



Application de recensement des brasseries locales de France

Projet réalisé dans le cadre du passage du Titre Professionnel de

Développeur Web et Web Mobile

Présenté par

Marie M

O'Clock  – Promo Vanadium (2023)

SOMMAIRE

REMERCIEMENTS	4
INTRODUCTION	5
COMPÉTENCES DU RÉFÉRENTIEL COUVERTES PAR LE PROJET.....	6
ACTIVITÉ TYPE N° 1 : Développer la partie front-end d'une application web ou web mobile en intégrant les recommandations de sécurité	6
- Maquetter une application :	6
- Réaliser une interface utilisateur web statique et adaptable :	6
- Développer une interface utilisateur web dynamique :	6
ACTIVITÉ TYPE N° 2 : Développer la partie back-end d'une application web ou web mobile en intégrant les recommandations de sécurité	7
- Créer une base de données.....	7
- Développer les composants d'accès aux données	7
- Développer la partie back-end d'une application web ou web mobile	7
RÉSUMÉ DU PROJET	8
CAHIER DES CHARGES.....	8
Objectifs	8
Conceptualisation de l'application.....	9
MVP (<i>Minimum Viable Product</i>)	9
Liste des fonctionnalités.....	9
Mise en place de l'application.....	10
Arborescence.....	10
Wireframes.....	11
Charte graphique et logo / Accessibilité	13
Utilisateurs et <i>User Stories</i>	13
Public visé	13
Rôles	13
User Stories	14
Évolutions éventuelles.....	15
SPÉCIFICATIONS TECHNIQUES.....	15
Versioning	15
Choix technologiques	17
Front-End.....	17

Back-end	17
Accessibilité	18
Navigateurs compatibles.....	18
Types d'appareils.....	18
Architecture du projet	19
La Base de Données	19
MCD	20
MLD	20
Dictionnaire de données	20
MPD	21
Routes	22
Déploiement	24
Sécurité de l'application	25
Formulaires et <i>token CSRF</i>	25
Utilisateurs	27
Access Control	28
GESTION DE PROJET	28
Présentation de l'équipe et des rôles	28
Organisation de travail	29
Outils utilisés.....	30
Programme des sprints.....	31
RÉALISATIONS PERSONNELLES.....	33
PRÉSENTATION DU JEU D'ESSAI	41
VEILLE ET RÉSOLUTION DE PROBLÈMES.....	44
Veille technologique sur les vulnérabilités de sécurité.....	44
Résolution de problèmes.....	45
Recherche et traduction	47
CONCLUSION.....	49
ANNEXES.....	50

REMERCIEMENTS

Je tiens à remercier notre tutrice de promo lors de la formation : xxxxxxxx Elle a toujours été à nos côtés. Son soutien lors de mes moments de fatigue et de doutes a été précieux. Elle a toujours su trouver les mots justes, et son énergie et sa bonne humeur ont été plus que communicatifs.

Je tiens aussi à remercier le corps enseignant. En particulier MM xxx, xxx, xxx et xxx qui m'ont énormément appris. Grâce à eux, aujourd'hui, j'arrive à comprendre et écrire du code.

Pour reprendre une citation de xxx : "les erreurs m'ont rendue meilleure".

Merci aussi à mon équipe du projet : xxx, xxx et xxx. Nous avons passé plus de temps ensemble pendant un mois qu'avec notre propre famille et, malgré la distance, avons été très proches. De nouvelles amitiés sont nées de ce projet.

Enfin, je tiens à remercier infiniment ma moitié qui a été mon pilier durant cette formation et la rédaction des dossiers.

Dans mes moments de doutes, de fatigue, de trop-plein, etc., elle était là et m'a soutenue. Elle m'a permis d'aller chercher au fond de moi la force qu'il me restait quand je pensais ne plus pouvoir en faire plus. Sa foi inébranlable, sa confiance et son soutien ont été d'une aide précieuse et un réel moteur pour moi.

Merci, du fond cœur !

INTRODUCTION

J'ai toujours été dans la technologie, l'informatique et les médias. Dès mon plus jeune âge, j'ai vu mon père travailler sur son ordinateur, faire de la photographie, faire de la vidéo et du montage. Je devais avoir 3 ans et j'arrivais déjà à mettre notre télévision en route et à lancer le magnétoscope avec la VHS de mon dessin animé préféré. C'est donc tout naturellement ce vers quoi je me suis dirigée lorsque j'ai fait le choix de ma future vie professionnelle, à savoir : l'audiovisuel.

J'ai fait un BTS Audiovisuel option Technique d'Ingénierie et Exploitations des Équipements et en suis sortie diplômée. Ma carrière professionnelle a commencé la même année que l'obtention de mon diplôme, avec mon premier poste en tant que Technicienne d'Exploitation Vidéo dans une télévision locale près de chez moi.

Après avoir passé cinq années dans diverses télévisions locales en tant que Technicienne d'Exploitation Vidéo, Technicienne Audiovisuel, Ingénierie du Son et Vidéaste, j'ai senti que le milieu de la télévision ne m'attirait plus autant que cela. Je voulais continuer dans le milieu de l'audiovisuel, mais en tant que Vidéaste et Photographe. Malheureusement, les places ne sont pas faciles à décrocher. J'ai commencé à envisager de faire une reconversion. Mais dans quoi? Je n'en avais aucune idée. L'audiovisuel étant mon "premier amour", qu'est-ce que je pourrais bien trouver d'aussi passionnant?

J'ai décroché un poste de photographe automobile et, l'année suivante, l'entreprise a décidé de fermer. Le moment était venu pour moi de profiter de cette occasion pour enfin me reconvertir. Depuis quelque temps, j'entendais beaucoup parler de code, de développement web, et j'ai commencé à m'y intéresser. Je pensais que cela n'était pas pour moi et je me suis souvenue que, dans mes années de télévision, j'étais souvent allée chercher des images pour les journalistes via l'inspecteur de page, parce qu'ils n'arrivaient pas à les récupérer. Au final, je "trifouillais" du code depuis plus longtemps que je ne le pensais.

Je me suis rendue à l'évidence que j'aimais beaucoup le code et que le métier de Développeur Web était la meilleure voie à suivre pour moi. De la création et de la technique? Mais c'est tout moi! J'ai cherché la solution qui pourrait m'aller pour ma reconversion, et une discussion avec ma conseillère m'a définitivement orientée vers O'Clock.

Et me voilà maintenant, quasiment un an plus tard, et j'ai appris énormément de choses. Avec une formation enrichissante qui s'est conclue par le développement de *O'Beer*, l'idée de projet de deux nouvelles amies qui avaient envie de faire une application web sur la Zythologie. Grâce à *O'Beer*, j'ai développé des *hard-skills* et surtout des *soft-skills* et je suis passionnée de code. J'ai hâte de voir où ma vie professionnelle va m'emmener. En tout cas, je sais que ça va coder!

COMPÉTENCES DU RÉFÉRENTIEL COUVERTES PAR LE PROJET

O'Beer est un projet développé avec le framework *Symfony 5.4* et son ORM (*Object-Relational Mapping*) *Doctrine*.

Un *framework* – « cadre de travail » en français – est un ensemble de composants logiciels structurels qui permet aux développeurs d'être plus efficaces dans leur développement. Il offre une architecture et des composants prêts à l'emploi et réutilisables.

ACTIVITÉ TYPE N° 1 : Développer la partie front-end d'une application web ou web mobile en intégrant les recommandations de sécurité

- Maquetter une application :

Nous avons tout d'abord rédigé le cahier des charges et conçu tous les documents de conception de *O'Beer*. Nous avons rédigé les *user stories* et défini le *MVP (Minimum Viable Product, Produit Minimum Viable en français)* qui nous ont permis de créer les *Wireframes* du projet. Ceux-ci ont été créés grâce au l'espace de travail collaboratif *Whimsical*. Enfin, nous avons intégré tous ces documents dans le cahier des charges.

- Réaliser une interface utilisateur web statique et adaptable :

Nous avons ensuite réalisé une intégration statique et *responsive mobile-first* du site (celle-ci n'utilisait que *HTML* et *CSS*). Nous voulions avoir une base de travail avant de l'intégrer dans le framework *Symfony* et de la dynamiser.

- Développer une interface utilisateur web dynamique :

Une fois l'intégration faite sur *Symfony*, nous avons utilisé *JavaScript* pour afficher un formulaire pour que le visiteur valide sa majorité avant d'entrer sur le site. Nous avons également utilisé ce langage pour développer une carte interactive sur la page d'accueil.

ACTIVITÉ TYPE N° 2 : Développer la partie back-end d'une application web ou web mobile en intégrant les recommandations de sécurité

- Créer une base de données

En utilisant les *user stories* et le *MVP*, nous avons ensuite réalisé le *MCD (Modèle Conceptuel des Données)* grâce au site *MoCoDo* qui a automatiquement généré le *MLD (Modèle Logique des Données)*. Nous avons utilisé la méthode MERISE pour les représentations séparées des données et des traitements.

Après avoir écrit le dictionnaire de données, nous avons créé notre base de données en utilisant l'*ORM Doctrine* de *Symfony*. Le *MPD (Modèle Physique des Données)* a été généré par *Adminer*, le gestionnaire *SGBD* (Système de Gestion de Base de Données).

- Développer les composants d'accès aux données

La connexion à la base de données est faite grâce aux informations de connexion qui sont stockées dans une variable d'environnement *DATABASE_URL*, placée dans le fichier *.env* à la racine du projet.

Les données sont sous la forme d'objets et de collections d'objets, ce qui fait que nous sommes dans la programmation orientée objet (POO).

La manipulation et la consultation des données se font grâce aux *Entity* (entité) et *Repository* (répertoire) de l'*ORM Doctrine*. L'*ORM* implémente les *patterns* objets « *Data Mapper* » et « *Unity of Work* » pour mapper (faire correspondre) un objet PHP à des éléments d'un système de persistance. Ils consistent, pour le premier, à synchroniser la donnée stockée en base avec les objets PHP, et pour le second, à gérer l'état des différents objets. Ils utilisent tous les deux *Entity Manager*, qui fait le lien entre les *Entity* (objets PHP) et la base de données.

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

L'application O'Beer est composée de nombreuses fonctionnalités. Celles-ci ne sont pas toutes accessibles à un visiteur.

Des rôles ont été créés afin de séparer les fonctionnalités pour un utilisateur non-connecté, un utilisateur connecté et un administrateur.

Un utilisateur connecté peut avoir une liste de favoris, un administrateur gère le *back-office* du site.

Un *Access Control* a été mis en place dans le fichier *security.yaml*, permettant ainsi de gérer plus facilement l'accès aux routes en fonction des rôles.

De plus, des *tokens* CSRF (*Cross-Site Request Forgery*) ont été mis en place sur les formulaires avec des requêtes HTTP *POST*.

RÉSUMÉ DU PROJET

Mésopotamie, il y a 6000 ans. Les Sumériens se nourrissaient principalement de pain très dur qu'ils faisaient tremper dans de l'eau pour le ramollir. Un jour, un Sumérien qui aurait voulu le consommer plus tard, mis son « pain à l'eau » de côté. C'est ainsi que la bière aurait été découverte, d'après les premiers écrits.

D'ailleurs, elle ne s'appelait pas « bière » mais « Sikaru », qui signifie « pain liquide ».

6000 ans plus tard, la bière plaît toujours à beaucoup de monde. Et notamment aux Français ! Un habitant de l'hexagone consomme, en moyenne, 33 litres par an. Et pour sa consommation, il a du choix ! Le marché de la bière, en baisse pendant 30 ans, est depuis 2014 en plein essor. Et ce, grâce aux microbrasseries. La nouvelle qualité de brassage a fait naître un nouveau mode de consommation, la *Zythologie*, qui est à la bière ce que l'*Oenologie* est au vin. Et pour cause, la production artisanale représente 6 à 8% du marché de la bière en France.

Or, depuis la crise et l'inflation (qui augmente considérablement le coût des matières premières), les microbrasseurs sont de plus en plus nombreux à mettre la clé sous la porte. En effet, ils vendent mais en très petite quantité et de manière assez peu régulière, n'ayant pas de partenariat qui leurs permettraient de passer du volume régulièrement.

C'est dans cette optique que nous avons voulu créer *O'Beer*. Un moyen pour les microbrasseurs d'être visibles et surtout connus par des clients potentiels, que ce soit des particuliers ou des professionnels. Notre site est à la fois orienté *BtoC* et *BtoB*.

Cette recherche constante de qualité permettra à terme à notre site de devenir une référence sur le recensement et le classement des brasseries et bières sur le marché régional, national et plus encore.

CAHIER DES CHARGES

Objectifs

L'objectif de *O'Beer* est de répertorier les différentes microbrasseries locales françaises en fonctions de leur localité (région). Les brasseries auront, par ce biais, une meilleure visibilité au près d'un public qui est majeur.

Lors de la navigation sur le site, un utilisateur pourra découvrir ce qu'est le *Craft Beer* (bière artisanale) et son histoire, les différents types et des anecdotes.

Pour les utilisateurs qui sont déjà connaisseurs, ils pourront découvrir de nouvelles

brasseries et aller y déguster leurs produits (et les noter sur le site dans une version ultérieure).

Les utilisateurs pourront, s'ils ont un compte, ajouter leurs brasseries préférées dans leurs favoris.

De plus, le site pourrait aussi par la suite mettre en relation les producteurs avec les distributeurs (grossistes ou revendeurs) qui seraient en recherche de nouveaux fournisseurs.

Conceptualisation de l'application

MVP (*Minimum Viable Product*)

Un *MVP* est une version rapide, testable et utilisable du projet. Il permet un lancement rapide pour obtenir un maximum de retours client dans un minimum d'effort.

Lors du premier sprint, nous avons défini en équipe le minimum requis pour notre application O'Beer. Notre *Produit Minimum Viable* se construit autour de la découverte, pour les utilisateurs majeurs, de l'ensemble du site. La gestion des favoris se fait uniquement pour un utilisateur connecté, il y a donc un formulaire d'inscription et un formulaire de connexion. La gestion des données en *back-office* se fait par un utilisateur ayant le rôle d'administrateur, lui donnant accès à toutes les fonctionnalités. Il peut voir, ajouter, modifier et supprimer une brasserie. Il peut également exécuter ces actions pour un utilisateur.

Liste des fonctionnalités

Du *MVP*, nous avons établi la liste des fonctionnalités du projet. Ces fonctionnalités concernent différentes parties de l'application.

Le site doit être *responsive*. La majorité des utilisateurs se sert de son *smartphone*, il faut donc que l'application puisse être consultée depuis un mobile.

Un formulaire de vérification d'âge légal (avec un message de prévention sur la consommation d'alcool) est mis en place à l'arrivée sur l'application pour s'assurer que l'utilisateur est majeur.

Le logo de l'application est un lien de redirection vers la page d'accueil, et ce sur tout le site.

Le menu (menu burger au format mobile) est accessible sur tout le site et contient les liens vers les différentes pages du site, hors *footer*.

Le *footer* (pied de page) est à chaque fin de page et contient les liens vers les « Mentions légales », « la Team » et « Nous contacter ».

Pages accessibles aux visiteurs (utilisateurs non-connectés) :

- | | | |
|---------------------|--------------------|----------------------------|
| - Accueil | - Mentions légales | - Formulaire de connexion |
| - Un Peu d'Histoire | - La Team | - Formulaire d'inscription |
| - Types de bières | - Nous contacter | |

- Liste de toutes les brasseries
- Liste des brasseries d'une région

- Page d'une brasserie

Fonctionnalités pour les visiteurs :

- Création de compte
- Message à la création de compte
- Connexion à leur compte

- Régions de la carte cliquables avec redirection vers la liste des brasseries de la région cliquée

Fonctionnalités pour les utilisateurs connectés :

- Liste de favoris
- Ajout d'un favori
- Suppression d'un favori
- Suppression de la liste des favoris

- Message à l'ajout d'une brasserie aux favoris
- Message à la suppression d'un favori
- Message à la suppression de la liste de favoris

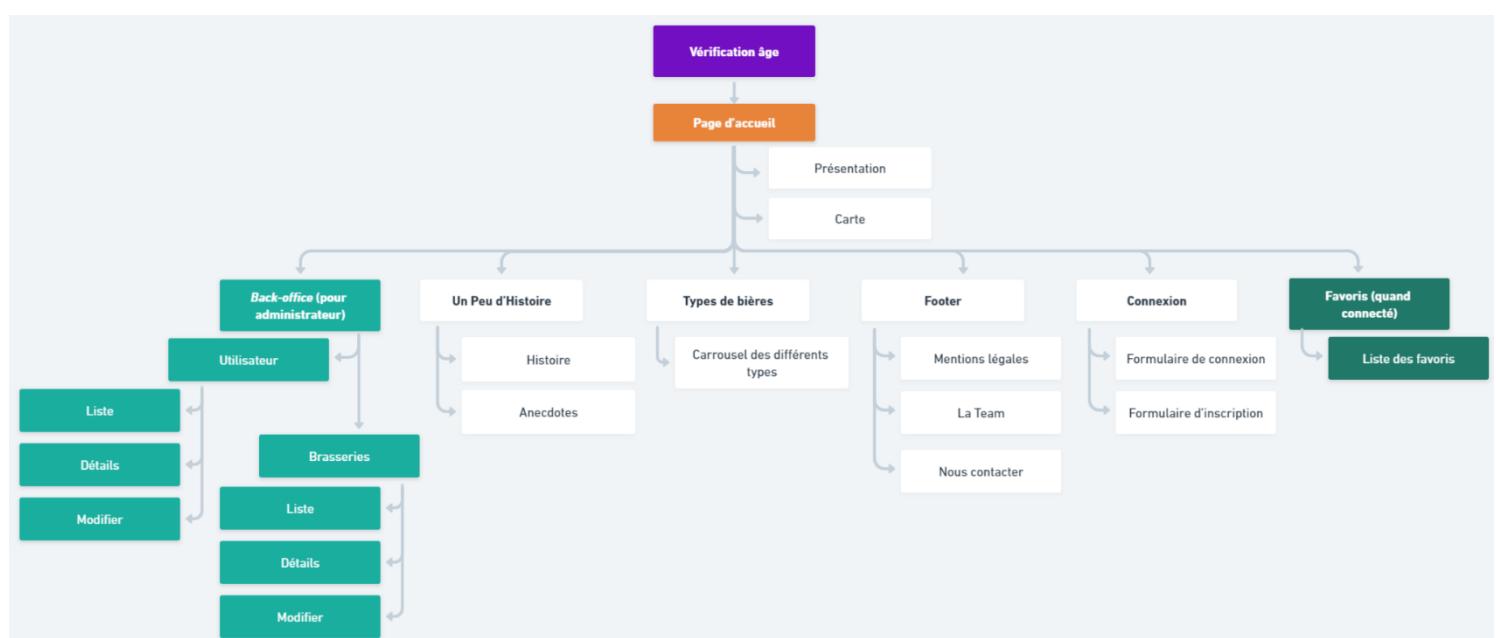
Fonctionnalités pour les administrateurs :

- Accueil du *back-office*
- Liste des utilisateurs
- Formulaire de modification d'un utilisateur
- Vue d'un utilisateur
- Formulaire d'ajout d'un utilisateur
- Suppression d'un utilisateur

- Liste des brasseries
- Formulaire de modification d'une brasserie
- Vue d'une brasserie
- Formulaire d'ajout d'une brasserie
- Suppression d'une brasserie

Mise en place de l'application

Arborescence



Wireframes

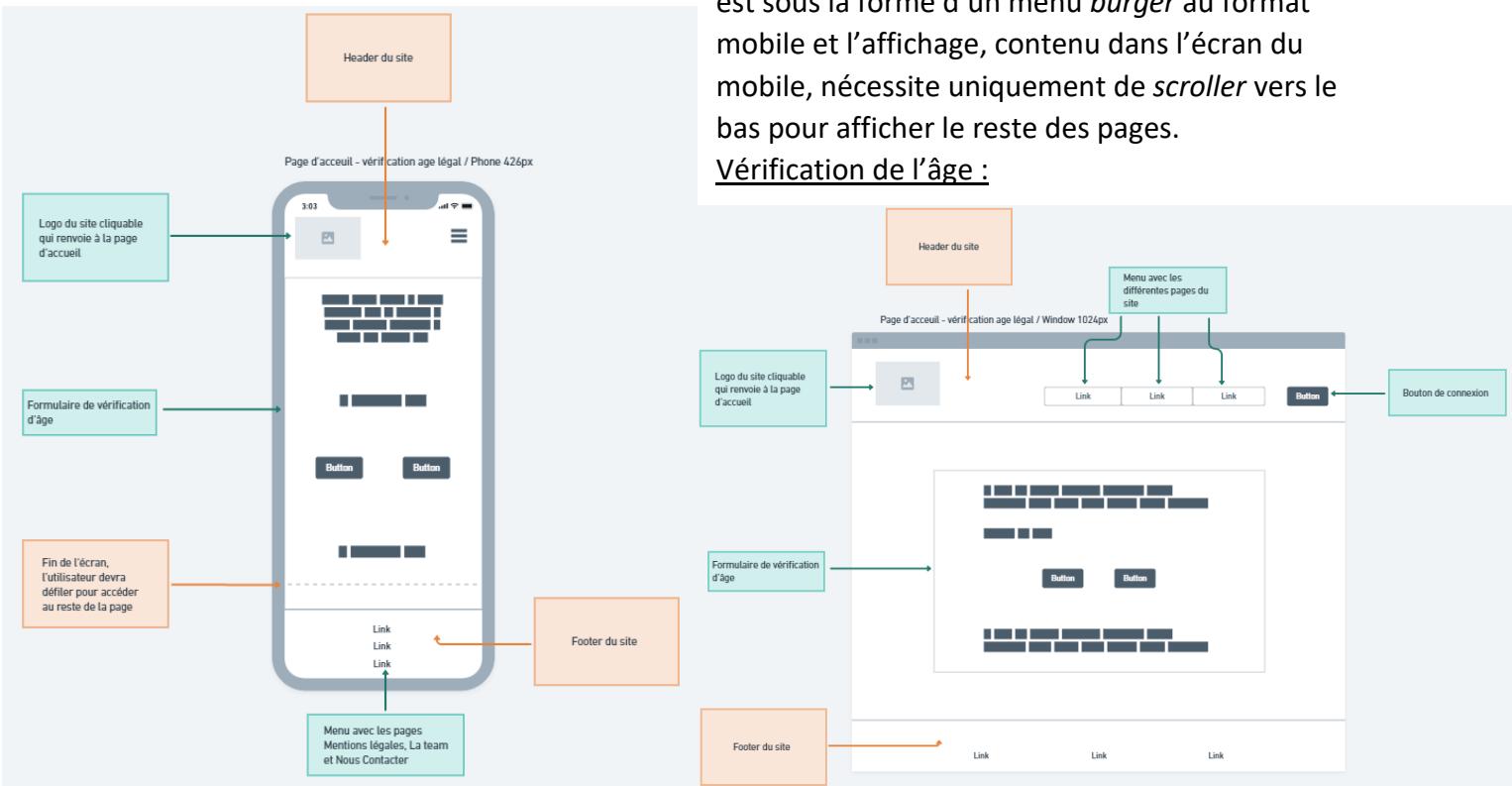
Leur nombre étant élevé, l'intégralité des *Wireframes* est disponible à cette adresse :

xx

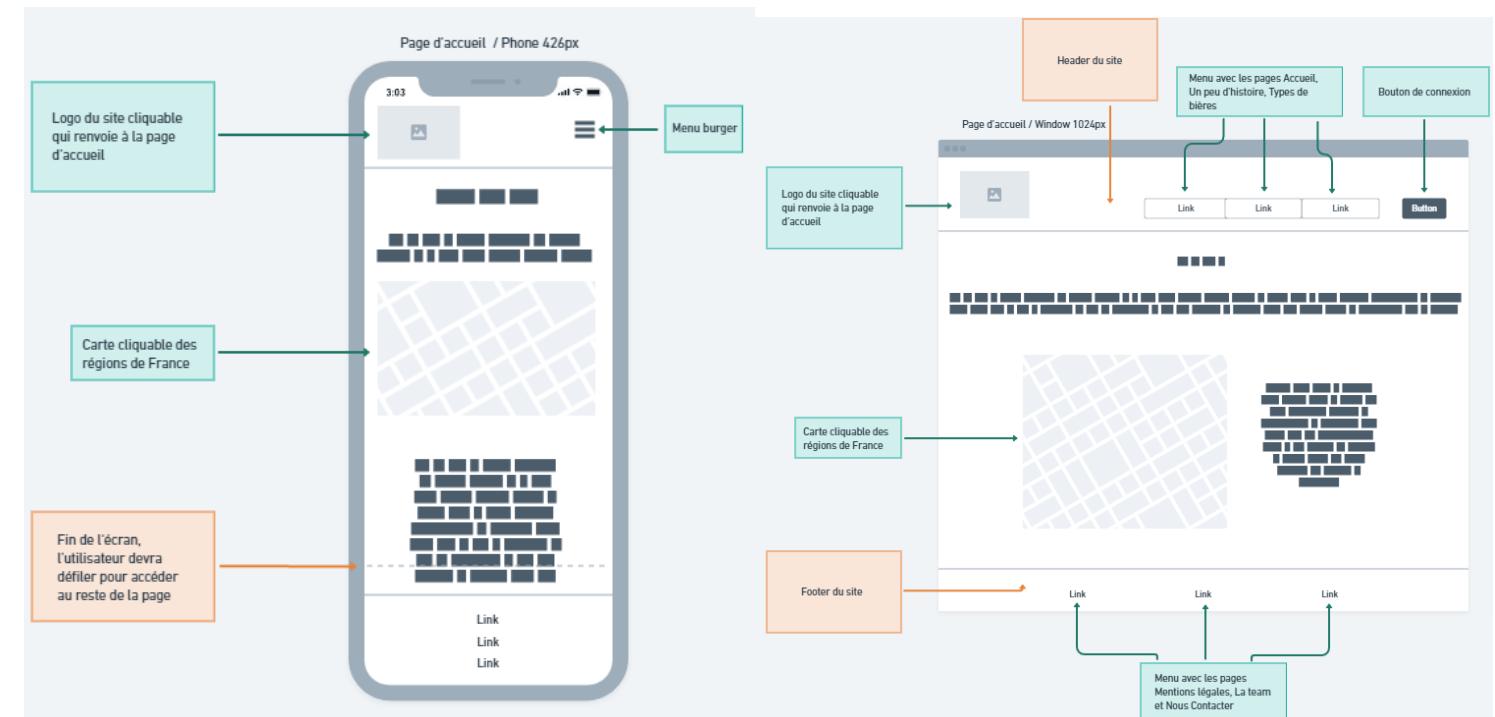
Comme évoqué dans les fonctionnalités, nous avons voulu un site *responsive*. Pour ce faire, nous avons abordé la conception des *Wireframes* selon une approche *mobile-first*.

Le menu est sous la forme d'un menu *burger* au format mobile et l'affichage, contenu dans l'écran du mobile, nécessite uniquement de *scroller* vers le bas pour afficher le reste des pages.

Vérification de l'âge :



Page d'accueil :



Formulaire d'inscription :

The diagram shows two versions of a sign-up form: a mobile version (Phone 426px) on the left and a desktop version (Window 1024px) on the right.

Mobile Version (Phone 426px):

- Champ du formulaire:** A series of five input fields.
- Bouton pour se connecter:** A button labeled "Button".
- Lien vers le formulaire de connexion:** A link labeled "Link".
- Lien vers la page d'accueil:** A link labeled "Link".
- Button:** A large dark blue button at the bottom.
- Links at the bottom:** Three links labeled "Link", "Link", and "Link".

Desktop Version (Window 1024px):

- Bouton pour créer son compte:** A button labeled "Button".
- Lien vers le formulaire de connexion:** A link labeled "Link".
- Lien vers la page d'accueil:** A link labeled "Link".
- Input Fields:** Fields for "My firstname", "My name", "Ma date de naissance", "mon@email.com", and "MonMotDePasse".
- Links at the bottom:** Three links labeled "Link", "Link", and "Link".

Formulaire d'ajout d'une brasserie dans le back-office :

The diagram shows two versions of a brewery addition form: a mobile version (Phone 426px) on the left and a desktop version (Window 1024px) on the right.

Mobile Version (Phone 426px):

- Bouton pour enregister la nouvelle brasserie:** A button labeled "Button".
- Bouton vers la page de gestion des brasseries:** A button labeled "Button".
- Input Fields:** Fields for "Nom brasserie", "Adresse", "Code postal", "Ville", "Votre région", and "Site web".
- Buttons at the bottom:** Two buttons labeled "Button" and "Button".
- Links at the bottom:** Three links labeled "Link", "Link", and "Link".

Desktop Version (Window 1024px):

- Bouton pour enregister la nouvelle brasserie:** A button labeled "Button".
- Lien vers la page de gestion des brasseries:** A link labeled "Link".
- Input Fields:** Fields for "Nom brasserie", "Adresse", "Code postal", "Ville", "Votre région", and "Site web".
- Links at the bottom:** Three links labeled "Link", "Link", and "Link".

Charte graphique et logo / Accessibilité

Dans le but d'avoir la meilleure accessibilité possible, nous avons choisi d'utiliser la police « Luciole ». Elle a été créée par le Centre Technique Régionale pour la Déficience Visuelle et par le studio *typographie.fr* dans le but d'aider les personnes malvoyantes.

Concernant le choix des couleurs, nous voulions un site au contraste élevé. C'est pourquoi nous avons choisi ces couleurs :

#FBB03B : Surbrillance liens menu, liens pied de page et carte

#999999 : Police pied de page

#3B3B3B : Fond carte

#393939 : Police site

#F0F0F0 : Fond site

#B2BABC : Police menu

#000 : Fond menu



(le fond noir est présent uniquement pour faire ressortir le blanc)

Ce logo est inspiré de différents modèles disponibles sur internet.

Après une discussion avec l'équipe, je l'ai réalisé à l'aide d'*Adobe Illustrator*.

Utilisateurs et *User Stories*

Public visé

Tous types de personnes majeures :

- Les connaisseurs souhaitant aiguiser leur savoir (découverte, notation),
- Les brasseurs qui souhaitent étendre leur clientèle,
- Potentiels futurs consommateurs,
- Grossistes/Revendeurs en recherche de nouveaux fournisseurs

Rôles

Il n'y a que deux rôles pour l'application. Le rôle *user* (utilisateur) et le rôle *admin* (administrateur). Un utilisateur a automatiquement le rôle *user* lorsqu'il crée son compte. Seul un administrateur peut attribuer le rôle d'*admin* à un utilisateur.

User Stories

En tant que	Je souhaite	Afin de
Visiteur	Valider ma majorité	Accéder à la page d'accueil
Visiteur	Parcourir le site	Accéder aux pages sans fonctionnalités
Visiteur	Ouvrir le menu connexion	Accéder au formulaire d'inscription
Visiteur	Ouvrir le menu connexion	Accéder au formulaire de connexion
Visiteur	Utiliser la carte	Découvrir les brasseries en fonction de la région sélectionnée
Visiteur	Afficher toutes les brasseries	Voir toutes les brasseries sur le site
Utilisateur connecté	Ajouter une brasserie à mes favoris	Avoir une liste de mes brasseries favorites
Utilisateur connecté	Supprimer un favori	Retirer une brasserie de mes brasseries favorites
Utilisateur connecté	Supprimer ma liste de favoris	Ne plus avoir de favoris
Utilisateur connecté	Accéder ma liste de favoris	Gérer ma liste de favoris
Utilisateur connecté	Ouvrir le menu connexion	Me déconnecter
Administrateur	Me connecter	Accéder au <i>back-office</i>
Administrateur	Accéder au <i>back-office</i>	Ajouter une brasserie
Administrateur	Accéder au <i>back-office</i>	Voir le détail d'une brasserie
Administrateur	Accéder au <i>back-office</i>	Modifier une brasserie
Administrateur	Accéder au <i>back-office</i>	Supprimer une brasserie
Administrateur	Accéder au <i>back-office</i>	Afficher les brasseries
Administrateur	Accéder au <i>back-office</i>	Ajouter un utilisateur
Administrateur	Accéder au <i>back-office</i>	Voir le détail d'un utilisateur
Administrateur	Accéder au <i>back-office</i>	Attribuer le rôle d'un utilisateur
Administrateur	Accéder au <i>back-office</i>	Modifier un utilisateur
Administrateur	Accéder au <i>back-office</i>	Supprimer un utilisateur
Administrateur	Accéder au <i>back-office</i>	Afficher les utilisateurs

Évolutions éventuelles

La durée de développement du site et les difficultés rencontrées ne nous ont pas permis de mettre en place l'ajout des bières, bien que la table soit créée dans la base de données. C'est ce que nous aimerais mettre en place lors de la V2. Entre autre, nous voulons également mettre en place la fonctionnalité d'envoi du formulaire de contact et que celui-ci arrive sur une adresse email dédiée au site internet. Il faudrait également rédiger les messages d'erreurs des formulaires en français et inclure des expressions régulières pour les mots de passes. Enfin, à titre personnel, j'aimerais revoir la fonctionnalité de favoris et l'intégration.

En tant que	Je souhaite	Afin de
Visiteur	Soumettre le formulaire de contact	Contacter l'équipe du site
Visiteur	Afficher les bières des brasseries	Connaître les produits proposés
Visiteur	Voir la bière du mois	Découvrir une bière mis à l'honneur
Visiteur	Trier les bières	Afficher les bières en fonction de leur type
Administrateur	Ajouter une bière	Compléter les informations d'une brasserie

SPÉCIFICATIONS TECHNIQUES

Versioning

La gestion de version permet de conserver l'historique du code source et de travailler plus facilement en équipe. Il était très important pour nous de garder les versions précédentes du code, notamment en cas de gros problème avec une fonctionnalité suite à des modifications dans le code. Pour cette gestion, nous avons décidé d'utiliser le logiciel de version décentralisé *Git* et le service web d'hébergement *GitHub*.

Afin d'optimiser notre manière de travailler et d'avoir le meilleur workflow possible, nous avons opté pour l'utilisation du modèle de Vincent Driessen : *Git-flow*.

Le *repository* de notre application comporte de multiples branches.

Nous avons notre branche de production *main* (principale en français). C'est la branche qui est visible par les utilisateurs lorsque le site est mis en production (mis en ligne).

À celle-ci s'ajoute la branche de pré-production *develop_bis*. Elle recueille les fonctionnalités (*features*) développées et nous permettait de vérifier que celles-ci ne « cassaient » pas le site lorsque l'on ajoutait des fonctionnalités.

Enfin, nous avons plusieurs branches « *feature* ». Chacune d'elles est dédiée à une fonctionnalité. Une fois terminée, la branche était « *merged* » (fusionnée) avec *develop_bis*.

Pour les fusions, nous avons utilisé la commande *git merge*.

En revanche, la fusion de *develop_bis* et de *main* s'est faite à l'aide d'une *Pull Request*. Après avoir validé la *PR*, la fusion entre *develop_bis* et *main* s'est faite sans aucun problème.

Au début du projet, nous avions des difficultés avec *Git* et *GitHub*. En effet, nous n'avions pas encore mis en place le modèle de fonctionnement *Git-flow* et cela nous a coûté du temps de développement. Des branches ont été créées en partant d'une première créée, puis d'autres depuis des branches plus avancées. Il nous a fallu remettre tout à plat et reprendre notre manière d'appréhender le *versioning*. Étant la personne la plus à l'aise, j'ai naturellement repris le rôle de *Git Master* attribué auparavant à un autre membre de l'équipe. Cette décision a été prise à l'unanimité.

J'ai aussi pris la décision de créer *develop_bis*. Les branches ayant été mal gérées au départ, j'ai eu peur que le *merge* entre elles ne résulte en une application non-fonctionnelle. Mais je sais que nous aurions dû n'avoir qu'une seule branche *develop*.

Concernant les *commits*, après discussion, la majorité de l'équipe préférait les écrire en français. Cependant, il aurait été préférable de les écrire en anglais. Ne voulant pas briser la cohérence, je les ai donc écrits en français.

Extraits de commits que j'ai effectué :

-	Commits on Aug 24, 2023
	Fonction ajout favoris et modification header page accueil MalMarie committed on Aug 24
	Dynamisation des brasseries de la page d'accueil MalMarie committed on Aug 24
	Création du BrewerieController et début dynamisation de favoris/index... ...html/twig MalMarie committed on Aug 24
-	Commits on Sep 2, 2023
	Gestion de l'unicité de l'email MalMarie committed on Sep 2
	Flash message pour la création d'un nouveau compte utilisateur MalMarie committed on Sep 2
	Suppression d'une brasserie de la liste des favoris MalMarie committed on Sep 2
	Flash message pour brasserie déjà présente dans les favoris MalMarie committed on Sep 2
	Ajout du template pour la popup, des assets et de la route dans le co... ...ntroller - modifications dans TypeController pour affichage sur le site MalMarie committed on Sep 2

Choix technologiques

Les choix techniques ont été faits lors du premier sprint du développement de l'application. Nous avons réfléchi à différentes possibilités et avons pesé « le pour et le contre » avant d'arrêter notre réflexion.

Front-End

Le choix technologique concernant le *Front-End* (partie dite client, avec le code visible par l'utilisateur) a été fait d'un commun accord.

Nous avons bien évidemment utilisé le langage de structuration de page *HTML 5 (HyperText Markup Language)* et les Feuilles de Style en Cascade *CSS 3 (Cascading Style Sheets)*.

Nous avons opté, pour la dynamisation de l'application, d'utiliser le langage de programmation de scripts *JavaScript*.

Enfin, nous avons aussi utilisé la librairie *CSS Bootstrap* pour nous aider à mettre le site plus rapidement en forme.

Lors du premier sprint, il a été envisagé d'utiliser le pré-processeur CSS *SASS*. Cependant, après en avoir longuement discuté, nous avons pris la décision de ne pas l'utiliser, afin que tout le monde puisse facilement et rapidement intervenir.

Nous avons ainsi pu mettre en place les fonctionnalités suivantes :

- le questionnaire de vérification d'âge à l'arrivée sur le site (cf annexes)
- la surbrillance du nom des régions au passage de la souris sur la carte (cf annexes)
- rendre le site responsive

Back-end

Deux membres de l'équipe ont effectué la spécialisation *Symfony* de la formation, nous avons donc décidé de faire de *O'Beer* un projet *Symfony 5.4*.

Étant donné que toute l'équipe avait travaillé depuis le début de la formation avec le langage de programmation *PHP*, le développement et l'intervention en cas de bug était facilité.

Nous avons installé un certain nombre de *bundles* (paquets) après l'installation du squelette de *Symfony*. Ils sont des ensembles de fichiers qui permettent de réaliser une ou plusieurs fonctionnalités. Ceux-ci ont l'avantage d'être réutilisables.

Concernant notre base de données, nous avons porté notre choix sur le système de gestion de bases de données relationnelles *MySQL*, en utilisant l'outil graphique de gestion de bases de données *Adminer*. La base de données et ses tables ont toutefois été créées grâce à *l'ORM Doctrine*.

Nous avons ainsi pu mettre en place les fonctionnalités suivantes:

- la création d'un compte utilisateur

- la connexion d'un utilisateur
- l'accès à différentes parties du site en fonction du rôle attribué
- l'ajout d'une brasserie et d'un utilisateur
- la modification d'une brasserie et d'un utilisateur
- la suppression d'une brasserie et d'un utilisateur
- l'ajout et la suppression d'un favori
- la suppression de la liste de favoris
- l'affichage de la liste des brasseries en fonction d'une région donnée

La fonctionnalité de favoris a été développée avec un service. La liste de favoris est stockée localement sur le support de l'utilisateur. À titre personnel, j'aurais préféré que les favoris soient stockés dans la base de données. Afin que l'utilisateur ait toujours sa liste de favoris identique, et non une différente selon son support. Cela offrirait une meilleure expérience utilisateur.

Accessibilité

Navigateurs compatibles

Nous avons développé l'application en veillant à ce que la compatibilité avec les navigateurs *Chrome* et *Firefox* soit assurée. L'environnement de développement fourni par l'école O'Clock ne disposait pas des navigateurs *Edge* et *Safari*. Nous n'avons donc pas pu assurer la compatibilité avec ces navigateurs lors du développement de l'application.

Après avoir déployé notre application, nous avons testé le site sur des navigateurs propre à chacun d'entre nous, et avons constaté que le site est aussi compatible avec *Bing* et *Opera*.

En revanche, nous n'avons aucunement prévu de rendre le site compatible pour *Internet Explorer*, étant donné qu'il a été arrêté par *Microsoft* et est remplacé depuis par *Microsoft Edge*.

Types d'appareils

Les types d'appareils compatibles avec notre application sont les *smartphones*, les tablettes et les ordinateurs. En effet, nous avons développé une application responsive et nous nous sommes assurés que l'affichage soit adapté en fonction de certains types de résolution.

Par manque de temps de développement, nous avons choisi de créer pour l'instant qu'un affichage mobile (base de 426px de large) et un affichage *desktop* (base de 1024px de large). Une tablette aura donc un affichage de type mobile en mode portrait et de type *desktop* en mode paysage.

Architecture du projet

Symfony est un *framework* MVC (Modèle-Vue-Contrôleur). Comme son nom l'indique, ce type d'architecture comporte trois types de modules :

- le modèle :

Il contient les données ainsi que la logique en rapport avec elles. C'est lui qui gère les requêtes du CRUD (*Create, Read, Update, Delete* – Créer, Lire, Éditer, Supprimer).

- la vue :

C'est la partie visible de l'interface graphique. Elle gère la disposition et l'affichage des pages grâce aux balises *HTML* qu'elle contient.

- le contrôleur :

Il contient la logique des actions effectuées par l'utilisateur. Il achemine les commandes des parties modèle et vue.

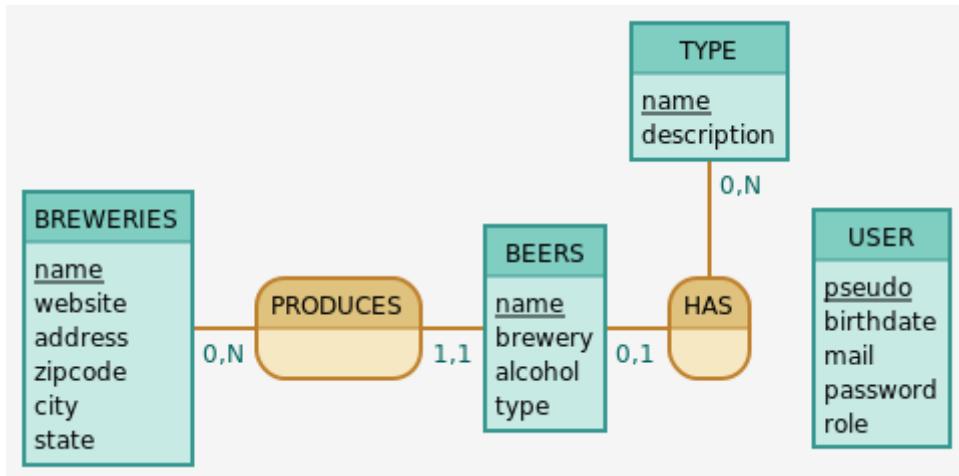
Lorsque l'utilisateur va vouloir afficher une donnée, il va faire une requête HTTP qui va demander une route. Le point d'entrée unique *index.php*, qui contient toutes les routes du site, va appeler le bon contrôleur et la bonne méthode. Le contrôleur va à son tour appeler une méthode d'un modèle, afin de récupérer dans la base de données, les données que l'utilisateur veut afficher. Le modèle retourne au contrôleur ses résultats sous la forme d'un objet PHP. Ce dernier passe les données à la vue qui affiche la page structurée incluant les données demandées au client.

Dans *Symfony*, nous sommes dans un *pattern Data Mapper*. Les *Entity* contiennent les propriétés des entités ainsi que les *getters* et *setters*. Les *Repository* contiennent eux les requêtes du CRUD pour communiquer avec la base de données. C'est l'*Entity Manager* qui manipule les objets. Ceux-ci sont indépendants de toute méthode en rapport avec la base de données.

La Base de Données

Par manque de temps, nous n'avons pas encore utilisé la table des bières, bien qu'elle existe déjà dans la base de données.

MCD



Le Modèle Conceptuel de Données a été pensé et créé lors du premier sprint grâce au site internet *Mocodo*. Nous avons réfléchi ensemble au nombre d'entités qu'il faudrait avoir pour que notre site soit fonctionnel.

Ce MCD montre une entité *User* esseulée. Elle est cependant utilisée par l'application pour les connexions et pour la gestion du *back-office*.

MLD

Le Modèle Logique de Données a automatiquement été généré par le site *Mocodo*. Le MLD indique les clés étrangères détenues dans les différentes tables.

Ici, on peut voir que l'entité **BEERS** détient deux clés étrangères : le nom (qui sera l'*id*) des types et le nom (qui sera l'*id*) des brasseries. Cela est dû au fait qu'elle détient le « 1 » dans ses relations avec **BREWERIES** et **TYPE**.

BEERS (name, brewery, alcohol, type, #nameType, #nameBreweries)

BREWERIES (name, website, address, zipcode, city, state)

TYPE (name, description)

USER (pseudo, birthdate, mail, password, role)

Dictionnaire de données

Pour aider à la création de la base de données, nous avons rédigé ensemble le dictionnaire de données. Celui-ci est une collection de métadonnées ou de données de référence nécessaire à la conception d'une base de données relationnelle. Nous y avons indiqué le type de chaque champ d'une table de la base de données.

ENTITY	Nom du champ	Type de données	Longeur max
Breweries	id	int	10

Breweries	name	varchar	255
Breweries	address	varchar	255
Breweries	zipcode	int	10
Breweries	city	varchar	255
Breweries	state	varchar	255
Breweries	website	varchar	255
Beer	id	int	10
Beer	name	varchar	255
Beer	alcohol	decimal	10
Beer	type (FK)	varchar	255
Beer	brewerie (FK)	int	100
Type	id	int	10
Type	name	varchar	255
Type	description	varchar	255
User	id	int	10
User	pseudo	varchar	255
User	birthdate	datetime	
User	mail	email	100
User	password	string	100
User	role	longtext (DC2Type:json)	

MPD

Une fois la base de données créée avec *Doctrine* grâce à la commande : « `php bin/console doctrine:database:create` », nous avons créé les différentes entités à l'aide du *maker* « `make :entity` » de *Symfony*. Après chaque nouvelle entité, nous effectuons la migration par les commandes « `php bin/console make:migration` » et « `php bin/console doctrine:migrations:migrate` » afin de créer la table correspondante dans la base de données.



Notre outil graphique de gestion de bases de données *Adminer* a automatiquement généré le Modèle Physique de Données.

Pour insérer les données dans la base, nous avons rédigé les requêtes SQL afin de ne pas faire l'opération à la main et pour avoir une sauvegarde.

Extrait de la requête d'ajout de brasseries :

```
/** AJOUT DE TOUTES LES BRASSERIES*/
INSERT INTO `brewerie` (`name`, `address`, `zipcode`, `city`, `state`, `website`) VALUES
('La Vardaf (Brasserie raisonnée)', '245 Chem. des Crettets', 74470, 'Bellevaux', 'Auvergne-Rhône-Alpes', 'https://lavaraf.fr/bieres-bi'),
('Bière bio des Monts d'Or', "18 Chemin des Carrières", 69250, "Curis-au-Mont-d'Or", 'Auvergne-Rhône-Alpes', 'https://www.bieresbio.fr/'),
('Brasserie Volcane', 'ZA de Largelier, 7 Rue de la Côte', 43100, 'Cohade', 'Auvergne-Rhône-Alpes', 'http://www.biere-volcane.fr/' ),
('Brasserie de Vezelay', 'rue du Gravier', 43100, 'Cohade', 'Bourgogne-Franche-Comté', 'https://brasserievezelay.fr/' ),
('Maddam (Brasserie de Chablis)', '4 rue des Vendanges', 89800, 'Chablis', 'Bourgogne-Franche-Comté', 'https://www.maddam-chablis.com/' ),
('Brasserie la bonne boulle', '14 Rue du Magasin', 25300, 'Pontarlier', 'Bourgogne-Franche-Comté', 'https://www.brasseriebonnebouille.c'),
('Merlin Hops Brewing more', '40 impasse Ru Land', 29760, 'Penmarch', 'Bretagne', 'https://www.merlin.beer/' ),
('Brasserie des Abers', '2 avenue de Portsall', 29830, 'Ploudalmézeau', 'Bretagne', 'https://www.facebook.com/brasserieedesabers/' ),
('L'Amer d'Iroise', '2 impasse du Steir', 29770, 'Plogoff', 'Bretagne', 'https://amer-iroise.beer/' ),
('Art & Brew', '3 Rue Durfort de Duras', 41600, 'Lamotte-Beuvron', 'Centre-Val-de-Loire', 'https://artandbrew.fr/' ),
('Brasserie Ouche Nanon', '2 Place du Marché', 18350, 'Ourouer-les-Bourdelins', 'Centre-Val-de-Loire', 'https://www.ouche-nanon.fr/' ),
('Micro Brasserie Nogentaise', '24 Rue Sainte-Anne', 28400, 'Nogent-le-Rotrou', 'Centre-Val-de-Loire', 'https://brasserie-nogentaise.p'),
('Brasserie Galibot', '5 Rue Jules Verne', 57600, 'Forbach', 'Grand-Est', 'https://www.brasserie-galibot.com/accueil' ),
('Reims Royal Beer', '15 Cours Anatole France', 51100, 'Reims', 'Grand-Est', 'https://www.reimsroyalbeer.fr/' ),
('Ardwen', '20 Av. Roger Ponsart', 08430, 'Launois-sur-Vence', 'Grand-Est', 'https://www.ardwen.fr/' ),
('Brasserie Quentovic', '27, rue du collège', 62990, 'Beaurainville', 'Hauts-de-France', 'http://quentovic.com/' ),
```

« *INSERT INTO* » sert à insérer, comme son nom l'indique, les données dans l'ordre des champs de la table *brewerie*. « *VALUES* » indique que les valeurs à ajouter sont après.

Afin d'établir la connexion entre l'application et la base de données, nous avons mis en place le fichier `.env` qui comporte la variable d'environnement `DATABASE_URL`. Pour des raisons de sécurité, bien que nous l'ayons fait un peu tard, nous avons également ajouté le fichier `.env` au fichier `.gitignore`, afin qu'il ne soit pas publié.

Routes

Une fois les fonctionnalités définies et les *user stories* rédigées, nous avons spécifié les routes pour chacune d'elles.

ROUTES PUBLIQUES				
Endpoint	Méthode	Description	Retour	Version
/	GET	Questionnaire de vérification d'âge puis affichage de la page d'accueil	200	1
/mentions-legales	GET	Affichage de la page des mentions légales	200	1
/equipe	GET	Affichage des photos de l'équipe	200	1
/contacts	GET	Affichage du formulaire de contact	200	1
/histoire	GET	Affichage de la page "Un peu d'histoire" et des anecdotes	200	1
/type-biere	GET	Affichage de la page des différents types de bières et leurs explications	200	1
/connexion	GET	Affichage du formulaire de connexion	200	1
/connexion	POST	Soumission du formulaire de connexion	201 + 303 (redirection)	1
/utilisateur/nouveau	GET	Affichage du formulaire de création de compte	200	1
/utilisateur/nouveau	POST	Soumission du formulaire de création de compte	201 + 303 (redirection)	1
/favoris/liste	GET	Liste des brasseries favorites	200	1

ROUTES BACK OFFICE				
Endpoint	Méthode	Description	Retour	Version
/back	GET	Affichage de la page d'accueil du <i>back-office</i>	200	1
/back/brewerie/	GET	Affichage de la liste des brasseries	200	1
/back/brewerie/new	GET	Affichage du formulaire d'ajout de brasserie	200	1

/back/brewerie/new	POST	Envoi du formulaire d'ajout de brasserie	201 + 303 (redirection)	1
back/brewerie/{id}/	GET	Affichage de la fiche d'une brasserie	200	1
back/brewerie/{id}/edit	GET	Affichage du formulaire de modification d'une brasserie	200	1
back/brewerie/{id}/edit	PUT	Soumission du formulaire de modification d'une brasserie	201 + 303 (redirection)	1
/back;brasseurs/{id}/delete	DELETE	Suppression d'une brasserie	204 + 303 (redirection)	1
/back/user/	GET	Affichage de la liste des utilisateurs	200	1
/back/user/new	GET	Affichage du formulaire d'ajout d'un utilisateur	200	1
/back/user/new	POST	Envoi du formulaire d'ajout d'un utilisateur	201 + 303 (redirection)	1
back/user/{id}	GET	Affichage de la fiche d'un utilisateur	200	1
back/user/{id}/edit	GET	Affichage du formulaire de modification d'un utilisateur	200	1
back/user/{id}/edit	PUT	Soumission du formulaire de modification d'un utilisateur	201 + 303 (redirection)	1
/back/user/{id}/delete	DELETE	Suppression d'un utilisateur	204 + 303 (redirection)	1

Déploiement

Lors de notre dernier *sprint*, nous avons déployé l'application sur le serveur fourni par l'école O'Clock. Le mien étant à cette adresse : <http://malmarie-server.eddi.cloud/projet-6-o-beer-back/public/> .

Pour ce faire, chaque membre de l'équipe a donc cloné le *repository* de l'application sur son serveur.

Afin d'établir la connexion avec la base de données, il a fallu modifier la variable d'environnement *DATABASE_URL* du fichier *.env*.

Il a fallu après lancer la commande de Doctrine « `php bin/console doctrine:database:create` » afin de créer la base de données.

Puis effectuer les migrations à l'aide de « `php bin/console doctrine:migrations:migrate` ».

Et enfin, il ne restait qu'à effectuer les requêtes SQL afin d'insérer les données dans la base.

Sécurité de l'application

La sécurité d'une application web n'a jamais été aussi importante qu'aujourd'hui. En effet, la moindre faille de sécurité permet à des individus malhonnêtes de récupérer des informations sur les utilisateurs. La CNIL indique même que « tout site web doit garantir son identité et la confidentialité des informations transmises ». La communauté en ligne *OWASP* (*Open Worldwide Application Securiy Project*) a d'ailleurs publié leur « TOP 10 des risques de sécurité d'application web » datant de 2021.

Pour sécuriser au maximum notre application dans le temps qui nous était imparti, nous avons mis en place différentes sécurités, en ayant tout d'abord installé le *SecurityBundle* de *Symfony*.

Formulaires et token CSRF

Nous sommes partis d'une règle simple : *Never Trust User Input* (ne jamais faire confiance aux données utilisateur).

Pour cela, nous avons utilisé le composant *Validator* de *Symfony*, les contraintes et les *Fields Type* afin de sécuriser les formulaires.

Validator permet de valider et nettoyer les données fournies par l'utilisateur au moment de la soumission tandis que les *Fields Type* imposent un type pour chaque champ du formulaire. Enfin les contraintes servent, comme leur nom l'indique, à contraindre les données à entrer dans les champs.

Je vais prendre l'exemple des brasseries.

Le formulaire impose de choisir la région par un menu déroulant. Cela est possible grâce au type *ChoiceType* auquel on a ajouté un label et défini le nom des régions disponibles ainsi que leur valeur.

```
>>>add('state', ChoiceType::class, [
    'label' => 'Choisissez votre région',
    'choices' => [
        'Votre choix...' => '',
        'Auvergne-Rhône-Alpes' => 'Auvergne-Rhône-Alpes',
        'Bourgogne-Franche-Comté' => 'Bourgogne-Franche-Comté',
        'Bretagne' => 'Bretagne',
        'Centre-Val-de-Loire' => 'Centre-Val-de-Loire',
        'Grand-Est' => 'Grand-Est',
        'Hauts-de-France' => 'Hauts-de-France',
        'Ile-de-France' => 'Ile-de-France',
        'Normandie' => 'Normandie',
        'Nouvelle-Aquitaine' => 'Nouvelle-Aquitaine',
        'Occitanie' => 'Occitanie',
        'Pays-de-la-Loire' => 'Pays-de-la-Loire',
        "Provence-Alpes-Côte-d'Azur" => "Provence-Alpes-Côte-d'Azur",
    ],
])
```

De même pour le renseignement du site internet,

```
->add('website', UrlType::class, [
    'attr' => [
        'placeholder' => 'Une URL en http:// ou
                           https://'],
        'label' => 'Site web de la brasserie',
        'default_protocol' => 'https',
    ]);
]);
```

On indique que l'on veut que la donnée fournie par l'utilisateur soit de type *Url*.

Une fois le formulaire rempli, lorsque l'utilisateur va le soumettre, la méthode *isValid()* va valider et nettoyer les données avant qu'elles ne soient envoyées dans la base de données.

Nous avons aussi voulu prémunir l'application d'attaques de type *CSRF (Cross-Site Request Forgery)*. Celles-ci sont des requêtes malveillantes dissimulées qui font faire une action à un utilisateur disposant des droits nécessaires, et cela sans qu'il en ait connaissance. Pour éviter cela, on utilise des *Token* (jetons) qui sont des valeurs uniques générées aléatoirement côté serveur et envoyées au client. Si le jeton du formulaire du client ne correspond pas à celui du serveur, alors l'action est stoppée.

Symfony, lors de la génération d'un formulaire, va automatiquement ajouter un *token* dans un champ *input* de type *hidden* à la fin de celui-ci, grâce à « {{ form_end(form) }} » placé à la fin du template *twig*. Ce qui est le cas de notre formulaire d'inscription :

```
 {{ form_start(form, {'attr': {'novalidate': 'novalidate'}}) }}
    {{ form_widget(form) }}

    <div class="d-grid gap-2 d-md-flex justify-content-md-center">
        <button class="btn btn-primary">{{ button_label|default('Créer') }}</button>
    </div>
{{ form_end(form) }}

<h1>Créer un compte</h1>
▼<form name="registration" method="post" novalidate="novalidate">
    ▼<div id="registration"> == $0
        ▶<div class="mb-3">...</div>
        ▶<fieldset class="mb-3">...</fieldset>
        ▶<div class="mb-3">...</div>
        ▶<div class="mb-3">...</div>
        ▶<div class="mb-3">...</div>
        <input type="hidden" id="registration_token" name="registration[_token]" value="f1dd2b52559eb91d824e.c5XyjePgmh6JT4jxfypNvswk8fYYcv0RTCHfJmd
dVoQ.AvyquqSM7XX4feCnBUY00_plw68gR7AhOBPneSglbskYrJHv2tSiVOIB-w">
    ▼<div class="d-grid gap-2 d-md-flex justify-content-md-center"> grid
        <button class="btn btn-primary">Créer</button>
    </div>
</form>
```

Dans le cas du formulaire de connexion, nous l'avons ajouté nous même dans le formulaire :

LoginFormAthenticator.php :

```
public function authenticate(Request $request): Passport
{
    $email = $request->request->get('email', '');
    $request->getSession() ->set(Security::LAST_USERNAME, $email);

    // Passport contains all informations about user authenticating information
    // (email / password)
    return new Passport(
        new UserBadge($email),
        new PasswordCredentials($request->request->get('password', '')),
        [
            // automatically adds CSRF token to check capabilities of this
            // authenticator
            new CsrfTokenBadge('authenticate', $request->request->get('_csrf_token')),
        ]
    );
}
```

login.html.twig :

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

Page de connexion :

```
<input type="hidden" name="_csrf_token" value="ba8fa39d74e9845ee3d.Mo3f7-haZwhLlt8UcK-Ls0Pf
6rtzi9E8Z3ot0HApv9Q.ed-Zg689FUIo2uomQNrz5rGute8R_vxPMgBIvhtLx5hB6affhxBQSgHFqA">
```

Utilisateurs

Pour éviter des doublons d'utilisateur dans base de données, nous avons mis en place l'unicité de l'email avec l'utilisation de « `use`

`Symfony\Bridge\Doctrine\Validator\Constraints\UniqueEntity` » et l'indication « `@UniqueEntity("email")` » au début de l'entité `User`.

```
/**
 * @ORM\Entity(repositoryClass=UserRepository::class)
 * @UniqueEntity("email")
 */
class User implements UserInterface,
PasswordAuthenticatedUserInterface
/**
 * @ORM\Column(type="string", length=180, unique=true)
 * @Assert\NotBlank
 * @Assert\Email
 */
private $email;
```

De plus, les mots de passes sont hashés avant d'être enregistrés en base de données, et ce grâce à `PasswordHasher` et sa méthode `hashPassword()`.

Access Control

Enfin, l'accès aux différentes parties du site est contrôlé par l'utilisation des rôles attribués aux utilisateurs. Cela est effectué dans le fichier `security.yaml` et les rôles sont récupérés

`access_control:` dans l'entité `User`.

```
# ROUTES DISPO POUR LES UTILISATEURS CONNECTÉS
- { path: '^/favoris/list', roles: ROLE_USER }
# ROUTES DISPO UNIQUEMENT POUR L'ADMIN
- { path: '^/back', roles: ROLE_ADMIN }
```

La route des favoris n'est accessible qu'aux utilisateurs ayant le rôle `ROLE_USER`.

Pour les routes accédant au *back-office*, celles-ci sont accessibles uniquement pour un utilisateur ayant le rôle `ROLE_ADMIN`.

Les autres routes de l'application sont, elles, accessibles par tous les utilisateurs, avec ou sans rôle. Un utilisateur ayant le rôle `ROLE_USER` qui essaiera d'accéder à une route du *back-office* recevra une erreur 404 « Page perdue ».

GESTION DE PROJET

Présentation de l'équipe et des rôles

Notre équipe était composée de quatre personnes. L'attribution des rôles s'est faite avant même le début du développement du projet. Nous avons discuté entre nous afin de savoir quel poste pourrait convenir à chacun. Enfin, nous avons créé un tableau avec *Google Sheets* avec les différents rôles et les membres de l'équipe. Chacun devait mettre un ordre de préférence pour chaque rôle. Lors de la phase de développement, une réorganisation a eu lieu afin de réattribuer les rôles aux personnes ayant le plus de compétences pour elles.

ation avancée ». C'est elle qui a trouvé l'idée du rôle qu'elle a endossé son rôle. Très impliquée, elle n'a pas rencontrées. Elle a majoritairement travaillé sur le front-end, la intégration HTML et CSS en amont.

n de faire la spécialisation *Symfony*. Cependant, j'ai commencé pour le back, ce qui fait que j'ai beaucoup travaillé

sur les deux parties. J'ai énormément codé en *pair-programming* avec xxx. Je me suis aussi découverte relativement à l'aise avec *Git*, ce qui a fait que j'ai rapidement récupéré le rôle de Git master.



XXX XXXX

Référent Technique

Il a également effectué la spécialisation « Intégration avancée » avec xxx. Bien qu'il ait eu des difficultés, il n'a cessé de chercher des solutions aux problèmes qu'il rencontrait, tout cela toujours dans la bonne humeur. C'est pour cela qu'il a eu son rôle.



XXX XXXX

Scrum Master et Lead Dev Back

Il a effectué la spécialisation *Symfony* avec moi. Il a les qualités d'un leader et pris son rôle très à cœur. Il a fait preuve d'un grand calme lors des moments de stress. Il est à l'aise aussi bien avec le front qu'avec le back, tout en ayant une préférence pour le back. Il m'a été d'une grande aide et d'un grand soutien lorsque je rencontrais des difficultés.

Organisation de travail

Pour notre organisation de travail, nous avons choisi d'utiliser la méthode *Agile*. Celle-ci a pour but d'apporter de la souplesse et de la performance dans la gestion du projet, et d'inclure le client au centre de celui-ci.

Plus précisément, nous avons choisi de développer notre application selon la méthode agile *Scrum* (comme la mêlée du rugby). Nous avons agi en équipe pour gagner en efficacité en utilisant les forces de chacun pour mener à bien le projet. Nous avons travaillé en cycles de développement courts, appelés *sprints*, d'une durée d'une semaine.

O'Beer a été développé en quatre *sprints*, soit quatre semaines.

Avant les sprints, il y a la première phase d'un projet géré en méthode *Scrum* : celle où le client explicite ses besoins.

Vient ensuite la seconde phase, celle de la conception de la solution.

Enfin, il y a la troisième phase qui correspond à la construction du produit.

Lors des deux premières phases, le *Product Owner* et le client listent toutes les fonctionnalités attendues en écrivant des *User Stories*. Celles-ci sont contenues dans le *Product Backlog*, qui peut être amené à évoluer au cours du projet en fonction des retours du client. Il sert de référentiel.

La troisième phase est celle qui comporte les *sprints*.

Chaque *sprint* commence par le planning des tâches à effectuer. En présence du *Product Owner*, l'équipe *Scrum* va choisir dans le *Product Backlog* les fonctionnalités qui seront développées pendant le *sprint* qui arrive. L'équipe tient toutefois compte du statut bloquant d'une tâche pour pouvoir en développer une autre, de la valeur ajoutée, de la complexité et du temps nécessaire à sa réalisation. L'équipe se crée ainsi un *Sprint Backlog* (carnet de *sprint*). Contrairement au *Product Backlog*, celui-ci ne sera pas modifié en cours de *sprint*.

Durant toute la durée des *sprints*, il y a chaque jour une réunion de quelques minutes qui permet de faire le point sur les réalisations effectuées la veille et celles que l'on prévoit de faire le jour même. On parle également des difficultés rencontrées. Cette réunion est animée par le *Scrum Master* et s'appelle le *Daily Scrum*.

À la fin de chaque *sprint*, une *Sprint Review* (revue de *sprint*) est tenue par l'équipe *Scrum* en présence du client et du *Product Owner*. Un point sur ce qui a été réalisé et livré par l'équipe est fait. On parle des éventuels points bloquants, de si on est en retard ou en avance, on rappelle les objectifs et on fait une démonstration. On peut aussi aborder le budget, le planning, la concurrence, etc... tout ce qui permet de faire les meilleurs choix pour le produit final.

Enfin, il y a la rétrospective. C'est une réunion qui se fait sans le client où l'équipe *Scrum* analyse le fonctionnement du *sprint* écoulé. On cherche à trouver ce qui peut être amélioré pour le prochain *sprint*. Le but est d'apprendre des erreurs et problèmes du *sprint* passé afin que les *sprints* suivants se déroulent dans les meilleures conditions possibles.

Outils utilisés

Nous nous sommes servis de l'outil en ligne Trello (cf annexes) pour la réalisation de nos *Product Backlog* et *Sprint Backlog*.

Google Drive nous a servi de dépôt. Tous les documents nécessaires au projet étaient à disposition de l'équipe et de l'équipe pédagogique. Le dossier *Drive* a été fourni par l'école O'Clock.

Pour la réalisation des *Wireframes*, nous nous sommes servis de Whimsical. L'avantage de cet outil est qu'il est collaboratif et en ligne. On pouvait donc en permanence se référencer aux maquettes faites.

Visual Studio Code était déjà installé dans l'environnement de développement fourni par l'école, nous nous sommes donc servis de cet IDE (*Integrated Development Environment*) pour coder.

Git nous a permis de versionner notre code, qui a ensuite été sauvegardé à distance sur GitHub.

Pour les discussions écrites de l'équipe et le partage d'informations importantes, nous avons utilisé Slack.

Tous les *daily scrums* et toutes les rétrospectives de *sprint* ont été faits sur *Discord*. Celui-ci nous permettait de pouvoir se parler ainsi que de partager nos écrans en streaming. Nous ne nous sommes pas servis de la webcam.

Lors de *pair-programming*, nous étions en appel et streaming sur *Discord*, tout en effectuant un *LiveShare* via *Visual Code Studio*.

Programme des sprints

Comme dit précédemment, la durée de développement du projet étant très courte, nos *sprints* avaient une durée d'une semaine.

Lors du *Sprint 0*, qui correspondait aux deux premières phases, nous avons établi les besoins du projet.

C'est là que les rôles ont été attribués, et que nous avons mis en place les journaux de bord de l'équipe et les journaux individuels.

À la fois en équipe et seul, nous avons également rédigé et réalisé tous les documents de conception :

- cahier des charges,
- MCD, MLD, Dictionnaire de Données,
- User Stories,
- *Product Backlog*,
- routes,
- *Wireframes*,

Tâches effectuées lors du *sprint 1*, en pair-programming, parfois en solo :

- préparation intégration
- installation du squelette *Symfony*
- mise en place de la base de données
- requêtes SQL pour remplir la base de données
- création de la base de données chez toute l'équipe
- création de contrôleurs et de vues
- création de branches sur GitHub
- début de développement de la fonctionnalité de favoris
- début du développement du questionnaire de vérification d'âge
- découverte et gestion de conflits lors des *commits* et des *merge*
- debug

Tâches effectuées lors du *sprint 2*, en pair-programming, parfois en solo :

- intégration dans les *templates Twig*
- intégration de la carte
- développement du questionnaire de vérification d'âge

- finition fonctionnalité de favoris
- mise en place du *back-office*
- mise en place du CRUD des brasseries dans le *back-office*
- mise en place du CRUD des utilisateurs dans le *back-office*
- fonctionnalité de connexion
- fonctionnalité de création de compte
- mise en place de l'*Access Control*
- affichage des brasseries
- gestion de conflits lors des *commits* et des *merge*
- gestion erreur et création page 404
- debug

Tâches effectuées lors du *sprint 3*, en pair-programming, parfois en solo :

- création des derniers *templates*
- dynamisation des *templates*
- dynamisation de la carte
- redirection au clic sur une région de la carte
- mise en place des carrousels
- debug
- gestion de conflits lors des *commits* et des *merge*
- correction du responsive
- sécurisation des formulaires
- déploiement

Bien que le temps fût court, nous prenions quand même le temps d'effectuer tous les matins un *Daily Scrum*. Le but de ce genre de réunions est de durer environ 15minutes, afin de ne pas trop déborder sur le temps de développement et de retourner rapidement à nos tâches à effectuer. Malheureusement, nos *daily scrums* ont duré tout le long du projet beaucoup plus que 15 minutes.

Malgré toute la communication faite, nous n'avons pas réussi à solutionner nos problèmes d'organisation et de processus.

RÉALISATIONS PERSONNELLES

Pour mes réalisations personnelles, j'ai contribué à la réalisation des documents de conception lors du *sprint 0*.

En outre, j'ai fait beaucoup de *Wireframes*, notamment ceux concernant le *back-office*, et ai fait la majorité des commentaires autour.

Lors de la phase de développement, j'ai pu utiliser mes compétences acquises au cours de la formation à la fois pour la partie *Front* et pour la partie *Back*.

Affichage menu favoris et menu connexion en fonction du rôle

Notre application ayant deux rôles et donc des fonctionnalités inhérentes à ceux-ci, j'ai voulu que le menu de navigation s'adapte en fonction.

Pour ce faire, j'ai effectué des conditions dans le *template base.html.twig*, qui sont comparables à celle de PHP.

```
{% if is_granted('ROLE_USER') %}  
    <li class="nav-item">  
        <a class="nav-link" href="{{ path('appFavoritesList') }}>  
            Mes Favoris  
        </a>  
    </li>  
{% endif %}
```

Ici, si l'utilisateur a le rôle utilisateur (*ROLE_USER*), alors la liste affiche un nouvel élément « Mes Favoris » avec son lien vers la page de la liste de favoris.

```

<!--Dropdown-->
{# admin connected #}
{%- if is_granted('ROLE_ADMIN') %}


- {{ app.user.pseudo }}


  - ###### Administrateur
  - Se déconnecter
  - Backoffice
  - Utilisateurs
  - Brasseries


{%- elseif is_granted('ROLE_USER') %}
{#USER CONNECTED#}


- {{ app.user.pseudo }}


  - ###### Membre
  - Se déconnecter


{%- else %}
{# user disconnected #}
<!--Dropdown form-->


<button class="btn btn-secondary dropdown-toggle" type="button" data-bs-toggle="dropdown">
Connexion

<a class="dropdown-item" href="{{ path('app_login') }}>Se connecter</a>
<a class="dropdown-item" href="{{ path('app_user_new') }}>S'inscrire</a>


```

Dans le cas ici, on peut voir trois blocs distincts ayant le même style de disposition.

Dans les deux cas d'utilisateurs connectés, il y a le statut indiqué dans la balise `<h6>`. Il y a aussi toujours le lien pour la déconnexion.

Dans le cas d'un administrateur connecté, il y a les liens vers la page d'accueil du *back-office*, ainsi que les liens directs vers la gestion des utilisateurs et des brasseries.

Dans le cas d'un utilisateur connecté, le lien de déconnexion est affiché et je lui indique son statut de membre.

Dans le cas d'un visiteur non authentifié, il y a le lien vers la connexion et celui vers la création de compte.

Requête DQL pour afficher la liste des brasseries par ordre alphabétique

J'ai développé les méthodes nécessaires afin de pouvoir afficher la liste de toutes les brasseries. Pour ce faire, j'ai tout d'abord écrit et testé la requête SQL dans *Adminer*. Une fois fonctionnelle, j'ai pu développer les méthodes nécessaires.

Pour qu'elle fonctionne, j'ai eu besoin de créer une méthode dans *BrewerieRepository*.

```
/**  
 * Find all ordered by title ASC using DQL  
 */  
public function findAllOrderedByStateAscDql()  
{  
    // Need to use EntityManager to create a request with Doctrine  
    $entityManager = $this->getEntityManager();  
  
    // Request creation  
    $query = $entityManager->createQuery(  
        // SELECT all object $brewerie FROM Brewerie  
        // ordered alphabetically by state and name  
        'SELECT  
        FROM App\Entity\Brewerie AS b  
        ORDER BY b.state, b.name ASC')  
;  
  
    // returns an array of Product objects  
    return $query->getResult();  
}
```

Puisque l'on est dans un pattern *Data Mapper*, j'ai besoin d'utiliser *EntityManager* de *Doctrine* pour faire la requête.

J'ai choisi d'utiliser le format DQL pour écrire ma requête. Celle-ci a pour but de sélectionner toutes les brasseries dans la table *brewerie*, en

organisant par ordre alphabétique de régions, puis de brasseries.

Puis j'ai développé la méthode *list()* dans *BrewerieController.php*.

```
/**  
 * Road for reach all breweries by list  
 *  
 * @Route("/brasseries/liste", name="app_breweries_list")  
 */  
public function list(BrewerieRepository $brewerieRepository): Response  
{  
    // Get all breweries in alphabetical order of states and name with  
    // findAllOrderedByStateAscDql from BrewerieRepository  
    $breweries = $brewerieRepository->findAllOrderedByStateAscDql();  
  
    // Redirect to template list.html.twig  
    return $this->render('brewerie/list.html.twig',  
    [  
        'breweries' => $breweries,  
    ]);  
}
```

Pour qu'elle fonctionne, j'ai besoin de la

requête contenue dans *BrewerieRepository*. Je mets donc ce dernier en paramètre de ma fonction pour pouvoir appeler la méthode dans la fonction. Je stocke dans mon objet *\$breweries* le résultat de ma méthode *findAllOrderedByStateAscDql()*. Je génère ensuite le template *list.html.twig* et lui passe l'objet *\$breweries* afin de pouvoir utiliser son contenu pour dynamiser le template.

Dans mon template *list.html.twig*, je peux maintenant me servir du résultat de ma requête afin de dynamiser les différentes balises.

```
<h2 class="fs-1 text-dark fw-bold mb-4 text-center"> {{ app.request.get('state') }} </h2>

{%- for brewerie in breweries %}

    <div class="row g-0 border rounded overflow-hidden flex-md-row shadow-sm bg-white position-relative">
        <div class="p-4 d-flex flex-column position-static text-center">
            <a href=" {{ path('appFavoritesAdd', {id:brewerie.id}) }} " class="fs-1 my-2 mx-3 link-danger">
                <i class="bi bi-bookmark-x"></i>
            </a>

            <div class="fs-2 fw-bolder text-warning mb-1 ">{{ brewerie.name }}</div>
            <div class="mb-1 text-secondary">{{ brewerie.address }} </div>
            <div class="mb-1 text-secondary">{{ brewerie.zipcode }} {{ brewerie.city }} </div>
            <div class="mb-1 text-secondary">{{ brewerie.state }} </div>
            <div class="mb-1 text-secondary"><a class="fst-italic text-muted" target="_blank" href="{{ brewerie.website }}">{{ brewerie.website }}</a></div>

            <a href="{{ path('appBrewerieShow', {id: brewerie.id}) }}" class="fs-1 mt-3 text-danger align-self-start">
                <i class="bi bi-arrow-right-square"></i>
            </a>
            </div>
        </div>
    {%- endfor %}
</div>
```

Je crée une boucle *for...in* au début du template. Celle-ci est l'équivalente de la boucle *foreach* en PHP.

Pour chaque brasserie dans brasseries (*for brewerie in breweries*, *breweries* étant la clé du tableau envoyé par le contrôleur lors du rendu du *template*), je récupère son nom (*name*), son adresse (*address*), son code postal (*zipcode*), sa ville (*city*), sa région (*state*), et son site internet (*website*) qui est aussi placé dans le *href* pour que l'utilisateur puisse cliquer dessus et être redirigé vers celui-ci.

Il y a également deux balises *<a>* qui redirigent vers la page de détails de la brasserie grâce à « *path('appBrewerieShow', {id :brewerie.id})* » et vers l'ajout de la brasserie dans les favoris grâce à « *path('appFavoritesAdd', {id :brewerie .id})* ».

La boucle affichera ce modèle autant de fois qu'il y a de brasseries contenues dans l'objet *\$breweries*.

Favoris

Dans le MVC de l'application, la fonctionnalité primordiale pour l'utilisateur est de pouvoir faire une liste de favoris.

Pour développer cette fonctionnalité, j'ai utilisé un service.

```
●●●<?php

namespace App\Service;

use App\Entity\Brewerie;
use App\Repository\BrewerieRepository;
use Symfony\Component\HttpFoundation\Session\SessionInterface;

/**
 * Service that manages the favorites breweries in a session
 */
class FavoritesManager
{
    private $session;
    private $brewerieRepository;

    public function __construct(SessionInterface $session, BrewerieRepository $brewerieRepository)
    {
        // Setup the session
        $this->session = $session;
        $this->brewerieRepository = $brewerieRepository;
    }

    /**
     * Method for adding favorites breweries and stock in session
     *
     * @return void
     */
    public function add(int $id, Brewerie $brewerie = null)
    {

        // Declare $favorites as an array
        $favorites = $this->session->get('favorites', []);

        // Condition that verify if the brewery is already added to favorites
        // if not the brewery is added on the list
        if (!array_key_exists($id, $favorites))
        {
            // add the favorite in the array using id of the brewery
            $favorites[$id] = $brewerie;
            // update the favorites list
            $this->session->set('favorites', $favorites);
            // Return the message "added" from flash message
            return 'added';
        }
        // If exists, the method doesn't modify the favorites list
        else
        {
            // Return flash message "already added"
            return 'already added';
        }
    }

    /**
     * Method to clear all favorites from the list in the session
     *
     * @return void
     */
    public function clear()
    {
        $this->session->remove('favorites');
    }

    /**
     * Method to delete one brewerie from the favorite list
     *
     * @param integer $id
     * @return void
     */
    public function removeFavorite(int $id)
    {
        // Getting the list of the favorites brewerie added in the session
        $favorites = $this->session->get('favorites', []);
        // Using "unset" method to delete the brewerie depending on the id associated
        unset($favorites[$id]);
        // update of the favorites list
        $this->session->set('favorites', $favorites);
    }
}
```

J'ai créé un fichier *FavoritesManager* que j'ai placé dans un dossier nommé *Service*.

Je crée une méthode *__construct* qui me permet d'automatiquement initialiser la session et *BrewerieRepository* à chaque instanciation de la classe, afin de pouvoir les utiliser dans toute la classe *FavoritesManager*.

Pour ajouter un favori, je mets en paramètres l'*id* d'une brasserie sélectionnée et l'entité *Brewerie* dans ma méthode *add()*.

Je déclare une variable *\$favorites* et lui attribue le tableau « 'favoris' » récupéré dans la session.

J'écris ensuite une condition. S'il n'y a pas les clés *id* dans de tableau 'favorites', alors *\$favorites* récupère la brasserie grâce à son *id* et le tableau est ajouté à celui des favoris dans la session. Pour que l'utilisateur sache que l'action a été effectuée, un message 'added' est retourné.

Sinon, la brasserie était déjà dans les favoris et un message 'already added' est retourné.

Pour vider entièrement la liste de favoris, je retire le tableau ‘favorites’ de la session.

Pour supprimer un favori de la liste, j’ai besoin de l’*id* d’une brasserie en paramètre.

Je récupère le tableau ‘favorites’ dans la session et retire le favori dont l’*id* correspond à celui du paramètre.

```
class FavoritesController extends AbstractController
{
    private $favoritesmanager;
    public function __construct(FavoritesManager $favoritesManager)
    {
        // Get instance of favoritesManager by $favoritesManager as a service
        $this->favoritesmanager = $favoritesManager;
    }

    /**
     * Road to the favorites user list
     *
     * @Route("/favoris/liste", name="app_favorites_list", methods={"GET"})
     */
    public function list(): Response
    {
        // Redirect to the template list.html.twig
        return $this->render('favorites/list.html.twig');
    }

    /**
     * Add brewerie to favorites
     *
     * @param int $id brewerie id
     *
     * @Route("/favoris/add/{id<\d+>}", name="app_favorites_add", methods={"GET"})
     * @see https://symfony.com/doc/5.4/session.html#basic-usage
     */
    public function add(int $id, Brewerie $brewerie = null)
    {
        // Handle the case that the searched brewery doesn't exist
        if ($brewerie === null)
        {
            throw $this->createNotFoundException('Brasserie indisponible');
        }

        // Call of function "add" from the favoritesManager + message if the brewerie is added
        if ($this->favoritesmanager->add($id, $brewerie) == 'added')
        {
            $this->addFlash('success', "{$brewerie->getName()} a été ajouté à votre liste de favoris.");
        }
        // Call of function "add" from the favoritesManager + message if the brewerie is already added
        elseif ($this->favoritesmanager->add($id, $brewerie) == 'already added')
        {
            $this->addFlash('warning', "{$brewerie->getName()} existait déjà dans votre liste de favoris.");
        }

        // Redirect to the template list.html.twig from favorites
        return $this->redirectToRoute('app_favorites_list', ['brewerie' => $brewerie]);
    }

    /**
     * Method to clear all favorites from the list
     *
     * @Route("/favoris/suppression", name="app_favorites_clear", methods={"GET"})
     */
    public function clear()
    {
        // Call of clear method from FavoritesManager
        $this->favoritesmanager->clear();

        // Message flash for confirm clearing
        $this->addFlash('success', 'Votre liste de favoris a été supprimée.');

        // Redirect to the template list.html.twig
        return $this->redirectToRoute('app_favorites_list');
    }

    /**
     * Methode to delete one brewerie from favorites list
     *
     * @Route("/favoris/suppresion-element/{id<\d+>}", name="app_favorites_remove", methods={"GET"})
     */
    public function remove(int $id, Brewerie $brewerie = null)
    {
        // Call of removeFavorite methode from FavoritesManager
        $this->favoritesmanager->removeFavorite($id);
        // Add a flash message to confirm deletion
        $this->addFlash('success', "{$brewerie->getName()} a bien été supprimée.");
        // Redirect to the template list.html.twig
        return $this->redirectToRoute('app_favorites_list');
    }
}
```

J’ai créé ensuite le contrôleur *FavoritesController* grâce à la commande de *Symfony* « *symfony console make:controller FavoritesController* »

Je crée également une méthode *__construct* pour initialiser *FavoritesManager* et pouvoir l’utiliser lors de l’instanciation de la classe.

Pour ajouter un favori, la méthode *add()* prend en paramètres l’*id* et *Brewerie*.

J’écris ensuite une suite de conditions.

Si *\$brewerie* est strictement égale à *null*, alors j’utilise la méthode *createNotFoundException()* qui

est un raccourci pour créer un objet *NotFoundHttpException*, qui génère une réponse HTTP 404 dans Symfony avec le message « Brasserie indisponible ».

Si la méthode *add* du *FavoritesManager* a retourné « *added* » suite à l'ajout d'une brasserie, alors j'envoie le message « [Nom de la brasserie] a été ajoutée à votre liste de favoris. ». Dans le cas où la brasserie était déjà dans les favoris, j'envoie le message « [Nom de la brasserie] « existait déjà dans votre liste de favoris. »

Puis je redirige l'utilisateur vers la route *app_favorites_list* à laquelle je passe l'objet *Brewerie*, qui affichera la liste des favoris dynamisée.

Pour vider la liste de favoris, la méthode *clear()* du *FavoritesManager* et renvoie le message « Votre liste de favoris a été supprimée. ». Puis je redirige l'utilisateur vers la page de favoris.

Enfin, la méthode *remove()*, qui prend en paramètres l'*id* et *Brewerie*, appelle la méthode *remove()* du *FavoritesManager*. À la suppression d'un favori, j'envoie le message « [Nom de la brasserie] a bien été supprimée. » et je redirige vers la page de favoris.

Template *list.html.twig* des favoris :



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

{% block title %}Mes favoris{% endblock %}

{% block body %}

<!-- main -->
<div class="container-xl bg-light p-5">

    <div class="col-12 col-lg-12">
        <h2 class="fs-1 text-dark fw-bold mb-4 text-center">Nos brasseurs vous remercient </h2>
        {{ include('_flash_messages.html.twig') }}

        {% for index, brewerie in app.session.get('favorites', []) %}

            <div class="row g-0 border rounded overflow-hidden flex-md-row mb-4 shadow-sm bg-white position-relative text-center">
                <div class="p-4 d-flex flex-column position-static">
                    <a href="{{ path('app_favorites_remove', {id:brewerie.id}) }}" class="fs-1 my-2 mx-3 link-danger"><i class="bi bi-bookmark-x-fill"></i></a>
                    <div class="fs-2 fw-bolder text-warning mb-1 ">{{ brewerie.name }}</div>
                    <div class="mb-1 text-secondary">{{ brewerie.address }}</div>
                    <div class="mb-1 text-secondary">{{ brewerie.zipcode }}</div>
                    <div class="mb-1 text-secondary">{{ brewerie.state }}</div>
                    <div class="mb-1 text-secondary"><a class="text-secondary" href="">{{ brewerie.website }}</a></div>

                    <a href="{{ path('app_brewerie_show', {id: brewerie.id} ) }}" class="fs-1 mt-3 text-danger align-self-start">
                        <i class="bi bi-arrow-right-square"></i>
                    </a>
                </div>
            {% endfor %}
        </div>
    </div>

    <div class="col">
        <a href="{{ path('app_favorites_clear') }}" class="btn btn-danger btn-sm" onclick="return confirm('Vider votre liste de favoris ?');">Vider la liste</a>
    </div>
</div>

{% endblock %}
```

Je crée une boucle *for...in* pour pouvoir dynamiser la page.

Pour les parties à dynamiser, j'utilise « *brewerie* » qui est la clé de mon tableau *favorites* dans la session, et j'accède aux différentes valeurs dont j'ai besoin : *name*, *address*, *zipcode*, *city*, *state*, *website*.

Les balises *href* redirigent également vers la suppression du favori ou la suppression de toute la liste de favoris.

Cette fonctionnalité a donc été faite à l'aide d'un service et en utilisant la session de l'utilisateur, par décision majoritaire du groupe. Or, sur chaque support qu'il utilisera, l'utilisateur aura d'autres favoris.

Cette manière de faire, de mon point de vue, n'était pas la bonne. En effet, l'expérience utilisateur (UX) ne sera pas bonne. Qui a envie de perdre ses favoris lorsqu'il change de support ?

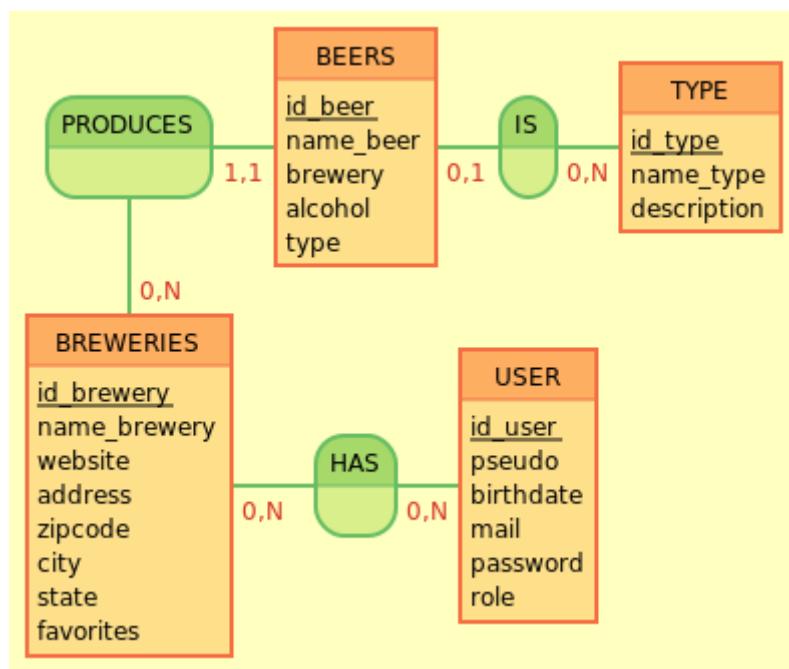
Personnellement, j'aurais préféré que les favoris de l'utilisateur soient stockés en base de données. Comme cela, il récupérera tous ses favoris après s'être connecté et ce, peu importe le support.

J'ai donc prévu de modifier cela pour la version 2 de l'application.

Pour que les brasseries favorites de chaque utilisateur soient enregistrées en base de données, il faudra ajouter une propriété *favorites* à la brasserie.

Celle-ci sera reliée à l'entité *user* par une relation *ManyToMany*. Par conséquent, une table de liaison *has_favorites* sera créée. Elle ne comportera que des clés étrangères, qui seront les *id* de *user* et de *brewerie*. *User* ne sera plus esseulée.

Le MCD deviendra alors :



Le MLD :

BEERS (id_beer, name_beer, brewery, alcohol, type, #id_brewery, #id_type)
BREWERIES (id_brewery, name_brewery, website, address, zipcode, city, state, favorites)
HAS (#id_user, #id_brewery)
TYPE (id_type, name_type, description)
USER (id_user, pseudo, birthdate, mail, password, role)

Et le MPD :



PRÉSENTATION DU JEU D'ESSAI

Les jeux d'essai sont des descriptions d'une suite d'actions et des résultats attendus. Ils ont pour objectif de valider une fonctionnalité d'une application.

Étant donné que la fonctionnalité de création de compte est celle qui a été testée de façon la plus approfondie, le tableau ci-dessous et les essais décrits ne concernent que cette fonctionnalité. Ils ont été réalisés avec le formulaire de création de compte lors de la soumission du formulaire grâce au « clic » sur le bouton « Créer ».

Le formulaire est composé de 5 champs (dont 4 sont obligatoires) avec un champ qui est composé de 3 listes déroulantes (celui de la date de naissance).

Les champs du mot de passe et de confirmation du mot de passe ont une contrainte d'au minimum 6 caractères et doivent être identiques.

Le champ de l'email doit correspondre au format d'un email.

Champ testé	Action utilisateur	Action attendue	Action réelle	Conclusion
Pseudo	Ne saisit rien	Aucun message d'erreur, l'utilisateur peut ne rien mettre	Aucun message d'erreur	Test concluant
Pseudo	Ne saisit que des espaces	Aucun message d'erreur, l'utilisateur peut ne mettre que des espaces	Aucun message d'erreur	Test concluant

Pseudo	Saisit un ou des caractères	Aucun message d'erreur	Aucun message d'erreur	Test concluant
Birthdate	Choisit un jour, un mois et une année	Aucun message d'erreur	Aucun message d'erreur	Test concluant
Birthdate	Modifie les valeurs des jours dans l'inspecteur	Message d'erreur : valeur non valide	Message d'erreur : valeur non valide	Test concluant
Email	Ne saisit rien	Message d'erreur : ce champ ne peut pas être vide	Message d'erreur : ce champ ne peut pas être vide	Test concluant
Email	Saisit un email sans @	Message d'erreur : le format de l'email n'est pas valide	Message d'erreur : le format de l'email n'est pas valide	Test concluant
Email	Saisit un email sans extension de domaine	Message d'erreur : le format de l'email n'est pas valide	Message d'erreur : le format de l'email n'est pas valide	Test concluant
Email	Saisit une adresse email	Aucun message d'erreur	Aucun message d'erreur	Test concluant
Mot de passe	Ne saisit rien	Message d'erreur : ce champ ne peut pas être vide	Message d'erreur : ce champ ne peut pas être vide	Test concluant
Mot de passe	Ne saisit qu'un caractère	Message d'erreur : trop court, ce champ doit contenir 6 caractères ou plus	Message d'erreur : trop court, ce champ doit contenir 6 caractères ou plus	Test concluant
Mot de passe	Ne saisit que des espaces	Message d'erreur : ce champ ne peut pas être vide	Aucun message d'erreur	Ce cas n'a pas été traité, le test est donc non concluant
Mot de passe	Saisit un mot de passe d'au moins 6 caractères	Aucun message d'erreur	Aucun message d'erreur	Test concluant
Confirmer le mot de passe	Ne saisit rien	Message d'erreur : ce champ ne peut pas être vide	Message d'erreur : ce champ ne peut pas être vide	Test concluant

Confirmer le mot de passe	Ne saisit qu'un caractère	Message d'erreur : trop court, ce champ doit contenir 6 caractères ou plus	Message d'erreur : trop court, ce champ doit contenir 6 caractères ou plus	Test concluant
Confirmer le mot de passe	Ne saisit que des espaces	Message d'erreur : ce champ ne peut pas être vide	Aucun message d'erreur	Ce cas n'a pas été traité, le test est donc non concluant
Confirmer le mot de passe	Saisit le même mot de passe d'au moins 6 caractères	Aucun message d'erreur	Aucun message d'erreur	Test concluant
Mot de passe et confirmer le mot de passe	Saisit différente pour chaque champ	Message d'erreur : les mots de passe de correspondent pas	Message d'erreur : les mots de passe de correspondent pas	Test concluant
Tous les champs	Les champs sont correctement remplis	- Ajout du nouvel utilisateur dans la base de données (back) - Redirection vers la page d'accueil et affichage d'un message de confirmation de création de compte qui invite l'utilisateur à se connecter (front)	- Ajout du nouvel utilisateur dans la base de données (back) - Redirection vers la page d'accueil et affichage d'un message de confirmation de création de compte qui invite l'utilisateur à se connecter (front)	Test concluant

Créer un compte

Pseudo

Birthdate
 Jan 1 1903

Email
 ⓘ
This value should not be blank.

Mot de passe
 ⓘ
This value should not be blank.
This value should not be blank.

Confirmer le mot de passe

Créer

Créer un compte

Pseudo

Birthdate
 Jan 1 1903

Email
 ⓘ
This value should not be blank.

Mot de passe
 ⓘ
Les mots de passe ne correspondent pas.

Confirmer le mot de passe

Créer

Créer un compte

Pseudo

Birthdate
 Jan 1 1903

Email
 ⓘ
This value is not a valid email address.

Mot de passe
 ⓘ
This value is too short. It should have 6 characters or more.
This value is too short. It should have 6 characters or more.

Confirmer le mot de passe

Créer

VEILLE ET RÉSOLUTION DE PROBLÈMES

Veille technologique sur les vulnérabilités de sécurité

On parle de plus en plus aujourd’hui des problèmes de *hacking* et des fuites de données qui en découlent.

Comme tout le monde, j’en ai entendu parler et j’essaie de me protéger autant que possible lorsque je vais sur internet. J’ai même souscrit à un VPN afin d’avoir une connexion privée et sécurisée.

Cependant, c’est lors de ma formation que j’ai vraiment commencé à réaliser (parce que le sujet est tellement vaste qu’il faudrait des années pour en faire le tour) ce que protéger son application signifiait et tout ce que cela impliquait. C’est d’ailleurs grâce à elle que j’ai découvert la communauté en ligne OWASP (*Open Worldwide Application Security Project*)

qui est une référence pour le développement d'applications web en matière de sécurité des données. Elle a notamment défini les 10 principales attaques web.

Ma veille sur les vulnérabilités de sécurité (et pour tout autre sujet du développement web) s'effectue principalement par des recherches à l'aide mon navigateur, les *posts* sur *LinkedIn* et les *posts* sur des groupes de développement web sur *Facebook*, qui, à force, me proposent de nouveaux articles à lire. Je ne fais pas confiance à tout ce qui est dit et j'essaie de croiser les informations. Cependant, je lis les commentaires afin de voir et d'essayer de comprendre les autres points de vue.

Dans tous les cas, je ne lis que très rarement des articles provenant de sites qui ne sont pas orientés de base dans la programmation.

Dans cet optique, j'ai régulièrement effectué des recherches sur comment *Symfony* sécurise l'application face aux injections SQL et aux attaques CSRF. Avec notamment <https://symfony.com/doc/current/security.html> et <https://symfony.com/doc/current/security/csrf.html>.

Pour être sûre que l'on avait fait de bons choix, mais aussi à causes des explications assez nébuleuses et des choses dites de manière implicite dans la documentation. Heureusement des blogs ont confirmé ce que je pensais.

Par exemple, celui-ci : <https://www.vaadata.com/blog/security-for-symfony-developers-part-1-injection-xss-auth/> .

Résolution de problèmes

Lorsque j'effectue des recherches afin de résoudre un problème, je cherche généralement de la manière suivante :

- j'écris ma requête en privilégiant l'anglais au français et j'utilise des termes simples,
- à l'affichage des résultats, je sélectionne en fonction des sources et de la date de rédaction,
- je ne me fie pas à une seule version, je croise les informations

Lors de ma sélection d'articles, je privilégie d'abord ceux de la documentation de la technologie. Ensuite viennent les articles d'organismes (comme la W3C), puis ceux de MDN. Je lis également ceux d'organismes de formations reconnus et de création d'applications web (OpenClassrooms, Ionos entre autres). Puis je lis des articles de sites et blogs orientés dans la programmation.

J'utilise également *StackOverflow* en parallèle. En effet, avoir différentes solutions à un problème donné permet de développer une meilleure réflexion. Cependant, les réponses données sont souvent faites par des personnes avec des années d'expériences, les comprendre est parfois difficile.

Dans le cadre du développement de l'application j'ai eu à résoudre différents problèmes.

Front :

Lors de l'intégration du menu dans le header (qui est sous forme de liste) il y avait des puces qui gênaient pour la mise en page. Après analyse de la partie concernée dans l'inspecteur de page, j'ai vu qu'il y avait un pseudo élément « ::marker » qui s'était ajouté. J'ai donc pensé que c'était lui posait problème. Au final, il suffisait d'ajouter « *list-style-type : none ;* » aux ** pour enlever les puces.

J'ai d'abord exécuté cette recherche :

- Mots clés de la recherche : css ::marker on inspector but not in code
<https://developer.mozilla.org/en-US/docs/Web/CSS/::marker>

Après avoir lu la documentation, la réponse ne m'indiquait pas ce que je recherchais. J'ai donc sélectionné cet article qui lui m'a apporté la solution.

<https://getcssscan.com/blog/how-to-remove-bullets-from-list-items> .

Back :

Une fois les routes créées, il allait gérer le cas d'une route non existante, la fameuse 404. Pour cela, je savais qu'il me fallait créer un nouveau contrôleur. Après avoir essayé différentes méthodes qui n'ont pas fonctionnées, j'ai effectué cette recherche : Symfony create 404 error controller

J'ai d'abord été sur ce site <https://symfonycasts.com/screencast/symfony2-ep3/error-pages> Mais je n'ai pas réussi à utiliser template « *error.html.twig* » de *Symfony*.

Je suis donc retournée sur la liste de résultats sur *Google* et j'ai vu cet article <https://divleaf.ru/en/404-symfony-controller-php> dont le sous-titre correspondait à mes attentes. Bien que je ne connaisse pas ce site, l'indicateur de confiance de mon anti-virus m'a confirmé qu'il était sûr. La *code review* que j'ai pu faire m'a aidé à comprendre ce qu'il fallait que je développe.

Pour la création des formulaires, j'ai utilisé la documentation sur les formulaires de *Symfony* <https://symfony.com/doc/current/forms.html>. Elle m'a servi à comprendre le fonctionnement des formulaires et m'a permis de trouver la liste des types de champs.

Recherche et traduction

Pour la traduction, j'ai choisi de traduire une partie de la documentation de *Symfony* que j'ai utilisé dans la partie précédente.

<https://symfony.com/doc/current/forms.html>

(Afin réduire la zone de sélection pour la capture d'écran, j'ai mis l'attribut *hidden* pour cacher l'exemple de code ; la partie traduite est entre les accolades)

Processing Forms

The [recommended way of processing forms](#) is to use a single action for both rendering the form and handling the form submit. You can use separate actions, but using one action simplifies everything while keeping the code concise and maintainable.

Processing a form means to translate user-submitted data back to the properties of an object. To make this happen, the submitted data from the user must be written into the form object:

This controller follows a common pattern for handling forms and has three possible paths:

1. When initially loading the page in a browser, the form hasn't been submitted yet and `$form->isSubmitted()` returns `false`. So, the form is created and rendered;
2. When the user submits the form, `handleRequest()` recognizes this and immediately writes the submitted data back into the `task` and `dueDate` properties of the `$task` object. Then this object is validated (validation is explained in the next section). If it is invalid, `isValid()` returns `false` and the form is rendered again, but now with validation errors.

By passing `$form` to the `render()` method (instead of `$form->createView()`), the response code is automatically set to [HTTP 422 Unprocessable Content](#). This ensures compatibility with tools relying on the HTTP specification, like [Symfony UX Turbo](#):

3. When the user submits the form with valid data, the submitted data is again written into the form, but this time `isValid()` returns `true`. Now you have the opportunity to perform some actions using the `$task` object (e.g. persisting it to the database) before redirecting the user to some other page (e.g. a "thank you" or "success" page);

Traduction :

Ce contrôleur suit un modèle commun de traitement de formulaires et dispose de trois chemins possibles :

1 - Au chargement initial de la page dans le navigateur, le formulaire n'a pas encore été soumis et `$form->isSubmitted()` retourne `false`[faux]. Le formulaire est donc créé et rendu ;

2 - Lorsque l'utilisateur soumet le formulaire, `handleRequest()`[méthode du composant `FormInterface` qui inspecte la requête donnée et appelle {@link submit()} si le formulaire a été soumis] le reconnaît et écrit immédiatement les données fournies dans les propriétés `task` et `dueDate` de l'objet `$task`. Cet objet est ensuite validé (la validation est expliquée dans la prochaine section). Si c'est invalide, `isValid()`[méthode du composant `FormInterface` qui indique si le formulaire et ses enfants sont valides] retourne `false`[faux] et le formulaire est à nouveau affiché, mais maintenant avec les erreurs de validation.

En passant `$form` à la méthode `render()` (plutôt qu'à `$form->createView()`), le code de réponse est automatiquement défini sur HTTP 422 Unprocessable Content. Ceci assure la compatibilité avec des outils qui s'appuient la spécification HTTP, comme Symfony UX Turbo ;

3 – Quand l'utilisateur soumet le formulaire avec des données valides, les données sont à nouveau écrites dans le formulaire, mais cette fois `isValid()` retourne `true`[vrai]. Maintenant vous avez l'opportunité d'effectuer certaines actions en utilisant l'objet `$task` (par exemple le conserver dans la base de données) avant de rediriger l'utilisateur vers une autre page (par exemple une page « merci » ou « succès ») ;

CONCLUSION

Ce projet de groupe fait en un mois a été une expérience enrichissante et formatrice. Elle nous a donné un bon avant-goût de ce qu'il se passe dans le quotidien d'un développeur web.

Cette période m'a énormément appris, aussi bien d'un point de vue technique que sur moi-même. Je me suis rendue compte que j'avais pu beaucoup progresser au niveau pédagogie et transmission de ce que j'avais moi-même appris. De plus, le travail que je fais sur moi-même depuis quelques années paie et m'a aidé à gérer les nombreuses situations de stress de ce mois d' « Apothéose ».

Seule petite ombre au tableau concernant notre application *O'Beer*, quand je regarde en arrière : nous avons passé énormément de temps à faire des réunions, trop sans doute, et avons parfois trop tardé à corriger certains problèmes. Pour autant, je suis fière de pouvoir présenter *O'Beer* avec son MVP mené à terme. C'était un challenge vraiment difficile mais nous l'avons relevé en équipe.

Cela m'a permis d'être à la fois sur le front et sur le back, ce qui est le but d'un développeur web fullstack, et d'aider mes camarades.

Je suis fière d'avoir autant progressé et d'avoir énormément contribué à ce que *O'Beer* puisse être présenté aujourd'hui. En revanche, je suis un peu frustrée par le résultat, j'espérais pouvoir aller plus loin dans le délai imparti mais quatre semaines, c'est vraiment court ! Si je le peux, je vais continuer à travailler un peu sur ce projet car j'ai beaucoup d'idées de fonctionnalités à développer dessus, et également envie d'améliorer l'intégration.

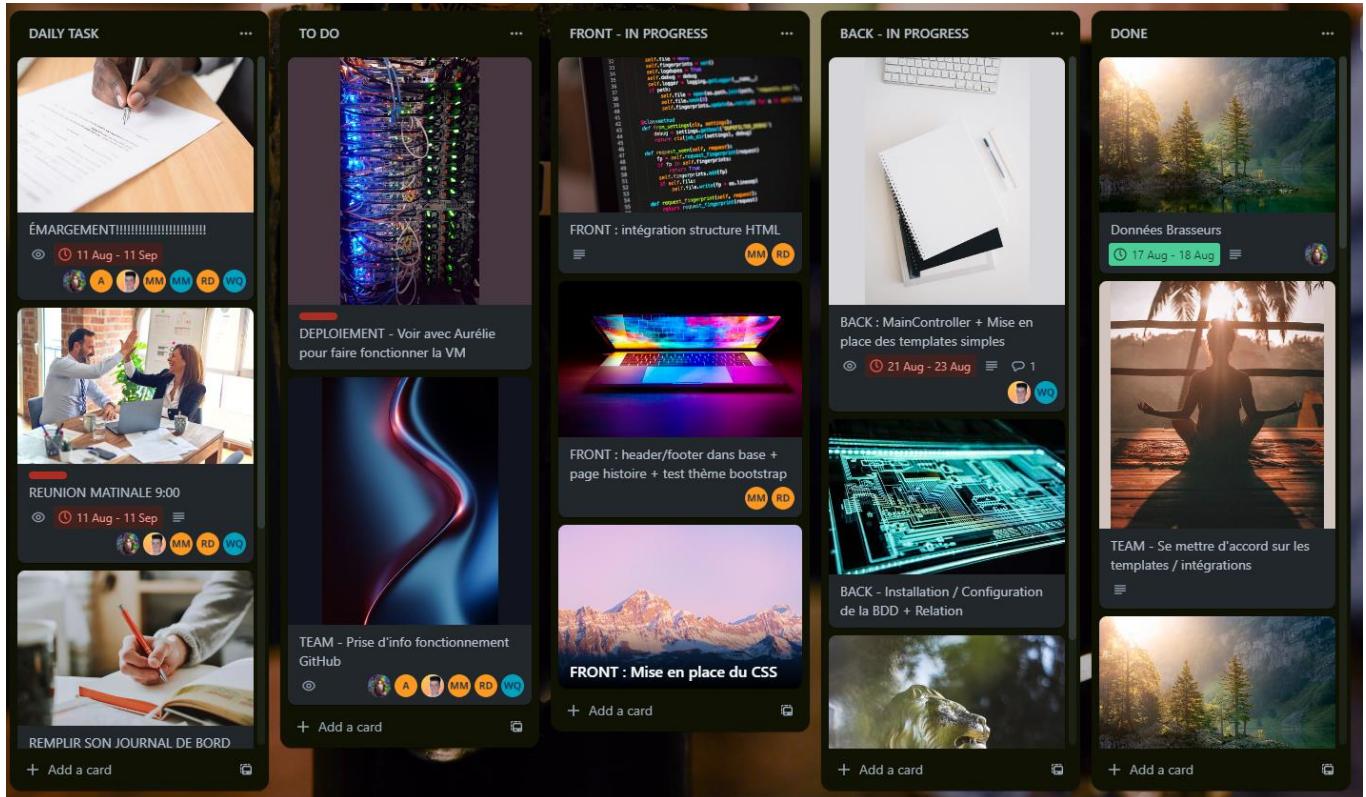
Quand je repense à ce que je savais en commençant la formation (c'est-à-dire rien) et que je vois comment je m'en sors avec du code aujourd'hui, je suis extrêmement fière de moi. Je ne suis pas parfaite, loin de là même, mais je ne suis qu'au début de mon apprentissage. Nous n'avons effleuré que « la partie émergée de l'iceberg », il y a encore tellement de choses à apprendre et à approfondir ! #juniorDev4Life comme on dit.

Post-formation, je vais continuer à apprendre et à coder. J'ai d'ailleurs quelques idées de projets personnels en tête.

Je vais aussi postuler pour décrocher un stage, si ce n'est un emploi, dans diverses entreprises, afin de pouvoir commencer ma nouvelle vie professionnelle.

ANNEXES

Trello, Product Backlog :



Popup2.js :



```
const popup = document.getElementById("popup-majority")
const main = document.getElementById("main")
const isMajor = localStorage.getItem("isMajor")
if (!isMajor) {
    displayPopup()

    const isMajorBtn = document.getElementById("is-major-btn")
    isMajorBtn.addEventListener("click", () => {
        localStorage.setItem("isMajor", true)
        displayPage()
    })

    const isNotMajorBtn = document.getElementById("is-not-major-btn")
    isNotMajorBtn.addEventListener("click", () => {
        history.back()
    })
}

function displayPopup() {
    popup.style.display = "block"
    main.style.display = "none"
}

function displayPage() {
    popup.style.display = "none"
    main.style.display = "block"
}
```

Carte.js :



```
const map = document.querySelector('#map')
const paths = map.querySelectorAll('.map__image a')
const links = map.querySelectorAll('.map__list a')

//Polyfill du foreach
if (NodeList.prototype.forEach === undefined) {
  NodeList.prototype.forEach = function (callback) {
    [].forEach.call(this, callback)
  }
}

/*Surveillance entre map et list */
const activeArea = function (id) {

  map.querySelectorAll('.is-active').forEach(function (item) {
    item.classList.remove('is-active')
  })

  if (id !== undefined) {

    document.querySelector('#list-' + id).classList.add('is-active')
    document.querySelector('#region-' + id).classList.add('is-active')
  }
}

paths.forEach(function (path) {

  path.addEventListener('mouseenter', function () {

    const id = this.id.replace('region-', '')

    activeArea(id)

  })
})

links.forEach(function (link) {
  link.addEventListener('mouseenter', function () {

    const id = this.id.replace('#list-', '')

    activeArea(id)

  })
})

map.addEventListener('mouseover', function () {

  activeArea()

})
}
```