

Rapport d'alternance

EFREI – Mastère Dev Manager Full Stack 2022-2024

Présenté par : Jérémy Weltmann

Maitre d'apprentissage : Fabien Scolas (Carbonscore)

Entreprise : Carbonscore



Table des matières

Présentation de l'entreprise.....	3
Information clés :	3
Organigramme de l'entreprise :	4
Affiliation à différents groupes :	4
Les Partenaires :	4
Les Clients :	5
Le service d'affectation.....	5
Mon rôle au sein de l'entreprise :	5
Présentation du projet.....	6
Objectifs du projet.....	7
Gestion de Projet.....	7
Les réunions :	16
Les Difficultés Rencontrées	17
Réalisation :	18
Outils utilisés :	18
Les Tests Unitaires.....	21
La gestion du Versionning	22
Fonctionnement Général de l'entreprise	23
Stratégie générale de l'entreprise	23
Démarche Qualité :	23
Bilan du projet	26
Conclusion.....	27

Dans le cadre de mon mastère Dev Manager Full Stack à l'EFREI, j'ai été recruté en tant que **Apprenti Développeur Full Stack** au sein de l'entreprise **TechupClimate**.

Je vais maintenant vous présenter en détail, l'entreprise qui m'a accompagné durant ces deux années.

Présentation de l'entreprise

TechupClimate est une entreprise spécialisée dans une **solution logicielle** qui se nomme « **CarbonScore** » dont l'objectif est de **sensibiliser les collaborateurs** des entreprises sur **l'impact environnemental du numérique**. Cette solution vise à leur faire prendre conscience que leurs usages du numérique peuvent entraîner des répercussions sur notre planète.

Il s'agit d'une **startup** créée le 1er aout 2019 par Jean Christophe Bories, et enregistrée sous le statut de Société par actions simplifiée.

L'entreprise se positionne comme un acteur majeur de la transition écologique de nos usages numérique, afin de garantir une économie durable tout en prenant en compte les pratiques du quotidien qui pourraient nuire à la planète.

La solution logicielle CarbonScore permet aux collaborateurs des entreprises d'obtenir un **indicateur**, le « CarbonScore » qui évalue si leur **performance** en termes de **consommation numérique** est **respectueuse** de l'environnement ou non.

Afin de sensibiliser les collaborateurs, diverses fonctionnalités ont été mises en place, telles que des suggestions, des quizz, ainsi qu'un système de badges pour récompenser les utilisateurs qui prennent conscience de leur impact et le réduisent. Un très grand nombre de statistiques sur les usages numériques du quotidien a également été mis en place sous la forme d'un tableau de bord, permettant de déterminer quelles utilisations numériques doivent être réduites.

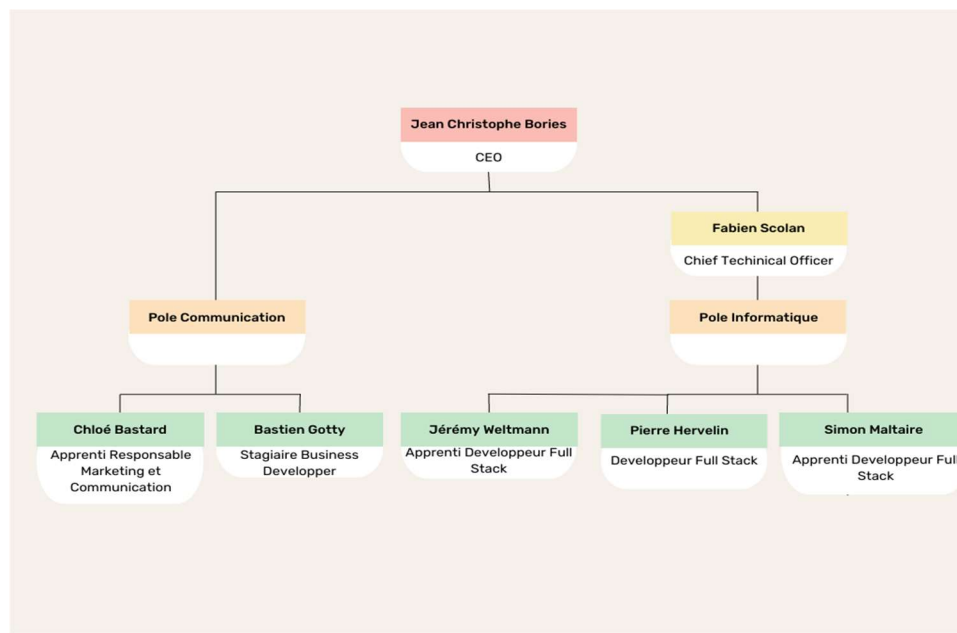
L'entreprise compte actuellement **7 employés** qui assurent la maintenance, l'amélioration du logiciel « CarbonScore », ainsi que sa promotion.

Information clés :

Le siège de l'entreprise est situé au 14 rue de l'Hermitage à Pontoise (95300).

Le 7 octobre 2022, elle a enregistré un capital social de 13 426.00€.

Organigramme de l'entreprise :



Affiliation à différents groupes :

Pour renforcer son impact auprès des entreprises, TechupClimate est devenue membre de divers groupes d'actions du numérique tels que :

- Plante Tech' Care, un groupe qui regroupe les acteurs du numérique qui lutte pour un numérique responsable.
- L'association pour la transition Bas Carbone, dont l'objectif est de sensibiliser, former, fédérer et donner des moyens d'action concrets aux organisations et aux citoyens pour réussir leur transition bas carbone.
- France Digital, une association qui regroupe les startups du numérique afin de défendre l'innovation en France et en Europe
- L'INR, une association qui a pour but de réfléchir sur le numérique responsable.

En étant membre de ces différents groupes, elle démontre son engagement envers la responsabilité environnementale et sa volonté de promouvoir le numérique responsable.

Les Partenaires :

TechupClimate compte également de nombreux partenaires tels que Microsoft, Keyrus, Okuden, Unatera.

Ces partenariats permettent à l'entreprise de bénéficier des technologies et des services proposés par ses partenaires, comme avec l'utilisation des données MS Graph de Microsoft, qui permet de récupérer les données des solutions Microsoft 365.

Les Clients :

La clientèle de TechupClimate est **diversifiée**, car **toutes les entreprises** sont **impactées par le numérique** et doivent donc faire attention à leurs usages quotidiens.

Parmi les clients de TechupClimate, on compte de grandes entreprises comme :

- Rémi Cointreau
- Sonergia
- Banque Populaire

Elle compte aussi des **PME** comme ecoCo2 et ainsi que des **TPE**, ce qui démontre que la solution proposée par TechupClimate est adaptée à tous types d'entreprise.

Le service d'affectation

J'ai été intégré à l'équipe de développement, responsable du développement du logiciel « CarbonScore » ainsi que du site vitrine qui permet aux clients de découvrir les différents services proposés par ce logiciel.

En tant qu'apprenti Développeur Full Stack, je suis amené à travailler avec tous les membres de l'entreprise. Je collabore principalement avec l'équipe de développement, avec laquelle nous travaillons sur de nombreuses tâches. Ils m'aident également à résoudre les problèmes techniques que je peux rencontrer et à améliorer ma manière de développer, afin qu'elle corresponde aux attentes de l'entreprise et permettre de délivrer un logiciel de qualité. Je collabore également avec l'équipe Commerciale/Marketing, qui peut remonter des idées ou contenus qu'ils souhaiteraient intégrer dans le logiciel.

Mon rôle au sein de l'entreprise :

Au sein de TechupClimate, mon rôle est **d'améliorer le logiciel CarbonScore** en intégrant de **nouvelles fonctionnalités** suggérées par les clients ou par l'équipe globale de l'entreprise lors de réunions.

Mon rôle comporte également des tâches consistant à **corriger les bogues** qui pourraient être remontés, afin de garantir un logiciel fiable. Je suis aussi chargé de **modifier des fonctionnalités existantes pour les améliorer**, afin qu'elles répondent aux nouvelles attentes des clients, et de **migrer certains éléments du logiciel vers une nouvelle version**.

En tant que Développeur Full Stack, je suis **impliqué dans toutes les phases de développement**. Tout d'abord, je participe à la réunion de backlog où sont discutées des nouvelles fonctionnalités à développer. Ensuite, **je commence par la maquette de la fonctionnalité en respectant les spécifications demandées lors de la réunion du backlog**.

Après avoir réalisé ma maquette, je la sou mets à mon équipe pour obtenir des retours, ce qui me permet de faire des ajustements si nécessaire pour qu'elle corresponde aux attentes. Une fois la maquette acceptée, je passe au développement de la partie back-end de la fonctionnalité, c'est-à-dire la partie qui va récupérer ou insérer des données

dans la base de données et créer des routes API REST afin de pouvoir y accéder ensuite dans ma partie front-end.

Après avoir terminé la partie back-end, je peux utiliser les fonctionnalités créées pour afficher les données pertinentes dans ma partie front-end, où je me charge de développer l'interface utilisateur en suivant la maquette.

Que ce soit pour la partie front-end ou le back-end, chaque fois que je crée une nouvelle fonction, je dois réaliser des tests unitaires afin de vérifier que toutes les fonctionnalités couvrent tous les scénarios possibles et d'éviter certains bogues.

Après avoir présenté la structure dans laquelle je travaille, je vais maintenant vous **présenter un projet** sur lequel j'ai travaillé afin **d'illustrer le fonctionnement d'un projet au sein de mon entreprise**.

Présentation du projet

Le projet que j'ai décidé de vous présenter est la **refonte du tableau de bord de la plateforme Carbonscore**. J'ai choisi de parler de ce projet, car il s'agit d'un projet de **grande taille** pour l'entreprise nécessitant **4 mois de développement** sur lequel je suis intervenu, ainsi que plusieurs autres membres de l'équipe de développement.

J'ai aussi décidé de parler de ce projet car il a **nécessité des tâches impliquant du front-end et du back-end**. Un exemple de projet représentant toute la **complexité du métier de développeur web full stack** en jonglant sur de nombreuses compétences : développement, web design, gestion de projet.

Le développement de ce projet a été initialisé à la **suite de retours de certains clients utilisant la plateforme**, qui trouvaient le tableau de bord **difficile à prendre en main** et qui a été remonté à l'équipe de développement par l'équipe de communication lors des réunions hebdomadaires avec l'ensemble de l'équipe.

Le tableau de bord de la plateforme Carbonscore est une des pages les plus **essentielles** de la plateforme permettant d'avoir un récapitulatif quotidien sur les consommations des données des outils Microsoft 365. Nous retrouvons aussi, l'indicateur Carbonscore, qui permet de savoir si un utilisateur a un usage respectueux de l'environnement. De plus, des équivalences sont fournies pour traduire les données de consommation en termes concrets pour les utilisateurs, facilitant ainsi leur compréhension.

Le tableau de bord était difficile à prendre en main en raison de la **grande quantité d'informations affichée à l'écran**, ce qui rendait la navigation et la compréhension des données complexes pour les utilisateurs. La refonte a donc visé à **améliorer la présentation et l'organisation** de ces informations pour améliorer **l'expérience utilisateur**.

Mettre en place cette refonte du tableau de bord était essentiel pour garder une base solide d'utilisateurs sur l'application.

Objectifs du projet

Le principal objectif de cette refonte du tableau de bord était de créer un tableau de bord plus simple à comprendre et à prendre en main.

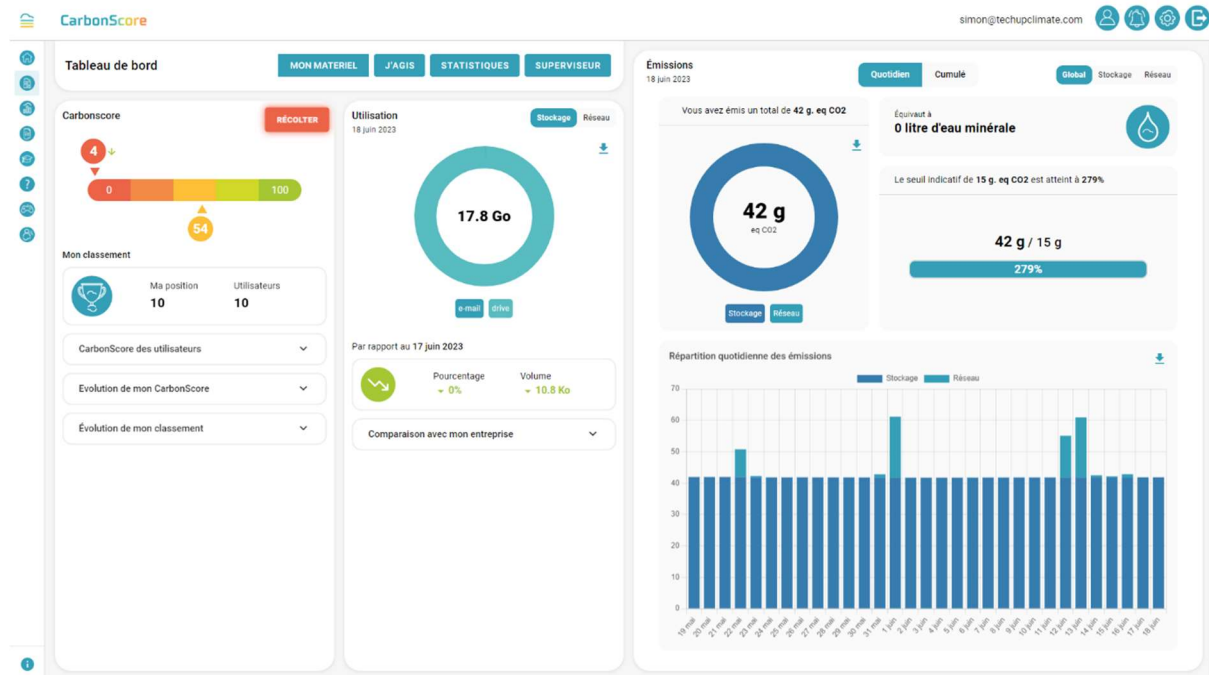


Figure : Ancien tableau de bord

Sur notre plateforme, nous collectons les données de consommations quotidiennes des utilisateurs de leur application de la suite Microsoft 365. Ce projet visait également à intégrer la **consommation mensuelle et annuelle** des outils Microsoft 365, des données qui étaient absentes des rapport **Microsoft MsGraph** et qu'il a fallu créer.

Microsoft MsGraph est notre source de données, nous permettant de récupérer les informations des outils de Microsoft 365 grâce à des appels à leur API.

Une **fonctionnalité très demandée parmi nos clients** qui souhaitait avoir un récapitulatif plus poussé en comparant les consommations des utilisateurs par mois et par année afin de voir l'évolution des consommations des utilisateurs et voir si la plateforme avait impact sur leurs usages de leurs outils informatiques.

La refonte du tableau de bord a également nécessité des **ajustements** au niveau de **l'affichage des graphiques GraphJS**. Il a fallu non seulement repositionner les légendes à l'extérieur de certains graphiques pour améliorer leur lisibilité, mais aussi modifier la manière dont certains éléments étaient affichés au centre des graphiques, comme dans le cas des graphiques en forme de donut. Ces modifications ont permis d'optimiser l'espace disponible et d'améliorer la présentation visuelle des données.

Gestion de Projet

Sur ce projet comme dans tous les autres projets, nous travaillons avec la méthode de gestion **agile Kanban**, qui vise à **s'adapter aux demandes changeantes du client** afin que le produit corresponde **aux nouvelles attentes du client**. Cette approche est commune à toutes les méthodes de gestion agile, mais Kanban a comme principal

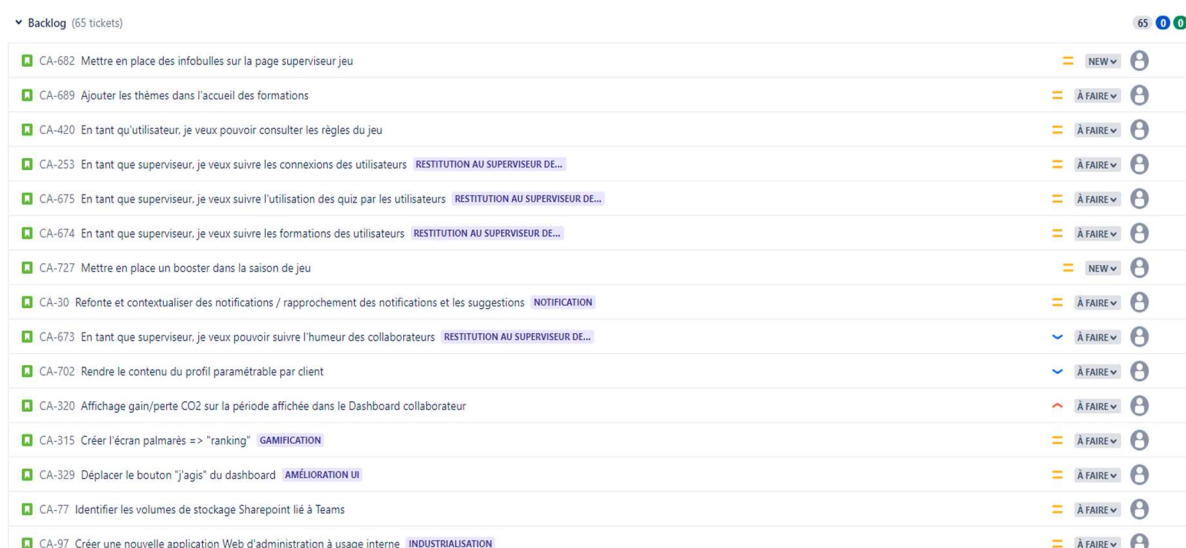
atout de prôner une **visualisation graphique des tâches** à l'aide de **tableaux de bord**.

Cette visualisation permet de **hiérarchiser la priorité des tâches** et de suivre **l'avancement d'un projet** ainsi que **d'améliorer la collaboration entre les différents membres de l'équipe**, par exemple, si on remarque qu'une tâche est en cours depuis longtemps sur le tableau de bord, on peut lui proposer de l'aide sur ce projet complexe.

Afin de réaliser cette méthode de gestion, nous utilisons le **logiciel Jira**, qui nous permet d'avoir une vue globale de tous les projets en cours ou à faire.

Toutes les personnes de l'équipe peuvent avoir accès à l'ensemble des projets.

Pour gérer les différents projets, deux tableaux de gestion de projet ont été créés :



▼ Backlog (65 tickets)		65	0	0
CA-682	Mettre en place des infobulles sur la page superviseur jeu	NEW		
CA-689	Ajouter les thèmes dans l'accueil des formations	À FAIRE		
CA-420	En tant qu'utilisateur, je veux pouvoir consulter les règles du jeu	À FAIRE		
CA-253	En tant que superviseur, je veux suivre les connexions des utilisateurs	À FAIRE		
CA-675	En tant que superviseur, je veux suivre l'utilisation des quiz par les utilisateurs	À FAIRE		
CA-674	En tant que superviseur, je veux suivre les formations des utilisateurs	À FAIRE		
CA-727	Mettre en place un booster dans la saison de jeu	NEW		
CA-30	Refonte et contextualiser des notifications / rapprochement des notifications et les suggestions	À FAIRE		
CA-673	En tant que superviseur, je veux pouvoir suivre l'humeur des collaborateurs	À FAIRE		
CA-702	Rendre le contenu du profil paramétrable par client	À FAIRE		
CA-320	Affichage gain/perte CO2 sur la période affichée dans le Dashboard collaborateur	À FAIRE		
CA-315	Créer l'écran palmarès => "ranking"	À FAIRE		
CA-329	Déplacer le bouton "J'agis" du dashboard	À FAIRE		
CA-77	Identifier les volumes de stockage Sharepoint lié à Teams	À FAIRE		
CA-97	Créer une nouvelle application Web d'administration à usage interne	À FAIRE		

Figure : Tableau Backlog

Un **tableau Backlog** regroupe **tous les projets qui n'ont pas encore débuté**. On y **retrouve aussi des idées non-concrètes de projet** qui sont encore sujet à débat.

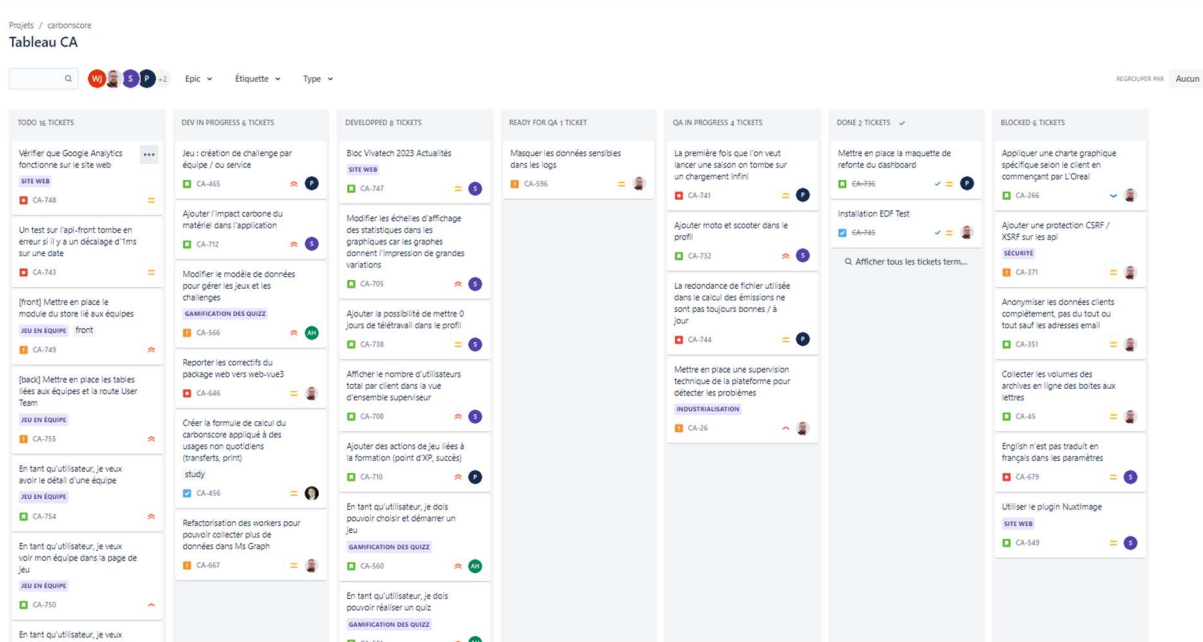


Figure : Tableau de bord Jira

Un tableau de bord Jira qui va regrouper les projets en cours. Il permet de suivre l'avancée des différents projets. Ce tableau est basé sur le modèle de **ticketing**.

Les tickets sont classés sous différentes catégories selon leur avancement respectif :

- **TODO** : Etat initial d'une tâche lors du lancement d'un projet.
- **Dev in Progress** : Tâches en cours de développement.
- **Developed** : Tâches qui ont été développées en attente d'une revue de code.
- **Ready For QA** : Revue de code effectuée et en attente de passer sur un environnement de test.
- **QA in Progress** : Phase de test de la fonctionnalité
- **Done** : La fonctionnalité est sur l'environnement de production.

La liste des ticket Done sont vidés au fur et à mesure afin de ne pas être encombrée par un très grand nombre de tickets inutile.

- **Blocked** : Le ticket est suspendu pour une certaine raison en attendant d'avoir une solution.

Au lancement d'un projet, les différentes tâches sont transférées du tableau de backlog au tableau de bord, et ont le statut TODO. Ensuite, celui qui est affecté à cette tâche va mettre à jour le statut du ticket selon l'avancée du ticket, ce qui permet d'améliorer le suivi global de l'entreprise sur un projet.

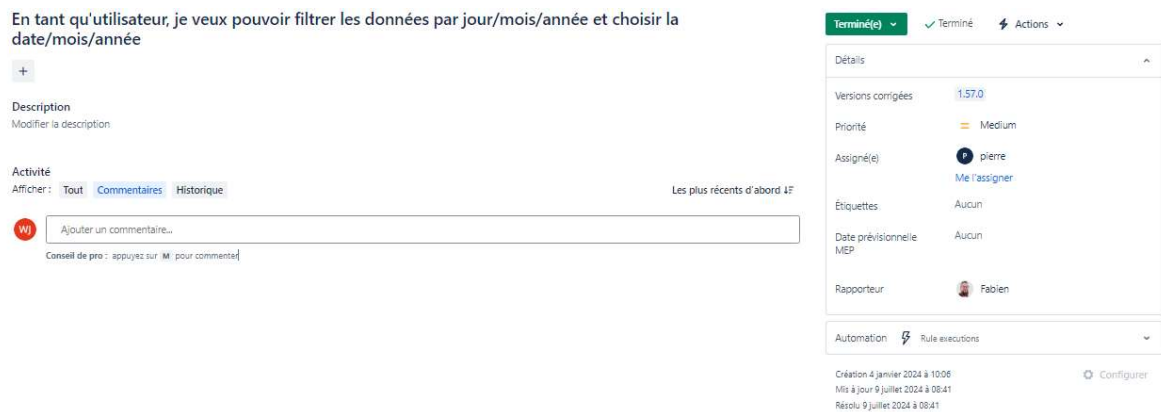


Figure : Un ticket sur Jira

Un **ticket** est associé à une **personne** qui a le rôle de **développer** la fonctionnalité.

Une légère description est indiquée sur le ticket afin de décrire la fonctionnalité à implémenter. On a aussi une mention de l'émetteur du ticket, ce qui va nous permettre de savoir qui est la personne a contacté pour avoir des précisions sur le ticket. Il est aussi possible d'ajouter des images pour inclure des maquettes facilitant ainsi le développement de la fonctionnalité.

Chaque ticket est identifié par une **nomenclature unique**, CA-numeroduticket, qui permet de le retrouver facilement, notamment lors de l'utilisation de l'outil de versionning que je détaillerai plus loin dans ce rapport.

Les tickets dans les colonnes sont triés selon leur niveau de priorité. Le niveau de priorité d'un ticket **dépend de la demande accrue des clients**, mais aussi si elle impacte directement l'application. Par exemple, un ticket de bug qui empêche l'utilisateur d'accéder à l'application va être traité en premier. Le niveau de priorité du ticket dépend aussi du niveau de difficulté de la fonctionnalité a implémenté, si une tâche importante est une tâche très facile à implémenter, elle sera priorisée par rapport à une autre qui serait plus longue à implémenter.

Comme il s'agit d'une **petite taille d'entreprise**, tous les projets ne sont **pas affectés** à un **chef de projet**. La plupart du temps, chaque développeur travaille tout seul sur une fonctionnalité et développe chaque partie de la fonctionnalité front comme le back.

Mais, il arrive que pour une grosse fonctionnalité, une personne de l'équipe s'occupe **d'organiser** et de **préparer les tâches à venir** en amont en réalisant une **documentation** expliquant la fonctionnalité à développer ainsi qu'une partie documentation technique afin de nous aider lors du développement de la fonctionnalité, en découpant les tâches en plusieurs tickets, ainsi qu'une description des attendus.

Cette nomination n'a pas de caractère officiel, c'est souvent celui qui s'occupe de débiter la grosse fonctionnalité qui va préparer le terrain pour les autres membres de l'équipe.

Il faut savoir que selon la taille du projet, la **décomposition en taches** va être totalement **différente**. Par exemple, un projet de **petite taille (entre 1 semaines et 1**

mois développement) est décomposé en une **seule tâche** et sera réalisé par **un seul développeur**, un projet de **moyenne taille** (entre 1 mois et 3 mois de développement) sera décomposé en **plusieurs tâches** et sera réalisé par une **seule personne**. Un projet de **grande taille** (> 3 mois) sera décomposé en **plusieurs tâches** et sera réalisé par **plusieurs développeurs**.

Cette décomposition en petites tâches permet d'améliorer **la fluidité de l'avancement du projet** et de **gérer les imprévus** qui pourraient survenir tout au long du développement.

Aux vues de la grande taille du projet, les tâches ont été décomposées en diverses tâches **sous forme de l'épic** « Refonte du dashboard ».

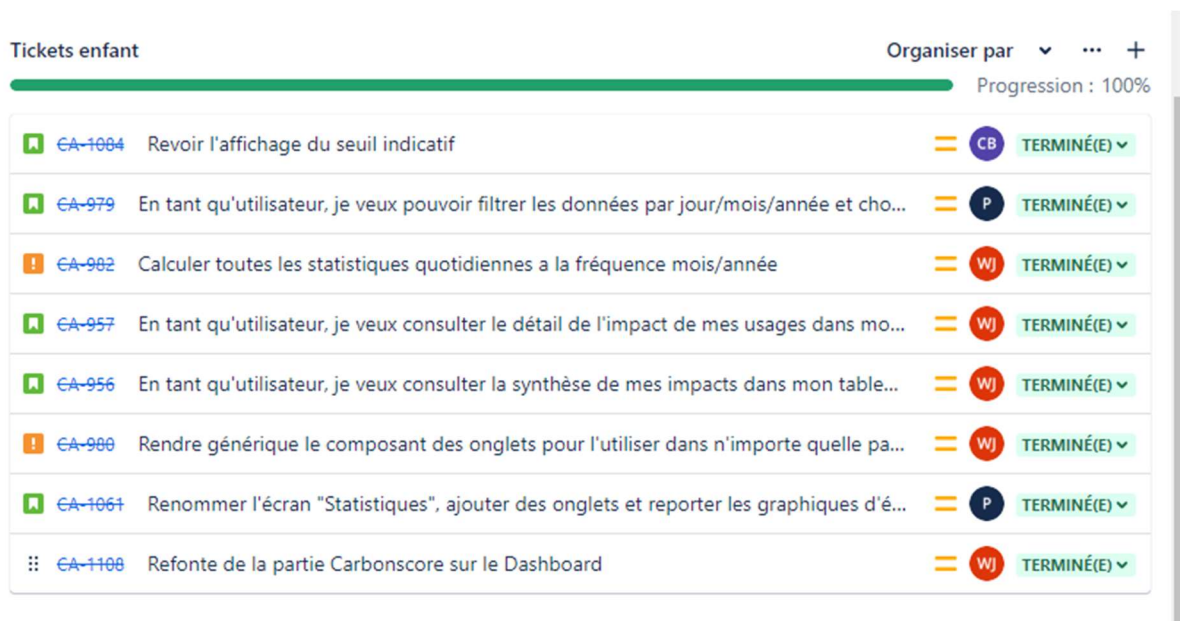


Figure : Décomposition des tâches sous l'épic « Refonte du dashboard »

Les tâches ont été décomposées sous forme de **users stories** pour les **fonctionnalités à implémenter sur le tableau de bord**. Les users stories sont des phrases simples qui décrivent le besoin d'un utilisateur et pour les **fonctionnalités techniques**, de simples **phrases décrivent l'ajout technique** à implémenter dans la plateforme.

Ce projet a sollicité **3 développeurs** pour l'ensemble du projet.

Un développeur s'est occupé de préparer le terrain en découpant les besoins des utilisateurs en une épic et en décomposant les besoins en plusieurs tâches. Il a ensuite rédigé une documentation technique sous le logiciel Confluence pour décrire les fonctionnalités à mettre en place pour répondre au besoin des utilisateurs. Il a également intégré **différents schémas d'architecture de base de données** avec l'outil **MERMAID** dans **Confluence** et a mis en place une architecture pour la partie back-end avec la **définition des différentes routes** et la définition **des retours** de ces différentes routes, ainsi que la création d'interfaces, permettant ainsi de garantir un **développement** et une **maintenance** plus **facile** de la fonctionnalité mise en place.

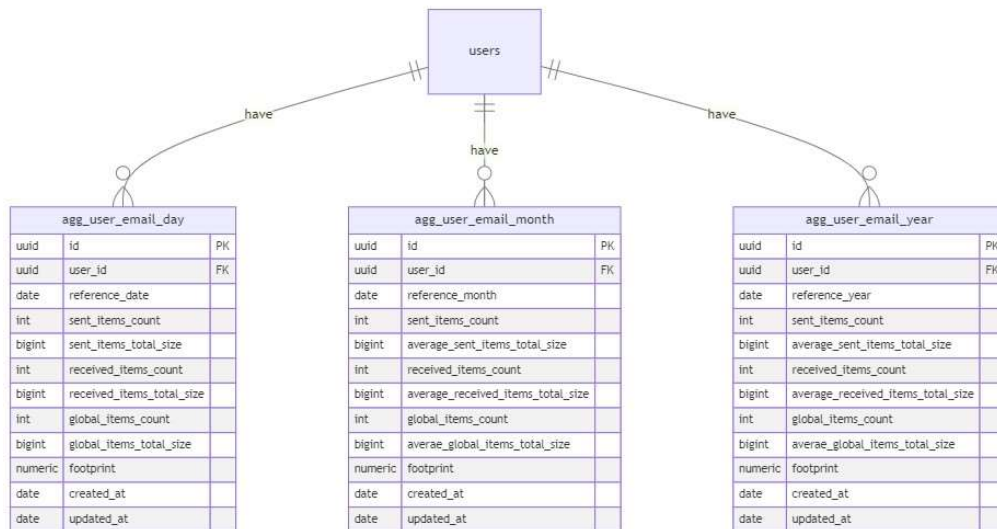


Figure : Schéma d'architecture de base de données réalisées avec MERMAID

Le design de ces différents schémas et architectures a été mûrement réfléchi lors de **réunions spécifiques** entre les différents membres de l'équipe de développement pour que tout le monde soit bien d'accord avec ce qui allait être mis en place.

POST - user news

users/:userId/news

Paramètre

```
1 {
2   news: ['Device', 'Dashboard']
3 }
```

Figure : Exemple de schéma d'architecture des routes d'un autre projet

Enfin, Il a maqueté les interfaces du nouveau tableau de bord grâce au logiciel **Figma** en suivant les spécifications définies dans la **documentation technique**.

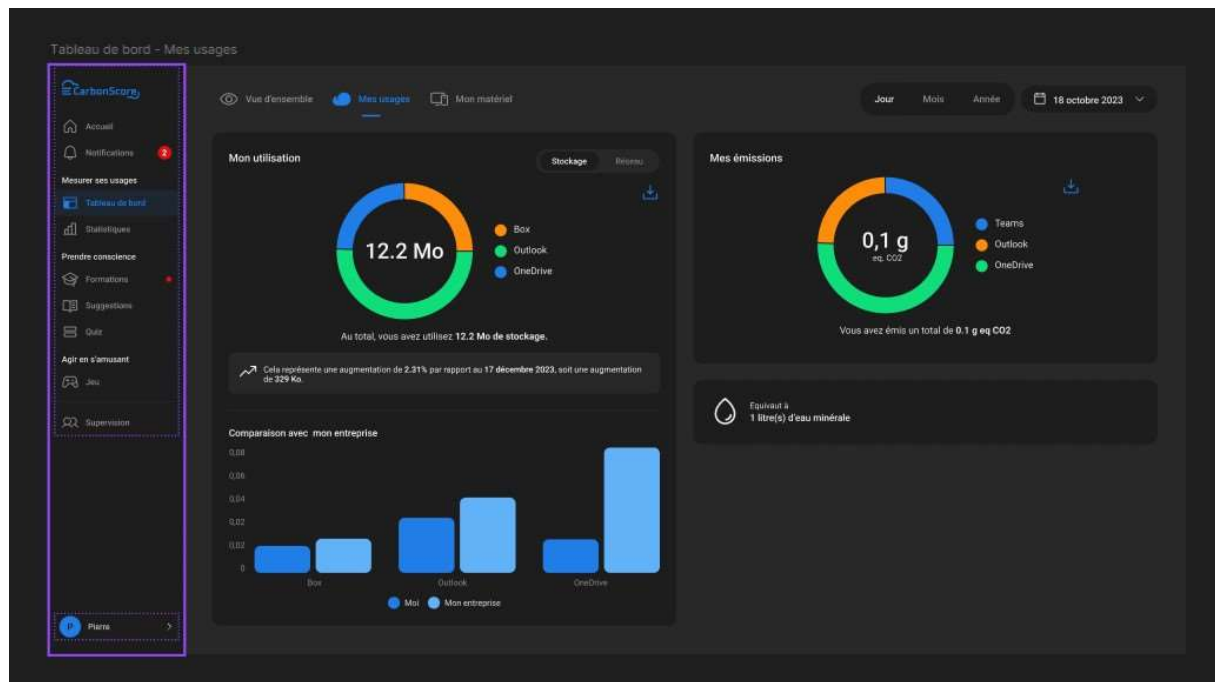


Figure : Maquette du tableau de bord réalisée sur Figma

Sur ce projet, je me suis occupé de différentes tâches :

- **Mise en place de la refonte du tableau de bord** en respectant les différentes maquettes conçues par un autre développeur.

La refonte du tableau de bord a inclus l'ajout d'un système d'onglets pour améliorer la navigation et l'organisation des informations. Deux onglets principaux ont été créés : un onglet "Vue d'ensemble" présentant l'impact environnemental des outils ainsi que l'indicateur Carbonscore, et un onglet « Mes usages » offrant une analyse détaillée de la consommation des utilisateurs par type d'usage. Ce système d'onglets, conçu comme un composant réutilisable, **facilite la navigation et l'accès aux informations pertinentes**, tout en **optimisant le développement** futur en évitant la duplication de code. Alors qu'auparavant, toutes ces informations étaient tous sur la même page, ce qui limitait sa lisibilité et sa compréhension.

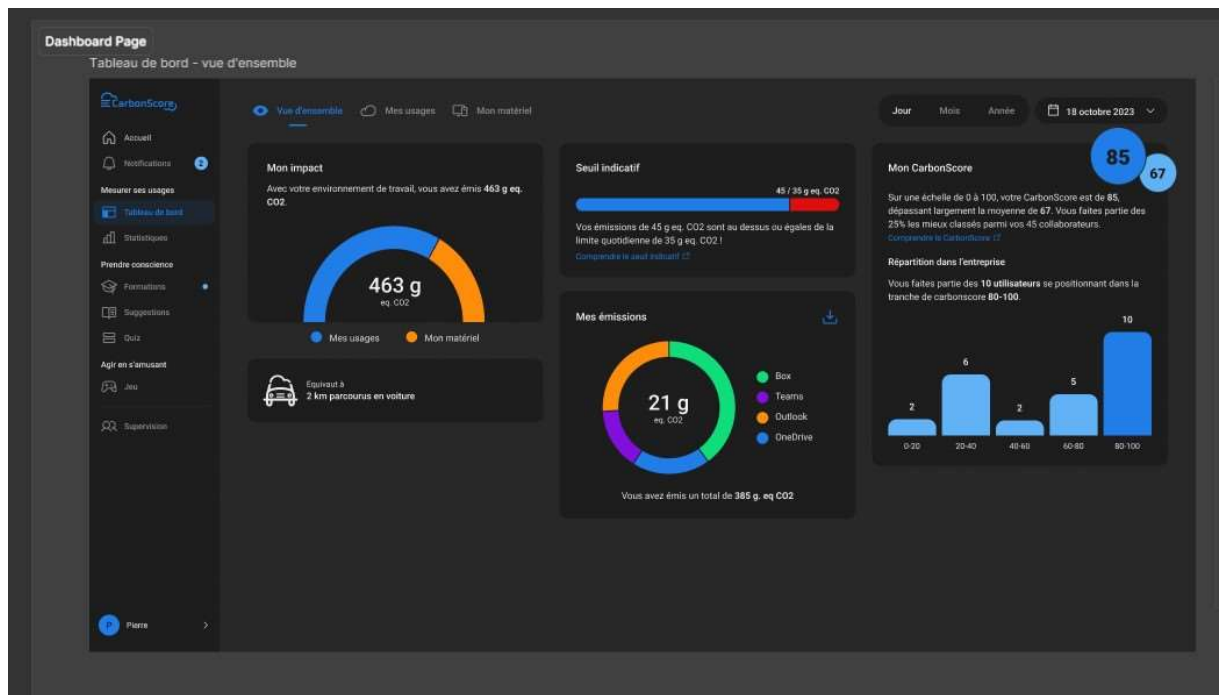


Figure : Tableau de bord « Vue d'ensemble »

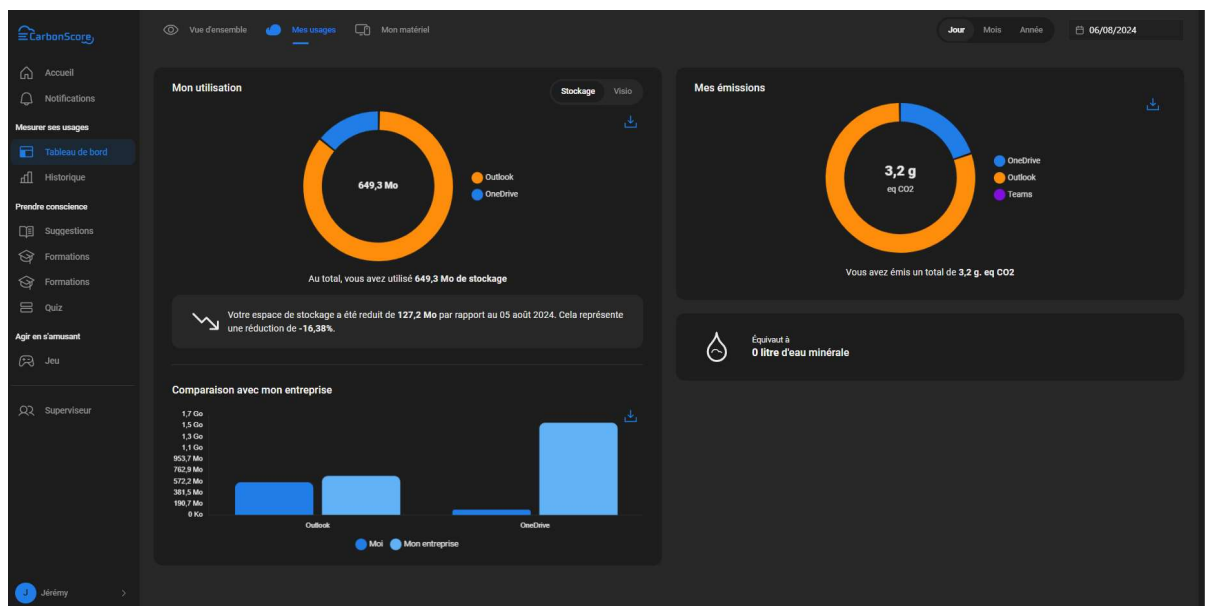


Figure : Tableau de bord « Mes usages »

- Je me suis également occupé de mettre en place la **partie back-end du calcul des consommations mensuelles et annuelles** des différents usages, en utilisant la documentation technique comprenant des explications sur son fonctionnement et des schémas d'architectures de base de données pour guider le développeur à réaliser la fonctionnalité back-end.

Sur la partie back-end, j'ai ajouté différentes routes POST pour ajouter à la plateforme, le calcul des données mensuel et annuel des usages des utilisateurs ainsi que de routes GET pour pouvoir récupérer ces données et les afficher par la suite dans le front-end.

Toutes ces routes respectent l'**architecture REST**. Il s'agit d'un style d'architecture pour concevoir des services web qui s'appuie sur les protocoles HTTP et des méthodes HTTP standards telles que GET, POST, PUT, DELETE afin d'effectuer des opérations CRUD (Create, Read, Update, Delete). Ces opérations permettent de manipuler des ressources identifiées par des URL uniques. Les données sont échangées dans un format standardisé, comme JSON, ce qui facilite l'interopérabilité entre différents systèmes.

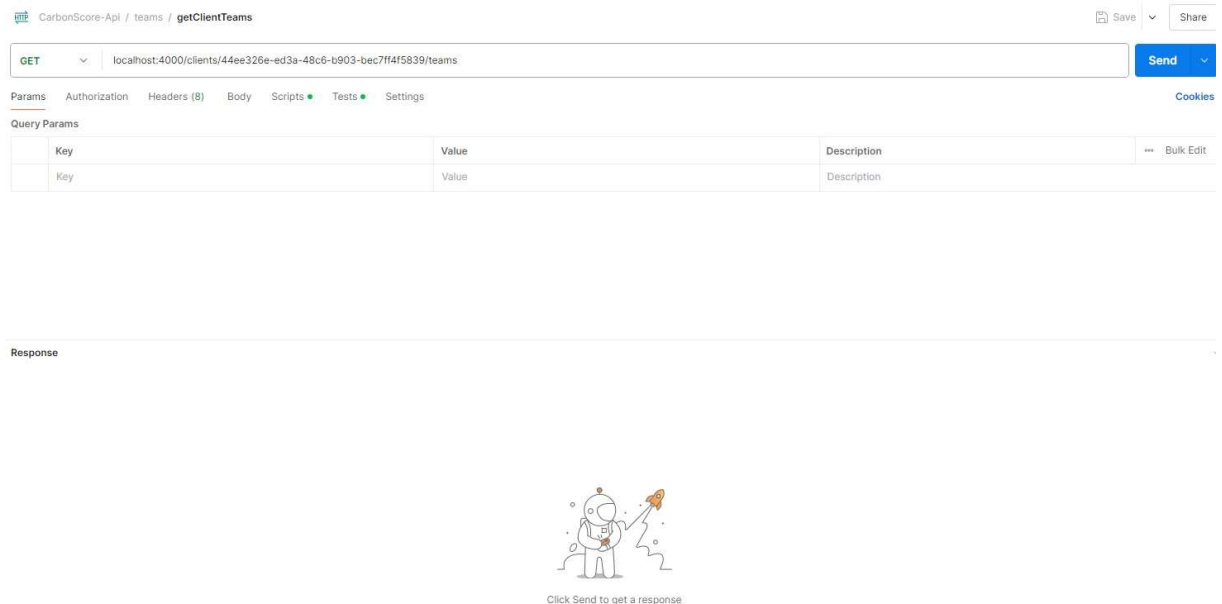


Figure : Postman, un outil de test des routes API

Pour tester de la bonne exécution de ces routes, nous utilisons l'outil **Postman**, afin de **tester les routes** créées en simulant différents **scénarios** et **paramètres**, ce qui permet de détecter les potentiels erreurs.

Avant la mise en production de la fonctionnalité, il est essentiel de tester que la route fonctionne sur le serveur de test, et non simplement en local afin d'éviter tout problème de configuration qui pourrait arriver lors de la mise en production.

La partie front-end de la fonctionnalité, permettant de filtrer les données de manière mensuelle et annuelle a été réalisée par la personne ayant conçu les différentes maquettes.

- Une autre développeuse web a été mobilisée afin de rectifier certains éléments implémentés lors de la refonte à la suite de retours de l'équipe de communication qui trouvait que certains éléments n'étaient pas assez intuitifs.

Les réunions :

Une des particularités du travail au sein de l'entreprise TechupClimate est que l'essentiel du travail se fait en **distanciel**, ce qui rend la **communication** entre les différents collaborateurs de l'entreprise **indispensable**.

Pour ce faire, nous utilisons le logiciel **Microsoft Teams**, qui permet d'organiser des réunions.

Au cours d'une semaine de travail, différents types de réunions sont organisés :

- Tout d'abord, nous avons une **réunion hebdomadaire** qui permet de faire un point sur **les actualités de l'entreprise en général** avec tous les services de l'organisation.
- Tous les matins, un **daily meeting** est organisé avec l'équipe de développement afin de suivre l'avancée de chacun sur les travaux en cours. On y décrit les tâches effectuées la veille et celles sur lesquelles on va travailler le jour même.
- Tous les mercredis après-midi, nous avons une réunion de backlog dans laquelle sont discutées, avec tous les membres de l'entreprise, la priorisation de nouveaux tickets dans le tableau de bord ainsi qu'un suivi des tickets en cours.

C'est lors des daily meetings que sont abordées, de manière orale, **les estimations de temps de réalisation de tâche** en fonction de la difficulté du ticket et du niveau de connaissance requis pour réaliser cette tâche.

Il est à la charge de **celui qui effectue le ticket** d'estimer **le temps de réalisation d'une tâche**. Le Chief Technical Officer (CTO) est simplement là pour confirmer, si cette estimation proposée est réaliste par rapport au ticket. Il n'y a donc pas de logiciel spécifique pour calculer efficacement la durée d'un projet ; tout se fait à l'oral, ce qui permet d'être plus flexible vis-à-vis des tâches à accomplir. Cette approche nous permet d'être plus **flexible** quant au **temps de développement d'un projet** et aux **difficultés rencontrées** tout au long du processus d'un projet.

Les réunions ne sont pas le seul moyen d'obtenir des informations ou de l'aide. Nous utilisons également les conversations sur Teams, que ce soit par message ou en vocal, pour obtenir des informations plus précises sur certains sujets et de l'aide si nous restons bloqués sur un ticket afin d'éviter de perdre trop de temps. Le chat permet des échanges rapides et informels, tandis que les appels vocaux sont utiles pour des discussions plus approfondies ou pour résoudre des problèmes complexes en temps réel.

Le partage d'écran est également un outil précieux lors de ces conversations, car il permet de montrer concrètement les réalisations et de s'assurer qu'elles sont conformes aux attentes. Cette approche favorise la collaboration et la résolution rapide des problèmes, tout en limitant les malentendus et les pertes de temps.

Les Difficultés Rencontrées

Comme pour tout projet de grande envergure, de **nombreuses difficultés** ont été rencontrées tout au long du processus de mise en place de ce projet.

Les principales difficultés rencontrées lors de la mise en place de ce projet ont été :

- **Phase de maquettage** : Il y a eu une certaine difficulté à trouver le bon design qui pourrait répondre aux besoins des clients. Il était particulièrement important de proposer une interface offrant une bonne expérience utilisateur et facile à prendre en main pour les utilisateurs car la page du tableau de bord est une des parties les plus importantes de l'application.

Pour cela, il a fallu analyser des designs similaires qui répondaient en partie à nos besoins sur le site **dribbble.com**, où de nombreux designers soumettent leurs créations en matière web design, pour inspirer d'autres designers dans leur travail de maquettage de pages web. De plus, différentes remarques ont été faites par d'autres membres de l'équipe de développement lors des daily meetings permettant de créer les maquettes les plus adéquates et satisfaisantes possible. Les retours sur Teams ont également été précieux, car chaque maquette est publiée sur un canal Teams pour obtenir l'avis de toute l'organisation, afin d'avoir un avis extérieur au métier du développement et de vérifier si les maquettes correspondent bien aux besoins.

- **Responsivité** : On a rencontré quelques difficultés au niveau de l'affichage responsive du tableau de bord, à la suite de retours de clients qui trouvaient que l'affichage avec un zoom de 125 % et de 150 % était trop serré sur le tableau de bord. Ces configurations sont très utilisées par certains utilisateurs, car Google Chrome, par exemple, applique un zoom par défaut de 125 %, ce qui rendait nécessaire l'adaptation à ce paramètre.

Pour résoudre ce problème, nous avons créé de nouvelles maquettes en tenant compte de ce paramètre, ainsi qu'une maquette spécialement conçue pour le mobile pour bien adapter le contenu en fonction de la plateforme de l'utilisateur.

- **Mise en place des données mensuelles et annuelles** : Au début du projet, il y a eu une réflexion sur la manière de calculer les données mensuelles et annuelles des utilisateurs.

Nous avons donc pris le temps, lors de réunions, afin de déterminer comment effectuer ces calculs et quelles architectures, il fallait mettre en place. Divers schémas d'architecture ont été mis en place et il a été décidé que les données mensuelles et annuelles seraient une moyenne des données quotidiennes des utilisateurs afin de représenter au mieux la consommation de données sur un mois ou une année donnée.

- **Adaptation des anciennes pages** : J'ai rencontré des difficultés lors de l'adaptation des anciennes pages du tableau de bord aux nouvelles

fonctionnalités. Il était difficile de déterminer quels composants des pages devaient être conservés, modifiés, ou recréés à partir de zéro en raison de la refonte.

Pour résoudre ce problème, j'ai sollicité des conseils auprès de mes collègues. Leur expertise m'a aidé à identifier clairement les éléments à conserver, à modifier ou à recréer.

- **Gestion des conflits** : En raison de la longue durée de développement de ce projet, plusieurs développements ont été intégrés pendant cette période, ce qui a conduit à gérer quelques conflits sur ma branche de développement.

Pour les résoudre, j'ai analysé les modifications apportées dans les nouvelles branches créées par l'équipe de développement, en essayant de les intégrer dans ma branche de développement. Lorsque j'avais du mal à gérer certains conflits complexes, j'ai demandé conseil aux autres membres de l'équipe.

- **Affichage des graphiques avec Chart.js** : J'ai rencontré des difficultés lors de l'affichage des graphiques sous forme de demi-cercle en utilisant la librairie Chart.js, ainsi que pour afficher les légendes en dehors de la div du graphique.

Pour résoudre ce problème, j'ai consulté la documentation de Chart.js et visionné des tutoriels sur YouTube, ce qui m'a permis de trouver une solution.

Un autre problème que j'ai rencontré concernait l'absence d'options intégrées dans Chart.js pour placer des éléments au centre d'un graphique en forme de donut.

Heureusement, Chart.js donne la possibilité de manipuler des éléments du graphique à l'aide de plugins. J'ai donc dû créer un plugin dans Chart.js, ce qui était assez complexe car je n'en avais jamais fait auparavant. J'ai également utilisé des vidéos YouTube et la documentation pour m'aider dans cette tâche.

Réalisation :

Outils utilisés :

Je vais maintenant vous présenter les divers outils que j'utilise au quotidien pour le développement de l'application web.

Environnement de développement :

Je travaille sous l'IDE **Visual Studio Code**, qui facilite grandement mon développement grâce aux nombreuses extensions disponibles dans son catalogue.

Maquettage des pages :

Pour la partie maquettage des pages, j'utilise le logiciel **Figma**, car il est facile à utiliser et propose de nombreux designs préconstruits qui nous inspirent pour la modélisation de nos futures pages.

Documentation

Pour la mise en place de la documentation technique, et d'utilisation, nous utilisons **Confluence**. Ce logiciel permet de créer des documentations collaboratives, facilitant ainsi une meilleure **coopération** entre les différents membres de l'équipe.

Voici les langages et frameworks que nous utilisons pour la partie **front-end** :



- **TypeScript** : Il s'agit d'un langage de programmation dérivé de JavaScript, qui se distingue par l'utilisation de types pour les fonctions et variables.



- **Vue.js** : Il s'agit d'un framework Javascript permettant de créer des applications web monopages.



- **CSS** : Il s'agit d'un langage de programmation utilisé pour gérer le style du site web.



- **Vuetify** : Il s'agit d'un framework basé sur Vue.js qui fournit un ensemble complet de composants d'interface utilisateur, facilite la création d'interfaces utilisateur réactives et esthétiques.

Pour la partie **back-end**, nous utilisons les outils suivants :



- **PostgreSQL** : Il s'agit d'un système de gestion de bases de données relationnelles et orientées objets.



- **Nest** : Il s'agit d'un framework Node.js qui facilite la création, l'appel et le test d'APIs grâce à une structure modulaire, garantissant une maintenabilité aisée.



- **PgAdmin** : Il s'agit d'un outil graphique de gestion de base de données qui simplifie la gestion et l'administration des bases de données, ce qui est particulièrement utile pour les configurations utilisant PostgreSQL.



- **Prisma** : Il s'agit d'un Objet Relational Mapping (ORM) moderne qui facilite les interactions avec la base de données en offrant un typage fort et une interface intuitive.

Extensions utilisées pour améliorer la qualité du code :

Afin d'améliorer la qualité du code produit lors du développement de nos projets, nous utilisons diverses extensions qui nous permettent d'améliorer la qualité du code :

- **SonarLint** : Extension qui applique les bonnes pratiques de développement en affichant des avertissements sur les problèmes de qualité de code.

- **Prettier** : Extension qui formate automatiquement des fichiers.
- **EsLint** : Extension qui vérifie la qualité du code et les problèmes de style de codage (comme la fermeture de balise par exemple).

Guide des bonnes pratiques :

Au sein de TechupClimate, un guide des bonnes pratiques a été implémenté par certains membres de l'équipe de développement afin de garantir la qualité et la cohérence du code. Ce guide couvre différents aspects du développement :

- **Gestion des versions avec Git** : Nous utilisons le workflow GitFlow pour structurer le développement en branches dédiées, comme celles dédiées aux nouvelles fonctionnalités, aux correctifs et aux versions de production. Les détails sur la gestion des versions avec Git seront abordés plus en détail dans une section ultérieure de ce rapport.
- **Développement en TypeScript** : Nous respectons les directives du Google TypeScript Style Guide qui propose de nombreuses règles à respecter dans le code pour garantir une cohérence et la qualité du code produit. Par exemple, une interface doit commencer par la lettre I, comme dans l'interface IUser afin de correspondre à la convention de nommage. De plus, nous appliquons le modèle early return pattern pour simplifier le code, dans lequel le retour d'une fonction est appelé si aucune des conditions n'est remplie évitant ainsi des niveaux de conditions imbriqués inutiles.
- **Développement en Vue.js** : Pour assurer une cohérence dans le développement des interfaces utilisateur, nous suivons des conventions strictes de nommage et de structure des composants. Par exemple, un composant générique sera nommé avec le préfixe « Base », comme « BaseButton » pour indiquer qu'il peut être réutilisé à travers l'application. Dans le même type, les pages sont nommées avec le suffixe « View », comme « DashboardView », pour indiquer qu'il s'agit d'une vue complète.
- **Gestion de la base de données avec Prisma** : Lors d'une mise à jour de la base de données, il est important de mettre à jour le schéma Prisma. Par exemple, lors de l'ajout d'une nouvelle table dans la base de données, nous mettons à jour le schéma Prisma avec la commande `npm run prisma : pull` puis nous formatons ce schéma et régénérons le client Prisma pour garantir que les nouvelles modifications sont correctement intégrées dans notre code.
- **Commentaires dans le code** : Nous évitons d'ajouter des commentaires dans le code, car les fonctions doivent être intuitives par leur nom, leur complexité et leur champ d'action. Ajouter des commentaires pourrait rendre le code moins lisible.

Ce guide des bonnes pratiques nous permet de travailler efficacement en équipe, en assurant que chaque développeur suive des normes communes afin de produire un code de qualité.

Les Tests Unitaires

Toutes les **fonctions créées** dans l'application doivent être minutieusement **testées** avec des **tests unitaires**, afin de s'assurer que toutes les conditions sont bien vérifiées et que la fonction réponde **correctement aux attentes**.

Pour mettre en place ces différents tests unitaires nous utilisons le framework **Vitest**, un outil idéal pour tester les fonctions de manière rapide et légère dans les applications JavaScript et TypeScript.

Au sein de Carbonscore, les tests unitaires doivent respecter le **pattern AAA** dans la structure du test :

- **Arrange** : Préparation de l'environnement de test, initialisation des objets nécessaires.
- **Act** : Exécution de la fonction à tester.
- **Assert** : Vérification que les résultats obtenus sont conformes aux attentes.

L'utilisation de ce pattern permet de rendre les tests plus lisibles et faciles à maintenir.

Nous appliquons également le **Test-Driven Development** (TDD) pour les tests unitaires. Le principe du TDD consiste à **écrire les tests avant d'écrire le code** d'une fonctionnalité.

- Le processus commence par la rédaction du test unitaire avant même de commencer à développer la fonctionnalité. Ce test définit les attentes de la fonctionnalité.
- Ensuite, le test est exécuté pour s'assurer qu'il échoue, ce qui est normal puisque la fonctionnalité n'a pas encore été implémentée.
- Après cette étape, le code minimum nécessaire pour faire passer le test est écrit.
- Une fois le code écrit, les tests sont réexécutés pour vérifier que le nouveau code passe le test.
- Enfin, le code est amélioré ou refactorisé tout en s'assurant que les tests passent toujours.

Le TDD assure que le code est testé de manière exhaustive dès le départ, réduisant les bugs et améliorant la qualité du code.

La gestion du Versionning

Pour la **gestion du versionning** de nos applications, nous utilisons **GitHub**.

L'application du workflow **GitFlow**, nous permet de structurer efficacement le développement et la gestion des versions de notre application web.

Ce workflow s'articule autour de plusieurs branches clés :

- **Master** : Cette branche représente l'environnement de production, celle déployée chez nos clients. Seuls les versions stables et prêtes à être déployés sont fusionnées dans cette branche.
- **Develop** : Cette branche est utilisée pour les développements en cours. Toutes les fonctionnalités sont fusionnées dans cette branche après avoir été validées.
- **Feature** : Chaque fonctionnalité est développée dans une branche spécifique, issue de la branche Develop.

Afin de pouvoir **identifier facilement la branche**, il y a une nomenclature à respecter : feature/numeroticket-descriptionduticket. Le numéro du ticket correspond à celui inscrit dans le **ticket Jira**.

- **Release** : Une fois que plusieurs fonctionnalités ont été intégrées et testées, une branche Release est créée à partir de develop pour récupérer les dernières modifications. Après validation, elle est fusionnée dans Master et Develop.
- **Hotfix** : En cas de problème critique en production, une branche Hotfix est créée à partir de Master pour corriger rapidement le bug. Cette branche est ensuite fusionnée dans Master et Develop pour que la correction soit intégrée dans le flux de développement.

Workflow des Commit et Pull Request

Ensuite, après la réalisation de la fonctionnalité, il est temps **d'émettre les changements sur la branche**.

Chaque commit doit suivre une nomenclature stricte pour faciliter sa traçabilité :

[type] numeroticket-description

Le **type de commit** peut varier selon le **type de changement** effectué sur notre branche. Par exemple, add sera utilisé pour la mise en place d'une nouvelle fonctionnalité, edit pour sa modification, fix pour la correction d'un bug ou test pour la mise en place de tests unitaires.

Avant de pouvoir fusionner notre branche, il est nécessaire **qu'au moins un collaborateur de l'équipe** approuve la pull request. Cela garantit une revue de code par des collègues, contribuant ainsi à produire un code de qualité.

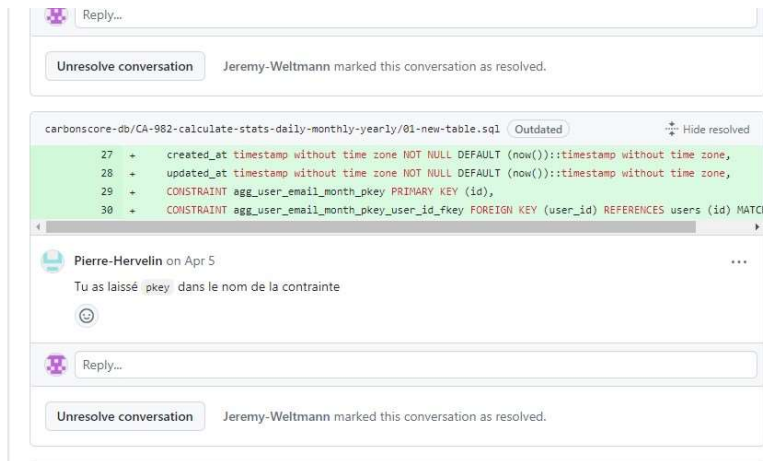


Figure : Revue de code sur GitHub

Automatisation CI/CD avec GitHub Actions

Dès qu'un changement est mis en place sur une branche, un processus automatisé de **CI/CD** est déclenché grâce à **GitHub Actions**. Ce processus **exécute les tests unitaires de l'application** pour s'assurer que les nouvelles fonctionnalités sont conformes aux attentes et qu'il n'y a pas de **régression**, par rapport aux versions précédentes de l'application. Il automatise également la **création de tags**, utilisés pour déployer la branche sur un **environnement de test**.

Fonctionnement Général de l'entreprise

Stratégie générale de l'entreprise

Dans l'entreprise, lors du développement d'une nouvelle fonctionnalité pour l'application web, celle-ci est **simplifiée au maximum** dans un premier temps. Cette approche permet **d'évaluer** rapidement si la fonctionnalité développée **correspond aux attentes des clients** sans **investir trop de ressources initialement**.

Si la fonctionnalité rencontre un retour positif, elle sera ensuite étendue, permettant ainsi une **évolution continue du produit** tout en **minimisant les risques de développement**.

Démarche Qualité :

Afin de garantir l'intégrité et la qualité de la solution proposée par l'entreprise, les projets passent par plusieurs phases bien définies :

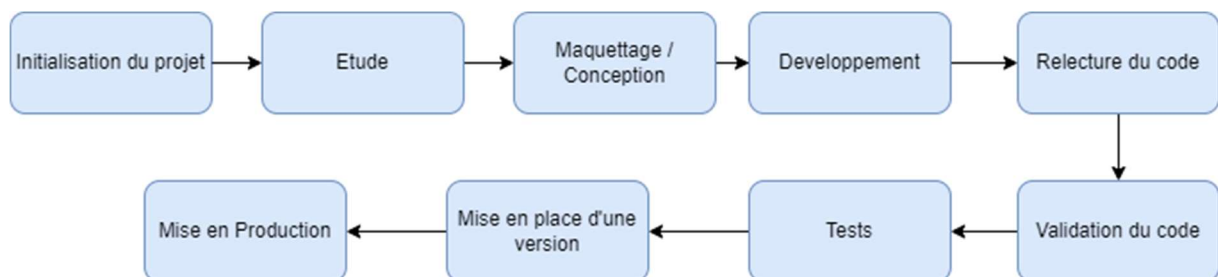


Figure : Schéma des différentes phases de développement d'un projet

1) Initialisation du projet :

Le développement commence par une demande spécifique d'un client ou par une fonctionnalité souhaitée par l'équipe de développement. Ces demandes spécifiques sont formulées par le **biais de mails envoyés à l'organisation** ou lors de **réunions dédiées**, où les discussions portent sur le logiciel et les améliorations potentielles.

De temps en temps, des **réunions brainstorming** sont également organisées pour permettre à **chaque membre de l'équipe de proposer des idées d'amélioration** pour le logiciel. Pour cela, nous utilisons Microsoft Teams, avec l'outil **WhiteBoard**, permettant de schématiser des idées en temps réel et de manière partagée.

2) Étude de faisabilité :

Une étude de faisabilité est ensuite réalisée afin d'évaluer la pertinence et la possibilité de mettre en œuvre la demande. Cette étape comprend la **rédaction d'une documentation** détaillant les besoins des clients ainsi que la découpe des tâches nécessaires à la réalisation du projet. Cette découpe permet de structurer le développement en plusieurs tickets, facilitant ainsi la gestion et l'attribution des différents tickets.

3) Maquettage et Conception

Les besoins identifiés sont ensuite **traduits en maquettes**, qui sont discutées lors des **daily meetings** avec l'ensemble de l'équipe de développement pour s'assurer qu'elles correspondent bien aux attentes. Si nécessaire, elles sont retravaillées. En parallèle, si le projet comporte des éléments de **back-end**, des **schémas d'architectures** sont créés pour définir les changements dans la base de données ainsi que l'architecture des routes à mettre en place. Cela permet de réfléchir en amont aux concepts à implémenter, évitant de devoir redévelopper certaines parties qui pourraient ne pas avoir répondu aux attentes, permettant ainsi de **gagner du temps de développement**.

4) Phase de développement

Une fois les maquettes validées et les schémas d'architecture en place, le développement peut commencer. Les **tâches** sont **réparties** entre les membres de l'équipe en fonction des tickets créés lors de l'étape de faisabilité. Le développement est ensuite réalisé en suivant les bonnes pratiques établies par l'entreprise.

Des points sont effectués quotidiennement lors des daily meetings pour suivre l'avancée du projet et identifier rapidement d'éventuelles difficultés rencontrées par les développeurs. Cela permet d'ajuster les priorités ou de fournir de l'aide lorsque nécessaire pour maintenir le projet sur la bonne voie.

5) Revue de Code et Tests :

À la fin du développement, une pull request est créée pour soumettre les modifications et **recueillir des retours** sur le code mis en place. Ensuite, des tests unitaires et des tests de non-régression sont exécutés avec GitHub Actions. Si les tests et le code sont validés, alors ils sont déployés sur un environnement et des tests de fonctionnalité

sont effectués, la fonctionnalité est déployée sur un environnement de test pour tester toutes les configurations possibles.

Tout d'abord, celui qui a mis en place la fonctionnalité doit **tester toutes les configuration de sa fonctionnalité** afin de faciliter le travail de ceux qui vont tester la fonctionnalité afin de diagnostiquer les bugs faciles à voir en premier, et en demandant aux autres développeurs de tester seulement si ses tests ont bien réussi.

Plusieurs environnements de test sont disponibles, permettant de tester les fonctionnalités sous différentes configurations pour s'assurer qu'elles répondent aux besoins des clients sans introduire de bugs.

C'est le **développeur** ayant réalisé la fonctionnalité qui définit la **plage de tests** pour valider que la fonctionnalité est conforme aux besoins.

Une **discussion** est créée sur **Teams** à chaque fois qu'une fonctionnalité est terminée, nous permettant d'obtenir des retours et des conseils sur certains éléments. Si les retours sont positifs et que les tests sont réussis, la fonctionnalité est mise en production. Si des remarques négatives sont émises, elles permettent d'identifier les problèmes, et de mettre en place des actions correctrices.

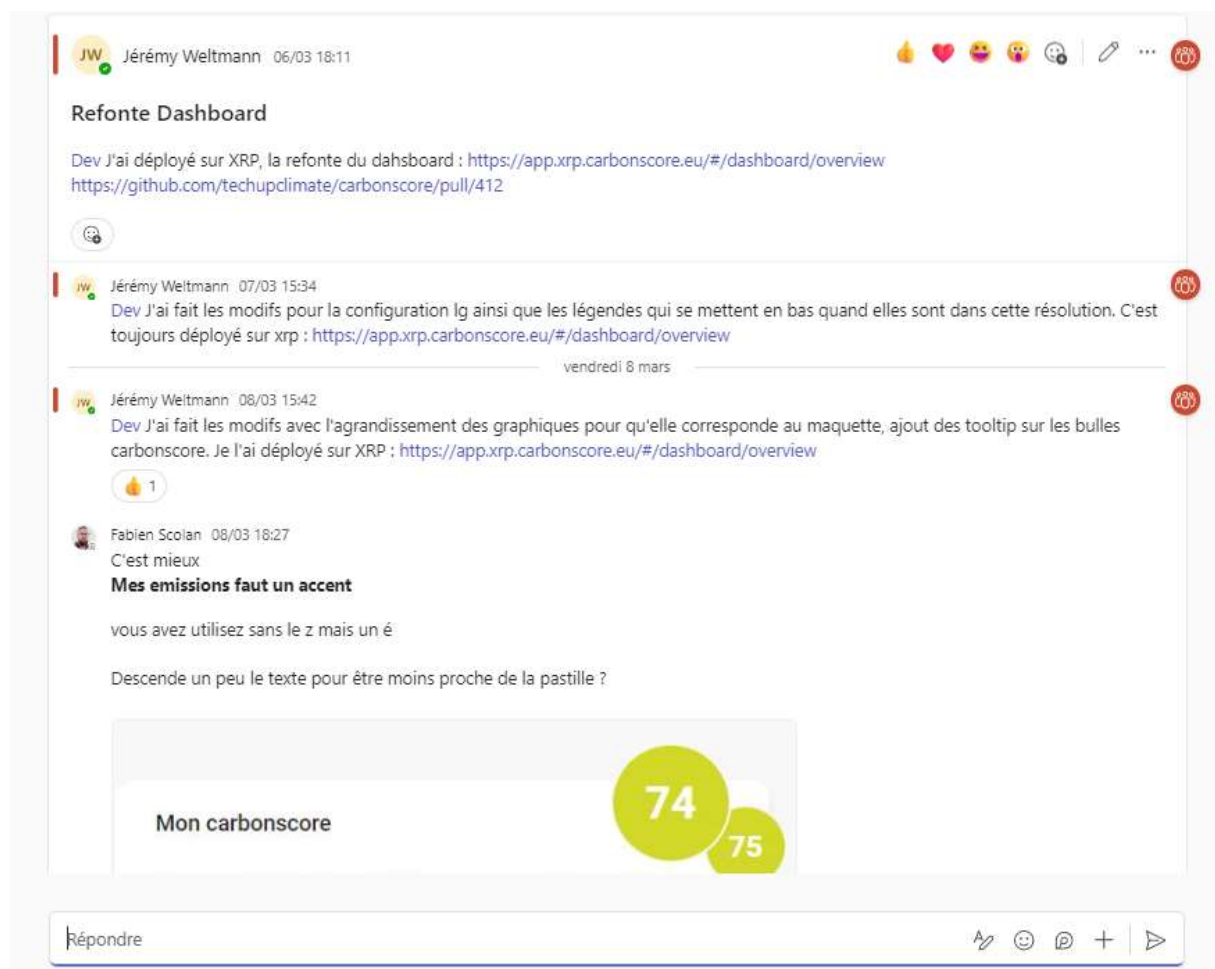


Figure : Discussion autour d'un projet sur Microsoft Teams

6) Mise en Production et Cycle de Vie Continu

Une fois le code validé et les tests réussis, une nouvelle version de l'application est déployée en production.

Toutefois, le projet ne s'arrête pas là ! Des retours peuvent encore être faits par les clients ou les membres de l'équipe, entraînant la création de tickets pour corriger d'éventuels bugs ou pour apporter des modifications supplémentaires.

Bilan du projet

Globalement, le projet est considéré comme une **réussite**, car il a été **déployé** et **mis en production** avec succès. Les clients ont **globalement apprécié** la refonte du tableau de bord, ce qui témoigne de la pertinence des améliorations apportées.

Cependant, la mise en production d'une fonctionnalité ne marque pas la fin du travail. Des **demandes d'amélioration ou de correction** peuvent être formulées à la suite des **retours des clients**, qu'il s'agisse d'optimiser certaines fonctionnalités ou de corriger de simples fautes d'orthographe. Ce suivi continu est essentiel pour **garantir la satisfaction des utilisateurs** et **l'amélioration constante du produit**.

L'estimation de la taille d'un projet se fait lors **de l'étude de la fonctionnalité au lancement d'un projet**. Initialement prévu pour une durée de **trois mois**, le projet a finalement nécessité **quatre mois** pour être finalisé.

Ce dépassement de délai est principalement dû à des **difficultés rencontrées dans la reproduction de certaines maquettes complexes**, nécessitant une **étude approfondie de la documentation de Chart.js**, ainsi qu'à une **répartition des tâches** qui aurait pu être **optimisée**. Ce dépassement a été détecté **au cours du développement du projet** durant les réunions quotidiennes avec l'équipe de développement, ce qui a nécessité **l'apport d'un développeur supplémentaire** pour finaliser certains éléments du projet plus rapidement.

Cependant, ce problème est aussi dû à la **petite structure de l'entreprise**, car les autres développeurs étaient **engagés sur d'autres tâches aussi importantes**, ce qui ne permettait pas de **découper plus les tâches**.

Les principales leçons à tirer de cette expérience sont :

- L'importance **d'évaluer la faisabilité des maquettes** dès le **début du projet**, en identifiant les **points techniques** potentiellement bloquants et en mettant en place les **solutions adéquates**.
- La nécessité d'une **répartition plus fine des tâches**, en séparant clairement **les responsabilités entre le front-end et le back-end**, afin **d'éviter les chevauchements et les retards**.

En appliquant ces leçons, les futurs projets pourront être menés de manière plus efficace, en respectant les délais et en garantissant la qualité du produit final.

Conclusion

Pour conclure, mon **expérience** en tant qu'Apprenti Développeur Full Stack au sein de l'entreprise **TechUpClimate** a été extrêmement **enrichissante**. J'ai eu l'opportunité de découvrir tous les **aspects du métier de développeur full stack**, grâce à **des tâches très variées**, qui m'ont permis de me former à chaque facette de ce rôle. J'ai travaillé sur des tâches de web design, de développement front-end et back-end et j'ai pu m'initier aux méthodes de gestion de projet. Cette diversité de missions m'a également permis de me familiariser avec des **technologies modernes** telles que Vue.js, TypeScript, PostgreSQL, et Nest, qui me seront précieuses dans la suite de ma carrière professionnelle. J'ai également appris à **travailler en équipe** dans le domaine du développement web, en adoptant des méthodes de travail collaboratives qui me seront très utiles à l'avenir.

J'ai évolué dans un environnement de travail bienveillant, où j'ai beaucoup appris au cours de ces deux années, grâce à l'ensemble de l'équipe de développement qui m'ont prodigué de précieux conseils.