ÉTABLISSEMENT SAINT-ADJUTOR

Rapport de stage – 2ème année BTS SIO SLAM

Date du stage : du 22 janvier au 1er mars 2024

Tuteur de stage : Nathalie Kesler

Étudiant : Medjeni Dhelil

Remerciement :

Tout d’abord, je tiens à remercier Nathalie KESLER, ma tutrice de stage, pour son accueil et son accompagnement, tout au long de mon stage au sein de l’entreprise PANGEE ONG.

Je la remercie pour toutes les nouvelles connaissances et les réponses à mes questions qu’elle a pu m’apporter. Cela m’a permis de réaliser au mieux le travail demandé et la réalisation de ce rapport de stage.

SOMMAIRE

1 Présentation de l'entreprise

1.1 Présentation

La société dans laquelle j'ai réalisé mon stage se nomme Pangee ONG. Il s'agit d'une ONG (Organisation Non Gouvernementale) œuvrant principalement dans le domaine de la paix à travers le monde. Dans cette ONG, plusieurs tâches sont réalisées liés à différents domaines tel que la vente qui est l'élément principal, mais aussi la comptabilité, l'événementiel et la réalisation de sites web.

PANGEE ONG est reconnue pour sa vente de différents produits tel que des tableaux et livres symbolisant et représentant la paix à travers le monde.

Dans le cadre de ses activités, Pangee ONG collabore avec divers acteurs, notamment des freelances, des bénévoles et des associés. Cette collaboration permet à l'organisation de mener à bien ses projets et initiatives.

Sous la direction de Nathalie KESLER, la dirigeante de l'entreprise, PANGEE ONG s'engage à offrir des solutions sur mesure répondant aux besoins spécifiques de ses clients. Que ce soit dans la vente de produits tel que des tableaux symbolisant la paix ou dans la création de sites web personnalisés, l'objectif principal de l'ONG est de répondre de manière efficace et adaptée aux demandes de sa clientèle.

Pour soutenir ses activités, PANGEE ONG travail avec des freelances notamment dans le domaine de la création de ses sites web. Ces collaborations reposent sur l'utilisation de différentes technologies, principalement des CMS tels que WordPress, Odoo, afin de répondre aux besoins variés de l'organisation et de ses clients.

1.2 Organigramme de l'ONG

L'entreprise Pangee ONG ne dispose pas d'un organigramme formel pour représenter sa structure organisationnelle. Cependant, cela ne nuit pas à la clarté de son fonctionnement, car les rôles et les responsabilités sont bien définis au sein de l'équipe dirigée par Nathalie Kesler.

2 Service d'accueil et moyens informatiques

2.1 Service d'accueil

L'organisation dispose d'un service d'accueil situé au 56 rue Mademoiselle à Paris, où j'ai effectué mon stage. Ce local est aménagé pour fournir un espace de travail fonctionnel et confortable aux stagiaires. Il comprend plusieurs bureaux équipés permettant aux stagiaires d'accomplir leurs tâches dans des conditions optimales. De plus, pour ceux qui n'ont pas leur propre équipement, plusieurs ordinateurs sont mis à disposition afin de faciliter leur travail et leur intégration au sein de l'organisation.

2.2 Moyens informatiques utilisés

Dans le contexte de cette ONG, les moyens informatiques jouent un rôle essentiel pour assurer la communication, la collaboration et le bon fonctionnement des opérations. Les stagiaires ont la possibilité d'utiliser leurs propres ordinateurs personnels pour effectuer leurs tâches professionnelles. Cependant l'ONG met également à disposition des PC fixes et portables dans le service d'accueil. Parmi les équipements disponibles, on trouve un MacBook Pro, un laptop Acer Gamer, deux postes avec unité centrale, ainsi qu'une tablette pouvant être utilisée comme un laptop. De plus, à l'étage, des postes avec unité centrale sont également accessibles.

Les ordinateurs, qu'ils soient personnels ou fournis par l'ONG, doivent répondre à certaines exigences minimales en termes de configuration matérielle et de logiciels. Par exemple, les stagiaires en développement utilisent des outils comme Visual Studio Code, tandis que ceux en comptabilité ont recours à des PGI spécialisés. Pour les autres stagiaires, des logiciels de bureautique tels que la suite Microsoft Office sont utilisés, notamment pour les activités liées à la vente.

Un Cloud est également à disposition pour tous les stagiaires, bénévoles ou associés, permettant de retrouver différentes informations tel que les différents mots de passes pour le WIFI, les adresses électroniques professionnelles…

2.3 Réseau informatique

L'ONG dispose d'un réseau informatique interne sécurisé pour faciliter la communication et la collaboration entre les stagiaires et les collaborateurs, qu'ils travaillent à distance ou dans les locaux de l'ONG. Ce réseau est protégé par des mesures de sécurité avancées telles que des pares-feux et un cryptage des données, garantissant ainsi la confidentialité et l'intégrité des informations échangée.

En plus du réseau interne, l'organisation utilise une gamme d'outils de communication en ligne intégrés, tels que WhatsApp et Google Meet, qui sont accessibles via le réseau sécurisé de l'ONG. Ces outils permettent une collaboration efficace et une communication fluide entre les équipes travaillant à distance et en présentiel. De plus, des espaces de travail collaboratifs sont mis à disposition pour faciliter le partage de documents et la coordination des projets entre les membres de l'organisation.

Ces initiatives visent à optimiser la productivité et à renforcer la cohésion au sein de l'ONG, en permettant à ses membres de travailler de manière transparente et efficace, quel que soit leur lieu de travail.

3 Présentation du projet

3.1 Présentation des exercices réalisés

3.1.1 Description du sujet de façon succincte

Dans le cadre de mon stage au sein de l'ONG PANGEE, j'ai eu l'opportunité de travailler sur trois projets distincts, dont deux en autonomie et un en groupe. Chaque projet était conçu pour me permettre de développer mes compétences techniques et de pratiquer sur de nouveaux concepts, tout en approfondissant mes connaissances existantes.

Le premier projet consistait à créer un site avec le framework Symfony, nommé KalaPaints. Mon objectif était de concevoir et d'implémenter la maquette du site en respectant les directives données. Pour cela, j'ai utilisé Figma pour créer deux versions de la maquette, une pour les ordinateurs de bureau et une pour les appareils mobiles/tablettes. Ce projet m'a permis de pratiquer sur des langages de programmation tels que HTML, CSS, PHP, et JavaScript, ainsi que de me familiariser davantage avec le fonctionnement de Symfony.

Dans le cadre du deuxième exercice, j'ai été chargé de compléter une maquette déjà existante et d'implémenter les pages manquantes en utilisant WordPress. Cette tâche m'a permis de renforcer mes compétences en développement WordPress et de comprendre les principes de conception et d'intégration de ce CMS.

Enfin, en ce qui concerne le dernier exercice, j'ai eu l'opportunité de réaliser un autre site en utilisant le framework React, en suivant une formation, ce qui m'a permis de découvrir et de

me former à cette technologie. Bien que ce fut un défi, j'ai réussi à créer un site fonctionnel en utilisant React, ce qui m'a donné une compréhension plus approfondie de ce framework.

Le design et les pages à créer lors de la formation React ont été réalisées de telle sorte à réaliser un site pas trop complexe, me permettant ainsi d'appréhender le fonctionnement du framework.

Ces projets m'ont non seulement permis de développer mes compétences techniques, mais ont renforcés mon envie d'apprendre et de pratiquer par moi-même, à résoudre des problèmes de manière créative et à m'adapter à de nouvelles technologies et environnements de travail.

3.1.2 Quelles sont les motivations pour la réalisation de ce travail ?

En ce qui concerne les motivations pour la réalisation des différents exercices, il y en a plusieurs :

La motivation personnelle :

J'ai toujours aimé faire du développement web, c'est donc tout naturellement que je me suis orienté vers un stage de ce type et donc vers ce type d'exercices.

Apprentissage et développement des compétences :

En effectuant ce stage, j'ai eu l'occasion de pratiquer sur les compétences que j'ai appris en cours pour travailler de manière autonome et efficace. Cela inclut la maitrise des outils de communication en ligne, la gestion du temps et l'organisation personnelle, ainsi que la capacité à résoudre les problèmes techniques éventuels rencontrés lors du travail.

Ce stage m'a également permis de développer de nouvelles compétences telles que l'apprentissage du framevork React.

Expérience préparatoire pour le monde professionnel actuel :

Ce stage m'a offert une préparation précieuse pour mon avenir professionnel. Cette expérience m'a permis de développer mon autonomie et ma capacité à travailler de manière indépendante et en groupe, ce qui est essentiel dans un monde du travail où l'on peut être amené à gérer des projets tant en équipe que solitaire. En prenant en charge des projets de bout en bout, j'ai acquis une compréhension approfondie des processus de développement et de gestion de projet, ce qui me sera certainement bénéfique dans ma future carrière.

De plus, j'ai pu mettre en pratique mes compétences de communication et de résolution de problèmes, ce qui m'a permis de surmonter les défis rencontrés et d'atteindre mes objectifs de manière efficace.

3.1.3 Quel est le demandeur ? En quoi mon travail va lui être utile ?

Le demandeur :

Le demandeur de mon travail était l'ONG elle-même, représentée par ma tutrice de stage. Bien que mes projets réalisés seul n'aient pas directement contribué aux projets spécifiques de l'entreprise, ils étaient conçus pour me permettre de pratiquer et de développer mes compétences en développement web. Mon travail s'inscrivait parfaitement dans le cadre de ma formation, me donnant l'opportunité de mettre en pratique les connaissances acquises en cours et d'acquérir de nouvelles compétences dans un environnement professionnel.

En accueillant des stagiaires comme moi, l'entreprise démontre son engagement envers la formation et le développement de futurs professionnels. Cette approche témoigne de la volonté de l'entreprise d'investir dans ses équipes et de répondre aux défis croissant du secteur de l'informatique. En fournissant des opportunités d'apprentissage pratique, l'ONG favorise le développement de compétences essentielles chez les stagiaires, renforçant ainsi sa propre expertise et sa capacité à innover dans son domaine d'activité.

Utilité de mon travail :

Bien que mon travail en tant que stagiaire fut principalement axé sur le développement de mes compétences existantes, il a également présenté des avantages potentiels pour l'ONG. A travers les exercices pratiques que j'ai réalisés, j'ai pu renforcer et acquérir de nouvelles compétences dans plusieurs domaines, tels que la programmation, la conception web et la gestion de projet.

En observant mes réalisations et mes progrès au cours du stage, l'entreprise a pu évaluer ce qu'un stagiaire de deuxième année de BTS SIO est capable de faire, même si cela reste subjectif et peut varier en fonction des individus.

3.2 Explication détaillée de chaque exercice

3.2.1 Présentation des outils utilisés

Dans le cadre du développement de mon travail lors de mon stage, j'ai utilisé plusieurs outils qui sont également utilisés par d'autres stagiaires ou freelances, afin de mener à bien les exercices qui m'ont été attribués. Ces outils ont été essentiels pour la communication, la gestion du temps, la programmation, l'apprentissage, ainsi que la répartition des différentes tâches.

Outils liés à la communication/accessibilité :

Tout d'abord, pour pouvoir communiquer avec ma tutrice ou bien d'autres stagiaires/freelance, j'ai utilisé WhatsApp de Facebook qui est une application de messagerie instantanée largement utilisée dans le monde professionnel pour sa facilité d'utilisation et sa disponibilité sur plusieurs plateformes. Grâce à WhatsApp, j'ai pu échanger rapidement sur différents points.

En ce qui concerne l'accessibilité, j'ai utilisé Google Drive qui est une plateforme de stockage en ligne et de partage de fichiers très répandue. Grâce à cet outil, j'ai pu accéder à différents documents afin d'avoir plusieurs informations tel que le mot de passe du Wifi ou bien encore les différents mots de passes nécessaires à l'utilisation de WordPress.

Environnement de travail/ Planification des tâches :

En ce qui concerne l'environnement de travail, j'ai utilisé le même IDE sur lequel je suis apte à travailler, Visual Studio Code qui est un éditeur de code léger, performant et largement utilisé par les développeurs.

Cet outil offre une interface conviviale, une multitude d'extensions personnalisables et des fonctionnalités avancées telles que la coloration syntaxique, l'auto-complétion, le débogage en temps réel et l'intégration avec des outils de gestion de versions comme Git.

Son adaptabilité en fait un choix populaire pour la programmation dans divers langages, ce qui en fait un outil idéal pour le développement de projets notamment durant mon stage.

Cependant, je reconnais l'importance des outils de gestion de versions dans un environnement de travail collaboratif, tel que Git. En effet, ces outils permettent de suivre les modifications apportées au code, de faciliter la collaboration entre plusieurs développeurs, et de gérer les différentes versions du projet de manière organisée.

Durant ce stage, j'ai décidé de ne pas en utiliser ce type d'outil, en effet, les deux projets que j'ai réalisés en autonomie n'en n'ont pas nécessité l'usage et, je trouve que l'utilisation de ces outils est plus utile dès lors que l'on travail en groupe sur des projets plus concrets qui impliquent plusieurs développeurs.

En ce qui concerne la planification des tâches, j'ai utilisé deux outils mais plus principalement un seul, Trello, qui est un outil de gestion de projet en ligne basé sur des cartes. Trello permet de créer des tableaux pour organiser les tâches, des listes pour suivre leur progression, et des cartes pour détailler les étapes à accomplir.

Avec sa facilité d'utilisation et sa flexibilité, Trello m'a permis de visualiser facilement l'état d'avancement de mes projets et de prioriser les tâches en fonction de leur niveau d'importances. Grâce à ses fonctionnalités très développées telles que les commentaires, j'ai pu établir une todo-list pour chacune des cartes, en commentant l'avancement de ces dernières, facilitant ainsi la coordination des projets de manière efficace.

Apprentissage/Formation

En ce qui concerne les outils utilisés à l'apprentissage de React, j'ai utilisé principalement YouTube qui est une plateforme de partage de vidéos en ligne. Cet outil regorge de tutoriels, de cours et de vidéos explicatives sur de nombreux domaines notamment React, ce qui en fait une ressource précieuse pour l'apprentissage de ce framework JavaScript.

J'ai également consulté la documentation officielle des différentes technologies que j'ai utilisées, que l'on nous allons voir dans le paragraphe suivant, afin de comprendre mieux leur fonctionnement.

Ensuite, j'ai également cherché sur Internet lorsque je rencontrais des erreurs, notamment sur des forums ou des personnes avaient rencontrées les mêmes erreurs que les miennes. Dans les cas où je ne trouvais pas de solution en ligne, j'ai également eu recours à ChatGPT, une intelligence artificielle avec laquelle je pouvais discuter pour obtenir des informations sur différents éléments.

Technologies/Langages

En ce qui concerne les technologies utilisées, j'ai utilisé Symfony qui est un framework PHP open-source et largement utilisé pour le développement d'applications web. Symfony offre une architecture solide, modulaire et extensible, ce qui facilite la création de projets web complexes et évolutifs. Ce framework est très utile en ce qui concerne le back-end d'une application, en effet il possède de nombreuses fonctionnalités robustes et évolutives qui permettent de gérer ce côté sans soucis. Ce framework fonctionne sous l'architecture MVC, correspondant aux modèles, vues, contrôleurs, offrant ainsi une base solide pour toute application web nécessitant une gestion sophistiquée du côté serveur.

J'ai également utilisé React, qui est un framework JavaScript très puissant et largement utilisé dans l'industrie du développement web. Mon apprentissage de ce framework m'a permis de créer des interfaces utilisateur dynamiques et réactives, tout en suivant les meilleures pratiques de développement. Grâce à son approche basée sur les composants, j'ai pu organiser efficacement le code et le rendre plus modulaire, facilitant ainsi la maintenance et l'évolutivité du projet. En combinant React avec d'autres technologies, que je détallerais ci-dessous, j'ai pu créer une application web moderne et performante, offrant une expérience utilisateur optimale.

J'ai aussi utilisé Tailwind CSS, une bibliothèque de styles CSS, que j'ai apprise en même temps que React. Cette bibliothèque offre une approche innovante pour la création d'interfaces utilisateur. Avec cette bibliothèque, j'ai pu accélérer le processus de développement en utilisant des classes prédéfinies pour styler les éléments HTML, ce qui m'a permis de créer des designs cohérents et esthétiques, respectant ainsi à la lettre la maquette.

De plus, j'ai utilisé Firebase qui est une plateforme de développement d'applications mobiles et web proposée par Google, qui fournit une infrastructure complète pour construire des applications de haute qualité rapidement. Avec Firebase, j'ai pu explorer divers aspects du développement, notamment la gestion de l'authentification des utilisateurs, la base de données en temps réel, le stockage des fichiers tel que des images, ainsi que des notifications push. Cette plateforme m'a permis de créer mon application, de manière simple, interactive et évolutive.

J'ai aussi utilisé Next.js, un framework JavaScript open-source permettant de construire des applications web React de manière efficace et performante. Next.js offre une approche basée sur les composants et une gestion simple du routage, ce qui facilite la création d'interfaces utilisateur dynamiques et réactives. Grâce à ses fonctionnalités avancées telles que le rendu côté serveur, le pré-rendu statique, ainsi que le support intégré pour les API, ce framework permet de développer des applications web évolutives et optimisées pour les performances.

En ce qui concerne le design des deux projets, j'ai utilisé Figma qui est un outil de conception et de prototypage d'interfaces utilisateur (UI) très complet et intuitif. Figma offre une interface utilisateur conviviale qui permet de créer facilement des maquettes, des wireframes et des prototypes interactifs. Grâce à ses fonctionnalités collaboratives en temps réel, il permet de travailler avec ses collègues de manière à pouvoir itérer rapidement sur les designs, recueillir des retours et apporter des modifications en conséquence. En résumé, en utilisant cet outil, j'ai pu concrétiser mes idées de manière visuelle et interactive, ce qui a grandement contribué à la réussite des projets.

Au sein de l'application faite en React, j'ai utilisé de nombreuses bibliothèques, telles que React Hook Form, React Router et d'autres encore. React Hook Form a été essentiel pour simplifier la gestion des formulaires, tandis que React Router a permis de gérer la navigation et les routes de manière fluide au sein de l'application. Ces bibliothèques, parmi d'autres ont joué un rôle crucial dans le développement d'une expérience utilisateur fluide et performante.

L'application a été réalisé en TypeScript, un langage de programmation open-source développé par Microsoft qui ajoute des fonctionnalités de typage statique en option à JavaScript. L'utilisation de TypeScript m'a permis de détecter et de corriger les erreurs de manière proactive lors de la phase de développement, ce qui a conduit à un code plus fiable et plus facile à maintenir. En plus d'améliorer la qualité du code, TypeScript offre une meilleure expérience de développement grâce à son système de type robuste, permettant une meilleure compréhension du code et une documentation plus précise.

3.2.2 Présentation des méthodes utilisées

En ce qui concerne les méthodes utilisées, j'en ai utilisé plusieurs afin d'accomplir les différentes tâches que je devais réaliser. Comme énoncé dans la partie ci-dessus, j'ai utilisé plusieurs outils afin de faciliter la planification, l'organisation et l'exécution des tâches. Voici les différentes méthodes que j'ai utilisé :

Méthode de recherche documentaire :

En fonction des problèmes rencontrés, ou tout simplement afin de savoir comment procéder pour utiliser tel ou tel chose, j'ai effectué des recherches documentaires afin de comprendre ce que je devais faire. J'ai donc consulté des ressources en lignes, telles que les documentations officielles pour les langages, ou des plateformes de vidéos, comme YouTube afin de me former sur les technologies utilisées lors de la réalisation du site en React.

Méthode de planification et d'organisation :

Pour la réalisation des deux projets en autonomie, j'ai adopté une approche méthodique en établissant un plan détaillé. J'ai procédé par « SPRINT » qui sont des itérations de développement de courte durée, généralement d'un ou plusieurs jours, durant lesquelles je me suis concentré sur des objectifs spécifiques et atteignables. Pour chacun des sprints, j'ai défini les tâches à accomplir, les ressources nécessaires et les délais à respecter. Cette approche m'a permis de structurer efficacement mon travail, de rester concentré sur les objectifs à atteindre et de maximiser ma productivité.

Chacun de ces sprints a été défini sur Trello, en effet, j'ai mis en place une carte pour chaque sprint, dans laquelle j'ai énuméré sous forme de todo-list les étapes à réaliser.

Méthode de développement itératif :

Pour la réalisation de chacune des tâches, j'ai opté pour une approché itérative étroitement liée aux sprints. Cela signifie que j'ai décomposé chaque tâche en étapes plus petites et réalisables, alignées avec les objectifs spécifiques définis pour chaque sprint. En décomposant les tâches de cette manière, j'ai pu progresser de manière incrémentielle tout en gardant une vision claire de l'avancement du projet à chaque itération. A chaque étape, j'ai pu évaluer les résultats obtenus, réajuster si nécessaire et avancer vers la réalisation des objectifs finaux du sprint.

Méthode de collaboration et de communication :

Pour chaque Sprint terminé, j'informais ma tutrice afin qu'elle soit au courant de l'avancement de mes projets. Je lui présentais les résultats obtenus, les défis rencontrés et les solutions envisagées. Cette communication régulière permettait de maintenir un suivi transparent et efficace, favorisant ainsi une collaboration étroite et constructive tout au long du processus de développement. De plus, cela nous permettait de discuter des prochaines étapes et des ajustements éventuels à apporter pour atteindre les objectifs fixés dans les délais impartis.

3.2.3 Planification des tâches confiées

La planification des tâches fut une étape importante pour assurer une réalisation efficace et dans les délais impartis. Pour cela, j'ai utilisé les outils de planification dont j'étais familier et comme évoqué ci-dessus, Trello.

Lors de la rédaction de chaque tâche, j'ai commencé par analyser en détail les exigences et les spécifications de chacune d'entre elles. J'ai identifié les étapes nécessaires à la réalisation des deux projets, en prenant en compte leur complexité et leur priorité. A partir de cette analyse, j'ai établi un plan détaillé, en définissant les différentes étapes à suivre et les délais à respecter.

La priorisation des tâches a également joué un rôle essentiel dans la planification. En tenant compte des échéances et des objectifs spécifiques, j'ai déterminé l'ordre dans lequel les tâches devaient être abordées. Cette approche m'a permis de travailler de manière structurée, en me concentrant d'abord sur les tâches les plus urgentes ou cruciales pour chacun des deux projets.

Pour suivre l'avancement de mes travaux, j'ai utilisé les fonctionnalités de suivi et de gestion de projet disponibles dans les outils de planification de Trello. Cela m'a permis de visualiser clairement l'état d'avancement de chaque tâche, de mettre à jour régulièrement les informations relatives aux délais et de signaler tout problème ou retard éventuel. Cette transparence dans la planification a facilité la communication avec mon tuteur et a permis d'ajuster les ressources et les priorités si nécessaire.

En tenant compte de la complexité et des contraintes de chaque tâche, j'ai estimé approximativement le temps requis pour leur réalisation. Cela m'a aidé à gérer efficacement mon emploi du temps et à respecter les échéances fixées. Bien entendu, j'ai toujours été conscient que les estimations pouvaient être sujettes à des ajustements en fonction de l'évolution du travail et des éventuels problèmes rencontrés en cours de route.

La planification des tâches confiées a été un processus itératif et flexible. J'ai régulièrement revu mon plan en fonction de l'évolution des exercices et des priorités de chaque projet. Cette approche m'a permis de m'adapter aux changements et de gérer efficacement les problèmes rencontrés.

3.2.4 Explications techniques des tâches confiées avec impressions écran commentées

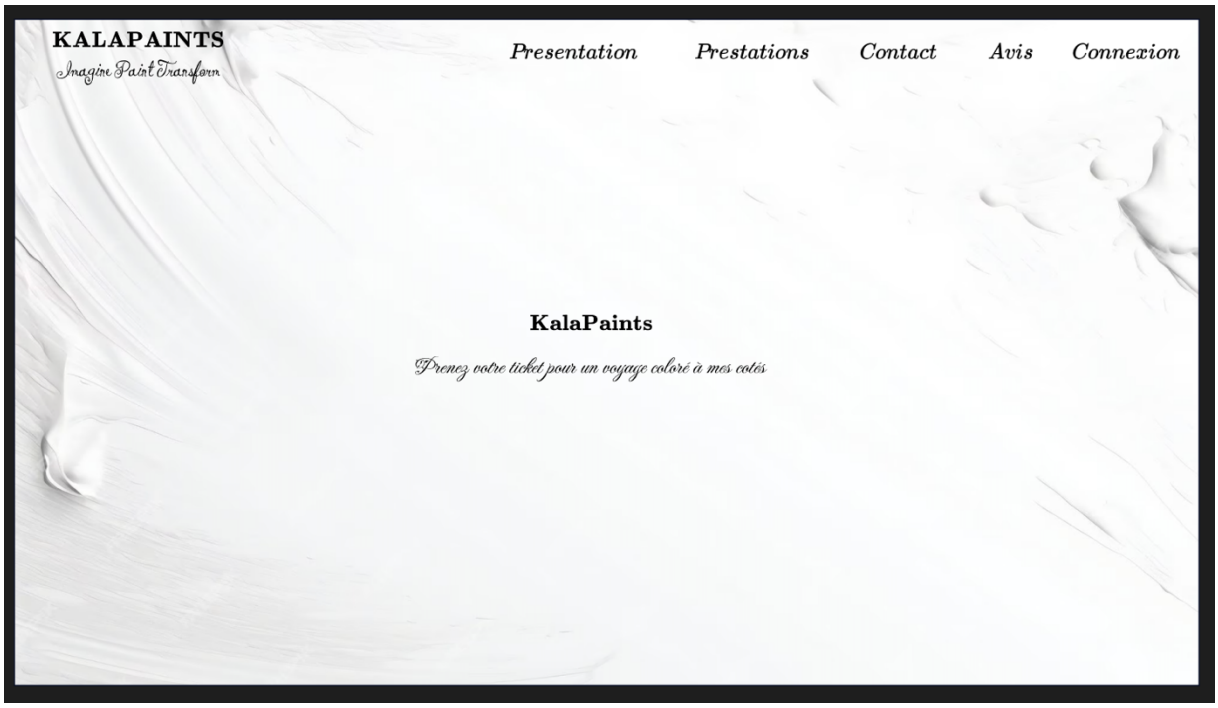
3.2.4.1 Premier projet : Création d'un site avec Symfony (from scratch)

Technologies utilisées lors de ce projet : Symphony, HTML, CSS, JS

3.2.4.1.1 Présentation et explication des maquettes

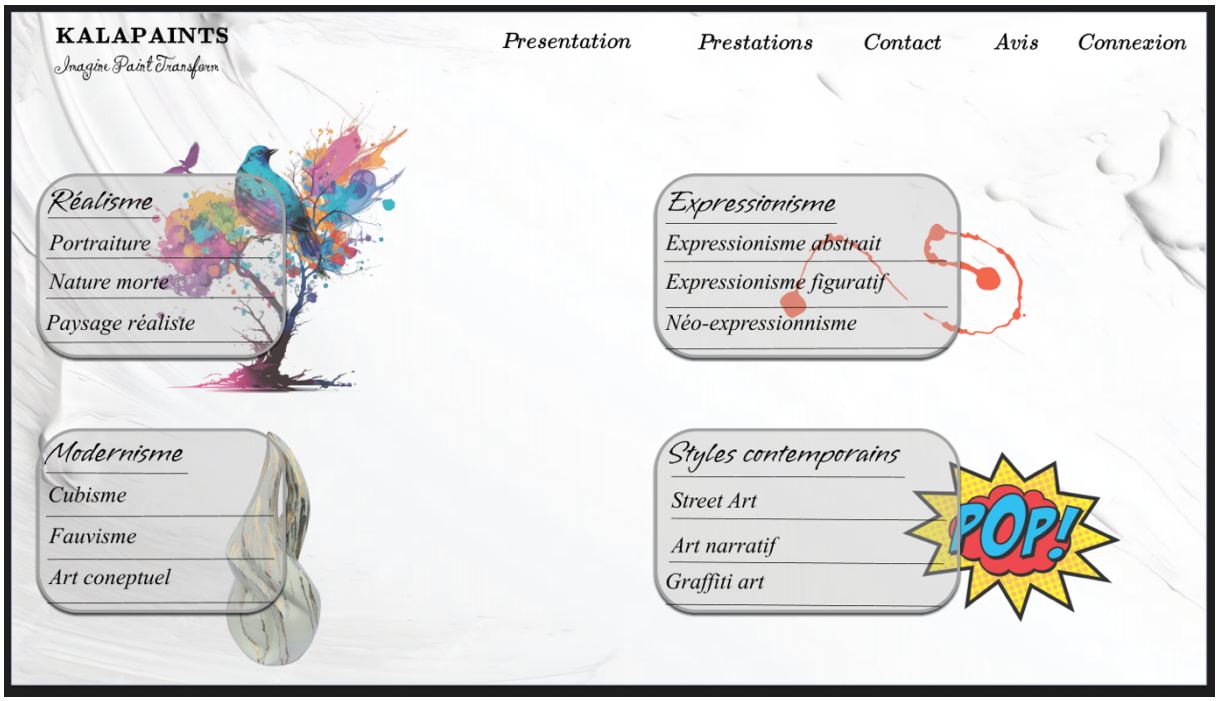
Voici les maquettes que j'ai réalisé :

Page d'accueil :



Page Présentation :

Page Prestations :

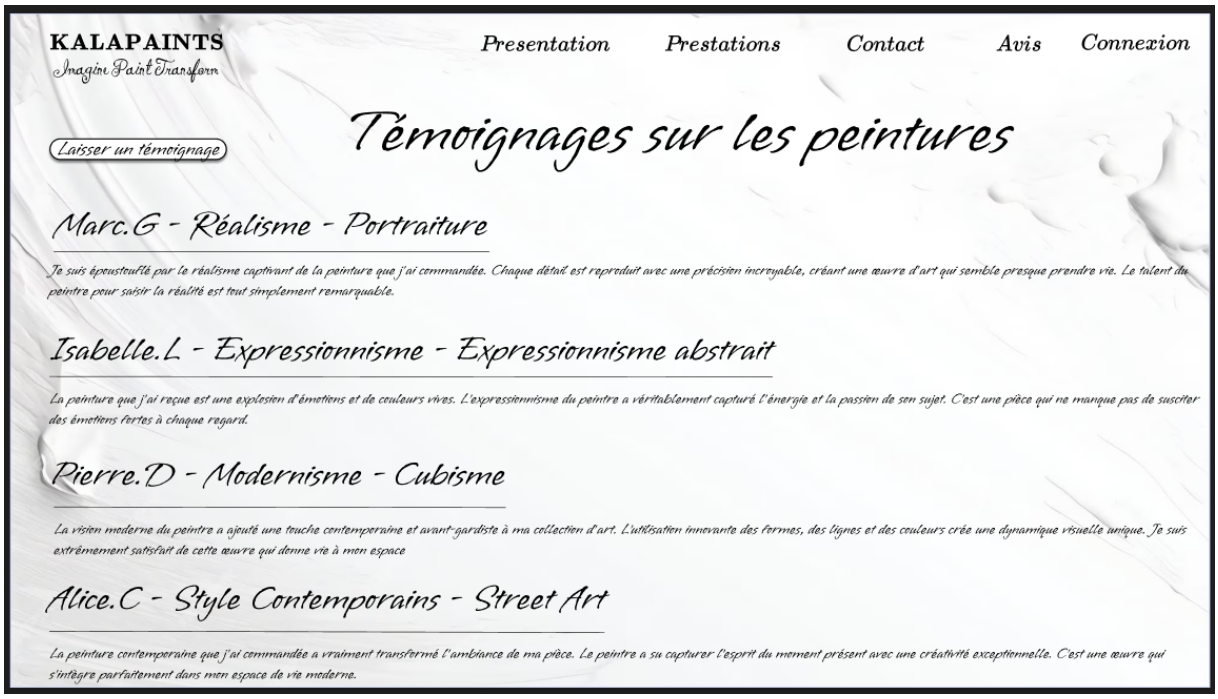


Page connexion :

Page d'inscription :

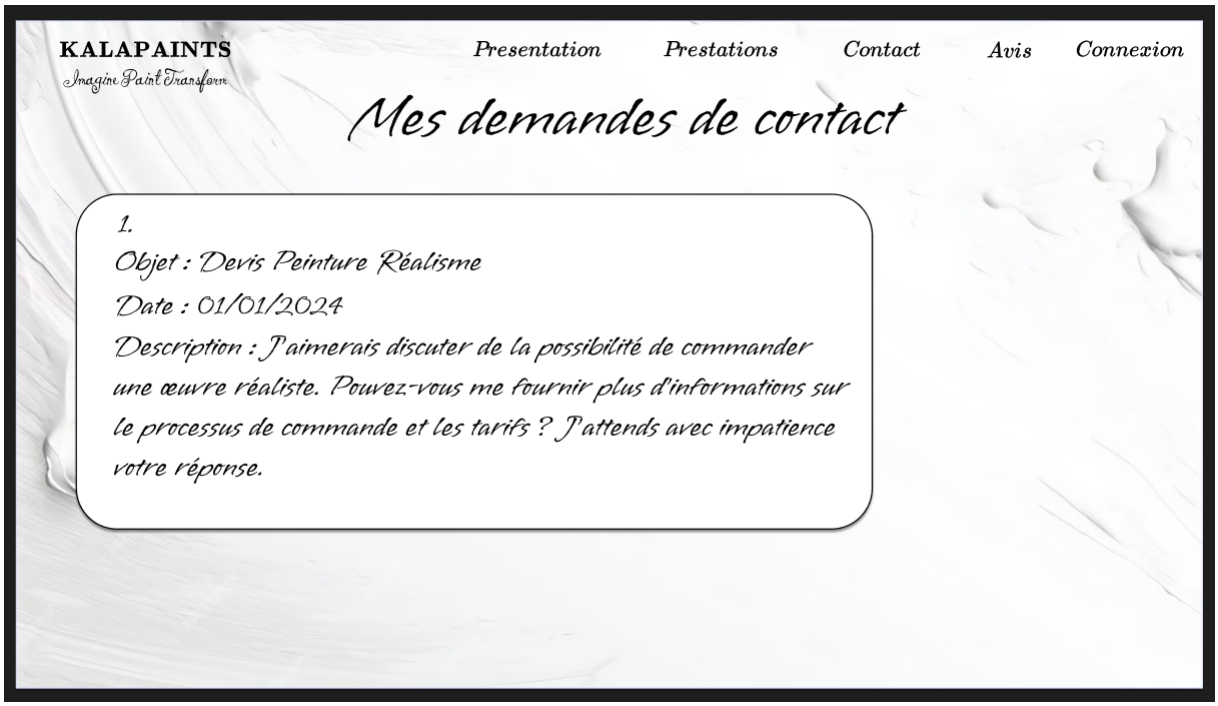
Page Contact :

Page Témoignages :

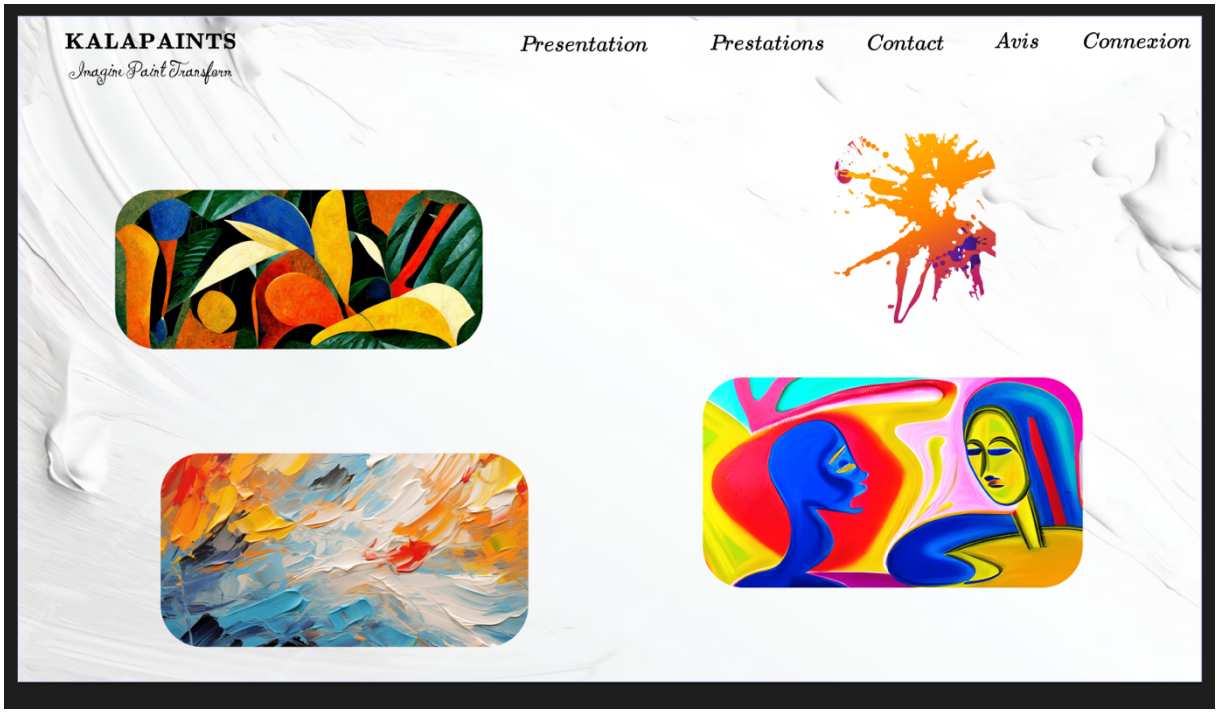


Page Laisser Témoignage :

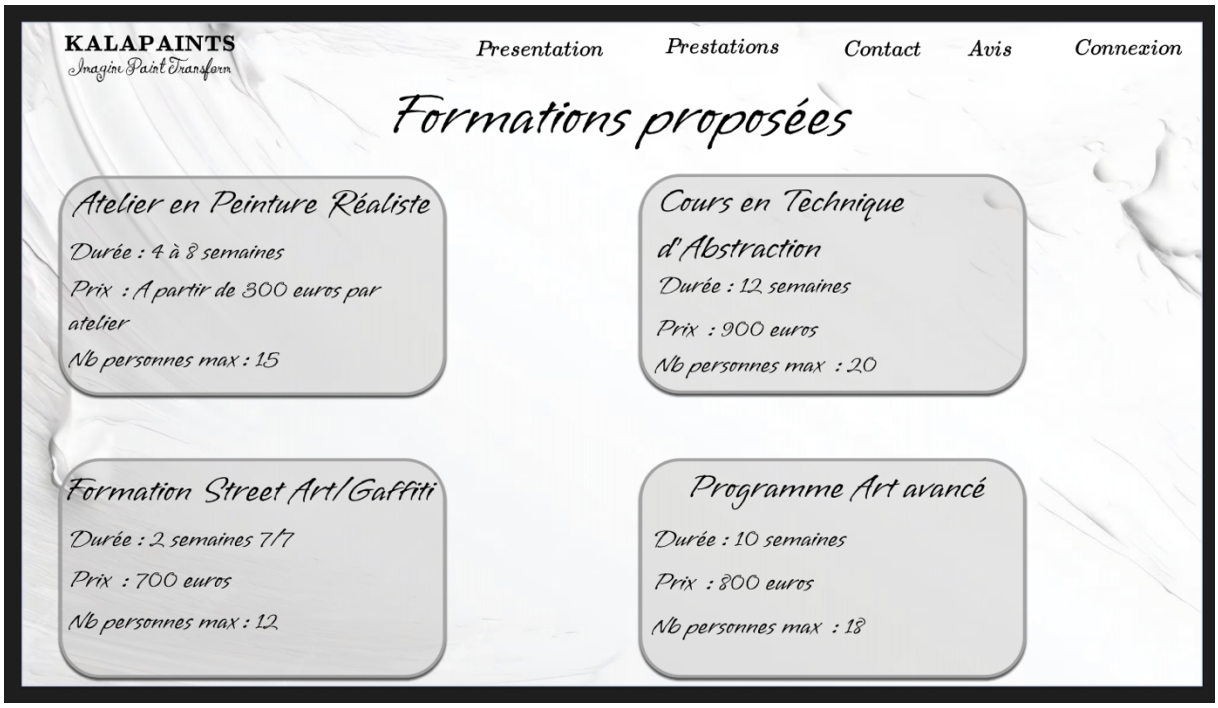
Page Demande de contact :



Page galerie :



Page formations proposées :



Mon objectif était donc de reproduire le site en respectant ce design, que j'avais créé from scratch.

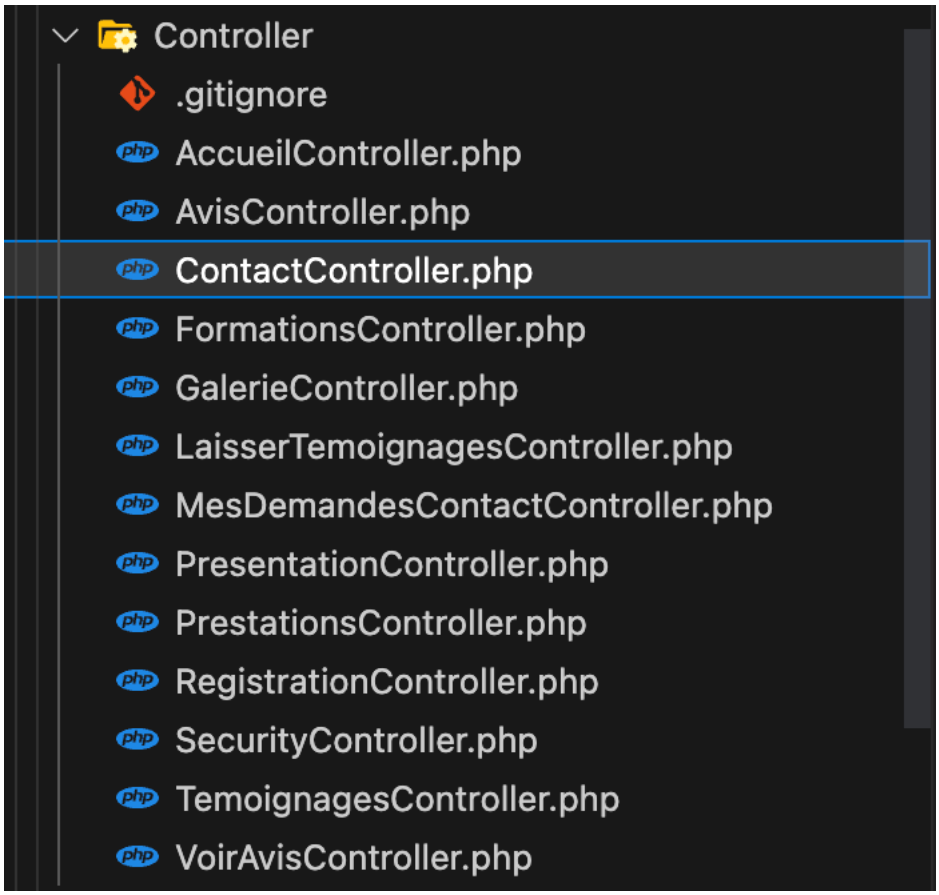
C'était un exercice que j'ai aimé réaliser, d'un point de vue UI. En effet, cela m'a permis de mettre en pratique mes compétences sur Figma, afin d'avoir une base solide avant de commencer à coder.

Concernant la difficulté de la réalisation des maquettes, ce n'était pas quelque chose de difficile à faire. En effet, Figma est un outil que j'avais déjà utilisé auparavant, que ce soit en cours ou à des fins personnelles. Les difficultés que j'ai rencontrées lors ce projet ont été davantage liées au développement, notamment avec des aspects que je ne savais pas ou que j'avais oubliés, que nous aborderons ci-après.

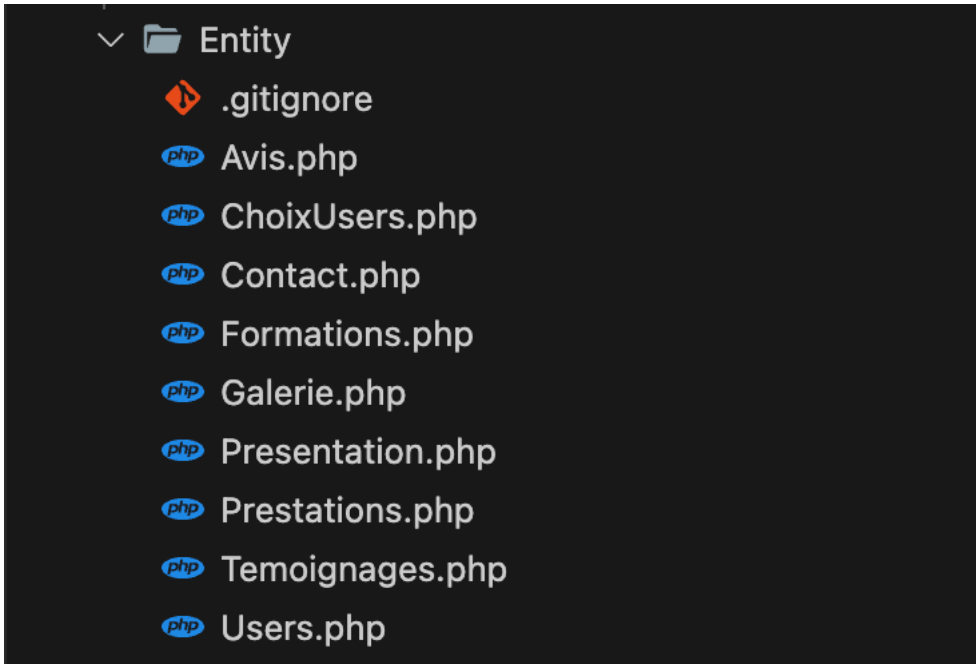
3.2.4.1.2 Explication du code de façon concrète (avec screen)

En ce qui concerne la structure du code, pour rappel en utilisant ce framework (Symfony), les pages sont donc réalisées en MVC (modèle, vue, contrôleur). J'ai donc initié le projet en utilisant plusieurs commandes Symfony, car ce framework propose une variété d'outils pour faciliter le développement et la gestion du projet.

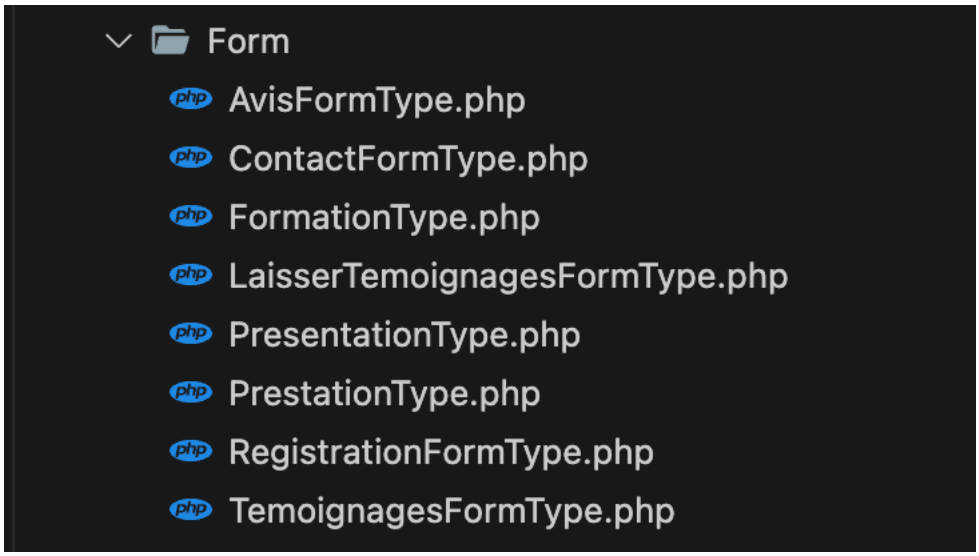
Par exemple, j'ai utilisé la commande 'symfony new' pour créer un nouveau projet Symfony, puis j'ai utilisé 'make : controller' pour générer des contrôleurs, dont voici le dossier les contenant :



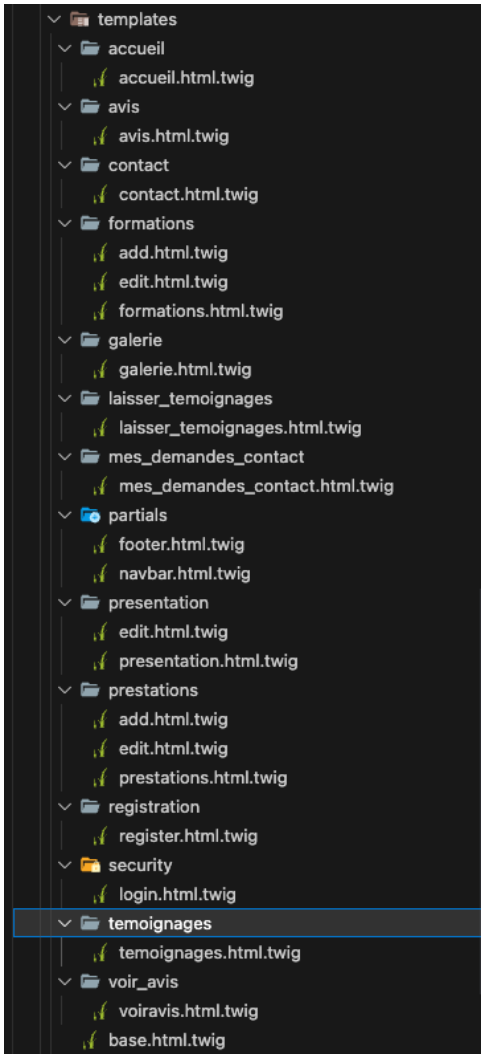
La commande ‘make : entity’ permettant de créer des entités de base de données, dont voici le dossier :



La commande ‘make : form’, qui permet de créer des formulaires nécessaires à l’application, dont voici le dossier :



Il y a également la commande ‘make : twig’ permettant de créer des vues (twig), dont voici le dossier :



Il y a aussi d’autres commandes qui ont été utilisées dans ce projet afin de le mener à bien. Par exemple, pour gérer les modifications de la structure de la base de données, j’ai utilisé la

commande ‘make : migration’ pour créer de nouvelles migrations. Une fois les migrations créées, j’ai confirmé les changements en base de données en utilisant la commande ‘doctrine : migrations : migrate’ abrégée en ‘d :m :m’. Ces migrations sont essentielles pour maintenir la cohérence de la base de données avec l’évolution de l’application.

En plus de ces commandes, j’ai également utilisé d’autres outils fournis par Symfony, afin de mener à bien le projet.

A présent, je vais vous présenter en détail les fonctionnalités de connexion et d’inscription que j’ai développées dans le cadre de mon stage. Ces éléments sont essentiels pour permettre aux utilisateurs d’accéder à la plateforme, notamment avoir accès aux différents formulaires, et de créer un compte personnel. Je vais expliquer comment ces fonctionnalités ont été conçues, en détaillant les différentes composantes telles que les vues, les contrôleurs, les formulaires et les styles utilisés pour respecter la maquette.

L’inscription commence par la saisie des informations nécessaires dans le formulaire dédié, telles que le nom, l’adresse e-mail, la ville, le numéro de téléphone.

Voici un extrait du formulaire, contenant toutes les informations à remplir lors de la création d’un compte :

```php
class RegistrationFormType extends AbstractType
{
    public function buildForm(FormBuilderInterface $builder, array $options): void
    {
        $builder
        ->add('email', EmailType::class, [
            'attr' => [
                'class' => 'form-control',
                'placeholder' => 'E-Mail',
                'error_bubbling' => true, // Désactive l'affichage automatique des erreurs

            ],
            'constraints' => [
                //new RegexMailConstraint(),
                // Regex du mail
                //new Regex('^[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.(fr|com|net)$/',
                //"Veuillez entrer une adresse e-mail valide. L'adresse e-mail doit suivre le format standard, comprenant une
            ],
            'label' => false
        ])

        ->add('plainPassword', RepeatedType::class, [
            'type' => PasswordType::class,
            'options' => [ 'attr' => ['class' => 'password-field']],
            'required' => true,
            'error_bubbling' => false, // Désactive l'affichage automatique des erreurs
            'first_options' => ['label' => false,
                                'attr' => ['placeholder' => '**********',
                                           'class' => 'register-password-input',
                                           'autocomplete' => 'new-password'],
                                'label_attr' => ['class' => 'register-label'],
                                'error_bubbling' => true,
                                ],

            'second_options' => ['label' => false,
                                 'attr' => ['placeholder' => "**********"]],
            'constraints' => [
                //new RegexPasswordConstraint(),
            ],
            'mapped' => false,
        ])
```

Voici l'explication de cette page :

Ce code utilise la classe 'RegistrationFormType', qui étend 'AbstractType', pour définir la structure du formulaire d'inscription de l'utilisateur.

A l'aide du builder ('FormBuilderInterface'), nous configurons les différents champs du formulaire.

Le champ 'email' :

- Nous utilisons la méthode 'add' pour ajouter le champ 'email' au formulaire, en spécifiant le type de champ ('EmailType : : class'). Cela garantit que l'utilisateur doit entrer une adresse email valide.
- Ensuite, nous définissons également des attributs HTML supplémentaires pour ce champ, tels que la classe CSS et le placeholder, à l'aide de l'option 'attr'.
- Nous désactivons l'affichage automatique des erreurs avec 'error_bubbling' => true.
- Enfin, nous définissons 'label' à false pour supprimer l'étiquette par défaut du champ.

Le champ 'plainPassword' (mot de passe) :

- Nous utilisons le type 'RepeatedType : : class' pour demander à l'utilisateur de saisir son mot de passe deux fois afin de s'assurer qu'il n'y a pas d'erreur de saisie.
- Ensuite, nous définissions les options 'first_options' et 'second_options' pour spécifier les options des deux champs de mot de passe. Dans ce cas, nous définissons des labels (« 'label' => false ») et des attributs HTML supplémentaires, tels que les classes CSS et les placeholders, pour chaque champ.
- Pour chaque champ de mot de passe, nous ajoutons des attributs tels que la classe CSS (« 'class' => 'register-password-input' ») et le placeholder (« 'placeholder' => '**********' »).
- Nous désactivons également l'affichage automatique des erreurs avec « 'error_bubbling' => true ».

- Pour le deuxième champ de mot de passe, nous ajoutons uniquement l'attribut 'placeholder', car les autres attributs sont déjà définis pour le premier champ.

Une fois que l'utilisateur a rempli ces champs, il soumet le formulaire. Les données saisies sont ensuite envoyées au contrôleur dédié, qui traite la requête et valide les données reçues, dont voici le code :

```php
class RegistrationController extends AbstractController
{
    #[Route('/register', name: 'app_register')]
    public function register(Request $request, UserPasswordHasherInterface $userPasswordHasher, UserAuthenticatorInterface
    $userAuthenticator, UsersAuthenticator $authenticator, EntityManagerInterface $entityManager): Response
    {
        $user = new Users();
        $form = $this->createForm(RegistrationFormType::class, $user);
        $form->handleRequest($request);

        if ($form->isSubmitted() && $form->isValid()) {
            // encode the plain password
            $user->setPassword(
                $userPasswordHasher->hashPassword(
                    $user,
                    $form->get('plainPassword')->getData()
                )
            );

            $entityManager->persist($user);
            $entityManager->flush();
            // do anything else you need here, like send an email

            return $userAuthenticator->authenticateUser(
                $user,
                $authenticator,
                $request
            );
        }

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

Voici l'explication de cette page :

Ce code représente le contrôleur RegistrationController, responsable du traitement des données soumises par le formulaire d'inscription.

- La méthode register est annotée avec #[Route('/register', name: 'app_register')], ce qui signifie que cette méthode est associée à l'URL /register et au nom de route app_register.

- La méthode prend en paramètres un objet Request ainsi que plusieurs services, notamment UserPasswordHasherInterface, UserAuthenticatorInterface, UsersAuthenticator, et EntityManagerInterface.

- À l'intérieur de la méthode, un nouvel objet Users est créé pour représenter l'utilisateur. Ensuite, un formulaire d'inscription est créé en utilisant la méthode createForm de l'objet $this, en spécifiant le type de formulaire RegistrationFormType et l'objet utilisateur $user.

- Le formulaire traite la requête actuelle avec handleRequest($request), ce qui permet de lier les données soumises par l'utilisateur au formulaire.

- Ensuite, le code vérifie si le formulaire a été soumis et s'il est valide avec if ($form->isSubmitted() && $form->isValid()). Si c'est le cas, le mot de passe de l'utilisateur est haché à l'aide de UserPasswordHasherInterface, puis l'utilisateur est persisté en base de données avec $entityManager->persist($user) et $entityManager->flush().

- Enfin, l'utilisateur est authentifié à l'aide de UserAuthenticatorInterface et UsersAuthenticator, et la méthode renvoie la réponse appropriée.

- Si le formulaire n'est pas soumis ou s'il n'est pas valide, la méthode rend la vue de formulaire d'inscription registration/register.html.twig, en passant le formulaire à la vue pour qu'elle puisse être affichée à l'utilisateur.

Maintenant je vais vous présenter le code de la vue (register.html.twig) :

```
<div class="container">
            <div class="infos">

                <div class="logo">
                    <img class="logoEntier" src="assets/images/logo_entier.png" alt="logo">
                </div>

                    <h1 class="titreh1"> Créer un compte </h1>

                {{ form_start(registrationForm) }}

                <div class="reponse">
                    {{ form_row(registrationForm.nom) }}
                </div>

                <div class="reponse">
                    {{ form_row(registrationForm.prenom) }}
                </div>

                  <div class="reponse">
                    {{ form_row(registrationForm.ville) }}
                </div>

                <div class="reponse">
                    {{ form_row(registrationForm.telephone) }}
                </div>

                <div class="reponse">
                    {{ form_row(registrationForm.email) }}
                </div>

                <div class="reponse">
                    {{ form_row(registrationForm.plainPassword.first) }}
                </div>

                <div class="reponse">
                    {{ form_row(registrationForm.plainPassword.second) }}
                </div>

                <div class="have-account">
                    <p class="phrase"> Vous avez un compte ?
                        <strong>
                            <a href="{{ path('app_login') }}"> Connexion </a>
                        </strong>
                    </p>
                </div>

                 {% if app.user %}
                    <div class="dejaCo">
```

Voici l'explication de cette page :

Ce code représente la vue associée à l'inscription de l'utilisateur. Voici une explication détaillée :

- {% extends 'base.html.twig' %} : Cette ligne indique que cette vue étend un modèle de base appelé 'base.html.twig', ce qui signifie qu'elle hérite de la structure et des éléments communs définis dans ce modèle.

- {% block title %}Créer un compte{% endblock %} : Ce bloc définit le titre de la page, qui sera affiché dans l'onglet du navigateur.

- {% block stylesheets %} ... {% endblock %} : Ce bloc permet d'inclure des fichiers de style CSS spécifiques à cette vue. Dans cet exemple, un lien vers un fichier CSS externe est ajouté pour gérer les styles de la création de compte.

- {% block body %} ... {% endblock %} : Ce bloc contient le contenu principal de la page. C'est ici que le formulaire d'inscription et d'autres éléments de la vue sont définis.

- Dans la section du corps de la vue :
    - On commence par définir une structure de conteneur ( ) pour organiser les éléments de la page.

    - On affiche le logo du site.

    - On affiche un titre (<h1 class="titreh1"> Créer un compte </h1>) pour indiquer à l'utilisateur qu'il est sur la page de création de compte.

    - On utilise {{ form_start(registrationForm) }} pour démarrer le formulaire d'inscription.

- Les différents champs du formulaire (nom, prenom, ville, telephone, email, plainPassword.first, plainPassword.second) sont affichés à l'aide de {{ form_row() }}.

- On fournit un lien vers la page de connexion (Connexion).

- On vérifie si l'utilisateur est déjà connecté, auquel cas un message de bienvenue avec un lien de déconnexion est affiché.

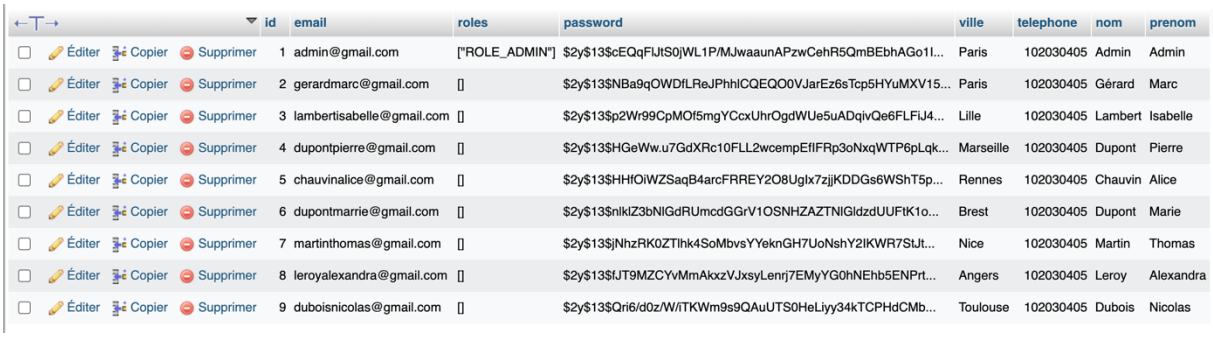
- Enfin, un bouton de soumission du formulaire est ajouté (<button class="crer-compte"> Créer un compte </button>).

- {% if app.user %} : Cette condition vérifié si un utilisateur est déjà connecté. La variable app.user est fournie par Symfony et contient les informations de l'utilisateur actuellement connecté. Si un utilisateur est connecté, le contenu à l'intérieur de ce bloc est affiché.

    - Dans ce bloc, nous affichons un message de bienvenue personnalisé avec le nom d'utilisateur ({{ app.user.userIdentifier }}) et un lien de déconnexion (Déconnexion.

    - {{ form_errors(registrationForm) }} : Cette ligne affiche les éventuelles erreurs globales du formulaire, telles que des messages d'erreur généraux qui ne sont pas associés à un champ spécifique.

Voici à quoi ressemble la table « users » avec les différents utilisateurs qui sont inscrits à l'intérieur de celle-ci :



	id	email	roles	password	ville	telephone	nom	prenom
Éditer Copier Supprimer	1	admin@gmail.com	["ROLE_ADMIN"]	$2y$13$cEQqFlJtS0jWL1P/MJwaaunAPzwCehR5QmBEbhAGo1I...	Paris	102030405	Admin	Admin
Éditer Copier Supprimer	2	gerardmarc@gmail.com	[]	$2y$13$NBa9qOWDfLReJPhhlCQEQO0VJarEz6sTcp5HYuMXV15...	Paris	102030405	Gérard	Marc
Éditer Copier Supprimer	3	lambertisabelle@gmail.com	[]	$2y$13$p2Wr99CpMOf5mgYCcxUhrOgdWUe5uADqivQe6FLFiJ4...	Lille	102030405	Lambert	Isabelle
Éditer Copier Supprimer	4	dupontpierre@gmail.com	[]	$2y$13$HGeWw.u7GdXRc10FLL2wcempEfIFRp3oNxqWTP6pLqk...	Marseille	102030405	Dupont	Pierre
Éditer Copier Supprimer	5	chauvinalice@gmail.com	[]	$2y$13$HHfOiWZSaqB4arcFRREY2O8UgIx7zjjKDDGs6WShT5p...	Rennes	102030405	Chauvin	Alice
Éditer Copier Supprimer	6	dupontmarrie@gmail.com	[]	$2y$13$nlklZ3bNlGdRUmcdGGrV1OSNHZAZTNlGldzdUUFtK1o...	Brest	102030405	Dupont	Marie
Éditer Copier Supprimer	7	martinthomas@gmail.com	[]	$2y$13$jNhzRK0ZTlhk4SoMbvsYYeknGH7UoNshY2IKWR7StJt...	Nice	102030405	Martin	Thomas
Éditer Copier Supprimer	8	leroyalexandra@gmail.com	[]	$2y$13$fJT9MZCYvMmAkxzVJxsyLenrj7EMyYG0hNEhb5ENPrt...	Angers	102030405	Leroy	Alexandra
Éditer Copier Supprimer	9	duboisnicolas@gmail.com	[]	$2y$13$Qri6/d0z/W/iTKWm9s9QAuUTS0HeLiyy34kTCPHdCMb...	Toulouse	102030405	Dubois	Nicolas

Nous pouvons apercevoir que tous les utilisateurs sont uniquement de simples utilisateurs, à l'exception de l'administrateur qui a le rôle « admin » dans la colonne associée, et qui possède donc des privilèges que nous détaillerons ci-dessous.

À présent, je vais aborder les formulaires, des éléments cruciaux sur le site car ils comprennent trois types : le formulaire de témoignage, le formulaire d'avis et le formulaire de demande de contact. Je vais vous présenter le code de celui permettant de laisser un témoignage, les deux autres étant pratiquement identiques.

Tout d'abord voici à quoi ressemble le formulaire :

```php
class LaisserTemoignagesFormType extends AbstractType
{
    public function buildForm(FormBuilderInterface $builder, array $options): void
    {
        $builder
            ->add('contenu', TextType::class, [
                'attr' => [
                    'placeholder' => 'Contenu',
                ],
                'label' => false
                ])


            ->add('users', EntityType::class, [
                'class' => 'App\Entity\Users', // Assurez-vous que le chemin est correct
                'choice_label' => 'nom', // Remplacez 'nom' par le champ que vous voulez afficher
                'label' => false,
            ])


            ->add('prestations', EntityType::class, [
                'class' => 'App\Entity\Prestations', // Assurez-vous que le chemin est correct
                'choice_label' => 'libelle', // Remplacez 'nom' par le champ que vous voulez afficher
                'label' => false,
            ])

            ->add('submit', SubmitType::class, [
                'attr' => [
                    'class' => 'btn btn-primary mt-4'
                ],
                'label' => 'Envoyer'
            ])
        ;
    }

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

Je ne vais pas répéter les mêmes explications que pour le formulaire d'inscription, car les autres formulaires sont pratiquement identiques. Un formulaire reste un formulaire, et il n'y a pas de grandes différences entre eux.

Maintenant voici le contrôleur lié à ce formulaire :

```php
<?php

namespace App\Controller;

use Symfony\Bundle\FrameworkBundle\Controller\AbstractController;
use Symfony\Component\HttpFoundation\Response;
use Symfony\Component\Routing\Annotation\Route;
use Doctrine\Persistence\ManagerRegistry;
use Symfony\Component\HttpFoundation\Request;
use App\Repository\TemoignagesRepository;
use App\Entity\Temoignages;
use App\Form\LaisserTemoignagesFormType;
use Symfony\Component\HttpFoundation\Session\SessionInterface;
use Doctrine\ORM\EntityManagerInterface;

class LaisserTemoignagesController extends AbstractController
{
    #[Route('/LaisserTemoignages', name: 'app_laisser_temoignages')]
    public function index(Request $request, EntityManagerInterface $entityManager): Response
    {
        // On crée un nouvel objet Temoignages
        $temoignages = new Temoignages();
        // On crée le formulaire grâce à LaisserTemoignagesFormType et on lui passe l'objet temoignages
        $form = $this->createForm(LaisserTemoignagesFormType::class, $temoignages);

        // On récupère les données du formulaire
        $form->handleRequest($request);

        // Si le formulaire est soumis et valide
        if ($form->isSubmitted() && $form->isValid()) {
            $temoignages = $form->getData();

            // On enregistre le temoignages en base de données
            $entityManager->persist($temoignages);
            $entityManager->flush();

            $this->addFlash('success', 'Votre temoignage a été envoyé avec succès !');

            // On redirige l'utilisateur vers la page d'temoignages
            return $this->redirectToRoute("app_laisser_temoignages");
        }
        return $this->render('laisser_temoignages/laisser_temoignages.html.twig', [
            // On envoie le formulaire à la vue
            'laisserTemoignagesForm' => $form->createView(),
        ]);
    }
}
```

Je ne vais pas entrer dans les détails de ce code, car il présente des similitudes avec celui de l'inscription que nous avons examiné précédemment. Toutefois, je tiens à souligner que le flux de travail et la logique de fonctionnement restent largement les mêmes. Les différences spécifiques entre les deux sont principalement liées aux entités utilisées, aux routes définies et à l'ajout d'un messages flash lors du succès de l'ajout.

Voici le message de succès affiché lorsque l'utilisateur a réussi à laisser un témoignage, un message similaire est également présent sur les autres pages. La mise en page de la page ci-dessous est complète à 95 %, ce qui explique pourquoi l'affichage ne correspond pas parfaitement à la maquette.

À présent, je vais vous présenter les privilèges accordés à un administrateur lorsqu'il est connecté, ainsi que les fonctionnalités spécifiques qui lui sont attribuées.

Tout d’abord, l'administrateur a la possibilité de modifier, d'ajouter ou de supprimer les différentes prestations présentées sur la page, ainsi que sur la page Formations, sans nécessairement avoir accès à la base de données.

De même, il peut supprimer un avis sur les pages Témoignages et Avis s'il le juge nécessaire, tout en ayant la possibilité, comme les autres utilisateurs, d'en ajouter un.

Enfin, sur la page de présentation, l'administrateur a le pouvoir de modifier le texte affiché, représentant sa propre description, s'il estime que celle-ci doit être ajustée.

Je vais maintenant vous présenter le code que j’ai réalisé pour mettre en œuvre ces différents privilèges administratifs. Pour illustrer cela, je vais m’appuyer sur la page de prestations, en vous présentant d’abord son formulaire. Je ne vais pas entrer dans les détails car, une fois de plus, il reste similaire aux autres.

Cependant, en ce qui concerne le contrôleur, il diffère légèrement des autres. En effet, il contient plusieurs routes permettant d’afficher, de supprimer, d’éditer et d’ajouter une prestation. Je vais maintenant vous expliquer les routes spécifiques à ce dernier.

Voici la partie du contrôleur permettant l'affichage des prestations :

```php
#[Route('/prestations', name: 'app_prestations')]
public function index(PrestationsRepository $prestationsRepository): Response
{
    $prestations = $prestationsRepository->findAll();
    return $this->render('prestations/prestations.html.twig', [
        'prestations' => $prestations,
    ]);
}
```

Dans ce code, nous avons une méthode nommée index() dans le contrôleur.

Cette méthode est associée à la route /prestations avec le nom app_prestations. Lorsque cette route est accédée, la méthode récupère toutes les prestations à partir du repository PrestationsRepository en utilisant la méthode findAll().

Ensuite, elle passe ces données à la vue 'prestations/prestations.html.twig' en tant que variable 'prestations'. Enfin, la méthode renvoie cette vue avec les données des prestations, qui seront affichées à l'utilisateur.

Voici la partie du contrôleur permettant l'édition d'une prestation :

```php
#[Route('/prestations/{id}/edit', name: 'app_prestations_edit')]
public function edit(Request $request, EntityManagerInterface $entityManager,
PrestationsRepository $prestationsRepository, int $id): Response
{
    // Récupération de la prestation à éditer grâce à son id
    $prestation = $prestationsRepository->find($id);

    // Si la prestation n'existe pas, on peut rediriger ou afficher un message d'erreur
    if (!$prestation)
    {
        // Redirection vers la liste des prestations avec un message d'erreur
        $this->addFlash('error', 'La prestation n\'existe pas.');
        return $this->redirectToRoute('app_prestations');
    }

    // Création du formulaire
    $editForm = $this->createForm(PrestationType::class, $prestation);

    // Traitement du formulaire
    $editForm->handleRequest($request);

    // Si le formulaire est soumis et valide
    if ($editForm->isSubmitted() && $editForm->isValid()) {
        // Enregistrement de la prestation
        $entityManager->flush();

        // Redirection vers la liste des prestations avec un message de succès
        $this->addFlash('success', 'La prestation a été modifiée avec succès.');
        return $this->redirectToRoute('app_prestations');
    }

    // Renvoi du formulaire à la vue
    return $this->render('prestations/edit.html.twig', [
        'editForm' => $editForm->createView(),
    ]);
}
```

Dans cette partie du contrôleur, nous avons une méthode nommée edit() associée à la route /prestations/{id}/edit avec le nom app_prestations_edit.
Cette méthode est responsable de l'édition d'une prestation spécifique identifiée par son ID.

Voici ce qui se passe dans cette méthode :

- Récupération de la prestation à éditer à partir de son ID en utilisant le repository PrestationsRepository.

- Vérification si la prestation existe. Si elle n'existe pas, un message d'erreur est ajouté et l'utilisateur est redirigé vers la liste des prestations.

- Création du formulaire d'édition en utilisant la classe PrestationType et la prestation récupérée.

- Traitement du formulaire soumis par l'utilisateur.

- Si le formulaire est soumis et valide, la prestation est mise à jour dans la base de données en appelant la méthode flush() de l'EntityManager.

- Un message de succès est ajouté, et l'utilisateur est redirigé vers la liste des prestations.

- Si le formulaire n'est pas soumis ou n'est pas valide, le formulaire est renvoyé à la vue pour être affiché à nouveau avec les éventuelles erreurs.

Voici la partie du contrôleur permettant la suppression d’une prestation :

```php
#[Route('/prestations/{id}/delete', name: 'app_prestations_delete')]
public function delete(EntityManagerInterface $entityManager, int $id): Response
{
    try {
        // Récupération de la prestation par son id
        $prestation = $entityManager->getRepository(Prestations::class)->find($id);

        // Vérifier si la prestation existe
        if (!$prestation) {
            throw new \Exception('La prestation avec l\'ID ' . $id . ' n\'existe pas.');
        }

        // Suppression de la prestation
        $entityManager->remove($prestation);

        // Enregistrement de la suppression en BDD
        $entityManager->flush();

        // Ajout d'un message de succès
        $this->addFlash('success', 'La prestation a été supprimée avec succès.');

        // Redirection vers la liste des prestations
        return $this->redirectToRoute('app_prestations');

    }

    catch (\Exception $e)
    {
        // Ajout d'un message d'erreur
        $this->addFlash('error', 'Une erreur s\'est produite lors de la suppression de la prestation.');

        // Redirection vers la liste des prestations
        return $this->redirectToRoute('app_prestations');
    }
}
```

Dans cette partie du contrôleur, nous avons une méthode nommée delete() associée à la route /prestations/{id}/delete avec le nom app_prestations_delete. Cette méthode est responsable de la suppression d'une prestation spécifique identifiée par son ID.

Voici ce qui se passe dans cette méthode :

- Récupération de la prestation à supprimer à partir de son ID en utilisant le repository PrestationsRepository.

- Vérification si la prestation existe. Si elle n'existe pas, un message d'erreur est ajouté et l'utilisateur est redirigé vers la liste des prestations.

- Suppression de la prestation de la base de données en appelant la méthode remove() de l'EntityManager.

- Enregistrement des changements dans la base de données en appelant la méthode flush() de l'EntityManager.

- Ajout d'un message de succès indiquant que la prestation a été supprimée avec succès.

- Redirection de l'utilisateur vers la liste des prestations.

Voici la partie du contrôleur permettant l'ajout d'une prestation :

```php
#[Route('/prestations/add', name: 'app_prestations_add')]
public function add(Request $request, EntityManagerInterface $entityManager, SessionInterface $session): Response
{
    // Création d'une nouvelle prestation
    $prestation = new Prestations();

    // Création du formulaire
    $addForm = $this->createForm(PrestationType::class, $prestation);

    // Traitement du formulaire
    $addForm->handleRequest($request);

    // Si le formulaire est soumis
    if ($addForm->isSubmitted()) {
        // Si le formulaire est valide
        if ($addForm->isValid()) {
            // On enregistre les données en BDD
            $entityManager->persist($prestation);
            $entityManager->flush();

            // Ajouter un message de succès dans la session
            $session->getFlashBag()->add('success', 'La prestation a été ajoutée avec succès.');

            // Redirection vers la liste des prestations
            return $this->redirectToRoute('app_prestations');
        }
        else
        {
            // Ajouter un message d'échec dans la session
            $session->getFlashBag()->add('danger', 'Le formulaire contient des erreurs. Veuillez le corriger.');
        }
    }

    // Renvoi du formulaire à la vue
    return $this->render('prestations/add.html.twig', [
        'addForm' => $addForm->createView(),
    ]);
}
```

Dans cette partie du contrôleur, nous avons une méthode nommée add() associée à la route /prestations/add avec le nom app_prestations_add. Cette méthode est responsable de l'ajout d'une nouvelle prestation.

Voici ce qui se passe dans cette méthode :

- Création d'une nouvelle instance de la classe Prestations, qui représente la nouvelle prestation à ajouter.

- Création du formulaire d'ajout de prestation en utilisant la méthode createForm().

- Traitement du formulaire : récupération des données soumises par l'utilisateur et association de ces données à l'objet Prestations.

- Si le formulaire est soumis :
    - Si le formulaire est valide, les données de la prestation sont enregistrées en base de données en utilisant la méthode persist() de l'EntityManager.

    - Ajout d'un message de succès dans la session pour informer l'utilisateur que la prestation a été ajoutée avec succès.

- Redirection de l'utilisateur vers la liste des prestations.

- Si le formulaire n'est pas valide, un message d'erreur est ajouté dans la session pour informer l'utilisateur des erreurs dans le formulaire.

- Renvoi du formulaire à la vue pour qu'il soit affiché à l'utilisateur.

A présent, je vais vous présenter les différentes vues liées aux prestations, il y en a trois, une « générale », une spécifique à l'édition et une spécifique à l'ajout.

Voici le code de la vue nommée 'prestations.html.twig', qui affiches ces dernières :

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

{% block title %}Prestations{% endblock %}

{% block stylesheets %}
    <link href="{{ asset('assets/css/style_prestations.css') }}" rel="stylesheet">
    <link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/font-awesome/5.15.4/css/all.min.css">
{% endblock %}

{% block body %}
    <body>

    <div class="container">
        <div class="title">
            <h1 class="titreh1">Prestations</h1>
        </div>

        <div class="flash-messages">
            {% for message in app.flashes('success') %}
                <div class="alert alert-success">
                    {{ message }}
                </div>
            {% endfor %}

            {% for message in app.flashes('danger') %}
                <div class="alert alert-danger">
                    {{ message }}
                </div>
            {% endfor %}
        </div>


       <div class="infos1">
            {{ _self.renderPrestationsBlock("Réalisme", prestations|filter(p => p.categorie == "Réalisme")) }}
            {{ _self.renderPrestationsBlock("Expresionisme", prestations|filter(p => p.categorie == "Expressionisme")) }}
        </div>

        <div class="infos2">
            {{ _self.renderPrestationsBlock("Modernisme", prestations|filter(p => p.categorie == "Modernisme")) }}
            {{ _self.renderPrestationsBlock("Styles contemporains", prestations|filter(p => p.categorie == "Styles contemporains")) }}
        </div>


        {% if app.user and is_granted('ROLE_ADMIN') %}
            <div class="btnAjouter">
                <a class="add" href="{{ path('app_prestations_add') }}">Ajouter une prestation</a>
            </div>
        {% endif %}
```

```
        <div class="temoignages">
            <a href="{{ path('app_temoignages') }}">Voir les témoignages</a>
        </div>
    </div>
    </body>

    {% block javascript %}
        <script>
            document.addEventListener("DOMContentLoaded", function() {
                const deleteButtons = document.querySelectorAll('.btnDelete');

                deleteButtons.forEach(button => {
                    button.addEventListener('click', function(event) {
                        event.preventDefault(); // Empêcher le lien de s'exécuter immédiatement

                        const confirmMessage = button.getAttribute('data-delete-message');
                        const confirmDelete = confirm(confirmMessage);
                        if (confirmDelete) {
                            window.location.href = button.getAttribute('href'); // Naviguer vers l'URL de suppression
                        }
                    });
                });
            });

            document.addEventListener("DOMContentLoaded", function() {
            // Sélectionnez tous les messages de succès
            const successMessages = document.querySelectorAll('.alert-success');

            // Masquer automatiquement les messages de succès après un délai
            successMessages.forEach(message => {
                setTimeout(() => {
                    message.style.display = 'none';
                }, 3000); // Masquer après 3 secondes
            });
        });
        </script>
    {% endblock %}

{% endblock %}
```

```
{% macro renderPrestationsBlock(categorie, prestations) %}
    <div class="cadre">
        <h1>{{ categorie }}</h1>
        {% for prestation in prestations %}
            <div class="prestation-item">
                <p>{{ prestation.libelle }}</p>

                {% if app.user and is_granted('ROLE_ADMIN') %}
                    <div class="btns">
                        <a class="btnAdmin" href="{{ path('app_prestations_edit', {'id': prestation.id}) }}">
                            <i class="fas fa-pencil-alt"></i>
                        </a>

                        <a class="btnAdmin btnDelete" href="{{ path('app_prestations_delete', {'id': prestation.id}) }}"
                            data-delete-message="Êtes-vous sûr de vouloir supprimer la prestation '{{ prestation.libelle }}' ?">
                            <i class="fas fa-trash-alt"></i>
                        </a>
                    </div>
                {% endif %}
            </div>
        {% endfor %}
    </div>
{% endmacro %}
```

Maintenant, je vais vous expliquer ce que fait le code de cette vue :

- {% extends 'base.html.twig' %} : Cela indique que cette vue hérite de la structure définie dans le fichier de base base.html.twig.

- {% block title %}Prestations{% endblock %} : Ce bloc définit le titre de la page, qui sera affiché dans l'onglet du navigateur.

- {% block stylesheets %} : Ce bloc permet d'inclure des fichiers de style CSS spécifiques à cette vue. Dans cet exemple, deux fichiers CSS sont inclus : style_prestations.css et all.min.css (qui est une bibliothèque d'icônes).

- {% block body %} : Ce bloc contient le contenu principal de la page.

- Dans la section {% for message in app.flashes('success') %}, on affiche tous les messages de succès générés par le contrôleur.

- Dans la section {% for message in app.flashes('danger') %}, on affiche tous les messages d'erreur générés par le contrôleur.

- La méthode renderPrestationsBlock est définie comme une macro dans Twig. Une macro est une fonction réutilisable dans les templates Twig qui permet de générer du code HTML de manière dynamique.
  Dans ce cas, la macro 'renderPrestationsBlock' prend deux arguments : categorie et prestations. Elle affiche les prestations regroupées par catégorie uniquement si l'utilisateur est connecté en tant qu'administrateur. Le code HTML généré par cette macro est utilisé pour afficher les différentes catégories de prestations sur la page.
  J'ai également ajouté des icônes pour la suppression et la modification en utilisant la bibliothèque Font Awesome qui est une bibliothèque d'icônes vectorielles qui propose une vaste gamme d'icônes pour une utilisation dans les applications web.

- Cette méthode est appelée deux fois dans cette vue, une fois pour le bloc infos1 et une fois pour le bloc infos2.

- Dans la section {% if app.user and is_granted('ROLE_ADMIN') %}, on affiche un lien permettant d'ajouter une nouvelle prestation seulement si l'utilisateur est connecté et a le rôle d'administrateur.

- Le JavaScript inclus à la fin de la vue dans le bloc {% block javascript %} ajoute des fonctionnalités supplémentaires à la page. Tout d'abord, il permet la confirmation avant de supprimer une prestation, assurant ainsi que l'utilisateur confirme son action.
  De plus, il assure une expérience utilisateur plus fluide en fermant automatiquement les messages de succès après un délai de 3 secondes, améliorant ainsi la lisibilité de la page.

Voici le code de la vue nommée ‘edit.html.twig’ qui est propre à la modification d’une prestation :

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

{% block title %}Editer une prestation{% endblock %}

{% block stylesheets %}
    <link href="assets/css/style_modifsPrestas.css" rel="stylesheet">
{% endblock %}

{% block body %}
    {{ form_start(editForm) }}
    {{ form_row(editForm.categorie) }}
    {{ form_row(editForm.libelle) }}
    <button type="submit"
            style="
                font-size:18px;
                border: 2px solid black;
                border-radius: 35px;
                background-color: black;
                color: white;
            ">
            Modifier
    </button>
    {{ form_end(editForm) }}
{% endblock %}
```

Maintenant, je vais vous expliquer ce que fait le code de cette vue :

- Dans cette vue, nous commençons par étendre le modèle de base "base.html.twig".
- Ensuite, nous définissons le titre de la page comme "Editer une prestation" dans le bloc "title".
- Dans le bloc "stylesheets", nous incluons un lien vers une feuille de style CSS externe pour gérer les styles spécifiques à cette page.
- Le bloc "body" contient le formulaire de modification de la prestation. Nous utilisons la fonction form_start() pour démarrer le formulaire et form_end() pour le terminer.
- Entre les deux, nous utilisons form_row() pour afficher chaque champ du formulaire.
- Enfin, nous avons un bouton "Modifier" pour soumettre le formulaire de modification. Ce bouton est stylisé avec des propriétés CSS spécifiques pour l'apparence.

Enfin, voici le code de la vue nommée ‘add.html.twig’ qui est propre à l’ajout d’une prestation :

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

{% block title %}Ajouter une prestation{% endblock %}

{% block stylesheets %}
    <link href="assets/css/style_addPrestas.css" rel="stylesheet">
{% endblock %}

{% block body %}
    {# Affichage des messages de flash #}
    {% for flashMessage in app.flashes('success') %}
        <div class="alert alert-success">{{ flashMessage }}</div>
    {% endfor %}
    {% for flashMessage in app.flashes('danger') %}
        <div class="alert alert-danger">{{ flashMessage }}</div>
    {% endfor %}

    {{ form_start(addForm) }}
    {{ form_row(addForm.categorie) }}
    {{ form_row(addForm.libelle) }}
    <button type="submit"
            style="
                font-size:18px;
                border: 2px solid black;
                border-radius: 35px;
                background-color: black;
                color: white;
            ">
            Ajouter
    </button>
    {{ form_end(addForm) }}
{% endblock %}
```

Maintenant, je vais vous expliquer ce que fait le code de cette vue :

- Dans cette vue, nous commençons par étendre le modèle de base "base.html.twig". Ensuite, nous définissons le titre de la page comme "Ajouter une prestation" dans le bloc "title".

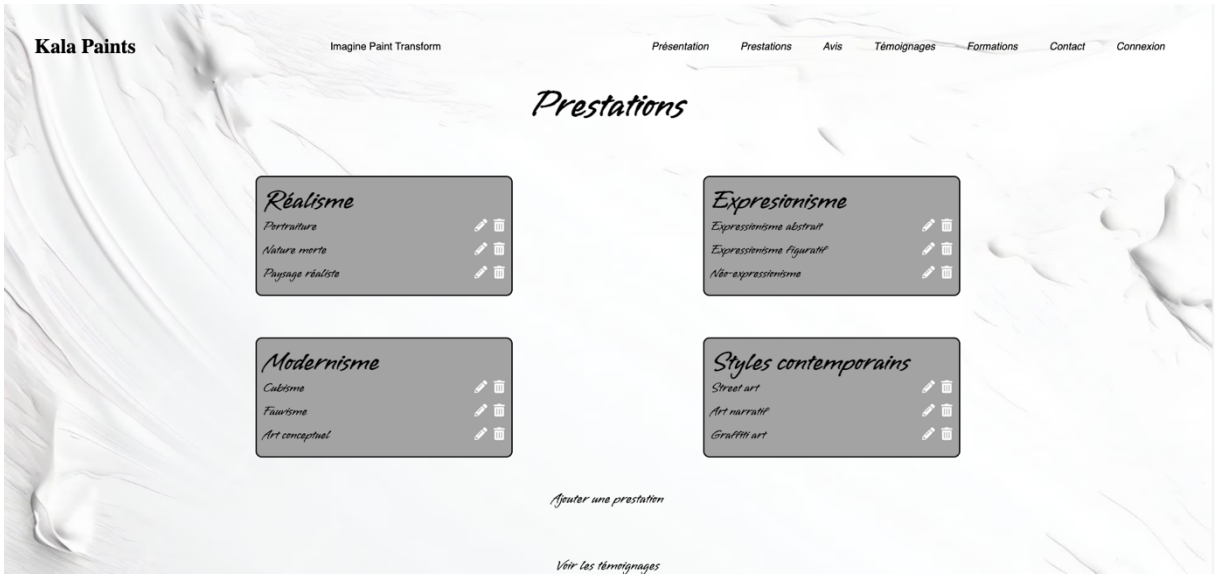
- Dans le bloc "stylesheets", nous incluons un lien vers une feuille de style CSS externe pour gérer les styles spécifiques à cette page.

- Le bloc "body" commence par afficher les messages flash de succès et d'erreur à l'aide d'une boucle Twig.

- Ensuite, nous utilisons form_start() pour démarrer le formulaire d'ajout de prestation et form_end() pour le terminer. Entre les deux, nous utilisons form_row() pour afficher chaque champ du formulaire.

- Enfin, nous avons un bouton "Ajouter" pour soumettre le formulaire d'ajout de prestation. Ce bouton est stylisé avec des propriétés CSS spécifiques pour l'apparence.

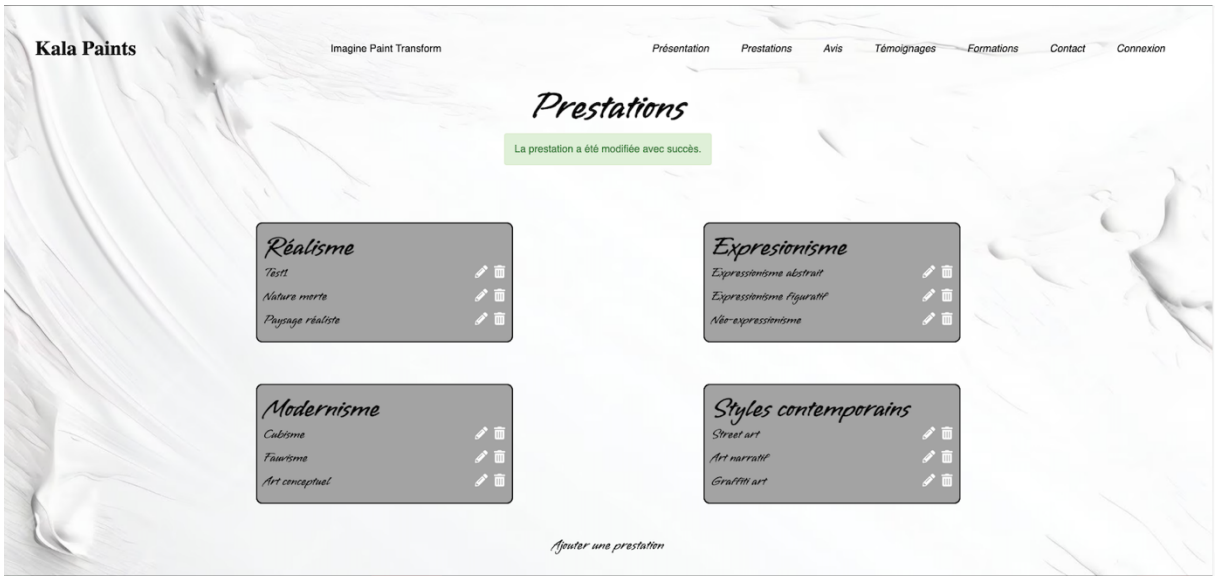
Voici ce à quoi ça ressemble, pour appel seul l'administrateur a cette vue :



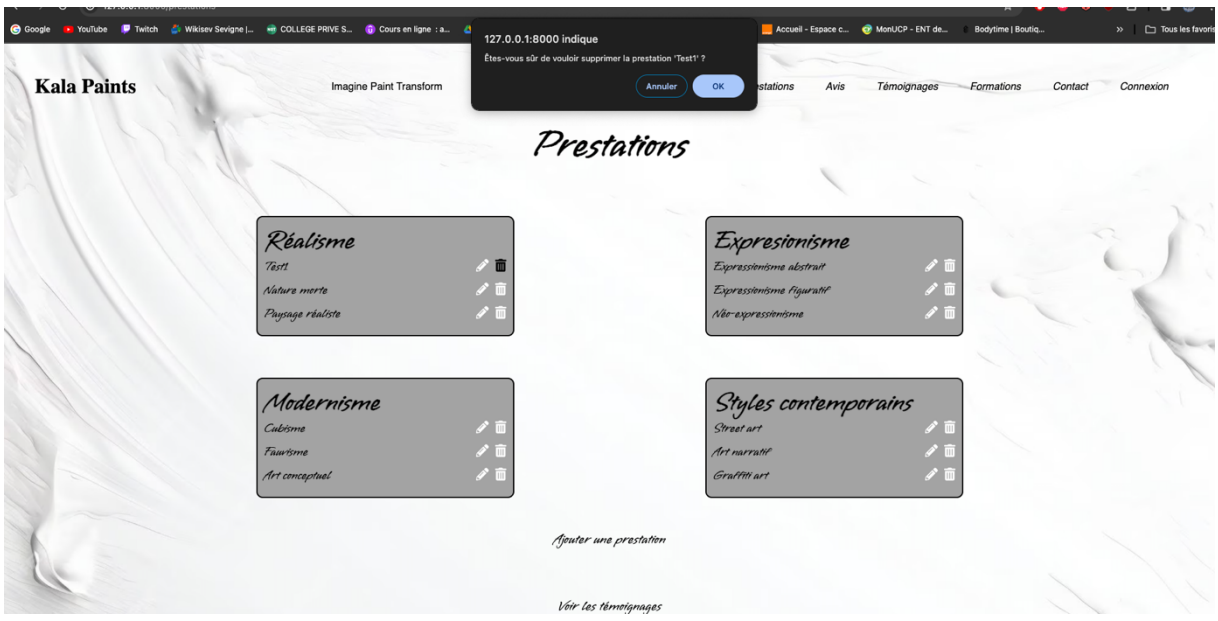
En cliquant sur les petites icônes, il peut éditer ou bien supprimer une prestation, voici la page sur laquelle il arrive s'il choisit de modifier par exemple la première prestation :

L'administrateur peut donc modifier les éléments présents dans la table 'Prestations' de la base de données, à savoir la catégorie et le libellé de la prestation.

Lorsqu'il clique sur 'Modifier', les modifications sont instantanément enregistrées dans la base de données et reflétées dans la vue. De plus, un message de succès s'affiche pendant 3 secondes pour confirmer que la modification a bien été prise en compte.
Par exemple, si l'administrateur remplace 'Portraiture' par 'Test1', voici ce qu'il verra, ainsi que les utilisateurs du site après la modification :

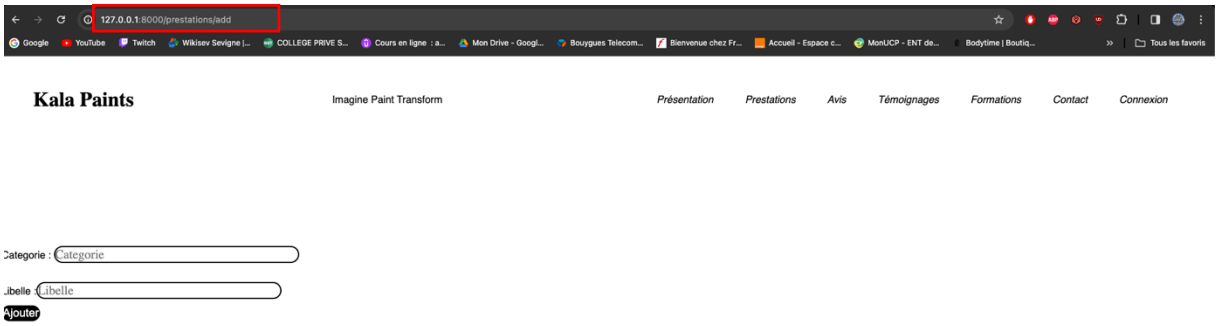


Voici maintenant ce qu'il voit s'il souhaite supprimer une prestation :



Le message de confirmation lui indique précisément s'il souhaite la suppression sur laquelle il a cliqué.

A présent, voici la page sur laquelle il arrive s'il clique sur le bouton 'Ajouter une prestation', on peut bien voir que l'URL est différente de celle de la modification :

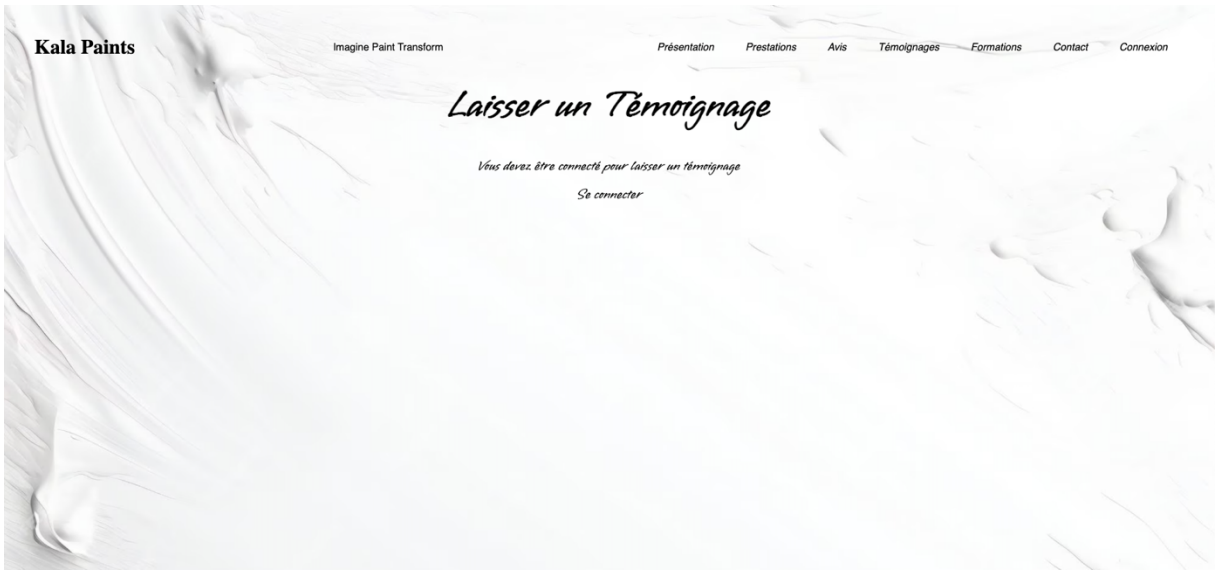


Étant donné que les autres pages du site sont pratiquement similaires sur le plan visuel et fonctionnel aux pages déjà présentées en détail, je ne vais pas les aborder de manière exhaustive dans ce rapport. En effet, la structure et le fonctionnement de ces pages sont largement répétitifs, ce qui se reflète également dans le code associé.

Cependant, je ne négligerai pas pour autant d'évoquer les subtilités et fonctionnalités uniques présentes sur ces pages, ainsi que la logique sous-jacente qui les distingue des autres. Ces détails permettront de fournir une compréhension complète du développement du site dans son ensemble.

Notamment en ce qui concerne les pages impliquant la communication avec l'utilisateur, telles que les pages 'Témoignages', 'Avis' et 'Contact'. Sur ces trois pages, l'utilisateur a la possibilité de laisser un témoignage, un avis ou de soumettre une demande de contact avec le peintre.

Cependant, ces actions sont uniquement accessibles aux utilisateurs connectés. Dans le cas où un utilisateur non connecté tenterait d'accéder à ces fonctionnalités, un message lui serait affiché, comme illustré dans le screen ci-dessous :



Ceci est présent grâce à ce else :

```
{% else %}
    <div class="pasCo">
        <p> Vous devez être connecté pour laisser un témoignage</p>
            <a href="{{ path('app_login') }}"> Se connecter </a>
        </div>
    </div>
{% endif %}
```

Une autre subtilité importante est la conception de la barre de navigation (Navbar), qui a été rendue responsive pour s'adapter à différents écrans. Je vais vous expliquer en détail comment cela a été réalisé dans le code CSS ci-dessous :

```css
*{
    margin: 0;
    padding: 0;
    text-decoration: none;
    list-style: none;
}

body{
    font-family: 'Poppins', sans-serif;
    background-size: cover;
}

header{
    height: 100vh;
    width: 100vw;
    background-image: url(images/landscape.png);
    background-size: cover;
}

.navbar {
    position: absolute;
    padding: 50px;
    display: flex;
    justify-content: space-between;
    width: 100%;
    align-items: center;
    box-sizing: border-box;
}

.navbar a{
    color: black;
}

.navbar .logo{
    font-size: 2em;
    font-weight: bold;
    font-family: 'Roboto';

}

.navbar .logo2{
    font-size: 1em;
}

.navbar .nav-links ul{
    display: flex;
    font-style: italic;

}

.navbar .nav-links ul li{
    margin: 0 25px;
}
```

```css
.navbar .menu_hamburger {
    display: none;
    width: 55px;
    height: 45px;
}


@media screen and (max-width: 1335px){
    .navbar{
        padding: 0;
    }

    .navbar .logo{
        position: absolute;
        top: 50px;
    }

    .navbar .logo2{
        position: absolute;
        top: 87px
    }


    .navbar .menu_hamburger{
        position: absolute;
        top: 50px;
        right: 20px;
        display: block;
    }

    .nav-links{
        top: 0;
        left: 0;
        position: absolute;
        background-color: rgb(255, 255, 255, 0.2999);
        backdrop-filter: blur(7px);
        width: 100%;
        height: 100vh;
        display: flex;
        justify-content: center;
        align-items: center;
        margin-left: -100%;
        transition: all 0.5s ease;
    }

    .nav-links.mobile-menu{
        margin-left: 0;
    }
    .nav-links ul{
        display: flex;
        flex-direction: column;
        align-items: center;
    }

    .navbar .nav-links ul li{
        margin: 25px 0;
        font-size: 1.5em;
    }

}
```

Maintenant, je vais vous expliquer ce que fait le code de ce style :

- * {...} : Cette règle applique les styles de base à tous les éléments HTML sur la page. Elle supprime les marges par défaut, les rembourrages, les décorations de texte et les puces de liste. Cela garantit une uniformité visuelle sur l'ensemble du site.

- body {...} : Ces styles sont appliqués au corps de la page HTML. Ils définissent la police de caractères utilisée pour tout le texte sur la page et spécifient que l'image de fond doit couvrir toute la surface disponible.

- header {...} : Cette section détaille les styles spécifiques de l'en-tête de la page. Elle lui donne une hauteur et une largeur de 100% de la vue et définit une image de fond qui s'étire pour remplir l'en-tête.

- .navbar {...} : Ces styles sont appliqués à la barre de navigation en haut de la page. La barre de navigation est positionnée de manière absolue pour qu'elle reste fixe en haut de la fenêtre même lorsque l'utilisateur fait défiler la page. Les éléments à l'intérieur de la barre de navigation sont disposés horizontalement avec un espacement égal entre eux.

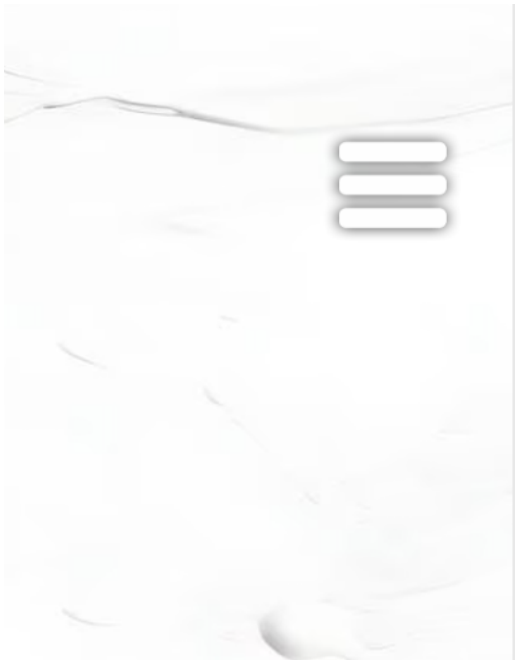
- .navbar a {...} et .navbar .logo {...} : Ces règles définissent le style des liens et du logo à l'intérieur de la barre de navigation. Les liens sont stylisés en noir et le logo est mis en gras et agrandi.

- .nav-links ul {...} : Cette section détaille les styles de la liste des liens dans la barre de navigation. Les liens sont affichés horizontalement les uns à côté des autres.

- .navbar .menu_hamburger {...} : Ces styles sont appliqués au bouton de menu hamburger utilisé sur les écrans de petite taille. Il est affiché initialement en tant que bloc et masqué sur les écrans plus larges, mais devient visible et cliquable lorsque la taille de l'écran est réduite.

- @media screen and (max-width: 1335px) {...} : Cette section contient des styles spécifiques qui s'appliquent uniquement lorsque la largeur de l'écran est inférieure à 1335 pixels. Ces styles ajustent la mise en page de la barre de navigation pour les appareils plus petits, en déplaçant certains éléments (comme le logo) et en redimensionnant les liens pour une meilleure lisibilité sur les écrans plus petits.

Voici à quoi ressemble la navbar une fois le responsive appliqué, avec un bouton burger qui apparait :



Voici le code HTML ou le style est appliqué :

```html
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <link rel="stylesheet" href="/assets/css/style_navbar.css">
</head>
<body>
    <nav class="navbar">
        <a href="/" class="logo">Kala Paints</a>
        <a href="/" class="logo2">Imagine Paint Transform</a>
        <div class="nav-links">
            <ul>
                <li> <a href="/presentation">Présentation</a></li>
                <li> <a href="/prestations">Prestations</a></li>
                <li> <a href="/voiravis">Avis</a></li>
                <li> <a href="/temoignages">Témoignages</a></li>
                <li> <a href="/formations">Formations</a></li>
                <li> <a href="/contact">Contact</a></li>
                <li> <a href="/login">Connexion</a></li>
            </ul>
        </div>
        <img class="menu_hamburger" src="assets/images/menu_hamburger.png" alt="menu hamburger">
    </nav>
</body>
<script>
    const menuHamburger = document.querySelector(".menu_hamburger")
        const navLinks = document.querySelector(".nav-links")

        menuHamburger.addEventListener('click',()=>{
        navLinks.classList.toggle('mobile-menu')
        })
</script>
```

Maintenant, je vais vous expliquer ce que fait le code de cette page :

- <head>...</head> : Cette section contient les métadonnées et les liens vers les ressources externes utilisées par la page. Les métadonnées spécifient l'encodage du caractère et la configuration de la vue pour les appareils mobiles. Le lien <link> charge la feuille de style CSS qui définit les styles de la barre de navigation.

- <body>...</body> : C'est la section principale de la page HTML qui contient tout le contenu visible par l'utilisateur. Dans ce cas, il contient la barre de navigation (<nav class="navbar">) et le script JavaScript associé.

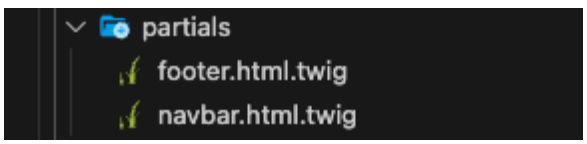
- <nav class="navbar">...</nav> : C'est l'élément qui englobe toute la barre de navigation du site. Il contient deux liens <a> pour le titre de l'entreprise et un pour le slogan de celle-ci. Il contient également un élément avec la classe .nav-links qui enveloppe la liste de liens de navigation.

- ... : C'est une liste non ordonnée qui contient les liens de navigation. Chaque lien est un élément <li> avec un lien <a> à l'intérieur.

- <script>...</script> : C'est une section qui contient du code JavaScript qui définit le comportement interactif de la barre de navigation sur les appareils mobiles. L'événement click est écouté sur l'élément .menu_hamburger, qui est l'icône du menu hamburger. Lorsque l'icône est cliquée, la classe .mobile-menu est ajoutée ou supprimée de l'élément .nav-links, ce qui affiche ou masque les liens de navigation en fonction de leur état actuel.

Voici le rendu final de la navbar responsive, c'est-à-dire ce que les utilisateurs verront sur téléphone ou tablette s'ils souhaitent naviguer sur le site à travers celle-ci.

La présence de la navbar sur toutes les pages est due au fait qu'elle est contenue dans un dossier nommé 'partials' dans les templates.



Cette navbar est ensuite incluse dans le fichier 'base.html.twig', dont voici le code ci-dessous, qui est étendu sur toutes les pages. Ainsi, la navbar se répète systématiquement sur chaque page du site.

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="UTF-8">
        <title>
            {% block title %}Kala Paints
            {% endblock %}
        </title>

        <link rel="icon" href="./assets/images/peintre.png">
        {% block stylesheets %}
            {{ encore_entry_link_tags('app') }}
        {% endblock %}

       {% block navFooter %}
            {#  Styles de la nav et du footer de toutes nos templates #}
            <link rel="stylesheet" href="/assets/css/style_navbar.css">
            <link rel="stylesheet" href="/assets/css/style_footer.css">
        {% endblock %}

        {% block javascripts %}
        {% endblock %}
    </head>

    <body
        class="body">
        {# -- Corps de page des templates -- #}
            {# header relatif a tout nos templates #}
            {% block header %}
                {# inclusion du Header #}
                {% include 'partials/navbar.html.twig' %}
            {% endblock %}

        {% block body %}{% endblock %}

        <footer>
            {# footer relatif a tout nos templates #}
            {% block footer %}
                {# inclusion du pieds de page #}
                {% include 'partials/footer.html.twig' %}
            {% endblock %}
        </footer>
    </body>
```

Voici une courte explication de ce fichier :

- La section <head> contient les métadonnées de la page, comme l'encodage, le titre de la page, et les liens vers les feuilles de style (stylesheets) et les scripts JavaScript.

- Les blocs {% block title %}, {% block stylesheets %}, et {% block javascripts %} permettent aux pages filles d'ajouter leur propre contenu spécifique pour le titre, les feuilles de style et les scripts JavaScript.

- Le bloc {% block navFooter %} permet d'inclure des styles supplémentaires pour la navbar et le footer de toutes les pages du site.

- Le corps de la page est défini dans le bloc {% block body %}. Ce bloc est rempli par le contenu spécifique de chaque page fille.

- Le header (en-tête) est inclus dans le bloc {% block header %} à l'aide de la directive {% include 'partials/navbar.html.twig' %}. Cela permet d'inclure la barre de navigation sur toutes les pages du site.

- Le footer (pied de page) est inclus dans le bloc {% block footer %} à l'aide de la directive {% include 'partials/footer.html.twig' %}. Cela permet d'inclure le pied de page sur toutes les pages du site, même si pour l'instant il n'est pas encore réalisé.

3.2.4.1.3 Difficultés rencontrées en général (maquette/code)

Dans l'ensemble, la transition des maquettes conçues avec Figma vers le code n'a pas posé de difficultés majeures. Ayant opté pour un design relativement simple et intuitif, l'élaboration des maquettes s'est déroulée de manière fluide, sans rencontrer d'obstacles significatifs. Cependant, les véritables défis ont émergé lors de la phase de développement, notamment dans la mise en œuvre de fonctionnalités clés telles que l'ajout, l'édition ou la suppression d'éléments.

Certaines de ces fonctionnalités, bien que relativement courantes dans le développement web, ont présenté des défis particuliers. Par exemple, la gestion des opérations CRUD (Create, Read, Update, Delete) nécessitait une compréhension approfondie des concepts fondamentaux du développement web ainsi que du framework utilisé. La mise en place de ces fonctionnalités, tout en assurant leur intégration harmonieuse dans le reste du système, a exigé une approche méthodique et une résolution créative des problèmes.

Dans ce contexte, la consultation de ressources en ligne et la recherche de solutions sur des forums spécialisés ont été des éléments clés pour surmonter ces difficultés. En fin de compte, ces défis ont offert l'occasion d'une bonne révision et de renforcer mes compétences existantes en développement web.

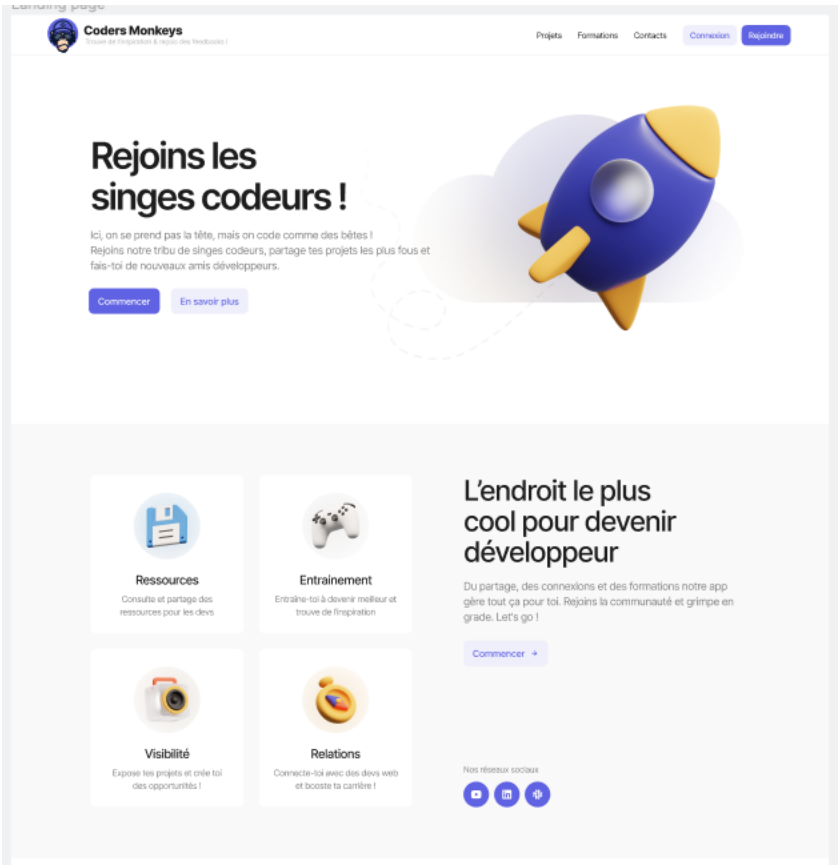
3.2.4.2 Deuxième projet : Création d'un site avec React (à l'aide d'une formation)

Technologies utilisées : React.js, Firebase, Next.js, Tailwind CSS

3.2.4.2.1 Présentation et explication des maquettes

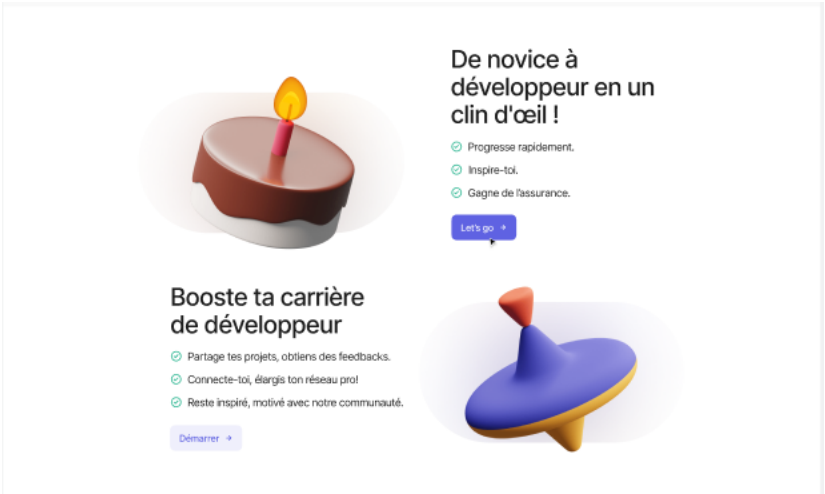
Voici les maquettes que j'avais à disposition :

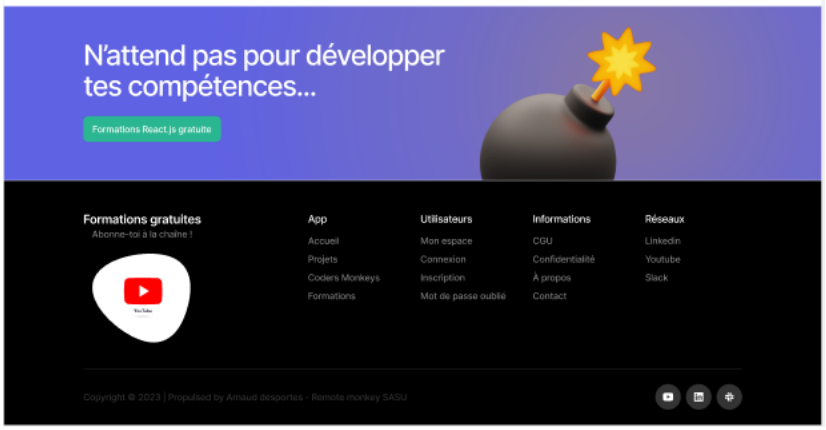
Landing page :



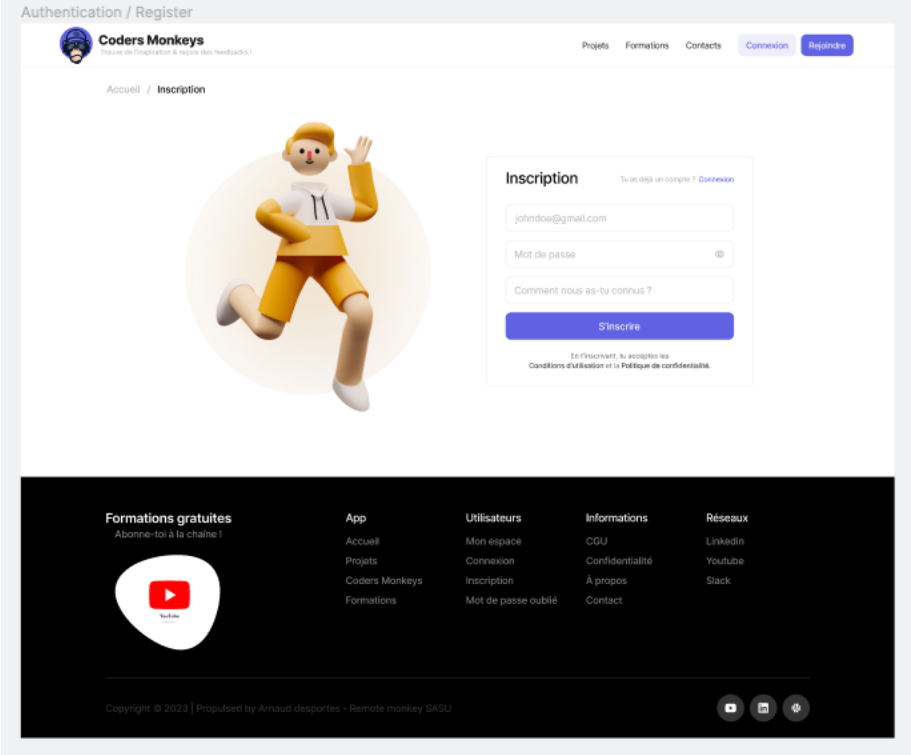

Coders Monkeys
"Rejoins-nous sur le Slack des Singes codeurs„
Rejoins-nous et obtiens de l'aide, des conseils et pourquoi pas des nouveaux potes !
Rejoindre le groupe d'aide


Formations React.js gratuite
Apprends à coder l'app des singes codeurs
Les projets des singes codeurs

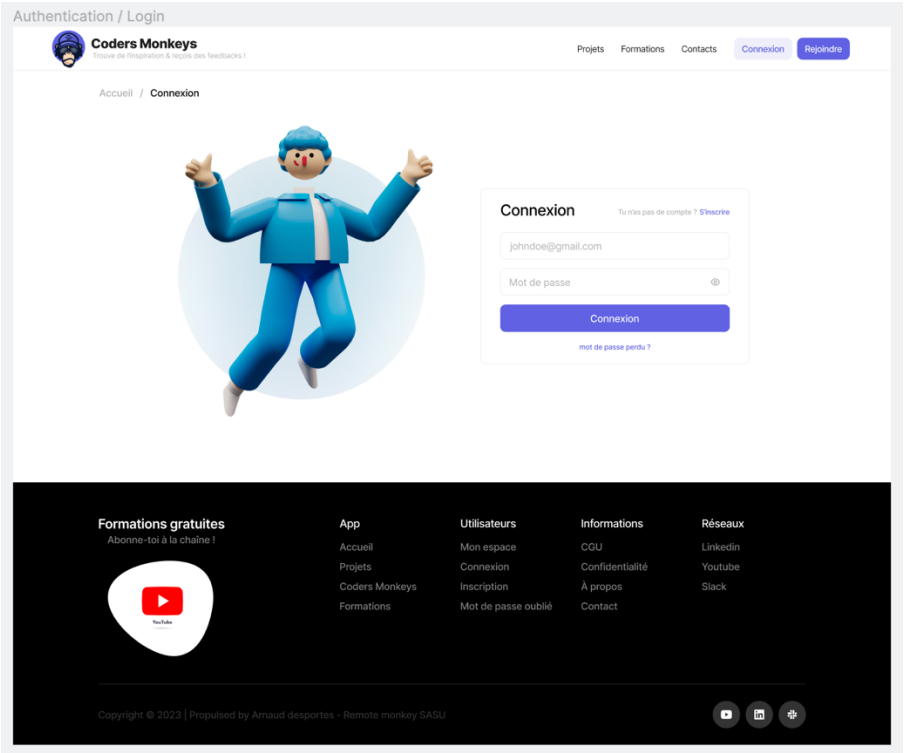

De novice à développeur en un clin d'œil !
Progresse rapidement.
Inspire-toi.
Gagne de l'assurance.
Let's go
Booste ta carrière de développeur
Partage tes projets, obtiens des feedbacks.
Connecte-toi, élargis ton réseau pro!
Reste inspiré, motivé avec notre communauté.
Démarrer


N'attend pas pour développer tes compétences...
Formations React.js gratuite
Formations gratuites
Abonne-toi à la chaine !
App
Accueil
Projets
Coders Monkeys
Formations
Utilisateurs
Mon espace
Connexion
Inscription
Mot de passe oublié
Informations
CGU
Confidentialité
À propos
Contact
Réseaux
Linkedin
Youtube
Slack

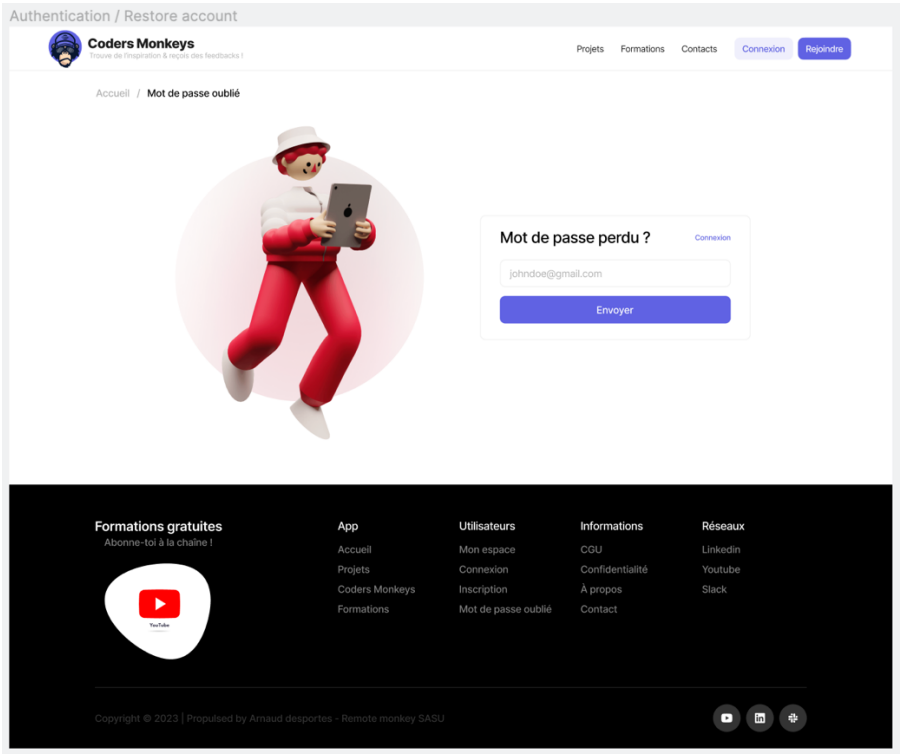
Page Inscription :



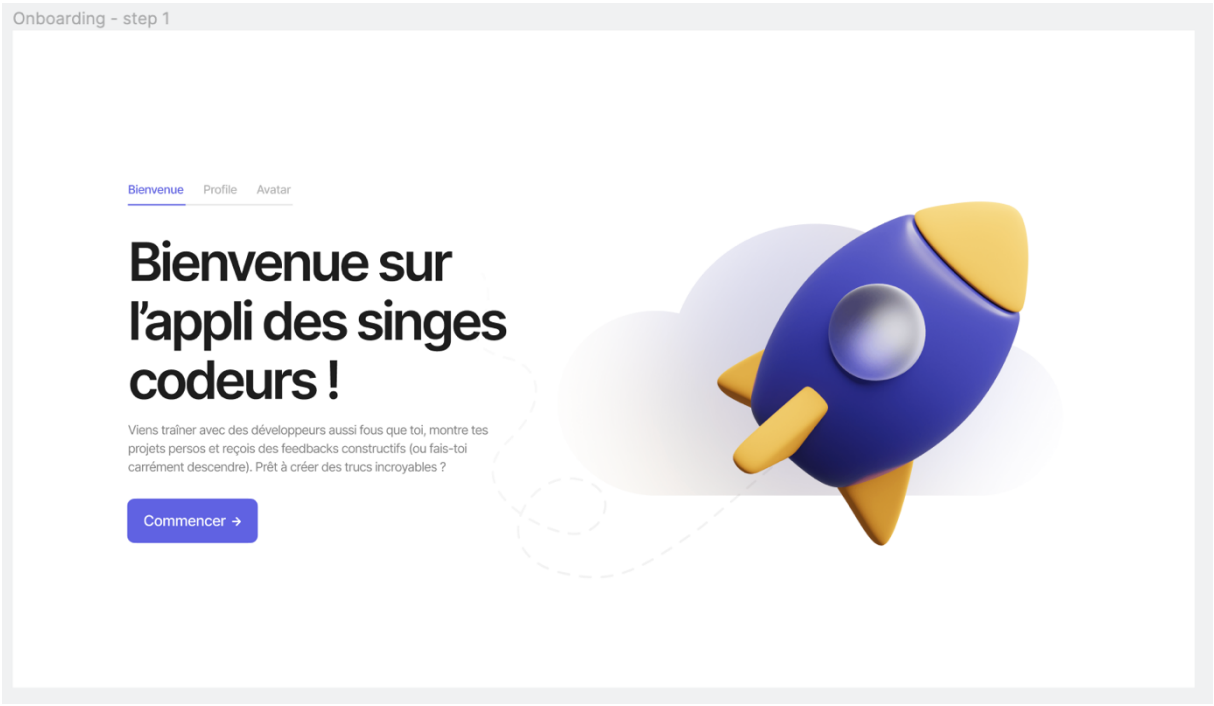
Page Connexion :



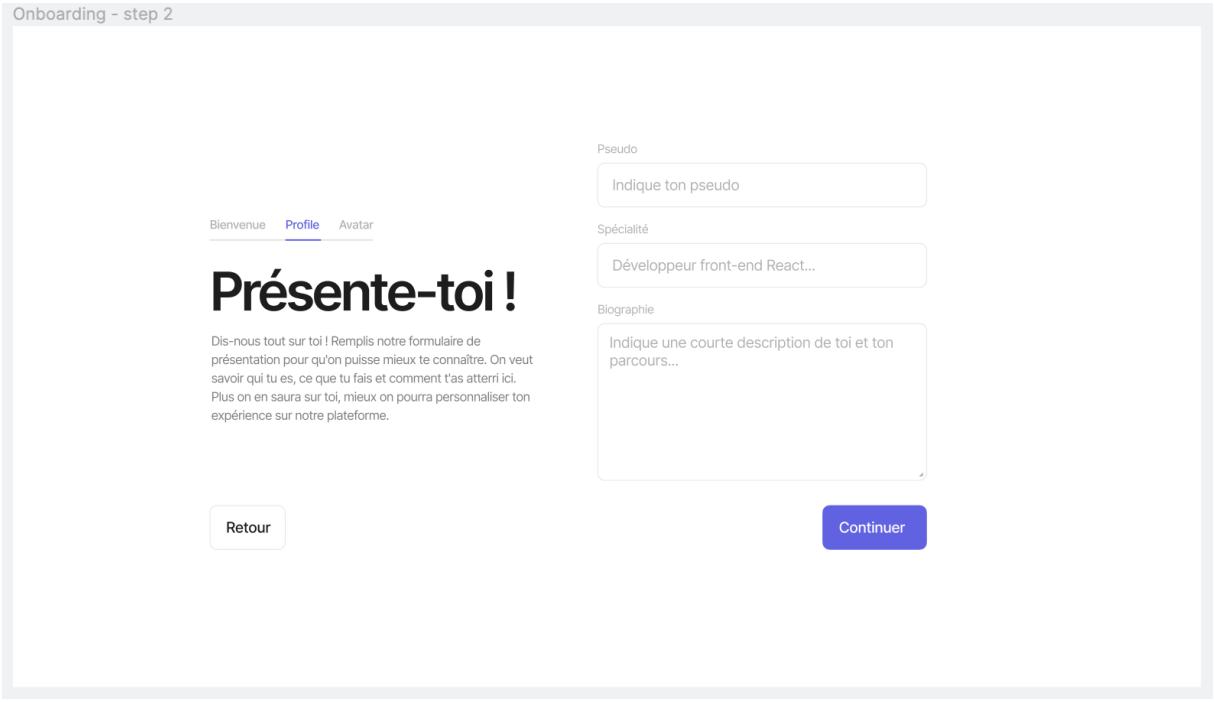
Page Mot de passe oublié :



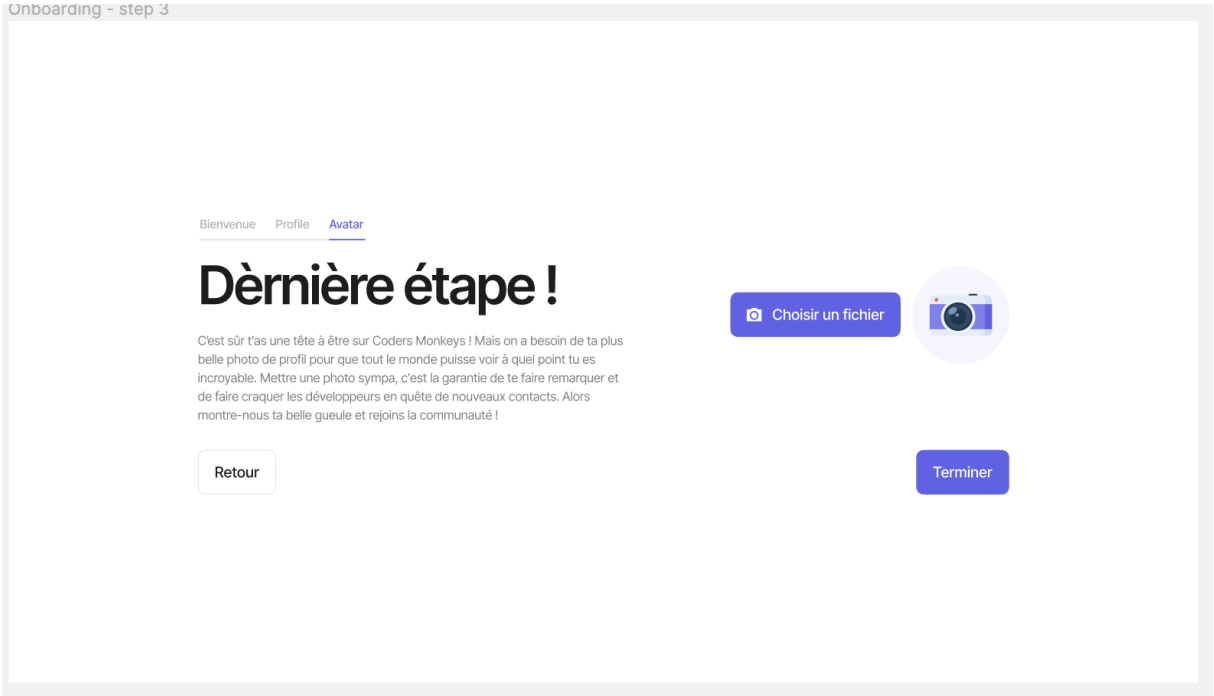
Onboarding Step 1 :



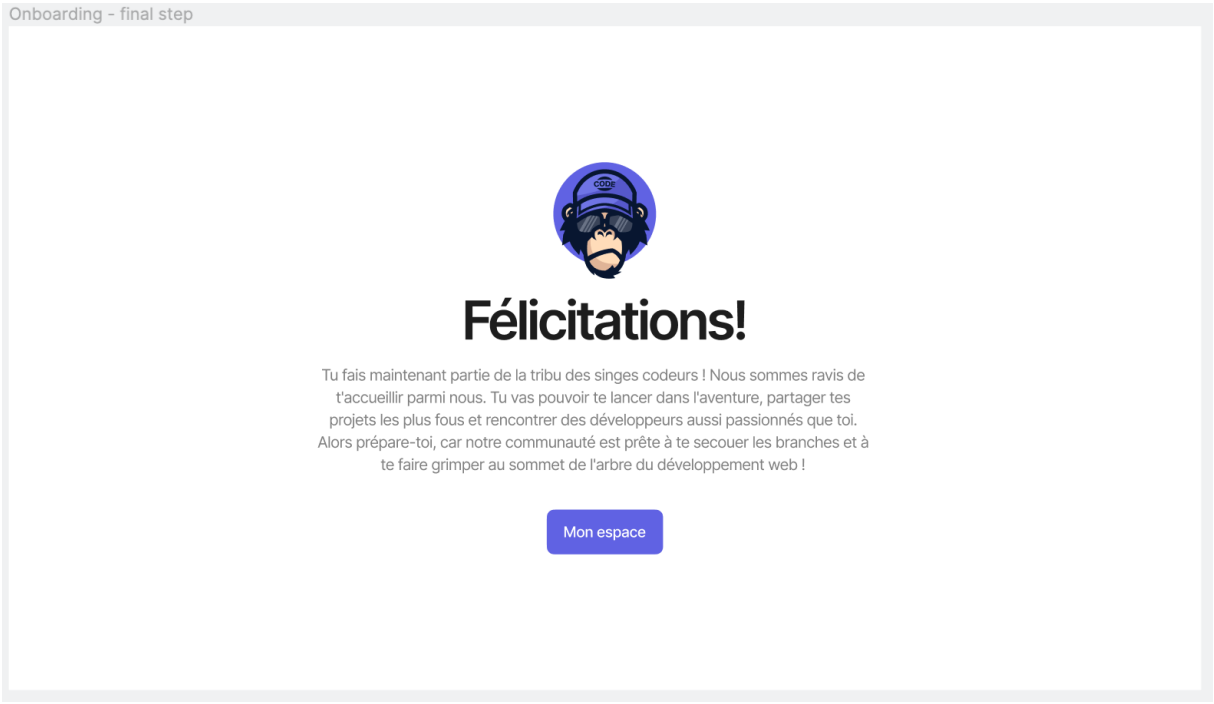
Onboarding Step 2 :



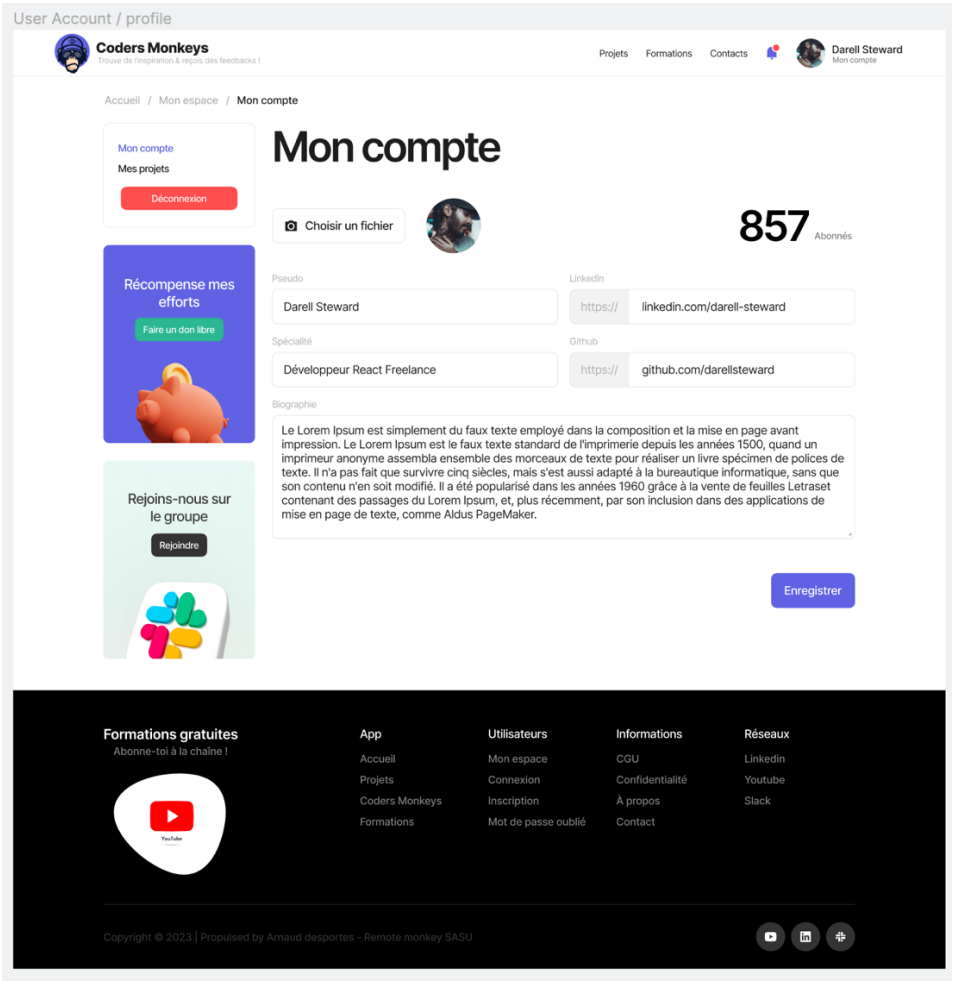
Onboarding Step 3 :



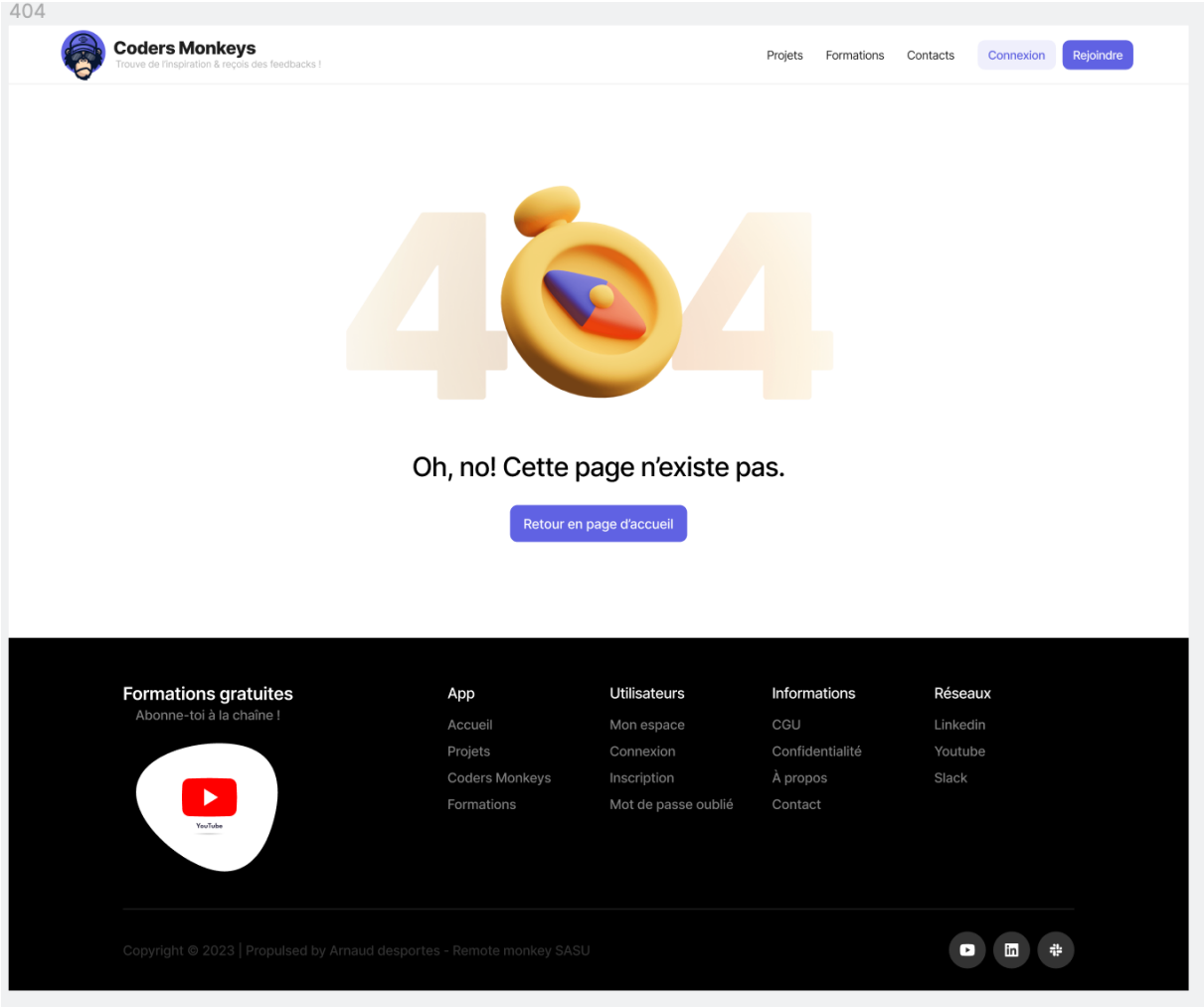
Onboarding final Step :



Page Account :



Page Erreur 404 :



Mon objectif était donc de reproduire ce site en respectant son design, qui était fourni avec la formation proposant la création de ce site. Le but de cette formation était d'avoir un premier aperçu de React, de comprendre le fonctionnement des composants, ainsi que les différentes particularités du framework. Cela incluait également l'apprentissage de Firebase, Tailwind CSS et Next.js.

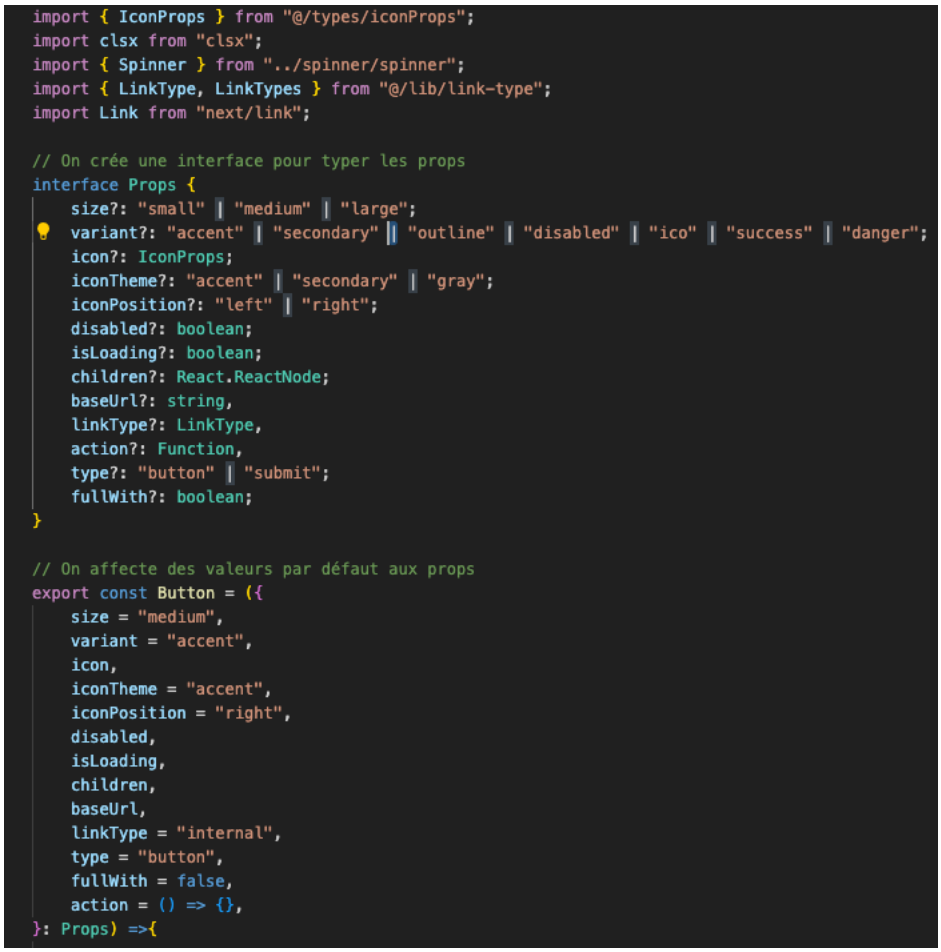
En ce qui concerne le design, j'ai pu découvrir comment une maquette est réellement réalisée de manière professionnelle, en intégrant et utilisant un design system pour assurer l'intégrité du site. Cela incluait notamment les boutons, leurs variantes, les couleurs, et d'autres éléments que nous allons examiner par la suite.

3.2.4.2.2 Explication du code de façon concrète (avec screen)

Je vais donc vous présenter les différents composants réalisés lors de cette application.

Dans un premier temps, le premier sprint consistait à la mise en place du projet, comprenant notamment la création du projet, sa configuration, ainsi que la mise en place du design system. Cela incluait la création des boutons, des icônes, des indicateurs de chargement (spinner), des logos et avatars de l'application.

Voici le code présent dans le fichier ‘button.tsx’ :



```tsx
import { IconProps } from "@/types/iconProps";
import clsx from "clsx";
import { Spinner } from "../spinner/spinner";
import { LinkType, LinkTypes } from "@/lib/link-type";
import Link from "next/link";

// On crée une interface pour typer les props
interface Props {
    size?: "small" | "medium" | "large";
    variant?: "accent" | "secondary" | "outline" | "disabled" | "ico" | "success" | "danger";
    icon?: IconProps;
    iconTheme?: "accent" | "secondary" | "gray";
    iconPosition?: "left" | "right";
    disabled?: boolean;
    isLoading?: boolean;
    children?: React.ReactNode;
    baseUrl?: string,
    linkType?: LinkType,
    action?: Function,
    type?: "button" | "submit";
    fullWith?: boolean;
}

// On affecte des valeurs par défaut aux props
export const Button = ({
    size = "medium",
    variant = "accent",
    icon,
    iconTheme = "accent",
    iconPosition = "right",
    disabled,
    isLoading,
    children,
    baseUrl,
    linkType = "internal",
    type = "button",
    fullWith = false,
    action = () => {},
}: Props) =>{
```

```
let variantStyles: string = "", sizeStyles: string = "", icoSize: number = 0;

switch (variant) {
    case "accent": // Default
        variantStyles =
            "bg-primary hover:bg-primary-400 text-white rounded"
        break;
    case "secondary":
        variantStyles =
            "bg-primary-200 hover:bg-primary-300/50 text-primary rounded"
        break;
    case "outline":
        variantStyles =
            "bg-white hover:bg-gray-400/50 border border-gray-500 text-gray-900 rounded"
        break;
    case "disabled":
        variantStyles =
            "bg-gray-400 border border-gray-500 text-gray-600 rounded cursor-not-allowed"
        break;
    case "success":
        variantStyles =
            "bg-secondary hover:bg-secondary-400 text-white rounded"
        break;
    case "danger":
        variantStyles =
            "bg-alert-danger hover:bg-alert-danger/75 text-white rounded"
        break;
    case "ico":
        if(iconTheme === "accent") { // Default
            variantStyles =
                "bg-primary hover:bg-primary-400 text-white rounded-full"
        }
        if(iconTheme === "secondary") {
            variantStyles =
                "bg-primary-200 hover:bg-primary-300/50 text-primary rounded-full"
        }
        if(i  let variantStyles: string
            variantStyles =
                "bg-gray-800 hover:bg-gray-700 text-white rounded-full"
        }
        break;
}
```

```
switch (size) {
    case "small":
        sizeStyles = `text-caption3 font-medium ${
            variant === "ico" ? "flex items-center justify-center w-[40px] h-[40px]" : "px-[14px] py-[12px]"
        }`;
        icoSize = 18;
        break;
    case "medium": // Default
        sizeStyles = `text-caption2 font-medium ${
            variant === "ico" ? "flex items-center justify-center w-[50px] h-[50px]" : "px-[18px] py-[15px]"
        }`;
        icoSize = 20;
        break;
    case "large":
        sizeStyles = `text-caption1 font-medium ${
            variant === "ico" ? "flex items-center justify-center w-[60px] h-[60px]" : "px-[22px] py-[18px]"
        }`;
        icoSize = 24;
        break;
}

const handleClick = () => {
    if(action) {
        action()
    }
}

const buttonContent = (
    <>
    {isLoading && (
            <div className="absolute inset-0 flex items-center justify-center">
                {variant === "accent" || variant === "ico" ? (<Spinner size="small" variant="white"/>) : (<Spinner size="small"/>)}
            </div>
            )}

            <div className={clsx(isLoading && "invisible")}>
                {/* Condition, si c'est une icone on l'affiche sinon on affiche children */}
                {icon && variant === "ico" ? (
                    <icon.icon size={icoSize}/>
                ) : (
                    <div className={clsx(icon && "flex items-center gap-1")}>
                        {icon && iconPosition === "left" && (
                            <icon.icon size={icoSize}/>
                        )}
                        {children}
                        {icon && iconPosition === "right" && (
                            <icon.icon size={icoSize}/>
                        )}
                    </div>
                )}
            </div>
    </>
)
```

Voici une explication de ce code :

- **Importations de dépendances** :

Les dépendances nécessaires sont importées. Par exemple, IconProps est importé depuis le fichier de définition de type iconProps.ts, clsx est utilisé pour gérer les classes CSS de manière conditionnelle, et Spinner et Link sont importés depuis d'autres fichiers de composants.

- **Définition de l'interface** Props :

Une interface TypeScript est définie pour typer les propriétés acceptées par le composant Button. Par exemple, size, variant, icon, etc. sont des propriétés acceptées par le composant et sont typées en fonction de leurs valeurs attendues.

- **Définition du composant** Button :

Le composant Button est une fonction qui accepte un objet Props en tant qu'argument. Cette fonction est exportée pour être utilisée dans d'autres parties de l'application.

- **Gestion des styles et des classes CSS** :

En fonction des valeurs des propriétés telles que variant et size, des chaînes de classes CSS sont définies. Ces classes sont déterminées à l'aide de structures conditionnelles comme switch ou if-else.

- **Gestion des actions** :

Une fonction handleClick est définie pour gérer les actions du bouton. Cette fonction est appelée lorsque le bouton est cliqué. Par exemple, elle peut exécuter une fonction spécifiée dans la propriété action.

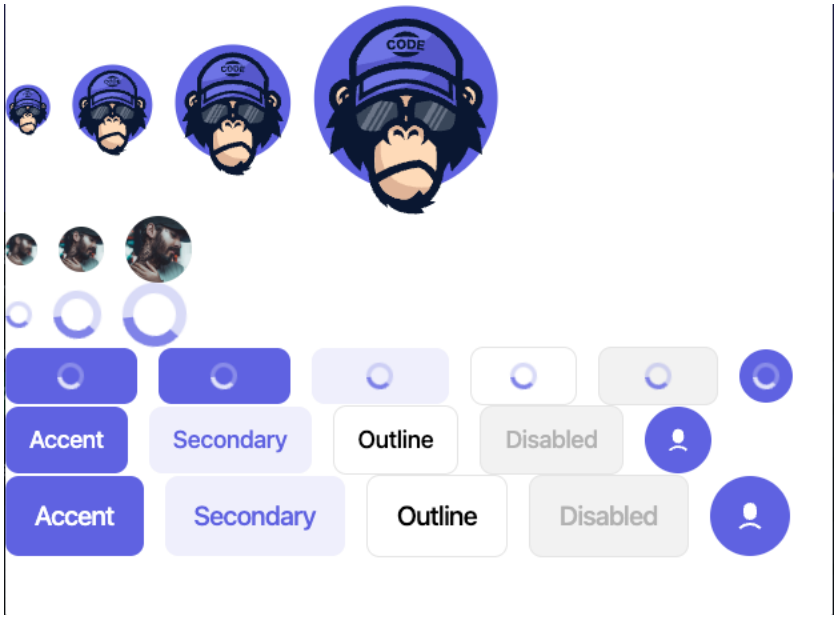
- **Rendu conditionnel du contenu du bouton** :

Le contenu du bouton peut varier en fonction des propriétés passées. Par exemple, si une icône est spécifiée dans la propriété icon, elle est affichée soit à gauche soit à droite du texte du bouton en fonction de la propriété iconPosition.

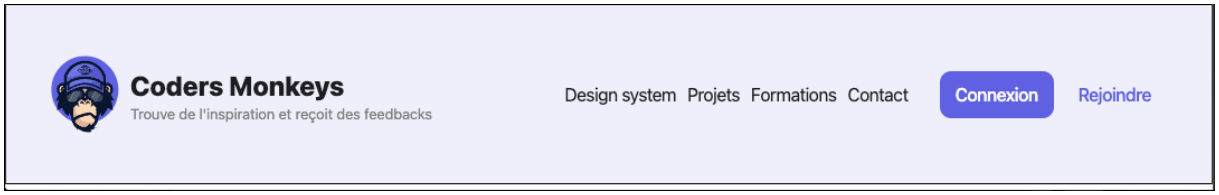
- **Rendu conditionnel du lien** :

Si une URL de base est fournie dans la propriété baseUrl, le bouton peut être rendu comme un lien externe ou un lien interne en fonction du type de lien spécifié dans la propriété linkType.

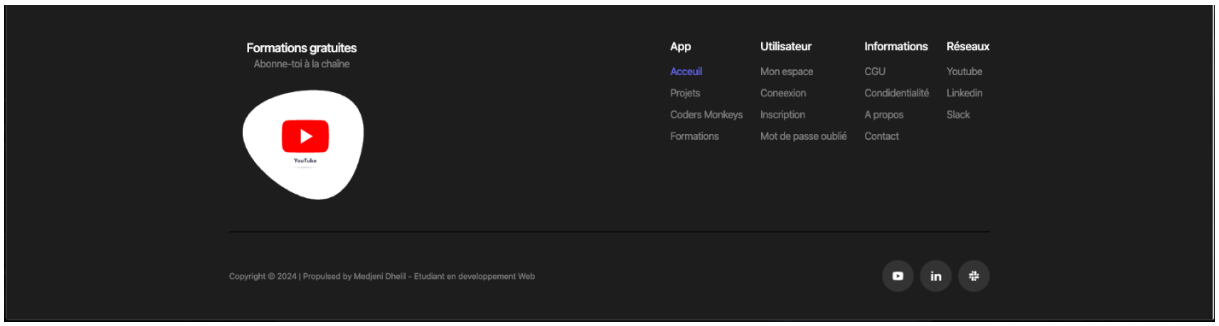
Voici à quoi ressemble les différents éléments du design system, après les avoir configurés, il manque juste la typographie, qui est faite mais pas présente sur le screen ci-dessous :



Ensuite, voici à quoi ressemble la navbar, après être terminée :



Voici également le footer, après être terminé :



Pour le second SPRINT, les principales étapes ont été la création de plusieurs éléments essentiels de l'interface utilisateur. Cela inclut la conception et la mise en place de la barre de navigation (navbar), la création du pied de page (footer), la construction de la page d'accueil (landing page), ainsi que la mise en place d'un fil d'Ariane pour une navigation simplifiée.

Je vais juste vous expliquer comment la landing page a été réalisée. Dans un premier temps, comme dans toute création d'application React, nous avons un fichier 'index.tsx' qui est généré automatiquement.

```tsx
import { Layout } from "@/ui/components/layout/layout";
import { Seo } from "@/ui/components/seo/seo";
import { LandingPageContainer } from "@/ui/modules/landing-page/components/landing-page.container";


export default function Home() {
  return (
    <>

      <Seo title="Coders Monkeys" description="Description..."></Seo>
      <Layout isDisplayBreadcrumbs={false}>
        <LandingPageContainer />
      </Layout>

    </>
  );
}
```

Voici les explications de ce composant :

- Importation des composants : Le code importe trois composants depuis des chemins spécifiques de l'application.
    - Layout est un composant qui gère la mise en page générale de la page, telles que les barres de navigation, les pieds de page, etc.
    - Seo est un composant qui gère les informations de référencement (SEO) de la page, telles que le titre et la description.
    - LandingPageContainer est un composant qui contient la logique et appelle la vue LandingPageView.

- Fonction Home : C'est une fonction fléchée (arrow function) nommée Home qui définit le composant principal de la page d'accueil.

- Retour de la fonction : La fonction retourne un fragment React (<>...</>) contenant deux éléments : Seo et Layout.
    - Seo est utilisé pour définir les métadonnées de référencement de la page, telles que le titre et la description.
    - Layout est utilisé comme conteneur principal pour la mise en page de la page d'accueil. L'attribut isDisplayBreadcrumbs contrôle l'affichage ou non des breadcrumbs.

- LandingPageContainer : Le composant LandingPageContainer est inclus à l'intérieur du composant Layout, ce qui signifie qu'il est rendu dans le contenu principal de la page.

Voici le composant LandingPageContainer, qui appelle la vue LandingPageView :

```tsx
import { LandingPageView } from "./landing-page.view"

export const LandingPageContainer = () => {
    return <LandingPageView />
}
```

Et voici le composant LandingPageView, qui contient les différents éléments composant la Landing Page :

```tsx
import { CallToActionView } from "@/ui/design-system/call-to-action.view.tsx/call-to-action.view"
import { CodersMonkeysSlackView } from "./coders-monkeys-slack/coders-monkeys-slack.view"
import { CurrentCourseCtaView } from "./current-course-cta/current-course-cta.view"
import { FeaturedView } from "./featured/featured.view"
import { HeroTopView } from "./hero-top/hero-top.view"
import { HighLightListView } from "./highlight-list/highlight-list.view"

export const LandingPageView = () => {
    return (
    <div>
        <HeroTopView />
        <FeaturedView />
        <CodersMonkeysSlackView />
        <CurrentCourseCtaView />
        <HighLightListView />
        <CallToActionView />
    </div>
    )
}
```

Voici le composant HeroTopView, représentant la première partie de la page d'accueil :

```tsx
import { Container } from "@/ui/components/container/container"
import { Button } from "@/ui/design-system/button/button"
import { Typography } from "@/ui/design-system/typography/typography"
import Image from "next/image"

export const HeroTopView = () => {
    return (
        <Container className="relative pt-40 pb-52 overflow-hidden">
            <div className="w-full max-w-2xl space-y-5">
                <Typography variant="h2" component="h1" className="max-w-lg">
                    Rejoins les singes codeurs !
                </Typography>
                <Typography variant="body-lg" theme="gray" component="p" className="max-w-xl">
                    Ici, on se prend pas la tête, mais on code comme des bêtes !
                    Rejoins notre tribu de singes codeurs, partage tes projets les plus fous et fais-toi de nouveaux amis développeurs.
                </Typography>
                <div className="space-x-5 pt-2,5">
                    <Button baseUrl="">Commencer</Button>
                    <Button baseUrl="" variant="secondary">En savoir plus</Button>
                </div>
            </div>
            <Image
                src="/assets/svg/rocket.svg"
                alt="Illustration d'une fusée pour matérialiser l'évolution du level des développeurs front-end"
                width={811}
                height={596}
                className="absolute top-0 right-0 z-0"
            />
        </Container>
    )
}
```

Voici une explication de ce code :

- Importations de composants et bibliothèques :
    - Container : Un composant de conteneur qui enveloppe le contenu de la section de héros.
    - Button : Un composant de bouton utilisé pour les appels à l'action.
    - Typography : Un composant de typographie pour afficher des textes stylisés.
    - Image : Un composant Next.js pour l'affichage d'images optimisées.

- Fonction HeroTopView : C'est une fonction fléchée (arrow function) qui définit le composant HeroTopView.

- Contenu de la section de héros :
    - Container : Le contenu de la section est enveloppé dans un composant Container, qui gère la mise en page et la taille du contenu.
    - Typography : Deux composants Typography sont utilisés pour afficher le titre et la description de la section.
    - Boutons : Deux boutons sont inclus pour inciter les utilisateurs à agir.
    - Image : Une illustration de fusée est ajoutée à la section pour apporter une touche visuelle et représenter l'évolution des développeurs.

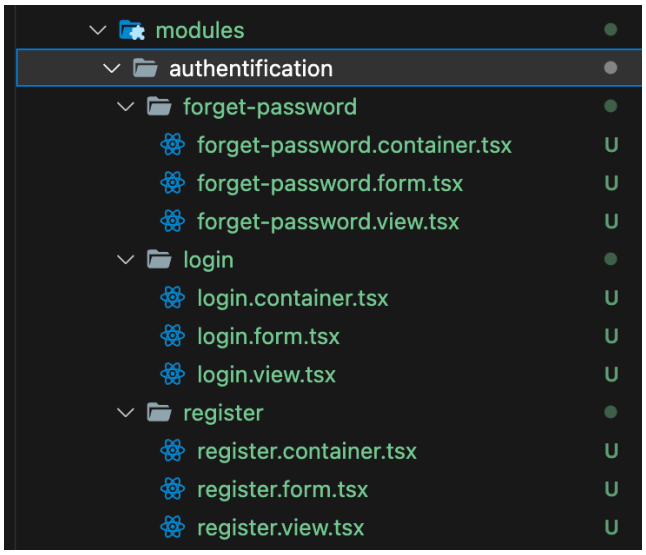
- Attributs des composants :
    - className : Des classes CSS sont ajoutées pour styler les éléments.
    - src, alt, width, height : Pour le composant Image, ces attributs, qui ont été mis en place lors de la réalisation du design system, définissent respectivement la source de l'image, son texte alternatif, sa largeur et sa hauteur.
    - variant, theme, component : Pour les composants Typography, ces attributs, qui ont également étaient mis en place lors de la réalisation du design system, définissent respectivement le style de la typographie, le thème de couleur et le type d'élément HTML.

Pour les autres composants, la procédure est similaire : chaque composant est construit selon les spécifications du design system, avec des attributs tels que className, src, alt, width, height, variant, theme, et component utilisés pour définir les styles, les propriétés de l'image, et les caractéristiques typographiques.

Cependant, comme les principes et le fonctionnement sont essentiellement les mêmes, seuls les détails spécifiques à chaque composant sont adaptés dans le code. Une explication détaillée de ces composants serait redondante car ils suivent essentiellement les mêmes principes et fonctionnements que celui-ci.

À présent, je vais me concentrer sur l'explication du fonctionnement et de la mise en place des pages 'connexion', 'création de compte' et 'mot de passe oublié', ce qui permettra de compléter la présentation des fonctionnalités clés de l'application.

En ce qui concerne la structure pour ces différents composants, j’ai créé des dossiers correspondant à chacune des pages dans le dossier « authentification » lui-même situé dans le dossier « modules », comme vous pouvez le voir ci-dessous :



Comme le montre le screen, pour chacune des pages, nous retrouvons trois composants communs : un container, un formulaire et une vue. Le container agit comme une enveloppe structurante, fournissant un contexte visuel pour les éléments de la page. Le formulaire permet à l'utilisateur d'entrer et de soumettre des données spécifiques, tandis que la vue est responsable de l'affichage des informations et de l'interaction avec l'utilisateur.

Si l’on prend le cas de la création d’un compte, voici le code du container :

```tsx
import { SubmitHandler, useForm } from "react-hook-form"
import { RegisterView } from "./register.view"
import { RegisterFormFieldsType } from "@/types/forms";
import { firebaseCreateUser, sendEmailVerificationProcedure } from "@/api/authentication";
import { toast } from 'react-toastify';
import { useToggle } from "@/hooks/use-toggle";
import { firestoreCreateDocument } from "@/api/firestore";


export const RegisterContainer = () => {

    const { value: isLoading, setValue: setIsLoadding} = useToggle({initial: false});

    const {
        handleSubmit,
        control,
        formState: { errors },
        register,
        setError,
        reset,
    } = useForm<RegisterFormFieldsType>();

    const handleCreateUserDocument = async (collectionName: string, documentID: string, document: object ) => {
        const { error } = await firestoreCreateDocument(collectionName, documentID, document);

        if (error) {
            toast.error(error.message);
            setIsLoadding(false);
            return;
        }
        toast.success("Bienvenue sur l'app des singes codeurs !")
        setIsLoadding(false);
        reset();
        sendEmailVerificationProcedure();
    }

    // Fonction qui va créer l'utilisateur
    const handleCreateUserAuthentication = async ({email, password, how_did_hear}: RegisterFormFieldsType) => {
        console.log("how_did_hear:", how_did_hear); // Affiche la valeur de how_did_hear dans la console
        const { error, data } = await firebaseCreateUser(email, password)
        if (error){
            setIsLoadding(false);
            toast.error(error.message)
            return
        }

        const userDocumentData = {
            email: email,
            how_did_hear: how_did_hear,
            uid: data.uid,
            creation_date: new Date()
        }
```

```
        handleCreateUserDocument("users", data.uid, userDocumentData)
    }

    // Fonction après avoir cliqué sur le bouton de soumission
    const onSubmit: SubmitHandler<RegisterFormFieldsType> = async (formData) => {
        setIsLoadding(true);

        const {email, password} = formData;

        if(password.length <= 5) {
            setError("password", {
                type: "manual",
                message: "Ton mot de passe doit conteni au minimum 6 caractères",
            })
            return
        }
        // On appel la fonction qui va créer l'utilisateur
        handleCreateUserAuthentication(formData);
    }



    return (
        <>
            <RegisterView
                form={{
                    errors,
                    control,
                    register,
                    handleSubmit,
                    onSubmit,
                    isLoading
                }}
            />
        </>
    )
}
```

Voici l'explication de ce code :

- Importations de modules et fonctions :
    - SubmitHandler et useForm de react-hook-form : Ce sont des hooks utilisés pour gérer les soumissions de formulaire et la gestion des formulaires dans React.
    - RegisterView : Il s'agit d'un composant React qui affiche le formulaire d'inscription à l'utilisateur.
    - RegisterFormFieldsType : C'est un type défini pour les champs du formulaire d'inscription.
    - firebaseCreateUser, sendEmailVerificationProcedure, et firestoreCreateDocument : Ce sont des fonctions d'API qui interagissent avec Firebase pour créer un utilisateur, envoyer une vérification par e-mail et créer un document dans Firestore respectivement.
    - toast : Il s'agit d'une bibliothèque utilisée pour afficher des notifications à l'utilisateur.

    - useToggle : Il s’agit d’une fonction utilisée pour gérer un état booléen de manière réactive dans un composant fonctionnel React. Elle permet de basculer entre les valeurs ‘true’ et ‘false’.

    - firestoreCreateDocument : Fonction permettant de créer un document dans la base de données Firestore de Firebase, soit une table. Cela permettra de stocker les informations utilisateur dans la base de données après une inscription.

- Définition du composant RegisterContainer :

    - Ce composant est une fonction React qui gère l'inscription des utilisateurs.

    - Il utilise le hook useToggle pour gérer l'état de chargement pendant le processus d'inscription.

    - Il utilise le hook useForm de react-hook-form pour gérer les états et les méthodes du formulaire d'inscription.

- Fonctions de gestion d'inscription :

    - handleCreateUserDocument : Cette fonction crée un document utilisateur dans Firestore après avoir créé l'utilisateur dans Firebase. Elle prend en paramètre le nom de la collection Firestore, l'ID du document et les données à enregistrer.

    - handleCreateUserAuthentication : Cette fonction crée un utilisateur dans Firebase en utilisant l'adresse e-mail et le mot de passe fournis dans le formulaire d'inscription. Elle gère également les erreurs éventuelles et appelle handleCreateUserDocument pour créer le document utilisateur.

    - onSubmit : Cette fonction est exécutée lorsque l'utilisateur soumet le formulaire d'inscription. Elle vérifie d'abord si le mot de passe est assez long, puis appelle handleCreateUserAuthentication pour créer l'utilisateur.

- Rendu du composant RegisterView :

    - Le composant RegisterView est rendu avec les propriétés nécessaires pour afficher le formulaire d'inscription à l'utilisateur.

Ensuite, voici le code du composant RegisterView, qui permet d’afficher les différents éléments à l’écran :

```tsx
import { Container } from "@/ui/components/container/container"
import { Box } from "@/ui/design-system/box/box"
import { Typography } from "@/ui/design-system/typography/typography"
import Image from "next/image"
import Link from "next/link"
import { RegisterForm } from "./register.form"
import { FormsType } from "@/types/forms"

interface Props {
    form: FormsType
}

export const RegisterView = ({form}: Props) => {
    return (
        <Container className="grid grid-cols-2 gap-20 mb-32">
            <div className="flex items-center">
                <div className="relative w-full h-531px">
                    <Image width={737} height={750} src="/assets/images/perso1.png" alt="Description de l'illustration" className="object-scale-down" />
                </div>
            </div>

            <div className="flex items-center">
                <Box padding_y="py-5">
                    <div className="flex items-center justify-between">
                        <Typography variant="h5" component="h1">
                            Inscription
                        </Typography>

                        <div className="flex items-center gap-2">
                            <Typography variant="caption4" component="span" theme="gray">
                                Tu as déjà un compte ?
                            </Typography>

                            <Typography variant="caption4" component="span" theme="primary">
                                <Link href="/connexion">
                                    Connexion
                                </Link>
                            </Typography>
                        </div>
                    </div>
                    <RegisterForm form={form} ></RegisterForm>

                    <Typography variant="caption4" theme="gray" className="max-w-md mx-auto space-y-1 text-center">
                        <div>En t'inscrivant, tu acceptes les </div>
                        <div>
                            <Link href="#" className="text-gray">
                                Conditions d'utilisation
                            </Link>
                            et la {" "}
                            <Link href="#" className="text-gray">
                                Politique de confidentialité
                            </Link>
                            .
                        </div>
                    </Typography>
                </Box>
            </div>
        </Container>
    )
}
```

Voici les explications de ce dernier, de manière simplifiée :

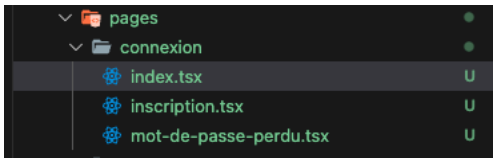
Dans cette vue, nous utilisons les composants créés lors du design system tels que le Container, la Typographie, les Liens, et les Images pour ajuster et organiser correctement les éléments visuels de la page d'inscription. Cela nous permet d'obtenir le même résultat visuel que celui présenté dans la maquette, tout en maintenant une cohérence stylistique et une convivialité dans l'interface utilisateur. De plus, nous appelons également le composant de formulaire d'inscription en utilisant la ligne suivante : <RegisterForm form={form} ></RegisterForm>.

Maintenant, voici le code du formulaire, contenant les différents inputs nécessaires :

```tsx
import { FormsType } from "@/types/forms"
import { Button } from "@/ui/design-system/button/button";
import { Input } from "@/ui/design-system/forms/input";

interface Props {
    form: FormsType;
}

export const RegisterForm = ({form}: Props) => {
    const { control, onSubmit, errors, isLoading, register, handleSubmit } = form

    return (
        <form onSubmit={handleSubmit(onSubmit)} className="pt-8 pb-5 space-y-4">
            <Input
                isLoading= {isLoading}
                placeholder="johndoe@gmail.com"
                type="email"
                register={register}
                errors={errors}
                errorMsg="Vous devez renseignez ce champ"
                id="email"
            />


           <Input
                isLoading = {isLoading}
                placeholder="Mot de passe"
                type="password"
                register={register}
                errors={errors}
                errorMsg="Vous devez renseigner ce champ"
                id="password"
            />

            <Input
                isLoading = {isLoading}
                placeholder="Comment vous nous avez connus ?"
                type="text"
                register={register}
                errors={errors}
                errorMsg="Vous devez renseigner ce champ"
                id="how_did_hear"
            />

            <Button isLoading={isLoading} type="submit" fullWith>S'inscrire</Button>
        </form>
    )
}
```

Ce code crée un formulaire d'inscription avec trois champs : email, mot de passe et une zone de texte pour indiquer comment l'utilisateur a connu le service. Chaque champ utilise le composant d'entrée (Input) du design system, auquel sont passés des paramètres tels que le type de champ, les erreurs éventuelles, et un message d'erreur personnalisé. Le formulaire utilise également le composant de bouton (Button) pour le bouton de soumission, avec une option pour afficher un indicateur de chargement lors de la soumission du formulaire

Ensuite, dans la structure du projet, dans le dossier « pages », trois dossiers sont créés, correspondant respectivement aux pages de connexion, d'inscription et de récupération de mot de passe oublié. Chacun de ces dossiers contient les fichiers nécessaires pour appeler leur container et, par conséquent, leur vue, comme le montre le screen ci-dessous.



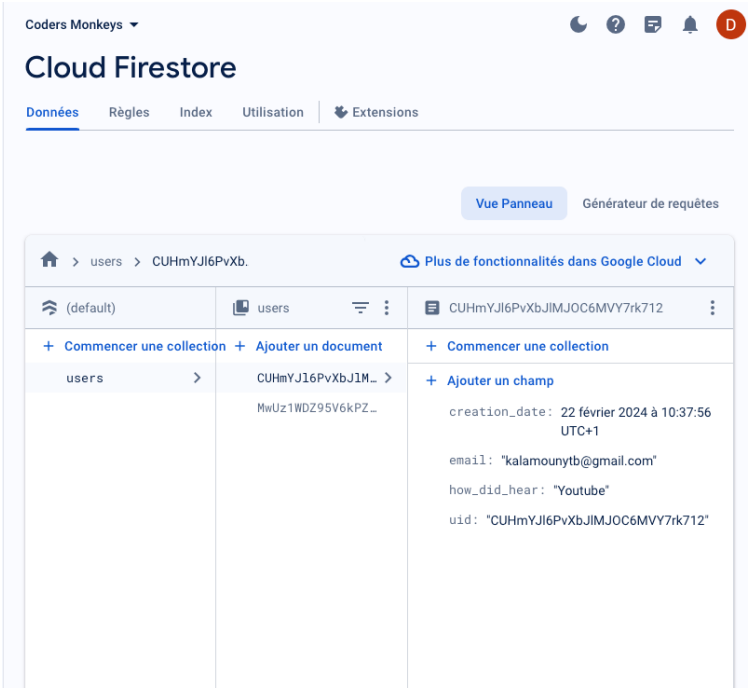
Si l'on reste toujours dans le cas de l'inscription, voici à quoi ressemble le code du composant 'inscription' :

```tsx
import { GUEST } from "@/lib/session-status";
import { Layout } from "@/ui/components/layout/layout";
import { Seo } from "@/ui/components/seo/seo";
import { RegisterContainer } from "@/ui/modules/authentification/register/register.container";


export default function Register() {
  return (
    <>

      <Seo title="Inscription sur Coders Monkeys" description="Page d'Inscription"></Seo>

      <Layout isDisplayBreadcrumbs={true} sessionStatus={GUEST}>
        <RegisterContainer />
      </Layout>
    </>

  );
}
```

Il utilise le composant de mise en page (Layout) pour afficher le contenu. La page utilise également un composat de référencement (Seo) pour améliorer le référencement de la page sur les moteurs de recherche. Le composant principal utilisé dans cette page est 'RegisterContainer', qui gère le processus d'inscription de l'utilisateur.

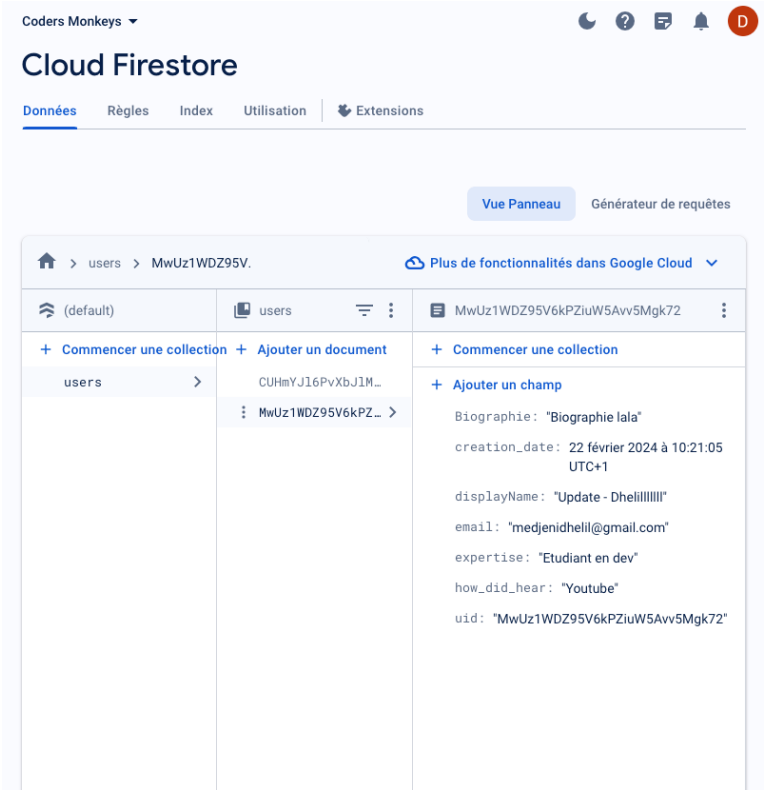
Il n'est pas nécessaire de présenter le code pour les autres fonctionnalités telles que la connexion et la réinitialisation du mot de passe, car la logique et la structure sous-jacentes restent largement similaires à celles de la page d'inscription. Bien que les composants utilisés, tels que le container, la typographie, les liens et les images, ainsi que le formulaire, soient également appliqués, ce qui diffère est simplement la mise en place des composants dans ces contextes spécifiques. De plus, la logique contenue dans les containers pour la connexion et la réinitialisation du mot de passe est propre à ces fonctionnalités, mais elle suit généralement les mêmes principes de conception.

Voici à quoi ressemble la table users, dans FireStore :



Sur la gauche, nous pouvons voir les différents éléments rentrés par l'utilisateur, notamment la date de création du compte, le mail, comment l'utilisateur a connu le site et son id.

Ceux ayant remplis le onboarding possèdent plus d'informations, comme le montre le screen ci-dessous :



Pour le troisième SPRINT, les étapes consistaient à :

- Créer la page d'enregistrement d'utilisateur.
- Optimiser les formulaires avec React Hook Form et Next.js.
- Maîtriser l'authentification avec Firebase pour créer et authentifier des utilisateurs dans l'application.
- Mettre en place les fonctionnalités de connexion, déconnexion et récupération de mot de passe.
- S'initier à Firestore pour comprendre la base de données NoSQL de Firebase.
- Approfondir le contrôle de l'authentification.

Enfin, le dernier et quatrième SPRINT comprend les étapes suivantes :

- Maîtrise de l'onboarding de l'application avec des composants dynamiques et une navigation fluide.
- Utilisation du pouvoir de l'onboarding pour transmettre des informations pour le Saas et collecter des données clés.
- Maîtrise de l'art du téléchargement d'images avec Firebase Storage.
- Surprendre les utilisateurs dès leur inscription pour une expérience utilisateur inoubliable.
- Développement du profil utilisateur de l'application avec React, Next.js et Firebase.

3.2.4.2.3 Difficultés rencontrées en général (maquette/code)

Dans l'ensemble, la transition des maquettes déjà réalisées dans le cadre de la formation, intégrant un design system professionnel, vers le code n'a pas posé de difficultés majeures. Cette expérience m'a permis de découvrir une utilisation professionnelle de Figma et de visualiser concrètement la mise en œuvre d'un design system dans un projet réel.

Cependant, les véritables défis ont émergé lors de la phase de développement de l'application. Même si les fonctionnalités à développer semblaient simples à première vue, il a fallu se concentrer pour bien comprendre les différents concepts, étant donné que je débutais sur ce framework. Malgré la formation suivie, certains aspects nécessitaient une compréhension approfondie, notamment en revisionnant plusieurs fois les mêmes vidéos, ou en cherchant sur internet, par exemple dans la mise en œuvre de fonctionnalités clés telles que l'ajout, l'édition ou la suppression d'éléments.

Dans ce contexte, j'ai dû maintenir une concentration maximale lors de la formation et consacrer plus de temps que prévu initialement. Cependant, cette expérience m'a été extrêmement enrichissante, me permettant de découvrir le fonctionnement et l'utilisation approfondie de ce framework. Cette phase de développement a été une réelle opportunité pour développer de nouvelles compétences en développement web.

4 Bilan du sujet traité

La réalisation de ces deux projets durant mon stage a été une expérience enrichissante. Travailler sur une application développée avec Symfony m'a permis de consolider mes connaissances sur ce framework et de découvrir de nouvelles astuces pour optimiser la performance et la maintenance des applications. D'autre part, la création d'une application en React m'a offert l'opportunité d'explorer un nouvel écosystème technologique et d'approfondir mes compétences en développement front-end.

Ces projets m'ont également donné l'occasion d'appliquer les bonnes pratiques de développement, de résoudre des problèmes concrets et de renforcer ma capacité à travailler de manière autonome en cherchant les réponses à mes questions par moi-même.

Globalement, cette expérience m'a permis de progresser dans ma carrière de développeur et de me sentir plus confiant dans mes compétences.

5 Bilan du stage

En conclusion, ce stage a été une expérience formidable qui m'a offert de précieuses opportunités d'apprentissage et de croissance professionnelle. Travailler sur des projets avec des technologies telles que HTML, CSS, JS, Symfony, React, Next.js, Tailwind, Firebase et Figma m'a permis de consolider mes compétences et d'explorer de nouveaux domaines passionnants du développement web.

La responsabilité de travailler en autonomie m'a donné un aperçu précieux du quotidien d'un développeur en entreprise ou en freelance. De plus, le fait de travailler en adoptant des méthodes utilisées dans le monde du travail, telles que la méthode SCRUM, a renforcé ma capacité à résoudre des problèmes de manière indépendante et à respecter des échéanciers fixés.
L'importance de rester à jour avec les dernières technologies, en particulier dans un domaine en constante évolution comme React, est devenue évidente. J'ai développé une habitude de veille technologique régulière pour rester informé des tendances et des innovations.

Ce stage a solidifié ma décision de poursuivre une carrière en tant que développeur fullstack. Les défis rencontrés tout au long du stage ont nourri ma passion pour la programmation et m'ont donné une vision claire de mes objectifs professionnels futurs.
Je suis reconnaissant pour cette expérience qui a été un jalon important dans mon parcours professionnel, me propulsant avec confiance vers de nouvelles opportunités et défis passionnants dans le domaine du développement web.