Novembre 2023

ELAN Formation

Douceur de Lune

Dossier de synthèse

Pour le titre professionnel de
développeur web - web mobile

Guillaume KEZIC

TABLE DES MATIÈRES

I. INTRODUCTION

1 A Remerciements 4
1 B Reconversion à ELAN 5
1 C Contexte.....6

II. LE PROJET

1. Présentation du projet
1 A L'objectif du site web 7
1 B Les fonctionnalités du site 8
1 C Les technologies utilisées 9

2. Compétences couvertes par le projet
2 A Compétences Back-end travaillées 13
2 B Compétences Front-end travaillées.....13
2 C Compétences transversales travaillées 14

3. Design et ergonomie
3 A UI / UX 14
3 B Le maquettage, zoning et wireframe 15
3 C Inspiration et couleurs 16
3 D La typographie 17
3 E L'arborescence 19
3 F La partie responsive 19

4. Le RGPD 23

5. Méthodes et conceptualisation des données
5 A Les méthodes : Merise, TrelloMoSCoW 25
5 B MCD 29
5 C MLD 31

6. L'architecture du projet
6 A Le design pattern MVC 32

6 B La couche modèle et le rôle de Doctrine 35
6 C Le contrôleur 37
6 D La vue 38

III. Construction du site

1. Fonctionnalités du site
1 A Inscription et connexion 40
1 B Compte utilisateur.....41
1 C Les cours et la présentation 43
1 D Le calendrier de réservation.....45
1 E Le système de réservation 57
1 F Le système de payement.....59
1 G Espace administrateur.....65

IV. SEO ET SÉCURITÉ

1. Référencement SEO 67

2. Les failles
2 A Failles XSS 68
2 B Failles CSRF 71
2 C Failles dictionnaire/force brute 72
2 D Injection SQL 74

V TRADUCTIONS

1 A TRADUCTION 76

VI AMÉLIORATIONS ET CONCLUSION

1. AMÉLIORATIONS
1 A Améliorations à court terme 78
1 B Améliorations à moyen terme 79
1 C Améliorations à long terme 79

2. CONCLUSION 80

I. INTRODUCTION

1 A Remerciements

À l'issue de ma formation, je souhaite exprimer ma gratitude à toutes les personnes qui ont contribué à faire de ce parcours un succès.

Je tiens à commencer par remercier mes formateurs, Stéphane Smail, Mickael Murmann et Quentin Mathieu. Votre aide, vos conseils pertinents et votre soutien continu ont été essentiels tout au long de ce chemin. Votre patience et votre engagement m'ont permis de rester motivés, surtout dans les moments de doute. Votre gentillesse et votre disponibilité a rendu ce parcours plus agréable également.

Je voudrais également remercier l'équipe de DAB Distribution qui m'ont permis d'avoir un stage pendant cette formation. Durant ce stage, j'ai acquis de nouvelles compétences. L'ambiance a été motivante et m'ont permis de passer un stage instructif et agréable.

Un grand merci à ma soeur, Laetitia. Tes conseils ont été précieux et tes encouragements m'ont aidé à rester motivé. Ta présence a vraiment fait la différence dans cette formation. Nos échanges concernant ta volonté sur ce que tu voulais pour ce site ont parfois été compliqués à intégrer dans un projet principalement destiné à un examen DWWM, mais je pense que nous avons fait du mieux possible pour concilier les 2.

Également a tous les autres étudiants DWWM de chez ELAN formation, l’ambiance a été très agréable du début à la fin, que ça soit dans les locaux d’ELAN mais également en dehors (notamment dans les bars !).
L’entraide n’a jamais été un problème bien au contraire, et ça a sans aucun doute, eu un impact positif sur cette formation.

Pour conclure, un merci sincère à tous ceux qui ont participé de près ou de loin à cette année pleine d'apprentissage et de découverte. Chacun d'entre vous a joué un rôle pour m'aider à atteindre ce stade, et pour cela, je vous suis profondément reconnaissant.

I. INTRODUCTION

1 B Reconversion et mon bilan chez ELAN Formation.

Suite à mon intérêt grandissant pour le domaine du web, en particulier le front-end, j'ai pris la décision de me réorienter professionnellement. Pour amorcer cette transition, je me suis auto-formé pendant deux mois à la maison aux langages HTML et CSS. Ce premier pas a déclenché mon désir de m'inscrire chez ELAN Formation.

En tant que novice complet en développement, l'étape préparatoire de deux mois offerte par ELAN Formation m'a permis de conforter mon choix et d'acquérir les bases nécessaires pour intégrer la suite du programme.
Le début de la formation ne fut pas de tout repos, notamment à cause de mon manque d'expérience en algorithme. Cependant, grâce à la pédagogie adaptée des formateurs, j'ai pu progresser à un rythme soutenu sans trop de difficultés.

Malheureusement, suite à un accident qui a résulté en une triple fracture de la jambe et la rupture d'un ligament croisé, j'ai dû poursuivre la formation à distance pendant deux mois. Les premières semaines après l'accident ont été extrêmement difficiles, au point où j'ai envisagé d'abandonner. La fatigue engendrée par l'accident m'empêchait de me concentrer et d'assimiler les connaissances transmises durant les cours. Cependant, l'encouragement des formateurs, en particulier Stéphane, à qui je tiens à exprimer une nouvelle fois ma gratitude, m'a permis de persévérer. (merci d'avoir pris le temps un dimanche midi, pour m'aider au début de Symfony pendant ton repas !). J'ai également pris des heures de cours particuliers par la suite pour ne pas prendre de retard.

Heureusement, tout est rentré dans l'ordre après mon stage. J'ai également pu reprendre la formation en présentiel. Mon parcours chez Elan Formation a été semé d'embûches, mais je suis fier de ce que j'ai accompli et des compétences que j'ai acquises. Je ne regrette en aucun cas d'avoir entrepris cette reconversion.

Je souhaite d'ailleurs continuer sur cette lancée en m'orientant vers une formation plus axée sur le front-end pour compléter mes compétences en back-end.

I. INTRODUCTION

1 C Contexte

L'idée de ce projet est né de manière assez spontanée. Ma sœur possédant un site web pour son activité professionnelle, avait créé son site via un CMS.

Ayant toujours eu de mon côté envie de l'améliorer et du sien d'avoir un site entièrement réalisé par un être humain et non une machine (pour le côté artisanat et personnalisation etc..), j'ai décidé de transformer cette idée en projet concret.
J'ai ainsi choisi de développer un site que ma sœur pourrait également utiliser.

Nous avons donc échangé régulièrement afin que je puisse cerner ses besoins et ses attentes, comme je l'aurais fait avec un véritable client.

Par conséquent, j'ai également dû concevoir une interface d'administration intuitive. Mon objectif était de la rendre la plus simple possible pour permettre au client de réaliser lui-même les modifications de bases si besoin. (comme par exemple, la possibilité d'interdire la réservation pour un jour ou une plage horaire)

En ce qui concerne le choix technologique, j'ai opté pour JavaScript pour le développement du calendrier. C'était une décision guidée par mon intérêt pour ce langage que j'ai cherché à apprendre en parallèle de la formation. De plus, je rencontrais des difficultés à obtenir ce que je voulais avec FullCalendar.
Ce défi s'est finalement révélé bénéfique et m'a poussé à me dépasser.

J'ai d'ailleurs récemment commencé à découvrir et apprendre l'utilisation de React, pour continuer dans la lignée de l'apprentissage JavaScript et également pour mon intérêt pour le Front-end.

Pour conclure, ce projet a été une opportunité pour moi d'appliquer mes nouvelles compétences en développement et de me confronter à des problématiques réelles, de créer un projet en fonction des besoins d'un client, et de trouver des solutions face aux imprévus.

II. LE PROJET

1. Présentation du projet

1 A. L'objectif du site

Devenir mère, ou être une mère récente, représente un changement majeur dans la vie d'une femme. En plus d'être une femme, on devient également une mère. Ce rôle s'accompagne de transformations physiques et mentales, de doutes et de peurs, mais aussi d'un sentiment de bonheur, de satisfaction et d'amour. Ces changements peuvent créer des tourbillons émotionnels difficiles à gérer, ce qui est tout à fait normal.

C'est dans ce contexte que ce projet a vu le jour : la création d'un site web destiné aux jeunes parents, principalement les mamans, mais pas seulement.

Le site propose des cours de relaxation, de portage, des conseils et du bien-être, le tout dans un design moderne et accessible.

Ce site a été créé principalement pour valider ma formation DWWM. Cependant, il a également pour objectif d'être utilisé par ma sœur dans le cadre de son activité professionnelle. Actuellement, elle utilise un site réalisé en CMS, mais nous avons convenu qu'un site "fait main" serait plus avantageux à bien des égards.

C'est donc avec ces deux objectifs en tête que j'ai décidé de créer un site sur ce thème, qui répondrait aux exigences du REAC* tout en répondant aux besoins de ma sœur, la cliente.
J'ai beaucoup discuté avec elle pour comprendre ses besoins, le public visé, les habitudes des utilisateurs sur son site actuel, mais aussi sur ce qu'elle attend au niveau visuel et de la charte graphique.

Mon but était d'adapter au mieux ce projet pour elle, comme je le ferais pour un véritable client.

*https://www.banque.di.afpa.fr/EspaceEmployeursCandidatsActeurs/EGPResultat.aspx?ct=01280m03&type=t

II. LE PROJET

1 B. Les fonctionnalités du site

Le site offre aux visiteurs la possibilité de découvrir le parcours professionnel de de la cliente, de la contacter par mail, de consulter les différents cours proposés et de réserver un cours. Une interface d'administration est également mise à disposition pour le gestionnaire. Mais également d'accéder à une autre application, une boutique en ligne qui vendrait des produits en rapport avec les cours concernés (side-project présenté dans la partie amélioration)

En résumé, le site comprend les fonctionnalités clés suivantes:

- **Inscription et création de compte et connexion :** Les utilisateurs peuvent s'inscrire, créer un compte et se connecter.

- **Modification des informations de l'utilisateur :** Les utilisateurs ont la possibilité de modifier leur pseudo, leur adresse mail, téléphone ou leur mot de passe. Ainsi que supprimer leur compte.

- **Navigation :** Les utilisateurs peuvent naviguer à travers différentes pages pour en savoir plus sur le parcours de la professionnelle et le déroulement des différents cours.

- **Réservation de cours :** Les utilisateurs peuvent sélectionner un cours et le réserver via un calendrier.

- **Paiement en ligne :** Les utilisateurs ont la possibilité de payer en ligne directement avec Stripe, ou de régler sur place le jour du cour.

- **Contact et localisation :** Les utilisateurs peuvent contacter la cliente par mail et interagir avec une carte pour mieux visualiser l'emplacement des cours.

- **Interface d'administration :** Une interface a été créée pour permettre au gestionnaire d'ajouter ou de supprimer facilement un cours, ainsi que de gérer les jours et horaires des cours.

II. LE PROJET

1 C. Les technologies utilisées

HTML (Hypertext Markup Language) : HTML est le langage de balisage utilisé pour structurer et présenter le contenu d'une page web. Il permet de définir les éléments et les balises qui composent une page, tels que les titres, les paragraphes, les images, les liens hypertexte, les tableaux, etc.

CSS (Cascading Style Sheets) : CSS est un langage de feuille de style utilisé pour décrire la présentation et l'apparence d'un document écrit en HTML. Il permet de contrôler la mise en page, les couleurs, les polices de caractères, les marges, les bordures et d'autres aspects visuels des éléments d'une page web.

PHP (PHP: Hypertext Preprocessor) : PHP est un langage de programmation côté serveur conçu spécialement pour le développement web. Il est utilisé pour créer des sites web dynamiques et des applications web. PHP est souvent intégré dans des fichiers HTML et peut interagir avec des bases de données pour récupérer et stocker des informations.

JS (JavaScript) : JavaScript est un langage de programmation côté client utilisé pour ajouter des fonctionnalités interactives aux pages web. Il est principalement utilisé pour manipuler le contenu des pages web, gérer des événements et communiquer avec les serveurs pour mettre à jour dynamiquement les informations affichées.

SQL (Structured Query Language) : SQL est un langage de programmation utilisé pour communiquer avec les bases de données. Il permet de créer, modifier et interroger des bases de données, de manipuler des tables, d'insérer, mettre à jour et de supprimer des données, ainsi que d'exécuter des requêtes complexes pour extraire des informations spécifiques.

II. LE PROJET

En plus des langages associés j'ai utilisé des outils et des bibliothèques qui m'ont permis de réaliser le projet plus facilement et de gagner un certain temps tout en donnant un rendu très agréable aussi bien dans le code (Back-end) que dans les vus utilisateurs (Front-end).

Symfony : Symfony est un framework de développement PHP "open source" avec architecture MVC (Modèle-vue-contrôleur) qui vise à accélérer la création et a maintenance des applications web et à remplacer les tâches de codage récurrentes.

Comme une grande «bibliothèque de solutions efficaces et prêtes à l'emploi», Symfony regroupe de nombreux composants qui facilitent le développement
L'utilisation d'un tel framework est de :

- Limiter le codage fastidieux,
- Limiter le codage sans valeur ajoutée,
- Réduire le temps de développement,
- Guider le développeur et fiabiliser son travail.

Enfin, il possède une énorme communauté a travers le monde, cela fait de Symfony un des framework les plus utilisés, ce qui est un gage de qualité et de pérennité. (et c'est Français !)

- J'ai utilisé les gestionnaires de dépendances suivants, pour installer, mettre à jour et gérer les bibliothèques et dépendances nécessaire a ce projet :

Composer : Composer est un gestionnaire de dépendances pour les projets PHP. Il permet de gérer les bibliothèques et les dépendances externes d'une application PHP de manière simple et efficace. Il s'assure que toutes les dépendances sont correctement gérées.

npm : (Node Package Manager) est le gestionnaire de paquets par défaut pour les applications en JavaScript et les environnements basés sur Node.js. Il permet d'installer,et de mettre à jour les dépendances JavaScript d'un projet.

II. LE PROJET

- Les bibliothèques suivantes :

React.js : React est une bibliothèque JavaScript libre. Son but principal est de faciliter la création d'application web, en créant des composants dépendant d'un état et générant une page HTML à chaque changement d'état.

Leaflet : Leaflet est une bibliothèque JavaScript open-source utilisée pour créer des cartes interactives sur des sites web. Il permet d'afficher des cartes, de marquer des points d'intérêt, d'ajouter des couches personnalisées et de fournir des fonctionnalités de zoom et de déplacement. En plus de ça, Leaflet est très léger.

Beberlei : Beberlei est une bibliothèque PHP qui contient une collection d'extensions pour Doctrine. Il ajoute un grand nombre de nouvelles fonctions DQL (Doctrine Query Language) qu'on peut utiliser dans les requêtes. Ces fonctions incluent des fonctions de chaînes de caractères, de date et de numérique qui sont disponibles dans SQL mais pas dans DQL.

Doctrine : Doctrine est une suite d'outils PHP qui facilite l'interaction avec les bases de données. Doctrine est souvent utilisé comme couche d'abstraction de base de données (Database Abstraction Layer, DAL) et comme un outil de mappage objet-relationnel (Object-Relational Mapping, ORM).

- Le mode de payement en ligne Stripe, a été choisi pour ce projet :

Stripe : Stripe est une plateforme de paiement en ligne qui facilite le traitement des paiements sur les sites web et les applications. Elle offre des fonctionnalités de paiement sécurisé, de gestion des abonnements, de facturation et de gestion des transactions.

II. LE PROJET

- J'ai décidé d'utiliser EasyAdmin pour gérer l'interface administrateur.

EasyAdmin : EasyAdmin est une extension du framework Symfony, un outil de génération automatique d'interfaces d'administration pour les applications web. Il permet de créer rapidement des interfaces d'administration basées sur des entités (tables de base de données) en utilisant des fonctionnalités de CRUD (Create, Read, Update, Delete).

- Le projet a été réalisé sur l'éditeur de code VSC

VSC : VSC (Visual Studio Code) est un éditeur de code source développé par Microsoft. Visual Studio Code est gratuit, open source et disponible pour Windows, macOS et Linux.

- Enfin, les sites suivants on étés utilisés

Font awesome : Font Awesome est une bibliothèque de logos qui sont fournis sous forme de polices web. Ces icônes peuvent être facilement intégrées dans un site web et stylisées à l'aide de CSS.

Google Fonts : Google Fonts est une bibliothèque gratuite de polices de caractères hébergée par Google. Elle offre une large variété de polices open source que les développeurs et les concepteurs peuvent utiliser dans leurs projets web.

Midjourney : Midjourney est une intelligence artificielle, elle permet de générer et modifier des images à partir d'une description textuelle (prompt).

Dbdiagram.io : dbdiagram.io est un outil en ligne qui permet aux utilisateurs de créer, de visualiser et de partager des diagrammes de base de données. Il permet de dessiner des diagrammes de base de données en utilisant une syntaxe de script simple.

2. Compétences couvertes par le projet

2 A. Compétences Back-end travaillées

• **Élaborer le modèle conceptuel et logique de données :** MCD / MLD.

• **Concevoir et créer une base de données :** stockage organisé de données selon le schéma physique, gestion des données à l'aide d'un SGBD, maitrise des instructions de création, de modification et de suppression (CRUD).

• **Développer des composants d'accès aux données :** manipulation des données à l'aide du langage SQL et DQL (doctrine)

• **Développer la partie back-end d'une application web :** Développement de la partie back-end du site à l'aide de Symfony.

• **Respecter les normes de sécurité :** veille technologique sur la sécurité informatique et sur les vulnérabilités connues, consultation du site OWASP.

• **Intégrer Stripe :** Mise en place d'un système de payement avec l'api Stripe.

2 B. Compétences Front-end travaillées

• **Développer une interface utilisateur web static :** réalisation de plusieurs pages à l'aide des langages HTML, CSS (sans Bootstrap).

• **Développer une interface utilisateur web dynamique :** intégration de scripts événementiels avec un langage de script client (JavaScript). Ajout de composants en utilisant REACT.

• **Réalisation d'une interface adaptable :** utilisation de media queries pour rendre le site responsive.

• **Création d'un calendrier de réservation :** calendrier de réservation entièrement en HTML et JavaScript (sans Bibliothèque)

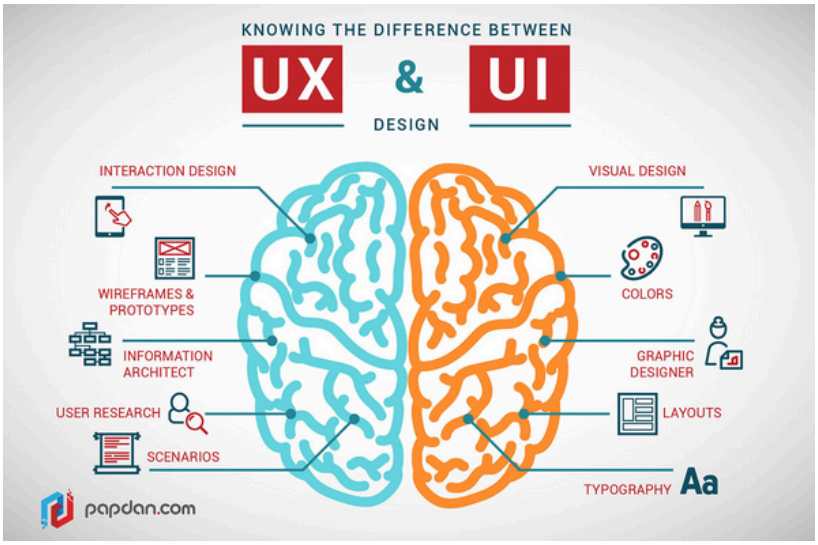
2 C. Compétences transversales travaillées

- **Utilisation de l'anglais durant l'activité professionnelle en développement web et web-mobile :** recherche d'information et de solutions en anglais ainsi que la lecture de documentations techniques officielle en anglais.

- **Mise en place d'un système de veille informationnelle :** actualiser et partager ses compétences en développement web et web mobile à l'aide de plusieurs sources.

- **JavaScript et REACT :** Apprentissage des bases JavaScript, nécessaire à la mise en place du calendrier de réservation, et de la constructions de composants intégrés au projet.

3. Design et ergonomie



3 A UI / UX

La réalisation de l'interface utilisateur (**UI**) et de l'expérience utilisateur (**UX**) me plaît beaucoup, et c'est entre autre cet intérêt qui m'a guidé vers la formation DWWM, puis je l'espère, probablement vers un parcours de formation en UI/UX Design (que j'explique à la fin du dossier de synthèse dans la partie conclusion). J'aime beaucoup la fusion entre esthétique et fonctionnalité pour offrir non seulement une interface visuellement agréable, mais aussi une navigation intuitive et fluide pour les utilisateurs.

Par exemple, dans mes projets, j'ai mis en œuvre l'**UI** en sélectionnant des couleurs en adéquation avec le thème du site, mais aussi en fonction des envies de la cliente en adoptant une hiérarchie visuelle claire, et en assurant une cohérence visuelle sur toutes les pages, pour une **UI** agréable et facile à comprendre.

Quant à l'**UX**, j'ai accordé une attention particulière à la conception des parcours utilisateurs, en rendant les formulaires le plus succincts et intuitifs possible et en veillant à ce que les informations les plus importantes soient aisément accessibles aux utilisateurs. Ces démarches sont essentielles pour assurer une interaction harmonieuse et une expérience utilisateur agréable.

Je vais dans la suite, montrer comment j'ai conçu le maquettage en prenant en compte ce soucis de l'**UI** et de l'**UX,** d'abord en montrant la maquette du projet avec différentes vues, ensuite le choix des couleurs et la typographie utilisée.

Enfin, un tableau montrant l'arborescence du site et pour finir, expliquer comment j'ai rendu celui ci responsive a l'aide des media queries.

3 B Zoning et maquettage

Lors de la phase initiale du projet, j'ai d'abord élaboré un zoning pour obtenir une vue d'ensemble et établir les fondements de l'architecture graphique.

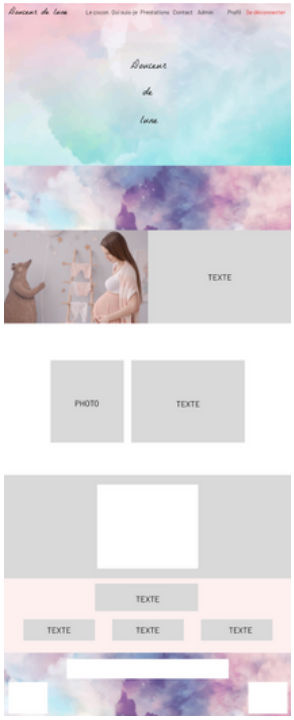
Bien que cet outil soit essentiel au processus créatif, j'ai choisi de ne pas l'inclure dans cette présentation pour des raisons d'espace et de pertinence. J'estime que le maquettage offre une vision plus concrète et révélatrice de la construction finale du site.

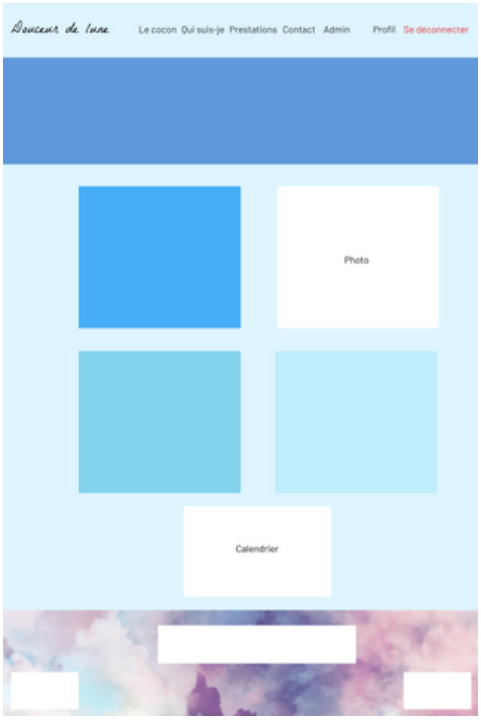
Contrairement au zoning, le mockup ajoute une couche de détails et de complexité en incluant des éléments tels que la couleur, la typographie, les icônes et même les images. Il sert de guide précis pour les développeurs et les designers, leur permettant de comprendre exactement comment chaque élément doit être codé et stylisé.

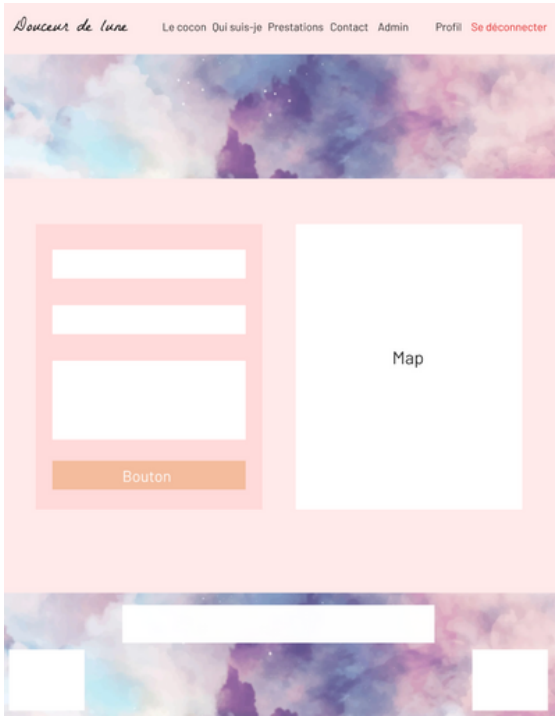
Sur les captures d'écran ci-dessous, j'ai par exemple décider de montrer respectivement les vues de ma page d'accueil, celle d'une page de présentation de cours et celle de ma page de contact.

On y voit également les couleurs et la typographie utilisée, ainsi que les images choisis en accord avec le client.

II. LE PROJET







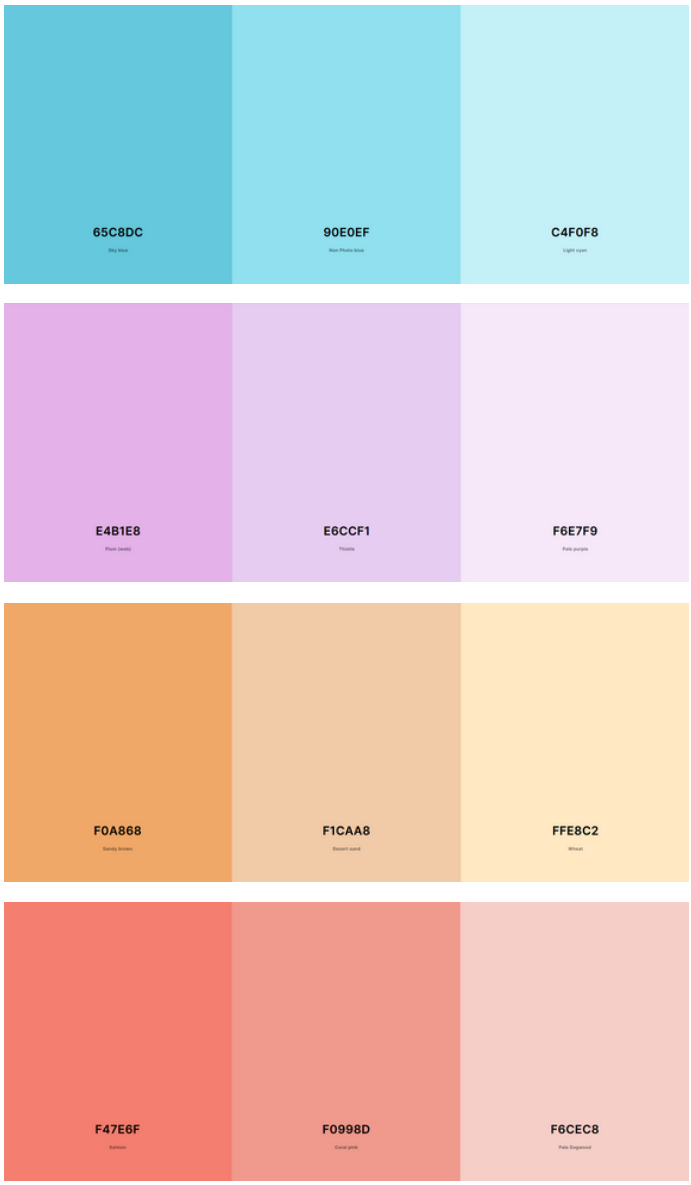
3 C Couleurs

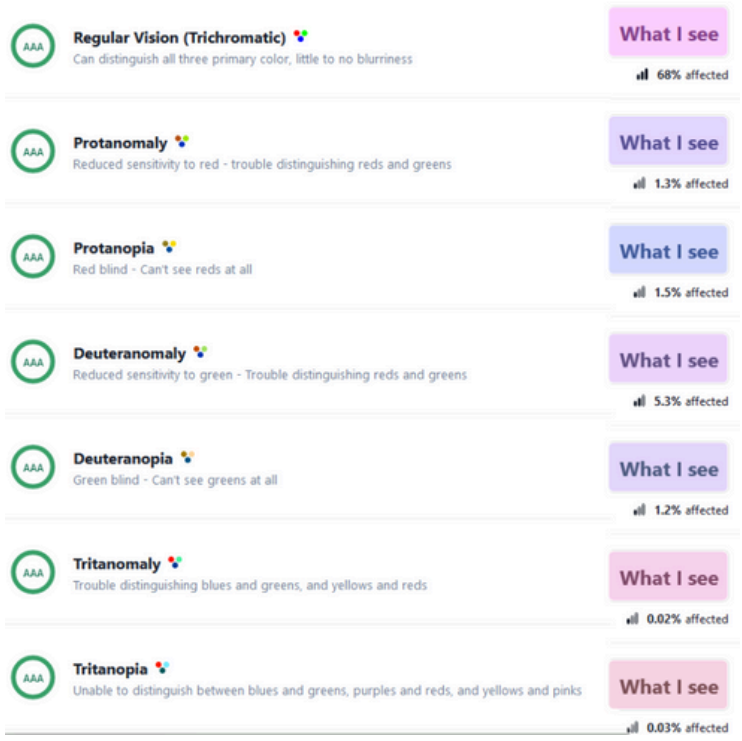
Pour la palette chromatique, j'ai opté pour des teintes pastel, transitionnant du froid au chaud.
Chaque teinte est déclinée en trois nuances distinctes (plus une version plus foncé pour les titres) pour enrichir la profondeur visuelle. L'intention étant d'éviter des couleurs trop "agressives" en privilégiant des couleurs douces et apaisantes, pour correspondre au thème du site.

De plus, chaque cours se distingue par sa propre couleur thématique, assurant ainsi une identification rapide et une cohérence sur la page de détail du cours mais aussi sur la page de présentation générale. Cela permet une meilleure expérience utilisateur, intuitive et agréable pour les visiteurs de notre plateforme.

Les couleurs ne sont pas simplement un choix esthétique, elles sont un outil de communication puissant qui peut influencer l'émotion et le comportement des utilisateurs. Une palette de couleurs bien conçue peut renforcer l'identité de la marque, diriger l'attention vers des éléments clés et même améliorer la lisibilité. De plus, les couleurs peuvent jouer un rôle crucial dans l'accessibilité, en aidant à différencier les éléments et en fournissant des repères visuels supplémentaires pour tous les utilisateurs, y compris ceux ayant des besoins spécifiques (daltoniens, mal-voyants etc..).

II. LE PROJET



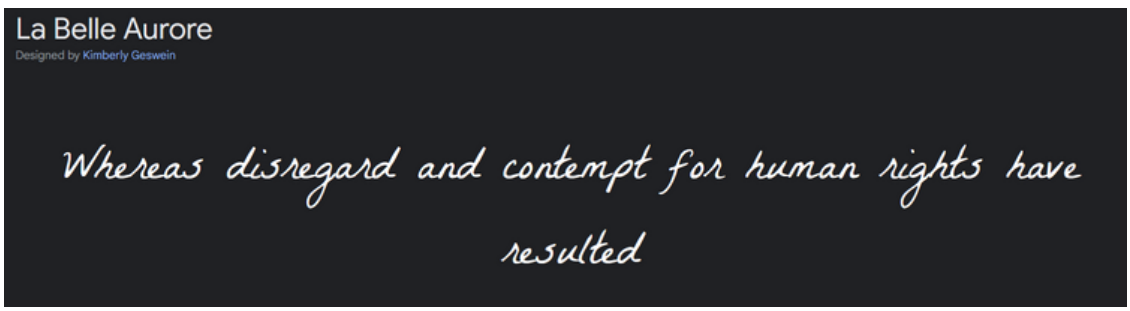


Résultat du site https://www.whocanuse.com/

Ce site permet d'améliorer l'accessibilité web pour les personnes ayant des déficiences visuelles, en proposant un score sur les couleurs affichées sur d'autres couleurs.

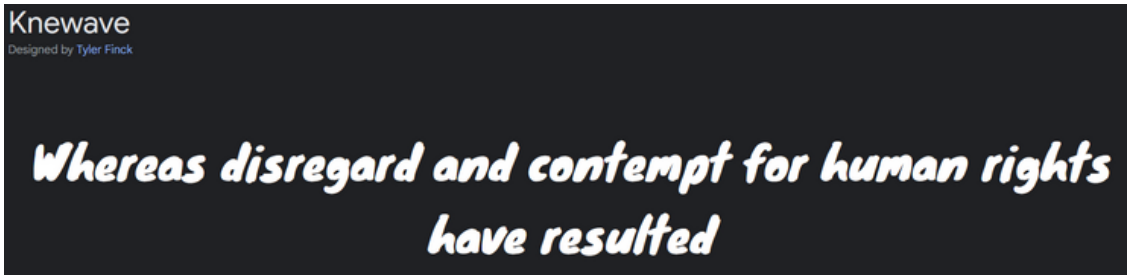
3 D Typographie

Seules les 3 polices suivantes ont été utilisées, trop de polices ont tendances a rendre le site web désordonné et difficile à lire. En limitant le nombre de polices, on garde une apparence cohérente et structurée qui facilite la navigation et la lecture.

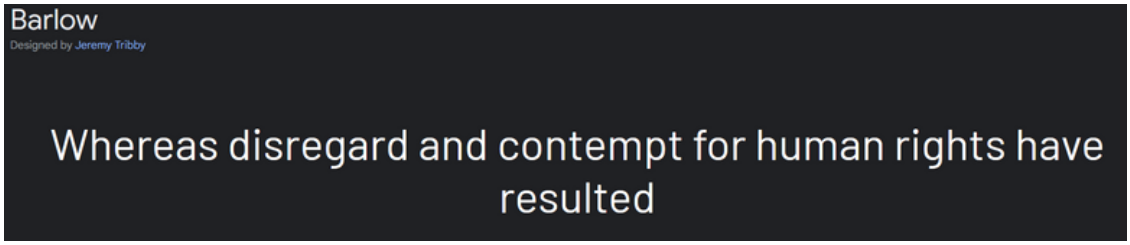


II. LE PROJET

La police "**La Belle Aurore**" a été choisie spécifiquement pour le nom du site. Elle se caractérise par son style manuscrit et ses angles doucement arrondis. Son esthétique s'harmonise parfaitement avec l'univers de la parentalité et des bébés



J’ai décidé d’utiliser la police "**Knewave**" pour les titres des pages et des cours. C'est une police épaisse et lisible qui capte l'attention de l'utilisateur. Ses courbes et arrondies s'inscrivent parfaitement dans l'ambiance du site.



Enfin, pour le contenu principal, comme les textes des cours, la section "présentation" et la partie "qui suis-je", j'ai opté pour la police "**Barlow**". Elle offre une très bonne lisibilité et s'intègre parfaitement au design général du site.

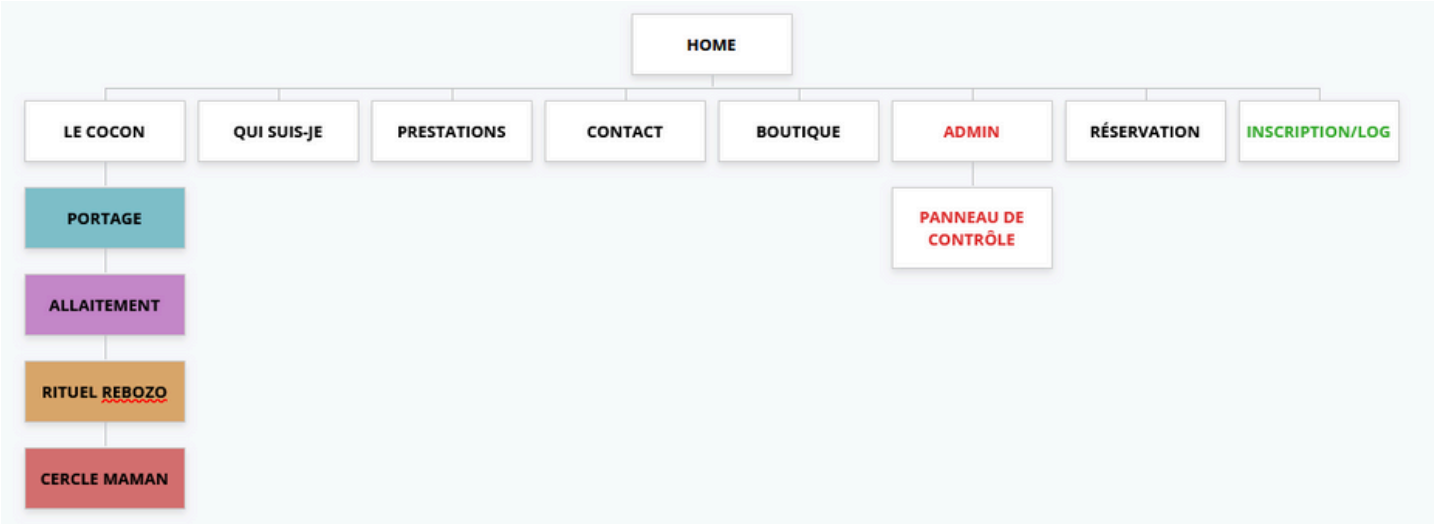
En plus de de l’importance de la typographie, la taille est également importante.

En effet, la taille de la police est un outil essentiel pour établir une hiérarchie visuelle et améliorer l'accessibilité du site. Elle guide l'utilisateur à travers le contenu et met en avant les informations clés.
Une taille de police bien choisie et flexible assure également une expérience utilisateur cohérente sur différents dispositifs et navigateurs.

Adaptabilité : Enfin, il est essentiel de choisir des polices qui sont flexibles et qui se rendent bien sur différents navigateurs et systèmes d'exploitation pour éviter des problèmes de compatibilité.

II. LE PROJET

3 E L'arborescence



3 F Responsive design

Le "responsive design" désigne la conception de sites web qui s'ajustent automatiquement à la taille de l'écran de l'utilisateur. L'objectif est d'assurer une expérience optimale, que le site soit consulté sur un ordinateur de bureau, un portable, une tablette ou un smartphone.
Pour réaliser ce responsive design, on utilise principalement des **media queries** et des "**breakpoints**". Ces outils permettent de reconfigurer l'affichage de l'application en fonction de différentes largeurs d'écran. Ainsi, la disposition des éléments, la taille des polices, et même l'expérience utilisateur (**UX**) peuvent être ajustées.

Le choix des unités et aussi très important dans la conception d'un design responsive :

- **PX** : Une unité fixe basée sur la résolution de l'écran.
- **%** : Une unité relative qui est un pourcentage de son élément parent.
- **EM** : Une unité relative à la taille de la police de l'élément parent.
- **REM** : Comme pour l'em, mais basé sur la taille de la police du document root.

Les **media queries**, en CSS, sont très importantes pour cela. Elles définissent des styles spécifiques selon la taille et l'orientation de l'écran ou encore sa résolution.

II. LE PROJET

Voici un exemple de la page de cours “Allaitement” et sa différence entre l’affichage Desktop (ordinateur de bureau) et Mobile (smartphone)


```css
@media (max-width: 1175px) {

.container1, .container2 {
    flex-direction: column;
}
```

Ici par exemple, afin d’adapter les container1 et 2 pour les tailles d’écran tablette est plus petit (le format tablette est en général entre 768px et 1200px) j’utilise les **media queries** pour passer la disposition en column plutôt qu’en row (nativement en row avec le display : **flex**)

Dans un dernier exemple, je vais également utiliser des **media queries** pour passer d’une ligne avec 3 produits à 2 pour les résolutions moyenne à 1 pour les résolutions les plus petites.

II. LE PROJET


```css
@media (max-width: 760px) {

    .enfant-cours {
        flex: 1 0 100%;
    }

    .enfant h2 {
        font-size: 1.8rem;
    }
```

Ici par exemple, afin d'adapter les container1 et 2 pour les tailles d'écran tablette est plus petit (le format tablette est en général entre 768px et 1200px) j'utilise les **media queries** pour passer la disposition en column plutôt qu'en row (nativement en row avec le display : **flex**)

Pour être plus précis encore : **flex** dans CSS fait partie de **Flexbox**, qui est un modèle de mise en page unidimensionnel pour le web. **Flexbox** permet d'aligner et de “distribuer” l'espace entre les éléments d'une interface, même lorsque leurs tailles sont inconnues ou dynamiques.

En l’occurrence ici, **flex** est un raccourci pour trois autres propriétés : flex-grow, flex-shrink, et flex-basis est permet de :

- Grandir pour occuper tout l’espace disponible dans le flex container (grâce à flex-grow: 1),
- Ne pas rétrécir en cas de manque d'espace (grâce à flex-shrink: 0)
- Et avoir une taille de base égale à 100% de la taille du flex container (grâce à flex-basis: 100%).

Étant donné que les écrans mobiles sont limités en largeur, il est important d'adopter une approche de design adaptative. Cette contrainte oblige à prioriser les informations et les éléments d'interface, en ne gardant que ce qui est vraiment indispensable pour l'utilisateur. Par contre, l'écran mobile offre un avantage considérable : la possibilité de "scroller" verticalement pour explorer le contenu en profondeur. Cela permet de créer des expériences utilisateur fluides et intuitives, où le contenu est révélé de manière progressive, incitant ainsi l'utilisateur à s'engager davantage avec le site ou l'application.
Le responsive desgin est également primordial pour un bon référencement.

Mobile first :

Concrètement, le **Mobile First** consiste à prioriser la version mobile puis adapter progressivement le web design pour les écrans plus larges, type ordinateurs. En d’autres termes, on créer dans un premier temps les pages d’un site web pour le smartphone puis on les fait “évoluer” pour qu’elles s’ajustent aux ordinateurs.
C’est une méthode que j’ai utilisé plusieurs fois, mais ne la maîtrisant pas entièrement, je me suis senti plus a l’aise en commençant par la version Desktop pour ce projet.

4 RGPD

Le **RGPD**, (Règlement Général sur la Protection des Données) à été introduit en 2018, et est une réglementation européenne conçue pour protéger les données personnelles des citoyens européens.
Il vise à donner aux citoyens le contrôle sur leurs informations personnelles, en garantissant qu'ils sachent comment et pourquoi leurs données (en particulier les données sensibles) sont utilisées.

Dans notre applications, nous allons informer les utilisateurs au sujet des cookies, des informations demandées et de leurs utilisation, la gestions de cookie est faite par l'intégration de **TarteAuCitron.io** qui un logiciel de gestion de consentement et de cookies recommandé par la **CNIL**.

La **CNIL** (Commission Nationale de l'Informatique et des Libertés) est l'autorité française de régulation en matière de protection des données. Elle est chargée de veiller à ce que la collecte et le traitement des données personnelles soient effectués dans le respect des droits et libertés des individus.

L'adoption d'un outil recommandé par la **CNIL**, comme **TarteAuCitron.io** pour la gestion des cookies, renforce non seulement la conformité de l'application aux réglementations en vigueur, mais aussi la confiance que les utilisateurs peuvent avoir sur ce site pour la gestion responsable de leurs données.

L'utilisateur peut également modifier ses informations ou même supprimer son compte conformément aux articles : Droit à la rectification (Article 16), Droit à l'effacement ou "droit à l'oubli" (Article 17) et Transparence (Article 12)

Notons que dans le cas d'une suppression de compte avec une commande en cours, l'utilisateur sera bien effacé de la base de donnée, mais que la commande, elle, restera.

II. LE PROJET

id	cours_id	user_id	dt_resa	dt_cours	payement_method	is_paid	user_phone	user_firstname	user_lastname	user_adress
62	3	(NULL)	2023-10-30 13:55:54	2023-10-08 15:00:00	payment-online	1	0000000000	Compte supp...	Compte sup...	Adresse suppr...

Pour permettre le NULL, j'ai du modifier la clé étrangère mappé pour user_id en SET_NULL

Lors d'un DELETE

SET NULL

```php
public function anonymizeReservations(): void {
    foreach ($this->reservations as $reservation) {
        $reservation->setUserFirstname('Compte supprimé');
        $reservation->setUserLastname('Compte supprimé');
        $reservation->setUserPhone('0000000000');
        $reservation->setUserAdress('Adresse supprimée');
    }
}
```

Pour respecter le **RGPD** j'ai créé une fonction pour anonymiser la réservation si l'utilisateur supprime son compte et éviter une réservation fantôme. Un PDF (avec domPDF) conservera cependant les informations.

J'ai également appliqué les obligations du **RGPD** en informant pourquoi ces données sont recueillis et dans quel but. L'utilisateur doit par exemple valider les conditions d'utilisation en s'inscrivant.

☐ J'accepte les termes et conditions

Les termes et conditions lui expliqueront entre autre, comment ses données personnelles sont collectées, pourquoi ses données sont nécessaires au bon fonctionnement du site, comment elles sont utilisées, et protégées.
Elles détaillent également les droits de l'utilisateur en ce qui concerne ses données, comme le droit d'accès, de modification et de suppression (comme expliqué plus haut).

Enfin, les termes et conditions expliqueront ce qu'est un cookie et pourquoi certains sont nécessaire et sont donc obligatoire, et comment désactiver les **cookies** sur les différents navigateurs.

J'ai des le début du projet essayé au maximum de respecter la philosophie **Privacy by design** et de construire mon application en respectant les obligations **RGPD.**

Pour finir, le **RGPD** impose une sécurité robuste, je détaillerais plus en détail le lien entre le **RGPD** et la partie sécurité dans la partie Sécurité.

II. LE PROJET

Maintenant que j'ai expliqué le projet, commençons par définir une méthode de travail et la construction des bases, le MCD et MLD.

Dans un premier temps, je vais expliquer quelles méthodes j'ai choisi pour construire ce projet et pourquoi ces méthodes ont été choisit.

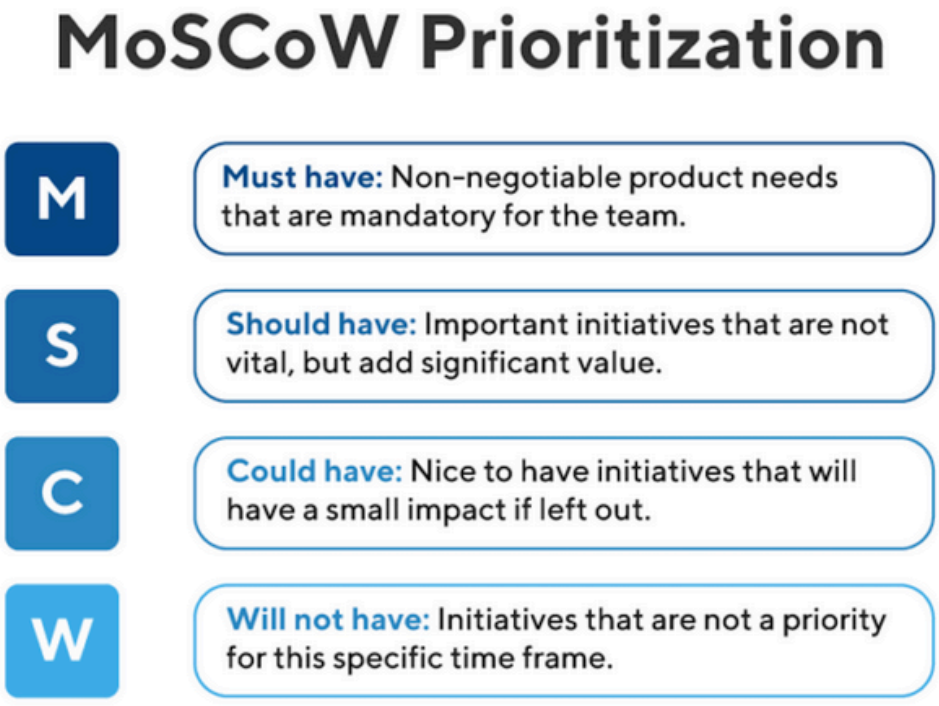
C'est également a ce moment que j'ai commencé mon Trello, et je vais expliqué pourquoi et comment il ma aidé.

Ensuite, je vais présenter le MCD et expliquer les différentes entités de mon projet ainsi que leurs relations.

Pour finir je vais présenter le MLD qui rentreras plus en détail, notamment en ce qui concerne les différentes clés dans les tables et les cardinalités.

5 A. Methodes Moscow, Merise et le Trello

Pour bien gérer ce projet et atteindre les objectifs associés, il est impératif de prioriser les points à traiter, les tâches à mener, les solutions à appliquer. Utilisée dans les approches agiles, la méthode **MoSCoW** est un outil pratique et simple à mettre en œuvre pour fixer les priorités.



II. LE PROJET

La méthode **MoSCoW** est une technique très utilisée dans la gestion de projet, notamment dans les domaines du développement. J'ai listé ici quatre points forts de cette méthode, qui sont pour moi les plus importants.

1 - Priorisation des Exigences : La méthode **MoSCoW** aide à prioriser les exigences en les classant dans quatre catégories : Must have (M), Should have (S), Could have (C), et Won't have (W). Cela permet de s'assurer que toutes les parties prenantes (ici, moi et le client) comprennent quelles sont les exigences essentielles à réaliser en premier lieu et quelles sont celles qui peuvent être retardées.

2 - Clarté et Communication : En classant les exigences en catégories clairement définies, la méthode **MoSCoW** aide à clarifier les attentes et à communiquer plus efficacement avec toutes les parties prenantes. Ici, travaillant seul, cette méthode me permet de me concentrer sur les points important et pas me disperser.

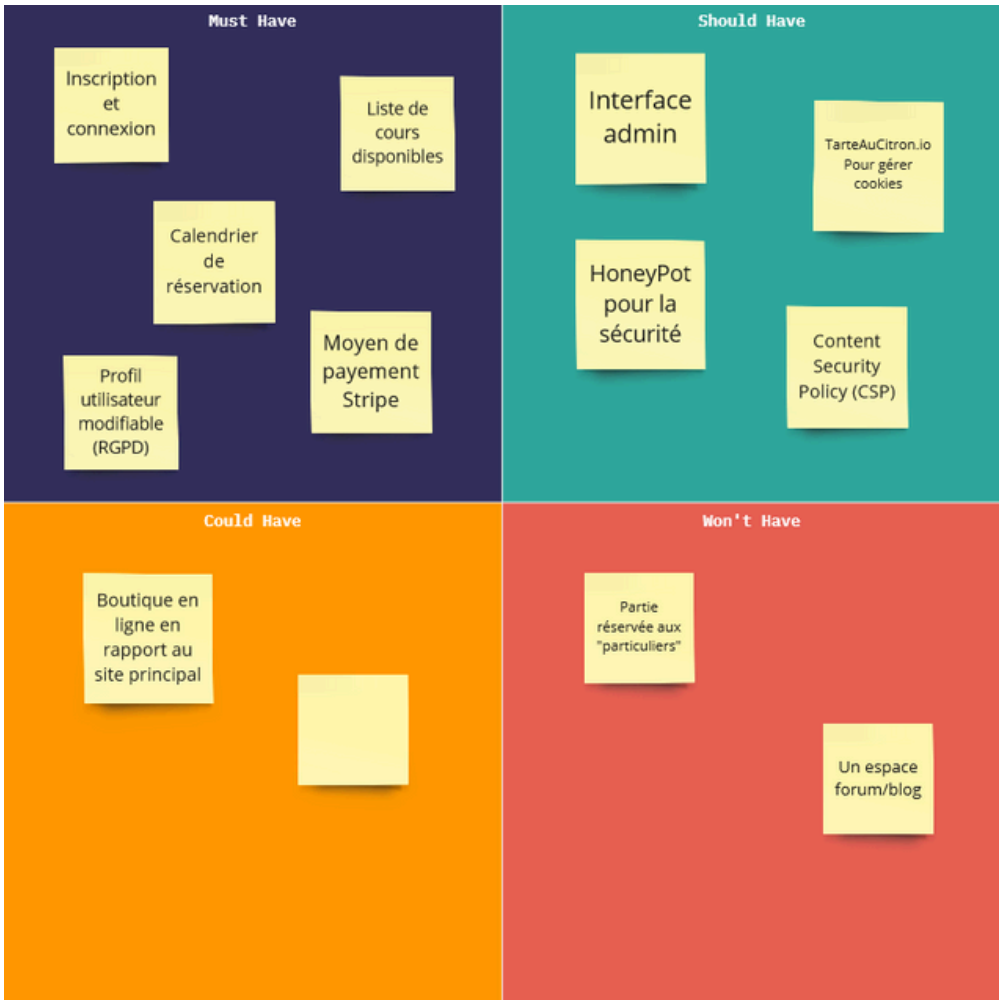
3 - Flexibilité : La méthode **MoSCoW** permet une certaine flexibilité dans la gestion du projet. Les exigences classées comme 'Could have' ou 'Won't have' peuvent être repoussées ou même supprimées si des contraintes de temps ou de budget le nécessitent. Cela permet de s'adapter aux changements inévitables qui peuvent survenir au cours d'un projet.

4 - Aide à la prise de décision : En identifiant clairement les priorités par rapport à celles qui sont moins importantes, la méthode **MoSCoW** facilite la prise de décision tout au long du projet.

Cela aide le développeur à se concentrer sur ce qui est vraiment important et à prendre des décisions sur ce qui peut être repoussé ou supprimé.

J'ai élaboré le tableau ci-dessous avec le client, pour connaître ses priorités, ce qu'il souhaite impérativement, en "option" ou qui ne sont pas très important dans un court ou moyen terme.

II. LE PROJET



Ici, nous avons donc le tableau **MoSCoW**, montrant les 4 parties :

- **Must Have :** Qui va déterminer le coeur du projet, indispensable. Ce sont les piliers du projet, sans elles, le site web ne pourrait pas remplir son objectif principal.

- **Should Have :** Qui va déterminer les tâches importantes mais non nécessaire au fonctionnement du site. Elles enrichissent l'expérience utilisateur et apportent une valeur ajoutée non négligeable, bien qu'elles ne soient pas vitales pour le fonctionnement de base.

- **Could Have :** Qui va déterminer ce qui peut être intéressant/pertinent mais pas dans l'immédiat. Souvent des 'bonus' qui pourraient améliorer le site à l'avenir, mais leur absence n'affecte pas la fonctionnalité actuelle du site.

- **Won't have :** Qui va déterminer les tâches envisageables uniquement quand les autres tâches ont été réalisées.

II. LE PROJET

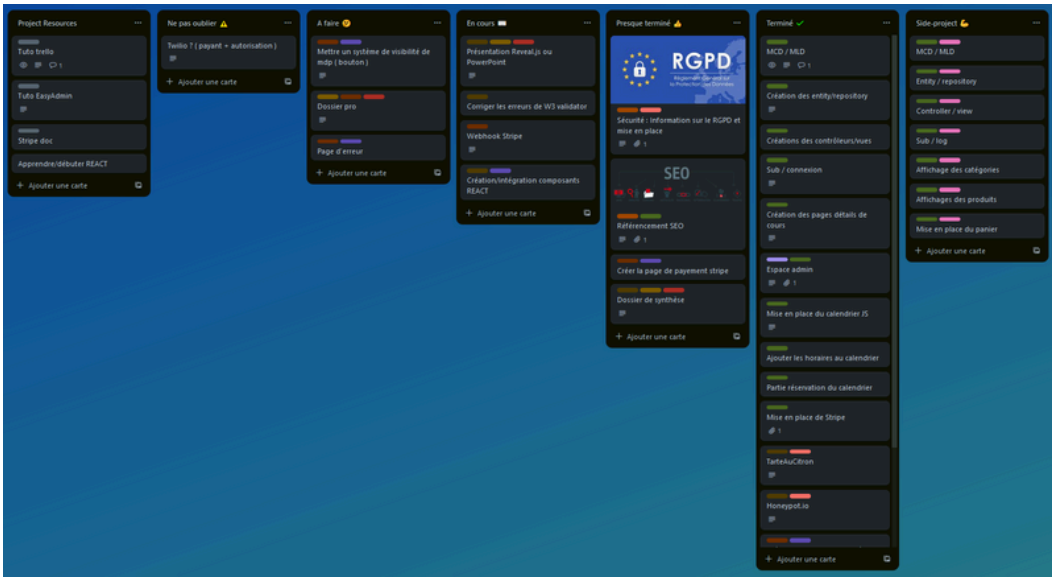
Après avoir défini les fonctionnalités principales du projet en utilisant la méthode **MoSCoW**, l'étape suivante consiste à établir un plan de route détaillé pour concrétiser les différentes étapes nécessaires au développement. Pour cela, j'ai opté pour l'utilisation de **Trello**, un outil de gestion de projet en ligne qui offre une visibilité et une organisation des tâches à effectuer.

Trello facilite la structuration du flux de travail grâce à un système de colonnes, dans lesquelles il est possible d'intégrer différentes catégories comme "À faire", "En cours", et "Terminé". Ces colonnes permettent non seulement de suivre l'avancement du projet, mais aussi d'assurer que le(s) développeur(s) reste focalisée sur les tâches prioritaires, évitant ainsi les risques de se disperser ou de se perdre.

Trello offre également la possibilité d'ajouter des notes ou des liens aux tâches, ce qui se révèle utile pour pouvoir retrouver rapidement une documentation ou un fichier de référence sans avoir à le rechercher à nouveau. Ainsi, en centralisant toutes les informations, ça permet non seulement de conserver une trace de ce qui a été accompli, mais aussi d'anticiper et d'organiser efficacement ce qui reste à faire.

L'usage de Trello s'inscrit donc dans une démarche proactive visant à garantir la fluidité et la transparence tout au long du cycle de développement du projet.

Voici mon tableau Trello servant à la réalisation du projet au 3/4 de son terme.



II. LE PROJET

Enfin j’ai utilisé la méthode **Merise** pour faciliter la création de ma base de données.
Merise est une méthode informatique dédiée à la modélisation. Le gros avantage de cette méthode est qu’elle permet de cadrer le projet informatique et de “discuter” en se comprenant entre client et développeur.
(et c’est français  !)
Merise est au final un outil analytique qui facilite la création de base de données et de projets informatique. Concrètement la méthode **Merise** permet de :

Modéliser des données :

- **Merise** aide à élaborer un modèle de données clair et structuré. Ca facilite grandement l'implémentation avec des outils comme Doctrine. **Merise** permet de définir clairement comment les entités sont structurées et liées entre elles.

Gerer des relations

- Cette méthode offre une visualisation claire des relations entre entités, rendant ainsi plus aisé le travail de création de jointures SQL ou de relations dans un ORM comme Doctrine.

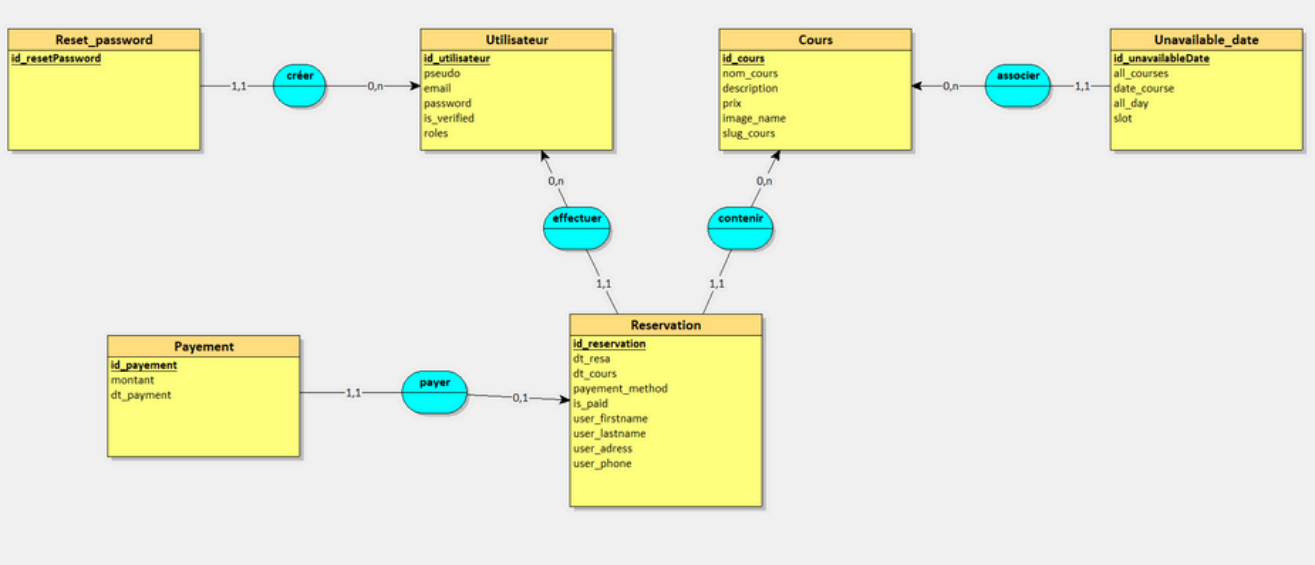
5 B. MCD

Le **MCD**, (ou Modèle Conceptuel de Données), est un outil qui permet de représenter de manière schématique et structurée les informations que l'on souhaite gérer dans une base de données.
Il s'agit en quelque sorte d'un "plan" qui identifie les différentes entités (qui deviendront des tables sur le **MLD**), leurs relations et leurs attributs, avant de passer à la phase de création proprement dite de la base de données. C'est une étape essentielle pour garantir une conception solide et efficace de la base de données.

Le **MCD** ansi que le **MLD** ont été réalisés en fonction de ce que voulait le client, mais également après une réflexion ensemble, autour des fonctionnalités, et de l’expérience utilisateur voulu.

II. LE PROJET

Le **MCD** ainsi que le **MLD** ont été réalisés en fonction de ce que voulait le client, mais également après une réflexion ensemble, autour des fonctionnalités, et de l'expérience utilisateur voulu.



Avec cette structure nous avons donc les cardinalité suivantes :

- Un cours peut être réservé plusieurs fois, mais une réservation concerne un seul cours. (**Many-to-One**)
- Un utilisateur peut effectuer plusieurs réservations, mais une réservation est faite par un seul utilisateur. (**Many-to-One**)
- Une réservation peut avoir un seul paiement, et un paiement concerne une seule réservation. (**One-to-One**)
- Un cours peut avoir plusieurs dates non disponibles, mais une date non disponible peut concerner un seul cours (ou tous les cours si all_courses est activé donc **Many-to-Many**). (**Many-to-One** hors all_courses)

Les cardinalités définissent comment les éléments dans une base de données sont connectés. Elles sont importantes car elles aident à comprendre et à mettre en place les règles de gestion de la base de données.

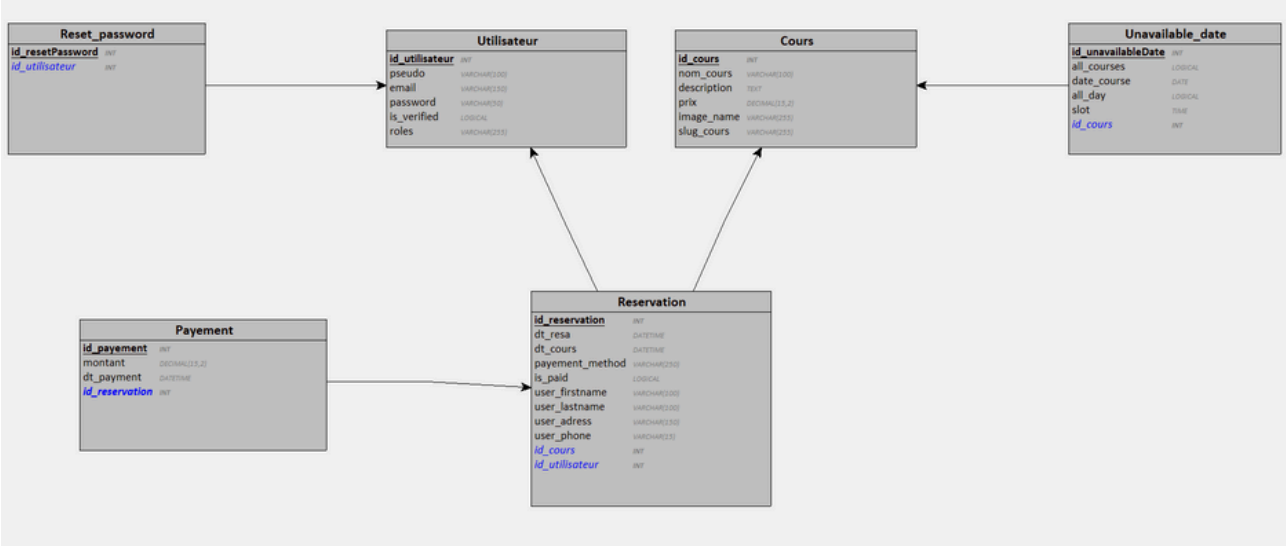
C'est grâce aux cardinalités que l'on est ensuite capable de créer soit des clés étrangères (expliquées dans la partie **MLD** juste après) soit des tables intermédiaires.

II. LE PROJET

5 C. MLD

Le **MLD**, ou Modèle Logique de Données, est une étape qui suit la réalisation du **MCD**. Il traduit le **MCD** en une structure plus technique, se rapprochant de ce qui sera effectivement créé dans la base de données.

Le **MLD** détaille les tables, leurs clés, et les relations entre elles, en préparation de la mise en place concrète de la base de données. Il est essentiel pour assurer que la base de données fonctionne de manière optimale et conforme aux besoins identifiés lors de la phase de conception.



Voici un listing des clés présentes :

Cours: Clé primaire: id

User: Clé primaire: id

Reservation: Clé primaire: id
Clés étrangères: id_cours: clé primaire id de cours
id_user: clé primaire id de user.

Payement: Clé primaire: id
Clé étrangère: id_reservation: clé primaire id de reservation.

Unavailable_date: Clé primaire: id
Clé étrangère: id_cours: clé primaire id de cours (cette clé est “optionnelle”, car une date non disponible peut concerner tous les cours si le champ all_courses est activé).

Pour définir simplement ce qu'est une **clé primaire** et **étrangère**.
Une **clé primaire** est un identifiant unique pour chaque enregistrement dans une table de base de données (on parle ici de table et non plus d'entité). Dans l'entité Cours de mon projet, la **clé primaire** est représentée par la propriété $id. Cette propriété $id assure qu'il n'y a pas deux cours identiques dans la table, et chaque cours a une valeur unique pour cette **clé primaire**, ce qui permet de le retrouver facilement en base de données.

Une **clé étrangère** est un champ dans une table qui fait référence à la **clé primaire** d'une autre table, établissant ainsi une relation entre les deux tables. Dans mon projet, une Reservation a une relation avec un Cours, et cette relation est gérée par une **clé étrangère**. Cette **clé étrangère** dans la table Reservation fait référence à la **clé primaire** $id de la table Cours, permettant ainsi de savoir quel cours est réservé pour chaque réservation.

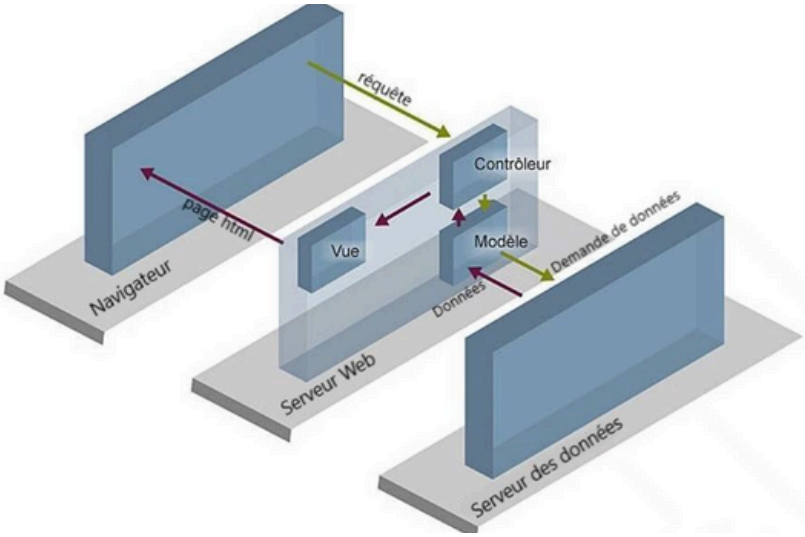
6 L'architecture du projet

6 A. Le design pattern MVC (MVP)

Un design pattern, est une solution générale réutilisable à un problème qui se produit fréquemment dans la conception. Il sert à résoudre un problème qui peut être utilisé dans de nombreuses situations différentes. Les design patterns peuvent grandement faciliter la conception en favorisant la réutilisabilité, la modularité, et la lisibilité du code.
Les design patterns sont des "solutions abstraites", ce qui veut dire qu'ils ne sont pas liés à un problème ou une technologie spécifique, mais sont plutôt des guides pour structurer et résoudre des problèmes de manière plus efficace.

Le modèle **MVC**, ou Model-View-Controller, est un "ensemble de bonnes pratiques" couramment utilisées pour développer des applications web. Il divise une application en trois composants connectés entre eux, ce qui permet de séparer les tâches internes et facilite la maintenance et l'évolution de l'application.
J'ai ici joint un schéma qui explique brièvement le fonctionnement de ce design pattern **MVC.**

II. LE PROJET



Avant d'expliquer le chemin globale d'une requête **HTTP** dans un modèle **MVC**, voici une définition de cette dernière :

Une requête **HTTP** (HyperText Transfer Protocol) est un message envoyé par un client (en général un navigateur web) à un serveur pour demander une action ou des informations. Elle inclut des éléments comme la méthode (GET, POST*, etc.), l'URL, les en-têtes, et éventuellement un corps de message. Le serveur répond à la requête avec une réponse **HTTP**, qui contient le code de statut, les en-têtes, et souvent un corps de message avec les données demandées.

1. Requête HTTP : Lorsque l'utilisateur envoie une requête **HTTP**, généralement en cliquant sur un lien ou en soumettant un formulaire sur le navigateur. Cette requête est envoyée au **contrôleur** en passant par le routeur (qui va déterminer à quel **contrôleur** l'envoyer)

2. Contrôleur - Obtention des données : La requête atteint **le contrôleur** de l'application. Le rôle du **contrôleur** est de traiter la requête et de déterminer quelles actions entreprendre. Si la requête nécessite des données (par exemple, récupérer un cours en base de donnée), le **contrôleur** demande au **modèle** de récupérer ces données. C'est aussi lui qui fait le lien entre le modèle et la vue.

3. Modèle - Demandes des données : Le modèle est responsable de la logique métier et de l'interaction avec la base de données. Le modèle exécute les opérations nécessaires pour obtenir, mettre à jour, supprimer ou ajouter (**CRUD**) les données de la base de données.

* Je définis les méthodes **POST** et **GET** dans la partie calendrier de réservation page 46

Une fois cette opération terminée, le **modèle** renvoie les données au **contrôleur**.

4. Contrôleur à la Vue : Le **contrôleur** passe ces données à la **vue**, qui est un template qui prend ces données et les structure de façon à être présentée à l'utilisateur. La **vue** génère alors le contenu HTML basé sur les données fournies.

5. Réponse HTML : Le contenu HTML généré est renvoyé au navigateur de l'utilisateur comme réponse à la requête HTTP initiale. Le navigateur affiche ce contenu, ce qui permet à l'utilisateur de voir les informations ou les résultats de son action. (Sa requête HTTP)

MVP :

Bien que le design pattern **MVC** soit très répandu et souvent cité lorsqu'il s'agit de décrire l'architecture des applications web, notamment avec Symfony, en réalité ils sont plus proche du **MVP** (Modèle Présentation Contrôleur).

La principale différence se trouve dans le rôle de la **"Vue"** selon le pattern **MVC**. Dans un système **MVP** , la **vue** est purement dédiée à l'affichage, alors que la logique de présentation, qui décide comment les données sont préparées pour l'affichage, est gérée par un composant distinct appelé **"Présentation"**. Dans le cas de Symfony, cette logique de présentation est effectuée par le **contrôleur.**

Donc, même si le terme **MVC** est couramment utilisée, il serait plus exact de dire que Symfony, suit un schéma **MVP**. La **vue**, reçoit ses données préparées du **Contrôleur** et les affiches.
Cela renforce la séparation des préoccupations, chaque composant ayant un rôle bien défini : **le Modèle** pour la logique des données, le **Contrôleur** pour la logique de présentation et la **vue** pour l'affichage.
En conclusion, l'usage du terme "**MVC**" est devenu si courant qu'il est souvent utilisé pour décrire des architectures qui suivent en réalité le schéma **MVP** . L'appellation "**MVC**" est donc souvent un abus de langage, mais elle est largement acceptée et comprise.

6 B. La couche modèle et le rôle de Doctrine

Modèle (Model) : La gestion des données est principalement réalisée grâce à Doctrine. Doctrine est **l'ORM** utilisé par défaut par Symfony.

C'est un programme qui permet de faire l'interface entre Symfony et une base de données relationnelle, il permet de mapper, ou de relier, des objets à des enregistrements dans une base de données. Cela permet de travailler avec les données en utilisant des objets orientés objet, tout en stockant et récupérant ces données d'une base de données relationnelle.

Cette capacité de Doctrine à simplifier l'interaction entre Symfony et la base de données, introduit ce qu'on appelle une "**couche d'abstraction**".

La **couche d'abstraction** est un niveau de code qui sépare les opérations de haut niveau (plus abstrait et CRUD entre autre) des opérations de bas niveau (moins abstrait comme les requêtes SQL brut).
Du coup, grâce à Doctrine, la couche modèle offre une **couche d'abstraction** qui permet de travailler avec des objets et des méthodes de haut niveau sans avoir à rédiger des requêtes SQL brutes. L'**ORM** s'occupe de traduire ces opérations de haut niveau en requêtes SQL pour interagir avec la base de données, facilitant la gestion des données et la logique métier de l'application.

L'un des avantages de cette approche est l'**encapsulation**, un principe fondamental de la programmation orientée objet.

L'**encapsulation** permet de regrouper les données et les méthodes qui les manipulent au sein d'un même objet, tout en restreignant l'accès direct aux certaines données, ce qui aide à rendre sécurisé et modulable l'application.

Avec Doctrine, l'accès aux données est encapsulé de manière à fournir une interface cohérente et sécurisée pour la gestion des données. Les détails de la base de données et les requêtes SQL sont masqués, ce qui permet de se concentrer sur la logique métier sans se soucier des détails spécifiques de la base de données.

Cette **encapsulation** se manifeste concrètement dans la conception de mes entités, qui regroupent les données et les méthodes associées en un seul bloc. Chaque entité représente une table dans la base de données, et ses attributs correspondent aux colonnes de cette table.

Par exemple l'entité Cours, regroupe plusieurs attributs indispensables tels que le nom, la description, le prix, et un slug.
De plus, elle possède un identifiant unique ID, mappé en tant que clé primaire, ainsi qu'une relation avec l'entité Reservation, symbolisée par une clé étrangère.
Ce mapping entre les entités et la base de données est facilité par Doctrine, et montre bien le fonctionnement de la **couche d'abstraction** et de l'**encapsulation** dans la gestion des données de l'applicaiton.

```php
#[ORM\Id]
#[ORM\GeneratedValue]
#[ORM\Column]
private ?int $id = null;

#[ORM\Column(length: 100)]
private ?string $nom_cours = null;

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

#[ORM\Column]
private ?float $prix = null;

#[ORM\OneToMany(mappedBy: 'cours', targetEntity: Reservation::class)]
private Collection $reservations;

#[ORM\Column(length: 255)]
private ?string $slug_cours = null;
```

Au coeur de Doctrine et du **modèle** on a l'**EntityManager**, L'**EntityManager** permet de manipuler les entités comme des objets, facilitant ainsi les opérations **CRUD** (Create, Read, Update, Delete). Globalement, il est responsable de la gestion du cycle de vie des objets, y compris leur persistance dans la base de données, leur récupération, leur mise à jour et leur suppression

```php
public function remove(Cours $entity, bool $flush = false): void
{
    $this->getEntityManager()->remove($entity);

    if ($flush) {
        $this->getEntityManager()->flush();
    }
```

Ici par exemple, on va supprimer un cours grâce à l'**EntityManager** et sa méthode remove()

Enfin, autre acteur de cette couche modèle, le **repository.**
Il permet de faire des sélections grâce à des fonctions natives Symfony (comme find(), findOneBy(), findAll() et findBy()) ou il est également possible d'implémenter des fonctions personnalisées selon les besoins de l'application.

```php
#[Route('/', name: 'app_home')]
public function index(CoursRepository $coursRepository): Response
{

    $cours = $coursRepository->findAll();

    return $this->render('home/index.html.twig', [
        'controller_name' => 'HomeController',
        'listeCours' => $cours,
    ]);
}
```

C'est par exemple le cas ici, ou nous appelons la méthode findAll() grâce au CoursRepository.
En résumé, le **Modèle** dans le modèle MVC encapsule la logique métier et la gestion des données de l'application, en utilisant souvent un **ORM** comme Doctrine pour fournir une couche d'abstraction et simplifier l'interaction avec la base de données.

6 C. Le contrôleur

Contrôleur (Controller) : Le contrôleur est lui, le cœur de l'architecture **MVC**. Il reçoit les requêtes de l'utilisateur, traite ces requêtes en interagissant avec le **modèle** si nécessaire, puis renvoie une réponse appropriée à l'utilisateur en utilisant la **vue**.
Dans Symfony, le **contrôleur** est souvent une classe qui contient des méthodes, appelées actions, qui définissent comment répondre à différentes requêtes.

```php
#[Route('/', name: 'app_home')]
public function index(CoursRepository $coursRepository): Response
{

    $cours = $coursRepository->findAll();

    return $this->render('home/index.html.twig', [
        'controller_name' => 'HomeController',
        'listeCours' => $cours,
    ]);
}
```

Si on reprends le même exemple que précédemment, nous avons bien ici une function dans le **controlleur** qui s'occupe de recevoir la requete, demande au Repository de chercher les informations grace a findAll(), et renvoie les informations à la vue appropriée. Il s'occupe bien de faire le lien entre le **modèle** et la **vue**.

le **contrôleur** joue également un rôle vital au niveau de la sécurité, en validant les données reçues des utilisateurs avant de les passer au **modèle**, il minimise les risques d'injections malveillantes ou d'autres formes d'attaques (expliqué plus en détails dans la partie sécurité).

Sur le plan de la performance, le **contrôleur** peut optimiser le traitement des requêtes en interagissant avec le **modèle** seulement lorsque c'est nécessaire.

Dans notre exemple précédent, nous avons vu une action typique d'un **contrôleur** qui reçoit une requête, interroge le modèle via le repository et renvoie les données à la **vue**.

C'est dans ce processus que l'**injection de dépendances** est précieuse.
Elle permet d'injecter des services comme les repositories directement dans les **contrôleurs**, facilitant ainsi la modularité du code.

Pour conclure, le **contrôleur**, en orchestrant l'interaction entre la **vue** et le **modèle**, et en gérant la sécurité, permet une organisation claire et une gestion efficace des requêtes utilisateur, ce qui rend plus robuste l'application.

6 D. La vue

Vue (View) : La **vue** est responsable de la présentation des données à l'utilisateur. Elle prend les données fournies par le **contrôleur** et les affiche à l'utilisateur, entre autre avec le moteur de template **Twig**.

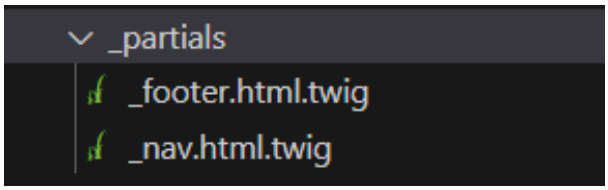
En utilisant **Twig**, il est également possible d'inclure des éléments réutilisables, comme des en-têtes ou des pieds de page, et d'appliquer des filtres pour formater les données, comme la mise en forme des dates ou des chaînes de caractères.

```
{# Include de la nav #}
{% include "_partials/_nav.html.twig" %}

    {% block body %}

    {% endblock %}

{# Include du footer #}
{% include "_partials/_footer.html.twig" %}
```



Dans cet exemple, nous utilisons {% include %} de **Twig** pour inclure des fichiers provenant d'autres emplacements.
Cette approche rend le code plus lisible en le divisant en parties distinctes et facilite la gestion de chaque composant.

On peut également appliquer des filtres comme :

- **slice** `{% for cours in listeCours|slice(0, 3) %}` ici, dans cette boucle ou je liste les cours, je vais par exemple “découper” un tableau (d’objet) ou une chaine de caractère en plusieurs parties, pour l’affichage.

- **raw** `<p>{{ cours.description | raw  }}</p>` pour afficher le contenu de la chaîne sans conversion ou d'échappement des caractères spéciaux.

- **Date** `{{ reservation.dtResa | date("m/d/Y") }}` pour formater une date.

Ces filtres et éléments réutilisables facilitent la création d'une interface utilisateur riche et interactive.
Comme écris plus haut et en reprenant le même exemple, lorsqu'une action est exécutée dans le contrôleur, par exemple, la récupération de la liste des cours via la méthode findAll() du repository, les données récupérées sont ensuite passées à la **vue**. Dans Symfony, cela se fait en retournant une réponse du **contrôleur** avec les données nécessaires incluses, comme ceci :

```php
#[Route('/', name: 'app_home')]
public function index(CoursRepository $coursRepository): Response
{

    $cours = $coursRepository->findAll();

    return $this->render('home/index.html.twig', [
        'controller_name' => 'HomeController',
        'listeCours' => $cours,
    ]);
}
```

On “return” bien à la vue : cours/index.html.twig, la liste des cours.

Pour résumé, la **vue**, joue un rôle crucial en rendant l'interaction entre l'utilisateur et l'application agréable et intuitive. Elle forme le pont entre le **contrôleur** et l'utilisateur, en présentant les données de manière structurée et esthétique. Avec Twig dans Symfony, elle permet une gestion efficace et modulaire de la présentation.

III. CONSTRUCTION DU SITE

1. Fonctionnalités du site

1 A. Inscription et connexion

Après avoir sélectionné les méthodes de travail, réalisé le maquettage ainsi que les modèles conceptuel et logique de données (MCD et MLD), je peux maintenant entamer la construction du site de manière plus efficace.

Afin de respecter le **RGPD** et ne pas stocker d'informations qui ne seraient pas directement utiles à la réservation d'un cours, seuls les éléments indispensables sont demandés :

- Un pseudo (utilisable pour une partie forum a venir, expliqué dans les améliorations)
- Une adresse e-mail
- Un mot de passe
- Une vérification de mot de passe
- Une case pour accepter les termes et conditions.

Après l'inscription, l'utilisateur est averti qu'il a bien été enregistré et qu'il a reçus un e-mail de confirmation. Cet e-mail de confirmation contient un lien pour authentifier l'utilisateur et le redirige sur la page de connexion.

Pour des raisons pratiques et par choix du client, un utilisateur peut parcourir les pages sans être connecté, en revanche pour réserver un cours, il devra obligatoirement se connecter et donc être inscrit.

1 B. Compte utilisateur

Pour la sécurité du site, mais aussi pour la sécurité de l'utilisateur et de son profil, j'ai mis en place des restrictions de caractère, ainsi qu'un nombre de caractères minimum requis.

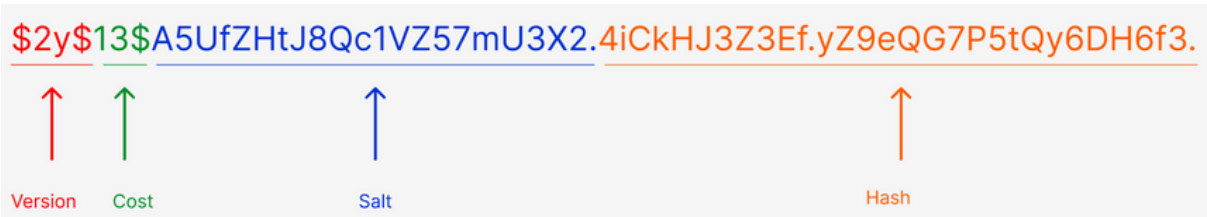
III. CONSTRUCTION DU SITE

```
'constraints' => [
    new NotBlank([
        'message' => 'Veuillez entrer un mot de passe',
    ]),
    new Length([
        'min' => 14,
        'minMessage' => 'Votre mot de passe doit contenir au moins {{ limit }} caractères.',
        'max' => 50,
        'maxMessage' => 'Votre pseudo ne doit pas dépasser {{ limit }} caractères.'
    ]),
    new Regex([
        'pattern' => '/^(?=.*[A-Z])(?=.*[a-z])(?=.*\d)[A-Za-z\d]{14,}$/',
        'message' => 'Votre mot de passe doit contenir au moins une lettre majuscule, une lettre minuscule, et un chiffre.',
    ]),
],
```

Ceci est un exemple de restriction pour le mot de passe, faite a l'aide d'une **regex.**

Une expression régulière (**regex**) permet de vérifier si une chaîne de caractères correspond à un format spécifique, ou de trouver/remplacer des portions de texte. Les **regex** sont utiles pour la validation des données, comme ici, pour s'assurer que le mot de passe respecte certaines règles. Cela a pour but de garantir un mot de passe robuste.

Une fois le mot de passe envoyé, il va être enregistré en base de donnée.
Symfony effectue le hashage nativement avec Bcrypt et inclu un salt ce qui le rend spécifiquement conçu pour le hachage de mots de passe. Il est particulièrement robuste contre les attaques arc-en-ciel (technique pour retrouver un mot de passe à partir de sa valeur de hachage, en exploitant un "compromis" entre le temps de calcul et l'espace mémoire)



Une fois inscrit et connecté, l'utilisateur doit pouvoir consulter son compte. Afin de respecter le **RGPD,** l'utilisateur doit pouvoir modifier ses informations personnelles et supprimer son compte. (selon les articles, Droit à la rectification (Article 16), Droit à l'effacement ou "droit à l'oubli" (Article 17))

III. CONSTRUCTION DU SITE

La mise en page de chaque cours est sobre, mais contient toutes les informations nécessaires. Cela permet de maintenir un design élégant et épuré, sans surcharge inutile, tout en étant suffisamment documenté pour répondre à un maximum de questions que l'utilisateur pourrait se poser avant de s'inscrire éventuellement. Cette approche facilite la navigation et la compréhension, tout en offrant une expérience utilisateur agréable et efficace.

Les informations relatives aux cours dans cette section (menu déroulant détaillant les cours) sont codées en “dur”, c'est-à-dire qu'elles sont directement en HTML sur la page de la vue. Ce choix a été fait principalement parce que ces cours sont fondamentaux pour l'activité professionnelle du client et ne seront donc jamais supprimés.

De plus, ces cours ne subiront aucune modification et aucun autre cour "fondamental" ne sera ajouté dans un avenir proche ou même à moyen terme.

L’utilisateur dispose donc de la possibilité de modifier son pseudo, comme son E-mail, et son mot de passe.

Enfin, il peut également supprimer son compte à tout moment.

1 C. Les cours

Dès l'arrivée de l'utilisateur sur le site, il est accueilli par la page d'accueil.
Il a la possibilité de naviguer à travers cette page principale en faisant défiler le contenu, ou bien d'explorer le menu situé dans l'en-tête du site. Un point d'intérêt particulier dans ce menu est l'option "**Le Cocon**".

Il s'agit d'un menu déroulant qui répertorie les cours proposés, fournissant des détails précis sur chacun d'entre eux. Cette fonctionnalité offre aux utilisateurs une vue d'ensemble claire et détaillée des cours disponibles, leur permettant de choisir celui qui correspond le mieux à leurs besoins et à leurs intérêts.

Chaque cour va appartenir a un code couleur, pour une raison esthétique, mais surtout pour les différencier et s’y retrouver plus facilement. Un intérêt tout particulier a été apporter aux couleurs du menu déroulant, notamment pour les personnes malvoyantes ou daltoniennes. (test sur https://www.whocanuse.com)

III. CONSTRUCTION DU SITE

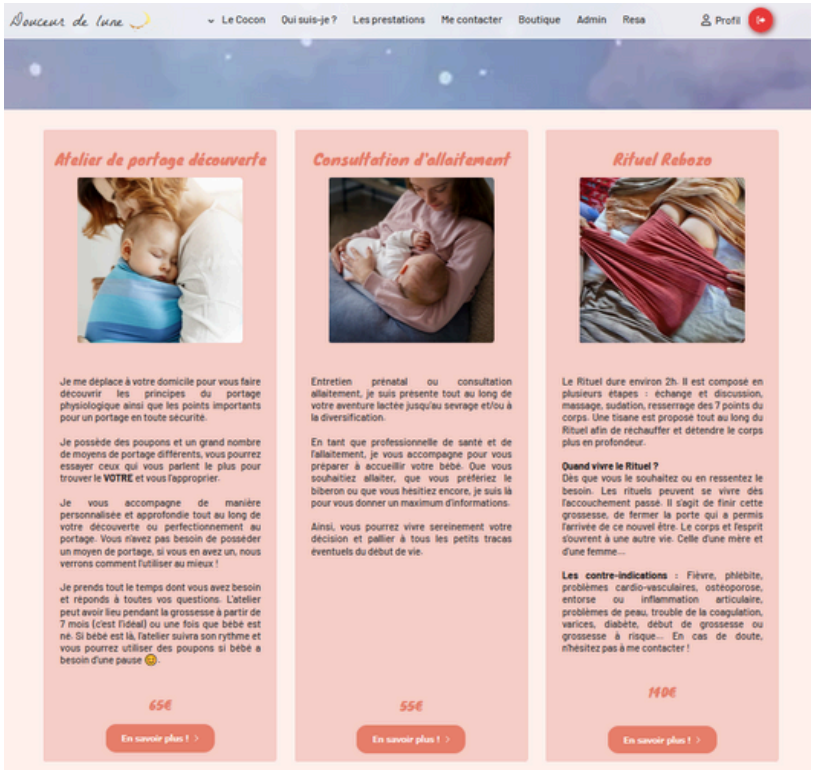
La section "**Les prestations**" est affichée de manière dynamique, ce qui permet de lister facilement les cours en affichant toutes les données importantes : le titre du cours, une description rapide, une image représentative du cours, ainsi que le prix.

```php
#[Route('/showCours', name: 'app_cours')]
public function index(CoursRepository $coursRepository): Response
{
    $cours = $coursRepository->findAll();

    return $this->render('pagesInfo/showCours.html.twig', [
        'listeCours' => $cours,
    ]);
}
```

```twig
{% for cours in listeCours|slice(0, 3) %}
    <div class="enfant enfant-cours pastel-blue1">
        <h2>{{ cours.nomCours }}</h2>
        <div class="img-dim"><img class="image-responsive" src="{{ asset('imgVich/' ~ cours.imageName) }}" alt="Image
        <p>{{ cours.description | raw }}</p>
        <p class="price2">{{ cours.prix }}€</p>
        <a class="details-link" href="{{ path('detailCours', {templateName: cours.slugCours}) }}#reservation-form">
            <button class="buttonMore">
            En savoir plus !
                <div class="arrow-wrapper">
                    <div class="arrow"></div>
                </div>
            </button>
        </a>
    </div>
{% endfor %}
```

Un lien est également ajouté, redirigeant l'utilisateur vers la page détaillée du cours et le calendrier de réservation. Cette fonctionnalité facilite une navigation plus pratique et fluide, permettant aux utilisateurs d'accéder rapidement à toutes les informations nécessaires pour faire un choix éclairé.



III. CONSTRUCTION DU SITE

1 D. Le calendrier de réservation

Le calendrier de réservation constitue le cœur du projet et représente la partie qui m'a demandé le plus de temps et d'énergie (ainsi que de nombreux comprimés de **Doliprane !**)

J'ai décidé de le créer en grosse partie en HTML et JavaScript afin de pouvoir le personnaliser comme la cliente le voulait, et de pouvoir répondre au maximum a ces exigences.

JavaScript est un langage que nous avons peu étudié pendant la formation, mais il m'attire particulièrement, ce qui a facilité la compréhension et l'apprentissage des bases nécessaires à la réalisation de ce calendrier.

Avant de rentrer dans le vif du sujet, je vais d'abord présenter le design du calendrier et expliquer ses fonctionnalités. Ensuite, je parlerai plus en détail du fonctionnement, et pour finir, de la partie réservation.

Pour le calendrier également, j'ai opté pour un design plutôt minimaliste, en veillant à ce que son utilisation soit intuitive pour l'utilisateur.

Il intègre les fonctionnalités suivantes, qui seront expliquées en détail par la suite :

- Une liste déroulante de mois, navigable vers la droite ou la gauche.
- Six semaines affichées pour le mois en cours, ainsi que pour les jours du mois suivant et précédent.
- La possibilité de sélectionner un jour précis en cliquant sur celui-ci.

III. CONSTRUCTION DU SITE

- Une fois le jour sélectionné, un menu contenant les horaires disponibles (non grisées) apparaît.

- Des boutons radio pour choisir le mode de paiement.

- Un bouton de réservation, qui devient cliquable uniquement si l'utilisateur a sélectionné un jour et une heure
  (par défaut le mode de paiement est “en ligne”)

Pour l'affichage, le calendrier se présente de la manière suivante, en HTML.

Dans ce code nous avons la structure du calendrier qui donne le résultat ci-dessus.

```
calendar.innerHTML = `
<h2 id='calendar-title'>Réservez votre cours</h2>
<div id="calendar-container">

    <div id="month-container">
        <div id="prev-month-calandar-selector" class="month-selector"><</div>
        <h3 id="current-month"></h3>
        <div id="next-month-calandar-selector" class="month-selector">></div>
    </div>

    <table>
        <thead>
            <tr>
                <th>L</th>
                <th>M</th>
                <th>M</th>
                <th>J</th>
                <th>V</th>
                <th>S</th>
                <th>D</th>
            </tr>
        </thead>
        <tbody id="calendar-body">
        </tbody>
    </table>

</div>

<div id="slot-container">
</div>
```

Une div pour montrer les flèches avant et arrière, pour changer de mois, mais aussi le mois actuel.

Une table avec un th (table header) par jour, qui seront donc les cellules.

Il y a également plusieurs champs cachés qui sont utilisés pour stocker les informations sur la réservation dans le screenshot suivant.

III. CONSTRUCTION DU SITE

```
<form method='POST' action="/save_reservation" id="calendar-form">
    <input type="hidden" name="courseName" value="${courseName}">
    <input type="hidden" name="slotDate" id="calendar-form-date" value="">
    <input type="hidden" name="slotTime" id="calendar-form-time" value="">
    <h4 id="payment-method-title">Choisissez votre moyen de paiement</h4>
    <div id='payment-method-container'>

        <div class='payment-method radio-button'>
            <input type="radio" name="payment-method" id="payment-sur-place" class="radio radio-button__input" value="payment-sur-place">
                <label for='payment-sur-place' class="radio-button__label">
                    <span class="radio-button__custom"></span>
                    <p class="p-test">Sur place</p>
                </label>
        </div>

        <div class='payment-method radio-button'>
            <input type="radio" name="payment-method" id="payment-online" class="radio radio-button__input" value="payment-online" checked>
                <label for='payment-online' class="radio-button__label">
                    <span class="radio-button__custom"></span>
                    <p class="p-test">En ligne</p>
                </label>
        </div>

    </div>
    <button type="submit" id="calendar-submit-button" disabled>Réserver</button>
</form>`;
```

Nous avons par exemple :

- **courseName** qui va être récupérer avec la constante détaillée plus tard.

```
const courseName = splitURL[splitURL.length-1];
```

- **slotDate** qui va être la cellule du jour sélectionné
- **slotTime** qui va être la cellule du créneaux horaire sélectionné

Mais un également un moyen de payement, ici 2 boutons radio, car on pourra en sélectionner qu'un des deux.

- **Sur place**
- **En ligne**

Pour finir, un bouton "**Réserver**" afin de permettre a l'utilisateur de réserver.

Ces informations seront envoyées côté serveur à l'url/save_reservation grâce a l'action du formulaire, quand l'utilisateur soumettra le formulaire, grâce à la méthode **POST*.** `<form method='POST' action="/save_reservation" id="calendar-form">`
On notera que le bouton est "**disable**", et sera cliquable uniquement après le jour et la date sélectionnés.`<button type="submit" id="calendar-submit-button" disabled>Réserver</button>`
Pour finir, une fois le squelette HTML du calendrier généré, il va falloir remplir et afficher ce calendrier, pour cela j'ai utilisé plusieurs variable et fonctions dont 3 très importantes **generateCalendar**(), **displayCalendarContent**(),et **selectUnit**()

* La méthode POST est une méthode de requête HTTP qui est utilisée pour envoyer des données à un serveur (côté client vers côté serveur). Le navigateur envoie une requête POST au serveur avec les données du formulaire dans le corps de la requête. Contrairement à la méthode GET qui est utilisée pour récupérer des informations du serveur, GET inclut les données de la requête dans l'URL ce qui est pratique pour les requêtes simples, mais moins sécurisé et moins adapté pour envoyer des données sensibles ou volumineuses.

```
var actualDate = new Date();
var numWeeks = 6;
const splitURL = window.location.href.split("/");
const courseName = splitURL[splitURL.length-1];
var startTime = new Date();
startTime.setHours(8, 0, 0);
var timeSlots = [];
```

Pour afficher et remplir le calendrier on va principalement utiliser les variables et constantes suivantes :

- **var actualDate** => on créer un objet date auquel on va appliquer getMonth (actualDate.getMonth) qui est une méthode natif a Javascript, et qui va nous permettre d'avoir un tableau [0-11] pour afficher les mois, janvier étant 0.

- **var numWeeks** => afficher six semaines sur un calendrier est une convention qui assure l'uniformité, la flexibilité et la praticité pour l'utilisateur.

- **const splitURL** => On va séparer l'URL actuelle en utilisant le séparateur "/". par exemple : http://localhost:8000/cours/atelier-de-portage donnera ["http:", "", "localhost:8000", "cours", "atelier-de-portage"]

- **const courseName** => On récupère le dernier élément de l'URL séparée. En utilisant .length -1 on obtiendra toujours le dernier élément vu que l'indexation des tableaux en JavaScript commence à 0.

- **var startTime** => On définit l'heure de début pour la génération des créneaux horaires. Qu'on va paramétrer à 8heures, 0 minute, et 0 seconde.

- **var timeSlots** => On créer un tableau vide pour stocker les créneaux horaires

Ensuite on va faire une boucle pour créer les slots horaire.

```
while(startTime.getHours()<18 || startTime.getMinutes()<30){
var startTime_hours = startTime.getHours() > 9 ? startTime.getHours() : "0"+startTime.getHours();
var startTime_minutes = startTime.getMinutes() > 9 ? startTime.getMinutes() : "0"+startTime.getMinutes();
timeSlots.push(startTime_hours+":"+startTime_minutes);
startTime.setMinutes(startTime.getMinutes() + 30);
}
```

While continuera de s'exécuter tant que l'heure de startTime est inférieure à 18 ou que les minutes de startTime sont inférieures à 30. Au final la boucle s'arrêtera après avoir généré un créneau horaire pour 18h00.

Ensuite on va utiliser un opérateur ternaire pour déterminer comment formater l'heure. Si l'heure est supérieure à 9, elle utilise l'heure telle quelle. Sinon, elle ajoute un "0" devant l'heure pour la formater correctement (par exemple, "08" pour 8 heures).

On va utiliser la même logique pour formater les minutes. Et ajouter les créneaux horaire formatés au tableau timeSlots.

Ensuite on va augmenter les minutes de startTime de 30. Ca permet de passer au créneau horaire suivant. Si startTime était à 08h00, il passera donc à 08h30.

Une fois les horaires créées, on appelle la fonction **generateCalendar**().

```
function generateCalendar() {
var calendar = document.getElementById("calendar");

calendar.innerHTML = `···
</form>`;

// Présenter juste après
displayCalendarContent();

    const paymentMethodElements = document.querySelectorAll('.radio-button__label');
    paymentMethodElements.forEach(element => {
        element.addEventListener("click", function() {
            clearAllClasses();
            setPaymentButtonColor(element, courseName);
        });
    });

    const prevMonthSelector = document.getElementById("prev-month-calandar-selector");
    const nextMonthSelector = document.getElementById("next-month-calandar-selector");

    prevMonthSelector.addEventListener("click", function(){
        goBackOneMonth();
    })
    nextMonthSelector.addEventListener("click", function(){
        advanceOneMonth();
    })
}
```

La fonction **generateCalendar**() est chargée de générer le "squelette" du calendrier, tandis que **displayCalendarContent**() s'occupe de remplir ce squelette.

generateCalendar() va sélectionner l'élément HTML avec l'ID "calendar" et le stocker dans la variable calendar.
On va ensuite utiliser un “template literals” pour créer la structure du calendrier en HTML (on peut le voir sur le screenshot de cette section page 45)

Ce sont des chaînes de caractères dynamiques, qui peuvent être générées à partir de données ou d'autres expressions.

On ajoute ensuite des écouteurs d'événements aux éléments de méthode de paiement. Lorsqu'un utilisateur clique sur une méthode de paiement, les classes de tous les boutons sont effacées (pour réinitialiser les styles) et la couleur du bouton de paiement est définie en fonction de courseName (le slug) avec **setPaymentButtonColor**.

Pour finir, on ajoute des écouteurs d'événements aux boutons de navigation du calendrier.

```
function goBackOneMonth(){
actualDate.setMonth(actualDate.getMonth() - 1);
displayCalendarContent();
}
```

Lorsqu'un utilisateur clique sur le bouton du mois précédent, la fonction **goBackOneMonth**() est appelée, et lorsqu'il clique sur le bouton du mois suivant, la fonction **advanceOneMonth**() est appelée.

Une fois le squelette crée avec **generateCalendar**(), nous allons devoir le remplir, et c'est justement le rôle de **displayCalendarContent**(), qu'on a vu précédemment dans la function **generateCalendar**().

```
function displayCalendarContent(){

const actualDateParam = formatDate(actualDate);
var request = new XMLHttpRequest();

request.open("POST", "/days_unavailable/"+courseName);
request.setRequestHeader("Content-Type", "application/x-www-form-urlencoded");
```

Dans cette fonction nous allons appeler la fonction **formatDate**() pour formater la date actuelle **actualDate** et stocker le résultat dans **actualDateParam**.

Ensuite on initialise une requête **AJAX***. La requête est configurée pour envoyer une requête **POST** à l'URL /days_unavailable/ suivie du courseName (slug). Elle est également configurée pour envoyer des données au format "application/x-www-form-urlencoded".

* **AJAX (Asynchronous JavaScript and XML)** est une technique qui permet aux navigateurs web de communiquer avec un serveur après le chargement initial de la page, échangeant des données de manière asynchrone sans avoir à recharger la page. Elle rend les applications web plus réactives.

Ce format est le plus courant. Il est utilisé pour envoyer des données au format d'un formulaire HTML. Les données sont codées en URL, sous forme de paires nom/valeur. Il existe également d'autres format comme application/json, utilisé pour envoyer des données au format JSON ou application/xml pour le format xml.

`request.onreadystatechange = function() {` est une fonction de rappel qui sera exécutée chaque fois que l'état de la requête change. À l'intérieur de cette fonction, plusieurs actions sont effectuées :

```
request.onreadystatechange = function() {

    if (request.readyState === XMLHttpRequest.DONE && request.status === 200) {
        var unavailableDays = JSON.parse(request.responseText);
        console.log(unavailableDays);
        var calendarElement = document.getElementById("calendar");
            calendarElement.className = '';

            switch(courseName) {···
            }

        startDate = new Date(actualDate);
        startDate.setDate(actualDate.getDate() - actualDate.getDate() + 1);
        startDate.setDate(startDate.getDate() - (startDate.getDay()==0?7:startDate.getDay()) + 1);

        var calendarBody = document.getElementById("calendar-body");
        calendarBody.innerHTML = "";

        document.getElementById("current-month").innerHTML = getMonthName(actualDate.getMonth()) + " " + actualDate.getFullYear();

        var currentDate = new Date(startDate);
        var selectedDate = document.getElementById("calendar-form-date").getAttribute("value");
```

a. Analyse de la réponse : Si la requête est terminée et que le statut est 200 (ce qui signifie que la requête a réussi), la réponse est analysée en tant que JSON pour obtenir une liste des jours non disponibles. Cette liste est ensuite enregistrée dans la variable **unavailableDays**.

b. Mise à jour des styles du calendrier : Var calendarElement récupère l'élément du calendrier et réinitialise ses classes. Ensuite, en fonction du nom du cours, elle ajoute une classe spécifique pour styliser le calendrier. ici les couleurs des boutons et des hovers pour les dates et jours.

c. Définition de la date de départ : On va créer une nouvelle date stockée dans **startDate** qui va définir la première date à afficher dans le calendrier. On ajuste la date pour qu'elle commence au premier jour du mois et s'aligne sur le premier jour de la semaine.

Exemple : Si **actualDate** = au 15 novembre2023

- `startDate.setDate(actualDate.getDate() - actualDate.getDate() + 1);`
  - **startDate** sera le 1er novembre 2023.
- `startDate.setDate(startDate.getDate() - (startDate.getDay()==0?7:startDate.getDay()) + 1);`
  - Si le 1er novembre 2023 est un mercredi, cette ligne ajuste **startDate** pour être le lundi précédent, soit le 27 août 2023.

En bref, cela ajuste **startDate** pour qu'il soit le lundi précédant le 1er jour du mois de actualDate.

d. Génération du contenu du calendrier : Ensuite on va générer le contenu du calendrier. `calendarBody.innerHTML = "";` commence par effacer tout contenu existant dans calendarBody.

On récupère l'élément HTML ayant l'ID "**current-month**" et met à jour son contenu pour afficher le nom du mois et l'année de la date **actualDate**.

```
document.getElementById("current-month").innerHTML = getMonthName(actualDate.getMonth()) + " " + actualDate.getFullYear();
```

La fonction **getMonthName**() est utilisée pour convertir le numéro du mois (0-11) en son nom complet (Exemple, 10 pour "Novembre").

On récupère la valeur de l'attribut "**value**" de l'élément HTML ayant l'ID "calendar-form-date".

```
var selectedDate = document.getElementById("calendar-form-date").getAttribute("value");
```

Cette valeur représente la date actuellement sélectionnée par l'utilisateur dans le calendrier. On la stocke dans la variable **selectedDate**.

Ensuite, elle utilise une boucle pour générer chaque semaine du calendrier. Chaque “**tr**” représentant une semaine.

```
for (var i = 0; i < numWeeks; i++) {
    var row = document.createElement("tr");
```

À l'intérieur de cette boucle, une autre boucle est utilisée pour générer chaque jour de la semaine.

```
for (var j = 0; j < 7; j++) {
    var td = document.createElement("td");
    var cell = document.createElement("div");

    td.appendChild(cell);
    cell.innerHTML = currentDate.getDate();
    cell.classList.add("calandar-unit");
    cell.setAttribute("data-date", currentDate);

    var currentDateParam = formatDate(currentDate);
```

Chaque itération de la boucle interne crée une nouvelle cellule “**td**” pour représenter un jour dans la semaine. Un élément “**div**” est également créé et ajouté à cette cellule pour afficher des informations supplémentaires sur le jour. On peut ensuite afficher la date dans la cellule.

Ensuite on va ajouter une série de condition pour afficher des classes a cette cellule :

```
if(currentDateParam==selectedDate){
    cell.classList.add("selected");
}
```

selected pour la cellule qu’on va sélectionner(couleur du thème de la page dans la cellule), **mitigate** si la date ou l’heure est déjà réservée (cellule grisé) ou, si le jour actuel est dans la liste des jours non disponibles, la classe "unavailable" est ajoutée à la cellule **unavailable** pour les cellules non disponible (cellule barrée par un trait rouge).

```
if(unavailableDays.includes(`${currentDate.getDate()}`)){
    cell.classList.add("unavailable");
}
```

La fin de la requête permet d’ajouter la cellule à la ligne de la semaine, et la date actuelle est mise à jour pour passer au jour suivant.

```
row.appendChild(td);
currentDate.setDate(currentDate.getDate() + 1);
```

Après avoir généré tous les jours de la semaine, donc de la boucle interne, la ligne est ajoutée à l'élément calendarBody.

```
calendarBody.appendChild(row);
```

e. Pour finir, l’envoi de la requête AJAX : On prépare les données à envoyer avec la requête AJAX (la date actuelle formatée) et on envoie la requête.

```
var data = "date=" + encodeURIComponent(actualDateParam);
request.send(data);
```

Pour résumé, on envoie une requête **AJAX** pour récupérer les jours non disponible d’un cours spécifique. À la réception de ces données, elle ajuste le style du calendrier en fonction du type de cours et initialise la structure du calendrier. Elle parcourt chaque jour, en stylisant les jours non disponibles, et les jours sélectionnés. Des écouteurs d'événements sont ajoutés pour permettre des interactions utilisateur, comme la sélection d'un jour spécifique.

III. CONSTRUCTION DU SITE

Nous avons maintenant la génération du squelette du calendrier grâce a **generateCalendar** qui créer une structure HTML nécessaire pour afficher le calendrier, y compris les éléments pour le mois, les jours de la semaine, les boutons de navigation.

On peut afficher le contenu du calendrier grâce a **displayCalendarContent**(), elle envoie une requête **AJAX** pour récupérer les jours où un cours n'est pas disponible. Ensuite, elle remplit le calendrier avec les jours du mois, en marquant les jours non disponibles en conséquence.

Mais pour permettre a l'utilisateur de sélectionner une date, une heure et de récupérer ces informations pour la réservation, on va devoir créer une fonction pour permettre ça, et c'est le rôle de la 3ième fonctions importantes, **selectUnit**().

```
function selectUnit(e){

if(e.target.classList.contains("selected")){
    return false;
}

// Récupére la date de l'unité cliquée et la formate
var date = new Date(e.target.getAttribute("data-date"));
var dateParam = formatDate(date);
```

Dans un premier temps on va vérifier si la cible contient déjà la classe "**selected**", si c'est le cas la fonction s'arrête.
Ensuite, on va récupérer la date de l'unité cliqué, la récupérer et formater pour être utilisée dans les requêtes plus tard.

On va générer la mise à jour de l'interface utilisateur en fonction de la date sélectionnée.

```
// Désactive le bouton de soumission du formulaire du calendrier
document.getElementById("calendar-submit-button").setAttribute("disabled", "");

// Réinitialise le champ de sélection de l'heure
document.getElementById("calendar-form-time").setAttribute("value", "");

// Récupére toutes les unités du calendrier
var calendarUnits = document.querySelectorAll(".calandar-unit");
// Boucle sur toutes les unités pour enlever la classe 'selected'
for (var i = 0; i < calendarUnits.length; i++) {
    calendarUnits[i].classList.remove("selected");
}
💡 Ajoute la classe 'selected' à l'unité cliquée
e.target.classList.add("selected");

// Met à jour le champ date du form avec la date de l'unité sélectionnée
document.getElementById("calendar-form-date").setAttribute("value", dateParam);
```

III. CONSTRUCTION DU SITE

Dans cette partie, on va désactiver le bouton de soumission, réinitialiser le champ de sélection de l'heure, désélectionner les unités du calendrier précédentes et marquer uniquement celle sélectionnée. Enfin on met a jour le formulaire avec cette date.

Ensuite, on va préparer l'affichage des créneaux horaires en créant les éléments nécessaires, ici les titres "**h3**" et la liste des créneaux horaires "**ul**".

```
var slotList = document.getElementById("slot-list");
if(slotList){
    slotList.remove();
}

// Récupére le conteneur des créneaux horaires
const timeSlotsContainer = document.getElementById("slot-container");

// Créé le titre de la liste des créneaux horaires
const timSlotTitle = document.createElement("h3");
timSlotTitle.setAttribute("id", "slot-title");
// Créé la liste des créneaux horaires
const timeSlotsList = document.createElement("ul");
timeSlotsList.setAttribute("id", "slot-list");
```

Avant d'afficher les nouveaux créneaux horaires, les anciens créneaux (s'ils existent) sont supprimés.

Pour finir, une requête **AJAX** est initiée pour récupérer les créneaux horaires qui ne sont pas disponibles pour la date sélectionnée. Lorsque la requête est terminée et que la réponse est reçue, les créneaux horaires indisponibles sont extraits et traités.

```
// Création d'une nouvelle requête XMLHttpRequest pour récupérer les créneaux indisponibles
var request = new XMLHttpRequest();
request.open("POST", "/course_unavailability/"+courseName);
request.setRequestHeader("Content-Type", "application/x-www-form-urlencoded");

// Callback à exécuter quand la requête change d'état
request.onreadystatechange = function() {

    // Si la requête est terminée et que la réponse est 200 (OK
    if (request.readyState === XMLHttpRequest.DONE && request.status === 200) {

        // On récupére la liste des créneaux indisponibles à partir de la réponse de la requête
        var unavailableSlots = JSON.parse(request.responseText);
```

```
        // Boucle sur tous les créneaux horaires
        for (var k = 0; k < timeSlots.length; k++) {

            // Créé un élément de la liste pour chaque créneau horaire
            var slot = document.createElement("li");
            slot.classList.add("slot");
            slot.innerHTML = timeSlots[k];

            // Si le créneau est dans la liste des créneaux indisponibles ou tous indisponibles, ajouter la classe 'unavailable'.
            if(unavailableSlots.includes(timeSlots[k]) || unavailableSlots.includes("all day")){
                slot.classList.add("unavailable");
            }
            // Sinon, ajoute la classe 'available'.
            else{
                slot.classList.add("available");
            }
            // Ajoute le créneau à la liste
            timeSlotsList.appendChild(slot);
            // Ajoute un écouteur d'événement pour gérer le click sur le créneau
            slot.addEventListener("click", selectTimeSlot);
        }
    }
};
// Prépare les données à envoyer avec la requête
var data = "date=" + encodeURIComponent(dateParam);
// Envoye la requête avec les données
request.send(data);
}
```

Si la réponse est 200, on va à afficher une liste de créneaux horaires, en marquant visuellement ceux qui sont disponibles et ceux qui ne le sont pas.
De plus, on permet à l'utilisateur de sélectionner un créneau horaire en cliquant dessus, déclenchant ainsi une action ou une mise à jour de l'interface.
Enfin, on prépare et encode les données pour les envoyer. Ensuite, la requête est envoyée au serveur.

Pour résumer le fonctionnement global de ce calendrier repose principalement sur trois fonctions clés : **generateCalendar()**, **displayCalendarContent()** et **selectUnit()**.

La fonction **generateCalendar()** est chargée de générer le "squelette" du calendrier, tandis que **displayCalendarContent()** s'occupe de remplir ce "squelette" avec les dates et les informations. Enfin, **selectUnit()** permet à l'utilisateur de sélectionner une date spécifique sur le calendrier.

Cependant, il existe également d'autres fonctions qui jouent un rôle importantes dans le fonctionnement détaillé du calendrier.

Par exemple les fonctions **goBackOneMonth()** et **advanceOneMonth()** permettent à l'utilisateur de naviguer facilement entre les mois, en revenant au mois précédent ou en avançant au mois suivant.

La fonction **selectTimeSlot()** offre la possibilité de sélectionner un créneau horaire spécifique, ce qui est essentiel pour la fonction de réservation du calendrier.

III. CONSTRUCTION DU SITE

1 E. Le système de réservation

Une fois le calendrier opérationnel, l'étape suivante consiste à permettre à l'utilisateur de réserver un cours. Pour ce faire, nous allons exploiter les champs cachés mentionnés précédemment :

```
<input type="hidden" name="courseName" value="${courseName}">
<input type="hidden" name="slotDate" id="calendar-form-date" value="">
<input type="hidden" name="slotTime" id="calendar-form-time" value="">
```

On va pouvoir ainsi récupérer le nom du cours, le jour et l'heure, et les utiliser pour créer une réservation grâce à la fonction save_reservation().

```
#[Route('/save_reservation', name: 'save_reservation')]
public function save_reservation(Security $security, Request $request, EntityManagerInterface $em): Response
{
    // On utilise le service Security pour obtenir l'utilisateur actuellement connecté et on le stocke dans la variable
    $user = $security->getUser();
    // On récupère le nom du cours à partir de la requête HTTP et on le stock
    $courseSlug = filter_var($request->request->get('courseName'), FILTER_SANITIZE_SPECIAL_CHARS);
    // On utilise l'EM, on récupère le chemin pour l'entité Cours, on cherche le cours correspondant au slug.
    $course = $em->getRepository("App\Entity\Cours")->findOneBy(['slug_cours' => $courseSlug ]);
    // On vérifie si un utilisateur est connecté.
    if (!$user) {
        // On utilise l'objet Request pour obtenir la session en cours.
        $session = $request->getSession();
        // On stocke l'URL que l'utilisateur avait l'intention de visiter dans la session pour la redirection
        $session->set("intentedUrl", "cours/".$courseSlug);
        // On redirige l'utilisateur vers la page de connexion.
        return $this->redirectToRoute('app_login');
    }
    // On récupère la date du créneau à réserver à partir de la requête HTTP.
    $date = $request->request->get('slotDate');
    // On récupère la lheure du créneau à réserver à partir de la requête HTTP.
    $time = $request->request->get('slotTime');
```

Dans cette fonction, on commence par récupérer l'utilisateur et le slug du cours et le stocker. Ensuite, on cherche dans la base de données le cours correspondant à ce slug. Parallèlement, on vérifie si l'utilisateur est connecté. Si ce n'est pas le cas, on le redirige vers la page de connexion, car une connexion est nécessaire pour effectuer une réservation. On récupère également les créneaux pour la date et le jour.

Ensuite, on crée une nouvelle réservation et définit ses attributs, y compris le cours, l'utilisateur, la date de réservation et la date du cours. La méthode de paiement est également définie en fonction de ce que l'utilisateur a spécifié dans la requête **HTTP** et on initialise **IsPaid** à 0.

```
$reservation = new Reservation;

// On définit les attributs de la réservation
$reservation->setCours($course);
//user récupéré avec getUser
$reservation->setUser($user);
//date actuelle de la resa (aujorud'hui)
$reservation->setDtResa(new DateTime());
//date du cours selectionné avec les slotDate et slotTime plus haut (jour + heure)
$reservation->setDtCours(new DateTime($date." ".$time));
$reservation->setPayementMethod($request->request->get('payment-method'));
$reservation->setIsPaid(0);
```

La fonction vérifie également la méthode de paiement de la réservation et redirige l'utilisateur vers la page de paiement si la méthode de paiement est "**payment-online**", ou vers la page du cours si la méthode de paiement n'est pas "**payment-online**".

```php
// On enregistre et flush la reservation en DB
$em->persist($reservation);
$em->flush();

//  On vérifie si la méthode de paiement de la réservation est "payment-online"
if ($reservation->getPayementMethod() == "payment-online") {
    return $this->redirectToRoute("user_details", ['id' => $reservation->getId()]);
}
// Si la méthode de paiement n'est pas "payment-online", on redirige l'utilisateur vers le cours après payement.
return $this->redirect("cours/".$courseSlug);
```

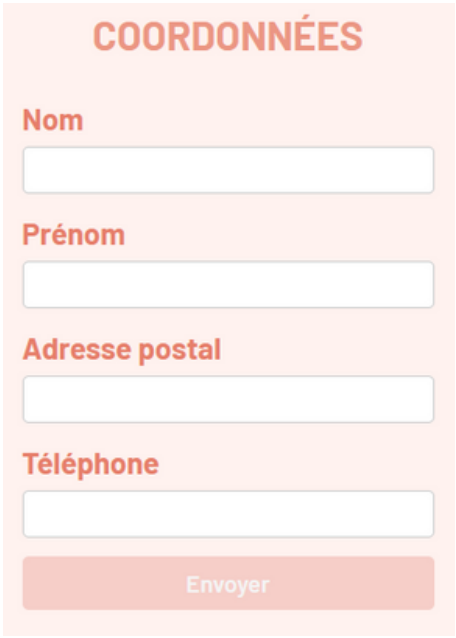
Pour finir, la réservation est ensuite enregistrée dans la base de données en utilisant l'**EntityManager**.

Une fois la réservation via le calendrier faite, l'utilisateur est redirigé vers un formulaire pour qu'il fournisse les coordonnées nécessaires a la rédaction de la facture et de la commande.

```php
#[Route('/user_details/{id}', name: 'user_details')]
public function userDetails($id, Request $request, EntityManagerInterface $em): Response
{
    // Récupère la réservation existante en utilisant l'ID
    $reservation = $em->getRepository(Reservation::class)->find($id);
    // Vérifie si la réservation existe
    if (!$reservation) {
        throw $this->createNotFoundException('Aucune réservation trouvée pour cet ID');
    }
    // Créer le formulaire
    $form = $this->createForm(UserDetailsFormType::class, $reservation);
    $form->handleRequest($request);
    if ($form->isSubmitted() && $form->isValid()) {
        // update de la réservation existante
        $em->flush();

        // Redirige vers page de paiement
        return $this->redirectToRoute("reservation_payment", ['id' => $reservation->getId()]);
    }

    return $this->render('payment/userDetails.html.twig', [
        'form' => $form->createView(),
        'reservation' => $reservation
    ]);
```



Une fois ce formulaire envoyé, nous avons toute les informations nécessaires pour la réservation en base de donnée, il ne reste plus qu'a récupérer les informations bancaires grace au formulaire charger avec Stripe payment-element et les envoyer au serveur Stripe.

```php
#[Route('/payment-success/{id}', name: 'success_payment')]
public function paymentSuccess(int $id, ReservationRepository $reservationRepository, EntityManagerInterface $em): Response
{
    $reservation = $reservationRepository->find($id);

    if (!$reservation) {
        throw $this->createNotFoundException('La réservation demandée n\'existe pas.');
    }
    // Mettre à jour le champ isPaid de la réservation
    $reservation->setIsPaid(1);
    $em->persist($reservation);
    $em->flush();
    // ***************
    // Génére le PDF
    $html = $this->renderView('invoice/invoice.html.twig', [
        'reservation' => $reservation,
    ]);
    $pdfContent = $this->pdfGenerator->generatePdf($html);

    // Sauvegarde le PDF
    $pdfPath = 'C:\Users\guillaumekezic.AzureAD\Desktop\invoice_' . $reservation->getId() . '.pdf';

    file_put_contents($pdfPath, $pdfContent);
    // ***************
    // Rendre la vue
    return $this->render('payment/success.html.twig');
}
```

Dans cette fonction on va récupérer l'id de la réservation en cours, mais surtout passer le booléen **IsPaid** de 0 à 1.
Plus tard et après l'intégration de Stripe, je pourrais faire passer **IsPaid** a 1 lorsque je recevrais la réponse de Stripe.
Je pourrais également mettre une limite de temps (de 10 min par exemple) pour compléter les informations bancaires à partir du moment ou **IsPaid** est set a 0, donc le début de la réservation en base de donnée.(en utilisant un Cron Job)

Désormais, l'utilisateur a la possibilité de sélectionner un cours, d'utiliser le calendrier pour choisir une date et une heure, et de finaliser sa réservation une fois le mode de paiement choisi.
Pour la gestion des paiements, j'ai opté pour l'utilisation de **Stripe**. Je vais expliquer son fonctionnement dans la section suivante.

1 F. Le moyen de payement

J'ai rapidement envisagé d'utiliser **Stripe** comme solution de paiement. En effet, j'avais déjà eu l'occasion de l'utiliser dans un projet précédent durant ma formation. Mais, j'avais utilisé une méthode plus simple et "guidée". Pour ce projet, j'ai décidé de réutiliser **Stripe**, mais en choisissant cette fois-ci la méthode la plus complète, offrant ainsi une plus grande personnalisation.
C'est un moyens de paiement en ligne très populaires, ce qui est un atout majeur pour les utilisateurs afin de renforcer leur confiance lors du payement sur le site

III. CONSTRUCTION DU SITE

Stripe utilise un système d'**endpoints** et de **webhooks** pour gérer les paiements.
L'**endpoint** est une URL spécifique de notre serveur qui communique avec Stripe via une API* sécurisée. Il permet d'envoyer des demandes spécifiques, comme initier un paiement.

Le **webhook**, est un mécanisme qui permet à Stripe de communiquer avec notre application. Lorsqu'un événement se produit (par exemple, un paiement est accepté ou refusé), Stripe envoie une notification à une URL spécifique (le **webhook**) sur notre serveur. Notre serveur peut alors traiter cette information, mettre à jour l'état de la commande dans notre système, informer l'utilisateur, etc.
En résumé, l'**endpoint** est utilisé pour envoyer des demandes à Stripe, et le **webhook** est utilisé pour recevoir des notifications de Stripe concernant les événements liés au paiement.

Ensemble, ils forment un "pont" sécurisé qui permet une communication bidirectionnelle entre notre application et Stripe.
Avant de commencer a créer le **Endpoint** et le **Webhook**, une mise en place des clés sont nécessaire.

Configuration des Clés:

- Clé Secrète: Configurer la clé secrète de Stripe.
- Clé Publique: Configurer la clé publique de Stripe.

```php
<?php

define("STRIPE_SECRET_KEY", "sk_test_
                            ");
```

Pour la clé secrète un fichier secret.php a été créer afin de pouvoir sécurisé cette clé, ou pour la modifier qu'une seule fois si je dois la changer, car elle peut être utilisée a plusieurs endroit.
Quant à la clé publique, elle sera inscrite dans le fichier JavaScript gérant le paiement, en tant que constante.

```
const stripe = Stripe("pk_test_51NAwNZJWIv9Iz53oUTcDu6QGU8c29nvuRPQ0jLe4WNom3WJmq6PBi4FTYmVfEq32lGuLGZQtc4voWr1iSbipNP5ce00ZSG7OWON");
```

***API (Interface de Programmation d'Application) :** c'est un ensemble de règles qu'une application peut suivre pour accéder et utiliser les fonctionnalités ou les données d'un autre service, logiciel ou plateforme. Elle sert de "pont" entre différents logiciels, afin qu'ils puissent communiquer entre eux.

III. CONSTRUCTION DU SITE

Les clés sont disponibles sur le dashboard de Stripe.

Le **Endpoint** se compose de plusieurs parties :

- **Configurer la clé secrète de Stripe :** `Stripe::setApiKey(STRIPE_SECRET_KEY);`

La clé secrète de Stripe est définie pour authentifier notre application auprès de Stripe.

- **Définition de l'endpoint secret :**

C'est la clé secrète qui est utilisée pour vérifier la signature de la requête entrante de Stripe, afin de s'assurer qu'elle provient réellement de Stripe.

```
$endpoint_secret = 'whsec_2                                                    ';
```

- **Récupération du payload et de la signature :**

```php
$payload = @file_get_contents('php://input');
$sig_header = $_SERVER['HTTP_STRIPE_SIGNATURE'];
$event = null;
```

Le contenu de la requête (payload) et la signature de la requête sont récupérés.

- **Construction de l'événement :**

```php
try {
$event = \Stripe\Webhook::constructEvent(
    $payload, $sig_header, $endpoint_secret
);
} catch(\UnexpectedValueException $e) {
http_response_code(400);
exit();
} catch(\Stripe\Exception\SignatureVerificationException $e) {
http_response_code(400);
exit();
}
```

Stripe fournit une signature pour chaque événement qu'il envoie au webhook. Cette partie essaye de construire un événement à partir du payload, de la signature de l'en-tête et d'un endpoint secret. Si la signature est valide, cela signifie que l'événement provient de Stripe et n'a pas été modifié pendant l'envoie

III. CONSTRUCTION DU SITE

La méthode constructEvent est utilisée pour construire l'événement à partir du payload, de la signature et de l'endpoint secret. Si la signature est invalide ou si le payload est mal formé, une exception sera levée et une réponse d'erreur sera renvoyée.

Gestion de l'événement :

```php
switch ($event->type) {
case 'payment_intent.succeeded':
    echo 'Payment success';
    $paymentIntent = $event->data->object;
default:
    echo 'Received unknown event type ' . $event->type;
}
```

L'événement est ensuite traité en fonction de son type. Ici le type d'événement est payment_intent.succeeded, et un message de succès est affiché.

Réponse :

```php
return new Response('Success', 200);
```

Une réponse avec le statut 200 est renvoyée à Stripe pour indiquer que l'événement a été reçu et traité avec succès.

Une fois le **endpoint** et le **webhook** configuré, il faut maintenant gérer le processus de paiement côté client, y compris la création et la confirmation des intentions de paiement, et l'affichage des messages à l'utilisateur.

Je vais détailler les plus importantes parties ici.

- **Initialisation de Stripe :**

```
const stripe = Stripe("pk_test_51NAwNZJWIv9Iz53oUTcDu6QGU8c29nvuRPQ0jLe4WNom3WJmq6PBi4FTYmVfEq32lGuLGZQtc4voWr1iSbipNP5ce00ZSG7OWON");
```

La clé publique est utilisée pour initialiser l'objet Stripe.

- **Récupération des informations de réservation et de l'utilisateur :**

```
const reservationId = document.getElementById('reservationId').getAttribute('value');
const userMail = document.getElementById('userMail').getAttribute('value');
```

Le code récupère les valeurs d'identifiant de réservation et de courriel de l'utilisateur à partir des éléments **HTML**.

- **Initialisation du formulaire de paiement :**

```
async function initialize() {
                                    // API fetch
  const { clientSecret } = await fetch("/create-payment-intent", {
    method: "POST",
    headers: { "Content-Type": "application/json" },
    body: JSON.stringify({ reservationId }),
  }).then((r) => r.json());

  elements = stripe.elements({ clientSecret });

  const paymentElementOptions = {···
  };

  const paymentElement = elements.create("payment", paymentElementOptions);
  paymentElement.mount("#payment-element");
}
```

La fonction **initialize**() fait un appel **AJAX** à l'endpoint /create-payment-intent pour obtenir un clientSecret, puis initialise les éléments de Stripe et crée un élément de paiement.

- **Gestion de la soumission du formulaire :**

```
document
  .querySelector("#payment-form")
  .addEventListener("submit", handleSubmit);

async function handleSubmit(e) {
  e.preventDefault();
  setLoading(true);

  const { error } = await stripe.confirmPayment({
    elements,
    confirmParams: {
      return_url: "http://localhost:8000/payment-success",
    },
  });
```

On joute un écouteur d'événement pour gérer la soumission du formulaire de paiement et confirmer le paiement avec Stripe.

- **Vérification du statut de l'intention de paiement :**

```
async function checkStatus() {
  const clientSecret = new URLSearchParams(window.location.search).get("payment_intent_client_secret");
  if (!clientSecret) {
    return;
  }
  const { paymentIntent } = await stripe.retrievePaymentIntent(clientSecret);
  switch (paymentIntent.status) {
    case "succeeded":
      showMessage("Payment succeeded!");
      break;
    case "processing": ···
    case "requires_payment_method": ···
    default: ···
  }
}
```

La fonction **Checkstatus**() vérifie le statut de l'intention de paiement en interrogeant Stripe et affiche un message en fonction du statut.

- **Affichage des messages et gestion de l'UI :**

```
function showMessage(messageText) {
  const messageContainer = document.querySelector("#payment-message");

  messageContainer.classList.remove("hidden");
  messageContainer.textContent = messageText;
```

La function **showMessage**() va s'occuper d'afficher les messages à l'utilisateur.

```
function setLoading(isLoading) {
  if (isLoading) {
    document.querySelector("#submit").disabled = true;
    document.querySelector("#spinner").classList.remove("hidden");
    document.querySelector("#button-text").classList.add("hidden");
  } else {
    document.querySelector("#submit").disabled = false;
    document.querySelector("#spinner").classList.add("hidden");
    document.querySelector("#button-text").classList.remove("hidden");
  }
}
```

Alors que **setLoading**() s'occupe de gérer l'état de chargement de l'interface utilisateur.

En résumé, JavaScript va gérer le processus de paiement côté client, tandis que le **WebhookController** gère les notifications de Stripe côté serveur. Ensemble, ils permettent de gérer les paiements dans notre application.

III. CONSTRUCTION DU SITE

1 G Espace administrateur

Pour la gestion de mon interface administrateur, j'ai choisi d'utiliser **EasyAdmin**. Ayant déjà eu l'occasion de l'utiliser à plusieurs reprises durant ma formation, je trouve que c'est un outil à la fois rapide et facile pour la gestion des tâches administratives. De plus, **EasyAdmin** se révèle être un excellent moyen d'initier les clients, souvent novices en la matière, à son fonctionnement. Ainsi, ils peuvent eux-mêmes réaliser certaines tâches de manière simple et autonome.

EasyAdmin fonctionne de façon assez simple, on créer un dashboard :

```php
public function configureDashboard(): Dashboard
{
    return Dashboard::new()
        ->setTitle('Douceur De Lune');
}
```

et on le configure selon ses besoins :

```php
public function configureMenuItems(): iterable
{
    yield MenuItem::linkToDashboard('Dashboard', 'fa fa-home');

    yield MenuItem::linkToCrud('Cours', 'fas fa-solid fa-arrow-right', Cours::class);

    yield MenuItem::linkToCrud('Payement', 'fas fa-solid fa-arrow-right', Payement::class);

    yield MenuItem::linkToCrud('Reservation', 'fas fa-solid fa-arrow-right', Reservation::class);

    yield MenuItem::linkToCrud('User', 'fas fa-solid fa-arrow-right', User::class);

    yield MenuItem::linkToCrud('UnavailableDate', 'fas fa-solid fa-arrow-right', UnavailableDate::class);
}
```

Ici, `public function configureMenuItems(): iterable` est une méthode qui retourne un ensemble d'éléments de menu. La méthode doit retourner un objet itérable, ce qui signifie qu'on peut utiliser une boucle foreach pour parcourir les éléments.

Ensuite, on va créer le lien du dashbord de l'application :

```php
yield MenuItem::linkToDashboard('Dashboard', 'fa fa-home');
```

III. CONSTRUCTION DU SITE

Pour finir, on va créer un lien vers la page **CRUD** (Create, Read, Update, Delete) pour l'entité que l'on souhaite modifier, ici Cours :

```
yield MenuItem::linkToCrud('Cours', 'fas fa-solid fa-arrow-right', Cours::class);
```

Pour approfondir un peu, `Cours::class` fait référence à la classe de l'entité Cours dans notre application. EasyAdmin, va automatiquement utiliser les informations contenues dans cette classe pour générer les formulaires et les vues nécessaires à la gestion des objets de type Cours.

Sans **EasyAdmin** nous devrions créer une function comme ceci dans le Controller:

```php
public function add(Request $request, EntityManagerInterface $em): Response
{
    $cours = new Cours();
    $form = $this->createForm(CoursType::class, $cours);
    $form->handleRequest($request);

    if ($form->isSubmitted() && $form->isValid()) {
        $em->persist($cours);
        $em->flush();

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

    return $this->render('nomDeLaVue/add.html.twig', [
        'form' => $form->createView(),
    ]);
}
```

Et un formulaire, par exemple CoursType comme ceci :

```php
public function buildForm(FormBuilderInterface $builder)
{
    $builder
        ->add('name')
        ->add('price')
        // etc..
    ;
}

public function configureOptions(OptionsResolver $resolver)
{
    $resolver->setDefaults([
        'data_class' => Cours::class,
    ]);
}
```

EasyAdmin simplifie grandement la gestion des opérations **CRUD** en proposant une interface visuelle pratique. Il automatise la création des formulaires et des vues nécessaires, permettant un gain de temps dans le développement et la gestion des données.

IV. SEO ET SÉCURITÉ

1 SEO

SEO (Search Engine Optimization) : L'objectif du **SEO** est d'améliorer le classement d'un site web sur des mots-clés pertinents afin d'attirer davantage de personnes et, par conséquent, d'augmenter sa popularité et/ou ses ventes. Le **SEO** englobe de nombreux aspects, allant de la structure et le contenu d'un site à sa performance et à la qualité de ses liens entrants.
Dans mon projet, j'ai utilisé plusieurs pratiques afin d'augmenter la visibilité et le référencement du site.

Balise meta description : Les balises meta description sont importantes, car Google peut les utiliser comme extraits pour représenter les pages dans les résultats de recherche Google.

```
{% block metaDescription %}
<meta name="description" content="Douceur de lune est un site proposant des cours d'allaitement, de portage de relaxation
{% endblock %}
```

Balises alt : L'optimisation des noms de fichiers d'images et du texte "**alt**" aide au référencement de recherche d'images, comme Google Images, à mieux les interpréter, et donc aide à la visibilité du site.

```
alt="Image d'une femme qui tient sont enfant contre elle dans une écharpe
```

URL distinctes : Les **URL** trop complexes, notamment celles qui contiennent de nombreux paramètres, peuvent gêner l'exploration, c'est une des raisons pour lesquelles j'ai utilisé un système de slug, afin de garder des **URL** lisibles et compréhensibles.

localhost:8000/cours/consultation-d-allaitement

Utilisation des balises <h1> : Les moteurs de recherche, comme Google, utilisent les balises **<h1>** pour comprendre la structure d'une page et identifier son sujet principal. Une balise **<h1>** pertinente donne aux moteurs de recherche une indication claire sur le contenu principal de la page.

Design responsive : Google utilise principalement la version mobile du contenu d'un site web pour l'**indexation** et le classement. Si un site n'est pas adapté aux dispositifs mobiles, cela peut affecter négativement son classement dans les résultats de recherche.

IV. SEO ET SÉCURITÉ

Design responsive : Google utilise principalement la version mobile du contenu d'un site web pour l'**indexation** et le classement. Si un site n'est pas adapté aux dispositifs mobiles, cela peut affecter négativement son classement dans les résultats de recherche, voir ne pas être référencé du tout.

Voici un screenshot du résultat de lighthouseX de Microsoft Edge, les points manquants en SEO sont du au projet qui est en localhost et donc de l'abscence des recherches avec le fichier robots.txt

Performances : Ce score évalue la rapidité avec laquelle la page se charge et devient interactive.
Accessibilité : Cette catégorie mesure si le site est accessible à tous les utilisateurs, y compris ceux ayant des handicaps.
Bonnes pratiques : Cette catégorie évalue divers aspects de la qualité du code et de la sécurité.
SEO : Ce score évalue la facilité avec laquelle les moteurs de recherche peuvent trouver et indexer le site.

2 Sécurité

2 A Failles XSS :

Les failles XSS (Cross-site Scripting), ou failles d'injection de code, sont des vulnérabilités qui permettent a un attaquant d'introduire du code malveillant via des entrées côté client.

En exploitant ces failles, un attaquant peut manipuler une application web pour, par exemple, voler des cookies ou des jetons de session, modifier des données, etc... Les conséquences d'une attaque XSS peuvent être dévastatrices et parfois irréparables. Il est donc essentiel de protéger le site contre de telles menaces.

IV. SEO ET SÉCURITÉ

Pour expliquer ce qu'est une session et un jeton de session :
Une session est une manière de stocker des informations pour un utilisateur durant sa navigation sur un site. Elles sont souvent utilisées pour garder en mémoire les informations d'identification des utilisateurs, ce qui permet aux utilisateurs de naviguer sur un site sans avoir à se reconnecter à chaque page. Les données de session sont stockées côté serveur, et un cookie contenant un identifiant de session unique est envoyé au navigateur de l'utilisateur pour maintenir un suivi de la session à travers les différentes requêtes.

Dans l'exemple suivant, j'ai utilisé le composant Form de Symfony pour définir la structure et la validation du formulaire de contact : (ici pour la partie message)

```
->add('message', TextareaType::class, [
    'constraints' => [
        new Assert\Length([
            'min' => 5,
            'minMessage' => 'Votre message doit avoir au moins {{ limit }} caractères.',
            'max' => 500,
            'maxMessage' => 'Votre message ne doit pas dépasser {{ limit }} caractères.'
        ]),
        new NotBlank(['message' => 'Entrez un message']),
        new Regex([
            'pattern' => '/^[a-zA-Z0-9\s.,?!\'"\n\r-]{5,500}$/',
        ]),
    ]
])
```

Ici, on va s'assurer que le message a une longueur comprise entre 5 et 500 caractères, que le message ne peut pas être vide, enfin, limiter les caractères acceptés pour le message, grâce à la contrainte **Regex,** pour le TextareaType.

L'utilisation de **Twig** va jouer un rôle important dans la lutte contre les failles **XSS**.
Twig échappe automatiquement les sorties pour éviter l'exécution de scripts malveillants.

```
{{ form_label(contact_form.message) }}
```

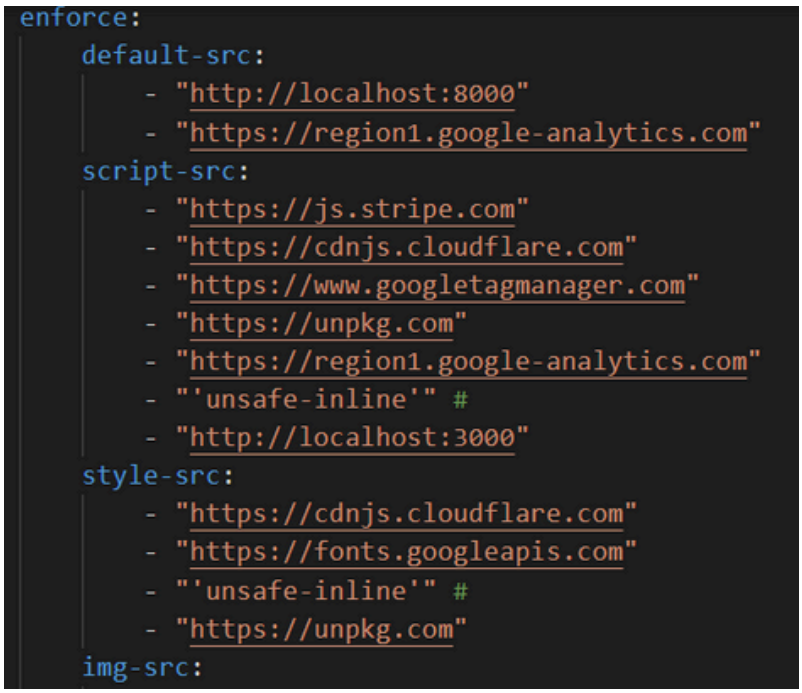
Ici, si quelqu'un essayait de soumettre un script malveillant via le champ message, **Twig** échapperait automatiquement cette entrée, empêchant le script de s'exécuter.

Grâce à la combinaison de la validation côté serveur et de l'échappement automatique des sorties côté client par **Twig**, nous avons une protection robuste contre les failles **XSS**.

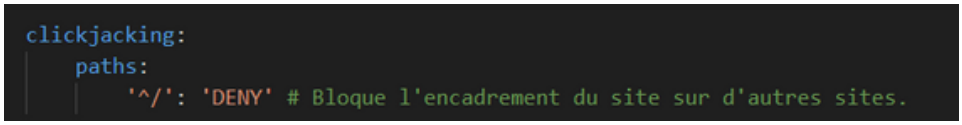
IV. SEO ET SÉCURITÉ

En plus de cette combinaison, j'ai mis en place un CSP, qui est l'une des meilleurs mesures contre les failles **XSS**

- **Utilisation d'un CSP** : Un Content Security Policy est une mesure de sécurité utilisée par les navigateurs web pour empêcher un large éventail d'attaques, notamment les attaques par injection de code, comme les attaques de type cross-site scripting (XSS).



Le screenshot si dessus montre un exemple de **CSP** réalisé avec le bundle nelmio_security. On peut y inclure toutes les sources de contenu autorisées, ce qui bloquera automatiquement les autres.



Nelmio_security protège également du **Clickjacking,** une attaque qui trompe l'utilisateur pour qu'il clique sur quelque chose de différent de ce que l'utilisateur perçoit. En général une iframe transparente supperposée sur le site web.

Ici, "DENY" signifie que le site web ne peut pas être encadré, même pas par lui-même. Cela empêche essentiellement toute autre page web d'encadrer le site à l'aide d'une iframe, ce qui protège contre les attaques de **clickjacking**

** https://developer.mozilla.org/fr/docs/Web/HTTP/CSP

2 B Failles CSRF :

Une faille **CSRF** (Cross-Site Request Forgery) est une attaque qui “trompe” un utilisateur pour qu'il exécute des actions non voulues sur un site web, en utilisant sa session active. Ce qui pourrait conduire à du vol d’information ou monétaire ou encore à des modifications non désirées dans les paramètres du compte utilisateur.

Pour contrer cette faille, l'utilisation d'un token **CSRF** est recommandé. Ce mécanisme est intégré nativement dans le bundle security de Symfony et fonctionne en 4 parties importantes.

- **1. Génération / Récupération du Token :** Lors de l'affichage du formulaire de connexion dans le template Twig, le token est généré ou récupéré via la fonction csrf_token('authenticate') et est stocké côté serveur.

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

- **2. Placement dans le champ caché :** Il est également insérer dans le champ caché du formulaire, afin qu'il soit envoyé lors de la soumission du formulaire.
- **3. Soumission du formulaire :** Quand l’utilisateur remplit le formulaire et l'envoie. Le token **CSRF**, qui est du coup en champ caché, est également envoyé dans la requête POST.
- **4. Validation du Token :** Lorsque le formulaire est soumis, la méthode authenticate créer un Passport qui contient entre autres un CsrfTokenBadge.

```
return new Passport(
    new UserBadge($email),
    new PasswordCredentials($request->request->get('password', '')),
    [
        new CsrfTokenBadge('authenticate', $request->request->get('_csrf_token')),
    ]
);
```

Ce badge représente la vérification du token CSRF. Le système de sécurité de Symfony va comparer le token CSRF fourni dans la requête (celui que l'utilisateur a soumis) avec le token CSRF stocké côté serveur pour cette session et cette intention ('authenticate' dans ce cas).
Voici une capture d’écran du token dans l’inspecteur.

IV. SEO ET SÉCURITÉ

```
<input id="contact__token" type="hidden" name="contact[_token]" value="5182e281ae89b7c7dff9630.9vBgeIRgLRwsqxELN3RuBC594h-wGJry5nfv…c.oIkCG80CQndi-Elffw5dW0M_ulqJd67flA6lcbaqxNfGujk_sRcaf0iaJQ">
```

```php
// Générer et stocker un token CSRF
if (empty($_SESSION['csrf_token'])) {
    $_SESSION['csrf_token'] = bin2hex(random_bytes(32));
}

$token = $_SESSION['csrf_token'];

// Vérification du token CSRF lors de la soumission du formulaire
if ($_SERVER['REQUEST_METHOD'] === 'POST') {
    // Compare le token POST avec le token de la session
    if (!isset($_POST['csrf_token']) || $_POST['csrf_token'] !== $_SESSION['csrf_token']) {
        die("CSRF token validation failed.");
    }
}
```

Sans symfony , il faudrait créer un token de cette manière :
En créant et en générant un token CSRF.
En utilisant $_SESSION['csrf_token'] = bin2hex(random_bytes(32)); pour le stocker dans la session et pour créer une chaine de caractère aléatoire et unique.
Lorsque le formulaire est soumis, le serveur vérifie que le token CSRF a été envoyé avec la requête.
Et comme avec symfony, le serveur compare ensuite le token CSRF soumis avec la valeur stockée dans la session

2 C. Attaque par dictionnaire et force brute :

L'attaquant utilise un fichier contenant de nombreux mots (un "dictionnaire") ou en essayant toutes les combinaisons possibles de caractères et tente de se connecter en utilisant chaque mot comme mot de passe. Le risques et de voler les informations personnelles d'un utilisateur mais aussi d'accéder au sytème administrateur.

Pour lutter contre ces failles, la **REGEX** mise en place va être particulièrement utile, en effet avec 14 caractères imposés, une majuscule et un chiffre, ca prendrait à un ordinateur, 9 million d'années selon security.org :

IV. SEO ET SÉCURITÉ

J’ai également mis en place un système de “ban temporaire” en utilisant un **honeypot**.
Un **honeypot** et un moyen de piéger une machine ou un utilisateur malveillant. Il s’agit d’un champ caché dans les formulaires, qui n’ai normalement pas remplissable par des utilisateurs bienveillants, en revanche une machine qui a pour but de forcer certains formulaire, vont remplir automatiquement tous les champs, et donc le **honeypot**.

```php
// On vérifie la soumission précédente
$session = $request->getSession();
$lastWeirdSubmission = $session->get('last_weird_submission');
if ($lastWeirdSubmission && (time() - $lastWeirdSubmission < 3600)) {  // 1h
    $this->addFlash('error', 'Veuillez attendre encore quelques minutes avant de réessayer.');
    return $this->redirectToRoute('app_home');
}

//Si le honeypot est rempli
if ($form->get('honeypot')->getData()) {
    // Surement un robot donc redirection et message
    $session->set('last_weird_submission', time());
    $this->addFlash('error', 'Veuillez attendre quelques minutes avant de réessayer.');
    return $this->redirectToRoute('app_home');
}
```

On peut voir sur le screenshot qu’on récupère la session de l'utilisateur actuel et on essaie d'obtenir la valeur associée à la clé “last_weird_submission”. Cette valeur, si elle existe, représente le moment (en timestamp UNIX*) de la dernière soumission “suspecte”.
Si une soumission suspecte a été précédemment enregistrée et que moins de 3600 secondes (1 heure) se sont écoulées depuis cette soumission, l'utilisateur reçoit un nouveau message qui lui demande d'attendre avant de réessayer. L'utilisateur est alors redirigé vers la page d'accueil.

Ensuite
```php
//Si le honeypot est rempli
if ($form->get('honeypot')->getData()) {
```
vérifie si le champ **honeypot** du formulaire est rempli.

Si c’est le cas, le moment actuel est enregistré dans la session sous la clé last_weird_submission. Ceci est fait pour se **souvenir** que cette soumission était suspecte et pour potentiellement empêcher l'utilisateur (ou le robot) de soumettre à nouveau dans le futur (3600 secondes ici). L'utilisateur est ensuite informé qu'il doit attendre avant de réessayer et est redirigé vers la page d'accueil.
On a donc un moyen efficace de lutter contre cette faille en combinant la **REGEX** et le honeypot.

*C’est une représentation du temps qui indique le nombre de secondes depuis le 1er janvier 1970 à 00:00:00 UTC

IV. SEO ET SÉCURITÉ

2 D Injection SQL

Lorsqu'un utilisateur envoie des requêtes, l'application web interroge la base de données afin de construire la réponse. Cependant, lorsque les informations fournies par l'utilisateur sont utilisées pour forger la requête à la base de données, un attaquant peut modifier cette dernière en l'utilisant pour injecter du code. Ainsi, cela permet à un attaquant d'interroger la base de données via une injection **SQL**, abrégé en **SQLi**.

L'injection **SQL** fait référence aux attaques contre les bases de données relationnelles telles que MySQL, Oracle Database ou Microsoft SQL Server. En revanche, les injections contre les bases de données non relationnelles, telles que MongoDB ou CouchDB, sont des injections NoSQL.

Dans notre projet, nous avons limité les injections **SQL** entre autre par l'utilisation de setParameter().

```
// La requête qui récupère les dates non disponibles utilise des paramètres nommés :date et :idCourse.
// Ces paramètres sont bindés à l'aide de la méthode setParameter(), ce qui prévient les injections SQL.
->setParameter(":date", $date)
->setParameter(":idCourse", $course->getId())
->getResult();
```

Lorsqu'une valeur est passée via setParameter(), elle n'est pas insérée directement dans la requête. À la place, la requête est envoyée avec des placeholders à la base de données, et les valeurs sont envoyées séparément. La base de données reçoit donc les valeurs séparément de la requête elle-même, ce qui empêche les injections **SQL**.

Pour être plus précis setParameter() va fonctionner de 3 façons :

1 Requêtes préparées :

- Lorsqu'une requête est crée avec des placeholders (comme :date ou :idCourse), On indique à la base de donnée qu'on attend une valeur.

- La requête est envoyée à la base de données qui la "prépare", elle analyse la syntaxe et détermine comment elle sera exécutée, sans connaître les valeurs exactes des placeholders.

2 On lit les valeurs :

- Quand on utilise setParameter(), on va "lier" une valeur spécifique à un placeholder. On indique à la base de données quelle valeur doit être utilisée pour quel placeholder.

- Les valeurs sont transmises séparément de la requête initiale à la base de données. Du coup, la base de données sait que ces valeurs ne doivent pas être exécutées comme du **SQL**, mais doivent plutôt être traitées comme des données.

3 L’exécution :

- Une fois que toutes les valeurs sont liées à leurs placeholders, la requête est exécutée par la base de données.

- Comme la base de données a déjà "préparé" la requête et sait comment elle sera exécutée, et que les valeurs sont transmises séparément, il n'y a aucun risque qu'une valeur entrée par un utilisateur soit exécutée comme du **SQL**. Cela prévient donc contres les injections **SQL**.

V. TRADUCTION

V Traduction

Comme traduction, j'ai décidé d'utiliser la partie de présentation de Stripe dans la documentation officielle, c'est avec cette documentation que j'ai commencé a intégrer Stripe a mon projet, comme moyen de payement.

"The Stripe API is organized around REST. Our API has predictable resource-oriented URLs, accepts form-encoded request bodies, returns JSON-encoded responses, and uses standard HTTP response codes, authentication, and verbs.
You can use the Stripe API in test mode, which doesn't affect your live data or interact with the banking networks. The API key you use to authenticate the request determines whether the request is live mode or test mode.
The Stripe API doesn't support bulk updates. You can work on only one object per request.
To optimize performance and reliability, Stripe has established rate limits and allocations for API endpoints.
The Stripe API differs for every account as we release new versions and tailor functionality. Log in to see docs customized to your version of the API, with your test key and data."

"The Stripe API uses API keys to authenticate requests. You can view and manage your API keys in the Stripe Dashboard.
Test mode secret keys have the prefix sk_test_ and live mode secret keys have the prefix sk_live_. Alternatively, you can use restricted API keys for granular permissions.
Your API keys carry many privileges, so be sure to keep them secure. Don't share your secret API keys in publicly accessible areas such as GitHub, client-side code, and so forth.
Authentication to the API is performed with HTTP Basic Auth. Provide your API key as the basic auth username value. You don't need to provide a password.
If you need to authenticate with bearer auth, for example, for a cross-origin request, use -H "Authorization: Bearer sk_test_4eC39HqLyjWDarjtT1zdp7dc" instead of -u sk_test_4eC39HqLyjWDarjtT1zdp7dc.
You must make all API calls over HTTPS. Calls that you make over plain HTTP will fail. API requests without authentication will also fail."

V. TRADUCTION

L'API de Stripe est organisé autour de la méthode REST. Notre API possède des URL prévisibles orientées ressources, accepte les corps de requête codés sous forme de formulaire, renvoie des réponses codées JSON et utilise des codes de réponse HTTP standard, mais aussi une authentification et des verbes.
Vous pouvez utiliser l'API Stripe en mode test, ce qui n'affecte pas vos données en direct ni n'interagit avec les réseaux bancaires. La clé API que vous utilisez pour authentifier la demande détermine si la requête est en mode direct ou en mode test.
L'API Stripe ne prend pas en charge les grosses mises à jour. Vous ne pouvez travailler que sur un seul objet par requête.
Pour optimiser les performances et la fiabilité, Stripe a établi des limites de débit et des allocations pour les endpoints de l'API.
L'API Stripe diffère pour chaque compte à mesure que nous publions de nouvelles versions et adaptons les fonctionnalités. Connectez-vous pour voir les documentations personnalisés selon votre version de l'API, avec votre clé de test et vos données.

L'API Stripe utilise des clés API pour authentifier les requêtes. Vous pouvez afficher et gérer vos clés API dans le tableau de bord Stripe.
Les clés secrètes du mode test ont le préfixe sk_test_ et les clés secrètes du mode live ont le préfixe sk_live_. Vous pouvez également utiliser des clés API restreintes pour des autorisations granulaires.
Vos clés API apportent de nombreux privilèges, alors assurez-vous de les conserver en sécurité. Ne partagez pas vos clés API secrètes dans des zones accessibles au public telles que GitHub, le code côté client, etc.
L'authentification auprès de l'API est effectuée avec HTTP Basic Auth. Fournissez votre clé API comme valeur de nom d'utilisateur d'authentification de base. Vous n'avez pas besoin de fournir un mot de passe.
Si vous devez vous authentifier avec l'authentification du porteur, par exemple pour une demande d'origine croisée, utilisez -H "Authorization: Bearer sk_test_4eC39HqLyjWDarjtT1zdp7dc" au lieu de -u sk_test_4eC39HqLyjWDarjtT1zdp7dc.
Vous devez effectuer tous les appels d'API via HTTPS. Les appels que vous effectuez via HTTP simple seront un échec. Les requêtes API sans authentification seront également un échec.

VI. AMÉLIORATIONS ET CONCLUSION

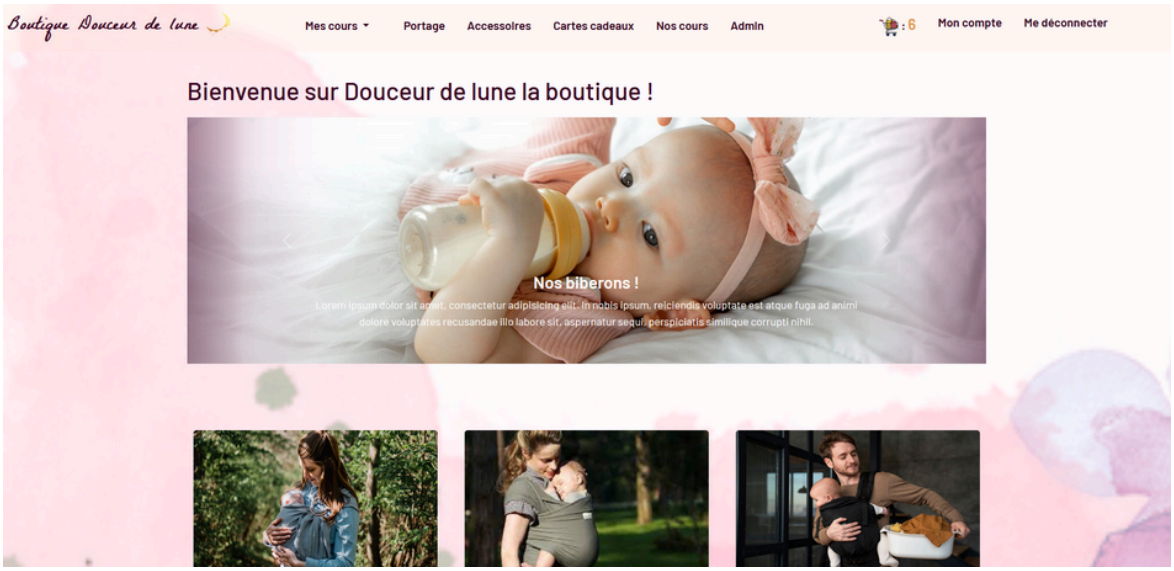
VI 1 Améliorations

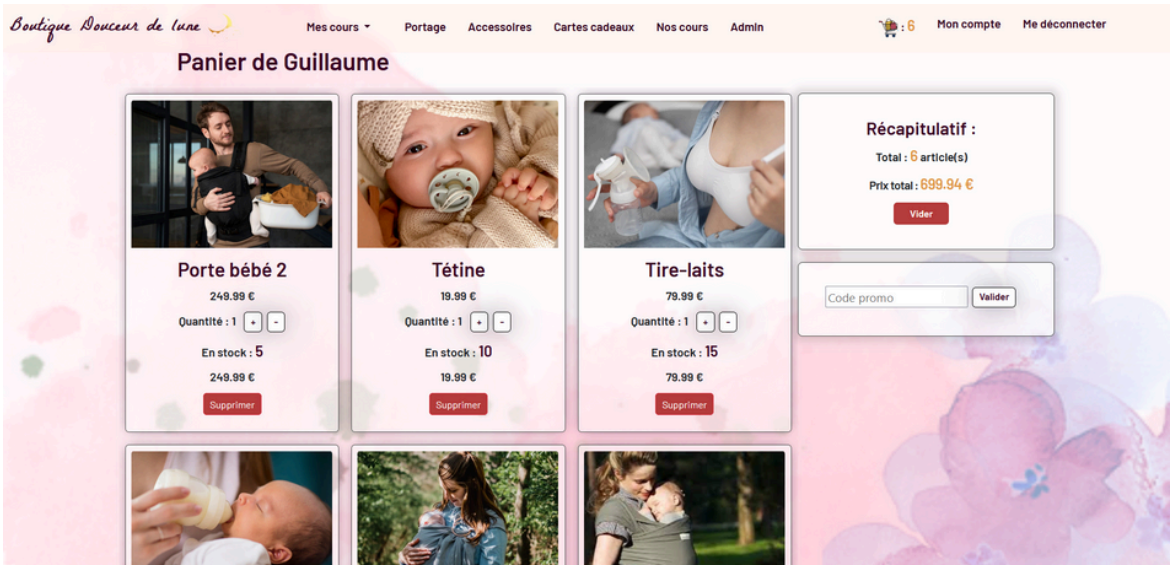
1 A Court terme

Comme premier acte d'amélioration à court terme, nous avons décidé, la cliente et moi de créer une boutique en ligne.

En effet, la cliente souhaite a terme, avoir la possibilité de vendre des écharpes de portage, portes-bébés, tétines, biberons etc.. créées par elle et un regroupement d'amis.

Ce side-project sera présenté rapidement durant la présentation du site, mais ne fait pas partie du projet principal concernant le titre DWWM. Il est cependant déjà très bien avancé et fonctionnel.





VI. CONCLUSION ET AMÉLIORATIONS

Le site utilisera cette fois-ci un mélange entre le framework **Bootstrap** et du CSS personnalisé, principalement afin de me familiariser avec ce framework, mais aussi pour le côté pratique et rapide qu'il apporte.
L'ajout de produit en panier, et le panier lui même sera géré en **AJAX**.

Il y a également un système de promotion pour le panier fonctionnel.

Comme autre amélioration a court terme, il y aura également la finalisation de mon intégration **Stripe**, le webhook n'est pas encore terminé mais déjà bien commencé.

1 B Moyen terme

Une section pour les "membres particuliers" (amis, réguliers, cas spécifiques) sera mise en place, avec un mot de passe pour accéder à la page.

Cette section proposera des liens et codes promo pour d'autres sites en rapport avec celui-ci (provenant d'amis et des collègues de la cliente)

Pour cela je compte modifier le security.yaml

```
password: '%env(APP_SECRET_PAGE_PASSWORD)%'
        roles: 'ROLE_USER'
```

Pour appliquer un mot de passe.

Utiliser php bin/console security:hash-password qui est un bundle pour hasher les mots de passe, que je mettrais dans le fichier .env dans une variable

```
# .env
APP_SECRET_PAGE_PASSWORD=LeMotDePasseHaché
```

1 C Long terme

Pour le long terme, la cliente aimerait dans le futur, intégrer une messagerie pour les utilisateurs, afin qu'ils puissent partager leur expérience. C'est encore en discussion avec la cliente, car elle souhaite le faire uniquement si les utilisateurs le demandent et sont suffisamment nombreux.

2 Conclusion

Le site développé est conforme au cahier des charges établi, tant du côté technique requis par l'examen DWWM, que du côté des exigences client. Celui-ci offre une fonctionnalité de réservation via un calendrier, permettant de sélectionner une date et une heure pour un cours. Cette réservation est enregistrée en base de donnée et offre à l'utilisateur la possibilité d'effectuer un paiement sécurisé via Stripe. De plus, le site propose une boutique où l'utilisateur peut découvrir des produits liés aux cours.

Du point de vue sécurité, le site est conçu pour contrer les vulnérabilités les plus courantes et bénéficie d'un CSP renforçant sa robustesse. Esthétiquement parlant, son design, bien qu'épuré, est en harmonie avec le thème, offrant ainsi une expérience utilisateur intuitive. Le client est d'ailleurs très satisfait du résultat.

Concernant ma formation, elle a été un défi majeur pour moi. Après une longue pause dans mes études et suite à un accident, retrouver un rythme d'apprentissage a été une épreuve.

Toutefois, je suis fier du chemin parcouru. N'ayant jamais codé auparavant, réaliser un tel projet en quelques mois était inimaginable pour moi au début.

Cette expérience m'a non seulement beaucoup appris, mais elle a aussi renforcé ma passion pour le domaine du développement. Je suis maintenant déterminé à poursuivre dans cette voie, envisageant notamment une formation en Conception et Design UI(niv 6)en alternance, chez ELAN ou ailleurs si nécessaire.

Je suis actuellement en discussion avec plusieurs agences en vue d'une alternance, ce qui me permettra de m'immerger davantage dans le monde professionnel.

Je vous remercie pour l'attention que vous avez portée à ce dossier de synthèse et espère qu'il répond à vos attentes.