'assistance de TWIG dans Webpack Encore : "encore_entry_link_tags()". Cette fonction rend automatiquement la balise de lien classique pour inclure un fichier CSS dans un projet. Le point d'entrée défini dans webpack.config.js est utilisé comme paramètre pour cette fonction d'assistance.

Le premier paramètre de la fonction addStyleEntry() dans le fichier "webpack.config.js" doit être identique avec le premier paramètre de la fonction encore_entry_link_tags(). Cela garantit que le fichier correct est rendu.

Au final, le test d'affichage a consisté ici à créer un fichier SCSS "_base.scss" et à passer la couleur verte au sélecteur H2.

Après avoir pu me convaincre que le titre "Hello World" était désormais affiché en vert, ce test s'est également terminé positivement. Au fur et à mesure que le travail avançait, j'ai structuré mes fichiers SCSS dans divers dossiers et le fichier CSS pertinent a pu être créé via le nœud "app.scss". Tout changement de style était également directement affiché correctement.

4.5.3 JAVASCRIPT

Le dernier test d'affichage concernait JAVASCRIPT. Semblable aux points d'entrée pour CSS, il existe également un point d'entrée pour JAVASCRIPT, qui est également défini dans le fichier de configuration Webpack Encore.

Pour ce faire, j'ai utilisé une deuxième fonction d'assistance de TWIG dans Webpack Encore : "encore_entry_script_tags()". Cette fonction rend automatiquement la balise de lien classique pour inclure un fichier JS dans un projet. Le point d'entrée défini dans webpack.config.js est utilisé comme paramètre pour cette fonction d'assistance.

Le premier paramètre de la fonction addEntry() dans le fichier "webpack.config.js" doit être identique avec le premier paramètre de la fonction encore_entry_script_tags. Cela garantit que le fichier correct est rendu.

Également similaire à SCSS, j'ai pu gérer mes fichiers JAVASCRIPT dans une structure de dossiers et compiler tous les fichiers dans un seul fichier via le nœud "app.js". Cela permet également de mieux gérer sa structure de dossiers en fonction de ses besoins.

Pour le JAVASCRIPT, le test d'affichage final consistait à créer un fichier JAVASCRIPT et à effectuer un simple appel console.log.

Après m'être assuré qu'un fichier CSS et un fichier JAVASCRIPT étaient correctement compilés et intégrés au projet, j'ai pu terminer cette première phase de test et me consacrer au point suivant, la base de données.

4.6 Installation base de données

La base de données a été développée sur la base des besoins du client analysés au début du projet.

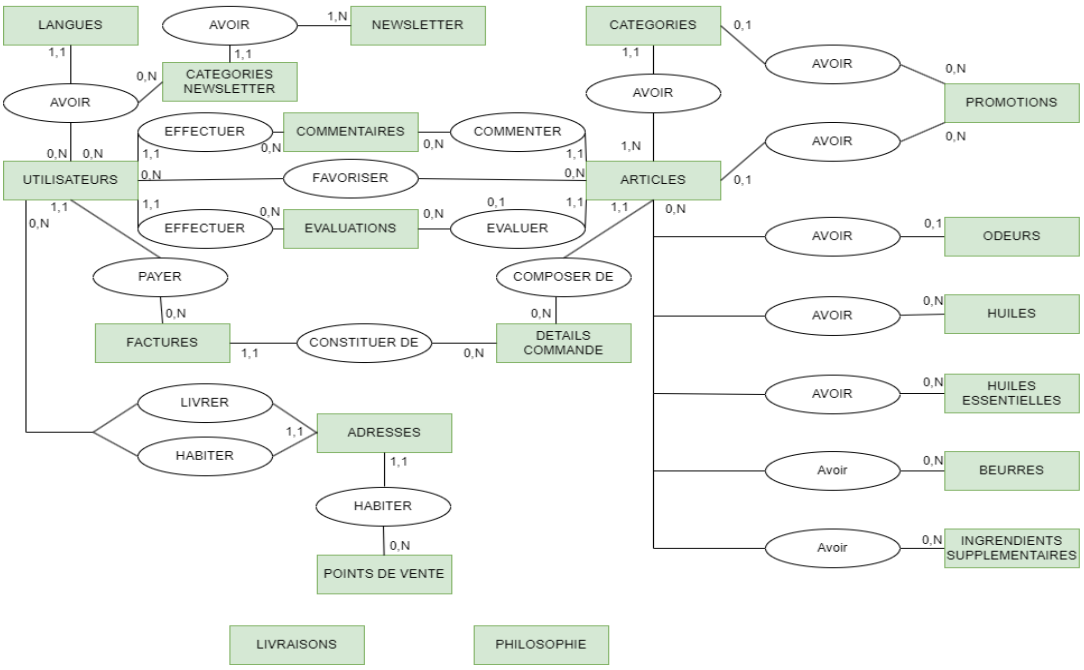
Ma méthode de travail comprenait également une réunion avec le client, qui avait lieu toutes les 2 semaines et dans le but d'analyser ensemble ce qui avait été élaboré et de discuter des prochaines étapes. Sur la base de ces rencontres, les besoins ont également évolué ou ont été reconsidérés en conditions réelles.

Par exemple, l'idée de réserver des événements via un calendrier a été abandonnée car le client. "Soap-Opera" ne pouvait pas évaluer l'impact sur la vie privée. Forts de cette réflexion, nous avons décidé ensemble de ne mentionner que la possibilité existante d'un événement dans la rubrique "Notre philosophie". La base de données a également été adaptée au fil du temps.

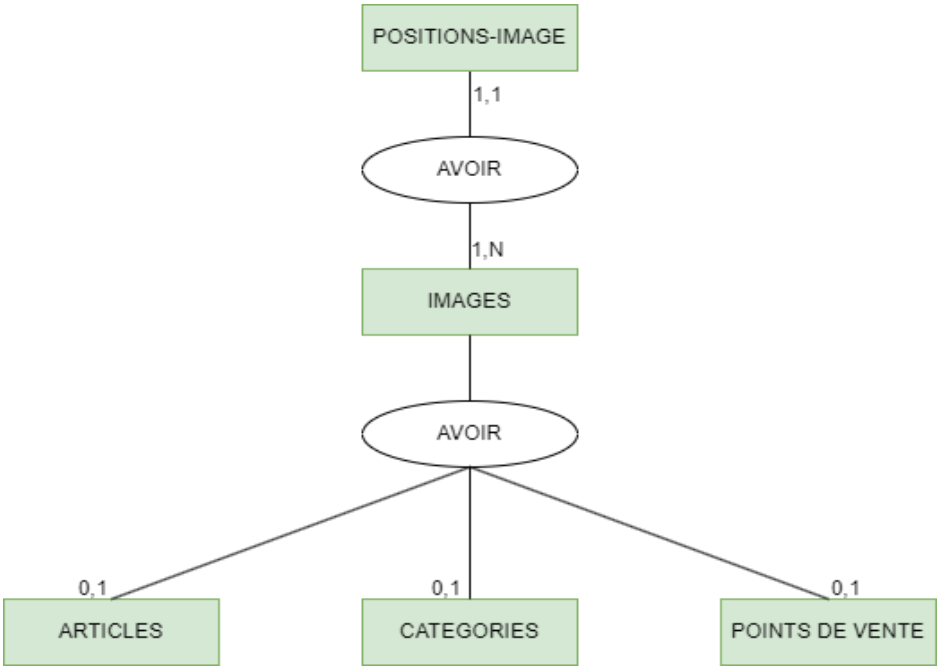
4.6.1 Le MCD (Modèle Conceptuel des Données)

Pour des raisons d'espace, j'ai divisé le schéma en général, images et traductions.

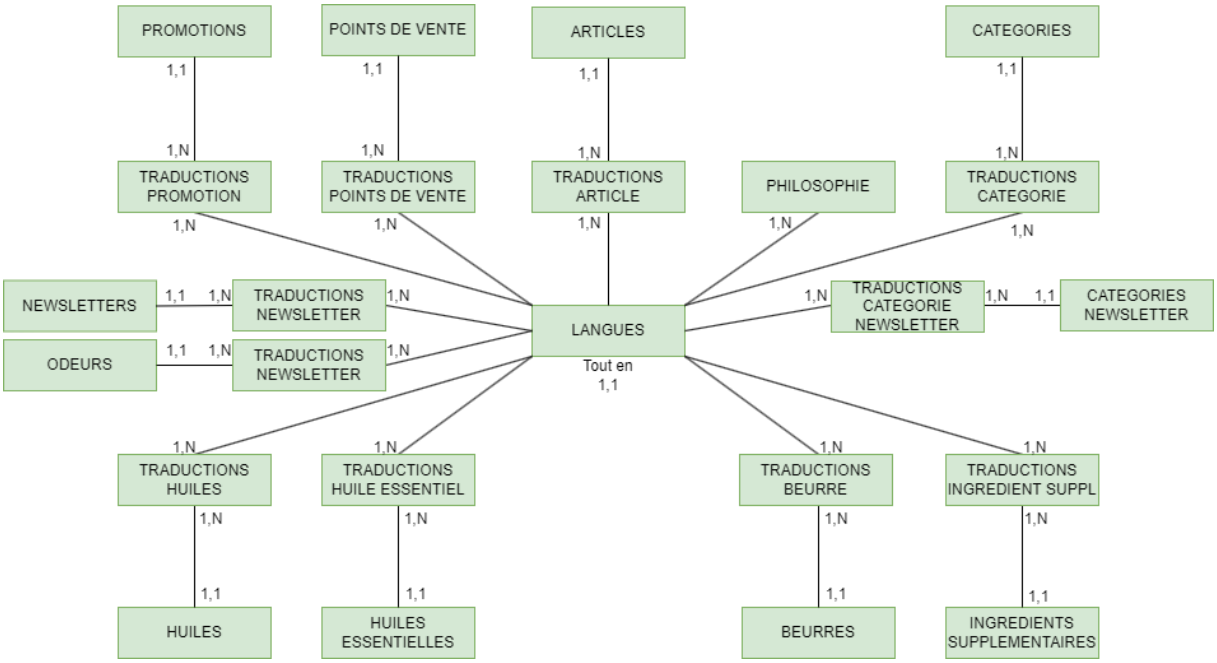
a) Général



b) Les images



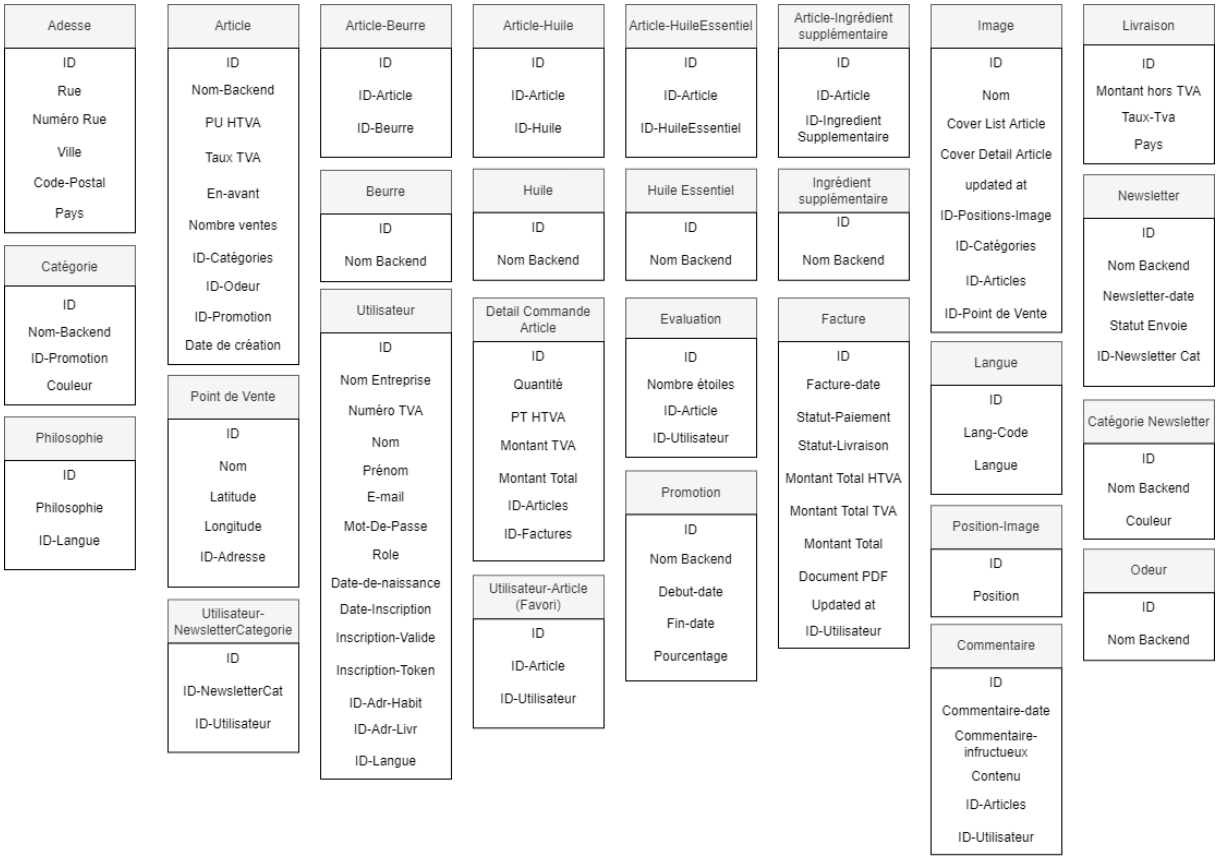
c) Les traductions



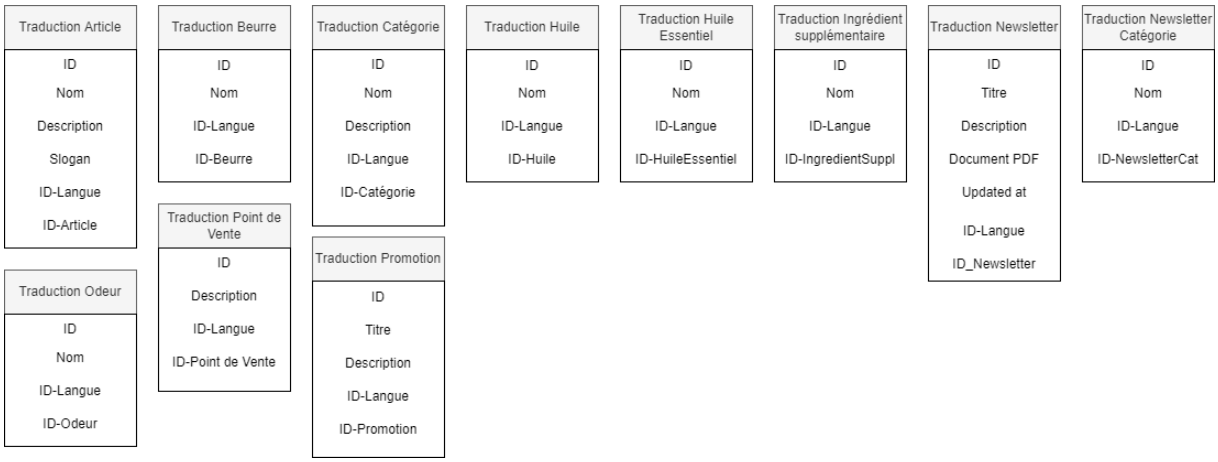
Le type de relation entre toutes les entités pour les traductions est "avoir".

4.6.2 Le MLD (Modèle Logique des Données)

a) Général



b) Les traductions



J'ai décidé de créer toute la base de données au début du projet. Cela s'est passé en plusieurs étapes.

La première étape consistait à créer les modèles graphiques et à créer les relations individuelles (MCD et MLD).

La deuxième étape a ensuite consisté à lier le projet à une base de données vide. Après avoir créé une base de données via WAMP et phpMyAdmin, je devais la connecter au projet. J'ai créé la connexion dans le fichier ".env", dont la variable d'environnement "DATABASE_URL" contient la connexion vers la base de données :

"DATABASE_URL="mysql://db_user:db_password@127.0.0.1:3306/db_name?serverVersion=5.7" "[6]

La troisième étape consistait à créer toutes les entités du projet, puis à les intégrer dans la base de données. J'ai créé les entités via la console et la commande :

"php bin/console make:entity"

Après chaque création d'entité, j'ai d'abord créé un fichier de migration :

"php bin/console make:migration"

Cela permet d'intégrer les entités dans la base de données ou de les supprimer. La véritable intégration dans la base de données se fait alors via la commande :

"php bin/console doctrine:migrations:migrate"

Étant donné que la base de données s'est adaptée à certains égards au fil du temps, le système de migration était très pratique pour moi personnellement, car j'ai pu ajuster les choses dans les entités par la suite sans aucun problème et les commandes de migration peuvent transmettre les changements directement à la base de données.

Les fichiers de migration sont tous enregistrés afin qu'on puisse voir à tout moment l'évolution complète de la structure de la base de données et/ou la reconstruire.

[6] Symfony Contributors (s. d.) Describing the data structure. https://symfony.com/doc/current/the-fast-track/en/8-doctrine.html

4.7 Installation interface d'administration

Maintenant que j'avais créé la base de données et les entités, j'ai décidé de créer également l'interface d'administration en fonction des entités créées.

Je l'ai fait dans le but de créer des entités au fur et à mesure de l'avancement du travail, par exemple en créant un article ou en insérant des images.

Afin de pouvoir créer des éléments facilement et rapidement au début, je ne travaillais pas avec un accès sécurisé à la zone d'administration à ce moment.

4.7.1 Installation EasyAdmin

Afin de créer une interface d'administration, j'ai travaillé avec EasyAdmin 3. D'une part, j'ai pu me familiariser avec cet outil lors du cours "Projet Web Dynamique", d'autre part, j'ai appris d'autres points intéressants grâce à la documentation[7], ce qui m'a beaucoup aidé dans la suite du travail.

J'ai d'abord dû installer le bundle:

"composer require easycorp/easyadmin-bundle"

4.7.2 Création de Dashboard (tableau de bord)

Après avoir installé ce bundle, j'ai pu créer mon "DashboardController".

"Les tableaux de bord sont le point d'entrée des backends et ils sont liés à une ou plusieurs ressources. Les tableaux de bord affichent également un menu principal pour naviguer dans les ressources."[8]

Pour générer un tableau de bord :

"php bin/console make:admin:dashboard"

4.7.3 Dashboard Route

"Chaque tableau de bord utilise une seule route Symfony pour servir toutes ses URL. La seule exigence est de définir la route dans une méthode de contrôleur nommée "index()", qui est celle appelée par EasyAdmin pour afficher le tableau de bord."[9]

---

[7] Symfony Contributors (s. d.) EasyAdmin. https://symfony.com/bundles/EasyAdminBundle/current/index.html
[8] Symfony Contributors (s. d.) Dashboards. https://symfony.com/bundles/EasyAdminBundle/current/dashboards.html
[9] Symfony Contributors (s. d.) Dashboard Route. https://symfony.com/bundles/EasyAdminBundle/current/dashboards.html

Cette route est essentiellement "/admin", mais je l'ai modifiée pour la rendre plus difficile d'accès depuis l'extérieur. Comme mesure de sécurité, j'ai ensuite changé la route en "/admin_s_op".

4.7.4 Le menu principal

"Le menu principal est lié à différents contrôleurs CRUD à partir du tableau de bord. C'est le seul moyen d'associer des tableaux de bord et des ressources. Pour des raisons de sécurité, un backend ne peut accéder aux ressources associées au tableau de bord que via le menu principal."[10]

Dans un premier temps j'ai créé un menu de base. Plus tard et avec une complexité croissante, j'ai affiné le menu principal (par exemple avec des sous-menus)

4.7.5 Les contrôleurs CRUD

"Les contrôleurs CRUD fournissent les opérations CRUD (créer, afficher, mettre à jour, supprimer) pour les entités Doctrine ORM. Chaque contrôleur CRUD peut être associé à un ou plusieurs tableaux de bord."[11]

Pour générer la structure de base d'un contrôleur CRUD :

"php bin/console make:admin:crud"

4.7.6 Les champs (Fields)

"Les champs permettent d'afficher le contenu des entités Doctrine sur chaque page CRUD. EasyAdmin fournit des champs intégrés pour afficher tous les types de données courants."[12]

Pour configurer les champs pour chaque entité, je me suis basé sur la documentation sur les champs.[13]

Maintenant que j'avais créé une interface d'administration de base, j'ai pu effectuer les premiers tests, par exemple si un élément nouvellement créé était également transféré dans la base de données.

---

[10] Symfony Contributors (s. d.) Main Menu.
https://symfony.com/bundles/EasyAdminBundle/current/dashboards.html
[11] Symfony Contributors (s. d.) CRUD Controllers.
https://symfony.com/bundles/EasyAdminBundle/current/crud.html
[12] Symfony Contributors (s. d.) Fields.
https://symfony.com/bundles/EasyAdminBundle/current/crud.html
[13] Symfony Contributors (s. d.) Fields.
https://symfony.com/bundles/EasyAdminBundle/current/fields.html

5 Les cas d'utilisation

5.1 Introduction

Après la phase préparatoire j'ai commencé à m'occuper des fonctionnalités et/ou cas d'utilisation. Je les ai divisés en plusieurs priorités. Ce principe m'a déjà aidé à définir une certaine structure de travail dans le cours "Projet Web Dynamique".

Chaque fonctionnalité individuelle est décrite en détail tout au long du rapport. Quel est l'objectif ? Comment ai-je implémenté la fonctionnalité ? Est-ce que j'ai eu des difficultés ? Comment les ai-je résolus ?...

Voici un aperçu :

Itération	Cas d'utilisation
Priorité 1	CU01 : Affichage de la page d'accueil
	CU02 : Rechercher des articles
	CU03 : Liste des articles
	CU04 : Détail de l'article
Priorité 2	CU05 : S'inscrire
	CU06 : S'authentifier via Email
	CU07 : S'authentifier via Google
	CU08 : S'authentifier comme Admin
Priorité 3	CU09 : Choisir une langue
Priorité 4	CU10 : Formulaire de contact
Priorité 5	CU11 : Newsletter
Priorité 6	CU12 : Commenter un article
	CU13 : Noter un article
	CU14 : Favoriser un article

	CU15 : Ajouter un article au panier
	CU16 : Gestion du panier
	CU17 : Acheter un article
Priorité 7	CU18 : Profile utilisateur
Priorité 8	CU19 : Consulter des statistiques
Priorité 9	CU20 : Consulter les points de ventes
Priorité 10	CU21 : Consulter notre philosophie
Priorité 11	CU22 : Mentions légales
	CU23 : Politique de confidentialité
	CU24 : Les conditions de vente (CGV)

5.2 Priorité 1

La première priorité pour moi était de créer la page d'accueil et tout le processus de recherche d'articles. Par processus de recherche, j'entends la recherche d'articles, une vue de liste d'articles à la suite d'une recherche et une vue détaillée des articles.

De plus, j'ai également couvert les articles qui incluent une promotion, car un type de recherche consiste à rechercher des articles en promotion.

5.2.1 CU01 : Affichage de la page d'accueil

Le Principe

a) Menu de navigation

Le menu de navigation est destiné à guider l'utilisateur à travers l'ensemble du site. De plus, l'utilisateur peut s'inscrire ou se connecter et il peut choisir la langue. La sélection de langues comprend l'allemand, le français et l'anglais. La langue par défaut est l'allemand.

En mode smartphone, le menu peut être appelé via une icône hamburger. A partir du mode tablette, le menu de navigation s'affiche sous forme de barre, mais au-dessus du logo.

Une fois connecté comme utilisateur, les 2 derniers éléments du menu deviendront "mon profil" et "se déconnecter". Si on est connecté comme admin, les 2 derniers éléments du menu deviendront "interface admin" et "se déconnecter".

2 animations plus petites ornent le menu : une fois pour l'élément de menu actif et une fois pour l'effet de survol.

b) Section "Bestseller"

Cette section présente les 3 produits les plus vendus et a pour but d'inciter l'utilisateur à les consulter.

c) Section "Nouveaux articles"

Cette section présente les 6 produits les plus récents et a pour but d'inciter l'utilisateur à les consulter.

d) Pied de page

Le pied de page contient les mentions légales, la protection des données et les conditions générales de vente. En plus, on y retrouve les informations sur le copyright et les ressources.

Approche pratique

a) Menu de navigation

Dans la logique TWIG, il y a un élément de base: "base.html.twig".

Tous les autres fichiers TWIG créés pour le site Web (à l'exception des modèles pour les e-mails) "hériteront" de cet élément de base. Par conséquent, il suffit de créer une zone "header" et "footer" une fois dans cette base et ces zones sont accessibles par la suite dans les autres fichiers TWIG.

Plus précisément, j'ai créé une section pour le menu de navigation dans le fichier "base.html.twig". Plus tard, j'ai dû ajuster le menu de navigation en fonction de l'état de la connexion (non connecté, connecté en tant qu'utilisateur, connecté en tant qu'administrateur).

Un deuxième point était d'intégrer également le logo. Cela m'a été livré au format "svg", que j'ai également intégré dans la structure HTML.

Le dernier point était de styliser le menu de navigation. D'une part pour chaque écran (responsive), d'autre part dans la composition des éléments. Pour cela j'ai créé un dossier "Layout" dans ma structure SCSS avec un fichier "_menuNav.scss" et "_header.scss".

Pour mettre cela en œuvre, je me suis basé sur les cours de "création site statique" de la première année. Dans la zone smartphone et tablette, par exemple, le menu hamburger est sous le logo, tandis que l'ensemble du menu est affiché au-dessus du logo dans des formats d'écran plus grands.

b) Section "Bestseller"

Tout ce qui est affiché dans la zone dynamique du fichier TWIG est préparé dans le HomeController, y compris les articles les plus vendus.

Comme dans la partie recherche, j'ai créé des requêtes spécifiques à la base de données, passé le résultat au contrôleur via la connexion à l'"ArticleRepository" et enfin envoyé le résultat au fichier TWIG pour la page d'accueil.

c) Section "Nouveaux articles"

Tout ce qui est affiché dans la zone dynamique du fichier TWIG est préparé dans le HomeController, y compris les articles les plus récents.

Comme dans la partie recherche, j'ai créé des requêtes spécifiques à la base de données, passé le résultat au contrôleur via la connexion à l'"ArticleRepository" et enfin envoyé le résultat au fichier TWIG pour la page d'accueil.

d) Pied de page

Toutes les informations supplémentaires pertinentes sont disponibles dans le pied de page. D'une part je donne diverses sources (icônes, images,...), d'autre part on peut consulter les mentions légales, la politique de confidentialité et les conditions générales de vente via le pied de page. J'ai reçu les textes de mon stage et les ai adaptés au projet au mieux de mes connaissances et de mes convictions.

5.2.2 CU02 : Rechercher des articles

Le Principe

La section de recherche est disponible sur toutes les pages pertinentes du site pour effectuer une recherche rapide d'articles à tout moment.

a) La recherche par catégories

Le premier type de recherche contient la possibilité de rechercher des articles via leurs catégories. Lorsqu'on sélectionne une catégorie, une liste des articles correspondants à la catégorie sélectionnée s'affiche.

b) La recherche par mots de clé

Le deuxième type de recherche contient une barre de recherche, utilisé pour rechercher des articles par mots-clés. Le résultat est également affiché sous forme de liste.

c) Les promotions

Un troisième type de recherche consiste à rechercher des articles en promotion. Cette option se trouve dans le menu de navigation.

Il est important de savoir qu'une promotion peut concerner un article OU une catégorie d'articles. Ceci peut être sélectionné dans l'interface d'administration. Si une catégorie reçoit une promotion, tous les articles de cette catégorie recevront automatiquement cette promotion.

Approche pratique

Il faut savoir que lorsque une entité a été créé, un fichier de "Repository"[14] est automatiquement créé à propos de cette entité, ce qui permet soit de faire une requête prédéfinie à la base de données (par exemple "findAll()" pour obtenir tous les éléments) soit de créer une propre requête, qui est généralement plus complexe.

[14] Symfony Contributors (s. d.) Querying for objects: The repository. https://symfony.com/doc/current/doctrine.html#querying-for-objects-the-repository

a) L'affichage

La recherche par catégorie et la recherche avancée sont affichées ensemble dans une section, tandis que la recherche de promotion se fait via le menu de navigation.

Étant donné que la section de recherche, tout comme le menu de navigation et le pied de page, est présente sur chaque page, j'ai décidé de créer un fichier TWIG séparé pour cette section, un fragment de modèle.

"Si un certain code TWIG est répété dans plusieurs modèles, vous pouvez l'extraire dans un seul "fragment de modèle" et l'inclure dans d'autres modèles."[15]

a1) La recherche par catégories

Une partie de ce fragment de modèle est réservée à la recherche par catégorie.

Pour afficher toutes les catégories, il fallait créer un "FragmentRechercheController" et récupérer toutes les catégories via le "CategorieRepository" avec la fonction "findAll()" pour pouvoir construire la recherche par catégorie. Les catégories seront envoyées par après vers le fragment TWIG "_recherche.html.twig" pour construire dans une boucle la recherche par catégorie.

Remarque : par la suite il y en a d'autres requêtes qui vont s'ajouter, par exemple une requête pour récupérer les traductions concernant les catégories, basé sur la langue stockée dans la Session.

Maintenant que toutes les catégories ont été transférées vers le fichier, je pourrais créer un lien pour toutes les catégories, qui à son tour mène à l'"Article Controller" et transmet l'ID de la catégorie respective. Ceci est très important pour que cet ID puisse être utilisé dans la deuxième phase pour filtrer les articles correspondants à cette catégorie.

a2) La recherche par mots de clé

La recherche par mots de clé se fait via un formulaire, dont les éléments sont un input pour rentrer les mots et un bouton de "submit" pour envoyer ces mots à l'"ArticleController", où les mots seront utilisés pour récupérer des articles correspondants.

a3) La recherche par promotion

La recherche par promotion se fait via la barre de navigation et a comme but de lister tous les articles qui sont en promotion.

---

[15] Symfony Contributors (s. d.) Reusing Template Contents. https://symfony.com/doc/current/templates.html#embedding-controllers

b) La recherche

L'étape suivante consistait à faire fonctionner les différents types de recherche.

J'ai défini 3 routes dans le ArticleController avec une méthode respective qui permet de traiter les différentes recherches.

b1) La recherche par catégories

En utilisant la recherche par catégorie et en sélectionnant une catégorie spécifique, tous les articles de cette catégorie doivent être filtrés et affichés par la suite sous forme d'une liste.

L'ID de la catégorie, qu'on a choisie hors de la recherche, sera récupéré comme paramètre dans la route et sert comme paramètre dans la fonction de recherche des articles correspondants à cette catégorie.

Ensuite, je mets tous les articles de la catégorie dans un tableau "$articlesCat", ce qui est nécessaire pour un traitement ultérieur.

b2) la recherche par mots de clé

La recherche par mots clés donne à l'utilisateur la possibilité de filtrer les articles selon certains critères de recherche individuels.

Après saisie des mots-clés dans l'input et en appuyant sur le bouton "submit", les mots-clés sont récupérés dans l'ArticleController et sauvegardés dans la session. Ceci est important pour la pagination, qui sera expliqué plus tard.

Ensuite, la table des mots clés est extraite de la session et donnée à la fonction de recherche en paramètre.

Une requête complexe sur base de ce tableau sera effectué qui d'une part va lier toutes les tables et tables de traduction liées à l'article et d'autre part créer une boucle à travers tous les mots clés pour construire la condition "where" de la requête.

Les articles vont être stockés dans un tableau "$tabArticles", ce qui est nécessaire pour un traitement ultérieur.

b3) la recherche par promotion

Un utilisateur peut à tout moment consulter les articles en promotion séparément via la barre de navigation.

Une promotion a une date de début et une date de fin. Dans la requête je veux récupérer uniquement les articles en promotion dont la date de début est inférieure de la date d'aujourd'hui et dont la date de fin est supérieure de la date d'aujourd'hui.

Ensuite, je mets tous les articles en promotion dans un tableau "$tabPromo", ce qui est nécessaire pour un traitement ultérieur.

c) Le traitement ultérieur

J'ai maintenant trouvé des moyens de récupérer les articles correspondants pour les trois recherches.

Cependant, ces articles sont traités plus loin, par exemple pour trouver les traductions correspondantes ou pour préparer une pagination pour la vue de liste.

Plus d'informations et de détails sont traités dans les chapitres suivants.

5.2.3 CU03 : Liste des articles

Le Principe

Après qu'un utilisateur a effectué une recherche, le résultat lui est présenté sous forme de liste. Les éléments de cette liste contiennent un bouton qui mène au détail respectif de l'article.

Une pagination est prévue, qui contient 4 articles par page.

Approche pratique

Puisqu'il existe trois types de recherches différents, j'ai décidé de créer un modèle TWIG qui crée dynamiquement la liste des articles et la pagination de la recherche en question (par exemple pour la recherche par catégorie).

J'inclus d'autres modèles (fragments) dans ce modèle, qui d'une part créent la liste des articles dynamiquement et d'autre part créent leur propre pagination avec leurs propres paramètres en fonction du type de recherche.

Les informations pour générer les différentes zones sont transmises par l'"ArticleController".

De plus, j'ai créé mon propre système de pagination, qui contient 3 variantes d'affichage différentes, selon la page sur laquelle l'utilisateur se trouve actuellement.

Chaque article présenté dans la liste contient une image, la catégorie sous forme d'icône, le nom, le slogan et un bouton pour pouvoir consulter le détail de l'article respectif.

Si un article est en promotion, cela sera également indiqué dans la vue liste. Un autre titre est ajouté en vert, ainsi que la durée exacte de la promotion.

5.2.4 CU04 : Détail de l'article

Le Principe

Une fois l'utilisateur a effectué une recherche et a obtenu une liste avec des articles correspondants à la recherche, il peut consulter une page de détail pour chaque article. La page de détail est accessible via un bouton correspondant.

La page de détail a deux objectifs.

a) Consulter des informations

La page de détail contient toutes les informations nécessaires sur l'article.

L'utilisateur trouvera ici le nom, une image, le slogan, mais aussi un système d'onglets avec des informations plus détaillées, comme une description longue, la catégorie et sa description, le prix et les ingrédients.

De plus, l'utilisateur peut également consulter les commentaires publiés sur cet article ainsi que le nombre de notes et la note moyenne.

b) Actions sur les articles

Une fois qu'un utilisateur est connecté au site, il a accès aux actions sur les articles. Pour cela, il doit avoir un compte. S'il n'a pas de compte, il n'a pas non plus accès aux actions.

Les actions sont présentées sous la forme d'une barre d'actions et comprennent : noter, commenter, favoriser et acheter. Chaque action correspond à un cas d'utilisation et sera examinée plus en détail dans les chapitres suivants.

Approche pratique

a) Consulter des informations

Dans cette priorité, je me suis d'abord concentré sur l'affichage de la page de détail et le fonctionnement du système d'onglets avec les différentes informations en JAVASCRIPT.

Pour créer le détail d'un article, j'ai d'abord créé une autre route dans l'"ArticleController". Cette route inclut un paramètre supplémentaire, l'ID de l'article. L'ID d'article est transmis via le bouton de détail dans la vue de liste.

Avec la récupération automatique de l'article dans le contrôleur, j'ai pu avoir toutes les informations pertinentes sur cet article directement disponibles. J'envoie ensuite ces informations au fichier TWIG "detail.html.twig" où j'ai construit ma vue.

b) Actions sur les articles

J'ai travaillé sur la fonctionnalité des actions dans une priorité ultérieure.

Avant tout, je me suis préoccupé de présenter la barre visuellement.

5.3 Priorité 2

Maintenant que j'ai créé la page d'accueil et certains éléments fixes (le menu de navigation, le pied de page, la section de recherche,...) et que j'ai défini les paramètres de base pour les actions de recherche (recherche - liste - détail), j'ai vu la seconde prioritaire dans le processus d'inscription, car un utilisateur ne peut effectuer des actions sur les articles que s'il est connecté.

Créer une connexion via un compte Google est également possible.

5.3.1 CU05 : S'inscrire

Le Principe

Chaque utilisateur peut s'inscrire à Soap-Opera et ainsi élargir son cercle d'interaction. Par exemple, il peut mettre en favoris, commenter et/ou noter des articles.

L'inscription elle-même se fait via un formulaire, où l'utilisateur remplit toutes les informations nécessaires sur sa personne.

Après avoir rempli le formulaire, l'utilisateur recevra un email dans lequel il devra valider son inscription.

Il doit également saisir un mot de passe soumis à certains critères (au moins 6 caractères, au moins 1 lettre, au moins 1 chiffre et au moins 1 caractère spécial). Il s'agit de créer un mot de passe plus complexe. Le mot de passe lui-même est stocké sous forme hachée dans la base de données.

Le processus d'inscription devrait également inclure quelques points supplémentaires intéressants, tels que la sélection de la langue ou des différentes catégories de newsletter.

Approche pratique

a) Création de formulaire

J'ai d'abord créé 2 formulaires, un basé sur l'entité utilisateur, un basé sur l'entité adresse. J'ai ensuite intégré le formulaire d'adresse deux fois dans le formulaire d'utilisateur, une fois pour l'adresse résidentielle et une fois pour l'adresse de livraison.

J'ai créé les formulaires à l'aide de la commande :

"php bin/console make:form"

b) Les contraintes

J'ai défini différentes règles pour les différentes propriétés des entités, comme la longueur minimale d'un mot de passe ou d'un code postal.

Problème rencontré

Si on définit ces règles dans l'entité, elles seront appliquées automatiquement lorsqu'on crée un formulaire basé sur cette entité.

Lorsque j'ai intégré les formulaires liés à l'adresse dans le formulaire utilisateur, j'ai remarqué que les règles pour les propriétés des adresses n'étaient pas vérifiées.

Solution apportée

Dans mes recherches, j'ai découvert que lorsqu'on intègre un formulaire dans un autre formulaire, les règles concernant les propriétés de cette entité doivent être redéfinies dans le formulaire.

Le formulaire d'inscription est basé sur l'entité utilisateur et non sur l'adresse. Pour que les règles des propriétés d'adresse soient également vérifiées, elles doivent être redéfinies dans le formulaire d'adresse.

c) Le traitement

Pour le processus d'enregistrement, j'ai créé un "RegistrationController". D'une part, la route "/registration" permet de créer la vue du formulaire et de la transmettre au fichier TWIG "inscriptionUtilisateur.html.twig, et d'autre part, le traitement est défini après soumission du formulaire.

Si un utilisateur s'inscrit et a rempli le formulaire d'inscription, la première étape consiste à vérifier si tous les champs ont été remplis correctement.

Problème rencontré

J'ai défini toutes les règles de base dans l'entité adresse au niveau des codes postaux. Par exemple, la longueur ne doit pas être inférieure à 4 ni supérieure à 5 (spécifique au pays Belgique - Allemagne).

Cependant, je n'ai pas trouvé de moyen de définir une règle de comparaison dans ces règles indiquant que les villes de Belgique ont un code postal à 4 chiffres et les villes d'Allemagne ont un code postal à 5 chiffres.

Solution apportée

Afin de garantir ce contrôle, j'ai défini mon propre contrôle dans le contrôleur. Si ce contrôle est positif, la suite du traitement est lancée, sinon l'utilisateur est redirigé vers le formulaire avec un message d'erreur.

Une fois qu'un utilisateur a rempli le formulaire d'inscription et respecté toutes les règles, plusieurs étapes sont lancées:

- Un "Token" sera généré, qui sera utilisé pour la confirmation de l'inscription par E-mail. Pour cela j'ai créé une fonction qui génère le "Token".

- Les adresses (domicile et adresse de livraison) sont comparées aux adresses de la base de données. Si une adresse existe déjà dans la base de données, elle est attribuée directement à l'utilisateur, sinon elle est d'abord créée puis attribuée.

- Le mot de passe est haché et enregistré dans la base de données. Le mot de passe n'est jamais enregistré sous sa forme originale dans la base de données.

- Un nouvel utilisateur est créé dans la base de données sur la base des données saisies dans le formulaire d'inscription.

d) La validation par E-Mail

Comme dernière étape de la phase d'inscription, un e-mail automatique est envoyé à l'utilisateur. Ceci est utilisé pour confirmer l'enregistrement et se fait en vérifiant un "Token". Ce "Token" est créé lors de l'inscription et attribué à l'utilisateur dans la base de données. Dans le même temps, le "Token" est également défini dans le mail comme paramètre stocké dans le bouton de confirmation.

Si l'utilisateur appuie maintenant sur ce bouton de confirmation dans l'e-mail, il est redirigé et le RegistrationController vérifie en interne si le "Token", qui est récupéré via le bouton de confirmation, peut être attribué à un utilisateur dans la base de données. Si tel est le cas, l'inscription est confirmée et le "Token" de l'utilisateur dans la base de données est supprimé car il n'est plus nécessaire.

Dès que le processus d'inscription est terminé et que la validation a été effectuée, l'utilisateur peut se connecter.

5.3.2 CU06 : S'authentifier via Email

Le Principe

Une fois que l'utilisateur s'est enregistré sur le site, il peut se connecter à tout moment. Il est important qu'il ne puisse se connecter qu'après avoir validé son inscription par email.

Hors de l'authentification, l'utilisateur peut changer son mot de passe, s'il a oublié. Un lien "mot de passe oublié?" donne accès à cette action. L'utilisateur est demandé à ce moment de rentrer son adresse d'Email. Une Email sera lui envoyer avec la demande de confirmation pour le changement de mot de passe.

Une fois confirmé le changement de mot de passe via le bouton dans l'Email, l'utilisateur a accès à un petit formulaire pour pouvoir changer son mot de passe. Une fois effectué le changement de mot de passe en respectant les contraintes soumises sur le mot de passe, le changement sera enregistré dans la base de données et l'utilisateur peut se connecter avec son nouveau mot de passe.

Étant donné que l'utilisateur peut également choisir une langue lors du processus d'inscription, le site Web doit être présenté dans cette langue une fois que l'utilisateur s'est connecté.

La fonctionnalité "se souvenir de moi" est également disponible.

Approche pratique

La première étape consistait à créer un formulaire d'authentification. Pour cela j'ai utilisé la commande suivante :

"php bin/console make:auth"

"La plupart des sites Web ont un formulaire de connexion où les utilisateurs s'authentifient à l'aide d'un identifiant (par exemple, une adresse e-mail ou un nom d'utilisateur) et un mot de passe. Cette fonctionnalité est fournie par l'authentificateur de connexion par formulaire."[16]

J'ai d'abord créé un "SecurityController" pour créer un formulaire de connexion. Dans la méthode de la route "/login" j'ai aussi récupéré l'URL courante afin de faire facilement une redirection par la suite.

Après il fallait créer un "Authenticator", qui contient une fonction "authentication()" qui est utilisée pour vérifier si l'utilisateur avec les données saisies existe dans la base de données. Ici, je vérifie également si l'inscription de l'utilisateur a déjà été validée.

Les fonctions "onAuthenticationSuccess()" et "onAuthenticationFailure()" dans ce "Authenticator" (dans mon cas "UtilisateurAuthenticator") permettent de définir d'autres étapes, selon que l'utilisateur existe ou non.

Pour créer la fonctionnalité "se souvenir de moi", j'ai suivi un tutoriel[17].

Pour pouvoir réinitialiser le mot de passe, j'ai ajouté un lien dans le formulaire. Si l'utilisateur le confirme, il est dirigé vers un premier formulaire où il doit renseigner son adresse email. Cela sert à vérifier si l'utilisateur existe dans la base de données. Un contrôle a lieu dans le SecurityController. Si l'utilisateur

[16] Symfony Contributors (s. d.) Security – Form Login.
https://symfony.com/doc/current/security.html
[17] SymfonyCast Contributors (s. d.) Remember me system.
https://symfonycasts.com/screencast/symfony-security/remember-me

existe, un email automatique sera envoyé avec un bouton pour changer le mot de passe. S'il n'existe pas, il reçoit un message d'erreur.

Si l'utilisateur utilise maintenant le bouton dans l'e-mail, il sera redirigé vers un deuxième formulaire où il pourra rentrer un nouveau mot de passe. Celui-ci doit être saisi deux fois pour confirmation.

Problème rencontré

Pour le formulaire "PasswordResetType", j'ai créé un formulaire basé sur l'entité utilisateur, mais uniquement sur la propriété mot de passe.

Mais toutes les propriétés de l'entité utilisateur ont été vérifiées lors de la soumission du formulaire.

Cependant, comme les autres propriétés de ce formulaire n'étaient pas prises en compte, la conséquence était un message d'erreur.

Solution apportée

La fonction "configureOptions()" permet de définir diverses options pour un formulaire, y compris la création de groupes de validations.[18]

Concrètement, cela signifie que certains groupes de validation peuvent être affectés aux propriétés de l'entité. Si on définit ce groupe de validation dans la fonction des options pour le formulaire, seules les propriétés correspondant à ce groupe de validation seront vérifiées. Le groupe de validation par défaut est default.

Le mot de passe a le groupe "reset" dans ses validations. Toutes les autres propriétés n'ont pas ce groupe. Ainsi le formulaire avec le groupe de validation "reset" dans les options ne vérifiera que le mot de passe. Le problème a donc été résolu.

5.3.3 CU07 : S'authentifier via Google

Le Principe

En plus de l'authentification classique, un utilisateur peut également se connecter via son compte Google.

Un bouton correspondant est fourni et redirige l'utilisateur vers son compte Google. Après confirmation il sera redirigé vers le site Soap-Opera.

Cependant, il ne peut utiliser ce service que s'il possède déjà un compte avec la même adresse e-mail chez Soap-Opera. Si ce n'est pas le cas, l'utilisateur sera

[18] Symfony Contributors (s. d.) Validation groups
https://symfony.com/doc/current/form/validation_groups.html

d'abord redirigé vers le formulaire d'inscription pour créer un compte avec l'adresse email du compte de Google.

Dès que l'utilisateur est passé par le processus d'inscription classique, il peut alors se connecter avec son compte Google sans aucun problème.

Approche pratique

J'ai d'abord dû installer 2 bundles :

- knpuniversity/oauth2-client-bundle
- league/oauth2-google

Ensuite, j'ai dû créer un compte et un projet dans la console développeur de Google. Là, plusieurs informations devaient être transmises, comme l'URI ou l'URI de redirection.

Lorsqu'on crée un projet, on obtient également un identifiant Google et un identifiant Google secret. Ces deux identifiants devront ensuite être définis ultérieurement comme des variables d'environnement dans le fichier .env.

Ces identifiants sont également définis dans le fichier de configuration config/packages/knpu_oauth2_client.yaml.

Ensuite j'ai créé un "GoogleController" avec 2 routes : "/connect/google" (route vers Google) et "/connect/google/check" (route depuis Google).

Un "GoogleAuthenticator", à son tour, vérifie si l'utilisateur est dans la base de données ou non. Ce processus est similaire à la vérification d'un utilisateur lors de la connexion ("UtilisateurAuthenticator").

S'il y a un utilisateur, la fonction "onAuthenticationSucces()" est appelée, dans laquelle j'extrais la langue de l'utilisateur de la base de données et l'enregistre dans la session. Cela garantit également que le site Web est affiché dans la langue de l'utilisateur.

Si l'utilisateur n'existe pas, je définis dans la fonction "onAuthenticationFailure()" que l'utilisateur est redirigé vers le formulaire d'inscription afin de créer un compte avec un e-mail de Google et ainsi pouvoir s'inscrire via Google dans le futur.

5.3.4 CU08 : S’authentifier comme Admin

Le Principe

L'accès à l’interface d'administration n'est accordé qu'aux personnes ayant le statut "ROLE_SUPER_ADMIN" ou "ROLE_FINANCE_ADMIN".

"ROLE_SUPER_ADMIN" donne accès à tous les éléments de menu, cela signifie à l'ensemble de l'interface d'administration.

"ROLE_FINANCE_ADMIN" donne uniquement accès à l'élément de menu 'comptabilité'.

De plus, l'URL d'accès à l'interface d'administration passera de /admin à "/admin_s_op" pour rendre l'accès via URL plus difficile.

L'URL redirige vers le formulaire d'authentification et après une authentification réussie, l'administrateur a accès à l'interface d'administration. Alternativement, il peut également utiliser le formulaire d'authentification classique via le menu de navigation.

Une redirection est fournie lors de la tentative de connexion avec un compte qui n'a aucun des rôles ci-dessus. Dans ce cas, l'utilisateur sera redirigé vers la page d'accueil avec un message d'erreur.

Une fois connecté en tant qu'administrateur, les 2 derniers éléments du menu de navigation passeront également à "interface admin" et "se déconnecter". L'élément de menu 'interface admin' donne alors également accès à l'interface d'administration.

Approche pratique

Dans un premier temps, j'ai défini les rôles, qui ont accès à l'interface d'administrateur, dans le fichier "security.yaml" sous la rubrique "security", et la route vers laquelle se fait la redirection si on n'y a pas accès avec ses identifiants.

J'ai ensuite créé deux nouveaux administrateurs via l'interface d'administrateur, avec les rôles correspondants. Enfin, j'ai intégré une condition dans le "DashboardController" pour le menu. J'utilise la fonction "isgranded()" pour tester le rôle de l'administrateur et ajuster le menu en conséquence. S'il a le rôle "SUPER", tout le menu est affiché, s'il a le rôle "FINANCE", seul l'élément comptable est affiché.

Lorsqu'un utilisateur s'inscrit sur le site, son mot de passe est toujours haché et stocké dans la base de données. J'ai voulu accorder le même fonctionnement lors de la création d'un utilisateur (admin) via l'interface d'administrateur.

Pour y parvenir, j'ai dû créer un "EventSubscriber".

"Lors de l'exécution d'une application Symfony, de nombreuses notifications d'événements sont déclenchées. Votre application peut écouter ces notifications et y répondre en exécutant n'importe quel morceau de code."[19]

[19] Symfony Contributors (s. d.) Events and Event Listeners https://symfony.com/doc/current/event_dispatcher.html

Concrètement, j'ai créé un "EasyAdminSubscriber" dans lequel je crée moi-même 3 fonctions : "setPassword()", "addUser()" et "updateUser()".

Dans cette logique, la première fonction est appelée dans les deux autres, de sorte que lors de l'ajout ou de la modification d'un utilisateur, le mot de passe est enregistré sous certaines conditions définies dans la première fonction.

5.4 Priorité 3

Comme point suivant, j'ai décidé de mettre en place un système de traductions (français, anglais et allemand), car il y a déjà pas mal de flux de données sur le site grâce à la possibilité de rechercher des articles.

De plus, un utilisateur peut choisir une langue lors du processus d'inscription afin que le site Web soit présenté dans cette langue lorsque l'utilisateur se connecte ultérieurement.

Sur ce point, j'ai travaillé avec la documentation [20] d'une part et je me suis inspiré d'un tutoriel[21] d'autre part.

5.4.1 CU09 : Choisir une langue

Le Principe

Un utilisateur a le choix entre trois langues, qu'il peut sélectionner via la barre de navigation.

La sélection est présentée sous forme de drapeaux nationaux. A noter que seules les langues inactives peuvent être sélectionnées. Par exemple, si la page est en allemand, l'utilisateur ne peut sélectionner que le français et l'anglais.

Lorsqu'un utilisateur enregistré se connecte, la page s'affiche dans la langue qu'il a sélectionnée lors du processus d'enregistrement. Comme l'utilisateur peut modifier ses données à tout moment dans son profil, il peut également y changer la langue.

Si un administrateur est créé via l'interface d'administration, la langue y sera également sélectionnable. Le site Web et l'interface d'administration seront affichés ultérieurement dans la langue sélectionnée.

Approche pratique

a) Les configurations

J'ai d'abord défini les langues dans le fichier de configuration "config/services.yaml" sous "app.locales".

Ensuite j'ai défini les langues pour TWIG dans le fichier de configuration "config/packages/twig.yaml" (via la variable "app.locales").

[20] Symfony Contributors (s. d.) Translations.
https://symfony.com/doc/current/translation.html

[21] Nouvelle Techno (2020, 01 feb) Live Coding: Créer un site multilingue avec Symfony 4
https://www.youtube.com/watch?v=JOAiup61byY

Afin de définir la langue par défaut, j'ai dû faire des changements dans le fichier de configuration "config/packages.translation.yaml". Nous avons choisi l'allemand comme langue standard car cette langue nous paraissait logique dans notre région. Un chemin par défaut vers les fichiers de traduction est également spécifié ici.

b) Le système de drapeaux

Maintenant que j'ai effectué les configurations de base, je me suis ensuite occupé de l'affichage des drapeaux des pays. L'idée derrière cela est que la langue actuelle n'est pas affichée sous forme de drapeau. Par exemple, si le site est actuellement en français, seuls les drapeaux allemand et anglais sont affichés.

Dans "base.html.twig" dans la barre de navigation, j'applique maintenant une boucle à cette variable. A chaque itération, je vérifie ensuite si la langue respective correspond à la langue de la session. Si tel n'est pas le cas, le drapeau s'affiche.

Chaque drapeau est défini comme un lien menant à une route que j'ai définie dans un "LocaleController". Cette route inclut une méthode qui stocke la langue sélectionnée (attachée en paramètre à cette route) dans la session.

c) Les fichiers de traduction

Il existe une commande pour générer automatiquement les fichiers de traduction dans la langue respective.

"php bin/console translation:extract –force (suivi par la langue "de", "fr" ou "en")"

Il existe différents domaines de traduction qui sont générés automatiquement, par exemple un domaine pour "EasyAdmin", un pour la sécurité, un pour les messages généraux, etc. On peut également créer ses propres domaines ultérieurement, comme je l'ai fait pour les formulaires, par exemple.

d) Les traductions

Les textes des traductions doivent être "marqués". Cela peut se faire de différentes façons. Pour tous les textes durs dans les fichiers TWIG, j'ai marqué ces textes avec les balises de TWIG.

Dans d'autres situations, j'ai utilisé la traduction en tant que service, par exemple pour les messages flash dans les contrôleurs.

Pour l'interface d'administration, j'ai dû définir le domaine dans le "DashboardController". Ceci établit la connexion aux fichiers de traduction dans ce domaine.

Pour les formulaires, j'ai travaillé avec l'objet "TranslatableMessage", où je connecte le texte à traduire avec une variable et définis mon propre domaine de traduction.

e) LocaleSubscriber

Maintenant que j'ai créé un système pour les traductions (drapeaux, session,...) d'une part et trouvé la possibilité de traduire différents textes à différents endroits du projet d'autre part, il me manquait une dernière étape pour rendre les traductions efficaces.

Pour l'instant il n'y a qu'un système théorique dans lequel une langue est stockée dans la session à la suite d'une action (cliquez sur le drapeau). Pour le moment, cependant, le système ne sait pas encore ce qui en résultera. Pour ce faire, on doit créer un "Subscriber", similaire à ce que j'ai fait lors de l'enregistrement du mot de passe lors de la création d'un administrateur.

Ce "Subscriber" fonctionne de la même manière que le "EventListener" en JAVASCRIPT.

Dans ce "Subscriber" je récupère la langue standard via le constructeur et la fonction "onKernelRequest()" vérifie s'il y a eu un changement de langue dans la session. Si tel est le cas, cette langue sera désormais utilisée, sinon la langue par défaut sera utilisée.

f) La langue de l'utilisateur

Dès qu'un utilisateur se connecte, la page est traduite dans la langue que l'utilisateur a indiquée lors de son inscription. A cet effet, la langue respective de l'utilisateur est récupérée dans les fichiers "UtilisateurAuthenticator" et "GoogleAuthenticator" et enregistrée dans la session.

Maintenant que j'ai créé le "Subscriber", qui contrôle la langue de la session après chaque requête, le site a été traduit dans la langue de l'utilisateur après une connexion réussie. Cependant il y avait un problème.

Problème rencontré

Avec l'authentification normale via le menu de navigation et le formulaire de connexion, la traduction s'est déroulée sans problème.

En essayant de me connecter en tant qu'administrateur via l'URL "/admin_s_op", j'ai remarqué que même si j'étais directement redirigé vers l'interface d'administration, la langue restait dans la langue par défaut. Mais j'ai donné à cet administrateur la langue française.

Solution apportée

Dans le fichier "UtilisateurAuthenticator", j'ai initialement défini la partie qui récupère la langue de l'utilisateur et la stocke dans la session dans la fonction "onAuthenticationSuccess()".

Je suis parti de l'idée qu'après une authentification réussie, des processus sont définis dans cette fonction qui est utilisée spécifiquement dans cette situation.

Dans ce cas, j'ai décidé d'utiliser la fonction "dd()" pour vérifier si un message de test est affiché dans les deux types d'authentification (via le menu de navigation ou via URL).

Le message s'affichait lors de l'authentification via le menu de navigation, mais pas lors de l'authentification via l'URL.

En conséquence, j'ai déplacé la fonction "dd()" vers la fonction "authentication()". Maintenant, le message s'affichait également après la tentative d'authentification via l'URL. Maintenant, j'avais trouvé un dénominateur commun.

Il ne me restait plus qu'à déplacer tout le code lié à la langue de l'utilisateur de la fonction "onAuthenticationSuccess()" vers la fonction "authentication()".

Le problème a été résolu et maintenant, après une connexion réussie via l'url, l'interface d'administration s'affichait directement dans la langue de l'administrateur.

g) La base de données

Pour assurer les traductions des textes de la base de données, j'ai travaillé avec des tables de traduction. Une table de traduction est créée pour chaque entité qui nécessitera éventuellement des traductions. L'entité obtient une propriété "nomBackend" qui sert de référence pour la table de traduction. La langue est définie comme la deuxième référence dans la table de traduction.

5.5 Priorité 4

5.5.1 CU10 : Formulaire de contact

Le Principe

L'élément de menu "contact" dans la barre de navigation mène à un formulaire de contact.

Cela inclut les champs Email, nom, sujet et message. Tous ces champs sont obligatoires.

Le formulaire de contact peut être utilisé pour poser des questions à Soap-Opera.

Approche pratique

Dans le menu de navigation, il y a un lien vers le formulaire de contact, qui a été créé dans un fichier TWIG séparé. Ici, je n'ai pas utilisé la commande "make:form", mais j'ai créé mon propre formulaire.

Ce formulaire redirige vers une route dans le "ContactController", que j'ai créé pour ce point.

Lors de la redirection vers le formulaire de contact, j'envoie également un certain nombre de variables au fichier TWIG, mais ce sont toutes des chaînes vides au début. Ces valeurs sont ensuite passées aux valeurs des différents champs du formulaire, avec pour résultat que le formulaire apparaît initialement vide.

Si on remplit maintenant le formulaire et qu'on l'envoie, les données saisies seront récupérées via une deuxième route et vérifiées pour diverses contraintes.

Si la validation a réussi, un mail automatique est envoyé à Soap-Opera via "TemplatedMail".

Si la validation a échoué, on sera redirigé vers le formulaire avec des messages d'erreur personnalisés. Les données précédemment saisies sont également renvoyées au formulaire afin que le formulaire apparaisse toujours rempli.

5.6 Priorité 5

5.6.1 CU11 : Newsletter

Le Principe

Un utilisateur a la possibilité, lors du processus d'inscription, d'accepter de recevoir des newsletters.

Il existe plusieurs catégories de newsletters, par exemple la catégorie "Événements" ou "Articles". L'utilisateur peut alors également choisir les catégories pour lesquelles il souhaite recevoir une newsletter.

On peut ensuite ajuster la newsletter dans son profil, c'est-à-dire se désinscrire ou ajouter des catégories.

Approche pratique

Pour mettre en œuvre cette idée, j'ai d'abord dû adapter la base de données, créer une table pour les catégories de la newsletter et la connecter aux utilisateurs.

Maintenant que la base de données a été ajustée à ce point, je devais encore ajuster le "DashboardController" et les "CRUDControllers" respectifs. Après cela, j'ai pu créer facilement des newsletters pour une certaine catégorie dans 3 langues différentes via l'interface d'administration.

Cependant, je devais encore trouver un moyen de les envoyer.

Afin d'envoyer les newsletters, j'ai décidé de créer une action supplémentaire dans le "CRUDController" pour les newsletters.

L'action "Envoyer Newsletter" est maintenant ajoutée aux actions par défaut comme "éditer" ou "supprimer". L'avantage de cette approche est que l'on peut définir une route où, après avoir utilisé l'action, un certain traitement peut être défini et que l'on peut également définir quand exactement l'action est disponible.

Pour définir quand l'action sera disponible ou pas, j'ai ajouté une propriété "statusEnvoie" à l'entité Newsletter. Ceci est un booléen et est "false" par définition.

Avec la fonction "onlyDisplayIf()" j'ai pu alors définir si le booléen est sur "false" l'action sera disponible. Après avoir utilisé l'action, le booléen change vers "true" et par conséquent, l'action de cette newsletter ne sera plus disponible. Il est à éviter qu'une newsletter soit envoyée plusieurs fois.

La fonction "linkToRoute()" détermine la route associée à cette action. L'identifiant de la newsletter est également transmis. Le traitement a ensuite lieu dans le "NewsletterController".

Ici j'ai récupéré la catégorie de la newsletter et tous les utilisateurs qui sont affectés à cette catégorie. Ensuite, je mets ces utilisateurs en boucle et j'envoie un e-mail automatique à chaque utilisateur dans sa langue à chaque itération.

Si on crée une newsletter via l'interface d'administrateur, l'administrateur peut ajouter un PDF à la table de traduction "TraductionNewsletter". Cela permet d'ajouter un PDF traduit pour chaque langue.

J'ai travaillé ici avec le bundle "VichUploader".

Comme pour les images, les fichiers ne sont pas enregistrés en tant que fichier dans la base de données, seul leur nom est enregistré. Les fichiers eux-mêmes sont enregistrés dans le projet sous un certain chemin.

Les PDF sont affichés sous forme de lien dans la liste des traductions de Newsletter dans l'interface d'administration. J'ai créé mon propre modèle pour cela ("admin/pdf.html.twig"). Un "base_path" définit alors où se trouve exactement le PDF. Les chemins exacts sont quant à eux définis dans le fichier de configuration "service.yaml".

Le "NewsletterController" vérifie s'il existe un PDF. Si tel est le cas, il sera joint à l'e-mail.

Le but de la newsletter était, d'une part, d'offrir un système convivial avec plus de liberté. À mon avis, cela est donné par le choix des catégories, puisque l'utilisateur a la possibilité de sélectionner plus en détail et selon ses besoins. Par contre, il m'importait de créer un système via l'interface d'administrateur qui rende l'envoi d'une newsletter le plus simple possible. Dans l'interface d'administration, toutes les actions sont désormais également prises en charge par un message. Si une newsletter est envoyée, un message de confirmation apparaît également pour l'administrateur.

5.7 Priorité 6

Un utilisateur inscrit a accès à une série des actions qu'un utilisateur non inscrit n'a pas. L'utilisateur peut alors, par exemple, ajouter un article au panier ou le commenter.

Ces actions ne sont affichées sur la page de détail d'un article que si l'utilisateur s'est authentifié.

Si l'utilisateur n'est pas authentifié, un message apparaît à la place de la barre d'action qui attire l'attention sur le fait qu'on doit être connecté pour pouvoir effectuer les actions.

Les différentes actions que l'utilisateur a effectuées sont alors récapitulées sur la page de profil de l'utilisateur, par exemple une liste des articles que l'utilisateur a commentés ou achetés.

5.7.1 CU12 : Commenter un article

Le Principe

L'utilisateur a la possibilité de commenter un article. Le commentaire n'est pas activé directement, mais après vérification du contenu par l'administrateur.

Les commentaires respectifs sont alors affichés dans le système d'onglet, à côté de la description de l'article et de la liste des ingrédients.

Approche pratique

La première action est "commenter" et permet à l'utilisateur de commenter un article. Si l'utilisateur clique ensuite sur "commenter", un petit formulaire pop-up s'ouvre dans lequel il peut saisir un commentaire. Cette animation pop-up a été créée via JAVASCRIPT.

Lorsque l'utilisateur a maintenant saisi son commentaire et cliqué sur soumettre, une route est consultée dans "ArticleController" dans laquelle un nouveau commentaire est créé. En même temps, l'ID de l'article est également transféré. Le commentaire est alors affecté à cet article et à l'utilisateur actuellement connecté.

Pour activer la vérification dans l'interface d'administrateur, j'ai ajouté une action aux commentaires: "publier".

Semblable aux newsletters, le commentaire contient une propriété "statutPubier", un booléen qui est par défaut à "false".

En utilisant l'action publier, ce booléen change vers "true", l'action disparaît (on n'a pas à authoriser un commentaire deux fois) et enfin le commentaire est activé et publié.

En conséquence, ce commentaire est désormais affiché dans la section "Commentaires" dans le détail de l'article et est lisible par tout le monde. Le nom de l'utilisateur et la date du commentaire sont publiés en tant qu'informations supplémentaires sur le commentaire. Les commentaires d'un article sont triés par date.

De plus, cet article sera ultérieurement répertorié dans le profil de l'utilisateur sous les articles commentés. Là, l'utilisateur a la possibilité de modifier à nouveau le commentaire.

5.7.2 CU13 : Noter un article

Le Principe

Un utilisateur enregistré peut également soumettre une note pour un article. Le système de notation est structuré de manière à ce que l'utilisateur puisse choisir un nombre d'étoiles entre 1 et 5.

L'utilisateur ne peut attribuer qu'une seule note à un article. S'il essaie à nouveau de noter le même article, il reçoit un message indiquant qu'il a déjà noté cet article.

La note est anonyme et est finalement incluse dans le calcul d'une note moyenne.

Le nombre total des notations et la notation moyenne qui en résulte sont affichés à titre d'information dans la zone "Description" du détail de l'article.

Approche pratique

Si l'utilisateur sélectionne cette option, un formulaire Pop-Up s'ouvre à nouveau. L'animation est également définie via JAVASCRIPT.

La différence avec les commentaires n'est que dans le traitement. Aucune vérification n'est requise pour les notations. L'utilisateur peut facilement définir un nombre d'étoiles à l'aide d'un système d'étoiles, qui doit correspondre à la qualité de l'article.

Une autre particularité du formulaire est un champ caché qui enregistre automatiquement le nombre d'étoiles.

Toutes les données sont désormais récupérées dans le "ArticleController", c'est à dire l'article via son ID, l'utilisateur connecté et le nombre d'étoiles du champ masqué du formulaire.

Cependant, il faut d'abord vérifier si l'utilisateur a déjà soumis une évaluation pour cet article. Si c'est le cas, il est redirigé vers le détail de l'article avec un message d'avertissement. Si ce n'est pas le cas, la note est enregistrée directement dans la base de données et est directement prise en compte lors de l'affichage du nombre de notes et de la note moyenne de cet article.

Tout comme pour les commentaires, les articles notés seront ensuite également répertoriées dans le profil de l'utilisateur. Là, l'utilisateur peut ajuster ou supprimer une évaluation.

5.7.3 CU14 : Favoriser un article

Le Principe

Un utilisateur enregistré peut ajouter un article à ses favoris.

Cette action fait apparaître une boîte de confirmation demandant si l'article doit être ajouté aux favoris. Si l'utilisateur confirme, l'article sera ajouté aux favoris et un message confirmera l'action.

Un article ne peut pas être ajouté deux fois en tant que favori. Si l'utilisateur essaie de le faire, il recevra un message indiquant que cet article a déjà été ajouté en tant que favori.

Approche pratique

Ici, je l'ai configuré pour que lorsqu'un utilisateur sélectionne cette option, une boîte de confirmation s'ouvre, permettant à l'utilisateur de confirmer son choix.

S'il confirme, son action sera ensuite traitée dans le "ArticleController". Comme pour les notes, la première étape consiste à vérifier si l'utilisateur a déjà choisi cet article comme favori.

Si c'est le cas, il sera redirigé vers le détail de l'article avec un message approprié. Si l'utilisateur n'a pas encore sélectionné cet article comme favori, cet article sera ajouté à l'utilisateur en tant que favori.

Les articles favoris sont ensuite également répertoriés dans le profil de l'utilisateur.

5.7.4 CU15 : Ajouter un article au panier

Le Principe

Une autre action disponible pour les utilisateurs enregistrés consiste à ajouter un article au panier.

Dès qu'un utilisateur est authentifié (en tant qu'utilisateur normal ou administrateur), un panier apparaît dans la barre de navigation.

S'il ajoute un article au panier via la barre d'action, le nombre d'articles dans le panier est également mis à jour et affiché visuellement.

De cette manière, l'utilisateur peut à tout moment se faire une idée du nombre d'articles se trouvant actuellement dans son panier.

Approche pratique

La première étape à ce stade était de créer un "PanierController".

Si l'utilisateur souhaite maintenant ajouter un article au panier, il doit à nouveau confirmer son choix via une boîte de confirmation. S'il le fait, le traitement continue dans le "PanierController" sous la route "add".

À ce stade, j'ai travaillé avec la session. Plus précisément, je crée plusieurs éléments dans la session où je peux mettre en cache des valeurs.

Par exemple, j'enregistre le panier sous forme de tableau associatif dans la session. Le but est de mettre à jour le panier dans la session à chaque fois qu'un article est ajouté.

D'autres valeurs sont également stockées dans la session, comme le nombre total d'articles dans le panier. Ces informations sont indispensables pour créer le panier si l'utilisateur souhaite le consulter.

La première étape de la fonction "add" consiste à vérifier si ces éléments sont déjà présents dans la session.

De plus, je transmets l'ID de l'article à travers la route pour recevoir facilement toutes les informations sur cet article dont j'ai besoin pour un traitement ultérieur. Cependant, je vérifie également si l'article existe même, car un utilisateur peut facilement modifier l'ID dans l'URL. Si l'article existe, il sera ajouté au panier dans la session; si ce n'est pas le cas, l'utilisateur sera redirigé avec un message correspondant.

Si cet article existe, il est vérifié si l'article est déjà dans le panier. Si c'est le cas, son nombre n'est augmenté que de 1. Si l'article n'est pas encore dans le panier, il sera ajouté avec le nombre 1.

Le nombre total d'articles dans le panier est également ajusté en conséquence. Si toutes les valeurs sont ajustées, elles sont enregistrées dans le panier de la session.

L'utilisateur peut ajouter un article au panier soit via le détail de l'article, soit via les listes du profil utilisateur.

Le nombre total d'articles, constamment mis à jour dans la session, est ajouté au panier dans le menu de navigation. De cette façon, l'utilisateur peut constamment voir combien d'articles se trouvent dans le panier. J'ai géré cela directement dans le fichier TWIG "base.html.twig".

Si l'utilisateur consulte maintenant le panier, celui-ci sera créé sur la base des informations de la session.

Afin de collecter toutes les informations sur un article, j'ai créé ma propre fonction dans le "PanierController". Elle s'appelle "infoArticlePanier()" et prend le panier en paramètre.

Cette fonction boucle le panier, parcourt tous les articles, extrait toutes les informations sur l'article de la base de données et calcule tous les montants (prix, frais de livraison,...). À la fin, j'appelle cette fonction dans la route "panier", qui donne un tableau avec toutes les informations pertinentes que je transmets au fichier TWIG.

5.7.5 CU16 : Gestion du panier

Le Principe

Tous les articles ajoutés sont affichés dans le panier sous la forme d'un tableau.

Le prix net, le taux de TVA, le prix TTC, le nombre d'articles et le prix final (prix TTC x nombre) sont affichés pour chaque article.

Il est également possible d'ajuster le nombre vers le haut ou vers le bas (jusqu'à un minimum de 1) ou d'utiliser l'action "supprimer" pour supprimer l'intégralité de l'article du panier.

Le montant total de tous les articles est affiché dans la partie inférieure du tableau.

Si l'utilisateur modifie le nombre d'articles, l'affichage dans la barre de navigation est également mis à jour afin que l'utilisateur ait toujours le nombre total d'articles en vue en un coup d'œil.

Approche pratique

La gestion de panier inclut la gestion de quantité de l'article.

Pour implémenter cela, j'ai ajouté un bouton + et un bouton - au nombre d'articles. Ces boutons conduisent à leur tour au "PanierController", soit à la route "add", soit à la route "remove".

À ce stade, j'ai également décidé d'utiliser AJAX. Ma première étape a été d'appliquer un "EventListener" aux boutons et de récupérer l'URL de l'attribut "href". Cette URL est ensuite utilisée dans une requête AJAX. J'ai créé pour ce traitement une fonction "changeQuantite()".

Dans une dernière étape, j'ai dû différencier dans les routes s'il s'agissait d'une requête normale ou d'une requête AJAX. Les modifications dans "ajouter" ou "supprimer" restent toujours les mêmes, de sorte que les informations relatives au panier sont mises à jour dans la session et rendues en réponse à la requête AJAX.

Si l'utilisateur modifie la quantité d'un article, la page n'est pas rechargée, seulement rafraîchie. Dans le même temps, cependant, grâce aux valeurs constamment mises à jour dans la session, tous les montants sont ajustés au nombre d'articles.

Si l'utilisateur a maintenant réglé son panier, il procède au paiement. Le panier mis à jour dans la session servira de base de paiement.

5.7.6 CU17 : Acheter un article

Le Principe

Après que l'utilisateur a rempli son panier et, si nécessaire, ajusté le nombre d'articles, il peut procéder au paiement via le bouton "payer".

Ce bouton est situé à côté du montant total et redirige l'utilisateur vers la plateforme externe Stripe. Stripe est une infrastructure de paiement web.

L'ensemble du processus de paiement est pris en charge par Stripe, par exemple l'authentification via 3D Secure 2.

Si le paiement est réussi, l'utilisateur est redirigé vers la page SoapOpera avec un message correspondant.

En cas de problème, comme un manque d'argent sur la carte, l'utilisateur reçoit un message directement sur l'interface Stripe ou est redirigé vers la page SoapOpera avec un message d'erreur correspondant.

Si le paiement est réussi, certaines données de la base de données seront mises à jour en interne (nombre des articles vendus, les facture,…) et le panier dans la Session sera vidé.

Sur la plate-forme Stripe, le futur administrateur peut également voir des statistiques relatives aux paiements.

Approche pratique

a) Configuration de Stripe

Tout d'abord, j'ai dû me familiariser avec le système car je n'ai jamais créé de système de paiement pour un site Web. J'ai largement consulté la documentation de Stripe[22].

Au départ, comme pour l'authentification via Google, j'ai dû créer un compte chez Stripe. Ce compte m'a permis d'effectuer des paiements de test. Lorsque je vais passer en production plus tard, je peux activer le compte pour passer du mode test au mode réel.

Après cela, j'ai dû installer Stripe dans mon projet via:

"composer requires stripe/stripe-php"

Dans le fichier ".env", j'ai ensuite dû définir ma clé secrète, que j'ai reçue de Stripe lors de la création du projet.

Dans le fichier de configuration "services.yaml", je devais encore lier cette clé en tant que service.

b) Le traitement

Maintenant que les configurations de base ont été créées, j'ai créé un "PaiementController" dans une étape supplémentaire. Dans ce contrôleur, j'ai récupéré la clé et défini la session de Stripe.

Cet objet contient toutes les informations sur le processus de paiement (la devise, le montant total,...). Ces informations sont maintenant transmises à Stripe.

L'utilisateur est alors redirigé vers la plateforme Stripe pour effectuer son paiement. Les routes "success" et "cancel" définissent les routes vers lesquelles l'utilisateur est redirigé en fonction du succès du paiement.

S'il y a eu un problème lors du processus de paiement, l'utilisateur est redirigé vers la route "cancel" avec un message d'erreur.

[22] Stripe Contributors (s.d.) Online-Zahlungen. https://stripe.com/docs/payments/accept-a-payment

Si le paiement réussit, la route "succès" est appelé, dans laquelle la deuxième partie du traitement a lieu.

Dans la deuxième partie du traitement, les entrées pour la base de données sont maintenant créées. D'une part, une facture est créée sur la base des informations de la session et, d'autre part, une commande est créée pour chaque article du panier. Toutes les commandes sont liées à la facture.

Également sur la base des informations de la session, un e-mail automatique est créé, qui est envoyé à l'utilisateur après paiement. Les informations sont présentées d'une part sous forme de tableau dans le mail comme un résumé et d'autre part envoyées au format PDF sous forme de facture réelle en pièce jointe. Le même e-mail de confirmation est également envoyé à la personne responsable chez Soap-Opera.

Dans l'interface administrateur, l'administrateur peut désormais également consulter les factures et les commandes associées. L'entité facture contient 2 booléens. Une fois pour le statut de paiement, une fois pour le statut de livraison. Le statut de paiement est directement mis sur "true" après un paiement réussi. Le statut de livraison reste sur "false" pour le moment.

Pour changer le statut de livraison en "true", j'ai utilisé la même méthode que pour activer les commentaires et envoyer des newsletters. J'ai ajouté une nouvelle action aux factures: délivrer.

Si l'administrateur utilise maintenant cette action, le booléen "statutLivraison" passe vers "true" et un second mail automatique est envoyé à l'utilisateur l'informant que la marchandise liée à la facture XY a été expédiée.

À ce stade, l'ensemble du processus de commande est terminé. Un utilisateur a commandé un article, l'a payé directement, puis toutes les étapes suivantes sont prises en charge automatiquement. La facture et les commandes sont créées et des mails automatiques sont envoyés. Enfin, l'administrateur a toujours la possibilité de modifier l'état de la livraison et cela est également communiqué de manière transparente à l'utilisateur via un e-mail automatique.

L'administrateur va également recevoir la facture, une fois via un email automatique, une fois directement chargeable via la liste des factures dans l'interface administrateur.

5.8 Priorité 7

5.8.1 CU18 : Profile utilisateur

Le Principe

Un utilisateur connecté a la possibilité de consulter son profil via le menu de navigation. Là, il peut d'une part ajuster ou modifier ses données personnelles, telles que son adresse ou son mot de passe, ou d'autre part recevoir un aperçu de ses interactions avec les articles, par exemple les articles qu'il a sélectionnés comme favoris.

Les articles répertoriés comme favoris ou comme articles achetés peuvent également être ajoutés directement au panier. Cela permet à un utilisateur d'acheter plus rapidement via son profil des favoris ou des articles déjà achetés.

Ce raccourci dans le processus d'achat est destiné à augmenter l'expérience d'utilisateur.

Approche pratique

J'ai d'abord créé un "UtilisateurController" dans lequel je collecte toutes les informations nécessaires pour remplir le profil de l'utilisateur avec ses données personnelles.

Ces informations sont ensuite transmises au fichier TWIG "profile.html.twig".

a) Les informations personnelles

Les informations personnelles sont divisées en deux zones. D'une part, toutes les données personnelles générales telles que le nom et le prénom sont affichées, d'autre part les adresses sont affichées séparément.

Les données personnelles ne sont pas seulement affichées, l'utilisateur a également la possibilité de modifier ces données. Pour cela j'ai créé 2 formulaires à l'aide de la commande "php bin/console make:form", un basé sur l'entité utilisateur, un basé sur l'entité adresse.

Un troisième formulaire a été créé pour également changer le mot de passe via le profil. L'utilisateur a également la possibilité ici de supprimer complétement son compte.

a1) Les informations générales

Le formulaire est structuré de la même manière que le formulaire d'inscription, mais sans adresse, car celles-ci peuvent être modifiées séparément.
Les champs sont validés de la même manière que pour le formulaire d'inscription.

a2) Les adresses

L'utilisateur a également la possibilité de modifier séparément son adresse personnelle ou son adresse de livraison. Pour les adresses, j'ai défini une route distincte dans l'"UtilisateurController" qui, comme lors de l'inscription, vérifie d'abord si l'adresse est déjà dans la base de données, si c'est le cas, l'adresse est attribuée directement à l'utilisateur. Si ce n'est pas le cas, l'adresse est d'abord créée dans la base de données puis attribuée à l'utilisateur.

Pour les pays, j'ai utilisé le type de champ "CountryType", qui par définition reflète tous les pays dans une sélection. J'ai limité cette liste de pays à la Belgique et à l'Allemagne en utilisant un "choice_filter" car les clients ne viennent que de Belgique et d'Allemagne.

Les champs sont validés de la même manière que pour le formulaire d'inscription, ainsi que la validation de la relation entre le code postal et le pays.

a3) Le mot de passe

En plus de pouvoir changer le mot de passe via le formulaire d'authentification, on peut également le changer dans le profil d'utilisateur.

Pour assurer la validation uniquement du mot de passe, j'ai créé un deuxième groupe de validation (reset) qui permet de valider uniquement la propriété mot de passe de l'entité utilisateur.

Cependant, aucune validation n'a lieu par mail, puisque la personne est déjà authentifiée dans ce cas.

a4) Supprimer le profile

Enfin, l'utilisateur a également la possibilité de supprimer complètement son profil. Il doit ensuite le confirmer via une boîte de confirmation.

Lorsqu'un utilisateur supprime son profil, tous les éléments qui le concernent sont également automatiquement supprimés, par exemple tous les commentaires, notes, factures, commandes, etc. Les adresses sont une exception, puisqu'un autre utilisateur peut également avoir cette adresse.

J'ai décidé de le faire parce que je suppose que lorsqu'un utilisateur supprime son profil, toutes les connexions à cet utilisateur dans la base de données seront également supprimées.

C'est aussi la raison pour laquelle Soap-Opera reçoit également une confirmation après un achat avec une facture PDF jointe. Cela garantit toujours que la comptabilité est complète, car il peut également arriver qu'un utilisateur supprime son profil immédiatement après un achat. Dans ce cas, la facture a peut-être été supprimée avant que l'administrateur ne puisse la télécharger depuis le menu liste de l'interface d'administration.

C'est peut-être un cas qui n'arrivera jamais, mais cela signifie que la comptabilité de Soap-Opera est toujours correcte et qu'aucune facture ne peut être perdue.

b) Les listes des articles

Il y a 4 actions qu'un utilisateur peut effectuer. Il peut commenter un article, le noter, le mettre en favori ou l'ajouter au panier. J'ai pensé qu'il pourrait être intéressant de lister tous les articles liés aux différentes actions dans le profil et d'autoriser certaines actions (rajouter au panier, modifier, supprimer, etc.).

b1) Les favoris

Les articles favorisés de l'utilisateur sont récupérés dans l'"UtilisateurController" et transmises vers le fichier TWIG "profile.html.twig". Ils sont ensuite affichés sous forme de tableau. L'image, le nom (sous forme de lien vers l'article détaillé) et les actions sont affichés. Les actions dans ce cas sont "ajouter au panier" et "supprimer". L'ajout au panier devrait augmenter l'expérience de l'utilisateur, puisqu'il n'a plus à chercher ses favoris. Un favori peut être supprimé via une boîte de confirmation.

Si l'utilisateur n'a pas de favoris, un message s'affiche pour la section indiquant que l'utilisateur n'a pas de favoris pour le moment.

b2) Les articles achetés

Les articles achetés de l'utilisateur sont récupérés dans l'UtilisateurController et transmises vers le fichier TWIG "profile.html.twig". Ils sont ensuite affichés sous forme de tableau. L'image, le nom (sous forme de lien vers l'article détaillé) et les actions sont affichés. La seule action dans la liste est "ajouter au panier" car la supprimer ou la modifier n'a aucun sens.

La liste de tous les articles achetés par un utilisateur est d'abord filtrée dans l'UtilisateurController avec la fonction "array_unique", car sinon un article apparaîtra plus d'une fois dans la liste s'il a été acheté plus d'une fois. La liste ici est uniquement destinée à donner un aperçu de tous les articles qui ont été achetés au moins une fois.

Si l'utilisateur n'a pas des articles achetés, un message s'affiche pour la section indiquant que l'utilisateur n'a pas des articles achetés pour le moment.

b3) Les articles commentés

Les articles commentés de l'utilisateur sont récupérés dans l'UtilisateurController et transmises vers le fichier TWIG "profile.html.twig". Ils sont ensuite affichés sous forme de tableau. L'image, le nom (sous forme de lien vers l'article détaillé), le commentaire et les actions sont affichés. Les actions dans ce cas sont "ajouter au panier", "modifier" et "supprimer".

Un commentaire sur un article peut également être modifié. J'ai créé mon propre formulaire pour cela, similaire à l'action "commentaire" sur le détail de l'article.

Si l'utilisateur décide de modifier un commentaire, il peut le faire via un formulaire qui s'ouvre lorsqu'il choisit l'action "modifier". Le commentaire modifié suivra le même processus que le commentaire d'origine, c'est-à-dire qu'il ne sera publié qu'après vérification par l'administrateur.

Un commentaire peut être supprimé via une boîte de confirmation.

J'ai créé un "CommentaireController" pour les actions "modifier" et "supprimer".

b4) Les articles notés

Les articles notés de l'utilisateur sont récupérés dans l'UtilisateurController et transmises vers le fichier TWIG "profile.html.twig". Ils sont ensuite affichés sous forme de tableau. L'image, le nom (sous forme de lien vers l'article détaillé), la note et les actions sont affichés. Les actions dans ce cas sont "ajouter au panier", "modifier" et "supprimer".

Une notation sur un article peut également être modifiée. J'ai créé mon propre formulaire pour cela, similaire à l'action "noter" sur le détail de l'article.

Si l'utilisateur décide de modifier une notation, il peut le faire via un formulaire qui s'ouvre lorsqu'il choisit l'action "modifier". La notation modifiée est appliquée directement et également prise en compte dans le calcul de la note moyenne pour cet article.

Une notation peut être supprimée via une boîte de confirmation.

J'ai créé un "EvaluationController" pour les actions "modifier" et "supprimer".

5.9 Priorité 8

5.9.1 CU19 : Consulter des statistiques

Le principe

L'administrateur peut consulter diverses statistiques dès qu'il est sur l'interface d'administration.

Approche pratique

L'idée ici est de personnaliser la page d'accueil de l'interface d'administration sans modifier le menu de gauche.

Pour mettre cela en œuvre, j'ai dû copier une partie des modèles (Templates) EasyAdmin du dossier "Vendor" dans ma propre structure de modèles. Cela m'a donné accès aux fichiers modèles et j'ai ensuite pu les concevoir en fonction de mes propres besoins.

Pour pouvoir afficher les statistiques, j'ai utilisé comme modèle le fichier "welcome.html.twig".

Après avoir défini la structure de base dans ce modèle, la deuxième étape consistait à préparer toutes les requêtes à la base de données dans le "DashboardController", qui fournit les informations nécessaires pour alimenter les statistiques.

Après quelques tests pour voir si j'obtenais les bons résultats, j'ai ensuite ajouté un bouton au modèle, qui affiche ensuite les statistiques respectives lorsqu'il est utilisé.

Sur ce bouton j'ai ensuite défini un "EventListener" en JAVASCRIPT, qui envoie une requête Ajax à l'URL (route du DashboardController) et reçoit en réponse toutes les informations pertinentes pour les statistiques.

Une fois toutes les données reçues, les différents contenus des statistiques sont créés et alimentés.

Pour les statistiques générales, j'ai créé des requêtes simples vers la base de données et traité directement les données pour les afficher. Pour les autres statistiques, j'ai utilisé la bibliothèque chart.js pour afficher des données diverses, et pour la plupart plus complexes, plus joliment à l'aide de graphiques ("pie", "bar").

Pour implémenter les différents graphiques, j'ai consulté la documentation de chart.js [23] pour comprendre le principe et paramétrer les graphiques selon mes données.

Voici deux définitions:

Pie: "Pie and doughnut charts are probably the most commonly used charts. They are divided into segments, the arc of each segment shows the proportional value of each piece of data. They are excellent at showing the relational proportions between data." [24]

Bar: "A bar chart provides a way of showing data values represented as vertical bars. It is sometimes used to show trend data, and the comparison of multiple data sets side by side." [25]

[23] Chart.js Contributors (2022) Chart.js
https://www.chartjs.org/docs/latest/

[24] Chart.js Contributors (2022) Doughnuts and Pie Charts.
https://www.chartjs.org/docs/latest/charts/doughnut.html

[25] Chart.js Contributors (2022) Bar Charts.
https://www.chartjs.org/docs/latest/charts/bar.html

5.10 Priorité 9

5.10.1 CU20: Consulter les points de ventes

Le principe

Un utilisateur a la possibilité de consulter les points de vente via la barre de navigation.

Dans l'affichage des points de vente, une carte est mise à disposition pour chaque point de vente individuel. Ainsi, l'utilisateur peut facilement trouver l'endroit.

Ces points de vente comprennent des magasins, des marchés,...

Approche pratique

Dans un premier temps, j'ai dû ajuster la base de données. J'ai supprimé la table partenaire (avec toutes les relations) et en ai créé une nouvelle, que j'ai nommée Points de vente.

L'idée est quasiment la même que celle d'origine, sauf qu'ici je peux regrouper tous les lieux (boutiques, marchés,...) où les articles sont proposés et vendus.

Après avoir également adapté l'interface d'administration à cet égard, j'ai créé les points de vente en fonction des informations (nom, adresse,...) qui m'ont été fournies par le client.

Maintenant que j'ai créé tous les points de ventes, j'ai créé un contrôleur de points de ventes pour obtenir toutes les informations nécessaires de la base de données, puis je les ai transmis à la vue pour afficher visuellement les points de ventes.

Les adresses sont également affichées, également sous forme de carte. À cet égard, j'ai géocodage avec "OpenStreetMaps", la bibliothèque JAVASCRIPT "Leaflet"[26] et l'API" Mapbox"[27].

Concrètement, je crée un container dans le fichier TWIG pour chaque point de vente, chacun contenant un espace pour la carte. J'attribue à cette zone la

[26] Narges Mirzaaghaei (2019, 24 jan) How to embed Open Street Maps in a webpage. https://medium.com/@nargessmi87/how-to-embede-open-street-map-in-a-webpage-like-google-maps-8968fdad7fe4

[27] Mapbox Contributors (s.d.) Documentation. https://docs.mapbox.com/

longitude et la latitude de l'adresse respective via des attributs de données (data-).[28]

Chaque adresse est traitée via l'API Mapbox dans le contrôleur pour filtrer les informations sur cette adresse, y compris la longitude et la latitude.

Pour me faciliter les choses, j'ai ajouté la longitude et la latitude comme propriétés à l'entité point de vente. Ainsi, si je parcours tous les points de vente récupérés dans le contrôleur en boucle, je récupère la longitude et la latitude en fonction de l'adresse transmise à l'API. Cela me permet de déterminer la longitude et la latitude de chaque point de vente et de les enregistrer en même temps.

Ensuite, je peux facilement définir la longitude et la latitude en tant qu'attributs de données pour chaque point de vente dans le fichier TWIG lors de la traversée des points de vente.

L'adresse est transmise à l'API pour obtenir les informations. Mais j'ai d'abord dû traiter l'adresse avec la fonction "urlencode()" afin que les caractères spéciaux puissent être lus via l'URL.

Ensuite, je récupère le contenu de ces attributs de données en JAVASCRIPT et avec ces informations, la carte est ensuite créée dans la zone respective en fonction de l'adresse du point de vente.

[28] MDN Contributors(2022, 29 jul) Using data attributes. https://developer.mozilla.org/en-US/docs/Learn/HTML/Howto/Use_data_attributes

5.11 Priorité 10

5.11.1 CU21 : Consulter notre philosophie

Le principe

Un utilisateur a toujours la possibilité de consulter la philosophie de Soap-Opera via le menu de navigation.

Il s'agit d'une page "à-propos" classique qui permet à l'utilisateur de comprendre la philosophie du projet.

Un diaporama est également prévu, qui souligne à nouveau la philosophie et présente Soap-Opera dans le travail quotidien.

Approche pratique

Afin de donner à l'administrateur la possibilité de créer ou de modifier lui-même les textes de la philosophie à long terme, j'ai créé une entité supplémentaire (similaire aux tables de traduction).

J'ai ensuite ajouté cette entité au menu d'administration afin qu'un administrateur puisse désormais créer le texte concernant la philosophie dans différentes langues.

Pour afficher ces textes, j'ai créé un "PhilosophieController" dans lequel je récupère la philosophie dans la langue respective qui est actuellement stockée dans la session et la transmet au fichier TWIG.

De plus, un texte standard est ajouté au fichier TWIG, ce qui indique que Soap-Opera propose également divers ateliers. L'utilisateur peut ensuite utiliser un bouton pour accéder au formulaire de contact afin de poser d'autres questions sur les ateliers.

L'idée initiale qu'un utilisateur pouvait réserver une place pour un atelier directement via un calendrier a été abandonnée lors de discussions intensives avec le client. La raison était que le client se sentait dépassé par cette option et ne pouvait pas évaluer comment les réservations affecteraient sa vie privée.

Une option de transition que j'ai suggérée était d'abord d'informer les utilisateurs de la possibilité d'ateliers et de collecter des informations à ce sujet via le formulaire de contact.

L'idée derrière cette suggestion était que le comportement de l'utilisateur est observé à cet égard et, si nécessaire, les options à ce stade sont adaptées au comportement à un moment ultérieur. Les questions suivantes doivent toujours être posées : Y a-t-il beaucoup de demandes ? Comment se passe la

coordination avec la vie privée ? Quand exactement les ateliers sont-ils offerts? Etc...

À cet égard, j'ai dû faire quelques changements dans la base de données, certaines tables ont dû être supprimées et de nouvelles créées.

Pour le diaporama, je me suis inspiré d'un exemple sur w3schools.com.[29] Les images correspondantes m'ont été fournies par Soap-Opera et je les ai attribuées via l'interface d'administrateur. Pour ce faire, j'ai créé une table "Position Image" dans la base de données.

Généralement, les images sont associées à un élément spécifique, comme un article ou un point de vente. Le tableau "Position Image" permet également d'affecter une image à une position, par exemple le diaporama de la page philosophie.

Une fois la position créée, j'ai pu insérer les images, leur attribuer cette position et enfin récupérer les images dans le contrôleur de philosophie pour créer le diaporama.

[29] W3schools Contributors (s.d.) How TO-Slideshow. https://www.w3schools.com/howto/howto_js_slideshow.asp

5.12 Priorité 11

5.12.1 CU22 : Mentions légales

Le principe

L'utilisateur peut consulter les mentions légales à tout moment via le pied de page du site.

Approche pratique

J'ai d'abord créé un "MentionsLegalesController" et y ai préparé une route. J'ai ensuite créé un lien vers cette route dans le fichier TWIG "base.html.twig" dans la zone de pied de page.

La route mène directement au fichier TWIG "mention_legales.html.twig", où j'ai inséré les textes respectifs. J'ai un modèle pour les textes de mon stage, que j'ai ensuite adapté au projet.

5.12.2 CU23 : Politique de confidentialité

Le principe

L'utilisateur peut consulter la politique de confidentialité à tout moment via le pied de page du site.

Approche pratique

J'ai d'abord créé un "PolitiqueConfidentController" et y ai préparé une route. J'ai ensuite créé un lien vers cette route dans le fichier TWIG "base.html.twig" dans la zone de pied de page.

La route mène directement au fichier TWIG "politique_confident.html.twig", où j'ai inséré les textes respectifs. J'ai un modèle pour les textes de mon stage, que j'ai ensuite adapté au projet.

5.12.3 CU24: Les conditions de vente (CGV)

Le principe

L'utilisateur peut consulter les conditions de vente à tout moment via le pied de page du site.

Approche pratique

J'ai d'abord créé un "CGVController" et y ai préparé une route. J'ai ensuite créé un lien vers cette route dans le fichier TWIG "base.html.twig" dans la zone de pied de page.

La route mène directement au fichier TWIG "cgv.html.twig", où j'ai inséré les textes respectifs. J'ai un modèle pour les textes de mon stage, que j'ai ensuite adapté au projet.

6 Déploiement du projet

Maintenant que toutes les fonctionnalités ont été créées et testées en phase de développement, l'étape suivante consistait à mettre le projet en ligne.

J'ai pu placer le projet sur "one.com" grâce à un ami.

6.1 La préparation

J'ai d'abord fait une copie du projet, d'une part pour garder la structure de base du projet en mode 'développement' et d'autre part pour avoir une structure que je puisse adapter à des besoins spécifiques pour que le projet puisse fonctionner en ligne.

6.2 La base de données

Nous (mon ami et moi) avons d'abord créé une base de données sur le serveur. Après cela, j'ai exporté et importé ma base de données sous forme de fichier "SQL". Dans une dernière étape, j'ai dû créer un accès à cette nouvelle base de données dans le fichier .env.

6.3 Le "upload" du code

Après avoir créé un sous-dossier et la base de données sur le serveur et modifié l'accès à la base de données en interne dans le code, nous avons téléchargé le projet sur le serveur via "Filezilla".

6.3.1 Webpack Encore

J'ai d'abord minifié le fichier CSS et le fichier JS via la commande "yarn run encore production" avant le "upload".

Étant donné que le projet se trouve dans un sous-dossier, j'ai dû ajuster le fichier "entrypoints.json" dans le dossier "build" dans le dossier "public".

6.3.2 Les images

En utilisant la même logique, partout où une image est affichée, j'ai dû ajuster le chemin vers celle-ci.

6.4 La redirection des routes

Afin d'assurer une redirection correcte des routes, j'ai également dû installer le bundle suivant : "symfony/apache-pack".

Ce bundle installe un fichier ".htaccess" directement dans le dossier "public", ce qui crée automatiquement une redirection correcte entre les routes. J'ai profité de ce fichier pour aussi forcer le "http" en "https".

6.5 Les mails

En mode développement, j'ai défini mes données d'accès personnelles comme "MAILER_DSN" dans le fichier ".env". Ici, cependant, le serveur a bloqué l'accès à ce "MAILER_DSN" pour des raisons de sécurité (spam). Mon ami m'a alors donné les données d'accès pour l'adresse e-mail interne sur le serveur afin que la livraison des e-mails automatiques puisse continuer à fonctionner sans problème.

6.6 Authentification via Google

Pour que l'authentification via un compte de Google continue de fonctionner, j'ai dû ajuster les URL dans l'outil de développement de Google.

6.7 Les tests

Puisque tout fonctionnait en mode développement, le but était de s'assurer que tout fonctionnait également en mode en ligne. Pour tester cela, j'ai testé à nouveau tous les cas d'utilisation. Ensuite, j'ai testé le site sur différents navigateurs et testé différentes vues de téléphone portable dans la zone "inspecter" pour m'assurer que le facteur "responsive" fonctionnait.

7 La conclusion finale

En créant cette page j'ai appris beaucoup de choses à plusieurs niveaux.

7.1 La gestion du projet

Tout d'abord, je souhaitais définir un workflow précis pour ne pas me "perdre" dans le projet en lui-même. Suivre une procédure précise me semblait la meilleure solution. Définir ce processus en priorités, était exactement ce que j'avais en tête. Comme j'ai utilisé une approche similaire pour créer le projet dans le cours 'projet web dynamique', j'avais déjà un peu de pratique.

7.2 Les technologies

Avant tout, je souhaitais approfondir et comprendre le Framework "Symfony".

Mais il s'agissait aussi de faire la connaissance avec d'autres technologies, comme par exemple chart.js pour les statistiques. Nous avons appris à connaître un peu cette technologie en deuxième année et je voulais l'utiliser dans le projet pour la tester.

De plus, j'étais également soucieux de comprendre encore mieux les liens entre les technologies, mais aussi d'intégrer d'une certaine manière tout ce que nous avons vu au cours des trois années de formation et donc de le comprendre encore mieux.

Donc, en résumé pour les technologies, je peux dire que j'ai pu consolider ce que j'avais appris jusqu'à présent, mais j'ai aussi appris beaucoup de nouvelles choses.

7.3 L'échange avec le client

Un autre facteur important était le contact avec un vrai client.

Au début du projet il y a eu une réunion avec le client pour créer ensemble un cahier de charge. Cependant, le projet lui-même a subi des changements importants au cours de la phase de développement. Pourquoi était-ce?

Je pense qu'il y avait beaucoup d'enthousiasme quand le projet a commencé. Et chaque idée était reprise directement dans le cahier de charge. Je ne pense pas avoir été assez critique avec mon client sur ce point et analysé les réels besoins ET limites.

Cependant, j'ai également insisté pour qu'une réunion régulière avec le client ait lieu toutes les 2 semaines pour voir ensemble si le projet va globalement dans le bon sens et si certaines choses doivent être ajustées.

Lors de ces réunions, j'ai pris conscience que certaines fonctionnalités ne pouvaient pas être implémentées. Pas pour des raisons techniques, mais simplement pour des raisons privées de la part du client.

Un exemple:

Le client proposant des ateliers, l'idée de départ était de proposer aux utilisateurs un calendrier pour réserver des ateliers. Dans l'analyse avec le client, cependant, il est vite apparu qu'un tel système dépasse les capacités privées du client. J'ai donc proposé une solution simple. L'utilisateur est informé dans la rubrique "notre philosophie" qu'il existe des ateliers et qu'il peut contacter le client pour toute question complémentaire via le formulaire de contact.

Le but de cette proposition était d'observer dans un premier temps le comportement de l'utilisateur. De nombreuses demandes arriveront-elles ? Y a-t-il une question à propos de ce point? Ultérieurement, les réponses à ces questions pourront être analysées et ce point pourra être ajusté si nécessaire sur le site.

Les réunions régulières ont également servi à fournir au client une vue d'ensemble et à s'assurer que tout était mis en place à sa satisfaction.

En toute autocritique, cependant, je dois dire que l'analyse de la première réunion aurait dû être meilleure et plus globale afin de créer un cahier de charge plus stable.

7.4 Développement personnel

En rassemblant ces différents points, je dois dire que je suis très content de ce projet.

Il y a 3,5 ans, j'ai pris la décision de faire une réorientation professionnelle dans un domaine avec presque 0 connaissances préalables. Ce que j'ai appris pendant la formation a dépassé mes attentes dans un sens positif. Il m'a fallu beaucoup de temps et d'engagement pour me familiariser rapidement avec tous les domaines et comprendre les différentes logiques derrière les différentes technologies, mais tous les efforts en valaient la peine.

Suis-je parfait maintenant? Pas du tout! Je suis également sûr qu'il y a des failles dans ce projet ou que certaines approches ne sont pas réfléchies à 100%, mais dans l'ensemble je suis très fier de pouvoir mener à bien un tel projet tout seul à soulever.

8 Les remerciements

Je tiens à remercier tous les professeurs pour leur engagement, leur patience et leur disponibilité. Dans le cadre de leur disponibilité, ils ont toujours essayé de m'aider avec des questions et des problèmes, tant pendant qu'en dehors des cours, ce qui m'a beaucoup aidé à progresser dans mon processus d'apprentissage.

Je tiens également à remercier mes camarades de classe. Pendant ces trois années, nous avons tous grandi ensemble, échangé des idées sur les différents contenus de la formation et nous nous sommes soutenus là où c'était nécessaire. J'ai rarement connu une ambiance aussi familiale dans un tel contexte.
Sur ce point, je tiens à remercier tout particulièrement une personne dont l'aide et le soutien ont été déterminants pour mieux comprendre et utiliser les différentes notions de programmation : **Pierre-Yves Flamand**.

Je tiens également à remercier :

- **SoapOpera** pour m'avoir donné l'opportunité et la confiance de développer un site web pour leur projet
- **Charléne Counson** pour ses conseils en design.
- **Christophe Nix** pour son aide à la mise en ligne du site.
- **Jenn von Montigny** pour ses images professionnelles.
- **Tous mes amis et ma famille** pour le soutien général pendant la formation.

9 Les ressources

Symfony Contributors (s. d.) Symfony Documentation.
https://symfony.com/doc/current/index.html

Github Contributors (s. d.) Creating a new repository.
https://docs.github.com/en/repositories/creating-and-managing-repositories/creating-a-new-repository

Symfony-Twig Contributors (s. d.) Twig Documentation.
https://twig.symfony.com/doc/3.x/intro.html

Symfony Contributors (s. d.) Symfony Documentation.
https://symfony.com/doc/current/frontend.html

Symfony Contributors (s. d.) Symfony Documentation.
https://symfony.com/doc/current/frontend/encore/simple-example.html

Symfony Contributors (s. d.) Describing the data structure.
https://symfony.com/doc/current/the-fast-track/en/8-doctrine.html

Symfony Contributors (s. d.) EasyAdmin.
https://symfony.com/bundles/EasyAdminBundle/current/index.html

Symfony Contributors (s. d.) Dashboards.
https://symfony.com/bundles/EasyAdminBundle/current/dashboards.html

Symfony Contributors (s. d.) Dashboard Route.
https://symfony.com/bundles/EasyAdminBundle/current/dashboards.html

Symfony Contributors (s. d.) Main Menu.
https://symfony.com/bundles/EasyAdminBundle/current/dashboards.html

Symfony Contributors (s. d.) CRUD Controllers.
https://symfony.com/bundles/EasyAdminBundle/current/crud.html

Symfony Contributors (s. d.) Fields.
https://symfony.com/bundles/EasyAdminBundle/current/crud.html

Symfony Contributors (s. d.) Fields.
https://symfony.com/bundles/EasyAdminBundle/current/fields.html

Symfony Contributors (s. d.) Querying for objects: The repository.
https://symfony.com/doc/current/doctrine.html#querying-for-objects-the-repository

Symfony Contributors (s. d.) Reusing Template Contents.
https://symfony.com/doc/current/templates.html#embedding-controllers

Symfony Contributors (s .d.) Security – Form Login.
https://symfony.com/doc/current/security.html

SymfonyCast Contributors (s. d.) Remember me system.
https://symfonycasts.com/screencast/symfony-security/remember-me

Symfony Contributors (s. d.) Validation groups
https://symfony.com/doc/current/form/validation_groups.html

Symfony Contributors (s. d.) Events and Event Listeners
https://symfony.com/doc/current/event_dispatcher.html

Symfony Contributors (s. d.) Translations.
https://symfony.com/doc/current/translation.html

Nouvelle Techno (2020, 01 feb) Live Coding: Créer un site multilingue avec Symfony 4
https://www.youtube.com/watch?v=JOAiup61byY

Stripe Contributors (s. d.) Online-Zahlungen.
https://stripe.com/docs/payments/accept-a-payment

Chart.js Contributors (2022) Chart.js
https://www.chartjs.org/docs/latest/

Chart.js Contributors (2022) Doughnuts and Pie Charts.
https://www.chartjs.org/docs/latest/charts/doughnut.html

Chart.js Contributors (2022) Bar Charts.
https://www.chartjs.org/docs/latest/charts/bar.html

Narges Mirzaaghaei (2019, 24 jan) How to embed Open Street Maps in a webpage.
https://medium.com/@nargessmi87/how-to-embede-open-street-map-in-a-webpage-like-google-maps-8968fdad7fe4

Mapbox Contributors (s. d.) Documentation.
https://docs.mapbox.com/

MDN Contributers(2022, 29 jul) Using data attributes.
https://developer.mozilla.org/en-US/docs/Learn/HTML/Howto/Use_data_attributes

W3schools Contributors (s. d.) How TO-Slideshow.
https://www.w3schools.com/howto/howto_js_slideshow.asp