Rapport de mission de troisième année

Coding Factory

Vahé Krikorian

2023-2024

Tuteur entreprise :
Massinissa Mohellebi
massinissa.mohellebi@cloud4dev.fr

Remerciements

Je tiens à exprimer ma profonde gratitude à M. Nabil Debbou et M. Xavier Raby pour m'avoir accueilli durant cette seconde année chez CloudForDev, mais également pour m'avoir offert l'opportunité de poursuivre cette expérience enrichissante après une première année déjà très formatrice.

Je remercie particulièrement mon tuteur M. Massinissa Mohellebi ainsi que M. Sid-Ali Djabella et M. Anis Abaza pour leur soutien, leur disponibilité, et les précieux conseils qu'ils m'ont prodigués tout au long de mon parcours.

Merci à toute l'équipe de CloudForDev pour leur confiance et leur accompagnement, et pour m'avoir intégré dans cette grande famille.

Table des matières

1. Introduction

2. Présentation de l'entreprise
   - La société présentée par elle-même
   - Comment a été créée l'entreprise
   - Les services offerts par l'entreprise
   - Cadre de travail

3. Missions réalisées

   Les challenges
   - Méthodes de travail
   - Gestion du temps

   Création d'une maquette pour la refonte graphique
   - Objectifs
   - Déroulement de la mission
   - Difficultés rencontrées
   - Les résultats

   Recréation du site vitrine et du générateur de site web
   - Objectifs
   - Technologie utilisée
   - Déroulement de la mission
   - Difficultés rencontrées
   - Les résultats

   Déploiement du site vitrine et du générateur de site web
   - Objectifs
   - Technologie utilisée
   - Processus de déploiement
   - Difficultés rencontrées
   - Les résultats

4. Conclusion

5. Bibliographies

Tables des figures

1. Illustration d'un sprint Jira en cours 9
2. Illustration d'un ticket en cours Jira 10
3. Illustration d'une merge request GitLab 11
4. Plusieurs formats de maquettes de la page Home 17
5. Schéma des technologies du générateur de site web 20
6. Schéma processus de création d'un composant 23
7. Illustration du composant bouton dans la maquette 24
8. Illustration du composant "Button" dans Strapi 29
9. Illustration page JoinUs dans Strapi qui contient plusieurs composants 30
10. Illustration de l'interface de Kubernetes de Cloud4dev 33
11. Interface de docker hub avec l'image du site web et l'image de Strapi 36

1. Introduction

Durant ma deuxième année d'apprentissage au sein de l'entreprise CloudForDev, j'ai eu l'opportunité de vivre une expérience encore plus enrichissante et déterminante dans mon parcours vers le métier de Développeur Web. Cette année, j'ai non seulement continué à évoluer dans l'ambiance dynamique et motivante de l'équipe, mais j'ai également constaté un renforcement de la culture d'innovation et de cohésion au sein de l'entreprise. CloudForDev a poursuivi son engagement à écouter les besoins de ses employés en organisant davantage de séminaires de cohésion d'équipe et de moments de réflexion, consolidant ainsi la vision collective du futur de l'entreprise.

Mon apprentissage s'est concentré sur un projet principal : la refonte graphique et technique du site vitrine et du générateur de site web de CloudForDev. En travaillant principalement de manière autonome, j'ai pu développer mes compétences en gestion de projet, en création de maquettes, en développement de nouveaux composants et en déploiement de solutions web. Ce projet majeur m'a permis de renforcer mon autonomie et ma capacité à mener à bien des projets complexes tout en apportant une réelle valeur ajoutée à l'entreprise.

2. Présentation de CloudForDev

La société présentée par elle-même

Cloud4dev c'est avant tout une histoire de collègues... Andji, Cao, Xavier et Nabil, un quatuor complémentaire animé par le goût d'entreprendre, conscient de sa capacité d'innovation et guidé par sa créativité. « Nous avons eu envie de penser et concevoir ensemble notre propre idée du Cloud à travers des solutions Microsoft.
À l'image d'un nuage, le Cloud est pour nous un état d'esprit qui rassemble des idées, des compétences, et crée un niveau d'expertise précis et agile. Notre projet se construit essentiellement à travers notre équipe, conformément aux valeurs humaines que nous partageons, ce qui nous permet d'être toujours plus proches de nos clients, et d'opérer des transformations digitales toujours plus performantes. »

La création de CloudForDev

CloudForDev est une entreprise fondée il y a environ 6 ans par Andji Sacarabany, Cao Truong Hoang, Nabil Debbou et Xavier Raby.
À l'origine, le siège social de l'entreprise était à Puteaux, dans les Hauts-de-Seine, mais afin de s'agrandir et de recruter plus d'employés, tout en améliorant la qualité de vie au travail, CloudForDev a déménagé dans de nouveaux bureaux. Ceux-ci sont situés à Joinville-le-Pont, dans le Val-de-Marne, depuis décembre 2022.

Les services offerts par l'entreprise

CloudForDev se spécialise dans les services de cloud computing et de développement. Leur plateforme propose des solutions souples et évolutives pour optimiser l'infrastructure informatique des entreprises, couvrant ainsi plusieurs domaines tels que le design et le développement, le Saas, le DevOps, le cloud computing, l'infrastructure et la sécurité des réseaux. En offrant des services de stockage, de traitement et de gestion

des données, ainsi que des outils de développement et de déploiement, les développeurs répondent aux besoins de toutes les entreprises. Leur approche met l'accent sur la sécurité et la performance, permettant à leurs clients, tels que Sonepar, Engie et Reed Expositions, d'améliorer leur agilité, leur efficacité et leur compétitivité sur les marchés.

Le cadre de travail

Les bureaux de Cloud4Dev sont situés dans le Val-de-Marne, à Joinville-le-Pont, dans un bâtiment accueillant plusieurs petites entreprises. L'emplacement est plutôt avantageux, étant juste en face de la gare, ce qui rend les bureaux bien desservis. L'équipe se compose d'une dizaine de personnes. La majorité travaille sur des projets pour des clients tels que Engie, Sonepar, Human4Help et Fraktion. L'autre partie de l'équipe, au sein de laquelle je travaille, se consacre aux projets internes comme le site vitrine et le déploiement de nos services.

Depuis la période de Covid-19, une charte de télétravail a été mise en place, applicable à tous les employés, qu'ils travaillent en alternance, à temps plein ou à temps partiel. Pour ceux qui se déplacent chez les clients, une journée de présence toutes les deux semaines est imposée, selon les modalités choisies par chaque client. Pour le reste de l'équipe, qui travaille sur les projets internes, deux jours de télétravail par semaine sont autorisés.

Afin de se recentrer sur les valeurs fortes de Cloud4Dev et de resserrer les liens en tant qu'équipe, l'entreprise à organisé un séminaire de cohésion d'équipe qui s'est déroulé à Malte du jeudi 21 septembre 2023 au dimanche 24 septembre 2023. Nous avons eu l'occasion d'apprécier de nombreuses activités, telles qu'une visite guidée en quad sur l'île de Gozo, afin d'apprendre à mieux nous connaître. Une session de réflexion sur le futur de l'entreprise a été complétée par plusieurs discussions lors des repas, que nous prenions dans des restaurants traditionnels, et lors du tour de l'île en bus touristique. Les moments de temps libre ont permis de réfléchir tout en profitant de la piscine, de la plage.

Pour conclure, ce séminaire à permis à l'équipe de Cloud4Dev de mieux se connaître, de renforcer les liens existants et d'en créer de nouveaux. J'ai pu découvrir Malte, ses attraits, et notamment ses paysages magnifiques et sa population accueillante.

Ce fut, pour moi, une merveilleuse expérience et un honneur de pouvoir participer à un séminaire de ce type, offert par l'entreprise.

3. Missions réalisées

Les challenges

- **Méthodes de travail**

Les méthodes de travail utilisées par Cloud4Dev n'ont pas évolué depuis l'année dernière, elles répondent correctement aux besoins de l'entreprise en matière de gestion de projet. Les outils utilisés permettent d'optimiser la productivité et d'assurer la qualité des livrables. La transparence, l'agilité et l'efficacité qu'ils apportent sont en parfaite adéquation avec la culture d'entreprise et leur objectif de développement.

CloudForDev, utilise la méthode de travail Scrum, qui est un cadre de gestion de projet agile, qui favorise la collaboration, l'agilité et la transparence. Il est basé sur des "Sprints" d'une durée définie. Un Sprint est une courte période de temps (deux semaines environ chez CloudForDev), pendant laquelle une équipe Scrum travaille pour terminer une quantité de travail définie. Cette approche permet à CloudForDev de livrer des produits de qualité tout en s'adaptant efficacement aux besoins changeants des clients.

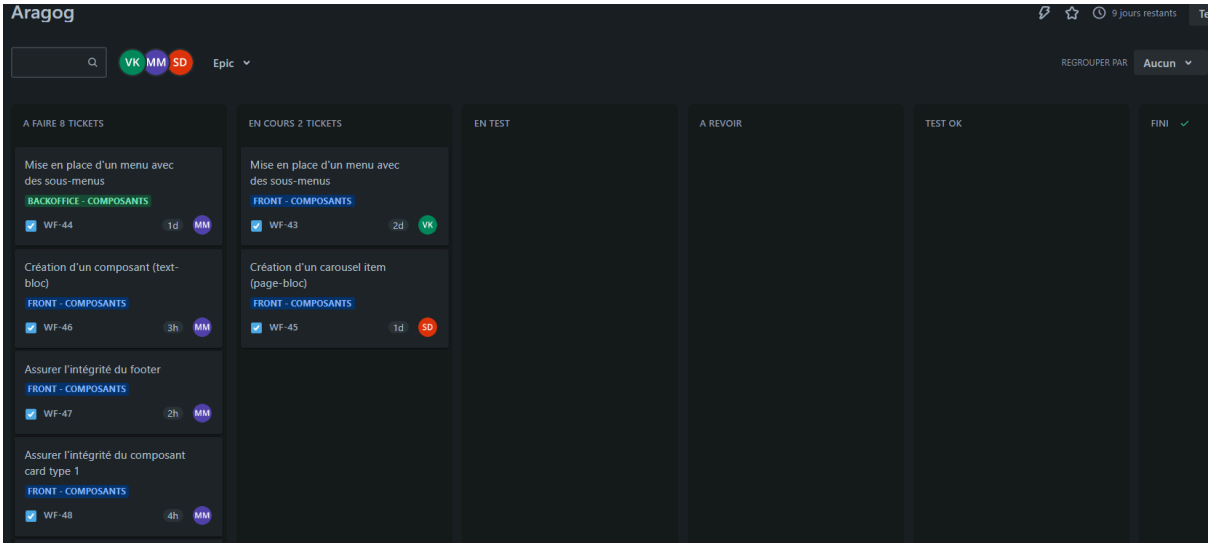
Afin d'organiser au mieux les Sprints, nous utilisons la plateforme de gestion de projets Jira. Jira nous permet de créer et d'organiser des tâches en leur attribuant des estimations de temps, et de répartir ces tâches entre les membres de l'équipe afin d'optimiser l'avancement du

Sprint. Grâce à Jira, nous pouvons également assurer un suivi précis du Sprint et conserver une visibilité sur sa progression.

Tous les projets de développement au sein de CloudForDev utilisent l'outil de *gestionning* de fichier Git, et GitLab pour le contrôle de version et le stockage centralisé du code source. Cela nous permet de travailler en équipe sur le même projet, de gérer les différentes versions du code et de fusionner les modifications en toute sécurité.

Comme expliqué précédemment, nous utilisons plusieurs outils afin de simplifier, organiser et partager le travail de développement au sein d'un projet.

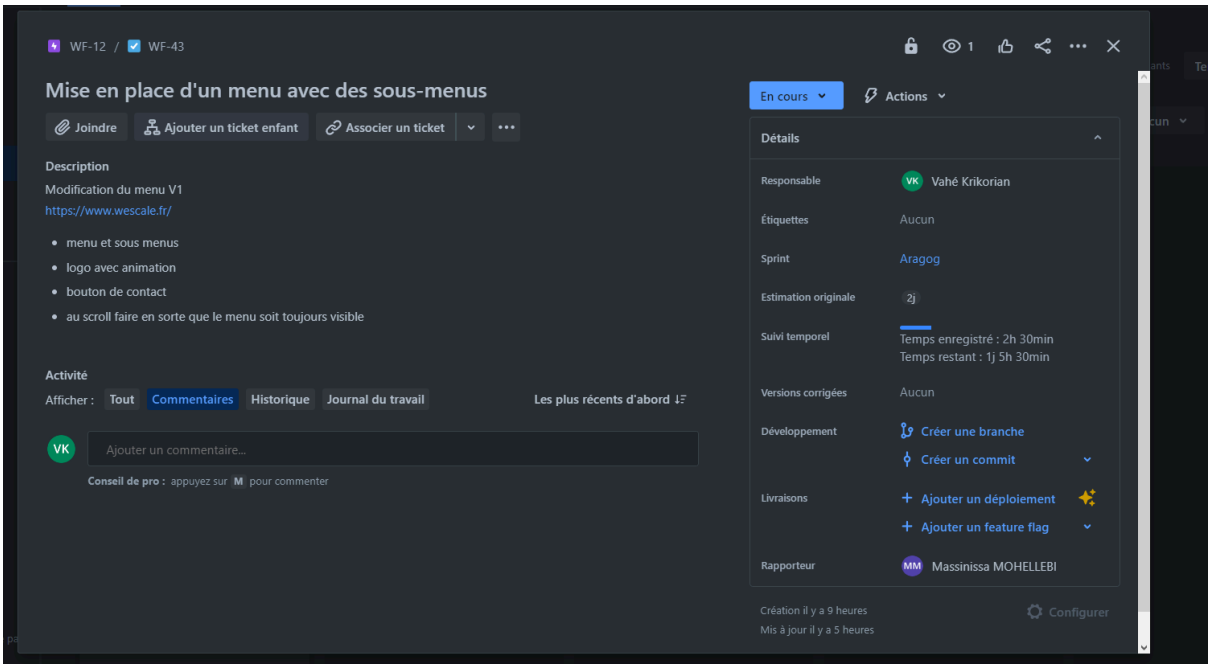
Voici par exemple un Sprint en cour :



En haut de la page, on peut voir les informations de ce Sprint : le nom, la durée, et les membres. En dessous, nous trouvons six colonnes, qui servent à ranger les différents tickets afin de voir très simplement où en est l'avancement de chacun des tickets. On peut ainsi rapidement évaluer l'état d'avancement du Sprint.
C'est dans la colonne de gauche que tous les tickets sont créés par le rapporteur du Sprint, qui y précise le titre, la description, le numéro du ticket, ainsi que l'estimation du temps nécessaire pour réaliser ce ticket. Quand un ticket à été commencé, il est déplacé dans la colonne "en cours". Une fois terminé, il est placé dans la colonne "en test", qui permet au rapporteur de le tester. Si le ticket comporte des bugs, ou s'il ne

correspond pas à l'attente du rapporteur, celui-ci va le déplacer dans la colonne "à revoir". En revanche, si le ticket correspond à l'attente du rapporteur, et qu'il n'y décèle aucun bug, il sera déplacé dans la colonne "test Ok" ; avant, finalement, d'achever son parcours dans la colonne "fini".

Voici par exemple un ticket du Sprint :

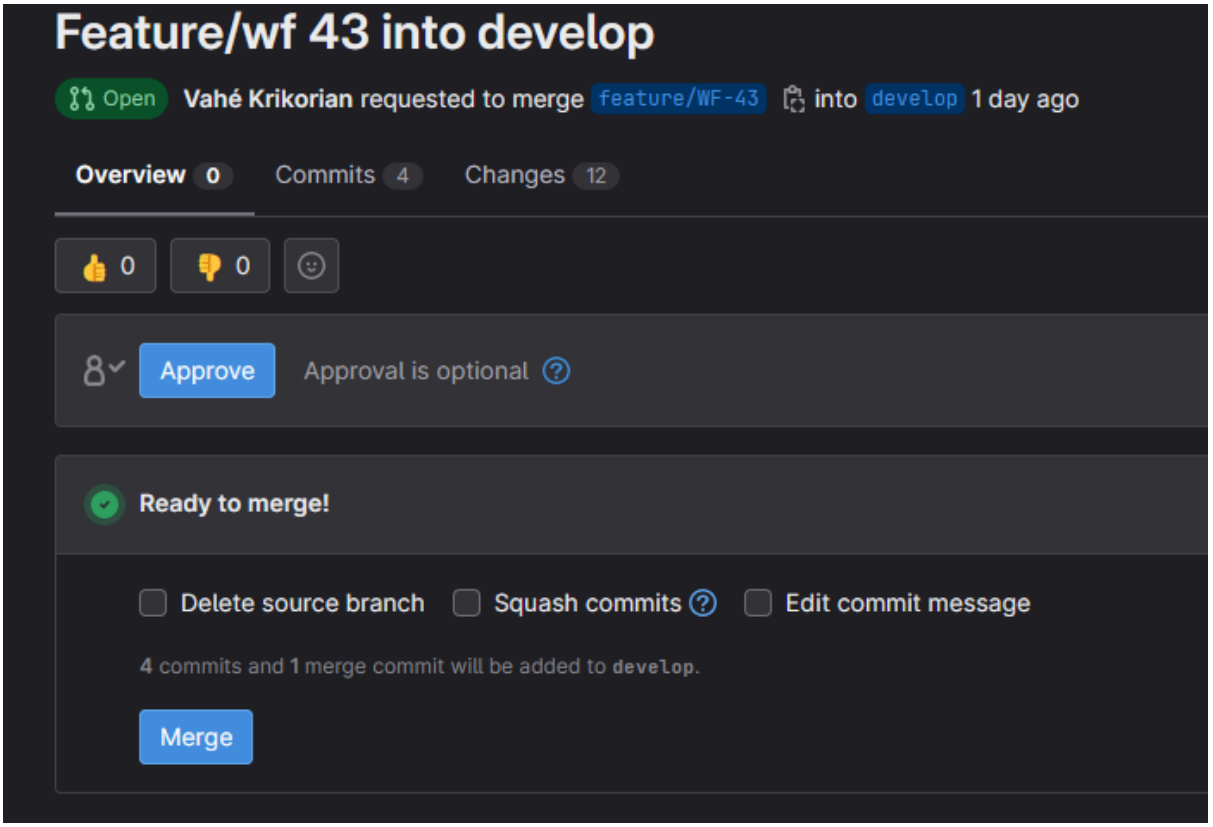


Dans ce ticket, nous voyons deux parties qui recèlent de nombreuses informations importantes. À gauche, on peut retrouver le numéro du ticket, le titre et la description de celui-ci. À droite se trouvent tous les détails du ticket, tels que son affectation, le sprint auquel il appartient, l'estimation de temps faite lors de sa création, et le suivi temporel, qui consiste à relever le temps que l'on a réellement passé sur le ticket en question.

Chaque projet dispose de son propre repository GitLab, qui peut être comparé à un dossier dans lequel le code source est stocké. Un repository GitLab permet de centraliser et de gérer les différentes versions du code d'un projet, ainsi que les modifications apportées par les développeurs du projet au fil du temps.

Dans chaque repository, il y a plusieurs branches, c'est en quelque sorte comme créer une "copie" du projet pour développer, réaliser le contenu sans interférer avec le code principal. La branche principale est appelée "Main", elle contient le code d'origine du projet. Lorsqu'un ticket est créé sur Jira, le développeur assigné à ce ticket doit créer une branche distincte portant le numéro du ticket ; par exemple, sur l'image ci-dessus, le numéro de ticket est "WF-43", donc le nom de la branche sera "WF-43".

Une fois que le développeur a terminé de travailler sur son ticket, il effectue une demande de fusion, appelée "merge request", ce qui signifie que le développeur va effectuer une demande au rapporteur du projet pour intégrer les modifications de sa branche vers la branche "develop". Voici un exemple de la manière dont se présente une merge request :



La branche "develop" est l'endroit où tous les codes issus des différentes branches de ticket sont rassemblés. Cela permet de tester l'ensemble du

code développé par les différentes personnes, avant de le transférer vers la branche principale ("Main").

L'étude de ces outils a été un défi pour moi. Certains d'entre eux étaient complètement nouveaux de mon point de vue, et il m'a fallu du temps pour les apprendre, et pour m'adapter à leur fonctionnement. Mais, au-delà de l'apprentissage, leur utilisation quotidienne au sein des projets peut également prendre du temps. Par exemple, sur Jira, il faut relever le temps passé sur chaque tâche, et s'engager dans les pratiques de la méthodologie Scrum, comme les réunions quotidiennes, pour suivre l'avancement du projet.

Cependant, l'utilisation de la méthodologie Scrum, Git/GitLab et Jira dans les projets au sein de CloudForDev est bénéfique. Ces outils favorisent une collaboration transparente et une gestion efficace du travail, contribuant ainsi au succès des différents projets. Malgré les efforts initiaux nécessaires pour s'adapter à ces outils, ceux-ci permettent de gagner un temps précieux dans l'organisation du projet, et offrent un meilleur suivi global.

- **Gestion du temps**

La bonne gestion du temps au sein d'un projet est primordiale pour le mener à bien, et pour respecter le délai imposé par les Sprints. La durée de ces derniers est définie en fonction des tâches à effectuer. Cependant, il s'agit d'un véritable défi, car il est nécessaire de savoir s'organiser efficacement au sein de chaque tâche.
D'ailleurs, la gestion du temps ne se limite pas aux tâches spécifiques du Sprint. En tant que développeur, il est essentiel de gérer son emploi du temps personnel pour être productif et efficace. Des stratégies telles que la gestion des priorités, l'organisation du travail par blocs de temps, et la mise en place de plages horaires dédiées à la concentration sans interruption peuvent grandement contribuer à optimiser son efficacité.

Création d'une maquette pour la refonte graphique

- **Objectifs**

L'ancien design du site vitrine et du générateur de site web ne correspondait plus à l'image moderne et innovante que CloudForDev souhaite renvoyer. De plus, il ne répondait plus aux standards actuels en matière de design web, en 2024. L'objectif principal de cette mission était donc de moderniser l'apparence générale du site vitrine et des composants du générateur de site web, ainsi que d'améliorer l'expérience utilisateur et l'interface visuelle.

- **Déroulement de la mission**

Cette mission a débuté par une phase de recherche intensive. J'ai d'abord examiné en profondeur ce que les concurrents de Cloud4Dev proposaient, évaluant le niveau de sophistication de leurs sites vitrine en termes de design UI et d'expérience utilisateur. Cette étape m'a permis de définir clairement les standards auxquels la maquette de Cloud4Dev devrait se conformer. Après avoir constaté que les sites des concurrents étaient souvent assez simples et peu innovants sur le plan du design, j'ai exploré le site Awwwards (1), une plateforme référençant les sites internet les plus remarquables en termes de design et d'expérience utilisateur. Cette démarche m'a permis de découvrir deux extrêmes : d'un côté, des sites basiques avec des composants simples, et, de l'autre, des sites impressionnants, utilisant des animations avancées et des techniques destinées à captiver les utilisateurs. Ce travail de recherche approfondi m'a fourni les bases nécessaires pour positionner le futur site de Cloud4Dev comme un site élégant mais accessible, en m'inspirant des meilleures pratiques observées sur Awwwards afin de garantir une interaction utilisateur satisfaisante et dynamique.

À ce stade, j'avais déjà une idée précise de l'apparence souhaitée pour les composants du générateur de site web et du site vitrine. L'étape suivante consistait à choisir les outils nécessaires afin de créer la maquette, et de me former à leur utilisation. J'ai opté pour Figma, une plateforme de conception d'interfaces utilisateur et d'expérience utilisateur en ligne.

Figma est très intuitive et permet le travail collaboratif en temps réel, ce qui a été particulièrement bénéfique pour que mon collègue puisse suivre mes avancées et valider les parties de la maquette qui lui semblaient correctes, tout en identifiant celles qui nécessitaient des modifications. Accessible via un navigateur web, Figma facilite la création de maquettes interactives et de prototypes sans nécessiter l'installation de logiciels locaux, grâce à sa bibliothèque de composants prêts à l'emploi, assurant ainsi une conception efficace et cohérente.

Pour me former efficacement à l'utilisation de Figma, j'ai suivi le tutoriel "Les bases de Figma 2023 (cours gratuit)" disponible sur YouTube (2). Ce tutoriel exhaustif m'a permis de maîtriser toutes les fonctionnalités essentielles de Figma, de la création de pages et de composants à l'ajout de textes et d'images, en passant par l'alignement précis des éléments. Grâce à cette formation, j'ai acquis les aptitudes nécessaires pour concevoir efficacement des maquettes claires et bien structurées.

De plus, j'ai puisé mon inspiration dans plusieurs composants présentés dans ce tutoriel pour intégrer des éléments pertinents et modernes dans la maquette de Cloud4Dev. Cette approche m'a permis de créer une interface utilisateur qui non seulement répond aux standards de design actuels, mais qui est également fonctionnelle et intuitive pour les utilisateurs finaux.

Une fois tout le travail de recherche et de formation terminé, j'ai enfin pu commencer la création de la maquette du site vitrine de Cloud4Dev. J'ai décidé de structurer le projet Figma en trois parties, chacune correspondant à un format d'écran : Maquette Desktop, Maquette Tablette et Maquette Mobile. Chaque partie montre comment le site apparaîtra selon l'appareil utilisé par l'utilisateur, l’affichage comprenant les pages suivantes : Home, About-Us, Join Us et Contact.

Pour chaque page, j'ai utilisé des composants similaires afin de permettre une réutilisation efficace dans le générateur de site web, avec des personnalisations adaptées.

Pour ce qui est de la création du contenu de ces composants, j'ai principalement récupéré le texte de l'ancienne maquette pour l'intégrer

dans les mêmes pages, mais refaçonné avec de nouveaux composants. En revanche, pour les illustrations, j'ai utilisé principalement des images vectorielles. Contrairement aux images classiques, qui sont constituées de pixels, une image vectorielle est composée de vecteurs et de formes géométriques. Cela permet aux images de rester nettes et non pixelisées, quelle que soit leur taille. Pour trouver des illustrations adaptées aux composants et à la page, j'ai utilisé un site appelé Freepik (3), qui répertorie de nombreuses créations d'utilisateurs. On y trouve des images, des vecteurs et des photos, sous divers formats, classés par thème ou mots-clés.

Je ne détaillerai pas la création des différentes pages ici, car j'ai simplement construit des composants visuels en appliquant les techniques apprises dans le tutoriel vidéo. À l'origine, j'avais prévu un système permettant de changer de thème de couleur sur le site web, avec des thèmes "light" et "dark" représentés par un petit bouton switch situé en haut à droite du menu. Toutefois, cette fonctionnalité n'a pas été jugée indispensable pour le site vitrine de Cloud4Dev et n'a donc pas été intégrée.

- **Difficultés rencontrées**

Dans ce projet de conception de maquette, contrairement à un projet de développement classique, qui consiste essentiellement en programmation de fonctionnalités et en utilisation de code, la principale difficulté ne fut dans le cas présent, ni l'aspect technique, ni la formation sur Figma, mais plutôt le travail de recherche en design. En effet, il a fallu trouver de nombreuses sources d'inspiration, en consultant des sites concurrents, aussi bien que des maquettes existantes, ou encore des images adaptées aux composants à intégrer.

J'ai dû analyser les designs de sites de concurrents, afin de saisir les tendances actuelles et de déterminer ce qui fonctionnait bien en termes d'interface utilisateur et d'expérience utilisateur. De plus, j'ai utilisé

beaucoup de ressources en ligne, y compris des plateformes comme Awwwards, pour m'inspirer d'exemples de design existant. Cela inclut l'exploration de sites internet, mais aussi l'étude de maquettes Figma réalisées par d'autres utilisateurs, ainsi que de divers concepts visuels.

Trouver des images pertinentes et de haute qualité pour les composants a également été un défi. Il fallait s'assurer que chaque élément soit cohérent avec l'identité visuelle de Cloud4Dev, et qu'il apporte une valeur ajoutée à l'ensemble de la maquette.

- **Résultats atteints**

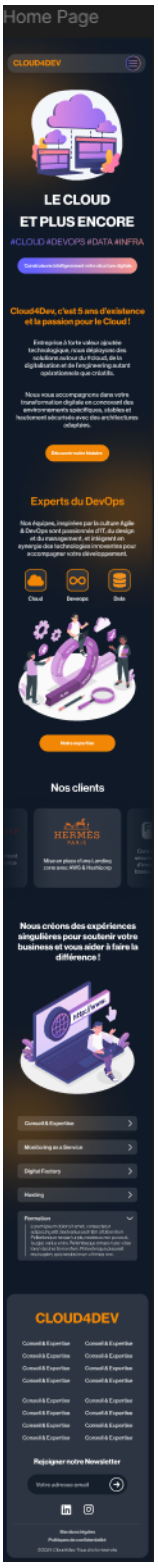
Finalement, malgré les différentes difficultés rencontrées, je suis parvenu à concevoir la maquette dans sa totalité, en incluant les pages : Home, About-Us, Join-Us et Contact. Cette maquette m'a fourni une référence précieuse, tant pour évaluer la taille des éléments de chaque composant en fonction de l'écran, que pour la mise en page globale. Elle a également été essentielle pour définir la charte graphique. En somme, cette maquette a été extrêmement utile en vue du développement des différents composants pour le générateur de site web.
Mais, outre le fait d'être réellement bénéfique pour ce projet de générateur de site web, la formation sur Figma et le travail de recherche approfondi m'ont permis d'acquérir des compétences précieuses. Désormais, je maîtrise Figma, et je l'utilise déjà lors de l'élaboration d'autres projets, que ce soit dans le cadre scolaire ou professionnel.

En page suivante, un aperçu de la maquette finale de la page “Home”, en trois formats :

Ordinateur

Tablette

Mobile



Recréation du site vitrine et du générateur de site web

- **Objectifs**

La mission qui m'a initialement été confiée était la création du site vitrine de Cloud4Dev. Pour répondre aux ambitions novatrices de l'entreprise, cette mission s'est transformée en la création d'un générateur de site web.

Le projet a démarré en décembre 2022, et j'y ai participé avec un collègue. Les anciens sites vitrine et générateur de site web ne correspondaient plus à l'image moderne et innovante que CloudForDev souhaite projeter. De plus, ils ne répondaient plus aux standards en cours en matière de design web.

En 2024, nous avons entrepris de relancer ce projet en utilisant des technologies et outils plus récents, et en procédant à une refonte graphique. En 2022, j'avais travaillé avec un collègue. Cette fois,j'ai réalisé la nouvelle version seul, la plupart du temps, prenant ainsi en charge l'intégralité du projet.

Grâce à cette mise à jour, nous avons pu intégrer de nouvelles fonctionnalités et améliorer les performances du générateur. Nous avons également travaillé sur la modernisation de l'interface utilisateur des composants du générateur, afin qu'elle soit plus en phase avec l'identité visuelle de Cloud4Dev. Le site vitrine a été conçu en utilisant ce générateur.

Les objectifs spécifiques du générateur de site web incluent également la personnalisation via une bibliothèque de composants réutilisables, la facilitation de la gestion du contenu, la garantie d'une expérience utilisateur cohérente, et l'optimisation des performances du site web.

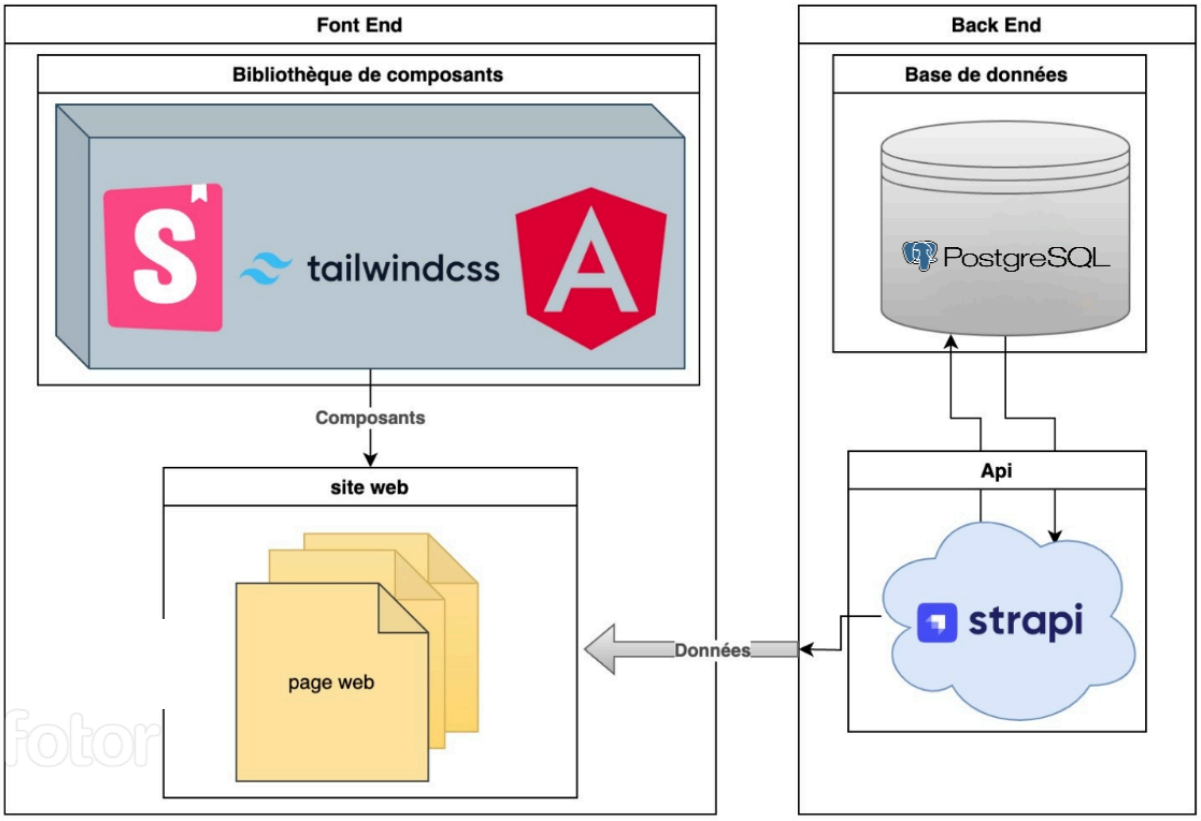
- **Technologie utilisée**

Dans le cadre du développement du site vitrine et du générateur de sites web, la phase initiale du projet a été caractérisée par l'utilisation de technologies clés, telles que MySQL pour la base de données, et Strapi version 3 pour la gestion des données côté back-end, via une API. Côté front-end, nous avons choisi Angular version 15, un framework JavaScript robuste, pour construire la bibliothèque de composants. Pour gérer les styles, l'approche adoptée reposait sur Tailwind CSS.

En 2024, dans le cadre de la refonte du projet, nous avons migré vers des technologies plus récentes et performantes. Angular version 17 a notamment remplacé la version 15, offrant ainsi une expérience de développement plus fluide, et une meilleure prise en main, grâce à ses nouvelles fonctionnalités et améliorations en termes d'ergonomie. De même, nous avons mis à niveau Strapi vers sa version 4, et avons ainsi bénéficié de performances accrues, d'une sécurité renforcée, et d'une interface plus conviviale de la gestion de contenu. En parallèle, nous avons migré notre base de données depuis MySQL vers PostgreSQL, ce qui nous permet de bénéficier d'une meilleure gestion des données, et d'une optimisation des requêtes.

Ces évolutions technologiques ont non seulement amélioré la robustesse et les performances du générateur de sites web de Cloud4Dev, mais elles ont également renforcé sa capacité à répondre aux exigences modernes en matière de développement web.

En page suivante, un aperçu du schéma des technologies utilisées :



Procédons maintenant à une description détaillée des technologies utilisées dans le cadre du générateur de site web, en commençant par les technologies front-end, pour finir par celles qui concernent le back-end.

Technologies utilisées pour le front-end :

Le **front-end** d'une application web est la partie avec laquelle l'utilisateur interagit directement. C'est donc ce que vous voyez, et ce avec quoi vous interagissez lorsque vous visitez un site web, ou utilisez une application. Dans notre projet, nous utilisons Angular en TypeScript pour développer le front-end. C'est là que les composants de l'interface utilisateur sont construits et où les styles sont appliqués, en utilisant Tailwind CSS. L'objectif est de forger et proposer une expérience utilisateur intuitive et agréable.

Angular est un framework JavaScript populaire pour le développement d'applications web. Il a été développé par Google et est utilisé par de nombreuses entreprises et développeurs à travers le monde. Il est basé sur une architecture articulant des composants qui permettent de construire des applications web dynamiques et réactives. Il utilise le langage TypeScript, qui est une surcouche de JavaScript, offrant des fonctionnalités supplémentaires telles que le typage statique et les fonctionnalités orientées objet. TypeScript permet de détecter les erreurs de programmation plus facilement et d'améliorer la productivité du développement.

Tailwind CSS est un framework CSS utilitaire, qui permet de créer rapidement des interfaces utilisateur modernes et réactives. Contrairement à d'autres frameworks CSS comme Bootstrap ou Foundation, Tailwind CSS se concentre sur les classes utilitaires plutôt que sur les composants prédéfinis.

Technologies utilisées pour le back-end :

Le **back-end**, quant à lui, est la partie invisible de l'application, qui fonctionne en arrière-plan. Il gère les données, traite les requêtes de l'utilisateur et assure la logique métier de l'application. Dans notre cas, nous utilisons Strapi, avec une base de données PostgreSQL, pour gérer le contenu du site web. Strapi fournit une interface d'administration et une API qui permettent au front-end d'accéder aux données et de les manipuler de manière sécurisée. Il y a ainsi une séparation claire des responsabilités entre le front-end, qui se concentre sur l'interface utilisateur, et le back-end, qui gère la logique de données et le traitement des requêtes.

Strapi est un système de gestion de contenu (CMS) headless, ce qui signifie qu'il sépare le front-end du back-end au sein de votre application ou de votre site web. Plus spécifiquement, Strapi est une plateforme open source qui permet aux développeurs de créer et de gérer facilement des API (Application Programming Interfaces), afin de fournir du contenu à

diverses interfaces utilisateur. Dans notre projet, nous utilisons Strapi en tant qu'API de gestion du contenu. L'API fournit les noms des pages, les noms de routes associées, les composants spécifiques à inclure dans chaque page, ainsi que les valeurs des propriétés des composants. Ces données nous permettent d'organiser et de créer dynamiquement les pages du site web, d'associer les routes appropriées, de sélectionner les composants nécessaires, et de personnaliser leurs propriétés en fonction des préférences définies via l'API.

PostgreSQL est un système de gestion de base de données relationnelle (SGBDR), open-source, et réputé pour sa robustesse, sa fiabilité et ses performances élevées. Contrairement à MySQL, PostgreSQL est particulièrement apprécié pour sa conformité aux standards SQL avancés, sa gestion avancée des transactions, et ses capacités de traitement des données complexes.

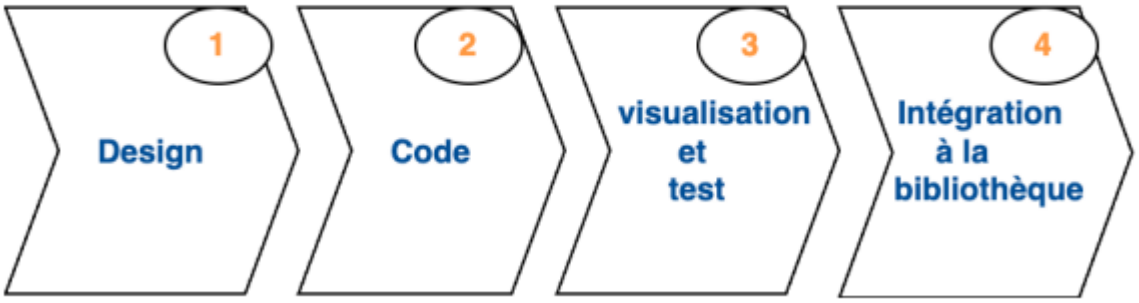
- **Déroulement de la mission**

Tout d'abord, nous avons créé une bibliothèque de composants réutilisables en utilisant Angular et Tailwind CSS. Une bibliothèque de composants réutilisables est un ensemble de modules ou d'éléments de conception pré-construits, prêts à être utilisés lors du développement de sites web, d'applications ou d'autres interfaces utilisateur. Ces composants sont conçus pour être flexibles, modulaires et personnalisables, ce qui permet aux concepteurs et aux développeurs de les utiliser de manière efficace.

Cette phase de développement s'est appuyée sur la maquette précédemment conçue. En examinant la maquette, nous avons pu identifier des composants similaires ou identiques, pouvant être utilisés à plusieurs endroits. Par exemple, deux composants distincts sur la maquette peuvent en réalité être fusionnés en un seul composant, avec des variables ajustables pour s'adapter à plusieurs utilisations. Cela permet de rationaliser le développement et d'assurer une cohérence visuelle et fonctionnelle sur tout le site.

La bibliothèque de composants développée pour ce projet offre une collection de composants réutilisables qui peuvent être combinés et personnalisés pour créer des sites web uniques. Chaque composant est conçu pour être facilement personnalisable. Les styles sont définis à l'aide de Tailwind CSS, qui offre une large gamme de classes prédéfinies pour modifier rapidement l'apparence des composants. De plus, les composants sont paramétrables, ce qui signifie que les utilisateurs peuvent ajuster les propriétés telles que la couleur, la taille, la disposition, etc., afin de répondre à leurs besoins spécifiques.

Processus de construction d'un composant



Pour construire un composant, il y a quatre étapes clés :

1. **Le design** : lors de cette étape, on définit le concept du composant ainsi que son apparence. Cela implique la création d'un design visuel, la spécification des fonctionnalités et l'identification des interactions utilisateur. La maquette sert de guide pour définir ces aspects et assurer une uniformité visuelle.
2. **Le code** : cette étape consiste à traduire le design en code, et à créer le comportement souhaité qui régira le composant. Les développeurs utilisent Angular pour structurer le code et intégrer les fonctionnalités nécessaires.
3. **La visualisation et le test du composant** : après avoir écrit le code, il est important de visualiser et de tester le composant, pour s'assurer qu'il fonctionne bien. Nous testons chaque composant indépendamment, constatant ainsi leur fiabilité et leur conformité aux spécifications de la maquette.

4. **L'intégration à la bibliothèque de composants** : une fois que le composant est terminé, nous l'intégrons à la bibliothèque de composants. Il peut donc être utilisé au sein d'autres composants plus complexes, ou directement dans des pages du site, par d'autres développeurs.

Ce processus garantit que les composants sont bien conçus, fonctionnels, et prêts à être utilisés au sein de différentes applications web. Il facilite également la réutilisation des composants, ce qui permet d'accélérer le processus de développement et de maintenir une cohérence dans l'apparence et les fonctionnalités des sites web.

Voici un exemple du composant "bouton" dans la maquette :

Comme le montre le design du composant bouton sur la maquette, il est essentiel de comprendre ses différentes utilisations et variations. Ce composant est utilisé à plusieurs reprises, que ce soit dans d'autres composants ou sur les pages en général. En analysant ces utilisations, nous pouvons en déduire qu'il faudra rendre ce composant dynamique et flexible en fonction des différents types de boutons présents sur la maquette.

Cependant, il ne faut pas uniquement se fier à la maquette. Il est crucial que le bouton soit personnalisable pour qu'un futur client de Cloud4Dev puisse le modifier selon ses besoins spécifiques ou en fonction de futures versions de la maquette. Cela doit être fait sans introduire une multitude de possibilités, car cela rendrait la personnalisation compliquée et nuirait à une expérience utilisateur fluide. L'objectif est de trouver un équilibre, en offrant suffisamment d'options de personnalisation pour répondre aux besoins variés, tout en maintenant une interface simple et intuitive pour le générateur de sites web. Cela permettra de garantir que les utilisateurs peuvent facilement adapter les boutons à leurs besoins spécifiques sans se sentir submergés par trop de choix.

Une fois que le design du composant est établi, nous intégrons ce composant dans le code avec angular.

Nous allons écrire le code au sein de quatre fichiers :

- **button.component.html**
- **button.component.ts**
- **button.component.css**
- **button.component.spec.ts**

Tout d'abord, dans le fichier button.component.html sont définis la structure et le contenu du composant bouton à afficher dans le navigateur :

```
@switch (item) {
  @case ("normal") {
    <!-- Configurations et styles spécifiques pour le bouton standard
-->
  }

  @case ("menu") {
    <!-- Configurations et styles spécifiques pour le bouton du menu
-->
  }

  @case ("big") {
    <!-- Configurations et styles spécifiques pour les boutons de
grande taille -->
  }

  @case ("linear-gradient") {
    <button
    class="btn btn-1 w-auto h-[42px] px-4 sm:px-10 md:px-14
lg:px-[90px] md:h-11 rounded-full text-nowrap text-xs sm:text-sm
transition-all ease-in-out duration-300 "
    [ngStyle]="{
      'background-image': finalLinearGradient,
      'color': isActive ? hoverTextColor : textColor,
    }"
    (mouseenter)="onHoverButton()"
    (mouseleave)="onLeaveButton()"
    (click)="onClick()">
      {{ text }}
  </button>
  }
}
```

On peut y retrouver plusieurs types de boutons, comme nous l'avons vu précédemment sur la maquette. J'ai donc décidé de les séparer en quatre types différents, chacun ayant un style et une taille spécifiques, afin de

convenir aux différentes utilisations et variations de ce bouton sur la maquette.

Les types sont les suivants :

- Le bouton "**normal**" à une taille assez standard, il est notamment utilisé pour envoyer des formulaires et des appels à l'action.
- Le bouton "**menu**" est conçu spécifiquement pour le menu, avec des styles et une taille adaptés pour s'intégrer harmonieusement.
- Le bouton "**big**" est utilisé pour les actions nécessitant une plus grande visibilité, avec des tailles plus grandes pour attirer l'attention de l'utilisateur.
- Le bouton "**linear-gradient**" utilise un arrière-plan en dégradé linéaire, idéal pour des mises en page modernes et esthétiques.

Puis, dans le fichier button.component.ts, les propriétés sont définies sous forme de @Input, permettant de personnaliser et d'adapter facilement le composant à d'autres sites web.

Voici quelques exemples d'entrées :

```
@Input() backgroundColor = "#2C3541";
@Input() startColorLinearGradient = "#6763FF";
@Input() endColorLinearGradient = "#F9968B";
@Input() hoverBackgroundColor = "#F28705";
@Input() textColor = "#fff";
@Input() hoverTextColor = "#fff";
@Input() text = "Nous Contacter";
@Input() borderColor = "#F28705";
@Input() valueToEmit: string = "";
@Input() item!: string;
@Input() isButtonLinkActive = false;
@Input() link = "";
@Input() linkOpenNewWindow = false;
```

Ces entrées permettent de personnaliser le composant de plusieurs façons :

- **Personnalisation des couleurs** : les propriétés backgroundColor, startColorLinearGradient, endColorLinearGradient, hoverBackgroundColor, textColor, hoverTextColor, et borderColor permettent de définir les couleurs de fond, de texte, de dégradé linéaire, ainsi que les couleurs de survol et de bordure.
- **Texte** : la propriété text permet de définir le texte affiché sur le bouton.
- **Comportement** : les propriétés isButtonLinkActive, link, et linkOpenNewWindow permettent de déterminer si le bouton agit comme un lien, quelle URL il doit ouvrir, et s'il doit ouvrir le lien dans une nouvelle fenêtre.
- **Sélection du type de bouton** : la propriété item permet de sélectionner le type de bouton (par exemple, "normal", "menu", "big", "linear-gradient").

Ces propriétés facilitent l'adaptation du composant aux besoins spécifiques du projet, qu'il s'agisse d'ajuster l'apparence ou le comportement. Par exemple, un bouton peut être configuré pour ouvrir une nouvelle fenêtre du navigateur lorsqu'il est cliqué, ou pour simplement transmettre des informations à son composant parent lors du clic. Pour gérer ces interactions, nous utilisons les `@Output` :

```
@Output() clickEvent: EventEmitter<any> = new EventEmitter();
```

Cette déclaration crée une sortie nommée clickEvent de type EventEmitter, permettant au composant d'émettre un événement lorsqu'une action spécifique, comme un clic, se produit. La méthode onClick() est utilisée pour déclencher cet événement :

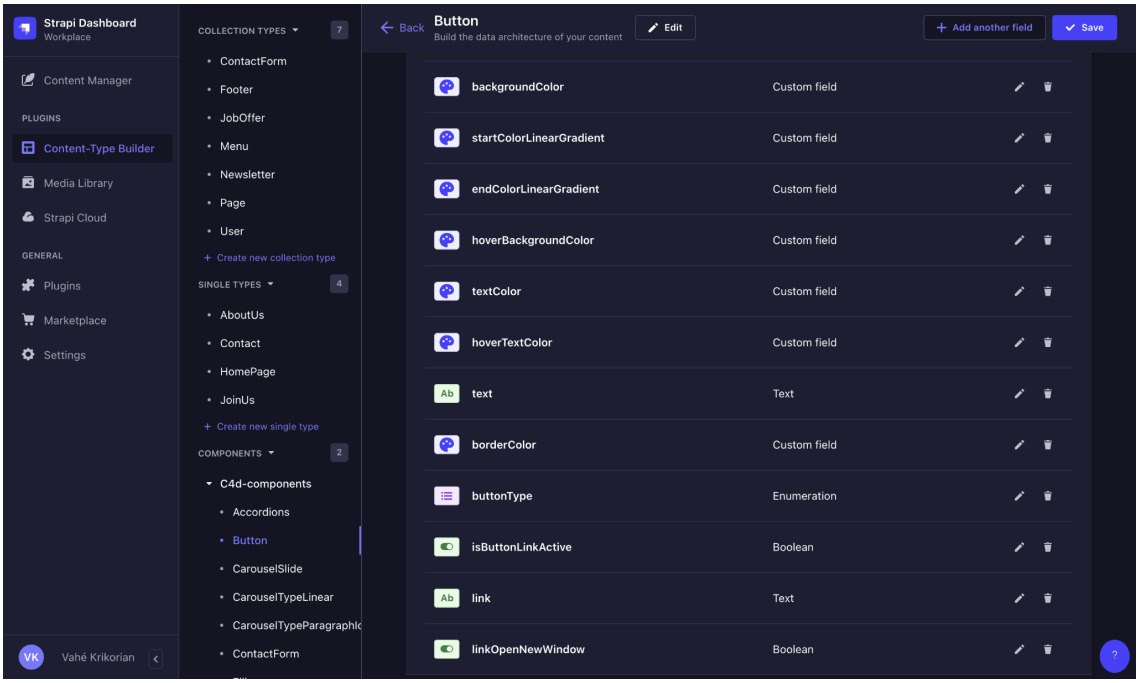
```
onClick() {
    this.isclick = !this.isclick;
    this.isActive = true;
>>> this.clickEvent.emit(this.valueToEmit);

    if (this.isButtonLinkActive && this.linkOpenNewWindow == false) {
      window.location.href = this.link;
    } else if (this.isButtonLinkActive && this.linkOpenNewWindow) {
      window.open(this.link, "_blank");
    }
  }
```

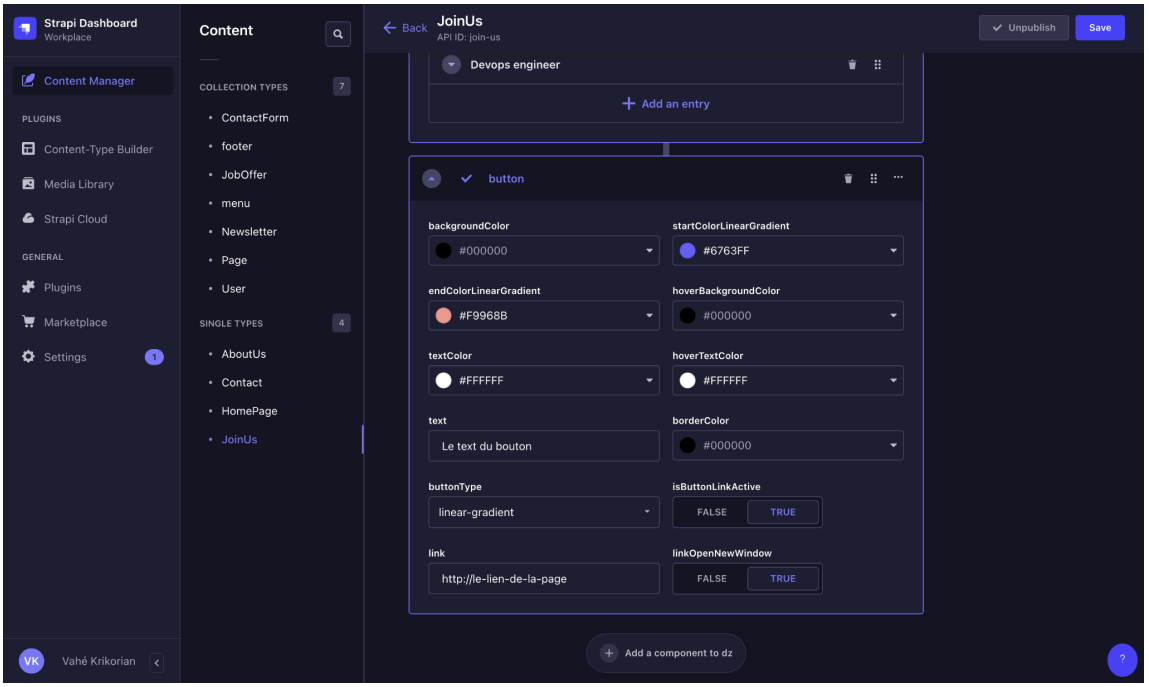
Nous pouvons d'ailleurs voir que les valeurs renseignées dans les @Input ont de l'influence. Il y a donc une logique à respecter lorsqu'on détermine le comportement du composant lors du clic.

La prochaine étape consiste à créer le composant "bouton", ainsi que tous les composants de notre bibliothèque dans Strapi, en les définissant avec leur nom respectif. Pour chaque composant, nous précisons également les propriétés associées. Par exemple, pour le composant "Button", nous créons un modèle avec ce nom et définissons les propriétés nécessaires, telles que la couleur, le texte, etc.

Ci-dessous, une image du composant "Button" dans strapi :



Ensuite, afin de pouvoir utiliser nos composants dans une page pour notre site web, nous utilisons les "Single Types". Par exemple, le modèle de la page JoinUs ci-dessous contient plusieurs composants de la bibliothèque de composants :



En utilisant cette structure dans Strapi, nous définissons des pages associées à des chemins spécifiques, où chaque page inclut les composants nécessaires avec leurs noms et propriétés correspondants.

Strapi gère les données essentielles à la construction du site web, telles que les noms des pages, les routes associées, les noms et propriétés des composants présents sur chaque page (couleur, dimension, emplacement, etc.). Ces informations sont utilisées pour générer dynamiquement le contenu des pages et afficher les composants avec leurs configurations spécifiques.

Cette intégration avec Strapi nous permet de gérer facilement le contenu et la structure du site web. Les utilisateurs peuvent créer de nouvelles pages, ajuster les propriétés des composants et mettre à jour le contenu, tout cela à partir de l'interface intuitive de Strapi. Cette approche offre

une grande flexibilité pour personnaliser et adapter le site web en fonction des besoins spécifiques.

- **Difficultés rencontrées**

Pendant ce projet, j'ai été confronté à plusieurs défis techniques et organisationnels. Sur le plan technique, la création des composants a nécessité une adaptation minutieuse pour assurer leur comportement dynamique. Cela impliquait de rendre toutes les propriétés des composants modifiables via Strapi, sans nécessiter de modifications directes dans le code. De plus, la gestion du côté responsive était cruciale : chaque composant devait s'adapter parfaitement à toutes les tailles d'écran, quelle que soit la configuration définie dans Strapi.

Un exemple concret de défi technique rencontré fut la migration du projet de Angular 15 à Angular 17 en 2024. Cette mise à jour majeure a entraîné des ajustements significatifs dans la structure du générateur de site web. Plusieurs éléments du projet précédent ont nécessité une refonte pour s'aligner avec les nouvelles normes et fonctionnalités d'Angular 17. Ce processus complexe a exigé un refactoring approfondi du code pour résoudre les incompatibilités et maintenir la performance du site.

De plus, sur le plan personnel, travailler seul sur ce projet a présenté des défis organisationnels et de gestion du temps. Il a fallu naviguer entre les exigences techniques, les ajustements de conception, et les mises à jour de version, tout en assurant la cohérence et la fonctionnalité globale du site web.

- **Les résultats atteint**

Malgré de nombreuses difficultés rencontrées, je suis parvenu à effectuer la migration de Angular 15 vers Angular 17, à créer de nouveaux composants plus complexes et à surmonter de nombreux défis tout au long du développement de ce projet.

Finalement j'ai donc recréé le site vitrine de Clou4Dev grâce au nouveau générateur de site web, offrant une solution conviviale et flexible pour la création de sites web. Nous continuons à travailler sur l'amélioration et l'expansion du générateur pour répondre aux besoins changeants des clients.

Ce projet m'a permis de renforcer mes compétences en Angular, notamment en apprenant à intégrer et utiliser la nouvelle version, ainsi qu'à maîtriser Tailwind CSS. J'ai également perfectionné la création de composants toujours plus complexes, dynamiques et personnalisables. De plus, j'ai acquis de nouvelles compétences en utilisant Strapi, notamment pour récupérer des informations à afficher dans les composants front-end et pour communiquer efficacement avec Strapi depuis Angular.

Enfin, ce projet m'a appris à surmonter les différentes difficultés rencontrées en me concentrant sur le résultat souhaité, développant ainsi ma capacité à résoudre des problèmes de manière autonome et efficace.

Déploiement du site vitrine et du générateur de site web

- **Objectifs**

L'objectif principal de la troisième mission qui m'a été confiée cette année est de déployer le site vitrine et le générateur de site web pour Cloud4Dev pour pouvoir faciliter la gestion et la mise à jour du contenu de manière centralisée et accessible. En mettant en place une infrastructure de préproduction, Cloud4Dev pourra ainsi exercer un contrôle direct sur le projet et apporter des modifications au contenu du site via Strapi, sans dépendre de l'environnement que j'utilise en local pour développer.

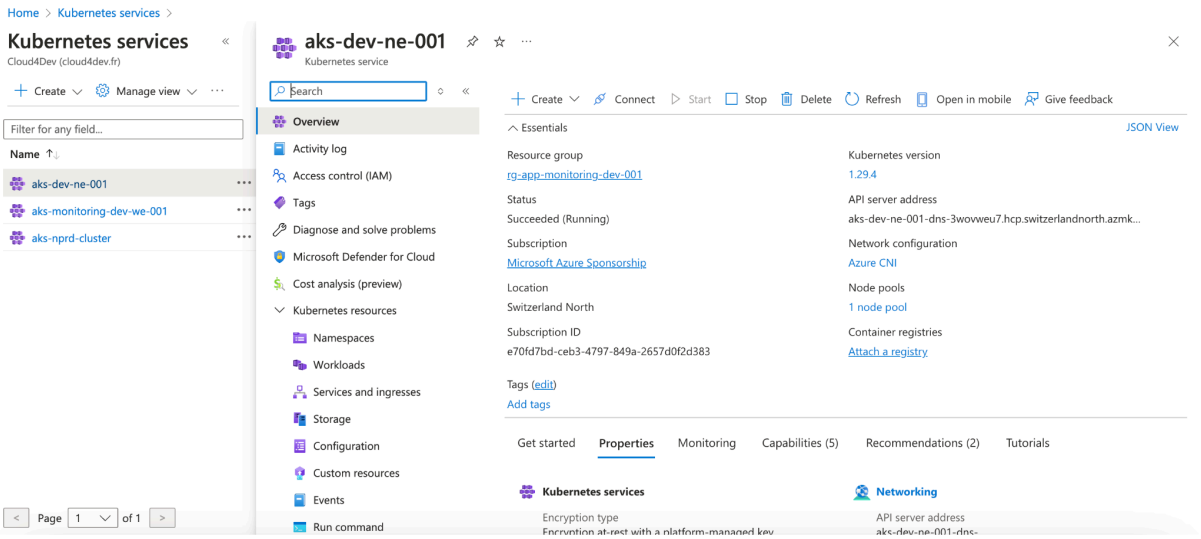
- **Technologie utilisée**

Nous avons utilisé Docker pour containeriser nos applications, ce qui permet de garantir un environnement cohérent et portable pour le développement, le test et la production. En encapsulant nos applications dans des conteneurs, nous pouvons les déployer de manière fiable et uniforme sur différents environnements.

Les images Docker de nos applications sont stockées sur Docker Hub, un registre en ligne qui facilite la gestion et le déploiement de ces images. En hébergeant nos images sur Docker Hub, nous pouvons facilement accéder aux versions les plus récentes de nos applications et les déployer rapidement sur différents serveurs.

Pour orchestrer et gérer nos conteneurs Docker, nous avons utilisé AKS (Azure Kubernetes Service). AKS nous permet de déployer, gérer et mettre à l'échelle nos applications conteneurisées avec Kubernetes, assurant ainsi une haute disponibilité et une gestion simplifiée des ressources. Grâce à AKS, nous bénéficions d'une infrastructure flexible et résiliente pour nos applications, avec des capacités de surveillance et de mise à l'échelle automatique.

Voici une illustration de l'interface de Kubernetes de Cloud4dev :



- **Processus de déploiement**

Durant le développement du générateur de site web, j'ai réalisé l'ensemble du travail en local. Cela signifie que le front-end avec Angular, l'API Strapi, ainsi que la base de données PostgreSQL utilisée par Strapi, étaient tous exécutés sur ma machine de travail.

Pour déployer le générateur en environnement de préproduction, il à fallu faire de nombreux ajustements.

Tout d'abord, pour connecter le front-end à Strapi en production, il a été nécessaire d'ajuster les configurations de requêtes. En local, le front-end se connectait directement à l'URL de Strapi en développement. Pour préparer le déploiement, j'ai modifié cette configuration en utilisant un fichier de variables d'environnement **.env**, où j'ai spécifié l'URL de Strapi en production. Cela a permis de diriger les requêtes vers l'instance de Strapi déployée en production pour l'envoi et la réception des données.

Ensuite, j'ai conteneurisé l'application front-end en utilisant Docker. Le Dockerfile pour le front-end est divisé en deux phases distinctes :

```
# Étape 1 : Construire l'application Angular
FROM node:18 AS build

# Définir le répertoire de travail
WORKDIR /app

# Copier les fichiers package.json et package-lock.json
COPY package*.json ./

# Installer les dépendances
RUN npm install

# Copier le reste du code de l'application
COPY . .

# Construire l'application Angular
```

```
RUN npm run build --prod

# Étape 2 : Servir l'application avec Nginx
FROM nginx:latest

# Copier l'application Angular construite depuis l'étape précédente
COPY --from=build /app/dist/c4d-fluid-design-system-v-2/browser
/usr/share/nginx/html

# Copier le fichier de configuration Nginx
COPY ./nginx.conf /etc/nginx/conf.d/default.conf

# Exposer le port 80
EXPOSE 80

# Démarrer Nginx
CMD ["nginx", "-g", "daemon off;"]
```

Pour construire l'image Docker, il faut s'assurer d'être bien dans le répertoire racine du projet, puis utiliser la commande suivante :

`docker build -t nom-image .`

Cette commande crée une image Docker à partir du Dockerfile dans le répertoire actuel.

Ensuite j'ai utilisé la commande : **`docker image ls`**

Cela permet de lister toutes les images Docker disponibles sur la machine que j'utilise, et de pouvoir vérifier que l'image a été correctement créée.

Enfin, pour tester l'image localement avant le déploiement et s'assurer que tout fonctionne correctement, j'ai utilisé :

`docker run -d -p 80:80 nom-image`

Cette commande permet de démarrer un conteneur à partir de l'image créée, en mappant le port 80 du conteneur au port 80 de la machine hôte.

Après avoir validé le bon fonctionnement de l'application en local, j'ai procédé au push de l'image sur Docker Hub. Cela permettra au fichier YAML du déploiement sur Kubernetes de récupérer l'image.

Les étapes sont les suivantes :

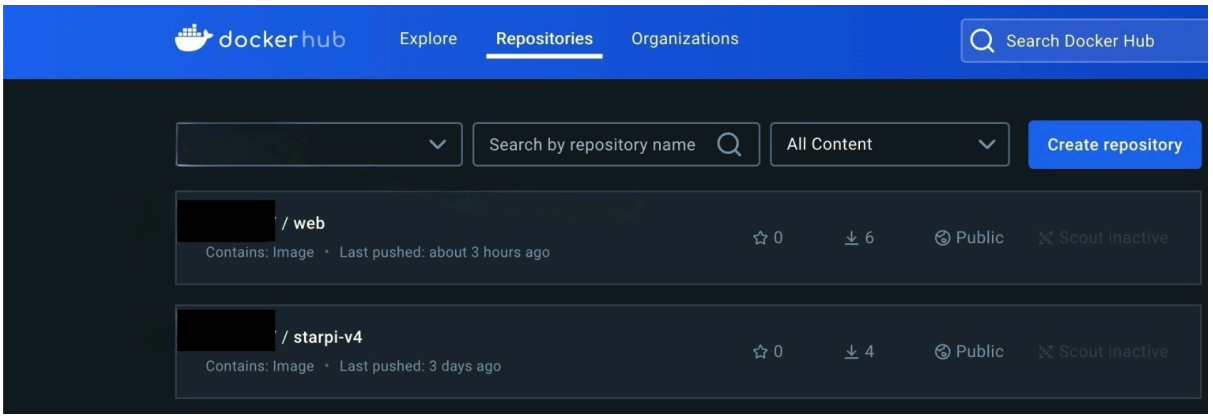
- **Taguer l'image** : **`docker tag nom-image nom-repo/nom-image`**

Cette commande assigne une nouvelle étiquette (tag) à l'image Docker que nous venons de créer. Cette étape est nécessaire pour préparer l'image à être poussée sur Docker Hub.

- **Pusher l'image** : **`docker push nom-repo/nom-image`**

Cette commande envoie l'image Docker vers le dépôt spécifié sur Docker Hub. Ca rend l'image accessible publiquement afin qu'elle puisse être utilisée sur Kubernetes. Une fois l'image poussée sur Docker Hub, le fichier YAML de déploiement sur Kubernetes pourra la récupérer et l'utiliser pour déployer l'application.

Voici l'interface de docker hub avec nos deux images :



Ensuite, pour déployer la partie front-end sur Kubernetes, nous utilisons des fichiers YAML qui définissent le déploiement et le service nécessaires :

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: front
spec:
  replicas: 1
  selector:
    matchLabels:
      app: front
  template:
    metadata:
      labels:
        app: front
    spec:
      containers:
        - name: front
          image: vahek/web
          imagePullPolicy: IfNotPresent
          ports:
            - containerPort: 80
---
apiVersion: v1
kind: Service
metadata:
  name: front
  labels:
    app: front
spec:
  type: ClusterIP
  ports:
    - port: 80
  selector:
    app: front
```

Ce fichier permet de déployer l'application front-end sur Kubernetes en utilisant l'image Docker que nous avons construite et hébergée sur Docker Hub grâce au Dockerfile précédent.

La partie “**deployment**” définit le déploiement de l'application front-end, incluant le nombre de réplicas (ici 1), l'image Docker à utiliser (vahek/web),

et le port exposé par le conteneur (port 80). Cela permet de déployer et gérer l'application dans un environnement Kubernetes.

La partie “**service**” permet de définir comment les pods déployés seront accessibles au sein du cluster Kubernetes, spécifiant un type de service (**ClusterIP**) et mappant le port 80. Cela permet de rendre l'application accessible au sein du cluster Kubernetes.

Le processus précédent explique les principales étapes pour déployer la partie front-end du générateur de site web.

Pour l'image de Strapi, plusieurs ajustements ont été nécessaires concernant la gestion de la base de données. Contrairement à l'environnement de développement local où la base de données PostgreSQL était hébergée sur ma machine de travail, en pré-production, la base de données est désormais exécutée sur la même machine que Strapi. Pour cela, nous utilisons une image officielle de PostgreSQL qui crée une nouvelle base de données fonctionnant sur le port 5432. Il a donc aussi fallu ajuster en fonction le dockerFile de Strapi avec les nouvelles informations comme le nom.

La procédure de conteneurisation et de déploiement pour Strapi suit un processus similaire à celui du front-end. Cependant, le Dockerfile pour Strapi est distinct, car il inclut non seulement la configuration de Strapi mais aussi les paramètres nécessaires pour se connecter à la base de données PostgreSQL, ainsi que d’autres dépendances spécifiques à Strapi.

Ensuite, nous déployons l'image Strapi sur Kubernetes, de manière similaire au front-end, en utilisant des fichiers YAML pour définir le déploiement et les services nécessaires.

Pour automatiser le déploiement, nous avons configuré des pipelines CI/CD avec GitLab sachant que :

- **Front-End :** lorsqu'un push est effectué dans le repository GitLab du front-end sur la branche **`develop`**, la partie front-end est

automatiquement mise à jour en pré-production grâce au pipeline CI/CD de GitLab.

- **Strapi :** de la même manière, lorsque des modifications sont poussées sur la branche **`develop`** du dépôt GitLab pour Strapi, l'image Docker est automatiquement mise à jour en pré-production.

Cependant, la base de données PostgreSQL de Strapi nécessite une mise à jour manuelle après chaque déploiement. À cet effet, je sauvegarde la base de données de développement et je fais un dump vers la base de données de l'environnement de pré-production.

- **Difficultés rencontrées**

Durant tout le processus de déploiement, que j'ai mené en collaboration avec un collègue, nous avons été confrontés à plusieurs difficultés importantes. L'une des premières difficultés a concerné la création des Dockerfiles nécessaires pour construire les images des différentes parties de l'application. Nous avons rencontré des problèmes récurrents liés aux chemins d'accès dans ces fichiers. À plusieurs reprises, les chemins spécifiés étaient incorrects, ce qui nous a contraints à ajuster et corriger les Dockerfiles à plusieurs reprises avant de parvenir à la configuration correcte.

Un autre problème important est survenu lors du déploiement du front-end. Après avoir réussi à surmonter les défis initiaux, nous avons constaté un problème lors des tests de la fonctionnalité du site vitrine et du générateur de site de Cloud4Dev en pré-production : il était impossible de naviguer entre plusieurs pages du site web. Après une longue recherche, nous avons découvert que ce problème était dû à une mauvaise configuration de Nginx, qui ne parvenait pas à gérer correctement les routes supplémentaires. Cette mauvaise configuration empêchait le serveur de rendre correctement les différentes pages du site, ce qui a nécessité des ajustements pour rectifier le problème.

Pour ma part, j'ai rencontré plusieurs défis techniques, surtout parce que mon expérience avec Docker se limitait jusqu'à présent à des projets de développement local sur ma propre machine. Je n'avais jamais eu à utiliser Docker dans un contexte de déploiement comme celui-ci, ce qui m'a obligé à apprendre sur le tas. J'ai dû comprendre comment fonctionnent les Dockerfiles et les fichiers YAML utilisés pour le déploiement sur Kubernetes.

- **Résultats atteints**

Finalement, malgré les difficultés rencontrées, que ce soit pour la préparation du générateur de site web en pré-production, la création des images Docker, ou la configuration des fichiers YAML, nous avons réussi à déployer avec succès la partie front-end (le projet Angular) ainsi que le CMS Strapi en pré-production. Voici l'url du front-end déployer : https://website.noprod.cloud4dev.app et voici l'url de strapi déployer : https://strapi.cloud4dev.app. Ce déploiement permet désormais de gérer et de mettre à jour le contenu de manière centralisée et accessible.

Un aspect clé de cette configuration est l'intégration continue que nous avons mise en place. À chaque fois que nous poussons du code sur la branche **`develop`**, les modifications sont automatiquement déployées en pré-production pour le front-end et Strapi. Cette automatisation facilite grandement la maintenance du projet. Par exemple, si Cloud4Dev souhaite que je corrige rapidement un problème avec un composant, que j'en crée de nouveaux, ou que je modifie une page, ces changements sont immédiatement visibles en pré-production. Cela permet une réactivité accrue et simplifie la gestion des ajustements et des évolutions du projet, tout en préparant efficacement la prochaine version pour la production, sans dépendre de l'environnement local de développement.

4. Conclusion

En définitive, les trois missions qui m'ont été confiées chez Cloud4Dev cette deuxième année m'ont permis de concevoir, développer et déployer le générateur de site web qui contient le site vitrine de Cloud4Dev. Cette expérience m'a également permis de renforcer mes compétences et d'acquérir encore plus d'expérience en tant que développeur full-stack au sein d'une entreprise dynamique.

Même si je n'ai pas travaillé avec une large équipe de développement cette année, j'ai été amené à collaborer avec quelques collègues, de façon intermittente. Travailler avec eux m'a permis d'apprendre beaucoup de choses, mais, la plupart du temps, j'ai dû travailler de manière autonome, ce qui a nécessité d'apprendre en autodidacte. Il était parfois moins attrayant de travailler sans aucun lien avec mes collègues, mais cette situation m'a permis de renforcer mes compétences de manière significative.

Outre le renforcement de mes compétences techniques, cette année m'a également permis de perfectionner des qualités professionnelles indispensables, telles que la ponctualité et la communication efficace en réunions. J'ai appris à mieux gérer mon temps et à optimiser mes échanges avec mes collègues, notamment parce qu'ils avaient souvent beaucoup de travail et peu de temps à me consacrer. Cette capacité d'adaptation a grandement contribué à la réussite des différentes missions qui m'ont été confiées.

Participer à l'édification de Cloud4Dev à travers les différentes missions que j'ai réalisées a été pour moi un grand honneur et une source de fierté. Cette expérience a consolidé ma passion pour le développement web et m'a ouvert de nouvelles perspectives pour mon avenir professionnel.

5. Bibliographie

1. Awwwards - Website Awards : https://www.awwwards.com
2. Les bases de Figma 2023 (cours gratuit) : https://www.youtube.com/watch?v=7a0QW3zJc0A&t
3. Freepik : https://fr.freepik.com

   Site web de Cloud4dev : https://www.cloud4dev.com

   Documentation officielle Angular : https://angular.io/

   Documentation officielle TailwindCss : https://tailwindcss.com/

   Documentation officielle Git : https://git-scm.com/

   Documentation officielle GitLab : https://about.gitlab.com/

   Site officielle Jira : https://www.atlassian.com/fr/software/jira

   Documentation officielle Docker : https://docs.docker.com

   Documentation officielle Strapi : https://docs.strapi.io/