

P I E R R E W I E T R I C H



# DOSSIER DE SYNTHESE

TITRE DÉVELOPPEUR WEB ET WEB MOBILE

ELAN FORMATION

2023

## **Remerciements**

Je souhaiterais d'abord remercier tous les formateurs d'Elan Formation, Mickael MURMANN, Quentin MATTHIEU, Stéphane SMAIL, pour leur aide, leur pédagogie, et leur patience. Ils m'ont permis d'aiguiller ce projet et de suivre la formation dans les meilleures conditions.

Je souhaite également remercier l'équipe d'Oxianet, avec laquelle j'ai beaucoup appris et reçu de précieux conseils.

Mes remerciements vont aussi à mes collègues de formation de Strasbourg, avec qui les cours étaient un vrai plaisir, dans la bonne humeur et l'entraide.

<b>A. Introduction</b>	<b>6</b>
<b>B. Conception du projet</b>	<b>7</b>
I. Le projet	7
II. Cahier des charges	8
1. User non authentifié	9
2. User authentifié	9
3. Barbier (Employé)	10
4. Barbier (Manager)	11
5. Admin	11
III. Arborescence du site	13
IV. Cibles	13
V. Design et ergonomie	14
1. Couleurs	14
2. Ergonomie	14
VI. Maquettage	15
VII. Benchmark des concurrents	16
VIII. Organisation du projet	17
1. Méthodologie	17
2. Le MVP (minimum viable product)	18
IX. RGPD	18
X. Environnement technique	20
1. Logiciels	20
2. Technologies utilisées	20
3. Langages utilisés	22
XI. Modèles de données	23

1. Modèle conceptuel de donnée	23
2. Modèle logique de donnée	24
<b>C. Développement du Projet</b>	<b>25</b>
I. Architecture du projet	25
II. Fonctionnalité représentative	28
1. Structure de la base de données	28
2. Mise en place de la prise de rendez-vous	29
3. Affichage du planning du barbier	40
4. Génération de facture	43
III. Sécurité	44
1. L'injection SQL	44
2. Faille XSS	45
3. Faille CSRF	46
4. Attaque par force brute ou par dictionnaire.	47
5. Anti-Spam	48
6. Faille Upload	50
7. Gestion des droits d'accès	51
8. Veille sur les autres failles de sécurité	52
IV. Responsive design	53
V. Référencement naturel	53
VI. API Platform	55
1. Mise en place	55
2. Exemple d'utilisation	57
VII. Situation de travail ayant nécessité une recherche	60
1. Situation	60
2. Traduction en français	60

VIII. Axes d'amélioration	61
<b>D. Conclusion</b>	<b>62</b>
<b>E. Annexes</b>	<b>63</b>

# A. Introduction

## **Présentation :**

J'ai baigné dans l'univers de l'informatique dès mon adolescence, où je passais le plus clair de mon temps sur l'ordinateur familial. Ce domaine m'a rapidement intéressé, et lorsque j'ai monté mon premier ordinateur, j'ai pu découvrir les plaisirs et la liberté qu'offre l'informatique. A la sortie du bac, incertain sur mes choix d'avenir, je me suis engagé sans conviction dans un cursus commercial puis j'ai obtenu une licence de marketing digital par la suite. Je me rapprochais de ce que j'aimais : la technologie. En parallèle de ma recherche d'emploi en marketing, j'ai décidé de faire mes premiers pas dans la programmation. Une formation en ligne plus tard, je codais mes premières lignes en Python. Au fur et à mesure de mon avancement, une idée germait dans ma tête : pourquoi ne pas faire un métier en rapport avec ce que j'aime, l'informatique ? La programmation a une dimension concrète, logique, libre que j'affectionne beaucoup et que je ne retrouvais pas dans le marketing digital. J'ai mené cette idée jusqu'au bout et je suis aujourd'hui très content de cette nouvelle aventure qui me motive au quotidien.

## **Elan Formation :**

Elan Formation est un organisme de formation qui dispose de locaux dans tout le Grand Est.

Il se démarque par une individualisation et une personnalisation des parcours de formation : chaque apprenant peut avancer à son rythme, selon son niveau, son besoin de formation et ses problématiques, tout en recevant un accompagnement « sur-mesure ».

Cette méthode d'apprentissage propre à Elan Formation permet une montée en compétence et en autonomie, avec pour objectif final, un accès à l'emploi.

# B. Conception du projet

## I. Le projet

### La genèse du projet :

L'idée de réaliser ce projet m'est venue car j'ai été confronté à un problème : celui de ne jamais trouver le bon barbier, de chez qui je sortirais satisfait. Malgré la dizaine de salons que j'ai pu essayer, impossible pour moi de trouver la perle rare.

Les applications recensant des barbiers existent, mais ne sont pas spécialisées dans ce domaine en particulier et proposent souvent des coiffeurs traditionnels en plus des barbiers, sans faire de distinction : difficile de s'y retrouver.

Les photos que l'on retrouve sur ces sites sont quasi exclusivement des photos du salon, alors que le plus important pour le client, c'est la qualité de la prestation, pas l'esthétique du salon de coiffure. Les avis, bien qu'intéressants, sont très subjectifs, et ne sont pas d'une grande utilité.

En outre, les barbershops proposent souvent des produits adaptés à l'entretien des cheveux et de la barbe, que l'on ne retrouve pas toujours en grande surface ou sur internet. J'aurais aimé pouvoir commander directement un produit spécifique à un salon. De ces problématiques est né BarberHub.

### Référentiel du titre professionnel :

front-end			back-end
 Maquetter une application	✓	✓	 Créer une base de données.
 Réaliser une interface utilisateur web statique et adaptable.	✓	✓	 Développer les composants d'accès aux données.
 Développer une interface utilisateur web dynamique.	✓	✓	 Développer la partie back-end d'une application web ou web mobile.

## II. Cahier des charges

Le cahier des charges d'un site web permet d'identifier la trame du site, son identité, sa structure et les fonctionnalités que l'on veut y implémenter. Il permet de formaliser les objectifs à atteindre et donne ainsi une vision claire de ce que devra comporter l'application.

Mon application s'adresse aux personnes en quête d'un barbershop, mais aussi aux barbiers et à leur employés.

L'administrateur est le seul à pouvoir ajouter des salons barbiers. Les barbiers peuvent prendre contact avec le site via un formulaire pour prouver qu'ils sont propriétaires d'un salon et avoir ainsi accès à des fonctionnalités supplémentaires : le gérant et les employés du salon autorisés pourront modifier les informations du salon à leur guise, tandis que toute l'équipe aura accès à son propre planning de rendez-vous.

Les internautes quant à eux pourront accéder à la liste de tous les barbiers d'une ville, avec leur localisation sur une carte, des photos des réalisations, des avis, et la possibilité de prendre rendez-vous et d'acheter des produits sur une boutique propre au salon.

Une partie blog présentera des articles et des tutoriels sur l'univers barbier.

Enfin, l'application intégrera une API qui donnera accès aux informations de tous les barbiers présents sur le site pour les développeurs qui souhaitent y accéder.

Pour clarifier les fonctionnalités qui doivent être implémentées, j'ai regroupé les fonctionnalités par rôle dans des tableaux. Nous retrouverons donc 4 rôles distincts et un sous rôle pour le rôle Barbier.

En plus des rôles listés, les rôles User, Barber et Admin ont bien entendu les mêmes droits qu'une personne non connectée. Les rôles Barber et Admin ont également les droits User.

### **Les différents rôles :**

- User authentifié
- User non authentifié
- Barber ( Manager ou Employé )
- Admin

## 1. User non authentifié

Fonctionnalité	Détail
Visualiser tous les barbiers d'une ville	- La liste générale des barbiers se présente sous forme de cartes où l'utilisateur a accès aux informations principales des différents barbershops : photo, note, et une courte description.
Filtrer les barbiers	- L'utilisateur a la possibilité de filtrer les barbiers en utilisant différents critères (nombre de likes, de commentaires, note, villes...)
Visualiser les détails d'un barbershop	L'utilisateur, lorsqu'il clique sur un barbier, a accès à toutes les informations de ce dernier : <ul style="list-style-type: none"><li>- Informations</li><li>- Horaires</li><li>- Position sur une carte</li><li>- Prestations</li><li>- Photo des réalisations</li><li>- Nombre de likes</li><li>- Notes et commentaires</li></ul>
Visualiser tous les barbiers sur une carte	- L'utilisateur a accès à une carte recensant tous les barbershops ajoutés sur l'application. Il sont représentés sous forme d'icône de barber pole (l'emblème des barbiers) et un clic sur une de ces icônes fait apparaître la carte du barbershop correspondant (photo miniature, nom, note). Lorsque l'utilisateur dé-zoom de la carte, les différents marqueurs de barbershops se regroupent sous la forme de clusters avec le nombre de marqueurs qu'il y a dans une zone donnée.
Visualiser tous les articles	Sous forme de carte : <ul style="list-style-type: none"><li>- Photo</li><li>- Titre de l'article</li><li>- Courte description</li></ul>
Visualiser le détail d'un article	<ul style="list-style-type: none"><li>- Photo</li><li>- Titre</li><li>- Contenu</li><li>- Auteur et date</li></ul>

## 2. User authentifié

Fonctionnalité	Détail
Pouvoir liker un barbier	- Un utilisateur connecté a la possibilité de laisser un j'aime sur un barbershop.
Pouvoir commenter un barbier	- Il a également la possibilité de laisser un commentaire sur un barbershop. Cela comprend une notation en étoile de 0 à 5 ainsi qu'un espace de texte pour décrire la prestation reçue.
Accéder à la page « Mon profil »	- L'utilisateur connecté a accès à une page « Mon profil », qui recense les barbiers likés par l'utilisateur ainsi que ses prochains rendez-vous et ses rendez-vous passés s'il en a.
Prendre rendez-vous chez un barbier	- Il a la possibilité de prendre rendez-vous chez un barbier, en définissant une date de début et une date de fin et en choisissant une ou plusieurs prestations.
Pouvoir commander un produit	- Ajouter au panier - Acheter
Pouvoir gérer ses données	- L'utilisateur peut supprimer son compte ou modifier ses données.

### 3. Barbier (Employé)

Fonctionnalité	Détail
Afficher ses rendez-vous	Le barbier employé a accès à la liste de ses prochains rendez-vous sous forme de calendrier. Il peut visualiser la personne qui prend rendez-vous, et la ou les prestations choisies.
Gérer ses rendez-vous	Il peut également modifier la date d'un rendez-vous ou supprimer un rendez-vous.
Pouvoir générer une facture	Le barbier peut générer une facture pour chaque rendez-vous.

## 4. Barbier (Manager)

Fonctionnalité	Détail
Modifier les détails du salon	Le barbier manager a accès à une page récapitulative des informations de son salon qu'il peut modifier (horaires, description, adresse...)
Ajouter/modifier des prestations et des prix	Le barbier manager ne peut pas ajouter de prestations (ces dernières sont définies par l'admin) mais il peut choisir les prestations qui seront présentes dans son salon et leur donner un prix.
Ajouter/modifier des produits à la boutique	Le barbier manager possède une interface lui permettant d'ajouter et de retirer des produits de sa boutique.
Ajouter/supprimer des employés	Le barbier manager peut ajouter des employés et les retirer lorsqu'ils ne travaillent plus chez lui.

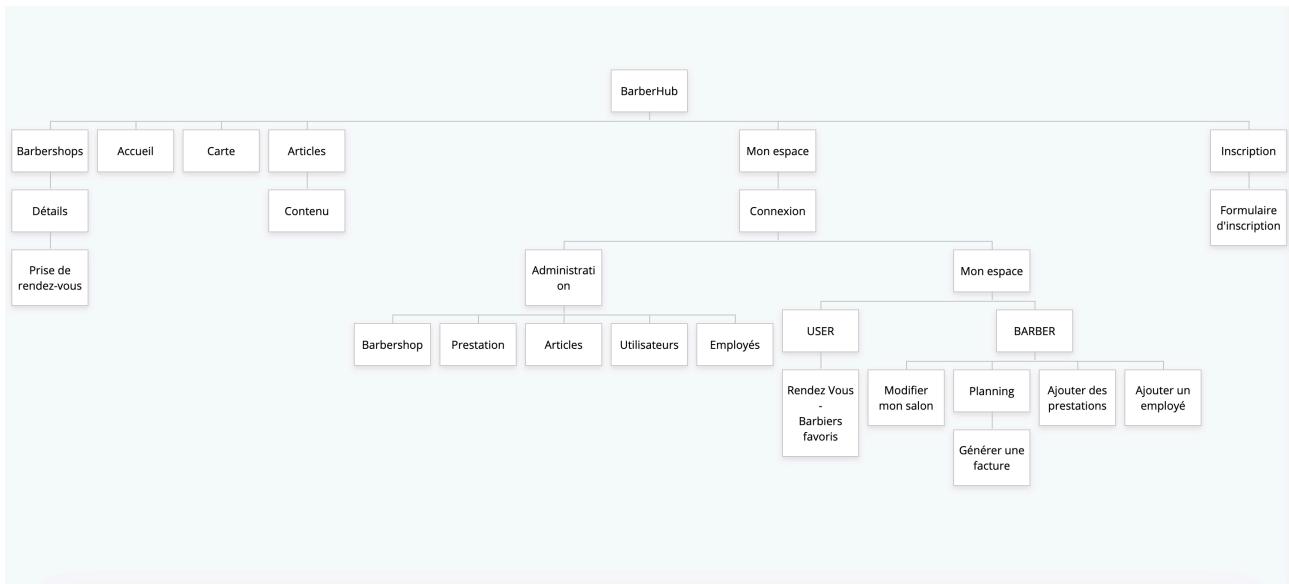
## 5. Admin

Fonctionnalité	Détail
L'admin peut ajouter/modifier/lire/supprimer (CRUD) : <ul style="list-style-type: none"> <li>- Barbershops</li> <li>- Articles</li> <li>- Avis</li> <li>- BarbershopPics</li> <li>- Personnel</li> <li>- Prestations</li> <li>- Rendez-Vous</li> <li>- User</li> </ul>	Voir le reste du tableau.
CRUD pour barbershop	L'admin peut ajouter, supprimer, modifier, créer un salon barbier.
CRUD pour articles	- L'admin est la seule personne pouvant ajouter des articles. Ces derniers se composent d'un titre, d'une image (ajoutée via une URL) et enfin d'une zone de texte avec un éditeur de texte pour rédiger le contenu de l'article. Ils peuvent être supprimés et modifiés.

Fonctionnalité	Détail
CRUD pour avis	<ul style="list-style-type: none"> <li>- L'admin peut modérer tous les avis laissés (suppression, modification.)</li> </ul>
CRUD pour barbershopPics	<ul style="list-style-type: none"> <li>- L'admin peut modifier, supprimer ou ajouter les photos des barbershops.</li> </ul>
CRUD pour Personnel	<ul style="list-style-type: none"> <li>- L'admin peut définir les rôles barbier et faire passer un User à un Personnel</li> </ul>
CRUD pour prestation	<ul style="list-style-type: none"> <li>- L'admin définit les prestations par défaut et peut en ajouter, supprimer, modifier.</li> </ul>
CRUD pour BarberPrestation	<ul style="list-style-type: none"> <li>- Même s'il n'est pas censé en ajouter (cela est réservé au barbershop), l'admin a un droit de suppression et de modification sur le prix des prestations renseignés par les barber.</li> </ul>
CRUD User	<ul style="list-style-type: none"> <li>- L'admin a la possibilité de bannir un utilisateur, de changer son rôle...</li> </ul>
- Valider/Retirer un barbershop	<ul style="list-style-type: none"> <li>- La validation permet l'affichage en page d'accueil et dans la liste générale des barbiers.</li> <li>- Les barbershops ajoutés ne sont pas publiés directement et doivent être validés par l'admin.</li> </ul>
- Visualiser tous les barbershops (validés et non validés)	<ul style="list-style-type: none"> <li>- Dans son espace Admin, l'administrateur a accès à une liste de tous les barbershops triés du plus récent au plus ancien.</li> <li>- Il peut valider ou retirer un barbier en un clic. Les barbiers validés seront affichés sur la page d'accueil et sur la page listant tous les barbershops.</li> </ul>

### III. Arborescence du site

L'arborescence d'un site web désigne l'organisation du contenu et des pages d'un site internet ainsi que les liens entre chacune de ces pages. C'est le squelette du site. Pour construire l'arborescence de mon site, j'ai suivi la règle des trois clics, qui est une règle informelle de webdesign : selon elle, un internaute doit pouvoir accéder à n'importe quelle information en 3 clics maximums depuis la page principale. Cela permet de rendre la navigation intuitive et améliore l'expérience utilisateur.



### IV. Cibles

L'application cible deux types de personnes en particulier :

- Les personnes qui se rendent régulièrement chez le barbier pour bénéficier de prestations ou acheter des produits
- Les barbiers qui ne disposent pas de système de réservation, de boutique en ligne, ou simplement les salons souhaitant accroître leur visibilité.

Les barbershops se sont énormément démocratisés ces dernières années, le public aussi bien professionnel que particulier est très large. Peu de salons ont un site vitrine et beaucoup passent simplement par les réseaux sociaux ou par le téléphone pour prendre des réservations, ce qui fait perdre du temps et interrompt les prestations.

Du côté des particuliers, il est maintenant difficile de s'y retrouver parmi toutes les offres plus ou moins qualitatives qui fleurissent rapidement. Certains salons s'attribuent la dénomination de barbier , ou sont catalogués comme tel, alors qu'ils n'ont pas les outils ou le savoir-faire nécessaires.

L'application propose une solution pour aider les consommateurs à faire leur choix, et permettre aux barbiers d'optimiser leur temps et d'accroître leur visibilité.

## V. Design et érgonomie

### 1. Couleurs

Les salons barbiers arborent souvent un style urbain, presque industriel, avec des décos vintage, de gros sièges en cuir, un sol en damier et des *barberpole* (l'emblème des barbiers). Le tout dans une ambiance très américaine. On voit aussi des salons barbiers qui arborent une déco plus sobre et classieuses, moins typée.

J'ai voulu mélanger ces deux ambiances pour définir la charte graphique de mon application, afin qu'elle soit sobre et élégante, tout en gardant quelques touches urbaines pour coller à l'esprit d'origine des salons barbiers américains.

Pour mettre cela en place, j'ai choisi des couleurs très sobres, couplées à une couleur vive pour apporter de la couleur et ne pas tomber dans la monotonie.



#000000

Le noir représente l'élégance, la simplicité, le luxe.



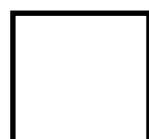
#F5F2F2

Le gris foncé permet d'apporter des nuances entre le noir et le gris clair.



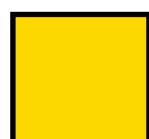
#F4F2F2

Le gris clair évoque la neutralité, la solidité et l'industriel.



#FFFFFF

Le blanc permet d'avoir un design moderne et simple.



#FCD700

Présent par petite touches, le jaune apporte une couleur vive au site et rappelle l'urbanisme.

### 2. Ergonomie

L'application doit être totalement responsive , ce qui signifie qu'elle doit pouvoir être utilisée autant sur un écran ultra-large que sur un téléphone classique, en passant par les tablettes et les moniteurs traditionnels.

Elle comportera beaucoup d'affichages, de formulaires, et de fonctions : si cela n'est pas un problème pour être affiché sur un grand écran, il n'en est pas de même pour un téléphone. Tous les affichages doivent rester lisibles et utilisables facilement sur petit écran.

Elle doit également être simple à utiliser et intuitive. Les fonctionnalités principales doivent être accessibles facilement, et simple à comprendre. L'utilisateur doit retrouver sur l'application les repères qu'il a sur les autres sites web.

## VI. Maquettage

Le maquettage permet de visualiser concrètement l'interface graphique d'un projet digital. Il englobe généralement 3 étapes distinctes : le wireframe, la maquette, et le prototype.

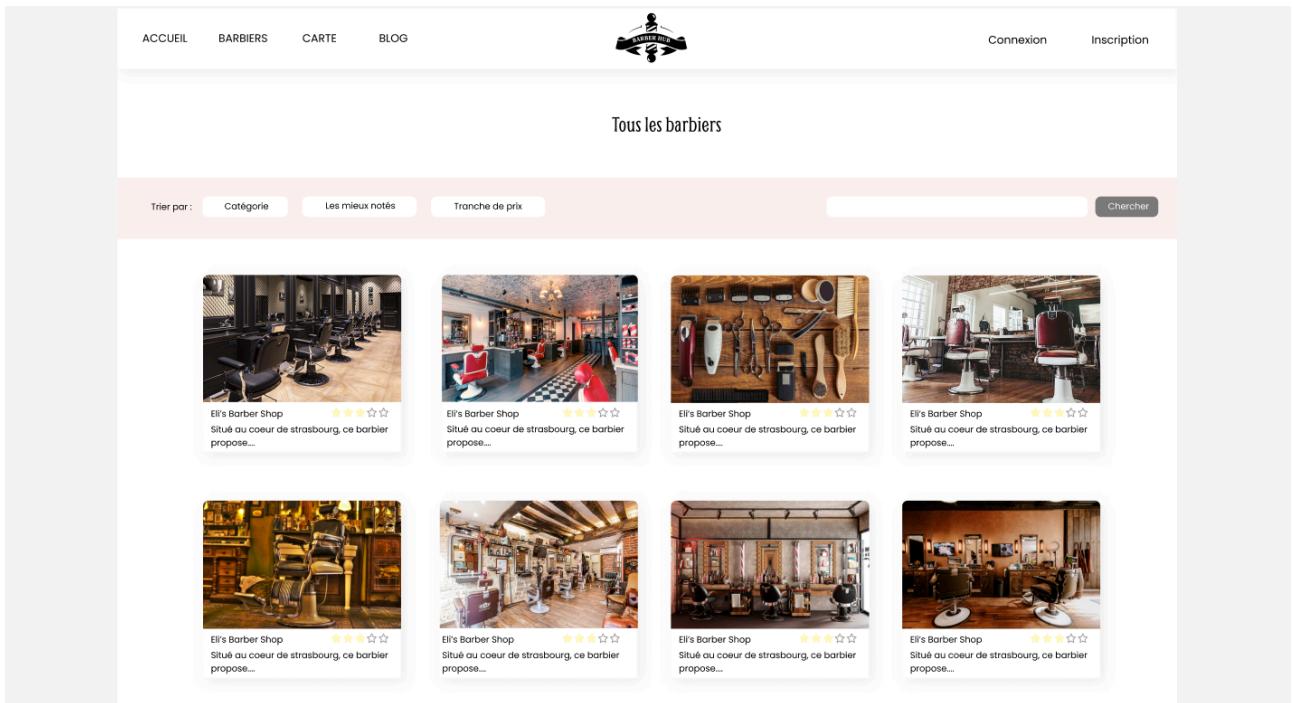
Le wireframe est une représentation basse fidélité du projet. Il représente le squelette du site : les informations à afficher, la hiérarchie de l'information, et une description simple des fonctionnalités prévues.

La maquette (ou mockup) correspond à une représentation haute fidélité de l'interface. Elle reprend les bases du wireframe mais vient ajouter le design, les couleurs, et rentre plus dans le détail. Elle reste tout de même statique, et on aura donc plusieurs pages fidèles au résultat voulu, mais sans possibilité d'interaction.

Le prototype, quant à lui, reprend la maquette et sa représentation haute fidélité, en y ajoutant une dimension dynamique. Il est possible d'interagir avec le prototype : cliquer sur des liens, défiler un carrousel, visualiser des animations...

Il est important lors de cette étape d'appliquer les bonnes pratiques d'UX et d'UI Design. L'UX désigne l'expérience utilisateur : concevoir une interface ergonomique, accessible, et facile à prendre en main. L'UI désigne quant à elle l'interface utilisateur : couleurs, typographies, icônes, qui rendront l'interface séduisante et agréable à utiliser. Ces deux disciplines mises en commun permettent d'obtenir un site ergonomique et visuellement plaisant pour l'utilisateur.

Pour ce projet, j'ai choisi de réaliser un maquettage. Il offre une fidélité plus haute que le wireframe sans prendre le temps que demande un prototypage. J'ai créé dans un premier temps une version desktop (ordinateur de bureau), pour avoir un aperçu concret du design final de mon application et pouvoir être aiguillé dans le développement front-end de mon application.



Maquette desktop de la page recensant tous les barbiers

J'ai choisi de faire un site très épuré, avec des informations et des liens clairs pour que l'expérience utilisateur soit la plus ergonomique possible. Le contenu principal du site est accessible facilement, des icônes permettent d'identifier rapidement les fonctionnalités. Les codes des sites web traditionnels sont repris pour que l'utilisateur retrouve ses habitudes de navigation : un clic sur le logo emmène à la page d'accueil, les fonctionnalités de connexion / inscription sont à droite...

La palette de couleur utilisée, détaillée dans la partie précédente, participe à rendre l'interface utilisateur agréable à l'œil et à identifier facilement les éléments de la page.

J'ai également maquetté la version mobile de l'application pour avoir une idée du rendu final sur téléphone. (voir Annexe 3). Il est important de créer au minimum ces deux maquettes pour les faire concorder et savoir comment les éléments d'une version vont s'adapter sur l'autre.

## VII. Benchmark des concurrents

Afin de constater ce qui existait déjà et ce qu'il était possible de faire, j'ai consulté plusieurs sites proposant des fonctionnalités similaires à mon application. Parmi ceux-ci, les principaux sites dont j'ai tiré mon inspiration sont :

- Planity : ce site auquel j'ai déjà eu recours plusieurs fois propose de découvrir des salons de coiffure de tout types mais aussi des instituts de beauté, tatoueurs et autres , avec des informations, des commentaires, et la possibilité de prendre rendez-vous. Planity est également très axé vers les professionnels, avec des incitations récurrentes à ajouter son établissement et la liste des avantages à le faire. La présentation des salons est très complète et m'a aiguillé sur les informations principales que j'aurai à afficher, ainsi que leur mise en forme.

- Kiute : ce site est spécialisé dans les salons de coiffure de tout type et les prestations de beauté. L'ergonomie ressemble beaucoup à Planity, avec un design plus coloré. J'ai également récupéré quelques éléments de présentation pour la mise en page des détails d'un barbier, le système de carte, ainsi que la gestion de la navigation mobile.

## VIII. Organisation du projet

### 1. Méthodologie

Ce projet a été réalisé en suivant la méthode Agile. Cette dernière préconise la fixation d'objectifs à court terme et la fragmentation du projet en sous partie. Le projet avance ensuite sous forme d'itération : une itération représente la mise en place d'une fonctionnalité. A la fin de chaque itération, j'ai fait le point sur ce que je venais de faire, ce que je devais encore faire, pour fixer de nouveaux objectifs ou ajuster certains éléments, et j'avancais dans la réalisation du projet à l'itération suivante.

Pour permettre une organisation optimale de la méthode Agile, j'ai mis en place la méthode MoSCoW. Cette méthode, créée en 1994, permet de hiérarchiser les tâches à réaliser selon leur ordre d'importance.

L'anagramme MoSCoW signifie :

- M pour *Must have this* ( tâches à faire impérativement )
- S pour *Should have this if at all possible* (activités essentielles à effectuer après avoir fait les tâches impératives )
- C pour *Could have this if it does not affect anything else* ( devrait être fait si cela n'affecte pas les autres taches )
- W pour *Wont have this time but would like in the future* (ne sera pas fait cette fois mais sera fait plus tard )

Pour mettre en place de manière visuelle la méthode MoSCoW, j'ai utilisé la méthode Kanban. Cette méthode préconise l'utilisation d'un outil visuel permettant de suivre les tâches à faire en temps réel, sous forme de tableau, dont les colonnes correspondent aux étapes de travail. Pour cela, j'ai utilisé Trello, outil en ligne très connu qui s'appuie sur la méthode Kanban.

J'ai donc organisé mon espace de travail Trello selon la méthode MoSCoW : j'ai créé 4 étiquettes de différentes couleurs qui correspondent chacun à un niveau hiérarchique MoSCoW. Ainsi, lorsque je crée une nouvelle tache, je peux lui assigner une étiquette selon son importance. Cela me permet d'avoir une vue d'ensemble sur les tâches que je dois réaliser et évaluer rapidement celles qui sont prioritaires et sur lesquelles je dois me concentrer.

J'ai également créé deux autres étiquettes pour me permettre de mieux organiser mes tâches : Tech et User Story.

User Story correspond à la description d'un besoin ou d'une attente d'un utilisateur. Dans mon cas, c'est par exemple pouvoir consulter la liste des barbiers, ou pouvoir prendre rendez-vous.

L'étiquette Tech correspond aux tâches que je dois mettre en place pour que les User Story soient possibles. Ce sont des tâches essentielles mais qui ne seront pas vues par les utilisateurs finaux. Par exemple, mettre en place l'environnement de travail, sécuriser les routes ou encore installer un bundle.

J'ai ainsi pu avoir une vue d'ensemble des tâches que je devais effectuer, classées selon leur importance (*voir Annexe 1*). Cela m'a beaucoup aidé dans l'organisation du projet.

## 2. Le MVP (minimum viable product)

L'acronyme MVP correspond à Minimum Viable Product, produit minimum viable en français. C'est une version de l'application qui rassemble seulement les fonctionnalités élémentaires de l'application.

Produire un MVP plutôt que de directement présenter un produit complet présente plusieurs avantages :

- Le test des fonctionnalités peut se faire directement, et en cas de problème, le code peut être revu plus facilement que sur un projet qui a une dimension plus grande et complexe.
- Agir directement sur des fonctionnalités qui ont été codées récemment, lorsque le code et la logique sont encore frais dans la tête des développeurs, plutôt que de devoir détecter des problèmes qui datent d'il y a plusieurs mois et dans lesquels il est difficile de se replonger.
- Recueillir des avis sur les principales fonctionnalités de l'application assez tôt, et centrés sur l'essentiel.

Dans le cadre de mon application, les fonctionnalités qui constituent le MVP du projet sont les tâches qui correspondent aux étiquettes Must Have de la méthode MoSCoW. On retrouve donc :

Fonctionnalité primaires
- Inscription/Connexion
- Voir la liste des barbiers
- Voir tous les barbiers sur une carte
- Ajouter/Modifier/Supprimer/Valider un barbier
- Ajouter/Modifier/Supprimer un article
- Pouvoir prendre rendez-vous chez un barbier ( une seule prestation )
- Pouvoir utiliser l'application sur un support mobile

## IX. RGPD

Le Règlement Général de Protection des données , dit RGPD (GDPR en anglais) a été adopté par le parlement en 2016 et ses mesures sont entrées en vigueur en 2018.

L'objectif du RGPD est de permettre aux citoyens de vérifier et faire respecter leurs droits sur leurs données personnelles. Parmi ces droits, on retrouve :

- Droit à l'information : comment sont collectées les données, et dans quel but.
- Droit d'accès : savoir quelles données collecte l'entreprise
- Droit d'opposition : pouvoir s'opposer à la collecte de ses données (sauf obligation légale)
- Droit de rectification : pouvoir modifier les données que l'on a transmises à une entreprise
- Droit à l'oubli : pouvoir demander la suppression de ses données personnelles à l'entreprise mais également à ses sous-traitants
- Droit à la portabilité : pouvoir récupérer ses données de façon lisibles.

Mais qu'est ce qu'une donnée personnelle précisément ? Selon la CNIL, une donnée personnelle est « toute information se rapportant à une personne physique identifiée ou identifiable » que celle ci l'identifie directement (nom, prénom) ou indirectement (numéro de téléphone, de carte vitale...)

### **Mesures de mise en conformité mises en place dans l'application :**

J'ai mis plusieurs recommandations de la CNIL en place dans mon application.

Premièrement, l'application respecte le principe de la minimisation des données. Cela consiste à ne récupérer que les données personnelles pertinentes et limitées au strict nécessaire au regard des finalités pour lesquelles elles sont traitées. Je ne récupère donc que l'adresse mail pour permettre l'inscription, et le numéro de téléphone pour la prise de rendez-vous afin de faciliter les contacts avec le barbier en cas d'imprévu. Aucune donnée superflue ou à des fins de traçage n'est collectée.

J'ai également mis en place le droit à l'oubli et à la rectification, en permettant à l'utilisateur de modifier ses données personnelles et de supprimer son compte. À la suite d'une suppression de compte, toutes les données de l'utilisateur sont écrasées de la base de données. S'il a laissé des commentaires, ces derniers sont anonymisés et le pseudo de l'utilisateur est remplacé par « Supprimé ».

On retrouve une Check-Box à l'enregistrement qui permet à l'internaute d'accepter les conditions d'utilisation et la politique de confidentialité du site, ainsi que deux pages qui expliquent ces points en détail.

Le droit à l'information et le droit d'opposition sont assurés par tarteaucitron.js, un gestionnaire de tags RGPD, qui permet d'afficher une bannière sur le site, avec les cookies qui seront utilisés, et la possibilité de les refuser ou de les accepter. Cela permet d'obtenir le consentement de l'utilisateur et de ne rien collecter sans son accord. (*Voir annexe 2*).

Pour finir, une page du site est entièrement consacrée à la politique de confidentialité, qui indique clairement à l'utilisateur quelles données sont collectées et quelle est la finalité de la collecte de ces données, en toute transparence.

## X. Environnement technique

### 1. Logiciels



Le projet a été développé en intégralité sur VsCode, un éditeur de code développé par Microsoft.



Pour réaliser le MCD et le MLD de l'application, j'ai utilisé l'application Looping. C'est un logiciel de modélisation conceptuelle de donnée libre d'utilisation et gratuit créé par des professeurs de l'université de Toulouse III.



Pour réaliser la maquette de l'application, j'ai utilisé l'application Figma. C'est un outil puissant qui permet d'élaborer le design du site mais aussi d'en faire le prototypage : cela permet d'avoir un aperçu du rendu final en situation réelle avec la navigation entre les pages, des menus déroulants...



Enfin, j'ai utilisé Insomnia, une application Open-Source permettant de tester et de débogguer des API. Cela a facilité la mise en place de certaines API au sein de l'application ainsi que la mise en place d'API Platform.

### 2. Technologies utilisées



Pour le développement de l'application, j'ai utilisé Symfony 6.2. C'est un framework PHP open source développé par la société française SensioLab, basé sur le pattern MVP (modèle - vue - présentateur). C'est l'une des structures les plus utilisées dans le développement web, avec une grosse communauté d'utilisateurs. Les modèles gèrent l'accès aux données, les vues sont les interfaces graphiques que voit l'utilisateur et les présentateurs contiennent la logique mise en place à chaque action de l'utilisateur (cf la partie architecture du site).

C'est un framework orienté objet : il est organisé de façon à programmer en manipulant des objets et en les faisant interagir. Un objet est un bloc de code, mêlant des variables et des fonctions, appelés respectivement attributs et méthodes : les attributs définissent les caractéristiques de l'objet tandis que les méthodes définissent les fonctions propres à l'objet.



## Doctrine

J'ai utilisé Doctrine, qui est un ORM (Object Relational Mapper) permettant de manipuler une base de données à travers des objets et non plus des valeurs brutes. Cela permet une meilleure structuration du code et un lien fort entre l'utilisation et le stockage des données. C'est une technologie qui est également utilisée nativement dans Symfony, même si son utilisation n'est pas obligatoire.



## Composer

J'ai également utilisé Composer, qui est gestionnaire de dépendance pour PHP. Il est utilisé pour gérer et intégrer des paquets ou des bibliothèques externes dans un projet.



## Git

Concernant la gestion des versions de mon projet, j'ai utilisé Git, un système de contrôle de version inventé et développé par Linus Tovalds. C'est un outil de développement qui aide les développeurs à suivre les changements apportés au code source au fil des ajouts de fonctionnalité. Il est d'usage sur des projets en équipe d'avoir une branche principale et une ou plusieurs branches de développement où sont testées les nouvelles fonctionnalités. Lorsqu'elles sont validées par les membres de l'équipe, on fusionne les branches. Cela permet d'identifier les bugs sans altérer le code source principal, et évite les régressions. Dans mon cas, j'ai utilisé GIT sans créer de branches car c'est un projet que j'ai mené seul, mais il m'a permis de suivre les différentes version de mon code et les modifications que je lui ai apporté.



## Github

Quand on parle de Git, on parle souvent également des solutions d'hébergement et de gestion de code. Pour ma part, j'ai utilisé Github, qui permet de rendre accessible en ligne le code de l'application, et qui apporte une interface graphique pour visualiser les fonctionnalités proposées par Git.



## PHP My Admin

PhpMyadmin est une application web qui permet de gérer et d'administrer des bases de données MySQL et Maria DB.

### 3. Langages utilisés



#### HTML et CSS

Pour gérer la structure des pages, j'ai utilisé HTML5 (HyperText Markup Language), qui est un langage de balisage permettant au navigateur d'interpréter comment placer les éléments de notre page.

Pour le design des pages, j'ai utilisé CSS3 (Cascading Style Sheets), un langage qui permet d'appliquer une mise en forme, des couleurs, et des effets au contenu.



#### TWIG

Pour gérer l'affichage des vues, j'ai utilisé TWIG, qui est un moteur de templates PHP utilisé nativement par Symfony. C'est un langage simple à comprendre, extensible (possibilité d'ajouter des filtres, des fonctions, des tags...) et également plus concis et clair pour des affichages complexes.



#### Javascript

Javascript est un langage de script très utilisé pour la réalisation d'applications web. Il permet d'implémenter des mécanismes complexes sur une page web : mettre à jour du contenu de façon dynamique, contrôler du contenu multimédia, animer des images... Il peut également être utilisé pour les serveurs web avec Node.js par exemple. Javascript s'allie très bien au HTML, CSS et PHP.



#### PHP 8.2

PHP (pour Hypertext Preprocessor) est un langage de programmation libre créé en 1994 par Rasmus Lerdorf et principalement utilisé pour la création d'applications web dynamiques.



#### SQL

SQL (pour Structured Query Language) est un langage permettant de manipuler les données et les systèmes de base de données relationnelles. Il permet principalement de communiquer avec les bases de données pour gérer les informations qu'elles contiennent.

# XI. Modèles de données

Un modèle de donnée est un modèle qui décrit la structure logique d'une base de données, les relations et les contraintes qui déterminent comment on pourra stocker et accéder aux données.

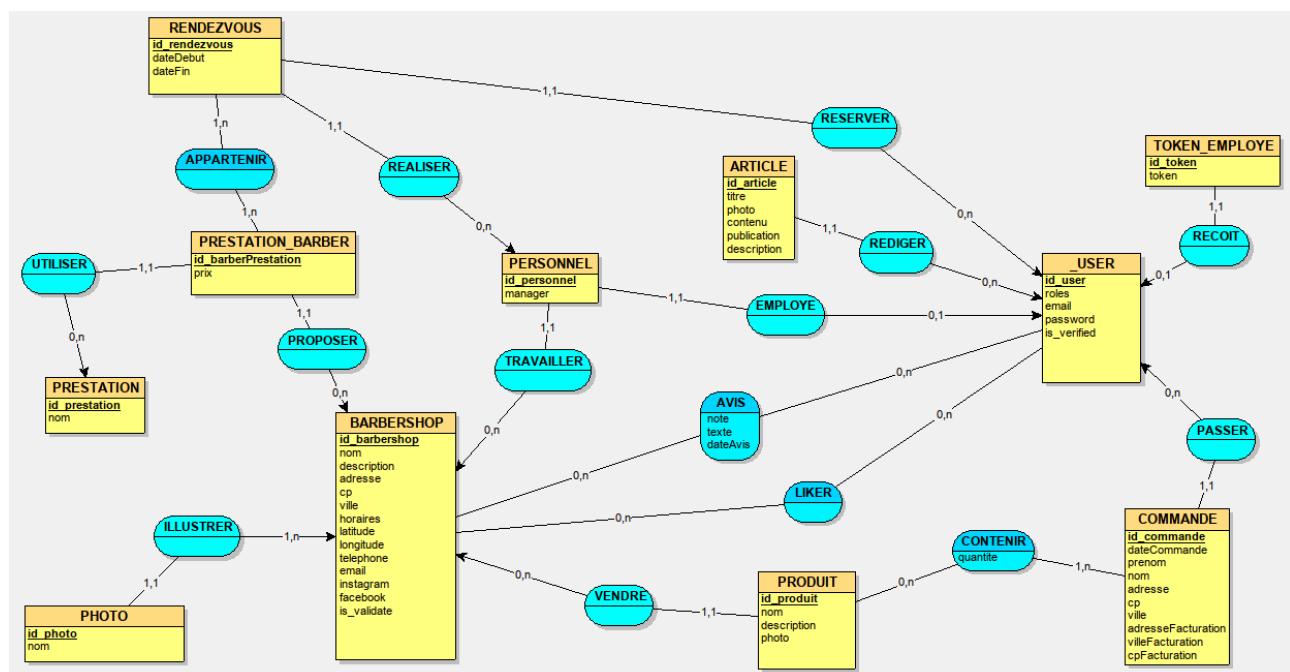
Pour réaliser ce projet, je me suis appuyé sur la méthode Merise. C'est une méthode d'analyse, de conception et de gestion de projet destinée à permettre la création d'un système d'information. Elle a été créée suite à une consultation nationale du ministère de l'industrie en 1977, qui désirait définir une méthode de conception de système d'information.

J'ai utilisé deux outils issus de cette méthode : le MCD (modélisation conceptuelle de donnée) et le MLD (modèle logique de donnée).

## 1. Modèle conceptuel de donnée

La modélisation conceptuelle de données est une représentation graphique d'une base de données, qui permet d'identifier les principales entités qui la constitueront, leurs relations et les attributs qui les composent. Il est abstrait et facilement compréhensible.

J'ai ainsi créé toutes mes entités, représentées par les rectangles jaunes. Chaque entité contient des attributs qui lui sont propres, ainsi qu'une clé primaire, qui permet d'identifier de façon unique chaque enregistrement d'une table : c'est l'id pour la quasi totalité de mes tables. On retrouve par exemple l'entité barbershop, qui est constituée de la clé primaire id\_barbershop et des attributs nom, description, adresse, horaires... On visualise également les relations qui les lient, représentées par les flèches noires. Les relations possibles sont définies en fonction du type d'interaction qu'on trouve entre les tables. Par exemple, un article peut être rédigé par un seul utilisateur, mais un utilisateur peut rédiger aucun ou plusieurs articles : on aura donc une relation 1,1 de article vers user et 0,n de User vers articles.



## 2. Modèle logique de donnée

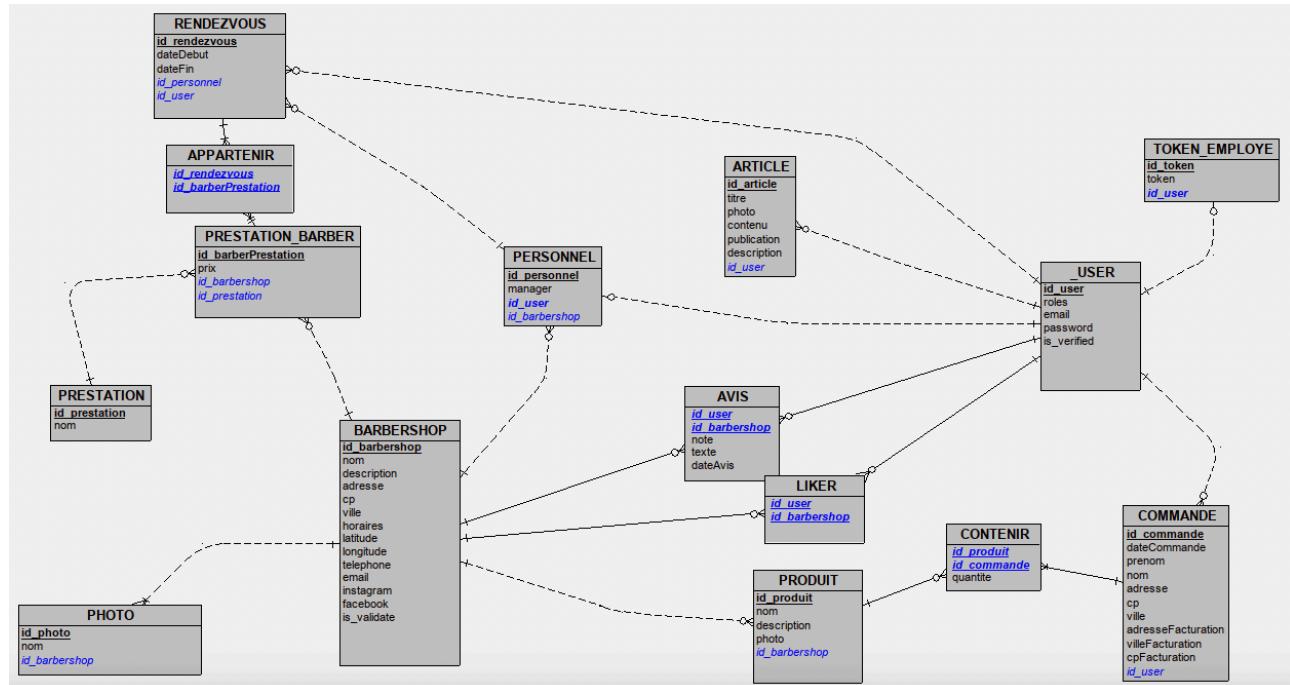
Le modèle logique de donnée est la représentation concrète des tables que l'on va obtenir en fonction des liens que l'on a définis dans le MCD, ainsi que les clés étrangères que l'on aura dans les tables qui sont en relation. Il est indépendant d'un langage de programmation et présente un niveau d'abstraction en dessous du MCD.

On peut ainsi observer l'apparition de clés étrangères dans certaines table possédant une relation avec une autre table. Une clé étrangère garantit l'intégrité référentielle entre deux tables et identifie une colonne ou un ensemble de colonnes d'une table comme référençant une colonne ou un ensemble de colonne d'une autre table.

Ainsi, la table Avis par exemple, récupère la clé étrangère id\_user et id\_barbershop, car chaque Avis est relié à un utilisateur et un barbershop (lien 1,1 vers user et lien 0,n vers barbershop). L'attribution de ces clés est définie selon les liens qui ont été tissés dans le MCD. La table Avis contiendra donc pour chaque avis, une clé étrangère user\_id et une clé étrangère barbershop\_id, qui feront référence à l'id de l'utilisateur qui a laissé le commentaire, et à l'id du barbershop sur lequel le commentaire a été laissé.

On obtient aussi, en fonction des relations, des tables associatives : ce sont les entités représentées par les ovales bleus dans le MCD. Ces dernières sont des tables à part entière dans le MLD. Certaines de ces tables contiennent des attributs en plus, tel que Avis, ou Contenir.

Le modèle logique de données se rapproche de l'apparence et de la structure finale de la base de données.



# C. Développement du Projet

## I. Architecture du projet

Comme nous l'avons vu précédemment, ce projet a été entièrement développé avec le framework Symfony. La structure du projet respecte donc la structure de Symfony, qui est basée sur le design Pattern MVP.

Commençons par définir ce qu'est un design pattern. Les développeurs, au fur et à mesure de la réalisation de projets, se sont retrouvés face à des problèmes de conception récurrents et similaires. Avec le temps, ils ont conceptualisé des solutions pour traiter au mieux ces problèmes et organiser une application. C'est ce qu'on appelle aujourd'hui des design patterns : ce sont des solutions éprouvées et fiables pour résoudre une problématique récurrente.

Ils présentent plusieurs avantages :

- Une maintenabilité accrue
- Une structure de code propre et organisée
- La possibilité de travailler en équipe plus facilement
- Un gain de temps considérable

Il existe une multitude de design pattern : on peut aussi citer Singleton ou encore Factory. Symfony utilise le design pattern MVP : modèle - vue - présentateur.

Ce modèle divise les responsabilités du système en 3 parties distinctes :

### **LA VUE :**

Elle correspond à l'interface graphique avec laquelle l'utilisateur va interagir. Elle présente les données envoyées par le présentateur. Elle n'effectue quasiment aucun traitement, si ce n'est quelques tâches légères. C'est aussi elle qui reçoit les interactions de l'utilisateur : clics, formulaires, bouton.. et qui envoie ces interactions au présentateur. Elle n'a pas de contact direct avec la couche modèle : les données passent systématiquement par le présentateur.

Les vues se trouvent dans le dossier templates du projet. Pour styliser ces vues, on a également un dossier « public », qui contient le CSS permettant la mise en forme des différents éléments.

Sur l'exemple ci-dessous, on retrouve un morceau de vue comportant un léger traitement permettant d'afficher la confirmation d'un rendez-vous :

```

19 Date : {{lastRDV.debut|date("d/m/Y")}} <br />
20 Heure : {{lastRDV.debut|date("H:i")}} - {{lastRDV.fin | date("H:i")}} <br />
21 {% for prestation in lastRDV.barberprestation %}
22 |   Prestation : {{prestation.prestation.nom}} {{prestation.prix}}€ <br />
23 {% endfor %}
24 |
25 Avec : {{lastRDV.personnel.user.pseudo}}
26
27
28 <p><a href="{{path('app_home')}}">Revenir à l'accueil</a> - <a href="{{path('app_myspace')}}">Voir mes rendez-vous</a></p>

```

Affichage des détails d'un rendez-vous dans une vue.

On peut voir des affichages de valeur dynamique, comme la date et l'heure du rendez-vous (ligne 19 et 20).

On retrouve également une boucle for, qui permet de parcourir le tableau lastRDV et d'afficher les prestations choisies. Le moteur de template Twig est utilisé dans les vues.

## LE PRESENTATEUR :

C'est la couche qui prend en charge les requêtes de l'utilisateur et qui met à jour la vue ou le modèle en conséquence. A la réception d'un événement, il enclenche les actions appropriées. Si une action nécessite un changement de données, il sollicite la couche modèle, récupère et formate (voire re-contrôle) les données, puis met à jour la vue. Si l'action ne nécessite pas de changement de données, il modifie seulement la vue. C'est la partie qui contient la logique, les calculs et les décisions : il peut comporter des algorithmes plus ou moins complexes, en fonction des actions à effectuer, répartis dans des classes distinctes.

Une classe est un ensemble de code, contenant des variables et des fonctions, permettant de créer des objets.

En bref, le présentateur joue le rôle de chef d'orchestre.

Pour reprendre l'exemple ci dessus, si l'utilisateur clique sur un des liens situés ligne 28, la fonction du présentateur correspondant à la route choisie sera appelée. Dans le cas d'un clic sur « revenir à l'accueil », c'est la fonction correspondant à la route app\_home qui est sollicitée.

Voici la logique mise en place par cette fonction :

```

15 // Home
16 #[Route('/', name: 'app_home')]
17 public function index(ManagerRegistry $doctrine): Response
18 {
19     // Récuperer les 3 derniers barbershops de la BDD
20     $lastBarbershops = $doctrine->getRepository(Barbershop::Class)->getLastThreeValidBarbershop();
21     //Récuperer les 2 derniers articles
22     $lastArticles = $doctrine->getRepository(Article::Class)->findBy([], ["date"=>"DESC"] , 2);
23     $isHomePage = true;
24     return $this->render('home/index.html.twig', [
25         'lastBarbershops' => $lastBarbershops,
26         'lastArticles' => $lastArticles,
27         'isHomePage' => $isHomePage
28     ]);
29 }

```

Fonction index dans le HomeController

Elle prend une instance de doctrine ManagerRegistry en argument, ce qui va nous permettre d'interagir avec la base de données grâce à la couche modèle.

On l'utilise à la ligne 20 et 22, pour récupérer les informations dont on a besoin. Doctrine passe par le Repository de la classe voulue et utilise les méthodes présentes dans ce dernier pour effectuer une requête vers la base de données puis renvoyer les informations demandées.

A la ligne 24, on indique vers quelle vue doivent être retournées les informations, et on passe les données que l'on a récupéré en paramètre pour pouvoir les afficher dans la vue.

## LE MODÈLE :

Il gère la logique métier du site. Cette couche contient des méthodes qui permettent d'interagir avec la base de données : sélectionner, supprimer, créer, mettre à jour... Elle assure la gestion de ces données et garantie leur intégrité.

Dans Symfony, c'est Doctrine qui s'occupe de cette partie. Comme on l'a vu, Doctrine est un ORM : object relationnal maper. Il permet de convertir les requêtes brutes SQL en requêtes DQL (Doctrine Query Language), qui permettent de récupérer des objets.

Les méthodes utilisées dans le présentateur pour interagir avec la base de données sont situées dans les repository ainsi que dans les entités. Dans les entités, on retrouve des méthodes qui permettent de récupérer ou de modifier chaque attribut de la classe : ce sont les getters et les setters.

Dans les Repository on retrouve des méthodes présentes par défaut, comme findBy ou findAll, qui sont incluses nativement avec Symfony et qui permettent de récupérer simplement des informations.

Ces méthodes sont génériques et ne permettent pas de faire des requêtes précises. Pour cela, on peut créer des méthodes personnalisées dans les Repository, à l'aide de requêtes DQL.

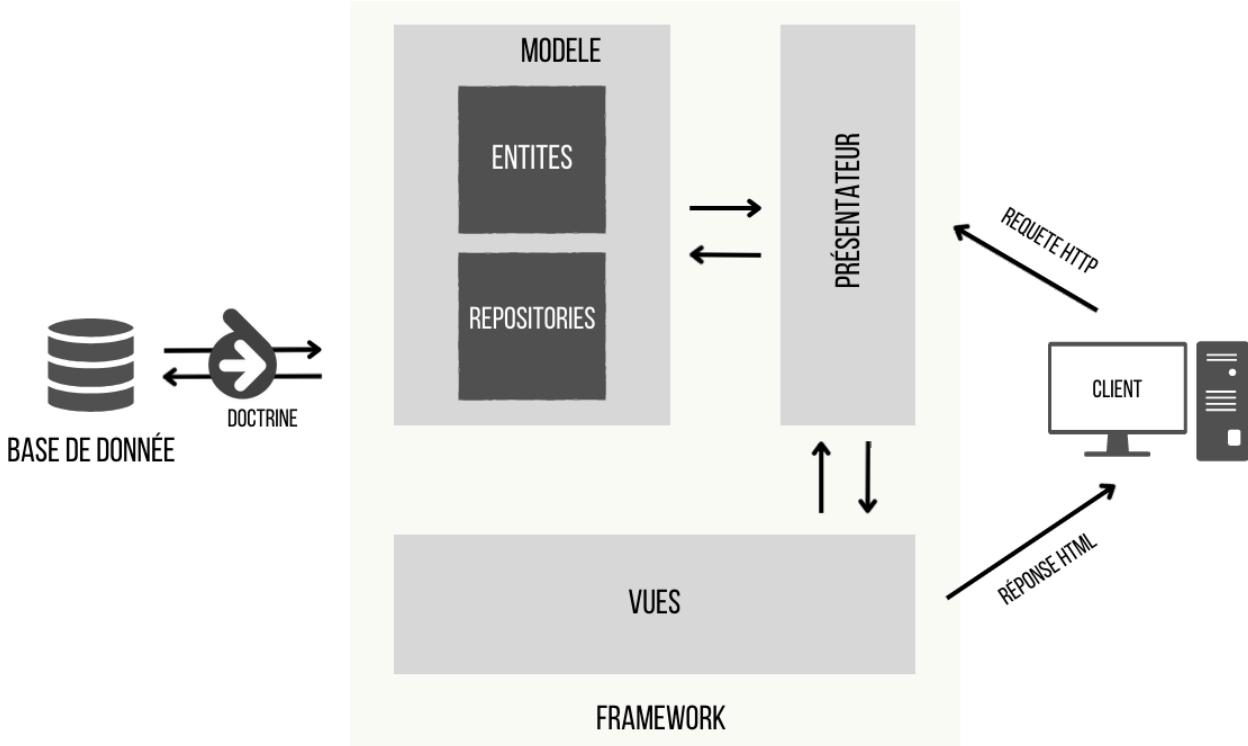
Voilà par exemple la requête getLastThreeValidBarbershop qui est utilisée dans le HomeController :

```
42     public function getLastThreeValidBarbershop()
43     {
44         $em = $this->getEntityManager();
45         $qb = $em->createQueryBuilder();
46         $query =
47             $qb->select('b')
48             ->from('App\Entity\Barbershop', 'b')
49             ->where('b.validate = true')
50             ->orderBy('b.creationDate', 'DESC')
51             ->setMaxResults(3)
52             ->getQuery();
53
54         return $query->getResult();
55     }
```

Requête DQL dans le Barbershop Repository

Cette fonction crée une requête DQL qui va chercher les 3 derniers barbershops ayant l'attribut validate égal à True, et renvoie les résultats obtenus.

Pour résumer, on peut schématiser le design pattern MVP comme ceci :



Le client, logiciel permettant d'effectuer des requêtes (un navigateur par exemple), effectue une requête HTTP depuis son ordinateur vers le serveur, qui est lui un logiciel permettant de répondre à une requête HTTP en renvoyant des données. Une requête HTTP est un protocole de communication réseau qui utilise notamment des verbes HTTP pour indiquer au serveur les actions à effectuer : par exemple, GET servira à récupérer une information, POST à changer une information. Cette requête est interceptée par le présentateur qui va faire les actions nécessaires en faisant appel si besoin à la base de données grâce à la couche modèle, aux fonctions présentes dans les entités et les Repositories et à Doctrine, puis mettre à jour la vue. Cette dernière est rendue à l'utilisateur via une réponse HTML.

## II. Fonctionnalité représentative

Une des fonctionnalités importante de cette application est la gestion des rendez-vous. Au-delà de pouvoir consulter les barbershops d'une ville, l'utilisateur doit pouvoir prendre rendez-vous dans le salon qu'il souhaite, et le barbier doit pouvoir afficher son planning et générer une facture pour un rendez-vous.

### 1. Structure de la base de données

La première étape a été une étape de réflexion. Même si mon MCD avait été préparé en amont pour avoir une vision globale de la base de données de mon application, la prise de rendez-vous et l'affichage d'un planning

posent certaines problématiques auxquelles je n'avais pas pensé. J'ai dû réfléchir à la structure qui correspondrait le mieux à mes attentes. Pourra-t-on réserver plusieurs prestations dans un même rendez-vous ? Peut-on choisir la personne qui va nous faire la coupe ? Combien de temps vont durer les rendez-vous ? Comment récupérer toutes les informations dont j'ai besoin pour afficher les détails d'un rendez-vous ?

Une multitude de questions se posait, et les tables ainsi que les relations qui les unissent étaient déterminantes pour mener à bien l'implémentation de cette fonctionnalité.

J'ai décidé de donner la possibilité à l'utilisateur de choisir la personne qui allait effectuer la prestation, ce qui impliquait la création d'une table Personnel.

J'ai également décidé de permettre de réserver plusieurs prestations pour un même rendez-vous : même si ce n'est pas en place à l'heure actuelle, j'ai conçu cette application pour qu'elle soit évolutive et la base de données permet de gérer ce cas de figure via la table associative rendezvous\_barberprestation.

Enfin, par souci de simplicité et afin d'obtenir un MVP rapidement, j'ai décidé de définir la durée d'un créneau de rendez-vous à une durée fixe de 30 minutes. Sans durée de créneaux fixe, le calcul des créneaux disponibles devient un véritable casse-tête, surtout lorsque l'on ajoute plusieurs prestations à la suite.

Après avoir supprimé, modifié, et créé plusieurs tables, ma base de données était enfin prête et j'ai pu commencer à développer.

## 2. Mise en place de la prise de rendez-vous

Pour pouvoir afficher des rendez-vous, il faut déjà pouvoir les prendre. C'est donc la première étape sur laquelle je me suis penché. Afin d'avancer, je devais tout d'abord définir de quelle façon j'allais présenter la prise de rendez-vous. La sélection d'un créneau parmi des centaines d'options proposées peut vite être contre-intuitive pour l'utilisateur, il me fallait agencer cela de manière simple et ergonomique.

La première approche pour laquelle j'ai opté consistait à la sélection d'un jour dans un premier temps, puis un champ de sélection apparaissait, permettant de choisir un créneau.

Mais cette méthode présentait plusieurs inconvénients : premièrement, elle était peu pratique pour l'utilisateur, qui devait fournir plusieurs clics pour accéder aux créneaux disponibles. Ensuite, l'affichage d'un calendrier entier était peu judicieux : les rendez-vous chez le coiffeur sont rarement pris 6 mois à l'avance. De plus, les créneaux indisponibles n'étaient gérés qu'une fois le bouton de réservation cliqué, ce qui était très peu ergonomique.

J'ai donc changé d'approche : j'ai opté pour un affichage qui présente uniquement les créneaux disponibles en fonction du professionnel choisi.

Pour cela, mon objectif est de réunir, dans un tableau PHP, tous les créneaux disponibles en fonction du barbier choisi, pour ensuite les afficher dans une vue.

Les créneaux disponibles excluent donc :

- Les créneaux déjà réservés

- Les créneaux en dehors des heures d'ouverture
- Les créneaux avant l'heure actuelle

Je commence par créer un formulaire RendezVousType qui contiendra tous les champs nécessaires à la prise de rendez-vous : début, nom, prénom, téléphone, personnel. Il ne comporte pas de champ « fin » et le champ « début » est en HiddenType, c'est à dire qu'il est caché.

```
-- 26 |     →add('debut', HiddenType::class, [
27 |         |     'mapped' => false,
28 |     ])

```

En effet, l'utilisateur cliquera sur un créneau et la valeur sera directement injectée dans le champ « début ». La date de fin quant à elle sera automatiquement calculé à partir de la date de début, car les créneaux font tous 30 minutes.

J'affiche ensuite dans ma vue le formulaire précédemment créé. Cette boucle TWIG me permet d'afficher les boutons radio qui permettront à l'utilisateur de choisir le barbier qu'il désire. J'utilise form\_widget et form\_label pour n'afficher que l'input et le label du champ. Cela me permet de les encapsuler dans une div et de faire le style graphique plus facilement.

```
22 <div id="personnel-radio-container">
23   {% for personnelCheckbox in formAddRendezVous.personnel %}
24     <div class="personnel-radio">
25       {{ form_widget(personnelCheckbox) }}
26       {{ form_label(personnelCheckbox) }}
27     </div>
28   {% endfor %}
29 </div>
```

J'initialise une section creneauxContainer avec une structure de carroussel Splide à l'intérieur. Splide est une bibliothèque Javascript open source permettant de mettre en place simplement des carrousels d'images ou de contenu. C'est la div splid\_list qui contiendra nos créneaux.

```
48 <section id="creneauxContainer">
49   <div class="splide" aria-label="Splide Basic HTML Example">
50     <div class="splide__track">
51       <div class="splide__list">
52
53       </div>
54     </div>
55   </div>
56 </section>
```

Les boutons radio sont reliées à un écouteur d'évènement qui envoie une requête AJAX ( pour Asynchronous Javascript + XML, ce type de requête permet de réaliser des mises à jour rapides d'une page web sans avoir besoin de recharger la page entière) avec l'id du personnel choisi (récupéré ligne 19) vers la fonction getCreneauxByPersonnel(). C'est grâce à cet id qu'on va pouvoir récupérer les créneaux disponibles pour chaque barbier.

```

14 // On écoute le clic des input radio
15 $(document).on('change', 'input[type=radio]', function() {
16     // Si c'est checked
17     if (this.checked) {
18         // On récupere la value de personnel ID
19         var personnel_id = $(this).attr('value');
20
21         var data = {
22             personnel_id: personnel_id
23         };
24
25         $.ajax({
26             type: "POST",
27             url: "/getCreneauxbyPersonnel/" + personnel_id,
28             data: JSON.stringify(data),
29             contentType: 'application/json',

```

Voyons maintenant ce qu'il se passe dans le RendezVousController pour générer les créneaux :

Je commence par récupérer le contenu de la requête AJAX sous forme de tableau PHP (ligne 201 de la capture d'écran ci-dessous). Je récupère ensuite la clé personnel\_id de ce tableau, qui correspond donc à l'id de l'employé choisi par l'utilisateur.

A partir de cela, je récupère l'objet Personnel correspondant à l'id récupéré, via le PersonnelRepository. La couche modèle va donc envoyer une requête qui va chercher le personnel correspondant à l'ID renseigné et renvoyer un objet Personnel le contenant.

Cet objet va me permettre d'obtenir, via les méthodes présentes dans l'entité personnel, l'objet barbershop où travaille l'employé, ainsi que tous ses rendez-vous à venir. On récupère aussi les horaires du barbershop à partir de son objet.

```

200 // On récupère les données envoyée par la requête sous forme d'array
201 $data = json_decode($request→getContent(), true);
202 // Dans le tableau data on récupère personnel ID
203 $personnelId = $data['personnel_id'];
204
205 // On récupère l'employé qui corresponds au personnel ID récupéré
206 $repository = $entityManager→getRepository(Personnel::class);
207 $personnel = $repository→find($personnelId);
208
209 // On récupère le barbier et ses horaires
210 $barbershop = $personnel→getBarbershop();
211 $horaires = json_decode($barbershop→getHoraires(), true);
212
213 // On récupère les rendez vous déjà réservés de l'employé
214 $rdvsPersonnel = $personnel→getRendezvous();

```

On peut déjà stocker tous les rendez-vous du barbier dans un tableau unavailableRdvs, qui contiendra tous les créneaux indisponibles. Il suffit de boucler sur le tableau d'objet de rendez-vous de l'employé et d'ajouter chaque date de début dans le tableau unavailableRdvs.

```

216     // On stocke tous les rdv déjà pris dans un tableau
217     $unavailableRdvs = [];
218     foreach($rdvsPersonnel as $rdvPersonnel){
219         $unavailableRdvs[] = $rdvPersonnel->getDebut();
220     }

```

On peut maintenant ajouter à ce tableau unavailableRdvs tous les créneaux avant l'heure actuelle. Pour cela on commence par définir l'heure actuelle, qui est un objet DateTime auquel je rajoute 2h pour obtenir un fuseau horaire français UTC+2. On récupère ensuite le jour actuel, à partir de l'heure actuelle, et la date de fermeture et d'ouverture du barbier, grâce aux horaires que l'on avait récupérés précédemment. Les horaires étant stockés sous forme de JSON en base de données, j'utilise la clé « ouverture » et « fermeture » pour accéder aux valeurs associées. JSON (JavaScript Object Notation) est un format d'échange se présentant sous forme de texte léger et lisible facilement.

```

221     // On ajoute au tableau des RDV déjà pris tous les créneaux avant l'heure actuelle
222     $currentHour = new \DateTime('now +2hours');
223
224     $currentDay = strtolower(strftime('%A', $currentHour->getTimestamp()));
225     $startCurrentDayHour = $horaires[$currentDay]['ouverture'];
226     $closeCurrentDayHour = $horaires[$currentDay]['fermeture'];
227

```

Si l'heure d'ouverture ou l'heure de fermeture sont différentes de « Fermé », je récupère ces heures en objet DateTime pour pouvoir mettre des conditions sur la boucle while qui suit. Cela permet d'éviter d'avoir une erreur en bouclant sur un jour fermé car on ne pourrait pas récupérer d'objet DateTime sur une chaîne de caractères.

```

228     if($startCurrentDayHour != 'ferme' || $closeCurrentDayHour != 'ferme'){
229
230         $startHourDateTime = DateTime::createFromFormat('H:i', $startCurrentDayHour);
231         $closeHourDateTime = DateTime::createFromFormat('H:i', $closeCurrentDayHour);
232
233
234         while($startHourDateTime <= $currentHour && $startHourDateTime <= $closeHourDateTime){
235
236             $unavailableRdvs[] = clone $startHourDateTime;
237             $startHourDateTime->modify('+30 minutes');
238
239         }
240     }
241

```

Ainsi, tant que la date d'ouverture est inférieure ou égale à la date actuelle et que la date d'ouverture est supérieure ou égale à la date de fermeture, j'ajoute des créneaux de 30 minutes dans le tableau UnavailableRdvs.

Si l'heure d'ouverture est 8h et que l'heure actuelle est 13h, tous les créneaux de 8h à 13h seront ajoutés dans le tableau afin de les rendre indisponibles.

Notre tableau de créneaux indisponibles est maintenant complet. Il contient tous les rendez-vous des barbiers et tous les créneaux qui sont indisponibles avant l'heure actuelle.

On génère ensuite des créneaux horaires de 30 minutes sur une plage de temps donnée, à partir des horaires d'ouverture et de fermeture du barbier. Ces créneaux ne prennent pas encore en compte les créneaux indisponibles.

Je commence par définir la plage de temps désirée. Dans mon cas, je propose une plage de réservation d'un mois après la date actuelle. C'est ce que j'ai généralement constaté sur les sites de réservation de prestation que j'ai consultés. Pour étendre ou raccourcir ces créneaux, il suffit de changer la valeur « +1month » qu'on passe en paramètre de la méthode modify sur l'objet DateTime (ligne 245).

```
243 | //Plage de temps pour generer les creneaux
244 | $todayDate = new DateTime('now');
245 | $endDate = (new DateTime('now'))->modify('+1 month');
```

J'initialise ensuite le tableau qui va contenir tous les créneaux (ligne 248), puis je commence une boucle while qui fera des itérations tant que la date du jour actuel \$todayDate est plus petite que la date de fin de la période \$endDate.

```
247 | // On génère un tableau avec tous les créneaux de RDV pour le mois à venir
248 | $allCreneaux = [];
249 | while($todayDate <= $endDate){
250 |     // On récupère le jour de la semaine
251 |     $jourDeLaSemaine = strtolower(strftime('%A', $todayDate->getTimestamp()));
252 |
253 |     // On récupère les heures d'ouverture et de fermeture du barbier
254 |     $heureOuvertureBarber = $horaires[$jourDeLaSemaine]['ouverture'];
255 |     $heureFermetureBarber = $horaires[$jourDeLaSemaine]['fermeture'];
256 | }
```

A chaque itération de la boucle, je dois récupérer le jour de la semaine actuel pour pouvoir obtenir les horaires du barbershop ce jour-là. Pour ce faire, je passe \$todayDate en timestamp UNIX. Ce dernier correspond au nombre de secondes écoulées depuis le 01 Janvier 1970 à 00h00h00, ce qui permet de faciliter les calculs de temps en utilisant un format unique. J'utilise ensuite la méthode strftime() pour récupérer le jour de la semaine en lui passant le format '%A' qui corresponds au jour, et le timestamp du jour actuel.

Je peux par la suite récupérer l'heure d'ouverture et de fermeture du barbershop.

Ensuite, je mets en place une condition : si l'heure de fermeture ou d'ouverture du barbier est différente de « fermé », je récupère ces derniers en format DateTime. Cela permet de ne pas prendre en compte les créneaux se trouvant dans des jours de fermeture. Le format DateTime permet quant à lui d'avoir une date complète pour chaque jour et pas seulement la chaîne de caractères stockée en base de données.

```
257 | // Si le barbier est fermé on n'ajoute rien au tableau de créneau
258 | if($heureOuvertureBarber != 'ferme' || $heureFermetureBarber != 'ferme'){
259 |     $heureOuvertureObj = DateTime::createFromFormat('H:i', $heureOuvertureBarber);
260 |     $heureFermetureObj = DateTime::createFromFormat('H:i', $heureFermetureBarber);
```

J'initialise ensuite un nouveau tableau \$creneaux, qui contiendra les créneaux pour chaque jour, puis une deuxième boucle while, qui s'exécutera tant que l'heure d'ouverture est plus petite que l'heure de fermeture.

```

262     $creneaux = [];
263
264     while($heureOuvertureObj <= $heureFermetureObj){  

265
266         $creneauDateTime = clone $todayDate;  

267         $creneauDateTime->setTime(  

268             $heureOuvertureObj->format('H'),  

269             $heureOuvertureObj->format('i')  

270         );  

271
272         $creneaux[] = $creneauDateTime;  

273         $heureOuvertureObj->modify('+30 minutes');  

274     }

```

Ainsi, chaque itération de la boucle principale crée un jour puis cette deuxième boucle vient créer tous les créneaux \$creneauxDateTime (ligne 266) pouvant être contenus entre la date d'ouverture et de fermeture.

Chaque créneau est mis au format DateTime, et ajouté au tableau contenant tous les créneaux de la journée. J'ajoute ensuite 30min à l'heure d'ouverture, car j'ai défini en amont que chaque prestation durerait 30 min, je veux donc des créneaux toutes les 30 minutes. Cette ligne est particulièrement importante car elle permet d'éviter une boucle infinie : si l'heure d'ouverture n'est pas incrémentée, elle sera toujours plus petite que l'heure de fermeture et la boucle ne s'arrêtera jamais car la condition sera toujours vraie.

Une fois que tous les créneaux ont été générés, j'ajoute chaque tableau \$creneaux dans le tableau principal \$allCréneaux. Ce tableau est un tableau à deux dimensions qui contient en clé la date du jour et en valeur le tableau de créneaux.

```
276     $allCréneaux[$todayDate->format('Y-m-d')] = $creneaux;
```

A la fin de ces opérations, je rajoute un jour à la date actuelle pour passer au jour suivant et répéter la même opération jusqu'à ce que la date du jour soit supérieure ou égale à la date de fin.

```
278     $todayDate->modify('+1 day');
```

Nous avons donc un tableau contenant tous les rendez-vous indisponibles, et un autre tableau contenant des créneaux de rendez-vous pour un mois.

Les deux contiennent des objets DateTime formatés de la même manière. La dernière étape est donc d'enlever du tableau \$allCréneaux, tous les créneaux indisponibles présents dans \$unavailableRdv.

Pour cela, je boucle sur chaque rendez-vous présent dans le tableau \$unavailableRdv (ligne 282), puis je fais une seconde boucle sur chaque tableau de créneaux présent dans le tableau allCréneaux, avec pour clé la date et comme valeur \$dayCréneaux (ligne 284).

```

281     // On retire du tableau les rendez-vous déjà réservés
282     foreach($unavailableRdvs as $unavailableRdv){
283
284         foreach ($allCreneaux as $date => $dayCreneaux) {
285             // On regarde si il y a une clé ou unavailableRdv et DayCreneau sont identiques
286             $key = array_search($unavailableRdv, $dayCreneaux);
287             // Si la clé existe, on enlève le créneau
288             if ($key != false) {
289                 unset($allCreneaux[$date][$key]);
290             }
291         }
292     }
293 }
```

Ainsi, pour chaque créneau, je vais utiliser la méthode PHP `array_search` (ligne 286), qui prend en paramètre un tableau et une valeur. Cette fonction retourne la clé de la valeur qui correspond à la valeur recherchée dans le tableau ciblé.

Je stocke le retour de la fonction dans la variable `$key`. Je fais ensuite une condition pour vérifier si `$key` a une valeur différente de `false`. Si c'est le cas, c'est qu'une clé a été retournée et donc que le créneau recherché a été trouvé.

Il ne me reste plus qu'à retirer du tableau `$allCreneaux` la clé qui correspond au créneau indisponible avec la méthode `unset` de PHP.

Par souci d'esthétisme lors de l'affichage des créneaux, je retire aussi toutes les clés du tableau `$allCreneaux` qui sont associées à un tableau de créneaux vide, pour ne pas se retrouver avec des jours n'affichant aucun créneau dans la vue.

```

295     // Si un tableau de créneaux est vide, on l'enlève
296     foreach ($allCreneaux as $date => $dayCreneaux) {
297
298         if (empty($dayCreneaux)) {
299
300             unset($allCreneaux[$date]);
301
302         }
303     }
```

Pour cela, j'applique à peu de chose près le même principe qu'au dessus. Je boucle sur le tableau contenant tous les créneaux et je teste les tableaux de créneaux de chaque jour. Je vérifie si ce tableau est vide avec la méthode `empty()`, et si c'est le cas, je le retire du tableau `$allCreneaux`.

Le tableau contenant tous les créneaux est maintenant épuré de tous les rendez-vous réservés, de tous les créneaux situés en dehors des heures d'ouverture, de tous les créneaux avant l'heure d'ouverture, et il ne contient pas de tableau de créneaux vide. Je peux donc passer ce tableau final dans un fragment de vue qui servira à afficher tous les créneaux.

```
304     return $this->render('rendezvous/_creneauxbyBarber.html.twig', ['allCreneaux' => $allCreneaux]);
```

Dans ce fragment de vue, je fais simplement une boucle sur le tableau `allCreneaux` pour afficher les informations nécessaires à l'utilisateur et pouvoir récupérer la valeur sélectionnée par celui-ci.

```

2  {% for date, creneaux in allCreneaux %}
3  <div class="splide__slide date-creneaux-container">
4    <h3>{{ date|format_datetime(pattern="EEEE d LLL", locale='fr')|capitalize }}</h3>
5    <div class="creneaux-container">
6      {% for creneau in creneaux %}
7        {% set creneauValue = creneau.format('y-m-d H:i:s') %}
8        <button class="creneau-button" onclick="selectedCreneau(this,'{{ creneauValue }}')">{{ creneau.format('H:i') }}
9      {% endfor %}
10     </div>
11   </div>
12 {% endfor %}

```

On aura ensuite deux possibilités :

```

30 |           success: function(response) {
31 |             console.log(response);
32 |             // Ajout des créneaux au container
33 |             var creneauxContainer = $(".splide__list");
34 |             creneauxContainer.empty();
35 |             creneauxContainer.append(response);
36 |             // Initialisation du slider
37 |             initSplide();
38 |             // On enlève le bouton reservation car aucun créneau sélectionné
39 |             var reservationButton = document.getElementById('rendez_vous_submit');
40 |             reservationButton.style.display = 'none';
41 |           },
42 |           error: function(xhr, status, error) {
43 |             console.error(error);

```

Si la requête AJAX déclenché par le clic sur un des boutons radio est un succès, on ajoute ce qu'elle a renvoyé, c'est à dire le HTML du fragment de vue rempli avec les créneaux correspondant à l'employé sélectionné, à la div chargée de contenir nos créneaux (ligne 35).

Sinon, on affiche une erreur dans la console.

Maintenant que les créneaux sont générés et affichés (voir Annexe 4), il faut mettre la logique en place lors de la sélection d'un créneau par l'utilisateur.

Chaque créneau est un bouton, qui fait appel à la fonction selectedCreneau() au clic.

```

52  function selectedCreneau(button, value) {
53    event.preventDefault();
54    document.getElementById('rendez_vous_debut').value = value;

```

Cette fonction récupère le bouton cliqué et la valeur de celui-ci. Un event.preventDefault, permet d'empêcher le comportement de base du bouton, c'est à dire l'envoi du formulaire. Le champ caché « debut » est alors alimenté par la valeur sélectionnée (ligne54).

Récupérer le bouton cliqué sert seulement à lui appliquer des effets de style afin de rendre l'expérience intuitive pour l'utilisateur. Si un bouton est cliqué, il prendra une apparence différente des autres et le bouton de réservation apparaitra.

```

61  // Ajouter la classe "clicked-button" au bouton cliqué
62  var reservationButton = document.getElementById('rendez_vous_submit');
63  button.classList.add('clicked-button');
64  reservationButton.style.display = 'block';

```

Il faut maintenant mettre en place la logique qui permettra d'ajouter les rendez-vous en base de données lorsque l'utilisateur cliquera sur le bouton réserver.

Ce sera le rôle de la fonction addRDV.

```
39 | #[Route('/barbershop/{id}/rendezvous/{barberPrestation}/add', name: 'add_rendezvous')]
```

Elle prend dans son URL un objet barbershop et un objet barberprestation. Grâce à ce dernier, on récupère le barbershop choisi, son personnel, son ID, et ses horaires.

```
49 |     $barbershop = $barberPrestation->getBarbershop();
50 |     $personnels = $barbershop->getPersonnels();
51 |     $barbershopId = $barbershop->getId();
52 |     $horaires = $barbershop->getHoraires();
```

On pourra effectuer des vérifications sur la validité du rendez-vous plus tard grâce à ces informations.

Cette fonction sert aussi à créer le formulaire d'ajout de rendez-vous, auquel on passe l'id du barbershop sélectionné.

```
54 |     // Création du form avec envoi de $barbershopId au form builder
55 |     $form = $this->createForm(RendezVousType::class, $rendezvous, ['barbershopId' => $barbershopId]);
```

Cela nous permet dans le rendezVousType, pour le champ personnel, de ne proposer que les employés qui sont rattachés au barbershop passé en id en utilisant la méthode query\_builder du FormBuilder de symfony. Grâce à cela, je peux faire une requête DQL pour alimenter mon champ.

```
35 |         'query_builder' => function (EntityRepository $er) use ($barbershopId) {
36 |             return $er->createQueryBuilder('p')
37 |                 ->where('p.barbershop = :barbershopId')
38 |                 ->setParameter('barbershopId', $barbershopId);
```

La couche modèle envoie une requête à la base de données et récupère les objets personnels qui ont pour barbershop\_id l'id que nous avons passé au formulaire. Ils seront les seuls affichés sous forme de boutons radio.

La prise de rendez-vous nécessite de renseigner au minimum son numéro de téléphone. Le nom et le prénom sont facultatifs. Pour ne pas redemander à chaque fois ces informations à un utilisateur qui les aurait déjà remplies, je met une condition après la génération du formulaire.

```
61 |         if($this->getUser()->getTelephone()){
62 |             $form->remove('nom');
63 |             $form->remove('prenom');
64 |             $form->remove('telephone');
65 |         }
```

Si l'utilisateur en session a au moins renseigné son téléphone, je retire les 3 champs d'information du formulaire de façon à ce qu'ils ne lui soient pas proposés.

Le formulaire est complet et fonctionnel. La suite du code concerne maintenant la vérification des données envoyées, notamment la validité du rendez-vous.

Il n'est pas possible de sélectionner un rendez-vous invalide sans modifier le code HTML de la page, car comme on l'a vu, le tableau \$allCreneaux contient uniquement des créneaux disponibles. Toutefois, rien n'empêche un utilisateur mal intentionné d'ouvrir la console de son navigateur, de modifier l'URL pour changer la valeur d'un créneau, ou de sélectionner une prestation qui n'existe pas.

Pour contrer cela, j'ai mis en place une série de vérifications, d'abord sur le personnel et la prestation choisie, puis sur l'heure de début du rendez-vous.

Pour vérifier que la prestation choisie est bien proposée dans le barbershop sélectionné, j'utilise la méthode `in_array` en lui passant la prestation ainsi que le tableau d'objet de Prestations du barbershop.

```
41 |     // On vérifie si la prestation choisie est bien proposée par le barbier
42 |     if(!in_array($barberPrestation, $barbershop->getBarberPrestations()->toArray())){
43 |
44 |         return $this->redirectToRoute('app_home');
45 |     }
```

Si la prestation n'est pas contenue dans le tableau de prestation, l'utilisateur est redirigé sur la page d'accueil. Sur le même modèle, je vérifie que l'employé choisi fait bien partie du tableau d'objet Personnel du barbershop. S'il n'y figure pas, même traitement.

Je peux ensuite commencer les vérifications qui concernent le rendez-vous. Pour commencer, je m'assure que le rendez-vous a bien été pris en dehors des jours de fermeture.

```
93 |     // VERIF SI RDV PRIS UN JOUR DE FERMETURE
94 |
95 |     // Récupération du jour en français
96 |     setlocale(LC_TIME, 'fr_FR.UTF-8');
97 |     $jour = strtolower(strftime('%A', $heureDebut->getTimestamp()));
98 |     $horairesArray = json_decode($horaires, true);
99 |
100 |
101 |     if($horairesArray[$jour]['ouverture'] == 'ferme' || $horairesArray[$jour]['fermeture'] == 'ferme'){
102 |         notif()
103 |             ->position('x', 'right')
104 |             ->position('y', 'bottom')
105 |             ->addError('Le barbershop est fermé ce jour là.');
106 |
107 |         return $this->redirectToRoute('add_rendezvous',['id' => $barbershopId, 'barberPrestation' => $barberPrestation->getId()]);
    }
```

Après avoir récupéré l'heure de début issue du formulaire, je la convertis en objet date pour pouvoir la manipuler et la comparer plus facilement. Je récupère ensuite les horaires du barbershop en tableau PHP, et je vérifie que la valeur « ouverture » ou « fermeture » associée à la clé du jour n'est pas égale à « fermé ».

Si c'est le cas, j'envoie une notification d'erreur et je redirige sur la page de prise de rendez-vous.

Sur ce même modèle, 3 autres vérifications s'assurent successivement que l'heure de rendez-vous est bien définie, que le rendez-vous est bien pris après l'heure du jour, et que le rendez-vous est bien pris pendant les heures

d'ouverture. Je ne les listerai pas ici car elles sont très semblables à la vérification ci-dessus, seule la condition change.

La dernière vérification permet de s'assurer que le rendez-vous n'existe pas déjà. Cette dernière s'appuie sur une fonction que j'ai créée dans le RendezVousRepository et qui prend une date de début ainsi qu'un objet personnel en arguments :

```
42     public function checkIfRdvExist($debut, $personnel){  
43  
44         $result = $this->createQueryBuilder('r')  
45             ->andWhere('r.debut = :debut')  
46             ->andWhere('r.personnel = :personnel')  
47             ->setParameter('debut', $debut)  
48             ->setParameter('personnel', $personnel)  
49             ->getQuery()  
50             ->getOneOrNullResult()  
51     ;  
52  
53     return ($result != null);  
54 }
```

La requête DQL exécutée cherche un rendez-vous ayant une date de début et un personnel\_id égaux à ceux passés en paramètre. Elle retourne true ou false : true signifie que le rendez-vous existe déjà, sinon il n'existe pas.

Il suffit ensuite de faire une condition dans le contrôleur pour afficher une notification d'erreur et une redirection si checkIfRdvExist est égal à true.

Une fois toutes les vérifications passées, le rendez-vous est considéré comme valide : on peut donc ajouter l'heure de fin, l'utilisateur, la prestation, et l'heure de début en base de données, ainsi que le nom, prénom et téléphone s'ils ont été renseignés.

J'envoie ensuite un mail à l'utilisateur grâce au Mailer de Symfony, pour confirmer le rendez-vous et rappeler les détails de ce dernier.

```
155     $email = (new Email())  
156         ->from('admin@barberhub.com')  
157         ->to($user->getEmail())  
158         ->subject('Votre rendez vous chez '. $barbershop->getNom().'')  
159  
160         ->html( '<p>Cher '.$user->getPseudo().',</p> <p>Nous sommes ravis de vous confirmer votre rendez-vous
```

*Extrait du mail de confirmation de rendez-vous*

Enfin, j'initialise un jeton 'justBookedRdv' qui sera stocké en session et servira pour la page de confirmation, sur laquelle redirige la fonction après la prise de rendez-vous.

```
175 // Jeton de session pour indiquer que le user vient de prendre un RDV ( pour page confirm )  
176 $_SESSION['justBookedRdv'] = true;
```

Dans la fonction confirmRdv, on vérifie si le jeton de session n'existe pas ou s'il n'est pas égal à true. Si c'est le cas, on redirige vers la page d'accueil. Je récupère ensuite le dernier rendez-vous de l'utilisateur grâce à une méthode personnalisée du RendezVousRepository (ligne 319) et je repasse le jeton de session à False.

```

313     // Si le jeton de session hasRendezVous n'est pas défini ou n'est pas à true
314     if (!isset($_SESSION['justBookedRdv']) || $_SESSION['justBookedRdv'] != true) {
315         return $this->redirectToRoute('app_home');
316     }
317
318     $lastRDV = $ur->getLastRendezVous($user);
319
320     // Une fois la page consultée on repasse le jeton à true
321     $_SESSION['justBookedRdv'] = false;
322

```

Ceci me permet d'afficher la confirmation de rendez-vous seulement aux personnes venant de prendre rendez-vous. Il n'est pas possible d'accéder à cette page sans rendez-vous car il n'y aurait pas de jeton de session et la redirection serait effectuée.

Les rendez-vous restent toutefois consultables dans l'espace personnel de l'utilisateur.

La prise de rendez-vous étant maintenant fonctionnelle et sécurisée, il faut maintenant s'occuper de l'affichage des rendez-vous sur le planning du barbier.

### 3. Affichage du planning du barbier

Pour afficher le planning des barbiers, j'ai utilisé un plugin Javascript du nom de FullCalendar. Il permet de générer un calendrier et propose une multitude d'options pour le paramétrier et y ajouter des événements.

Après avoir installé le plug-in, il faut alimenter le calendrier avec les rendez-vous du barbier en session. J'ai choisi de donner la possibilité à l'utilisateur de changer entre deux types d'affichage : soit l'affichage de tous les rendez-vous, soit l'affichage des rendez-vous à venir.

Pour cela, je mets en place deux boutons dans ma vue, qui contiennent les valeurs « all » et « upcoming ».

Je place un écouteur d'évènement sur chacun de ces boutons, lié à une requête AJAX qui est chargée de faire passer le type d'affichage choisi dans le présentateur.

```

78     $('.rdv-button').click(function(e) {
79         e.preventDefault();
80         $('.rdv-button').removeClass('active');
81         $(this).addClass('active');
82
83         var displayType = this.value
84         $.ajax({
85             type: "POST",
86             url: "/monespace/getrdv",
87             contentType: 'application/json',
88             data: displayType,

```

La valeur de la variable display est définie par défaut sur upcoming. Elle prendra ensuite la valeur sélectionnée par l'utilisateur via le bouton.

```

107     $user = $this->getUser();
108     $personnel = $user->getPersonnel();
109     // affichage des rendez vous a venir par défaut
110     $display = $request->getContent();
111
112     if ($display === 'upcoming') {
113         $events = $pr->getUpcomingRendezVous($personnel);
114     } else {
115         $events = $user->getPersonnel()->getRendezVouses();
116     }

```

Si la valeur de display récupérée dans la requête (ligne 110) est égale à upcoming, donc la valeur par défaut, la variable events contiendra seulement les rendez-vous à venir grâce à une requête DQL créée dans le PersonnelRepository, qui récupère tous les rendez-vous après la date actuelle. Sinon, elle contiendra tous les rendez-vous du barbier.

Je peux ensuite boucler sur la variable events pour créer un tableau contenant les données qu'il me faut pour l'affichage des rendez-vous : l'id, le début, la fin, le titre et la prestation.

```

53     $rdvs = [];
54     // Boucle sur chaque rdv
55     foreach($events as $event){
56         $prestations = $event->getBarberPrestation();
57
58         // Boucle sur chaque collection de prestation
59         foreach($prestations as $prestation){
60             // tableau avec toutes les infos
61             $rdvs[] = [
62                 'id' => $event->getId(),
63                 'start' => $event->getDebut()->format('Y-m-d H:i:s'),
64                 'end' => $event->getFin()->format('Y-m-d H:i:s'),
65                 'title' => $event->getUser()->getPseudo() . " - " . $prestation->getPrestation()->getNom(),
66             ];
67         }
68     }

```

Une fois cela fait, j'encode le tableau PHP en JSON et je l'envoie à la vue pour qu'il puisse être utilisé par FullCalendar afin d'alimenter le calendrier.

```

70     // On le met en JSON et on l'envoie à la vue
71     $data = json_encode($rdvs);
72
73     return $this->render('user/rdv.html.twig', compact('data'));
74 }

```

Le JSON contenant tous les rendez-vous est récupéré dans la constante data. La suite du code concerne le paramétrage de FullCalendar : il est possible de définir la vue de base du calendrier (mois, semaine...), le fuseau horaire et la langue utilisée dans le calendrier, ainsi que les différents boutons de navigation proposés dans la barre d'outil, qui permettent notamment de changer de vue (entre semaine, jour, mois...). La ligne 41 permet de donner une source d'évènement au calendrier : c'est là que je passe la constante data préalablement définie.

```

27 | const data = JSON.parse('{{ data|raw }}')
28 |
29 | window.onload = () => {
30 |   let calendarEl = document.querySelector('#calendar');
31 |   let calendar = new FullCalendar.Calendar(calendarEl, {
32 |     initialView: 'dayGridMonth',
33 |     locale : 'fr',
34 |     timeZone : 'Europe/Paris',
35 |     navLinks: true,
36 |     headerToolbar: {
37 |       start : 'prev,next,today',
38 |       center:'title',
39 |       end: 'dayGridMonth,timeGridWeek,timeGridDay'
40 |     },
41 |     events: data,

```

A ce stade, les données sont affichées dans le calendrier. Il faut maintenant afficher les détails du rendez-vous lors du clic sur un événement.

Pour cela, on peut utiliser une propriété de fullCalendar, eventClick : lorsqu'un clic est détecté, une fonction se charge de récupérer toutes les informations du rendez-vous dans des constantes : titre, début, fin et id.

```

51 |   eventClick: function(info) {
52 |
53 |     const title = info.event.title
54 |     const start = new Date(info.event.start).toISOString().substr(11, 5)
55 |     const end = new Date(info.event.end).toISOString().substr(11, 5)
56 |     const id = info.event.id

```

J'initialise ensuite une route qui fera appel au FactureController et qui permettra de générer une facture. Dans cette route j'injecte l'ID du rendez-vous récupéré précédemment.

```

58 |   // On crée la route pour la facture
59 |   let invoiceUrl = "{{ path('app_facture', {id : 'id'}) }}";
60 |   // On remplace le placeholder par l'id
61 |   invoiceUrl = invoiceUrl.replace('id', id);

```

Ensuite , j'utilise la bibliothèque SweetAlert2 pour initialiser un message à destination de l'utilisateur contenant les détails du rendez-vous. Cette bibliothèque javascript permet de pousser la personnalisation des pop-ups d'alerte classiques présentes nativement dans les navigateurs.

```

63 |   Swal.fire({
64 |     title: title,
65 |     text: 'Début : ' + start + '\n' + 'Fin : ' + end,
66 |     confirmButtonColor: '#000000',
67 |     footer: '<a href=' + invoiceUrl + '> Editer une facture </a>'
68 |   })

```

Au clic sur un rendez-vous, l'utilisateur pourra donc voir le titre, le début et la fin du rendez-vous, ainsi qu'un lien en bas de l'alerte avec l'url créée précédemment, pour permettre au barbier d'édition une facture (voir Annexe 5).

## 4. Génération de facture

Au clic sur ce lien, la fonction generateInvoice du FactureController est déclenchée. Elle prend en paramètre un objet rendez-vous. Cet objet rendez-vous est retrouvé grâce à l'id que nous avons passé dans l'URL. A partir de ce dernier, je peux récupérer toutes les informations nécessaires à l'établissement d'une facture :

```
19 // On récupère le barbershop, l'employé, les prestations, le client et la date
20 $date = $rendezVous->getDebut();
21 $client = $rendezVous->getUser();
22 $employe = $rendezVous->getPersonnel()->getUser();
23 $barbershop = $rendezVous->getPersonnel()->getBarbershop();
24 $prestations = $rendezVous->getBarberPrestation();
```

Pour me faciliter la tâche et éviter de devoir faire cela dans la vue TWIG, je calcule directement dans le présentateur le prix total des prestations choisies, simplement en ajoutant tous les prix dans un tableau et en faisant la somme de toutes les valeurs de ce tableau grâce à la fonction native de PHP array\_sum.

```
26 // On fait un tableau avec le prix de toutes les prestations
27 $allPrice = [];
28 foreach($prestations as $prestation){
29     $allPrice[] = $prestation->getPrix();
30 }
31
32 // Calcul du total des prix avec array_sum
33 $total = array_sum($allPrice);
```

On peut maintenant passer à la création de la facture et à sa génération en PDF. Pour cela, j'ai utilisé le bundle DomPDF. Ce bundle open source permet de générer un PDF à partir de HTML.

J'initialise un objet domPdf et je lui passe un tableau d'option préalablement créé. J'initialise ensuite une variable qui contiendra l'HTML nécessaire à la création du PDF. Celui ci est généré sur la base d'une vue TWIG, à laquelle je passe tous les paramètres dont j'ai besoin pour générer ma facture.

```
40 $dompdf = new Dompdf($pdfOptions);
41
42 // Génération du HTML du PDF
43 $html = $this->renderView('facture/template.html.twig', [
44     'client' => $client,
45     'employe' => $employe,
46     'prestations' => $prestations,
47     'total' => $total,
48     //'image' => $base64,
49     'barbershop' => $barbershop,
50     'date' => $date
51 ]);
```

La vue est composée d'un tableau HTML qui prend la forme d'une facture, de règles de style CSS pour la mise en forme, et de variables TWIG pour rendre le tout dynamique.

```
150      {% for prestation in prestations %}  
151  
152          <tr class="item{% if loop.last %} last{% endif %}">  
153              <td>{{ prestation.prestation.nom }} - par {{employe.pseudo}}</td>  
154              <td>{{ prestation.prix }}€</td>  
155          </tr>  
156  
157      {% endfor %}
```

On retrouve ici par exemple une boucle for permettant d'afficher les prestations, leur prix, et la personne ayant réalisé la prestation.

Enfin, on injecte cet HTML dans l'objet domPDF, puis on fait le rendu et l'affichage sur le navigateur grâce aux fonctions de DomPDF render et stream.

```
60 $dompdf→loadHtml($html);
61
62 $dompdf→setPaper('A4', 'portrait');
63
64 $dompdf→render();
65
66 $dompdf→stream("invoice.pdf", [
67     "Attachment" => false,
68 ]);
```

On passe en paramètre le nom de la facture ainsi que « attachment », qui correspond au téléchargement automatique de la facture ou non.

Les barbiers sont maintenant en capacité d'afficher leur rendez-vous et de générer une facture pour chacun d'entre eux (voir Annexe 6).

### III. Sécurité

## 1. L'injection SQL

L'injection SQL est une technique permettant d'injecter des éléments, notamment du code de type SQL, dans les champs des formulaires web ou dans les liens des pages afin de les envoyer au serveur web. De cette façon, les attaquants contournent les contrôles de sécurité et peuvent accéder à la base de données, pour en récupérer les mots de passe, ou des coordonnées bancaires.

Premièrement, pour éviter qu'un attaquant récupère facilement des informations sensibles de la base de donnée, les mots de passe des utilisateurs sont hachés, c'est à dire qu'ils sont transformés en une chaîne de caractères aléatoire et unique appelée hash. Ce hash est généré grâce à un algorithme de hachage. Le hash est stocké dans la base de données au lieu du mot de passe en clair. Lorsque l'utilisateur tente de se connecter, le mot de passe fourni est à nouveau hashé et comparé au hash stocké dans la base de données. Si les deux correspondent, l'utilisateur est authentifié. Symfony utilise par défaut l'algorithme le plus efficace installé sur le système, entre Bcrypt et Argon2i.

```
46 | | | | | Symfony\Component\Security\Core\User\PasswordAuthenticatedUserInterface:  
47 | | | | |     algorithm: auto  
48 | | | | |     cost: 4 # Lowest possible value for bcrypt  
49 | | | | |     time_cost: 3 # Lowest possible value for argon  
50 | | | | |     memory_cost: 10 # Lowest possible value for argon
```

Un utilisateur malveillant, même si il arrive à accéder à la base de données, ne pourra donc pas visualiser les mots de passe en clair.

Doctrine intègre une protection native des injections SQL, sur les requêtes classiques tout du moins, qui sont préparées automatiquement. Un moule de requête réutilisable est créé avec des marqueurs nommés, et la valeur de ces marqueurs n'est donnée que lors de l'exécution de la requête. Cela permet d'échapper les caractères spéciaux qui peuvent être rentrés par un utilisateur malveillant et sécurise ainsi la requête. Lors de l'utilisation de requêtes DQL plus poussées, il est important de sécuriser les requêtes effectuées avec le QueryBuilder, en utilisant setParameter pour préparer manuellement les paramètres passés à la requête.

```
47 | | | $query =  
48 | | |     $qb->select('r')  
49 | | |     ->from('App\Entity\RendezVous', 'r')  
50 | | |     ->where('r.debut > :currentdate')  
51 | | |     ->andWhere('r.personnel = :personnel')  
52 | | |     ->setParameter('currentdate', new \DateTime())  
53 | | |     ->setParameter('personnel', $personnel)  
54 | | |     ->getQuery();
```

Sur cet exemple :currentdate et :personnel sont les marqueurs qui seront remplacés à l'exécution de la requête respectivement par un objet new DateTime et la variable \$personnel (ligne 52 et 53).

## 2. Faille XSS

La faille XSS est une attaque qui vise à insérer un code malveillant dans le contenu du site web ciblé, via un formulaire, ou la barre d'url. Il est ensuite inclus dans le contenu dynamique reçu par le navigateur et peut permettre d'altérer des données, des contenus, d'exécuter un malware... les possibilités d'exploitation sont nombreuses.

Pour se prémunir face à cette faille, le moteur de template twig, utilisé nativement dans Symfony, constitue une première barrière : il échappe automatiquement tout ce qui est affiché à l'écran, et propose des balises et des filtres pour sécuriser des portions de textes ou des variables.

La seconde barrière que l'on peut mettre en place est la fonction symfony isValid(), que l'on utilise avant le traitement de chaque formulaire. Cette fonction se base sur les paramètres renseignés par le formType pour vérifier si les données sont valides ou non.

```

22     $builder
23         ->add('titre', TextType::class, ['attr'=>
24             ['placeholder' => 'Titre de l\'article']
25         ])
26         ->add('photo', UrlType::class, ['attr'=>
27             ['placeholder' => 'URL de la photo']
28         ])
29         ->add('description', TextType::class, [
30             'attr'=>
31                 ['placeholder' => '200 caractères max.'],
32             'constraints' => [
33                 new Length([
34                     'max' => 200,
35                     'maxMessage' => 'La description ne peut pas dépasser {{ limit }} caractères.'
36                 ]),
37             ],
38         ])

```

*Exemple de contraintes pour le formulaire d'ajout d'article*

Par exemple, dans ce cas-ci, le champ titre devra contenir du texte, le champ photo devra contenir une URL, le champ description devra contenir une description de 200 caractères maximum... En fonction des champs, des filtres sont appliquées pour assainir et s'assurer que les données récupérées sont bien du type voulu.

Il est également possible d'ajouter des contraintes directement sur les propriété des entités, avec l'annotation ASSERT suivie du type de contrainte que l'on veut mettre en place et du message à afficher en cas de non respect de cette contrainte.

```

25 #[ORM\Column]
26 #[Assert\NotBlank(message: "La note ne peut pas être vide. ")]
27 private ?int $note = null;

```

Sur cet exemple, on retrouve une contrainte NotBlank associée au champ note de l'entité Avis. Cette contrainte précise que le champ ne doit pas être vide sous peine d'afficher un message « La note ne peut pas être vide ».

### 3. Faille CSRF

La faille CSRF (cross site request forgery) est une attaque où l'utilisateur est dupé pour effectuer une action non intentionnelle à son insu. Elle exploite le fait que les sites web font souvent confiance aux demandes provenant d'un utilisateur authentifié sans vérifier l'origine de la demande. L'attaque se déroule comme suit : l'utilisateur authentifié est redirigé vers un site web malveillant ou un courriel piégé contenant un lien vers le site cible. Lorsque l'utilisateur clique sur le lien, le site malveillant déclenche automatiquement une action sur le site cible, comme l'envoi d'un formulaire, l'ajout d'un nouvel utilisateur, ou la modification d'une donnée sensible. Cette action est exécutée au nom de l'utilisateur authentifié, sans que celui-ci en ait conscience.

Pour s'en prémunir, on peut utiliser des jetons, contenant des valeurs uniques générées aléatoirement, stockés côté serveur et envoyés au client. Lors de la soumission d'une action, ce jeton sera renvoyé au serveur et vérifié. Si le jeton est invalidé, l'action sera refusée.

Symfony intègre nativement un système de jeton CSFR et met en place un champs Hidden contenant un jeton sur les formulaires créés.

```

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

```

Ce champs hidden est généré automatiquement dans le formulaire de connexion. Voilà à quoi ressemble le token CSRF une fois généré :

```
<input id="registration_form__token" type="hidden" name="registration_form[_token]" value="e1c26a0c1a4502f4ab1b70ad8478144.fu4NN4tUu0CfqGyfIgKHQStWGfb2...w.F7hufc5l7TLKkQKmcGfJMn4-V4GAjQl6fKA_qAVEif5Pq29mvAbKntbtLQ">
```

#### 4. Attaque par force brute ou par dictionnaire.

Une attaque par force brute (bruteforce attack) consiste à tester, l'une après l'autre, chaque combinaison possible d'un mot de passe ou d'une clé pour un identifiant donné afin de se connecter au service ciblé.

L'attaque par dictionnaire (ou en anglais « dictionary attack »), quant à elle, est une méthode utilisée pour pénétrer par effraction dans un ordinateur ou un serveur protégé par un mot de passe, en essayant systématiquement tous les mots d'un dictionnaire donné.

Une première façon de se protéger face à ces attaques est de mettre en place une politique de mot de passe forte. La CNIL recommande 12 caractères au minimum, une majuscule, un signe de ponctuation ou un caractère spécial, et un nombre. De cette façon, le mot de passe prendrait trop de temps à être trouvé par les attaquants du fait de sa complexité (près de 226 ans pour un mot de passe de ce type), ce qui limite grandement les risques.

Depuis Symfony 6.3, on peut utiliser la contrainte `PasswordStrength`, qui oblige l'utilisateur à rentrer un mot de passe qui correspond au niveau de complexité défini dans la contrainte.

Dans le cadre de mon application, je suis passé par l'utilisation d'expression régulières (abrégé RegExp en anglais) qui permettent de s'assurer que le mot de passe utilisateur respecte bien un pattern défini. Une RegExp est une chaîne de caractères qui décrit, selon une syntaxe précise, un ensemble de chaînes de caractères possibles.

```
53 |     'constraints' => [
54 |       new Regex('/^(?=.*[A-Z])(?=.*[a-z])(?=.*[0-9])(?=.*[#?!$%^&*-]).{12,}$/', "Le mot de
55 |       passe doit contenir 12 caractères avec une majuscule, une minuscule, un chiffre et un
|         caractère spécial.")
|     ],
|   ],
| }
```

Chaque parenthèse de la Regex stipule qu'on doit trouver ce qui est entre crochet au moins une fois pour que les contraintes soit respectées. A la fin, entre accolade, le 12 définit le nombre de caractères minimum.

Cette contrainte dans le `RegistrationFormType` permet de s'assurer au niveau du back-end que le mot de passe choisi doit passer la regex pour que l'inscription puisse se poursuivre.

Pour une expérience utilisateur plus plaisante que simplement recevoir un message d'erreur lors de l'envoi du formulaire si le mot de passe ne respecte pas les règles, j'ai également mis en place une vérification front avec du Javascript. A l'aide de RegExp, j'ai mis en place une fonction permettant de tester le mot de passe saisi par l'utilisateur, qui s'active dès que l'utilisateur change de champ dans le formulaire grâce à un écouteur d'évènement.

A chaque changement dans les champs ciblés, les entrées sont testées dans la fonction validPassword. Il y a au total 5 vérifications, qui s'assurent que le mot de passe entré respecte bien la politique de mot de passe fort.

```

14 // On écoute l'envoi du formulaire
15 form.addEventListener('submit', function(e){
16     e.preventDefault();
17     if(validEmail(inputEmail) && validPassword(inputPassword)){
18         form.submit();
19     }
20 })

```

Un message est stocké dans une variable et affiché à l'utilisateur en fonction des tests qui ne sont pas passés.



Toutefois, cette vérification javascript peut facilement être désactivée par un utilisateur malveillant et ne constitue donc pas une sécurité supplémentaire. Elle permet seulement d'aiguiller l'utilisateur dans la création d'un mot de passe complexe. C'est la RegExp dans le formType qui oblige vraiment l'utilisateur à rentrer un mot de passe fort.

## 5. Anti-Spam

Les formulaires, et notamment les formulaires de contact, sont fréquemment sujets aux spams en tout genre, envoyés la plupart du temps par des robots qui parcourrent les sites à la recherche de formulaires.

Pour contrer ces robots, on peut utiliser un Captcha, comme le ReCaptcha de Google, qui va calculer un score basé sur le comportement de l'utilisateur, et décider s'il laisse l'accès libre au formulaire, s'il fait un test de reconnaissance d'image/cochage de case, ou s'il bloque totalement l'accès au formulaire.

Ayant pu expérimenter la mise en place de ReCaptcha 3 pendant mon stage, j'ai voulu tester une autre solution permettant de contrer ces robots spammeurs : le pot de miel, ou HoneyPot.

Le principe est simple : on rajoute des champs à un formulaire, qui ne sont pas visibles par un utilisateur lambda. Le robot, lui analyse le code HTML de la page et remplit automatiquement tous les champs qu'il voit. Si les champs cachés sont remplis lors de l'envoi du formulaire, c'est donc qu'un robot est passé par là.

Pour mettre cela en place, on commence par créer une classe HoneyPotType. Cette classe permet de mettre en place deux champs supplémentaires, « phone » et « mail », que l'on va cacher par la suite.

```

27 |     protected const DELICIOUS_HONEY_CANDY_FOR_BOT = "phone";
28 |     protected const FABULOUS_HONEY_CANDY_FOR_BOT = "mail";
29 |
30 |     public function buildForm(FormBuilderInterface $builder, array $options)
31 |     {
32 |         $builder->add(self::DELICIOUS_HONEY_CANDY_FOR_BOT, TextType::Class, $this->setHoneyPotFieldConfiguration())
33 |             ->add(self::FABULOUS_HONEY_CANDY_FOR_BOT, TextType::Class, $this->setHoneyPotFieldConfiguration())
34 |             ->addEventSubscriber(new HoneyPotSubscriber($this->honeyPotLogger, $this->requestStack));
35 |
36 |     }
37 |
38 |     protected function setHoneyPotFieldConfiguration(): array
39 |     {
40 |         return [
41 |             'attr' => [
42 |                 'autocomplete' => 'off',
43 |                 'class'        => 'candy',
44 |                 'tabindex'      => '-1'
45 |             ],
46 |             'row_attr' => ['class' => 'candy'],
47 |             'mapped'    => false,
48 |             'required'  => false
49 |         ];
50 |
51 |     }
52 |

```

La fonction build form ajoute les deux champs supplémentaires, avec leur type et leur configuration, qui est définie dans la fonction setHoneyPotFieldConfiguration(). Ces deux champs supplémentaires sont cachés des utilisateurs à l'aide de CSS. La fonction suivante permet de paramétriser les champs cachés.

Procéder de cette façon permet de simplement remplacer AbstractType par HoneyPotType pour mettre en place le HoneyPot sur un formulaire. Comme HoneyPotType hérite elle même de la classe AbstractType, le formulaire profitera des deux classes.

```
20 class RegistrationFormType extends HoneyPotType
```

On met ensuite en place une classe HoneyPotSubscriber, où on va pouvoir gérer les événements liés à la souscription du formulaire.

La fonction checkHoneyJar va permettre d'obtenir les données renvoyées par le formulaire et de mettre la logique en place en fonction de ce que l'on récupère.

Après avoir récupéré le contenu de la requête, on vérifie si les clés « phone » et « mail » existent. Si elles n'existent pas, c'est que le formulaire a été modifié : on envoie donc une erreur.

```

34 |     public function checkHoneyJar(PreSubmitEvent $event): void
35 |     {
36 |         $request = $this->requestStack->getCurrentRequest();
37 |
38 |         if(!$request){
39 |             return;
40 |         }
41 |
42 |         $data = $event->getData();
43 |
44 |         if(!array_key_exists('phone', $data) || !array_key_exists('mail', $data))
45 |         {
46 |             throw new HttpException(400, "Vous ne devriez pas faire ça ...");
47 |         }

```

On vérifie ensuite si les valeurs associées à « phone » et à « mail » ne sont pas vides. Si c'est le cas, c'est qu'on a affaire à un robot (ou à une personne qui s'est amusée à remplir les champs cachés en inspectant le code HTML, mais cela est moins probable).

```
49 [ 'phone' => $phone,
50   'mail' => $mail
51 ] = $data;
52
53
54 if($phone != "" || $mail != ""){
55   $this->honeyPotLogger->info("Bot spammeur potentiellement détecté. IP : '{$request->getClientIp()}'. Le
56   champs phone contenait : '{$phone}' et le champs mail contenait '{$mail}'.");
57
58   throw new HttpException(403, "Les robots ne sont pas les bienvenus ici !");
59 }
60
61 }
```

On utilise alors la librairie Monolog, qui permet de créer et de stocker des messages de log, et on ajoute un message dans un fichier de log HoneyPot, avec l'adresse IP du client qui a envoyé la requête, et les valeurs contenues dans les champs. Ensuite, on affiche une exception avec un message d'erreur pour bloquer le robot.

Les informations des robots spammeurs sont récupérées dans le fichier de logs à chaque tentative:

```
2 [2023-08-20T16:34:49.763614+00:00] honey_pot.INFO: Bot spammeur potentiellement détecté. IP : '127.0.0.1'. Le
3   champs phone contenait : 'fake data :( et le champs mail contenait 'fake data :(.
[] []
[2023-08-20T16:38:54.981201+00:00] honey_pot.INFO: Bot spammeur potentiellement détecté. IP : '127.0.0.1'. Le
  champs phone contenait : 'fake data :( et le champs mail contenait 'fake data :(.
[] []
```

Une suite logique à cela serait de mettre en place Fail2Ban, un outil qui analyse les logs d'un site, y repère des actions malveillantes et utilise le firewall du serveur pour interdire l'accès au serveur à l'IP associé à cette action. En lui fournissant les logs de notre fichier HoneyPot, il est possible de bannir automatiquement tous les robots spammeurs qui y figurent. Je n'ai malheureusement pas pu mettre en place cela sur ce projet car il a été développé en local.

## 6. Faille Upload

La faille upload consiste à injecter du code malveillant, par exemple un script PHP, dans les fichiers qui peuvent être transmis via un formulaire de téléchargement de fichier. Il suffit ensuite au pirate d'appeler son script pour que celui-ci s'exécute.

Pour pallier à cette faille, il faut contrôler dans un premier temps que l'utilisateur ne puisse uploader que des Images, et pas d'autres types de fichier. Pour ce faire, on peut rajouter des contraintes au champs image qui est de type FileType, en spécifiant une contrainte newImage, qui garantit que seul fichier image sera accepté. On peut lui passer des paramètres, comme une taille maximale ou un message d'erreur.

```

44     →add('images', FileType::class, [
45         'label' =>"Photo",
46         'multiple'=>false,
47         'mapped' =>false,
48         'required' =>false,
49         'constraints' => [
50             new Image([
51                 'maxSize' => '8M',
52                 'maxSizeMessage' => 'L\'image est trop grande',
53                 'mimeTypesMessage' => 'Le fichier n\'est pas une image valide'
54             ])
55         ])
56     ]

```

De plus, dans mon service de gestion des images, je donne un nom aléatoire à l'image, sur la base du hashage d'un identifiant unique basé sur l'horodatage actuel, et de nombres générés aléatoirement.

```

20     // On donne un nouveau nom a l'image
21     $fichier = md5(uniqid(rand(), true)) . '.webp';

```

Même si l'image contient du code malveillant, il devient quasi impossible pour un attaquant de faire appel à cette dernière car il ne connaît pas son nom. De plus, on lui attribue systématiquement une extension .webp pour éviter que les attaquants essaient de mettre plusieurs extensions de fichier afin que l'image passe quand même les vérifications ( un fichier barber.php.png sera transformé en barber.webp).

## 7. Gestion des droits d'accès

Paramétrier les droits d'accès est très important dans une application, pour que chaque utilisateur ne puisse utiliser que les fonctions qui lui sont destinées. Laisser un utilisateur lambda mal intentionné accéder aux fonctions d'administration par exemple, peut avoir des conséquences désastreuses.

Symfony intègre un système de rôles, qui sont enregistrés en base de données sous la forme d'attribut dans la table user. On retrouve 3 rôles dans mon application : ROLE\_USER, ROLE\_ADMIN, ROLE\_BARBER.

On a deux possibilité pour restreindre l'accès à une page, ou à une fonctionnalité :

On peut définir des chemins d'accès et les limiter à certains rôles depuis le fichier security.yaml :

```

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

```

Et on peut également annoter directement les méthodes de nos controller avec l'annotation IsGranted ainsi que le rôle que l'on veut autoriser.

```

20     #[Route('/administration', name: 'control_pannel')]
21     #[IsGranted('ROLE_ADMIN')]

```

En l'occurrence, la route administration n'est disponible que pour les rôles admin. Cette annotation permet aussi d'annoter une classe entière.

Pour autoriser plusieurs rôles à utiliser une fonction ou une classe, on utilise des expressions Symfony.

```
19  #[IsGranted(new Expression('is_granted("ROLE_ADMIN") or is_granted("ROLE_USER")'))]
```

Dans les vues, pour masquer certains boutons qui permettent d'accéder à des fonctions réservées à des rôles définis, on peut utiliser une condition TWIG avec is\_granted :

```
11  |  {% if is_granted('ROLE_ADMIN') %} | 11
12  |    <li><a href="{{path ('control_pannel')}}"> Administration </a></li> | 12
13  |  {% endif %} | 13
```

Dans cette situation, le lien vers le panneau d'administration sera masqué pour les utilisateurs qui n'ont pas le rôle Admin.

## 8. Veille sur les autres failles de sécurité

Pour mener ma veille sur les failles de sécurité qui peuvent toucher les applications web, je me suis appuyé sur le site de l'OWASP, l'Open Web Application Security Project. C'est une organisation à but non lucratif qui se consacre à la sécurité des applications web.

Elle propose un top 10 des risques les plus critiques encourus par les sites web, régulièrement mis à jour et alimenté par une équipe d'experts en sécurité du monde entier.

J'ai pu découvrir de nouvelles failles de sécurité :

**La faille SSRF** ( Server Side Request Forgery ) : cette faille permet à un attaquant d'inciter l'application côté serveur à envoyer des requêtes à un endroit non prévu, pour en extraire des données sensibles, scanner le réseau interne, cartographier les Ports et IP ouverts, forcer la connexion à des systèmes externes...

Cette faille est prise en compte nativement par Symfony, qui depuis la version 5.3 a implémenté dans le HttpClientComponent un composant qui bloque toutes les adresses IP internes par défaut.

**Les composants vulnérables et obsolètes** : certaines dépendances obsolètes ou présentant des problèmes de sécurité peuvent mettre en péril l'intégrité de l'application et permettre à des attaquants de rentrer dans le système ou d'y extraire des informations.

Pour s'en prémunir, il est recommandé de supprimer les dépendances inutiles ou obsolètes et de surveiller celles qui ne sont plus maintenues, de faire un inventaire continu des versions à la fois client et serveur, de télécharger les dépendances auprès de source fiables.

On retrouve également des failles qui sont liées à des problèmes de conception ou de configuration :

**La conception non sécurisée** : représente différentes insuffisances et des contrôles de conceptions manquants ou inefficaces en matière de sécurité web lors de la création d'une application.

**Mauvaises configurations de sécurité** : fonctionnalités inutiles activées ou installées, comptes par défaut activés et inchangés, système mis à niveau mais fonctionnalité de sécurité pas bien configuré, version de logiciel obsolète...

Pour s'en prémunir, il est recommandé de s'entourer de professionnel de la sécurité pour le développement d'une application, et d'instaurer des processus d'installation sécurisés pour limiter tous les risques.

## IV. Responsive design

L'application est entièrement responsive, c'est à dire qu'elle peut également être utilisée sur une tablette ou un téléphone. C'est une caractéristique importante, car selon [similarweb.com<sup>1</sup>](https://www.similarweb.com/fr/platforms/), en 2023, les mobinautes représentaient 66,05% du trafic contre 32,09% pour les internautes classiques. Il est donc essentiel de permettre à ces utilisateurs qui occupent une part conséquente du trafic de naviguer de manière ergonomique.

Pour cela, j'ai utilisé des média queries : un média query est un module de CSS3 qui permet de changer et d'adapter le contenu d'une page web en fonction des caractéristiques de l'appareil de l'utilisateur.

J'ai mis en place des médias queries sur des éléments qui posaient problème lors de la réduction de la taille de l'écran, tel que les tableaux, qui prennent beaucoup de place.

```
2328 @media (max-width: 768px){  
2329     #responsive-table thead {  
2330         display: none;  
2331     }  
2332     #responsive-table tr{  
2333         display: block;  
2334         margin-bottom: 40px;  
2335     }  
2336 }  
2337 }
```

Ici, à partir de 768px de large, certaines règles de la structure du tableau sont changées et ce dernier s'adapte beaucoup mieux au format téléphone.

## V. Référencement naturel

Le référencement naturel, abrégé SEO en anglais ( Search Engine Optimization ), consiste à utiliser un ensemble de techniques pour améliorer la position d'un site web sur les pages de résultat d'un moteur de recherche.

Pour parvenir à cet objectif, j'ai structuré mes pages et utilisé des éléments optimisés pour le référencement naturel.

J'ai utilisé des balises h1 à h6 pour structurer mon contenu et permettre au moteur de recherche de mieux en comprendre la hiérarchie, en passant dans ces balises des mots clés importants, comme ici le nom du barbershop :

```
11 <h1> {{barbershop.nom}} </h1>
```

J'ai aussi renseigné l'attribut alt des balises image : cet attribut permet de fournir un texte alternatif, qui facilite la compréhension aux robots d'indexation de ce qu'une image représente, et optimise de ce fait le référencement.

<sup>1</sup> <https://www.similarweb.com/fr/platforms/>, Part de marché du trafic Mobile vs. Desktop vs. Tablette , consulté le 14/09/2023

D'autre part , cet attribut permet d'améliorer l'accessibilité du site, car ce texte s'affiche si l'image ne charge pas, et aide les lecteurs d'écran à transmettre des images.

```
9 |     
```

Les liens internes et externes sont également des outils efficaces pour augmenter son référencement naturel. J'en ai placé de part et d'autre de l'application.

Les liens internes redirigent vers des pages internes au site, par exemple les conditions d'utilisation, les articles, tandis que les liens externes redirigent vers des sites extérieurs : par exemple Facebook, LinkedIn...

De plus, le responsive design, que l'on a vu précédemment, est favorisé par Google depuis 2016 et son algorithme « Google Mobile Friendly ». Un site qui n'est pas responsive design, fait fuir ses utilisateurs mobiles et Google comprend que le site ne plait pas, ce qui fait baisser le positionnement.

Pour finir, j'ai mis en place un fichier sitemap.xml. C'est un fichier qui joue un rôle important dans le référencement, car il permet aux moteurs de recherche de trouver les pages du site, de les explorer et de les classer, mais aussi de comprendre leur structure, ce qui facilite la navigation.

Il est généré dynamiquement dans le SitemapController, en récupérant chaque URL qui compose le site.

```
27 |     $urls[] = ['loc'    => $this->generateUrl('app_contact')];
28 |
29 |     foreach($barbershopRepository->findAll() as $barbershop){
30 |         $urls[] = [
31 |             'loc'    => $this->generateUrl('show_barbershop', ['slug' => $barbershop->getSlug()]),
32 |             'lastmod' => $barbershop->getCreationDate()->format('Y-m-d')
33 |         ];
34 |
35 |     }
```

On voit sur ce morceau de code que l'URL de la route app\_contact est récupérée dans un tableau \$urls, puis on ajoute également les routes qui mènent à tous les barbershop ainsi que leur date de création grâce à une boucle foreach.

Il suffit ensuite de boucler sur le tableau contenant toutes les URL pour générer les balises XML correspondantes dans une vue.

```

5  {% for url in urls %}
6    <url>
7      <loc>{{ hostname }}{{url.loc}}</loc>
8
9      {% if url.lastmod is defined %}
10     <lastmod>{{url.lastmod}}</lastmod>
11
12     {% endif %}
13
14     <changefreq>monthly</changefreq>
15
16     <priority>0.8</priority>
17
18   </url>
19
20  {% endfor %}

```

La balise `<loc>` est obligatoire : elle correspond à la route vers la page.

On retrouve 3 autres balises facultatives :

`<lastmod>`, qui correspond à la date de dernière modification.  
`<changefreq>`, qui correspond à la fréquence de modification de la page.  
`<priority>`, qui permet d'indiquer le degré d'importance d'une page de 1 à 0.

Ce fichier est maintenant accessible à l'adresse `/sitemap.xml` (voir annexe 7) et permet aux robots de référencement de mieux comprendre le site.

## VI. API Platform

### 1. Mise en place

Il me semblait intéressant de partager la liste des barbershop que j'ai constituée avec d'autre développeurs, qui pourraient alors créer leur propre application à partir de ces données. Pour ce faire, j'ai installé API Platform sur mon application, un framework permettant la mise en place simple d'une API REST.

Une API REST est une API qui respecte les contraintes du style d'architecture REST (Representational State Transfer). Les principes de conception sont :

- Interface uniforme : les demandes d'API doivent avoir la même apparence quelle que soit l'origine de la demande.
- Découplage client-serveur : les applications client et serveur doivent être totalement indépendantes les unes des autres.
- Sans état : chaque demande doit inclure les informations nécessaire à son traitement. Pas de session côté serveur.
- Mise en cache : pouvoir être mise en cache du côté client ou serveur, dans la mesure du possible.

- Architecture système en couches : doivent être conçus de manière à ce que ni client ni serveur ne sachent s'ils communiquent avec l'appli finale ou un intermédiaire.
- Code à la demande : si la réponse donnée contient du code, il ne doit être exécuté qu'à la demande.

Une fois API Platform installé, j'ai configuré les données que je voulais que l'API renvoie. J'ai choisi de ne partager que la liste des barbershops.

Pour cela il suffit de créer une annotation API RESSOURCE au dessus de l'entité que l'on veut rendre accessible par l'API. Je lui passe en argument les méthodes que j'autorise, ici seulement la méthode get, pour récupérer un barbershop ou la liste complète. Je n'ai pas voulu rendre possible la création, la mise à jour ou encore la suppression de barbershops aux utilisateurs de l'API.

```

21 #[ApiResource(
22   operations: [
23     new Get(),
24     new GetCollection()
25   ],

```

On peut ensuite définir les attributs de l'entité que l'on veut partager. J'ai choisi de partager les données basiques des barbershops, celles qui seront utiles aux utilisateurs de l'API, comme le nom, les coordonnées, les horaires... je n'ai pas autorisé les attributs moins intéressants, tels que le slug ou l'état de validation du salon.

Pour ce faire, je commence par créer deux groupes : un groupe en normalizationContext, ce qui correspond aux paramètres de sérialisation des données envoyées par l'API et un groupe en denormalizationContext, qui correspond aux paramètres de désérialisation lorsqu'un utilisateur ajoute des informations à l'application via l'API. Je n'utiliserai pour le moment que le groupe normalizationContext, mais denormalizationContext peut servir au cas où l'API venait à évoluer.

```

26 // On affiche que les données en groups read lors d'une requête GET
27 normalizationContext: ['groups' => ['read']],
28 denormalizationContext: ['groups' => ['write']],

```

Au sein de normalizationContext je crée un groupe read qui servira à regrouper les attributs que je veux partager en lecture à l'API. J'annote ensuite les attributs choisis en spécifiant qu'ils font partie du groupe read.

```

67 #[Groups('read')]
68 #[ORM\Column(type: Types::TEXT)]
69 private ?string $horaires = null;

```

Enfin, on peut mettre en place une dernière annotation qui nous permettra d'effectuer des recherches avec l'API, grâce à ApiFilter :

```

30 #[ApiFilter(SearchFilter::class, properties: ['nom' => 'partial', 'ville' => 'partial'])]

```

Cette dernière nous permet de faire des recherches basées dans mon cas sur le nom et sur la ville. L'utilisation de « partial » correspond à une requête SQL qui serait effectuée sous la forme de LIKE %TEXT%, (le champ recherché doit contenir « text ».)

L'API est maintenant prête à être consommée, nous pouvons passer à la création d'une petite application pour donner un exemple d'utilisation.

## 2. Exemple d'utilisation

Pour consommer cette API, j'ai créé une application très simple proposant un système de recherche par texte ou par ville, ainsi que la possibilité de voir les détails d'un barbershop en cliquant sur un bouton.

Je commence par créer un champ de saisie de texte ainsi qu'un sélect contenant des options de ville qui correspondent aux différentes villes des barbershops.

Je peux ensuite mettre en place les requêtes vers l'API. Je commence par créer un événement on-change sur le sélect et le champs de texte, qui déclenchera la fonction à chaque fois qu'un changement sera détecté sur ces champs.

Je récupère ensuite les valeurs des champs à la ligne 6 et 7 et j'initialise une variable contenant l'URL vers l'API à la ligne 9.

```
4      $("#searchbar , #selectCity").on('input change', function () {  
5  
6          var barberName = $('#searchbar').val();  
7          var barberCity = $("#selectCity").val();  
8          // On définit l'URL  
9          var apiUrl = 'http://127.0.0.1:8000/api/barbershops?ville=strasbourg&nom=barber'  
10
```

On met ensuite en place la logique qui permet de compléter la base d'URL en fonction des champs remplis par l'utilisateur. La recherche peut se faire dans n'importe quel ordre, sélectionner une ville puis taper un nom, l'inverse, ou seulement un seul des deux critères. L'URL complète se compose de cette façon pour une recherche avec deux critères :

'<http://127.0.0.1:8000/api/barbershops?ville=strasbourg&nom=barber>'

Le premier critère est ajouté avec un point d'interrogation tandis que le deuxième est ajouté avec une esperluette. Pour ne pas se retrouver avec une URL invalide qui comporterait deux points d'interrogation ou deux esperluettes, il faut donc tester si l'URL comprend déjà un point d'interrogation.

```

11     // Si value de searchbar ou select City pas vide on l'ajoute a l'URL
12     if ($('#searchbar').val() != "") {
13         apiUrl += (apiUrl.includes('?') ? '&' : '?') + 'nom=' + barberName;
14     }
15
16     if ($('#selectCity').val() != "") {
17         apiUrl += (apiUrl.includes('?') ? '&' : '?') + 'ville=' + barberCity;
18     }

```

Si elle en comporte un, c'est qu'un critère a déjà été ajouté à l'URL de base, on ajoute donc le deuxième critère avec un « & ». Si non, c'est que l'URL de base n'a pas été modifiée, dans ce cas on ajoute le critère avec un « ? ».

L'URL est maintenant prête, il ne reste plus qu'à effectuer la requête AJAX vers l'API pour récupérer les informations des barbershops.

```

20     // On envoie la requête
21     $.ajax({
22         url: apiUrl,
23         method: "GET",
24         dataType: "json",
25         success: function (data) {
26             // Sélection de la div qui contiendra les résultats
27             var resultsDiv = $(".result");
28             // On la vide à chaque recherche
29             resultsDiv.empty();
30             if ($.isEmptyObject(data)){
31                 var notFound = $("<p> Aucun élément ne corresponds à votre recherche. </p>");
32                 resultsDiv.append(notFound);
33             }

```

On construit une requête AJAX basique en passant l'URL de l'api précédemment construite en paramètre. Je sélectionne ensuite le container qui contiendra les résultats et je le vide à chaque recherche.

On fait ensuite face à deux cas. Si la réponse de la requête est vide, j'ajoute à mon container un message indiquant qu'aucun élément ne correspond à la recherche.

```

34     else{
35         // Pour chaque résultat, on l'affiche dans result
36         $.each(data, function (index, barber) {
37             var barberCard = $("<div class='barber-card'></div>");
38             var h1 = $("<h1>" + barber.nom + "</h1>");
39             var img = $("<img src='" + barber.photo + "' class='barber-img'>");
40             var adresse = $("<p><i class='fa-solid fa-location-dot'></i>&ampnbsp" + barber.adresse + ", " + barber.ville + "</p>");
41             var button = $("<a href=''" + barber.id + "'> Découvrir </a>");
42             barberCard.append(h1);
43             barberCard.append(adresse);
44             barberCard.append(img);
45             barberCard.append(button);
46             resultsDiv.append(barberCard);
47         });
48     }

```

Sinon, je boucle sur chacun des éléments que je récupère. Chaque élément représente un barbershop. Je récupère les informations dans différentes variables que je vais ajouter à une div barberCard, dont un lien « Découvrir », ligne 41, qui contient l'id du barber en attribut universel.

J'ajoute ensuite chaque barberCard au container des résultats.

Pour voir le détail d'un barbier, je procède à peu de chose près comme pour afficher tous les barbiers.

```

56 // DETAIL D'UN BARBER
57 $(document).on('click', '.discover', function (e) {
58     e.preventDefault();
59     var resultsDiv = $(".result");
60     // On récupère l'id du barbier dans le data-id du lien
61     var barberId = $(this).data('id');

```

J'utilise une délégation d'évènement ligne 57 sur le lien de classe discover, ce qui permet de mettre un écouteur d'évènements sur un élément chargé dynamiquement.

Je récupère l'id du barbier à partir des attributs universels du lien à la ligne 61. Cela me permet de construire la requête pour récupérer les informations du barbier dont j'ai besoin.

```

64 // On refait une requete pour recuperer seulement les donnees du barbier selectionne
65 $.ajax({
66     url: 'http://127.0.0.1:8000/api/barbershops/' + barberId,
67     method: "GET",
68     dataType: "json",
69     success: function (data) {
70         resultsDiv.empty();
71         var barberDetail = $("<div class='barber-detail'></div>");
72         var h1 = $("<h1>" + data.nom + "</h1>");
73         var img = $("<img src='" + data.photo + "' class='barber-img'>");
74         var desc = $("<p>" + data.description + "</p>");
75         var adresse = $("<p><i class='fa-solid fa-location-dot'></i>&ampnbsp" + data.adresse + ", " + data.ville + "</p>")
76         barberDetail.append(h1);
77         barberDetail.append(adresse);

```

Je passe à l'URL de base de l'API l'id du barbier sélectionné pour pouvoir récupérer uniquement ses informations. Puis sur le même principe que la requête pour récupérer tous les barbiers, j'efface la div contenant tous les résultats et j'ajoute les détails du barbier choisi.

Exemple de recherche :

## API BarberHub



**Barba Rossa**  
📍 19 Rue Sainte-Madeleine, Strasbourg



**James Le Barbier**  
📍 36 Rue Jacques Kabilé, Strasbourg



**PERFECT BARBERSHOP**  
📍 26 Rue des Carmes, Strasbourg

Découvrir

Découvrir

Découvrir

## VII. Situation de travail ayant nécessité une recherche

### 1. Situation

Lors de la conception de cette application, j'avais la volonté de mettre en place une carte interactive recensant tous les barbershops rentrés en base de données, avec la possibilité d'ouvrir un popup au clic, de naviguer entre ceux-ci, de personnaliser les icônes.. La solution qui se prête le mieux à cette fonctionnalité est Leaflet, une bibliothèque javascript de cartographie libre de droit.

Pour installer cette carte dans mon projet, je me suis appuyé sur la documentation officielle de leaflet, qui proposait un tutoriel de démarrage rapide en anglais.

#### **Voici un extrait de ce tutoriel :**

« *By default (as we didn't pass any options when creating the map instance), all mouse and touch interactions on the map are enabled, and it has zoom and attribution controls.*

*Before writing any code for the map, you need to do the following preparation steps on your page:*

- *Include Leaflet CSS file in the head section of your document:*
- *Include Leaflet JavaScript file after Leaflet's CSS:*
- *Put a div element with a certain id where you want your map to be:*
- *Make sure the map container has a defined height, for example by setting it in CSS:*
- *Now you're ready to initialize the map and do some stuff with it.*

*Note that the setView call also returns the map object – most Leaflet methods act like this when they don't return an explicit value, which allows convenient jQuery-like method chaining.*

*Next, we'll add a tile layer to add to our map, in this case it's a OpenStreetMap tile layer. Creating a tile layer usually involves setting the URL template for the tile images, the attribution text, and the maximum zoom level of the layer. OpenStreetMap tiles are fine for programming your Leaflet map, but read the Tile Usage Policy of OpenStreetMap if you're going to use the tiles in production.*

*Make sure all the code is called after the div and leaflet.js inclusion. That's it! You have a working Leaflet map now.*

*It's worth noting that Leaflet is provider-agnostic, meaning that it doesn't enforce a particular choice of providers for tiles. Also, Leaflet doesn't even contain a single provider-specific line of code, so you're free to use other providers if you need to.»*

### 2. Traduction en français

Avant d'écrire le moindre code pour la carte, vous devez suivre ces étapes de préparation sur votre page :

- Inclure le fichier CSS de Leaflet dans la section Head de la page
- Inclure le fichier Javascript de Leaflet après le fichier CSS de Leaflet
- Créer une div avec un id défini à l'endroit où vous voulez mettre votre carte
- Assurez-vous que le conteneur de la carte a une hauteur définie, en lui appliquant du CSS par exemple.
- Vous êtes maintenant prêt à initialiser la carte et à faire des choses avec.

Par défaut (comme nous n'avons pas passé aucune option en créant l'instance de la carte), toutes les interactions à la souris et tactiles sont activées, et il y a un crédit et un niveau de zoom par défaut.

Notez que l'appel de `setView` retourne aussi l'objet de la carte - la plupart des méthodes de Leaflet agissent comme cela, quand elles ne retournent pas une valeur explicite, ce qui permet une méthode de chaînage pratique ressemblant à jQuery.

Ensuite, nous allons ajouter une couche de tuile à notre carte. Dans ce cas, c'est une couche de tuile d'OpenStreetMap. Créer une couche de tuile requiert habituellement de paramétriser la template d'URL pour les images de tuiles, puis le crédit, et le niveau de zoom maximum de la couche.

Les tuiles d'OpenStreetMap sont très bien pour programmer votre carte Leaflet, mais lisez la Politique d'Usage des Tuiles d'OpenStreetMap si vous allez utiliser les tuiles en production.

Assurez-vous que tout le code est appelé après la div et l'inclusion de `leaflet.js`. C'est tout ! Vous avez maintenant une carte Leaflet qui fonctionne.

## VIII. Axes d'amélioration

En l'état, l'application est fonctionnelle et sécurisée. Toutefois, des améliorations peuvent être apportées afin d'améliorer l'expérience utilisateur.

Au niveau de la prise de rendez-vous, la possibilité d'ajouter plusieurs prestations pour l'utilisateur, et de définir des jours de fermeture exceptionnels pour le barbier, sont des fonctionnalités que j'aimerais ajouter. Les relations entre les tables et la base de données sont prêtes pour cet ajout.

La mise en place d'un système de notification SMS, avec un rappel envoyé la veille du rendez-vous, serait très intéressante pour éviter les oubliés. Les recherches que j'ai menées sur ce sujet m'ont conduit à des solutions payantes, notamment l'API Twilio, raison pour laquelle je n'ai pas pu la mettre en place pour le moment.

Enfin, je n'ai pas pu mettre en place la partie E-Commerce de mon application, par manque de temps. C'était une partie importante à mes yeux mais j'ai préféré peaufiner la partie prise de rendez-vous et gestion de planning au détriment de la mise en place de la boutique. C'est donc une amélioration que je mettrai en place dans le futur, en utilisant l'API de paiement Stripe.

Une application est amenée à évoluer constamment, et il y a beaucoup d'améliorations possibles mais ces quatre-là sont pour moi les améliorations prioritaires à mettre en place.

## D. Conclusion

Ce projet s'est avéré très formateur. J'ai pu découvrir beaucoup de notions, que ce soit en back-end ou en front-end, et améliorer mes compétences.

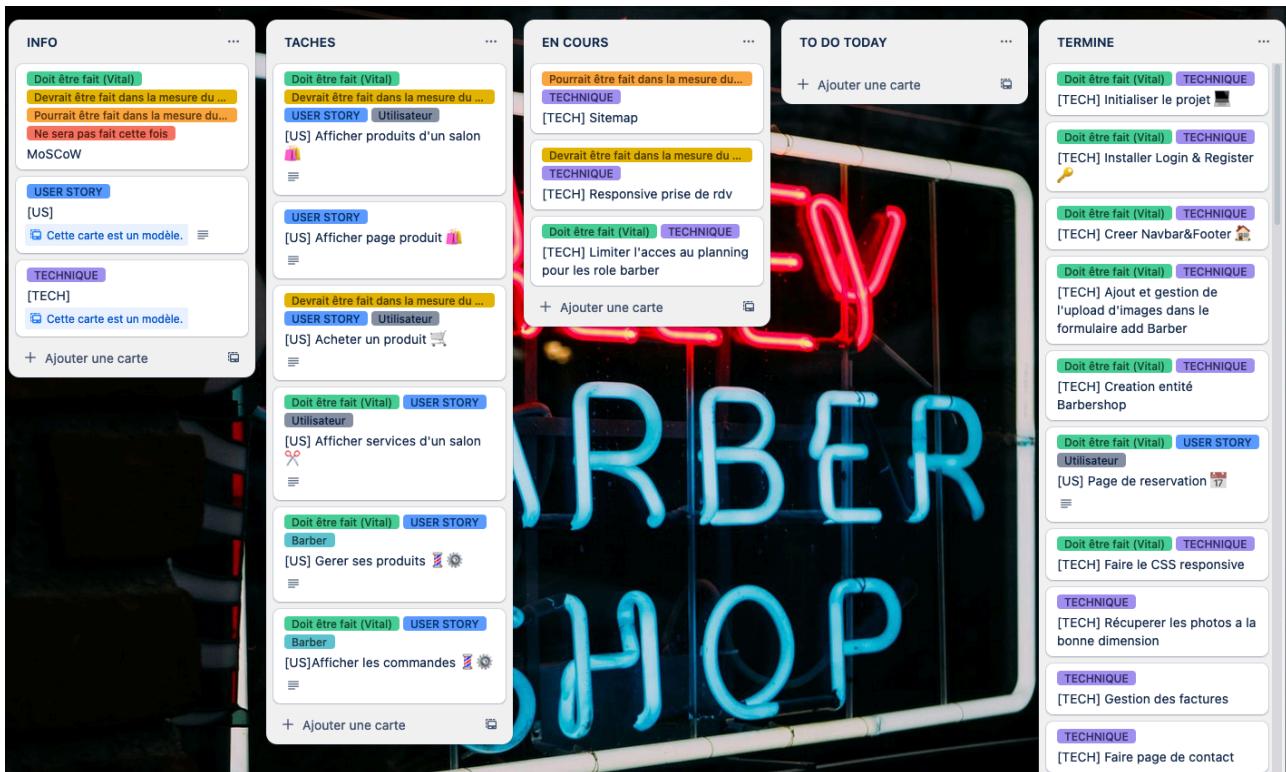
J'ai fait face à des difficultés, qui m'ont finalement été bénéfiques car elles m'ont poussé à continuer de chercher et à trouver de nouvelles solutions. J'ai également pu expérimenter la gestion d'un projet comme celui-ci sur plusieurs mois, la manière de s'organiser et la complexité de définir le temps pour mettre en place une fonctionnalité, qui font partie intégrante du métier de développeur.

Ce fut également un réel plaisir de voir jour après jour ce projet grandir et prendre forme.

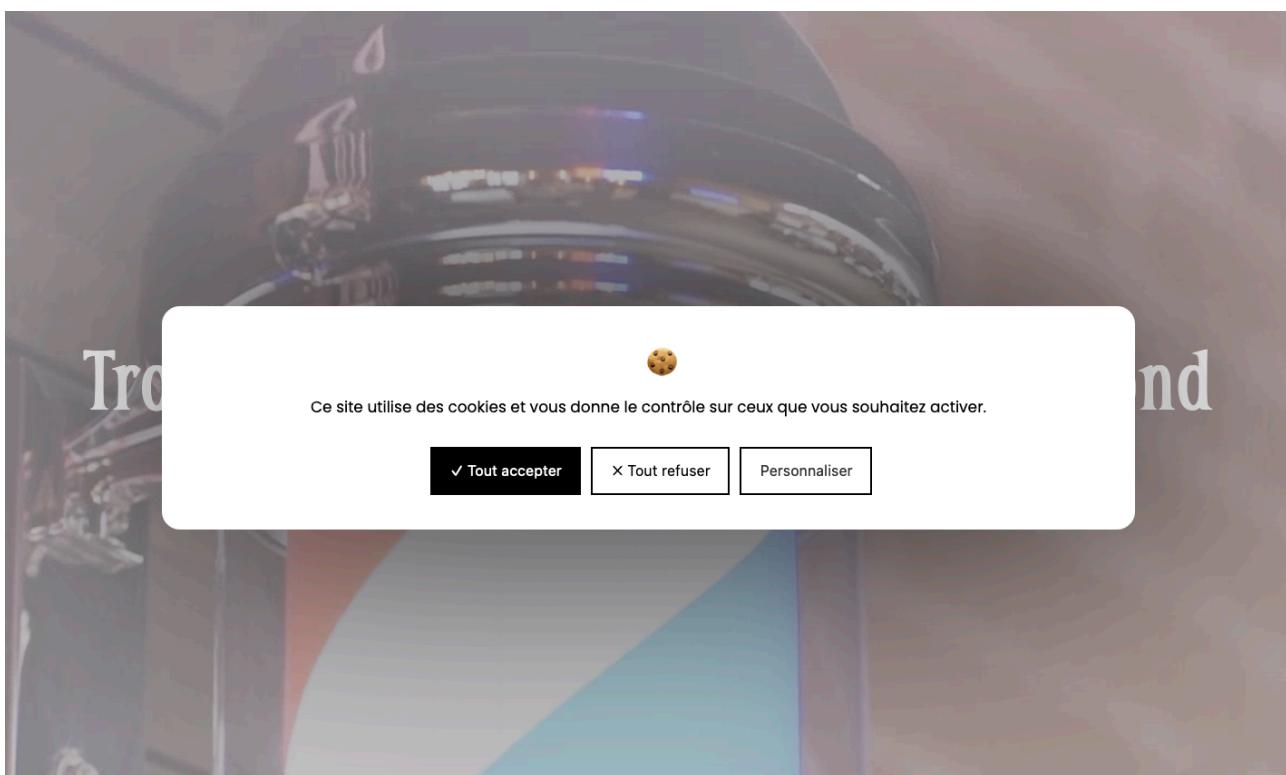
Ces mois passés en formation à coder m'ont conforté dans mon souhait de devenir développeur. Dans cette optique, je suis actuellement en Bachelor Développeur Concepteur D'Application au CESI de Strasbourg, en alternance au sein de l'agence web et mobile Adipso, dans laquelle j'occupe un poste de développeur fullstack.

# E. Annexes

Annexe 1 : tableau Trello de gestion du projet.



Annexe 2 : fenêtre de tarteaucitron.js permettant de gérer la collecte des cookies.



Annexe 3 : maquette mobile de la page d'accueil de l'application.

The image consists of three side-by-side screenshots of a mobile application interface:

- Screenshot 1 (Left):** A dark-themed search screen with a magnifying glass icon at the top left. The main text "TROUVEZ LE BARBIER QUI VOUS CORRESPONDS." is displayed over a background image of a vintage barber chair. A button labeled "Trouver mon barbier" is at the bottom.
- Screenshot 2 (Middle):** A search results screen titled "Tous les barbiers". It features a search bar, dropdown menus for "Rechercher..." (Tous) and "Toutes les villes", and a list item for "Vintage Barber" with a 5-star rating and a small image of the shop interior.
- Screenshot 3 (Right):** A detailed view of a specific barbershop. At the top is a banner for "Eli's Barber Shop" with a location pin and a "J'aime" button. Below it are sections for "Informations" (with placeholder text about the shop), "Prestations" (listing services like "Dégradé - 30€" with a "Reserver" button), and "Emplacement" (a map showing the shop's location in Strasbourg).

Annexe 4 : interface graphique de l'affichage des créneaux de rendez-vous.

#### Votre rendez vous :

Le Peigne Fin • 117 Route du Polygone, Strasbourg •

#### Votre prestation :

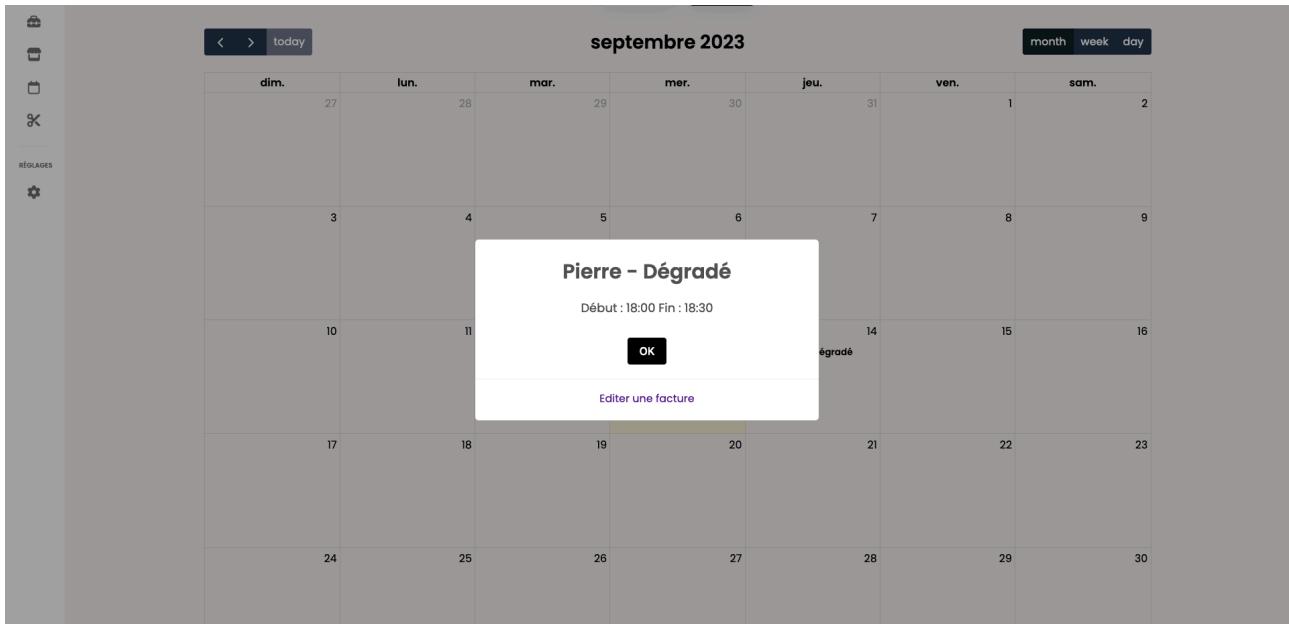
Taille de barbe • 30min • 20€

#### Choisissez un professionnel :

ADMIN	<input type="radio"/>	Pierre	<input checked="" type="radio"/>
-------	-----------------------	--------	----------------------------------

Mercredi 13 sept.	Jeudi 14 sept.	Vendredi 15 sept.	Samedi 16 sept.	Lundi 18 sept.
17:00	09:30	09:00	09:00	10:00
17:30	10:00	09:30	09:30	10:30
18:00	10:30	10:00	10:00	11:00
<b>18:30</b>	<b>11:00</b>	<b>10:30</b>	<b>10:30</b>	<b>11:30</b>
	11:30	11:00	11:00	12:00
	12:00	11:30	11:30	12:30
	12:30	12:00	12:00	13:00
	13:00	12:30	12:30	13:30
	13:30	13:00	13:00	14:00

Annexe 5 : Pop-up avec détail du rendez-vous.



Annexe 6 : exemple de facture pour un rendez-vous.

Le Peigne Fin		Pierre
117 Route du Polygone		pierre@barberhub.com
Strasbourg, 67100		Rendez vous du : 14/09/2023 18:00
Prestation	Prix	
Dégradé - par Pierre	35€	
<b>Total: 35 €</b>		

 Le 13/09/2023

*Annexe 7 : extrait du fichier sitemap.xml*

```
-<urlset>
-<url>
  <loc>http://127.0.0.1:8000/</loc>
  <changefreq>monthly</changefreq>
  <priority>0.8</priority>
</url>
-<url>
  <loc>http://127.0.0.1:8000/article</loc>
  <changefreq>monthly</changefreq>
  <priority>0.8</priority>
</url>
-<url>
  <loc>http://127.0.0.1:8000/barbershops</loc>
  <changefreq>monthly</changefreq>
  <priority>0.8</priority>
</url>
-<url>
  <loc>http://127.0.0.1:8000/map</loc>
  <changefreq>monthly</changefreq>
  <priority>0.8</priority>
</url>
-<url>
  <loc>http://127.0.0.1:8000/monespace</loc>
  <changefreq>monthly</changefreq>
  <priority>0.8</priority>
</url>
...
```