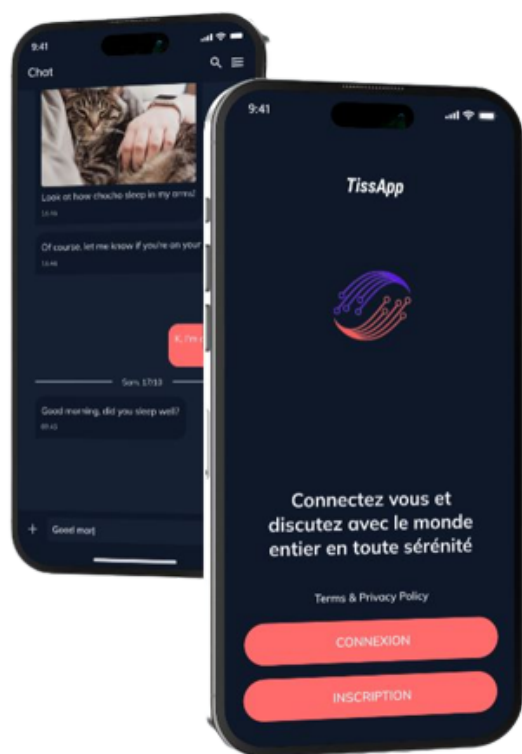


PRE Tchèssi

20 rue Francis de Pressensé
13001 Marseille



“TissApp”

Une application de chat conçue pour révolutionner la façon dont les employés communiquent et gèrent leurs dépenses..

DOSSIER DE PROJET PROFESSIONNEL

SOMMAIRE

❖ Introduction	6
Présentation personnelle	7
Présentation du projet en anglais	8
Compétences couvertes par le projet	10
❖ Organisation et cahier de charges	12
Analyse de l'existant	13
Les utilisateurs du projet	14
Les fonctionnalités attendues	15
> Application mobile	15
> Application web (Panel-admin)	19
Contexte technique	22
❖ Conception du projet	23
Choix du développement	24
> Choix du langage	24
> Choix des Frameworks	26
> Logiciels et autres outils	28
Organisation du projet	30

Architecture du projet	31
❖ Conception front-end de l'application	33
Arborescence du projet	34
Charte graphique	35
Wireframe	36
Maquettage	36
❖ Conception backend de l'application	37
la base de données	38
> Mise en place de la base de données	38
> Conception de la base de données	39
> Model conceptuel de base de données	42
> Modèle logique de données	44
> Modèle physique de données	46
❖ Développement du backend de l'application	47
Organisation	48
Arborescence	49
Fonctionnement de l'API	50
Middleware	52

Routage	53
Controller	56
Service	57
Model	58
Sécurité	60
> Credential stuffing : vol du login et password	60
> Chiffrement des données sensibles	61
> Json Web Token (JWT)	61
> Gestion des droits	63
Exemple de problématique rencontrée	65
Recherche anglophone	66
Exemple d'envoi de données avec images	67
Documentation	73
Tests	74
❖ Développement du front-end de l'application	76
Arborescence	77
Pages et composants	79
Sécurité	80
Problématique rencontrées	81

Exemple navigation imbriquées	83
Exemple de formulaire de mise à jour du profil	85
❖ Conception de l'espace administrateur	88
Conception de la partie administration	89
User Story	89
Choix du langage et Framework	90
Conception du front-end de l'application du site web.	91
> Charte graphique	91
> Maquettage	91
Conception du backend du site web	94
❖ Conclusion	95
Annexe	96

Introduction

Présentation personnelle

Bonjour à tous, je me nomme Tchèssi Pre. J'ai 42 ans , je suis ravi de vous présenter TissApp un nouvel outil d'affaires innovant, notre propre application de chat conçue pour révolutionner la façon dont nos employés communiquent et gèrent leurs dépenses. Cette application sert de plateforme complète pour l'interaction, offrant aux employés un espace pour échanger des idées, discuter de projets et collaborer de manière efficace.

Elle favorise non seulement une communication ouverte, mais renforce également le sens de la communauté au sein de l'entreprise. En plus de la fonction de chat, l'application offre une fonctionnalité unique de partage de frais. Les employés peuvent facilement télécharger des images de leurs reçus ou factures, simplifiant ainsi le processus de remboursement des dépenses.

En combinant communication et gestion des dépenses en une seule application facile à utiliser, nous visons à augmenter la productivité et l'efficacité de notre entreprise. Je suis convaincu que cette application sera un outil précieux pour notre équipe.

Présentation du projet en anglais

Hello everyone, my name is Tchessi Pre, and I am a software developer. Today, I am delighted to present to you a unique application I have created specifically for businesses - TissApp. This platform is designed to streamline communication within an enterprise, improving team collaboration and productivity.

The name "TissApp" is derived from the initials of four individuals - Tchessi, Ismail, Samir, and Salim. Their combined efforts have resulted in an application that significantly enhances interdepartmental communication. The core feature of TissApp is its chat functionality. Employees can engage in discussions in a general chat room, fostering a sense of community and facilitating knowledge exchange. Not only can they share thoughts and ideas, but they can also post images of receipts for expense tracking. This provides a reliable and convenient way for staff members to manage their expenses within the organization.

Finally, the admin panel was built using Material-UI, a popular React UI framework that delivers an excellent user interface with Google's Material Design guidelines. The admin panel serves as a command center, allowing easy monitoring and management of the entire application. TissApp represents a significant step forward in how businesses can enhance communication and productivity. The use of modern technologies has resulted in an application that is both powerful and easy to use. I am incredibly proud to introduce it to you today, and I am confident that TissApp will be a valuable tool for enterprises.

In the creation of TissApp, I employed several cutting-edge technologies. The application's front end was developed using React Native, a popular framework for building native mobile apps using JavaScript and React. This choice guarantees a seamless and responsive user experience across different devices. The back end of the application was designed with Node.js and Express.js, providing the robust and efficient server-side capabilities that TissApp needs to ensure smooth operation. These technologies offer excellent scalability, a crucial aspect for any business application. Additionally, Socket.IO was used to enable real-time, bidirectional

communication between web clients and servers. This ensures instantaneous message delivery within the chat, making communication more efficient and fluid.

Compétences couvertes par le projet

La réalisation de ce projet nécessite une variété de compétences techniques et non techniques pour garantir son succès. Voici quelques-unes des compétences clés que ce projet a couvertes

- Maquetter une application
- Développer des composants d'accès aux données
- Développer la partie frontend d'une interface utilisateur web
- Développer la partie backend d'une interface utilisateur web
- Concevoir une base de données
- Mettre en place une base de données
- Développer des composants dans le langage d'une base de données
- Concevoir une application - Développer des composants métier
- Construire une application organisée en couches - Développer une application mobile
- Préparer et exécuter les plans de tests d'une application
- Préparer et exécuter le déploiement d'une application
- **Développement Full-Stack** : La création de l'application nécessite une connaissance approfondie du développement frontend et backend. Les technologies utilisées comprennent React Native pour le frontend, et Node.js, Express.js et Socket.IO pour le backend.
- **Développement UI/UX** : Pour assurer une expérience utilisateur optimale, des compétences en design d'interface utilisateur (UI) et en expérience utilisateur (UX) sont nécessaires. Le panel administrateur a été construit avec Material-UI, nécessitant une bonne connaissance de cette bibliothèque.
- **Gestion de bases de données** : La capacité de travailler avec des bases de données pour stocker et récupérer des informations est également importante. Cela comprend la conception de modèles de données, la création de requêtes et l'optimisation de la performance de la base de données.

- **Gestion de projet** : Outre les compétences techniques, des compétences en gestion de projet sont également nécessaires pour planifier, coordonner et superviser le projet du début à la fin.
- **Communication et collaboration** : Comme le projet implique une équipe (Tchessi, Ismail, Samir, et Salim), il est important d'avoir des compétences en communication et en collaboration pour travailler efficacement ensemble.
- **Connaissance du domaine d'entreprise** : Une compréhension claire des besoins de l'entreprise, en particulier en ce qui concerne la communication et la gestion des dépenses, est nécessaire pour construire une application qui répond efficacement à ces besoins.

En somme, ce projet a permis de couvrir un large éventail de compétences, rendant l'équipe bien équipée pour relever des défis similaires à l'avenir.

Organisation et cahier de charges

Analyse de l'existant

Avant la mise en place de notre nouvelle application de chat, les employés utilisaient probablement une combinaison de différentes plateformes pour communiquer entre eux et pour soumettre leur note frais. Cette situation pourrait entraîner des problèmes de communication et des incohérences dans la gestion des dépenses. Voici une analyse plus détaillée de l'existant :

Communication : Les employés utilisaient peut-être des emails, des appels téléphoniques, des messageries instantanées ou des réunions en face à face pour communiquer entre eux. Bien que ces méthodes soient efficaces, elles peuvent manquer de cohérence et ne pas favoriser une communication fluide et rapide.

Gestion des frais : Les employés devaient probablement conserver leurs reçus physiques et les soumettre manuellement à leur responsable ou au département de la comptabilité. Ce processus peut être laborieux, sujet à des erreurs, et prendre beaucoup de temps.

Suivi des dépenses : Sans une plateforme centralisée pour suivre les dépenses, les responsables et le département de la comptabilité peuvent avoir du mal à garder une trace précise des remboursements de frais.

En somme, l'absence d'une plateforme unifiée pour la communication et la gestion des dépenses a pu entraîner des inefficacités et des retards. Notre nouvelle application de chat, **TissApp**, a été conçue pour surmonter ces défis en offrant une solution centralisée et intuitive pour les besoins de communication et de gestion des dépenses de l'entreprise.

Les utilisateurs du projet

Les utilisateurs de ce projet seront principalement les employés de l'entreprise. Ils bénéficieront grandement de cette plateforme qui leur permettra de communiquer facilement entre eux, partageant idées et informations en temps réel.

Les avantages de l'application ne se limitent pas à la communication interne. Les employés pourront également partager leurs dépenses professionnelles de manière transparente et efficace. Ce système simplifié facilitera la gestion des remboursements, contribuant à un environnement de travail plus organisé et productif.

En plus des employés, les responsables de l'entreprise pourront également utiliser l'application. Ils pourront superviser les communications, s'assurer que les informations importantes sont partagées efficacement, et gérer le processus de remboursement des dépenses.

Enfin, l'équipe des ressources humaines et le département de la comptabilité pourront également bénéficier de cette application. Ils auront un aperçu clair et organisé des dépenses des employés, ce qui facilitera le suivi et l'approbation des remboursements.

En résumé, cette application est destinée à tous ceux qui sont impliqués dans la communication interne et la gestion des dépenses au sein de l'entreprise. Elle vise à rendre ces processus plus simples, plus clairs et plus efficaces.

Les fonctionnalités attendues:

→ APPLICATION MOBILE

◆ Page d'accueil

Authentification : La page d'accueil fournit une fonctionnalité d'authentification pour que les utilisateurs puissent se connecter à leur compte. Ceci inclut la connexion par email et mot de passe.

Inscription : Pour les nouveaux utilisateurs, il y a une option pour s'inscrire. Cela pourrait nécessiter de fournir des détails tels que le nom, prénom, l'email, et la création d'un mot de passe sécurisé et la confirmation de mot de passe.

La page d'accueil offre un environnement sécurisé pour la connexion et l'inscription, afin de protéger les données sensibles de l'utilisateur.

◆ Page de connexion

La page de connexion contient un formulaire que l'utilisateur peut remplir pour se connecter. Le formulaire inclut des champs pour l'adresse e-mail et le mot de passe. Si l'utilisateur n'a pas encore de compte, il a la possibilité de s'inscrire en cliquant sur le bouton "Inscription". Cela doit rediriger l'utilisateur vers la page d'inscription. Si l'utilisateur entre une adresse e-mail ou un mot de passe incorrect, il va voir un message d'erreur sur la page de connexion, lui indiquant que les informations d'identification sont incorrectes.

◆ Page d'inscription

La page d'inscription est une partie cruciale de l'application. Voici les fonctionnalités qu'elle pourrait inclure :

Formulaires d'inscription : Les utilisateurs devront remplir des formulaires avec des informations de base, comme leur nom, prénom, leur adresse e-mail, un mot de passe et une confirmation de mot de passe. La collecte de ces informations est nécessaire pour la création d'un compte.

Validation de l'adresse e-mail : Afin de vérifier l'authenticité des informations fournies, si l'utilisateur entre une adresse e-mail qui est déjà associée à un compte existant, il verra un message d'erreur sur la page d'inscription, l'informant que cette adresse e-mail est déjà utilisée.

Sécurité du mot de passe : Si l'utilisateur entre un mot de passe qui ne répond pas aux exigences de sécurité minimales, il doit voir un message d'erreur sur la page d'inscription, lui indiquant les exigences pour le mot de passe (par exemple, 10 caractères minimum, une lettre majuscule, une lettre minuscule et un caractère spécial) Une fois l'inscription réussie, l'utilisateur est redirigé vers la page de confirmation et peut se connecter à son nouveau compte.

Bouton d'inscription : Une fois que tous les champs nécessaires ont été remplis et que les conditions d'utilisation ont été acceptées, l'utilisateur peut cliquer sur le bouton d'inscription pour finaliser la création de son compte.

Message de confirmation : Après l'inscription, un message de confirmation peut être affiché pour informer l'utilisateur que son compte a été créé avec succès. Les utilisateurs sont redirigés vers la page de connexion où ils pourront se connecter avec leurs nouveaux identifiants.

Ces fonctionnalités sont destinées à garantir une inscription simple et sécurisée pour les nouveaux utilisateurs de l'application.

◆ Page de contacts

La page de contact affiche la liste des utilisateurs et une barre de recherche pour trouver un utilisateur spécifique. La liste des contacts avec leur photo de profil , leur nom d'utilisateur et leur statut en ligne. Possibilité de faire défiler la liste des contacts pour voir tous les contacts. Cette page permettra aux utilisateurs de trouver rapidement et facilement des contacts dans leur liste d'amis en utilisant la barre de recherche.

Les utilisateurs pourront voir qui parmi leurs contacts est en ligne ou hors ligne. Cela peut aider à savoir quand s'attendre à une réponse rapide. En cliquant sur un contact, ils pourront ouvrir la conversation générale et commencer à discuter en temps réel.

◆ Page de chat

La page de chat offre aux utilisateurs une interface intuitive pour communiquer facilement avec d'autres utilisateurs en temps réel en envoyant des messages texte, des photos et autres médias. Les messages seront organisés par ordre chronologique, ce qui permettra aux utilisateurs de suivre facilement la conversation. La barre de message sera facilement accessible et permettra aux utilisateurs de taper un message rapidement et simplement. Un bouton pour envoyer une photo ou prendre une photo sera également disponible pour permettre aux utilisateurs de partager des images avec leurs correspondants. Ce bouton leur permettra de prendre une photo directement depuis l'application ou de sélectionner une photo existante dans leur galerie de photos.

◆ Page de discussion

Cette page présente tous les utilisateurs avec leur photo de profil , le nom et prénom et leur dernier message envoyé.

◆ Page de profil

La page de profil permettra à l'utilisateur connecté de visualiser son propre profil et de le personnaliser en fonction de ses préférences. Il pourra ajouter ou modifier sa photo de profil, son nom complet et toute autre information qu'il souhaite partager avec les autres utilisateurs. Un bouton de retour sera disponible pour permettre à l'utilisateur de retourner à la liste des conversations. Il aura également la possibilité de cliquer sur sa photo de profil pour la changer. Un bouton pour sauvegarder les modifications sera disponible pour permettre à l'utilisateur de confirmer les modifications qu'il a apportées à son profil. Il pourra également se déconnecter de l'application à partir de cette page en cliquant sur un bouton de déconnexion.

En offrant ces fonctionnalités sur la page de profil, l'application aide à favoriser la communication et la transparence au sein de l'entreprise, tout en respectant la confidentialité et la sécurité de l'utilisateur.

→ APPLICATION WEB (Panel-admin)

L'application web, en particulier le panel administrateur, est un élément crucial pour la gestion et le contrôle de l'application de chat d'entreprise. Le panel d'administration sera accessible uniquement aux administrateurs de l'application et offrira une vue d'ensemble de l'application pour assurer la sécurité et la qualité de l'expérience de chat pour tous les utilisateurs. Voici les fonctionnalités qu'il inclut :

◆ Page de connexion

Cette page permet la connexion à l'espace administrateur. Le panel d'administration sera une interface réservée aux administrateurs de l'application de chat mobile. Cette interface permettra aux administrateurs de gérer les utilisateurs, les conversations et les paramètres de l'application.

◆ Une page de gestion des utilisateurs

Cette page affiche une liste complète de tous les utilisateurs de l'application. Les administrateurs peuvent avoir des informations détaillées sur chaque utilisateur et peuvent avoir la possibilité de modérer les comptes si nécessaire. Elle contient une barre de recherche pour rechercher un utilisateur grâce à son nom et son prénom. Elle affiche dans un tableau le nom, prénom, email, le rôle, la date de création, la date de mise à jour et un bouton supprimer, modifier et aussi une pagination.

◆ le tableau de bord

Le tableau de bord est le cœur de l'interface d'administration. Il affiche des informations clés sur l'état de l'application, y compris le nombre total d'utilisateurs, le nombre de messages échangés, le nombre d'administrateurs, et le nombre de personnes connectées actuellement. Il présente également les quatre derniers messages envoyés, ce qui permet aux administrateurs de surveiller l'activité de l'application en temps réel.

◆ Page de chat

Gestion des conversations : Les administrateurs pourront accéder à toutes les conversations en cours et examiner les messages pour repérer les comportements abusifs ou les violations des règles d'utilisation de l'application. Ils pourront également supprimer des conversations entières si nécessaire. En somme, le panel d'administration offrira aux administrateurs une vue d'ensemble de l'application et leur permettra de gérer efficacement les utilisateurs, les conversations et les paramètres pour assurer la sécurité et la qualité de l'expérience de chat pour tous les utilisateurs.

◆ Page de profil

La page de profil permet aux administrateurs de gérer leurs informations personnelles. Ils peuvent modifier leurs informations de profil, y compris leur nom, prénom et email.

◆ Modal d'inscription d'un nouvel utilisateur

Sur la page qui affiche les utilisateurs, j'ai intégré un bouton pour ajouter un utilisateur qui ouvre un modal qui facilite l'inscription de nouveaux utilisateurs. En cliquant sur un bouton, les administrateurs peuvent faire apparaître ce modal qui contient un formulaire d'inscription. Ce formulaire comprend des champs pour les informations nécessaires comme le nom, prénom, l'email et le mot de passe. Une fois les informations saisies et validées, un nouvel utilisateur est créé dans la base de données. Cela offre aux administrateurs un moyen facile et rapide d'ajouter des nouveaux utilisateurs à l'application.

◆ Modal d'envoi de message dans le chat général

De plus, j'ai mis en place un modal permettant aux administrateurs d'envoyer un message dans le chat général. Ce modal contient un champ où les administrateurs peuvent saisir leur message, un bouton pour télécharger une image. Une fois le message rédigé, ils peuvent cliquer sur un bouton "Envoyer" pour publier le message. Ce message sera alors visible par tous les utilisateurs dans le chat général. Cela offre un moyen efficace pour les administrateurs de communiquer des informations importantes ou des annonces à tous les utilisateurs simultanément.

Contexte technique

Pour la création de l'application TissApp, plusieurs technologies ont été utilisées afin de garantir une performance optimale et une utilisation conviviale. Voici le contexte technique :

Pour le développement de mon application j'ai utilisé le Framework **React Native** pour créer mon application native pour Android et iOS. **Node.js** et **Express** ont été utilisés pour créer un serveur d'application pour gérer les données des utilisateurs.

J'ai utilisé Socket.io pour la gestion des communications en temps réel entre le serveur et les clients. Socket.io permet d'établir une connexion bidirectionnelle entre le serveur et le client, ce qui facilite l'envoi et la réception de messages en temps réel.

Enfin, pour créer le panel admin, j'ai utilisé le Framework **Matériel-UI** qui est basé sur React.

En utilisant ces technologies, l'application TissApp offre une expérience utilisateur de haute qualité, avec une interface réactive, des communications en temps réel, et une gestion efficace des données côté serveur. De plus, l'utilisation de JavaScript à la fois pour le front-end et le back-end facilite le développement et la maintenance de l'application.

Conception du projet

Choix du développement

La conception du projet TissApp a impliqué des choix réfléchis concernant le développement et le choix des langages. Voici une explication de ces choix :

→ Choix du langage



J'ai choisi de développer ce projet en utilisant exclusivement JavaScript qui est pris en charge par tous les navigateurs web modernes, ce qui en fait un choix idéal pour le développement d'applications web.

De plus JavaScript est un langage interprété, ce qui signifie qu'il est exécuté directement par le navigateur sans nécessiter de compilation. Cela permet d'obtenir des performances élevées, ce qui est crucial pour une application de chat en temps réel comme TissApp.

En outre, JavaScript est présent dans toutes les applications web et mobiles, ce qui en fait un choix logique pour un projet de chat mobile. Il est également important de noter qu'il est difficile de trouver une page web qui n'utilise pas JavaScript pour dynamiser son contenu de nos jours. En utilisant JavaScript pour mon projet, je peux être sûr que je travaille avec une technologie largement adoptée et qui est constamment mise à jour et améliorée.



Node.js est une plateforme open source qui permet d'exécuter du code JavaScript côté serveur. C'est un choix très populaire pour le développement d'applications web et d'API en raison de ses performances élevées et de sa capacité à gérer simultanément de nombreuses connexions. Voici pourquoi j'ai choisi nodejs pour le développement de TissApp :

JavaScript sur le serveur : Il permet d'exécuter du code JavaScript côté serveur, il facilite le développement full stack en JavaScript. Cela signifie qu'un seul langage peut être utilisé à la fois pour le développement front-end et back-end, simplifiant ainsi le processus de développement et de maintenance.

Performance : Il utilise le moteur JavaScript V8 de Google Chrome, qui compile le JavaScript en code machine, rendant l'exécution du code plus rapide et plus efficace.

Scalabilité : Node.js est léger et permet de gérer un grand nombre de connexions simultanées, ce qui le rend idéal pour les applications à grande échelle et à haut débit.

Écosystème : Il dispose d'un vaste écosystème de modules et de bibliothèques disponibles via le gestionnaire de paquets npm (Node Package Manager), qui peuvent aider à ajouter rapidement des fonctionnalités à une application.

En utilisant Node.js pour le back-end de TissApp, l'application bénéficie de performances élevées, d'une grande scalabilité et de la simplicité du développement full stack en JavaScript.

→ Choix des frameworks



Pour créer mon API, j'ai choisi d'utiliser le framework **Express.js** qui est un framework de Node.js. Ce choix s'explique par plusieurs raisons. Tout d'abord, Express.js est largement utilisé et possède une grande communauté active, ce qui signifie que je peux trouver facilement de la documentation, des tutoriels et des plugins.

De plus, Express.js est très flexible et permet de créer des applications web robustes et évolutives en utilisant des middlewares pour ajouter des fonctionnalités supplémentaires telles que la gestion des sessions, la sécurité, les routes, etc. Enfin, il est possible de l'utiliser avec différents moteurs de templating et de base de données, ce qui facilite l'intégration avec d'autres technologies.



J'ai choisi d'utiliser le framework React Native pour la création de mon application mobile, car il offre plusieurs avantages. Tout d'abord, étant écrit en javascript, il est facilement accessible pour moi qui ai une bonne expérience dans ce langage de programmation. Ensuite, il permet de

développer une seule fois le code source pour les deux plateformes mobiles majeures, iOS et Android, ce qui permet de gagner du temps et de l'argent dans le processus de développement.



Socket.IO est une bibliothèque JavaScript qui permet des communications bidirectionnelles en temps réel entre les navigateurs web (ou clients) et les serveurs. Elle utilise la technologie WebSockets, mais fournit également une compatibilité avec une grande variété de navigateurs qui ne prennent pas en charge WebSockets. Voici pourquoi Socket.IO a été utilisé dans le développement de TissApp :

L'une des principales utilisations de Socket.IO est pour les applications de chat en temps réel, comme TissApp. Socket.IO permet d'envoyer et de recevoir des messages instantanément.

Avec Socket.IO, la communication peut se faire dans les deux sens entre le client et le serveur. Cela signifie que le serveur peut pousser des messages vers le client à tout moment, et non seulement en réponse à une demande du client.

En utilisant Socket.IO dans le développement de TissApp, l'application peut offrir une expérience de chat en temps réel fluide et réactive, avec une large compatibilité entre différents navigateurs et environnements.

→ Logiciels et autres outils

Afin de mener à bien ce projet, j'ai utilisé divers outils et logiciels, notamment :

- **IDE (Integrated Development Environment)** : J'ai utilisé **Visual studio code** pour écrire et organiser le code de manière efficace. Il offre de nombreuses fonctionnalités utiles comme la complétion automatique du code, la mise en évidence de la syntaxe, et l'intégration avec des systèmes de contrôle de version comme Git.
- **Git** : Git est un système de contrôle de version qui permet de suivre et de gérer les changements dans le code source. Il est essentiel pour le développement en équipe, mais il est également très utile pour un développeur individuel.
- **GitHub** : C'est une plateforme de gestion de code source basées sur Git qui fournit un espace pour stocker, partager et collaborer sur le code source de l'application.
- **Postman** : Postman est un outil de test d'API qui peut être utilisé pour envoyer des requêtes HTTP à l'API de l'application et voir les réponses. Il est très utile pour le débogage et le test de l'API.
- **Jest** : j'ai utilisé jest pour tester certaine fonctionnalité de mon code
- **Node Package Manager (npm)** : npm est le gestionnaire de paquets de Node.js. Je l'ai utilisé pour installer et gérer les dépendances du projet.
- **React Native Debugger** : C'est un outil essentiel pour le débogage des applications React Native. Il permet d'inspecter le code, de suivre l'état et les props de vos composants, et de voir les logs de console.
- **Expo** : Expo est un ensemble d'outils et de services pour le développement d'applications React Native qui facilite le processus de développement, de construction, de déploiement et de mise à jour de l'application. **PHPMyAdmin**: C'est une base de données relationnelle. Je l'ai utilisé pour stocker les données de l'application.
- **ESLint** : Cet outil de linting pour JavaScript est utilisé pour détecter les erreurs et les problèmes de style de code, améliorant ainsi la qualité du code.
- **Figma** pour la conception de mes maquettes
- **Canva** pour la création du logo

- **Lucidchart et Drawio (extension sur VS code)** est un outil en ligne de création de diagrammes et de visualisations qui est souvent utilisé pour la modélisation de bases de données. Ils proposent une interface glisser-déposer facile à utiliser qui permet de créer des diagrammes de bases de données relationnelles (ERD), des organigrammes, des diagrammes UML, et bien d'autres types de visualisations.

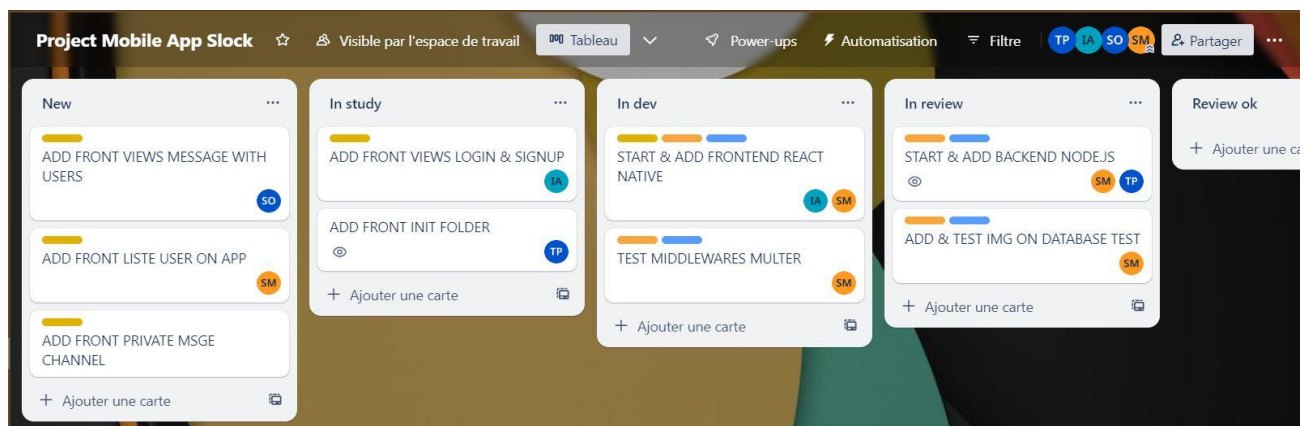
Organisation du projet

L'organisation du projet de développement de TissApp est un élément clé pour assurer une progression fluide et une livraison réussie.

Durant ma période de formation, j'ai dû mener ce projet en parallèle d'autres travaux à réaliser pour mon entreprise. Pour gérer efficacement mon temps, j'ai dû mettre en place une organisation rigoureuse. J'ai commencé par dresser une liste des tâches principales et indispensables. J'ai utilisé des outils comme **Trello** en centre de formation et **Jira** en entreprise afin de lister les tâches à effectuer.

Avant de commencer à coder, une planification soignée est nécessaire. Cela implique de définir les fonctionnalités requises, d'estimer le temps nécessaire pour chaque tâche, de définir les jalons du projet et d'établir un calendrier de développement.

Exemple:



Architecture logicielle

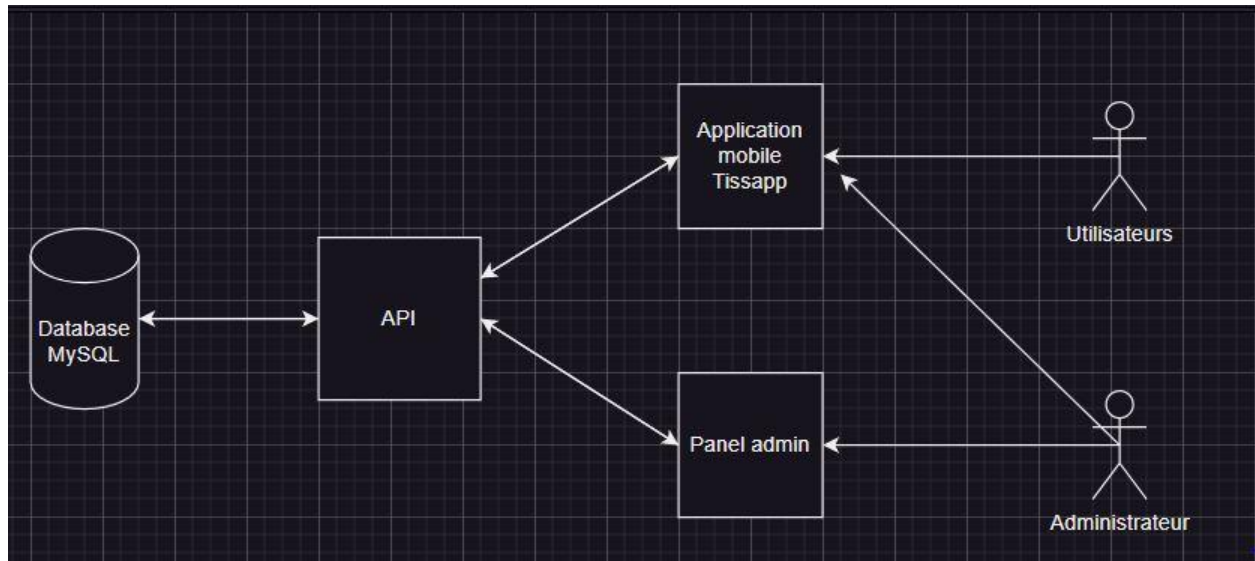
TissApp est conçu pour deux types d'utilisateurs principaux: les utilisateurs simples et les administrateurs. Ces deux types d'utilisateurs ont des capacités et des niveaux d'accès différents selon leur rôle.

Utilisateurs simples : Les utilisateurs simples sont les employés ordinaires qui utilisent l'application pour communiquer entre eux et partager leurs dépenses. Ils ont uniquement accès à l'application mobile TissApp. Là, ils peuvent participer aux discussions du chat général, partager des photos de leurs tickets de frais, voir le profil des autres utilisateurs et bien plus encore.

Administrateurs : Les administrateurs, d'autre part, ont un accès plus étendu. Non seulement ils ont accès à toutes les fonctionnalités de l'application mobile, mais ils ont également accès au panel d'administration. Ce panel leur permet de superviser les activités sur l'application, de gérer les utilisateurs, d'analyser les données en temps réel et de configurer divers aspects de l'application. Ils ont le pouvoir de modérer les discussions, de contrôler les comptes des utilisateurs et d'ajuster les paramètres de l'application si nécessaire.

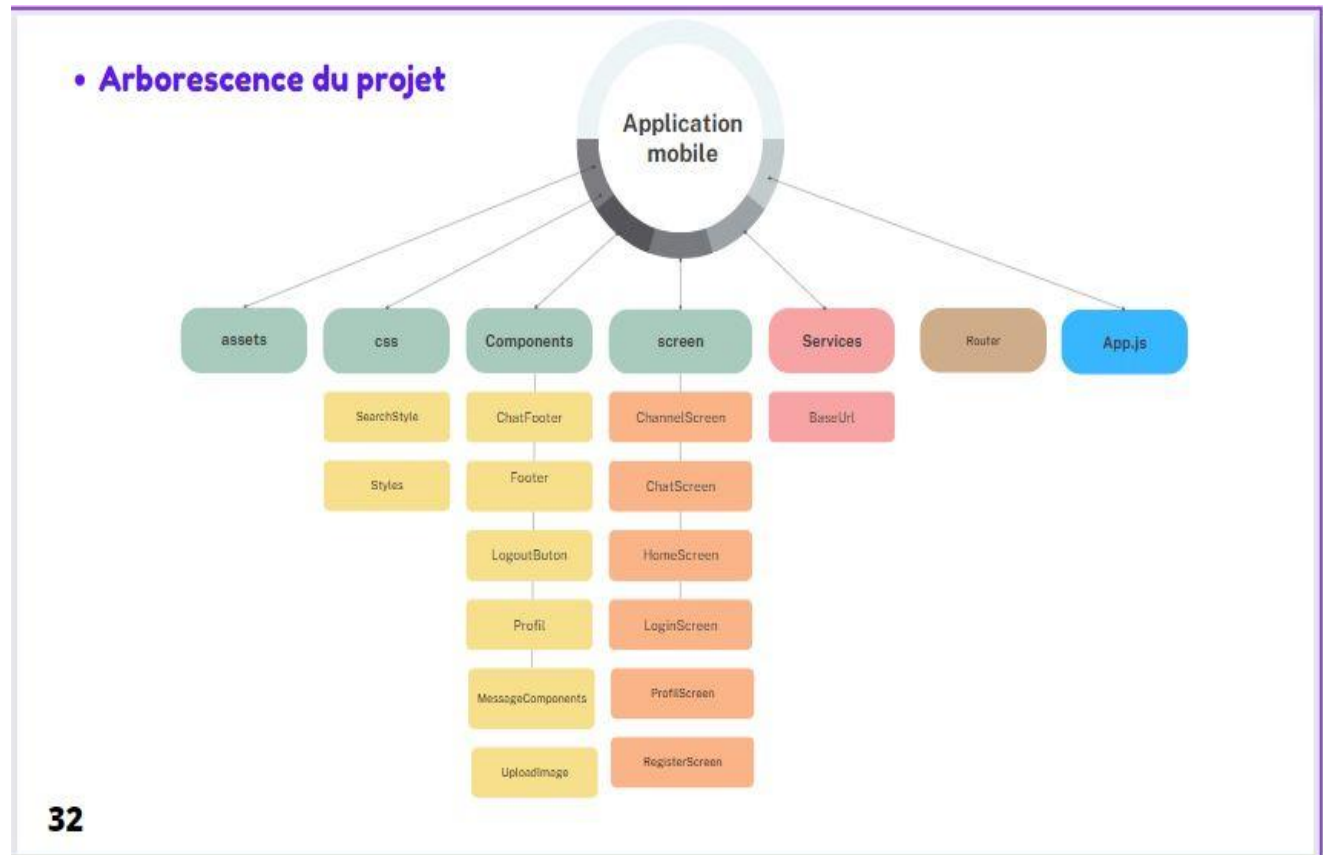
Les deux types d'utilisateurs interagissent avec une API commune, qui communique avec la base de données MySQL pour récupérer et stocker les données nécessaires. L'API est construite de manière à garantir que chaque utilisateur ne peut accéder qu'aux informations et aux fonctionnalités qui lui sont autorisées selon son rôle.

Cette conception permet de maintenir un équilibre entre la facilité d'utilisation pour tous les utilisateurs et le contrôle nécessaire pour les administrateurs, tout en garantissant la sécurité et l'intégrité des données de l'application.

Schéma de l'architecture logiciel :

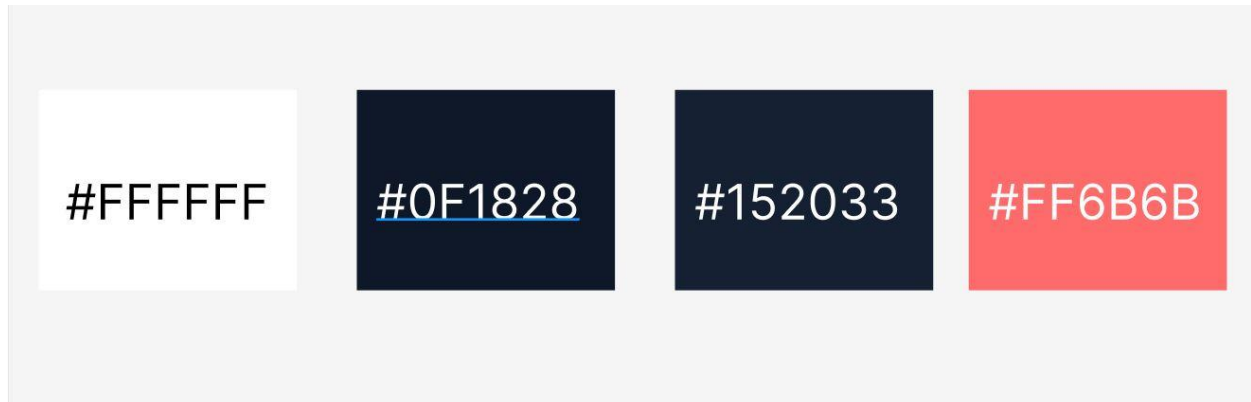
Conception du front-end de l'application

Arborescence du projet



Charte graphique

Une fois que j'ai établi l'organisation du projet avec son arborescence, j'ai entamé la création du logo en utilisant l'outil Canva. Ensuite, j'ai extrait les couleurs du logo pour élaborer une charte graphique.



Couleur police

Background color

Background color 2

Couleur bouton



—> Logo de l'application

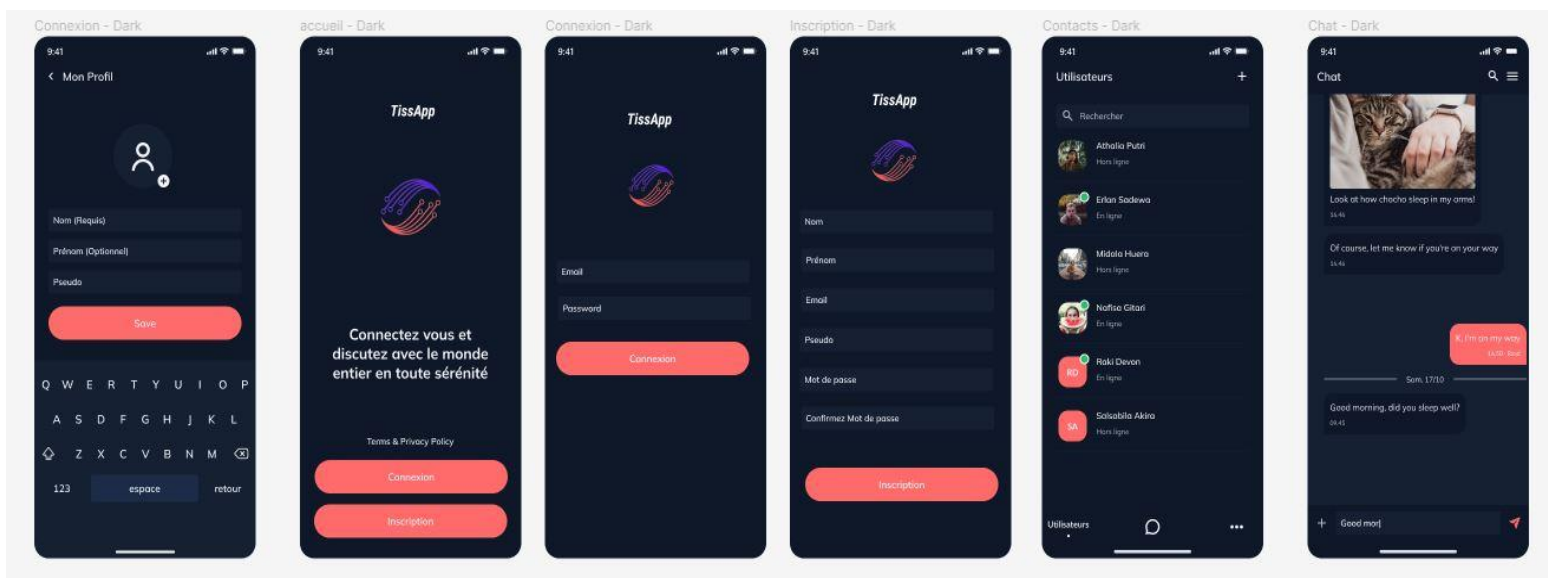
Wireframe

Le wireframe a été créer avec figma



Maquette

La maquette aussi a été réalisé avec figma



Conception backend de l'application

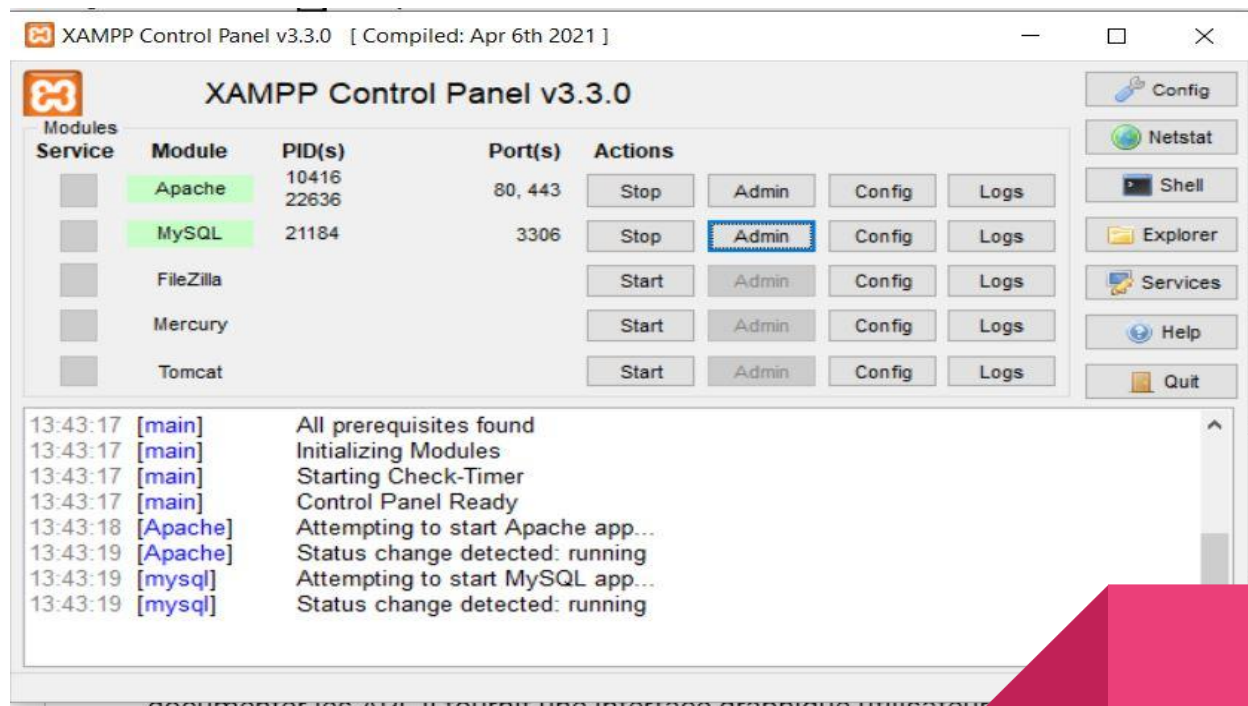
Base de données

→ Mise en place de la base de donnée

Dans le cadre de ce projet, la mise en place de la base de données a été réalisée en utilisant **Sequelize**, un ORM pour **Node.js**, **XAMPP** comme environnement de développement local, et **PhpMyAdmin** comme interface pour la gestion de la base de données.

J'ai commencé par installer **XAMPP**, qui m'a fourni un environnement de développement local facile à utiliser avec une distribution de MySQL, une base de données très répandue compatible avec Sequelize. XAMPP m'a permis de gérer facilement mon serveur local et ma base de données MySQL, ce qui a simplifié le processus de développement.

Ensuite, j'ai défini les modèles de données nécessaires à l'aide de Sequelize, y compris User, Posts et j'ai défini les relations entre eux. Un User peut avoir plusieurs Post, et chaque Post est lié à un User. Les méthodes **hasMany** et **belongsTo** de Sequelize ont été utilisées pour créer ces relations, garantissant ainsi une correspondance claire entre les données de l'application et la structure de la base de données.



→ Conception de la base de données

La conception de la base de données est un élément central du projet. Elle a nécessité une compréhension approfondie des besoins de l'application de chat, ainsi qu'une capacité à prévoir ses évolutions futures. La base de données a été conçue pour être à la fois flexible et performante.

Dans le cadre de ce projet, j'ai conçu une base de données relationnelle pour stocker et gérer les données de l'application de chat. La base de données comprend deux tables principales : `User` et `Post`. Ces tables sont liées entre elles de manière à optimiser les performances et à faciliter les requêtes.

- **Table User** : Cette table stocke les informations de chaque utilisateur, y compris un identifiant unique (**id**), un nom d'utilisateur (**lastName**), un prénom de l'utilisateur (**firstName**), un email (**email**) et un mot de passe (**password**). Le mot de passe est crypté à l'aide d'une fonction de hachage pour garantir la sécurité des données de l'utilisateur.
- **Table Post** : Cette table stocke les messages envoyés par les utilisateurs. Chaque enregistrement comprend un identifiant unique (**id**), le contenu du message (**content**), et une image (**imageUrl**) l'identifiant de l'utilisateur qui a envoyé le message (**userId**).

J'ai ensuite défini les relations entre ces tables en utilisant les méthodes fournies par Sequelize. Par exemple, un User peut envoyer plusieurs Message, donc j'ai utilisé la méthode **User.hasMany(Post)** pour établir cette relation. Inversement, chaque Message appartient à un User. Ce que j'ai défini en utilisant **Post.belongsTo(User)**.

Exemple de code du modèle User:

```

module.exports = (sequelize, DataTypes) => {
  Tchessi, 5 months ago | 1 author (Tchessi)
  class User extends Model { ...
  }
  User.init(
    Tchessi, 5 months ago * Back-end api init
    {
      firstName: {
        type: DataTypes.STRING,
        allowNull: false,
      },
      lastName: {
        type: DataTypes.STRING,
        allowNull: false,
      },
      email: {
        type: DataTypes.STRING,
        allowNull: false,
        validate: {
          isEmail: true,
          //utilisation d'une méthode pour pouvoir afficher un message d'erreur customisé
          async ensureEmailIsUnique(email) {
            if (await User.findOne({ where: { email } }))
              throw new Error(
                'Un compte existe déjà avec cette adresse mail !'
              );
          },
        },
      },
      password: {
        type: DataTypes.STRING,
        allowNull: false,
        validate: {
          ensurePasswordIsStrongEnough,
        },
      },
      imageUrl: DataTypes.STRING,
      deleted: {
        type: DataTypes.BOOLEAN,
        defaultValue: false,
      },
      admin: {
        type: DataTypes.BOOLEAN,
        defaultValue: false,
      },
      status: {
        type: DataTypes.STRING,
        defaultValue: 'offline',
      },
    },
    {
      sequelize,
      modelName: 'User',
    }
  );
};

```

Exemple du code du modèle Post:

```

8  module.exports = (sequelize, DataTypes) => {
    Tchessi, 5 months ago | 1 author (Tchessi)
9      class Post extends Model {
10         /**
11          * Helper method for defining associations.
12          * This method is not a part of Sequelize lifecycle.
13          * The `models/index` file will call this method automatically.
14          */
15         static associate(models) {
16             Post.belongsTo(models.User, { foreignKey: 'userId' });
17             Post.hasMany(models.Comments);
18             Post.hasMany(models.Likes);
19         }
20
21         readableCreatedAt() {
22             return moment(this.createdAt).locale('fr').format('LL');
23         }
24     }
25     Post.init(
26         {
27             userId: DataTypes.INTEGER,
28             content: DataTypes.TEXT,
29             imageUrl: DataTypes.STRING,
30             likesCount: DataTypes.INTEGER,
31         },
32         {
33             sequelize,
34             validate: {
35                 eitherContentOrImageUrl() {
36                     if (!this.content && !this.imageUrl) {
37                         throw new Error('Vous ne pouvez pas créer de publication vide !');
38                     }
39                 },
40             },
41             modelName: 'Post',
42         }
43     );
44
45     Post.afterDestroy(async (post) => {
46         if (post.imageUrl) {
47             await deleteFile(post.imageUrl);
48         }
49     });
50
51     Post.afterUpdate(async (post) => {
52         if (post.dataValues.imageUrl !== post._previousDataValues.imageUrl) {
53             await deleteFile(post._previousDataValues.imageUrl);
54         }
55     });
56
57     return Post;
58 };
59

```


→ Modèle Conceptuel de Données (MCD)

Le **Modèle Conceptuel de Base de Données (MCBD)** est une représentation visuelle des entités d'une base de données, des attributs de ces entités et des relations entre elles. C'est un outil utile pour comprendre comment les différentes parties d'une base de données interagissent.

Dans le cadre de la conception de la base de données de l'application de chat, j'ai établi un Modèle Conceptuel de Base de Données (**MCD**) pour illustrer les relations entre les différentes entités.

Les principales entités de notre base de données sont : `User` et `Post`.

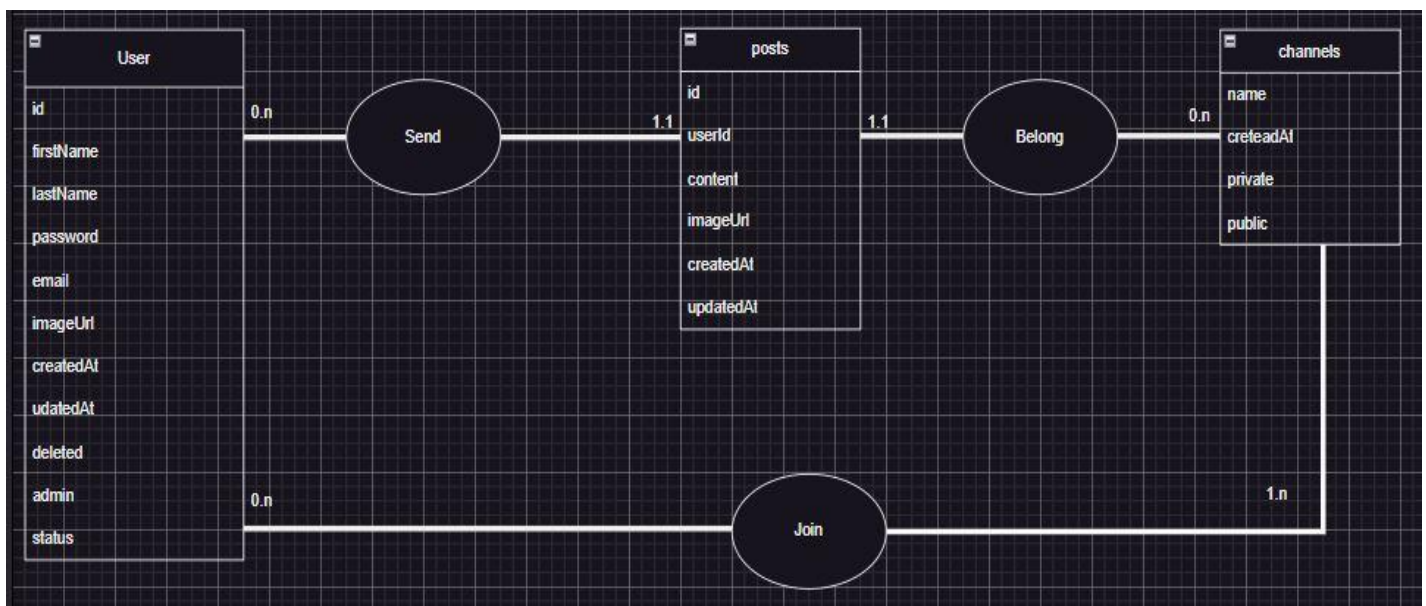
Entité User : Cette entité représente un utilisateur de l'application de chat. Elle a plusieurs attributs, dont **id** (identifiant unique), **lastname** (nom de l'utilisateur), **firstname** (prénom de l'utilisateur), **email** (adresse email de l'utilisateur), **imageUrl** (la photo de profil de l'utilisateur), **password** (mot de passe de l'utilisateur, stocké de manière sécurisée), le **statut** (montre si l'utilisateur est hors ligne ou en ligne) et **admin** (le rôle de l'utilisateur). En suite **created_At** (la date de création de l'utilisateur) et **updated_At** (la date de mise à jour des données de l'utilisateur).

Entité Post : Cette entité représente un message envoyé par un utilisateur. Elle comprend les attributs **id** (identifiant unique), **content** (contenu du message), **userId** (l'identifiant de l'utilisateur qui a envoyé le message), et **imageUrl** (l'url de l'image envoyée). En suite **created_At** (la date de création du message) et **updated_At** (la date de modification du message).

Les relations entre ces entités sont les suivantes :

- **Un User** peut envoyer plusieurs Message, ce qui est une relation "un à plusieurs" entre User et Message. Cette relation est représentée par une flèche allant de User à Message.
- **Un Post** appartient à un User ce qui est une relation "un à un" entre Message et User. Ces relations sont représentées par des flèches allant de Message à User.

Diagramme MCD:



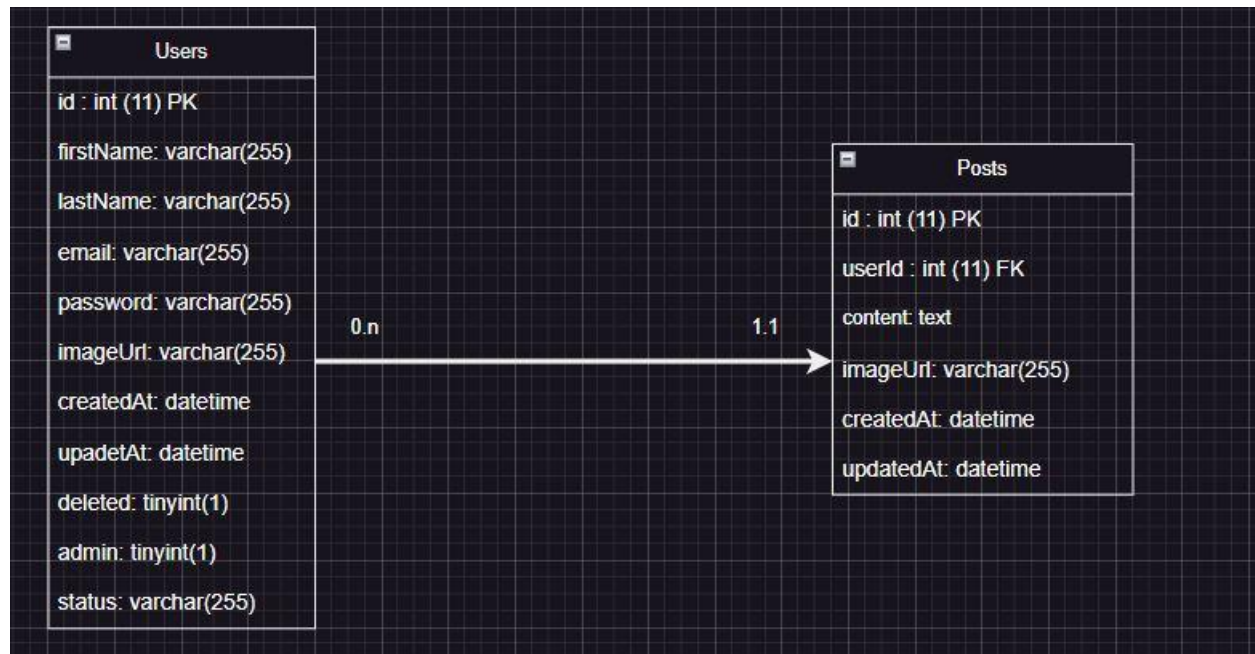
→ Modèle Logique de Données (MLD)

L'une des principales étapes de la création de l'application de chat était la conception et la mise en œuvre du **Modèle Logique de Données**. Le **MLD** est crucial pour définir comment les informations sont stockées, gérées et récupérées dans la base de données. Voici le MLD que j'ai développé pour ce projet :

Table User : Cette table stocke les informations sur chaque utilisateur. Elle comprend les attributs suivants : **User(id, firstName, lastName, imageUrl, email, password, status, admin, created_At, updated_At)**

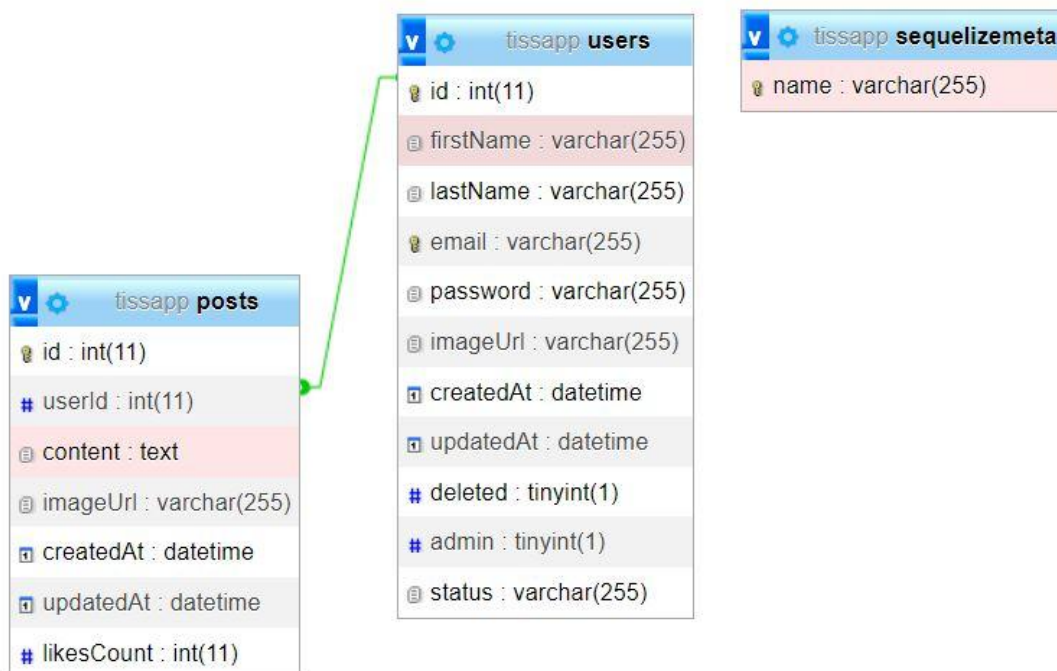
- **id** (Clé Primaire) : C'est l'identifiant unique de chaque utilisateur.
 - **lastname**: C'est le nom d'utilisateur,
 - **firstname**: C'est le prénom d'utilisateur,
 - **email** : C'est l'adresse e-mail de l'utilisateur qui est unique pour chaque utilisateur,
 - **password** : C'est le mot de passe de l'utilisateur, qui est stocké de manière sécurisée.
 - **imageUrl**: l'url de la photo de profil de l'utilisateur
 - **status**: stocke l'état de connexion
 - **admin**: le rôle de l'utilisateur qui lui est associé.
-
- **Table Post** : Cette table stocke les messages envoyés par les utilisateurs. Elle comprend les attributs suivants : **Post(id, content, userId, imageUrl)**
 - **id** (Clé Primaire) : C'est l'identifiant unique de chaque message.
 - **content** : C'est le contenu du message.
 - **imageUrl**: url de l'image posté par l'utilisateur
 - **userId** (Clé Étrangère) : C'est l'identifiant de l'utilisateur qui a envoyé le message.

Diagramme MLD:



→ Modèle Physique de Données (MPD)

Le Modèle Physique de Données (**MPD**) offre une représentation graphique des structures de la base de données et de leurs relations. Pour l'application de chat que j'ai développée, le MPD est composé de deux tables principales : User, Post.



Développement du backend de l'application

Organisation

Dans le but de maximiser l'efficacité et la réutilisation du code, j'ai orienté la conception de mon backend pour qu'il serve à la fois mon application web et mobile. Cela m'a amené à créer une API, éliminant ainsi le besoin de dupliquer la logique métier.

Mon objectif principal était de rendre mon backend plus performant. Pour ce faire, j'ai porté une attention particulière à la structuration et à l'optimisation de mon code, ce qui a nécessité une recherche approfondie.

J'ai adopté une approche modulaire pour la programmation, divisant mes programmes en différents modules. Cette démarche a non seulement rendu mon code plus lisible, mais elle a également simplifié sa maintenance pour les futures versions.

Dans le but d'éviter les redondances, j'ai mis en place des fonctions et services réutilisables. Ces éléments ont permis d'optimiser mon code tout en renforçant sa structure et son efficacité.

La logique de mon code a été soigneusement organisée en différents services et fichiers, créant une structure claire qui facilite sa compréhension et sa gestion.

Arborescence

J'ai adopté une architecture pour mon backend qui met en avant la séparation des préoccupations, permettant d'éloigner la logique métier des routes de l'API. Mon application est organisée en plusieurs couches distinctes :

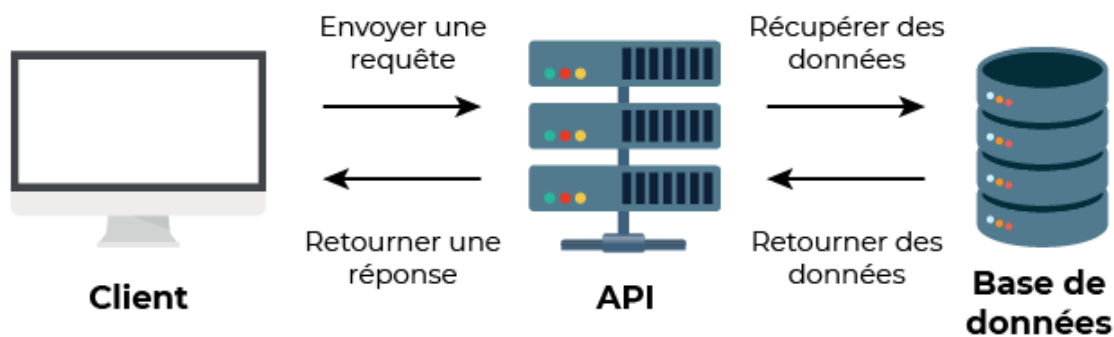
- Routeur,
- Couche applicative, subdivisée en contrôleurs et services,
- La couche de données qui inclut les modèles.

Pour donner vie à cette structure, j'ai organisé mon backend en une série de dossiers, chacun ayant une fonction définie :

- Le dossier "**Config**" : Il contient tous les fichiers de configuration nécessaires pour le fonctionnement de l'application.
- Le dossier "**Controllers**" : Il regroupe les contrôleurs, où chaque route a son propre contrôleur.
- Le dossier "**Middleware**" : Il comprend des sous-dossiers pour chaque route, chacun contenant les fichiers middleware associés.
- Le dossier "**Migration**" : Il est utilisé pour gérer toutes les migrations de bases de données.
- Le dossier "**Models**" : Il renferme les modèles de toutes mes tables, avec un modèle spécifique pour chaque table et fichier.
- Le dossier "**Public**" : Il contient les fichiers statiques accessibles par le client, tels que les images, les feuilles de style CSS et les scripts JavaScript.
- Le dossier "**Routes**" : Il regroupe tous les fichiers de routes, un pour chaque opération CRUD d'une table de base de données.
- Le dossier "**Seeds**" : Il est utilisé pour stocker les données initiales de la base de données.
- Le dossier "**Service**" : Il contient des fichiers qui définissent les logiques métier.
- Le dossier "**Tests**" : Il est destiné aux tests unitaires.

Fonctionnement de l'API

Lorsqu'une requête client arrive à mon API, le routeur l'intercepte d'abord, analysant l'URL. En fonction de la route et de la méthode HTTP utilisée, le routeur dirige la requête vers le contrôleur approprié. Ce contrôleur, à son tour, sollicite un service spécifique qui communique avec le modèle correspondant pour récupérer ou modifier les données nécessaires. Une fois les données récupérées et traitées par le service, une réponse est formulée et envoyée au client en format JSON, accompagnée d'un code de statut approprié.



Voici les codes de statut utilisés dans ce projet et leur signification :

- **200** : OK - La requête a abouti avec succès.
- **201** : CREATED - La requête a été exécutée avec succès et a engendré la création d'une nouvelle ressource.
- **204** : NO CONTENT - La requête a été traitée avec succès mais il n'y a pas de contenu à renvoyer.
- **400** : BAD REQUEST - Le serveur ne peut traiter la requête en raison d'une syntaxe incorrecte.
- **401** : - La requête nécessite une authentification.

- **403** : FORBIDDEN - La requête est comprise par le serveur mais refusée.
- **404** : NOT FOUND - La ressource demandée n'a pas été trouvée sur le serveur.
- **405** : METHOD NOT ALLOWED - La méthode de requête est reconnue par le serveur mais n'est pas supportée par la ressource cible.
- **500** : INTERNAL SERVER ERROR - Le serveur a rencontré une erreur interne.

Concernant les méthodes HTTP utilisées dans ce projet, on retrouve :

- **GET** : Utilisée pour récupérer des données.
- **POST** : Utilisée pour enregistrer de nouvelles données.
- **PUT** : Utilisée pour mettre à jour l'intégralité d'une ressource.
- **PATCH** : Utilisée pour une mise à jour partielle d'une ressource.
- **DELETE** : Utilisée pour supprimer une ressource.

Middleware

Les **middlewares** jouent un rôle fondamental dans le flux de traitement des requêtes au sein de mon application. Ils interviennent entre la réception d'une requête par le serveur et son traitement par un contrôleur spécifique. Les middlewares peuvent être vus comme une série de fonctions à travers lesquelles la requête "voyage" avant d'arriver à sa destination finale. Les middlewares peuvent effectuer diverses tâches, notamment :

Vérification de l'authentification : Avant qu'une requête ne soit traitée, un middleware peut vérifier si l'utilisateur qui a envoyé la requête est authentifié. Il peut par exemple vérifier si un token d'authentification est présent dans les en-têtes de la requête. Si l'utilisateur n'est pas authentifié, le middleware peut bloquer la requête et renvoyer une erreur au client.

Filtrage des données entrantes : Un middleware peut également vérifier les données envoyées par le client pour s'assurer qu'elles sont valides et sûres. Par exemple, il peut vérifier si les données respectent un certain format, s'assurer qu'elles ne contiennent pas de code malveillant et que toutes les données requises sont présentes.

Gestion des erreurs : Certains middlewares sont utilisés pour gérer les erreurs qui peuvent se produire lors du traitement d'une requête. Si une erreur se produit, le middleware peut la capturer, la consigner dans un fichier de journal et renvoyer une réponse appropriée au client.

Logging : Les middlewares peuvent également être utilisés pour consigner des informations sur les requêtes et les réponses. Cela peut être utile pour le débogage et la surveillance de l'application.

En somme, les middlewares fournissent un mécanisme puissant et flexible pour gérer différents aspects du traitement des requêtes, de la sécurité à la validation des données, en passant par la gestion des erreurs et le logging.

Routeage

Le **routeage** est une partie essentielle de toute application web et **Express.js** fournit un moyen simple et efficace pour gérer les routes de votre application. Le système de routage d'Express.js permet de définir différentes routes pour différentes requêtes HTTP et de lier ces routes à des fonctions spécifiques qui seront exécutées lorsqu'un client envoie une requête à ces routes. Pour mettre en place le routage dans mon application, j'ai utilisé le routeur d'Express.js, qui est un middleware intégré. Le routeur permet de regrouper les routes associées en un seul endroit et de les manipuler comme un seul objet.

Chaque route dans une application Express.js est définie par une combinaison d'une méthode HTTP (GET, POST, PUT, DELETE, etc.), un chemin (l'URL à laquelle la route correspond) et une ou plusieurs fonctions de gestion (les fonctions qui sont exécutées lorsque la route est atteinte).

Voici un exemple de comment une route peut être définie avec Express.js:

```
back-end > routes > user.js > ...
You, last month | 2 authors (Tchessi and others)
1  const express = require('express');
2  const router = express.Router();
3
4  const userCtrl = require('../controllers/user');
5
6  const auth = require('../middleware/auth');
7  const multer = require('../middleware/multer-config');
8
9  router.post('/signup', userCtrl.signup);
10
11 router.post('/login', userCtrl.login);
12
13 router.put('/edit', auth, multer, userCtrl.editUser);
14
15 // Admin routes
16 router.put('/edit-admin/:id', auth, multer, userCtrl.editUserAdmin);
17 router.delete('/users-delete/:id', auth, userCtrl.deleteUserAccount);
18 module.exports = router;
19 |
```

Dans votre code :

```
const express = require('express');  
const router = express.Router();
```

```
const express = require('express');
```

Cette ligne de code importe le module Express.js dans votre fichier. Express est un framework pour Node.js qui simplifie le développement d'applications web. Il fournit un ensemble robuste de fonctionnalités pour les applications web et mobiles.

```
const router = express.Router();
```

Cette ligne de code crée une nouvelle instance du routeur Express. Un routeur est un objet qui permet de définir des routes. Chaque route peut avoir une ou plusieurs fonctions de gestion, qui sont exécutées lorsque la route est atteinte.

Les routes suivantes sont définies dans votre code :

```
router.post('/signup', userCtrl.signup);
```

Cette ligne crée une route pour les requêtes POST à l'URL '/signup'. Lorsqu'un client envoie une requête POST à cette URL, la fonction signup du contrôleur utilisateur (userCtrl) est exécutée. Cette fonction prend en charge l'inscription d'un nouvel utilisateur.

```
router.post('/login', userCtrl.login);
```

Cette ligne crée une route pour les requêtes POST à l'URL '/login'. Lorsqu'un client envoie une requête POST à cette URL, la fonction login du contrôleur utilisateur est exécutée. Cette fonction prend en charge la connexion d'un utilisateur.

```
router.put('/edit', auth, multer, userCtrl.editUser);
```

Cette ligne crée une route pour les requêtes PUT à l'URL '/edit'. Avant d'appeler la fonction `editUser` du contrôleur utilisateur, deux middlewares sont exécutés : `auth`, qui vérifie si l'utilisateur est authentifié, et `multer`, qui gère les fichiers entrants.

Les deux dernières lignes de code définissent des routes spécifiques à l'administrateur. L'administrateur peut éditer les détails d'un utilisateur et supprimer un compte utilisateur.

Enfin, le routeur est exporté pour pouvoir être utilisé dans d'autres parties de votre application avec la ligne `module.exports = router;`.

Controller

Dans mes contrôleurs, je me focalise exclusivement sur des tâches de validation, comme par exemple, la vérification des droits d'accès. En respectant cette pratique, j'assure qu'aucune logique métier n'est présente dans mes contrôleurs. Voici comment cela fonctionne :

- Lorsqu'une requête utilisateur arrive, c'est moi, à travers les contrôleurs, qui la reçoit en premier lieu.
- Mon rôle ici est de vérifier si l'utilisateur a les droits nécessaires pour effectuer l'action demandée. Cette étape est cruciale pour moi car elle garantit la sécurité et l'intégrité de mon application.
- Une fois que j'ai confirmé que l'utilisateur est autorisé, je transmet la requête validée à la partie de l'application qui peut traiter la demande.

En m'assurant de ne pas inclure de logique métier dans mes contrôleurs, je maintiens ceux-ci simples et efficaces, tout en respectant leur rôle principal : valider les requêtes et gérer le flux de données entre l'utilisateur et l'application.

Service

Dans mon dossier de service, j'ai deux fichiers qui jouent des rôles différents dans le fonctionnement de mon application.

- Le premier fichier gère l'authentification des utilisateurs. J'y ai implémenté des fonctions pour garantir la robustesse des mots de passe. Par exemple, la fonction ``ensurePasswordIsStrongEnough`` vérifie que le mot de passe respecte certaines règles de complexité. De plus, j'ai ajouté une fonction ``addAuthenticationOn`` qui s'occupe du cryptage des mots de passe avant leur enregistrement dans la base de données, et de l'authentification des utilisateurs lorsqu'ils se connectent. Grâce à la librairie ``bcrypt``, je peux chiffrer les mots de passe et comparer un mot de passe saisi à son hash stocké.
- Dans le deuxième fichier, j'ai mis en place une fonction pour gérer la suppression des fichiers dans le système de fichiers. Cette fonction, ``deleteFile``, est utilisée pour supprimer les fichiers (par exemple, les images) quand ils ne sont plus nécessaires. En utilisant le module ``fs`` de Node.js, je peux facilement supprimer un fichier en fournissant son chemin.

Ces deux fichiers illustrent comment je structure mon code en respectant le principe de séparation des préoccupations : chaque service a sa propre responsabilité et ne s'occupe que des tâches qui lui sont propres.

Model

Dans mon dossier `models`, j'ai trois fichiers importants : `index.js`, `User.js` et `Post.js`.

Dans le fichier `index.js`, j'ai configuré la connexion à la base de données en utilisant les informations fournies dans les variables d'environnement. J'ai créé une instance de Sequelize avec ces informations et l'ai assignée à la variable `sequelize`. J'ai utilisé `fs` pour lire les fichiers du dossier `models` et importer les modèles associés à l'aide de `sequelize.define()`. J'ai également associé les modèles entre eux si nécessaire, en utilisant les méthodes

```
back-end > models > index.js > filter() callback
You, 2 weeks ago | 2 authors (Tchessi and others)
1  'use strict';
2
3  require('dotenv').config();
4
5  const fs = require('fs');
6  const path = require('path');
7  const Sequelize = require('sequelize');
8  const basename = path.basename(__filename);
9  const env = process.env.NODE_ENV || 'development';
10 const config = {
11   username: process.env.DB_USERNAME,
12   password: process.env.DB_PASSWORD,
13   database: process.env.DB_NAME,
14   host: process.env.DB_HOST,
15   port: process.env.DB_PORT,
16   dialect: process.env.DB_DIALECT,
17 };
18 const db = {};
19
20 let sequelize;
21 sequelize = new Sequelize(
22   config.database,
23   config.username,
24   config.password,
25   config
26 );
27
28 fs.readdirSync(__dirname)
29   .filter((file) => {
30     return (
31       file.indexOf('.') !== 0 && file !== basename && file.slice(-3) === '.js'
32     );
33   })
34   .forEach((file) => {
35     const model = require(path.join(__dirname, file))(
36       sequelize,
37       Sequelize.DataTypes
38     );
39     db[model.name] = model;
40   });
```

``associate()`` définies dans chaque modèle. Finalement, j'ai exporté les instances de Sequelize et les modèles pour les rendre accessibles dans d'autres parties de l'application.

Dans le fichier ``User.js``, j'ai défini le modèle ``User`` en étendant la classe ``Model`` de Sequelize. J'ai spécifié les attributs du modèle tels que ``firstName``, ``lastName``, ``email``, ``password``, etc. J'ai ajouté des validations pour garantir que les données respectent certaines règles, comme l'unicité de l'adresse e-mail et la complexité du mot de passe. J'ai également ajouté des hooks pour exécuter des actions spécifiques avant ou après certaines opérations, comme la suppression d'une image lorsqu'un utilisateur est mis à jour. J'ai associé le modèle ``User`` à d'autres modèles, comme ``Post``, en utilisant la méthode ``associate()``.

Dans le fichier ``Post.js``, j'ai défini le modèle ``Post`` de manière similaire. J'ai spécifié les attributs tels que ``userId``, ``content``, ``imageUrl``, etc. J'ai ajouté des validations pour m'assurer que soit le contenu soit l'image sont présents. J'ai également utilisé des hooks pour effectuer des actions avant ou après certaines opérations, comme la suppression d'une image après la suppression d'une publication. J'ai associé le modèle ``Post`` à d'autres modèles, comme ``User``, en utilisant la méthode ``associate()``. Ces fichiers dans le dossier `models` définissent la structure de la base de données et les relations entre les différentes entités de votre application. Ils utilisent Sequelize pour faciliter les opérations de base de données et fournir des fonctionnalités supplémentaires telles que les validations et les associations.

Sécurité

→ Credential stuffing : vol du login et password

Le **credential stuffing** est une technique utilisée par les attaquants pour tenter d'accéder à des comptes en utilisant des paires de nom d'utilisateur et de mot de passe volées à partir de sources externes, telles que des fuites de données provenant d'autres sites. Les attaquants essaient d'utiliser ces paires de connexion pour accéder à différents services en ligne, en pariant sur le fait que de nombreux utilisateurs utilisent les mêmes informations d'identification pour plusieurs comptes.

Pour prévenir le credential stuffing et **protéger** les utilisateurs contre les vols de login et **de mot de passe**, j'ai mis en place les mesures suivantes :

- **Politiques de mot de passe robustes** : J'ai encouragé les utilisateurs à choisir des mots de passe forts en définissant des exigences minimales, telles que l'utilisation de caractères alphanumériques, de symboles et de longueur minimale.
- Vérification de la complexité du mot de passe : J'ai mis en place des vérifications côté serveur pour m'assurer que les mots de passe respectent les critères de complexité définis.
- **Utilisation du hachage des mots de passe** : Les mots de passe des utilisateurs ne sont pas stockés en clair dans la base de données. J'utilise l'algorithme de hachage **bcrypt** pour les sécuriser. Le hachage des mots de passe rend pratiquement impossible la récupération des mots de passe d'origine à partir des données stockées.
- **Contrôle des tentatives de connexion** : J'ai mis en place des mécanismes pour limiter le nombre de tentatives de connexion infructueuses. Cela peut inclure des mécanismes tels que les délais de blocage après un certain nombre d'échecs de connexion, les captchas ou la vérification à deux facteurs pour renforcer la sécurité.

En mettant en place ces mesures, j'ai renforcé la sécurité des informations d'identification des utilisateurs et réduit les risques de vol de login et de mot de passe par le biais du credential stuffing.

→ Chiffrement des données sensibles

Pour assurer la sécurité des données sensibles dans mon application, j'ai mis en place des mesures de chiffrement. Le chiffrement consiste à convertir les données en un format illisible, appelé texte chiffré, afin de les rendre inintelligibles pour toute personne non autorisée. Voici comment j'ai utilisé le chiffrement des données sensibles :

- **Chiffrement des mots de passe** : Les mots de passe des utilisateurs ne sont pas stockés en clair dans la base de données. J'ai utilisé l'algorithme de chiffrement bcrypt, qui est un algorithme de hachage fort et adapté pour le stockage sécurisé des mots de passe. Les mots de passe sont transformés en une chaîne de caractères chiffrée à sens unique. Ainsi, même si la base de données est compromise, les mots de passe d'origine restent inaccessibles.

→ JWT

Dans mon projet, j'utilise le **Jeton Web Token (JWT)** pour mettre en place une authentification sécurisée et stateless. Le **JWT** est un jeton qui permet d'échanger des informations sur l'utilisateur de manière sécurisée entre deux parties.

Le **JWT est composé de trois parties** : le **header**, le **payload** et la **signature**. Le header identifie l'algorithme utilisé pour générer la signature. Le payload contient les informations de l'utilisateur, telles que son identifiant, son adresse e-mail et son rôle. Les données du payload sont encodées en base64 pour des raisons de transmission sécurisée.

Il est important de noter que je n'insère aucune donnée sensible, telle que des mots de passe dans le JWT. Seules des informations non sensibles et non confidentielles sont incluses pour identifier l'utilisateur.

La signature est créée en combinant le **header** et le **payload** avec une clé secrète connue uniquement par le serveur. Cette signature est utilisée pour vérifier l'intégrité du jeton. Si la signature est invalide, le serveur rejettera le jeton.

Lorsqu'un utilisateur tente de se connecter à son espace, une demande est envoyée au serveur. Si les informations d'identification sont correctes, le serveur renvoie une réponse JSON contenant le jeton JWT. Ce jeton contient des informations sur la personne connectée. Le client enverra ensuite ce jeton avec chaque demande ultérieure, généralement dans l'en-tête Authorization. Ainsi, le serveur peut vérifier l'authenticité du jeton et accorder les autorisations appropriées à l'utilisateur sans avoir à stocker d'informations sur la session côté serveur. Cela rend le système stateless et facilite la mise en place d'une architecture scalable.

En utilisant le JWT, j'assure une authentification sécurisée et efficace dans mon projet, tout en minimisant la quantité de données stockées côté serveur et en garantissant l'intégrité des informations échangées entre le client et le serveur.

→ Gestion de droits

Dans mon projet, j'ai mis en place un système de gestion des droits pour contrôler l'accès aux différentes fonctionnalités de l'application en fonction des rôles des utilisateurs. Cela permet de garantir la sécurité et la confidentialité des données, ainsi que de prévenir les utilisations non autorisées.

Dans mon projet, j'ai défini deux rôles principaux : l'**administrateur** et l'**utilisateur**.

1. Rôle Administrateur :

- **Accès au panneau d'administration** : L'administrateur a le droit d'accéder à un panneau d'administration qui lui permet de gérer différents aspects de l'application.
- **Gestion des utilisateurs** : L'administrateur peut voir la liste des utilisateurs enregistrés dans l'application et a le droit de les supprimer si nécessaire.
- **Suppression des messages** : L'administrateur a le droit de supprimer des messages publiés par les utilisateurs, par exemple en cas de contenu inapproprié ou non conforme aux règles de l'application.

2. Rôle Utilisateur :

- **Fonctionnalités de base** : Les utilisateurs disposent des fonctionnalités de base de l'application, telles que l'envoi de messages, la visualisation du contenu partagé, etc.
- **Accès limité au panneau d'administration** : Les utilisateurs ne peuvent pas accéder au panneau d'administration et n'ont pas les droits pour gérer les utilisateurs ou supprimer des messages d'autres utilisateurs.

Ces deux rôles permettent de différencier les niveaux de privilèges et d'accès dans l'application. L'administrateur a des droits étendus pour gérer l'application dans son ensemble, tandis que les utilisateurs ont des droits limités mais peuvent profiter des fonctionnalités principales de l'application.

La gestion des droits est réalisée à travers les contrôleurs et les middlewares correspondants, qui vérifient le rôle de l'utilisateur lorsqu'il tente d'accéder à certaines fonctionnalités ou effectue des actions spécifiques. Si un utilisateur tente d'accéder à une fonctionnalité réservée à l'administrateur sans en avoir les droits, une réponse d'erreur appropriée sera renvoyée.

Cela permet de maintenir un niveau de sécurité et de confidentialité élevé dans l'application, en garantissant que seuls les administrateurs ont les droits nécessaires pour gérer les utilisateurs et les contenus de manière appropriée.

En mettant en place ce système de gestion des droits, j'assure que seuls les utilisateurs autorisés peuvent accéder aux fonctionnalités appropriées de l'application. Cela garantit la sécurité des données et prévient les abus ou les accès non autorisés.

Exemple de problématique rencontrée

Lors du développement de mon application de chat, j'ai été confronté à une problématique spécifique qui nécessitait une attention particulière. L'exemple suivant illustre cette situation :

Problématique : Implémentation **de Socket.IO** dans l'application de chat

Lors de l'implémentation de Socket.IO dans mon application de chat, j'ai rencontré certaines difficultés qui ont nécessité une attention particulière. Voici un exemple de problématique rencontrée :

1. Gestion des connexions et des déconnexions : L'une des difficultés majeures était la gestion des connexions et des déconnexions des utilisateurs. Lorsqu'un utilisateur se connecte à l'application, il doit être ajouté à la liste des utilisateurs connectés et être en mesure de recevoir les messages en temps réel. De même, lorsqu'un utilisateur se déconnecte, il doit être retiré de la liste des utilisateurs connectés et ne plus recevoir les messages.

Solution proposée :

- **Utilisation des événements de connexion et de déconnexion de Socket.IO** : J'ai utilisé les événements "**connection**" et "**disconnect**" fournis par Socket.IO pour détecter les connexions et les déconnexions des utilisateurs. Lorsqu'un utilisateur se connecte, j'ajoute son identifiant à la liste des utilisateurs connectés. Lorsqu'un utilisateur se déconnecte, j'enlève son identifiant de la liste.

- **Gestion des salles de chat** : Pour permettre à chaque utilisateur de rejoindre un salon de discussion spécifique, j'ai utilisé les fonctionnalités de Socket.IO pour créer et gérer des salles de chat. Lorsqu'un utilisateur rejoint un salon, je l'ajoute à la salle correspondante et il peut ensuite recevoir et envoyer des messages spécifiques à ce salon.

2. Échange de messages en temps réel : Une autre problématique était d'assurer l'échange de messages en temps réel entre les utilisateurs connectés. Lorsqu'un utilisateur envoie un message, tous les autres utilisateurs connectés doivent le recevoir instantanément.

Solution proposée :

- Utilisation des événements personnalisés : J'ai créé des événements personnalisés dans Socket.IO pour l'échange de messages. Lorsqu'un utilisateur envoie un message, j'émet un événement contenant les informations du message vers tous les utilisateurs connectés

Recherches anglophones

Lors du développement de l'application de chat, j'ai été confronté à plusieurs défis techniques pour lesquels j'ai dû rechercher des solutions appropriées. Pour y remédier, j'ai utilisé différentes ressources en ligne, notamment **Stack Overflow**, une plateforme communautaire de questions et réponses pour les développeurs.

J'ai également consulté la documentation officielle d'Express, un framework Node.js populaire utilisé pour la création d'API Web. Malgré le fait que cette documentation soit rédigée en anglais, j'ai pu exploiter les exemples de code et les explications fournis pour résoudre certains problèmes spécifiques liés à la mise en place de mon serveur de chat.

En utilisant "j'ai" dans votre rédaction, vous mettez l'accent sur votre propre implication dans la recherche de solutions et dans l'apprentissage autonome nécessaire pour surmonter les défis rencontrés lors du développement de l'application de chat.

Exemple d'envois de données avec images

Dans l'exemple suivant,, je vais vous montrer comment j'ai procédé pour développer la fonctionnalité de création d'une demande de remboursement de note de frais, en utilisant **Multer** pour la gestion des fichiers côté serveur, et **Expo Camera** et **Expo ImagePicker** côté client pour la capture et la sélection d'images.

Côté client (React Native avec Expo) :

- J'ai utilisé **Expo Camera** pour permettre à l'utilisateur de prendre une photo en utilisant la caméra de son appareil.
- J'ai utilisé **Expo ImagePicker** pour permettre à l'utilisateur de sélectionner une image existante dans la galerie de son appareil.
- J'ai intégré ces fonctionnalités dans l'interface utilisateur en ajoutant des boutons ou des gestes qui déclenchent l'ouverture de la caméra ou de la galerie et la capture ou la sélection de l'image.
- Une fois que l'utilisateur a capturé ou sélectionné une image, j'ai stocké l'URI de l'image dans l'état de l'application.

➤ `getPermissionsAsync`:

```
29 // Demande les permissions pour accéder à la caméra et à la galerie
30 const getPermissionsAsync = async () => {
31   const { status: cameraStatus } = await Camera.requestCameraPermissionsAsync();
32   const { status: mediaLibStatus } = await MediaLibrary.requestPermissionsAsync();
33
34   if (cameraStatus !== 'granted' || mediaLibStatus !== 'granted') {
35     alert('Vous devez autoriser l'accès à la caméra et à la galerie pour utiliser
36       cette fonctionnalité.');
```

J'ai créé une fonction asynchrone nommée **getPermissionsAsync** qui gère la demande des autorisations d'accès à la caméra et à la galerie.

Voici une brève explication de cette fonction :

- J'ai utilisé **Camera.requestCameraPermissionsAsync()** pour demander l'autorisation d'accéder à la caméra de l'appareil.
- Ensuite, j'ai utilisé **MediaLibrary.requestPermissionsAsync()** pour demander l'autorisation d'accéder à la galerie de médias de l'appareil.
- J'ai stocké les résultats de ces demandes d'autorisations dans les variables **cameraStatus** et **mediaLibStatus**.
- J'ai vérifié si les autorisations ont été accordées en comparant les valeurs de **cameraStatus** et **mediaLibStatus** à 'granted'.
- Si l'accès à la caméra ou à la galerie n'a pas été accordé, j'ai affiché une alerte pour informer l'utilisateur qu'il doit donner son autorisation pour accéder à la caméra et à la galerie afin de pouvoir utiliser cette fonctionnalité.

Cette fonction **getPermissionsAsync** est utile pour m'assurer que j'ai obtenu les autorisations nécessaires pour accéder à la caméra et à la galerie avant d'utiliser des fonctionnalités spécifiques qui dépendent de ces autorisations.

➤ **takePicture:**

```
52  const takePicture = async () => {  
53    await getPermissionsAsync();  
54    let image = await ImagePicker.launchCameraAsync({  
55      mediaTypes: ImagePicker.MediaTypeOptions.Images,  
56      allowsEditing: false,  
57      quality: 1,  
58    });  
59    if (!image.canceled) {  
60      setImage(image.assets[0].uri);  
61    }  
62  };  
63
```

La fonction **takePicture** est une fonction asynchrone qui permet de prendre une photo à l'aide de la caméra de l'appareil.

Voici une brève explication de cette fonction :

- J'appelle la fonction **getPermissionsAsync** pour m'assurer que j'ai obtenu les autorisations nécessaires d'accès à la caméra avant de prendre la photo.
- J'utilise **ImagePicker.launchCameraAsync()** pour ouvrir l'interface de la caméra de l'appareil et permettre à l'utilisateur de prendre une photo.
- J'ai configuré quelques options pour la prise de la photo :
 - **mediaTypes** : Je limite les médias capturables à des images uniquement.
 - **allowsEditing** : J'ai désactivé l'édition de l'image après la capture.
 - **quality** : J'ai défini la qualité de l'image à 1 (la meilleure qualité). La fonction stocke la photo capturée dans la variable image.
- J'utilise une condition pour vérifier si l'utilisateur a annulé la capture de la photo (!image.canceled). Si la capture de la photo n'a pas été annulée, j'utilise `setImage(image.assets[0].uri)` pour mettre à jour l'état de l'application avec l'URI de la photo capturée.

Cette fonction `takePicture` est utilisée pour permettre à l'utilisateur de prendre une photo à l'aide de la caméra de l'appareil et de stocker cette photo dans l'état de l'application pour une utilisation ultérieure.

➤ **AddPicture:**

```
39  const addPicture = async () => {  
40      let image = await ImagePicker.launchImageLibraryAsync({  
41          mediaTypes: ImagePicker.MediaTypeOptions.Images,  
42          allowsEditing: true,  
43          aspect: [4, 3],  
44          quality: 1,  
45      });  
46      if (!image.canceled) {  
47          setImage(image.assets[0].uri);  
48      }  
49  };  
50  
51
```

La fonction **addPicture** est une fonction asynchrone qui permet d'ajouter une image à partir de la bibliothèque de l'appareil.

Voici une brève explication de cette fonction :



- J'utilise **ImagePicker.launchImageLibraryAsync()** pour ouvrir la bibliothèque de médias de l'appareil et permettre à l'utilisateur de sélectionner une image.
- J'ai configuré quelques options pour la sélection de l'image :
 - **mediaTypes** : Je limite les médias sélectionnables à des images uniquement.
 - **allowsEditing** : J'autorise l'utilisateur à effectuer des modifications sur l'image sélectionnée.
 - **aspect** : J'ai spécifié le ratio d'aspect de l'image éditée à 4:3.
 - **quality** : J'ai défini la qualité de l'image à 1 (la meilleure qualité).
- La fonction stocke l'image sélectionnée dans la variable image.
- J'utilise une condition pour vérifier si l'utilisateur a annulé la sélection de l'image (**!image.canceled**).

- Si l'image n'a pas été annulée, j'utilise `setImage(image.assets[0].uri)` pour mettre à jour l'état de l'application avec l'URI de l'image sélectionnée.

Cette fonction **addPicture** est utilisée pour permettre à l'utilisateur de sélectionner une image à partir de la bibliothèque de l'appareil et de la stocker dans l'état de l'application à des fins ultérieures.

Côté serveur (Node.js avec Multer) :

- J'ai utilisé le module Multer pour gérer les fichiers côté serveur. Multer permet de traiter facilement les fichiers envoyés avec les requêtes HTTP.
- J'ai configuré Multer pour spécifier le dossier de destination des fichiers téléchargés et les paramètres de stockage (par exemple, la taille maximale du fichier).
- Lors de la réception d'une demande de remboursement avec un fichier image, j'ai utilisé Multer pour stocker le fichier dans le dossier spécifié sur le serveur.

```
back-end > middleware >  multer-config.js >  <unknown>
Tchessi, 5 months ago | 1 author (Tchessi)
1  const multer = require('multer');
2
3  const MIME_TYPES = {
4    'image/jpg': 'jpg',
5    'image/jpeg': 'jpg',
6    'image/png': 'png',
7  };
8
9  //indication de l'endroit où enregistrer les fichiers entrants et sous quel nom
10 const storage = multer.diskStorage({
11   destination: (req, file, callback) => {
12     console.log(file);
13     callback(null, 'public');
14   },
15   filename: (req, file, callback) => {
16     const name = file.originalname.split(' ').join('_');
17     const extension = MIME_TYPES[file.mimetype];
18     callback(null, name + Date.now() + '.' + extension);
19   },
20 });
21
22 module.exports = multer({ storage: storage }).single('image'); Tchessi, 5 months
23
```

Dans mon code, j'ai utilisé le module **Multer** pour configurer le traitement des fichiers dans le contexte d'un téléchargement d'images sur le serveur.

- J'ai commencé par importer le module Multer et déclaré un objet **"MIME_TYPES"** qui associe les types MIME des fichiers images à leurs extensions correspondantes. Ensuite, j'ai configuré le stockage des fichiers en utilisant **multer.diskStorage()**.
- J'ai spécifié le dossier de destination où les fichiers seront enregistrés et j'ai généré un nom de fichier unique en utilisant l'original name du fichier, un timestamp et l'extension correspondante.
- Enfin, j'ai exporté le middleware Multer en utilisant **multer({ storage: storage }).single('image')**. Cela me permet de traiter un seul fichier image à la fois lorsque j'utilise ce middleware dans une route de mon application. Ce code peut être adapté selon mes propres besoins spécifiques en modifiant, par exemple, le dossier de destination et en ajustant la logique de stockage des fichiers.

Envois de la demande de remboursement en photo :

- Lorsque l'utilisateur soumet la demande de remboursement avec les autres informations (description, montant, etc.), j'ai inclus l'URI de l'image sélectionnée dans les données de la requête.
- J'ai effectué une requête HTTP POST vers mon API en incluant les données de la demande, y compris l'URI de l'image.
- Du côté serveur, j'ai traité la demande de remboursement, enregistrant les informations et le fichier image dans la base de données ou tout autre système de stockage approprié.

En suivant cette approche, j'ai pu permettre aux utilisateurs de capturer ou de sélectionner une image avec **Expo Camera** ou **Expo ImagePicker**, et de l'inclure dans la demande de remboursement de note de frais dans leur message. Multer a été utilisé pour gérer le stockage du fichier image côté serveur.

Documentation

Pour la documentation de mon projet, j'ai utilisé les ressources suivantes :

- **Documentation de Multer** (middleware de gestion des fichiers côté serveur) : J'ai consulté la documentation officielle de Multer sur GitHub : [:https://github.com/expressjs/multer](https://github.com/expressjs/multer) . Cette documentation m'a fourni des informations détaillées sur la configuration et l'utilisation de Multer dans mon projet.
- **Documentation d'Expo Camera** (capture d'images avec React Native) : J'ai exploré la documentation officielle d'Expo Camera : [:https://docs.expo.dev/versions/latest/sdk/camera/](https://docs.expo.dev/versions/latest/sdk/camera/) . Cette documentation m'a fourni des exemples de code, des guides étape par étape et des explications sur les différentes options disponibles pour la capture d'images.
- **Documentation d'Expo ImagePicker** (sélection d'images depuis la galerie avec React Native) : J'ai consulté la documentation officielle d'Expo ImagePicker : <https://docs.expo.dev/versions/latest/sdk/imagepicker/> . Cette documentation m'a aidé à comprendre comment permettre aux utilisateurs de sélectionner des images depuis la galerie de leur appareil. Elle m'a fourni des méthodes d'utilisation détaillées, des options de configuration et des retours de données retournés par Expo ImagePicker.
- **Documentation d'Axios** (bibliothèque HTTP pour les appels réseau) : J'ai consulté la documentation officielle d'Axios : <https://axios-http.com/docs/intro> . Cette documentation m'a fourni des exemples de code, des explications sur la configuration des en-têtes, la gestion des erreurs et d'autres fonctionnalités avancées d'Axios pour effectuer des appels HTTP dans mon application.
- **Documentation officielle de React Native** : <https://reactnative.dev/>

En utilisant ces ressources, j'ai pu mieux comprendre les fonctionnalités spécifiques de mon projet, configurer correctement les bibliothèques utilisées et résoudre les problèmes éventuels rencontrés lors du développement.

Tests

Pour garantir la qualité des données exposées par mon API, j'ai utilisé Postman et Jest pour effectuer des tests à chaque modification ou création de route.

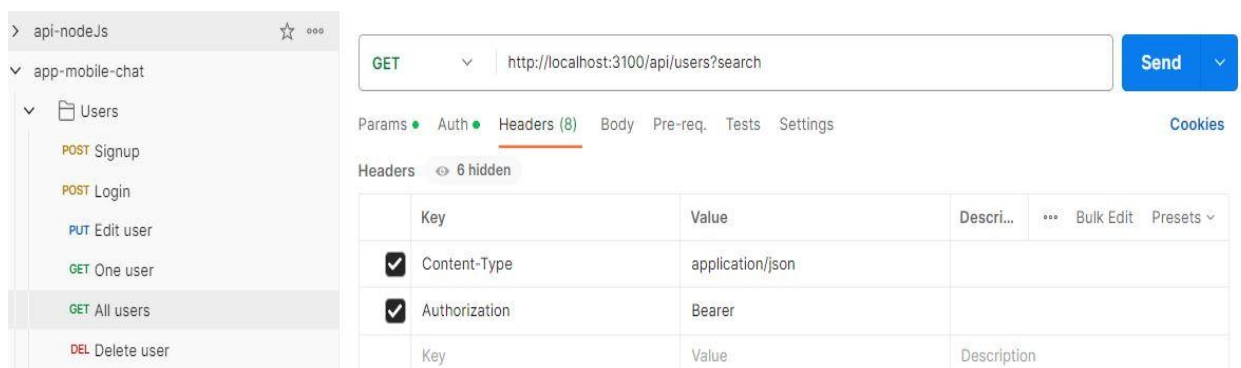
POSTMAN

Lors de chaque test, j'ai vérifié que les données envoyées correspondaient à celles attendues et que les réponses renvoient les données correctes. J'ai également vérifié que le statut de la requête HTTP était correct et que les erreurs étaient gérées de manière appropriée. Postman m'a permis de créer des tests d'intégration pour m'assurer du bon fonctionnement de mon API dans son ensemble.

Les tests sont exécutés automatiquement à chaque fois que j'envoie une requête, ce qui me permet d'avoir une vue d'ensemble de ce qui se passe. La syntaxe des tests avec Postman est verbeuse mais intuitive, elle ressemble aux bibliothèques d'assertions couramment utilisées en JavaScript. Je peux facilement tester le statut de la réponse et les données reçues.

De plus, j'ai créé des variables d'environnement dans Postman qui sont stockées et utilisées entre chaque requête. Cela facilite la gestion des données de test et permet de les réutiliser facilement dans différents scénarios.

En utilisant **Postman**, j'ai pu effectuer des tests approfondis sur mon API, m'assurer de la qualité des données exposées et détecter rapidement les éventuels problèmes ou erreurs.



JEST

Dans mon projet entièrement écrit en JavaScript, j'accorde une grande importance à une couverture de test suffisante. C'est pourquoi j'utilise **Jest**.

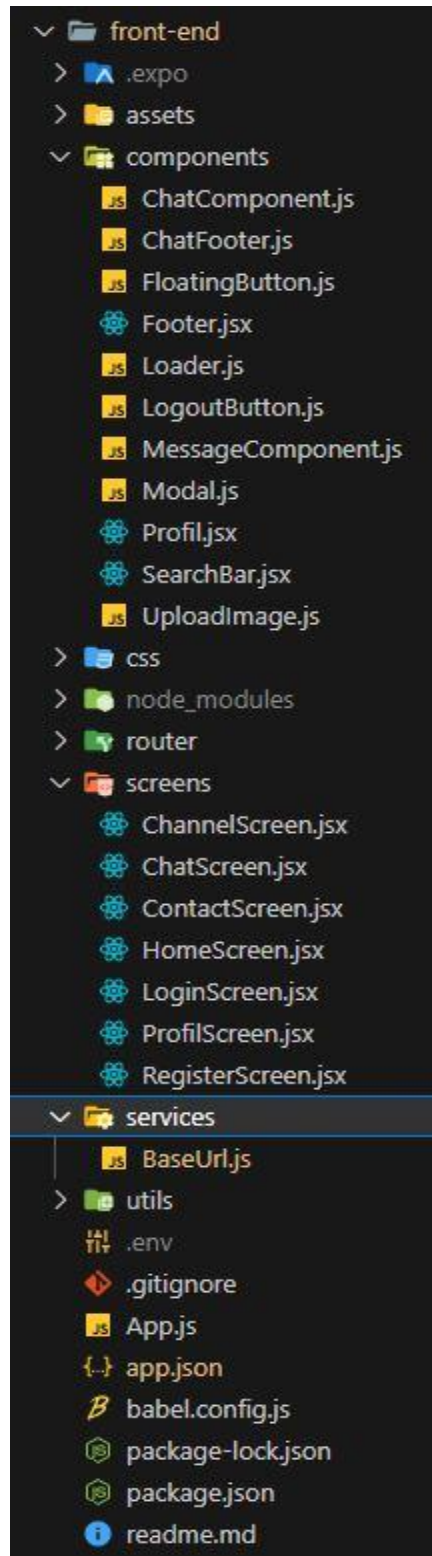
Jest est une bibliothèque JavaScript conçue spécifiquement pour les tests front-end et back-end. Je l'utilise principalement pour tester les services de mon projet. En utilisant Jest, je suis en mesure de créer des suites de tests pour les services, de définir des assertions pour vérifier le comportement attendu de mes fonctions, et d'exécuter des tests automatisés pour m'assurer du bon fonctionnement de mon code.

Jest offre une syntaxe simple et expressive pour écrire des tests, et il fournit également des fonctionnalités supplémentaires telles que le mocking des dépendances et la prise en charge des tests asynchrones.

En utilisant Jest, je peux m'assurer que mes services fonctionnent correctement, que les fonctionnalités sont bien implémentées et que les résultats des tests correspondent aux attentes. Cela me donne confiance dans la qualité et la fiabilité de mon code JavaScript.

Développement du front-end de l'application

Arborescence



Dans mon projet front-end, j'ai organisé les dossiers de la manière suivante :

- Dans le dossier "**assets**", j'ai stocké toutes les ressources statiques nécessaires à mon application, comme les images, les icônes et les polices.
- Le dossier "**components**" contient les composants réutilisables que j'ai développés pour mon application. Ces composants peuvent être utilisés à plusieurs endroits dans mon application.
- J'ai utilisé le dossier "**css**" pour stocker mes fichiers de style. J'y ai inclus des fichiers CSS ou des préprocesseurs CSS si j'en ai utilisé.
- Pour la gestion des routes de mon application, j'ai créé le dossier "**router**". J'y ai placé les fichiers de configuration et de gestion des routes, notamment si j'utilise une bibliothèque de routage telle que React Router.
- Les différents écrans ou pages de mon application se trouvent dans le dossier "**screens**".

J'ai organisé mes fichiers en fonction des différentes fonctionnalités de mon application.

J'ai regroupé les services ou les utilitaires nécessaires pour interagir avec des API externes, effectuer des appels réseau ou gérer l'authentification dans le dossier "**services**". Le dossier "**utils**" contient des fichiers utilitaires et des fonctions

réutilisables que j'ai développés et qui peuvent être utilisés à travers différentes parties de mon application.

- J'ai ajouté un fichier "**env**" pour stocker les variables d'environnement spécifiques à mon application, telles que les clés d'API ou les URL de serveur.
- J'ai inclus un fichier "**.gitignore**" pour spécifier les fichiers et dossiers que je souhaite ignorer lors des opérations de suivi et de commit avec Git.
- Le fichier "**app.js**" est le point d'entrée de mon application et il définit la structure globale de mon interface utilisateur.
- J'ai utilisé le fichier "**app.json**" pour configurer l'application, y compris des informations telles que le nom, la version et les icônes.
- J'ai configuré Babel en utilisant le fichier "**babel.config.js**" pour compiler mon code JavaScript si nécessaire.
- Les fichiers "**package-lock.json**" et "**package.json**" contiennent les dépendances de mon projet ainsi que les scripts de démarrage, de construction et de test.
- Enfin, j'ai inclus un fichier "**README.md**" qui sert de documentation pour mon projet, où je peux expliquer son fonctionnement, donner des instructions d'installation et d'utilisation, etc.

Cette structure de projet m'a permis de maintenir une organisation claire et logique de mes fichiers, ce qui facilite la collaboration et la maintenance de mon code.

Pages et composants

Dans mon projet, j'ai utilisé les dossiers "pages" que j'ai renommé screens et "composants" pour organiser mes fichiers de pages et de composants. En tant que développeur, je trouve cette structure très utile pour maintenir une organisation claire et cohérente du code.

-Dans le dossier "screens", j'ai créé des fichiers ou des répertoires pour chaque page de mon application. Par exemple, j'ai un fichier:

- **HomeScreen.jsx**: Ce fichier représente l'écran ou la page d'accueil principale de votre application. Il peut contenir des informations, des fonctionnalités ou des modules importants de l'application pour donner aux utilisateurs un point de départ central.
- **LoginScreen.jsx** : Ce fichier représente l'écran ou la page de connexion de votre application. Il contient généralement des champs de saisie pour les identifiants de connexion (nom d'utilisateur, mot de passe, etc.) et un bouton de connexion pour permettre aux utilisateurs de s'authentifier.
- **ProfilScreen.jsx** : Ce fichier représente l'écran ou la page de profil de l'utilisateur connecté dans votre application. Il affiche les informations de profil de l'utilisateur, telles que son nom, son image de profil.

- Le dossier "**components**" contient les composants réutilisables de mon application, tels que les boutons, un Modal, les cartes, etc. Ces composants peuvent être utilisés à la fois par les pages et les écrans pour favoriser la réutilisabilité du code. Par exemple:

- **ChatFooter.js** : Ce composant représente le pied de page d'une fonctionnalité de chat. Il peut contenir des éléments tels que la zone de saisie du message et le bouton d'envoi.
- **FlottingButton.js** : Ce composant représente un bouton flottant qui peut être utilisé pour déclencher une action spécifique dans votre application.
- **Footer.js** : Ce composant représente un pied de page générique qui peut être utilisé à travers différentes pages de votre application.

- **Loader.js** : Ce composant affiche un indicateur de chargement ou de chargement en cours pour donner à l'utilisateur une indication visuelle qu'une opération est en cours.
Logout.js : Ce composant représente un bouton ou un élément qui permet à l'utilisateur de se déconnecter de l'application. Il peut être utilisé dans le menu de l'application ou dans d'autres parties de l'interface utilisateur.

Sécurité

Dans le but de renforcer la sécurité de mon application, j'ai pris la décision d'ajouter une couche de sécurité côté front-end. Pour cela, j'ai pris des mesures de sécurité pour rendre par exemple la page d'inscription plus sécurisée.

Voici un résumé des mesures de sécurité mises en place :

- **Validation des champs** : J'ai inclus des vérifications pour s'assurer que les champs obligatoires sont remplis, tels que le prénom, le nom, l'e-mail et le mot de passe. Cela aide à empêcher les soumissions de formulaires incomplets ou incorrects.
- **Comparaison des mots de passe** : J'ai ajouté une vérification pour s'assurer que le mot de passe et la confirmation du mot de passe correspondent. Cela réduit les risques d'erreur lors de la saisie du mot de passe et garantit que l'utilisateur saisit correctement le mot de passe souhaité.
- **Protection contre les erreurs côté client** : J'ai capturé les éventuelles erreurs lors de la requête d'inscription et j'ai affiché un message d'erreur approprié à l'utilisateur. Cela permet de gérer les erreurs de manière sécurisée sans divulguer d'informations sensibles.
- **Utilisation d'Axios** : J'ai utilisé la bibliothèque Axios pour effectuer des requêtes HTTP sécurisées vers l'API back-end. Axios prend en charge l'utilisation de certificats SSL et permet de gérer les en-têtes de sécurité pour les demandes, ce qui renforce la sécurité des communications entre le client et le serveur.

- **Protection des données sensibles** : J'ai utilisé la fonctionnalité "**secureTextEntry**" pour masquer les caractères du mot de passe lors de la saisie. Cela aide à protéger les données sensibles, comme le mot de passe, contre les regards indiscrets.

Il est important de noter que la sécurité d'une application ne se limite pas seulement à ces mesures mises en place dans cette page d'inscription. La sécurité est un processus continu et doit être abordée de manière holistique, en mettant en place des mesures de sécurité tout au long de l'application, y compris côté serveur.

Problématique rencontrées

Au cours du développement de mon projet, j'ai été confronté à plusieurs problématiques que j'ai dû résoudre. Voici quelques exemples de problèmes rencontrés :

- **Gestion des erreurs** : L'une des principales problématiques que j'ai rencontrées concerne la gestion des erreurs. Il était important de mettre en place un système de gestion des erreurs efficace pour fournir des messages d'erreur clairs et informatifs aux utilisateurs en cas de problème. J'ai utilisé des techniques telles que la validation des données, la gestion des exceptions et les messages d'erreur personnalisés pour résoudre cette problématique.
- **Compatibilité des appareils** : Mon application devrait fonctionner sur différents appareils et plates-formes, ce qui a posé des défis de compatibilité. J'ai dû m'assurer que l'application était correctement testée sur une variété d'appareils et que les fonctionnalités étaient compatibles avec les différentes versions d'OS et de navigateurs. J'ai également utilisé des outils de développement et de débogage pour détecter les problèmes spécifiques à certaines plates-formes.
- **Sécurité** : La sécurité était une préoccupation majeure tout au long du développement de mon projet. J'ai dû mettre en place des mesures de sécurité telles que l'authentification, la gestion des autorisations, la protection contre les attaques XSS et CSRF, le chiffrement des données sensibles, etc. J'ai également effectué des tests de sécurité pour détecter et résoudre les vulnérabilités potentielles.

Dans mon projet, j'ai également utilisé la bibliothèque Socket.io pour permettre la communication en temps réel entre le client et le serveur. Cependant, l'utilisation de Socket.io a également posé certaines problématiques spécifiques que j'ai dû résoudre. En voici quelques exemples :

- **Gestion des événements** : L'une des problématiques majeures était de gérer efficacement les événements côté client et côté serveur. J'ai dû définir les événements appropriés pour les différentes actions et établir une logique de traitement des événements côté serveur. J'ai également dû gérer la réception et le traitement des événements côté client pour garantir une communication fluide entre les deux parties.
- **Gestion des connexions** : **Socket.io** gère automatiquement les connexions et les déconnexions des clients, mais j'ai dû m'assurer de gérer ces événements correctement. J'ai mis en place des mécanismes pour détecter les connexions . Cela permet d'assurer une expérience utilisateur fluide et de maintenir des connexions stables.

En surmontant ces problématiques, j'ai pu intégrer avec succès la communication en temps réel dans mon application en utilisant Socket.io. Cela a permis d'offrir une expérience interactive et dynamique aux utilisateurs.

Exemple navigation imbriquées

```
front-end > App.js > Auth
You, 3 weeks ago | 2 authors (Tchessi and others)
1 import React, { useState, useEffect } from "react";
2 import AnimatedSplash from "react-native-animated-splash-screen";
3 import { StyleSheet, Animated, View, StatusBar } from 'react-native';
4 import { NavigationContainer } from '@react-navigation/native';
5 import { createNativeStackNavigator } from '@react-navigation/native-stack';
6 import { createBottomTabNavigator } from '@react-navigation/bottom-tabs';
7 import AsyncStorage from '@react-native-async-storage/async-storage';
8 import HomeScreen from "../screens/HomeScreen";
9 import LoginScreen from "../screens/LoginScreen";
10 import RegisterScreen from "../screens/RegisterScreen";
11 import ContactScreen from "../screens/ContactScreen";
12 import ChannelScreen from "../screens/ChannelScreen";
13 import SettingsScreen from "../screens/ProfilScreen";
14 import ChatScreen from "../screens/ChatScreen";
15 import MaterialCommunityIcons from 'react-native-vector-icons/MaterialCommunityIcons';
16 import FlashMessage from "react-native-flash-message";
```

Dans cet exemple, vous pouvez voir une utilisation de la navigation imbriquée avec React Navigation dans une application React Native. Le code est organisé en plusieurs fichiers, notamment les écrans (screens) et les composants nécessaires à la navigation. Le composant principal est **StackNavigate**, qui gère la navigation de l'application. Il contient un état de loading pour afficher un écran de chargement pendant que l'application se charge. L'état `isLoggedIn` est utilisé pour vérifier si l'utilisateur est connecté ou non. En fonction de l'état, l'application affiche l'écran d'authentification (Auth) ou l'écran principal avec la navigation imbriquée. Le composant Auth utilise **createNativeStackNavigator** pour gérer la navigation entre les écrans d'inscription (**RegisterScreen**), de connexion (**LoginScreen**) et d'accueil (**HomeScreen**). Chaque écran a ses propres options de navigation, telles que la personnalisation de l'en-tête et la désactivation de l'affichage de l'en-tête. Ensuite, le composant StackNavigate utilise `createNativeStackNavigator` pour gérer la navigation entre les écrans principaux de l'application, tels que **ChannelScreen**, **ContactScreen**, **ChatScreen** et **SettingsScreen**. Chaque écran est associé à une icône de tab bar et à ses propres options de navigation. Enfin, dans le composant App, la navigation est enveloppée dans le composant **NavigationContainer** de React Navigation. Cela permet de gérer la navigation globale de l'application. Le composant **FlashMessage** est également ajouté pour afficher des messages flash à l'utilisateur. Dans l'ensemble, cet exemple montre comment utiliser la navigation imbriquée avec React Navigation pour créer une structure de navigation dans une application React Native. Chaque écran est défini dans un composant séparé, ce qui permet une meilleure organisation du code et une gestion plus facile de la navigation entre les écrans.

Exemple du code:

```

18  const Auth = () => {
19    // Stack Navigator for Login and Sign up Screen
20    return (
21      <Stack.Navigator >
22        <Stack.Screen
23          name="Home"
24          component={HomeScreen}
25          options={{
26            headerStyleInterpolator: forFade,
27            headerTintColor: 'white',
28            headerStyle: { backgroundColor: '0F1828#' },
29            title: 'Home',
30            headerShown: false,
31            tabBarStyle: { display: "none" },
32          }}
33        />
34        <Stack.Screen
35          name="Login"
36          component={LoginScreen}
37          options={{
38            headerStyleInterpolator: forFade,
39            headerTintColor: 'white', | Tchessi, 5 months
40            headerStyle: { backgroundColor: '0F1828#' },
41            title: 'Connexion',
42            tabBarStyle: { display: "none" },
43            headerStyle: { backgroundColor: '#0F1828' },
44            headerShown: false,
45          }}
46        />
47        < Stack.Screen
48          name="Register"
49          component={RegisterScreen}
50          options={{
51            headerStyleInterpolator: forFade,
52            headerTintColor: 'white',
53            headerStyle: { backgroundColor: '#0F1828' },
54            title: 'Inscription',
55            headerShown: false,
56            tabBarStyle: { display: "none" },
57          }}
58        />
59      </Stack.Navigator>
60    );
61  };
62

```

Exemple de formulaire de mise à jour du profil

```

16  const ProfilScreen = ({ route }) => {
17    const { userId } = route.params;
18    const [userfirstName, setUserfirstName] = useState('');
19    const [userlastName, setUserlastName] = useState('');
20    const [userEmail, setUserEmail] = useState('');
21    const [userProfileImageUrl, setUserProfileImageUrl] = useState(null);
22    const [currentUserId, setCurrentUserId] = useState(null);
23    const [currentUser, setCurrentUser] = useState('');
24
25
26    const [firstName, setFirstName] = useState('');
27    const [lastName, setLastName] = useState('');
28    const [isEditing, setIsEditing] = useState(false);
29
30
31    const getUser = async () => {
32      try {
33        const token = await AsyncStorage.getItem('token');
34        const decodedToken = jwt_decode(token);
35        setCurrentUser(decodedToken.userId);
36        let response = await axios.get(`${API_URL}api/users/${userId}`, {
37          headers: {
38            'Authorization': `Bearer ${token}`,
39          },
40        });
41        if (response.status === 200) {
42          setFirstName(response.data.user.firstName)
43          setLastName(response.data.user.lastName)
44          setUserfirstName(response.data.user.firstName);
45          setUserlastName(response.data.user.lastName);
46          setUserEmail(response.data.user.email);
47          setUserProfileImageUrl(response.data.user.imageUrl);
48          setCurrentUserId(response.data.user.id)
49        }
50      } catch (error) {
51      }
52    };
53  }

```



```
61
62 ✓ const handleEdit = async () => {
63 ✓   if (!isEditing) {
64     setIsEditing(true);
65     return;
66   }
67
68 ✓   if (firstName === '') {
69 ✓     showMessage({
70       message: 'Veuillez saisir votre prénom',
71       type: 'warning',
72       duration: 3000,
73       position: 'top',
74       floating: true,
75       style: { marginTop: 30 },
76     });
77     return;
78   }
79 ✓   if (lastName === '') {
80 ✓     showMessage({
81       message: 'Veuillez saisir votre nom',
82       type: 'warning',
83       duration: 3000,
84       position: 'top',
85       floating: true,
86       style: { marginTop: 30 },
87     });
88     return;
89   }
90
91 ✓   try {
92     const token = await AsyncStorage.getItem('token');
93     const response = await axios.put(
94       `${API_URL}api/auth/edit`,
95       {
96         firstName: firstName,
97         lastName: lastName,
98       },
99       {
100        headers: {
101          Authorization: `Bearer ${token}`,
102        },
103      }
104     );
105
106 ✓   if (response.status === 200) {
107     showMessage({
```

Dans cet exemple de formulaire de mise à jour du profil, nous utilisons React et React Native. Voici comment le code fonctionne :

- J'ai importé les dépendances nécessaires, y compris les composants et les bibliothèques externes.
- Le composant `ProfilScreen` reçoit la prop `route` qui contient les paramètres de la navigation, notamment l'ID de l'utilisateur.
- Nous utilisons plusieurs hooks `useState` pour gérer l'état des différentes variables, telles que le prénom, le nom, l'image de profil, l'ID de l'utilisateur actuel, etc.
- La fonction `getUser` est appelée lors du montage du composant et récupère les informations de l'utilisateur à partir de l'API en utilisant l'ID fourni. Les informations sont ensuite stockées dans l'état du composant.
- Le composant affiche les informations de l'utilisateur, telles que le nom, l'e-mail et l'image de profil. Si aucune image de profil n'est disponible, un conteneur avec les initiales de l'utilisateur est affiché.
- Si l'utilisateur actuel correspond à l'ID de l'utilisateur affiché, un formulaire de modification est affiché avec des champs de saisie pour le prénom et le nom. Les champs de saisie sont modifiables si l'utilisateur est en mode d'édition.
- Lorsque l'utilisateur appuie sur le bouton "**Modifier**", la fonction `handleEdit` est appelée. Si le composant est en mode d'édition, les valeurs des champs de saisie sont vérifiées et envoyées à l'API pour mettre à jour le profil de l'utilisateur. Si la mise à jour réussit, un message de succès est affiché. Si une erreur se produit, un message d'erreur est affiché.

Conception de l'espace administrateur

Conception de la partie administration

Dans le cadre du développement de la partie administration de mon site web, j'ai utilisé la bibliothèque Material-UI.

Material-UI est une bibliothèque de composants d'interface utilisateur basée sur les principes de design de Material Design de Google. En utilisant Material-UI, j'ai pu bénéficier d'un large éventail de composants prêts à l'emploi, tels que des boutons, des formulaires, des tables, des cartes, des icônes, etc. Ces composants sont conçus de manière cohérente, offrant une expérience utilisateur moderne et esthétique. L'avantage d'utiliser Material-UI est que les composants sont hautement personnalisables et réutilisables. J'ai pu les intégrer facilement dans mon application et les adapter en fonction des besoins spécifiques de la partie administration. Cela m'a fait gagner du temps et m'a permis de maintenir une cohérence visuelle dans toute l'interface. Material-UI propose également des fonctionnalités avancées telles que la gestion des thèmes, la gestion de l'état, les transitions animées, les mises en page responsives, etc. Ces fonctionnalités ont été très utiles pour créer une expérience utilisateur fluide et réactive dans la partie administration. En résumé, l'utilisation de Material-UI dans la conception de la partie administration de mon site web m'a permis de bénéficier de composants prêts à l'emploi, personnalisables et esthétiques, facilitant ainsi le développement d'une interface utilisateur cohérente et moderne.

User Story

En tant qu'administrateur, Je souhaite pouvoir gérer les utilisateurs de l'application de chat, afin de contrôler les accès et de maintenir un environnement sécurisé. En tant qu'administrateur, je peux afficher la liste de tous les utilisateurs enregistrés dans l'application de chat. Je peux rechercher des utilisateurs spécifiques en utilisant des filtres tels que leur nom, leur adresse e-mail, etc. Je peux afficher les détails d'un utilisateur, tels que son nom, son adresse e-mail, sa date d'inscription, etc. Je peux modifier les informations d'un utilisateur, telles que son nom, son adresse e-mail, son rôle, etc. Je peux désactiver ou supprimer un utilisateur de l'application de chat. Je peux gérer les autorisations des utilisateurs, telles que

la possibilité de voir la salle de chat, de supprimer des messages, etc. En tant qu'administrateur, Je peux gérer les salles de chat de l'application, afin de fournir une structure organisée pour les utilisateurs. Critères d'acceptation : Je peux afficher la liste de tous les messages de chat disponibles dans l'application. Je peux créer un nouveau message dans chat. Je peux supprimer une salle de chat de l'application. Je peux gérer les membres d'une salle de chat, tels que les ajouter, les supprimer, leur attribuer des rôles, etc.

Choix du langage et Framework

J'ai fait le choix d'utiliser **Node.js** avec **Express.js** pour le développement du backend de mon application de chat.

Node.js offre une exécution côté serveur rapide et événementielle, ce qui est parfait pour les applications en temps réel comme les chats. Avec Express.js, je peux facilement créer des routes et des gestionnaires de requêtes.

Pour le frontend, j'ai opté pour React avec **Material-UI**. React est une bibliothèque JavaScript populaire qui me permet de créer des interfaces utilisateur réactives et réutilisables. Les hooks de React simplifient la gestion de l'état et du cycle de vie des composants.

Material-UI est une bibliothèque de composants pour React qui suit les principes du **design matériel de Google**. Elle offre des composants esthétiques et réactifs prêts à l'emploi, ce qui facilite le développement de l'interface utilisateur de mon application de chat. J'apprécie également la flexibilité de Material-UI pour personnaliser les styles et les thèmes selon mes besoins.

En combinant Node.js et Express.js pour le backend avec React et Material-UI pour le frontend, j'ai une stack technologique complète pour développer une application de chat réactive et conviviale. Je veille à suivre les meilleures pratiques de sécurité en gérant les données utilisateur et en mettant en place des mécanismes d'authentification appropriés pour garantir la confidentialité et l'intégrité des conversations.

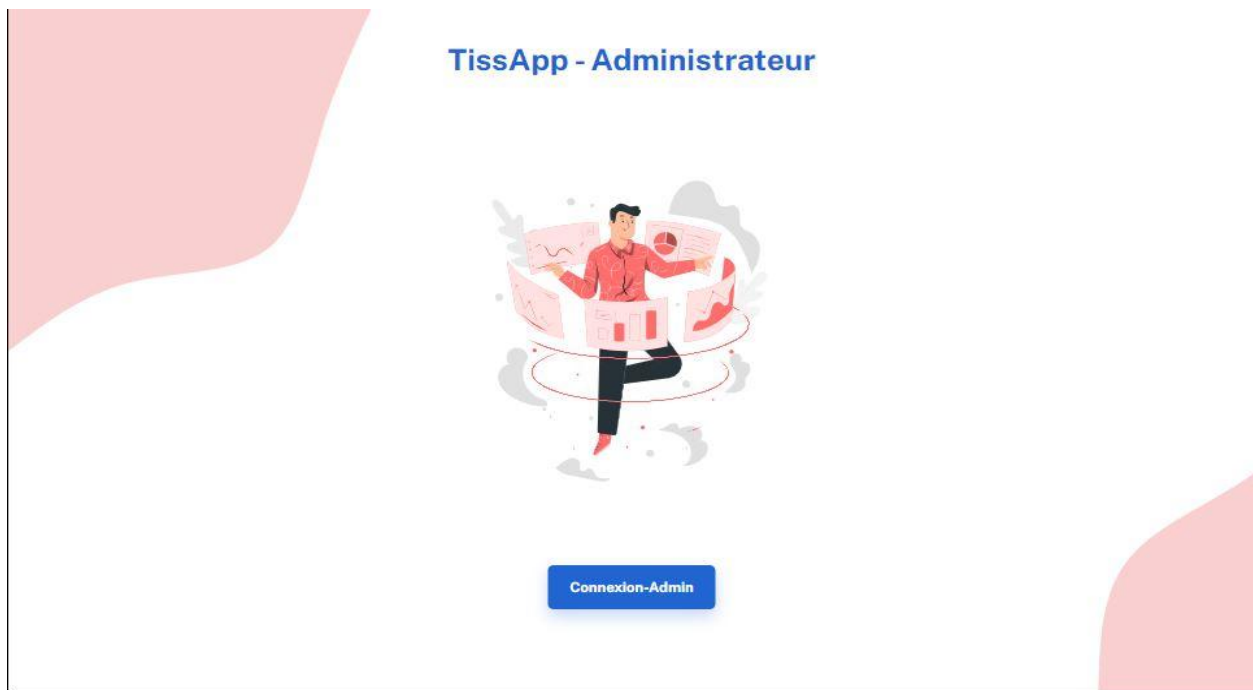
Conception du front-end de l'application du site web

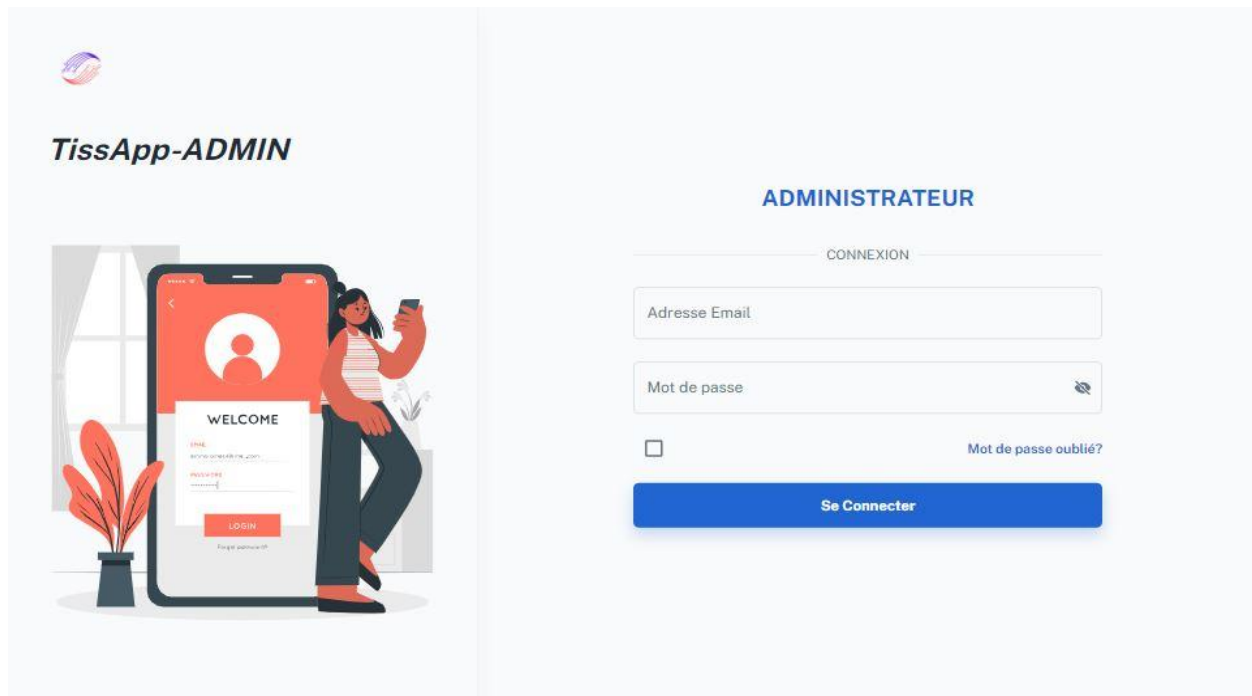
→ Charte graphique

Dans un souci de cohérence, j'ai choisi d'utiliser la même charte graphique pour mon site web que celle de l'application mobile présentée précédemment. Cela permet aux utilisateurs de retrouver une expérience visuelle familière, que ce soit sur l'application mobile ou sur le site web. Les couleurs, les polices et les éléments visuels utilisés sont donc les mêmes, ce qui renforce l'identité visuelle de l'ensemble du projet. Cette cohérence visuelle contribue également à offrir une expérience utilisateur harmonieuse et unifiée, que les utilisateurs naviguent sur l'application mobile ou visitent le site web.

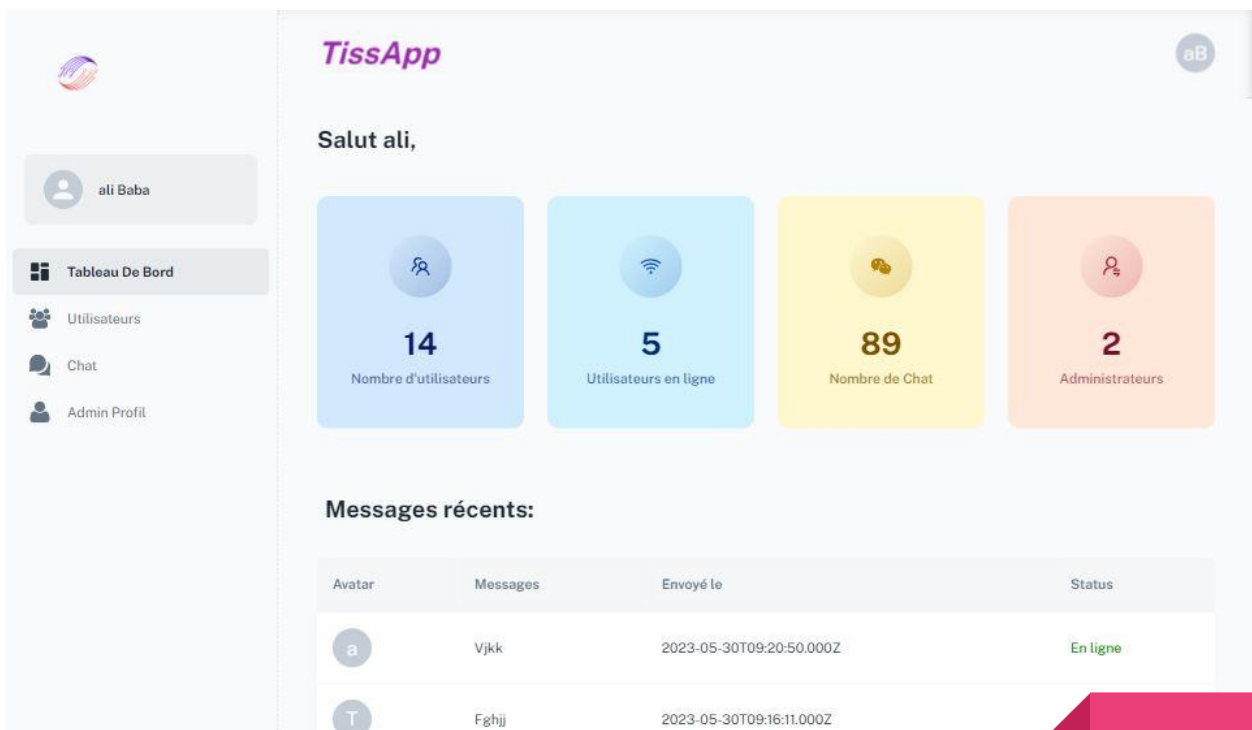
→ Maquettage

Page d'accueil:





Page de connexion:


The login page for TISSApp-ADMIN is divided into two main sections. On the left, there is a large illustration of a woman standing next to a giant smartphone. The phone screen displays a 'WELCOME' message, a login form with fields for 'EMAIL' and 'PASSWORD', and a 'LOGIN' button. On the right, the 'ADMINISTRATEUR' login form is displayed. It includes a 'CONNEXION' header, input fields for 'Adresse Email' and 'Mot de passe' (with an eye icon for visibility), a checkbox for 'Mot de passe oublié?' (Forgot password?), and a blue 'Se Connecter' button.

Tableau de bord:


The dashboard for TISSApp shows the user 'ali Baba' and provides a quick overview of system statistics. The statistics are presented in four colored cards: 14 users, 5 users online, 89 chat messages, and 2 administrators. Below these, a 'Messages récents' section displays a table of recent messages.

Avatar	Messages	Envoyé le	Status
	Vjkk	2023-05-30T09:20:50.000Z	En ligne
	Fghij	2023-05-30T09:16:11.000Z	

Page utilisateurs:

TissApp aB

Utilisateurs (14) + Ajouter

Recherche...

ID	Avatar	Nom	Prénom	Email	Rôle	Status	Créé le	Modifié le
3		Preza	Tiss	tiss@test.fr	Admin	En ligne	2023-04-15T13:30:13.000Z	2023-05-30T08:38:51.000Z
4		Baba	ali	ali@test.fr	Admin	En ligne	2023-04-15T14:08:11.000Z	2023-05-30T09:20:31.000Z
5		Pre	Mariama	mariama@test.fr	Utilisateur	Hors ligne	2023-04-17T11:48:09.000Z	2023-04-17T12:07:11.000Z
6		Preza	Tchess	tchesspreza@yahoo.fr	Utilisateur	Hors ligne	2023-04-17T14:02:33.000Z	2023-04-30T10:59:55.000Z

Page de chat:

TissApp aB

Chat (89) + Message

Avatar	Nom & prenom	Messages envoyés	Date d'envoi	Action
aB	ali Baba	Vjkk	2023-05-30T09:20:50.000Z	Supprimer
TP	Tiss Preza	Fghij	2023-05-30T09:16:11.000Z	Supprimer
TP	Tiss Preza	Xghhj	2023-05-30T09:14:34.000Z	Supprimer
TP	Tiss Preza	Hdyjd	2023-05-30T09:13:56.000Z	Supprimer
TP	Tiss Preza	Gdjfid	2023-05-30T08:51:39.000Z	Supprimer

Lignes par page: 5

Conception du backend du site web

Dans la partie backend de ce site, j'utilise la même API que celle utilisée par l'application mobile. Cela permet d'avoir une seule source de données et de fonctionnalités pour les deux interfaces. En utilisant la même API, je peux réutiliser le code existant et éviter de dupliquer la logique métier. Cela simplifie également la maintenance, car les mises à jour et les améliorations apportées à l'API bénéficient à la fois au site web et à l'application mobile. En utilisant une approche basée sur des services, le backend fournit les fonctionnalités nécessaires pour gérer les utilisateurs, les données et les opérations spécifiques à la partie administrative du site web.

Conclusion

En conclusion, mon projet professionnel a consisté à développer une application de chat avec une partie mobile et une partie administration sur un site web. J'ai utilisé les technologies modernes telles que React Native, React JS, Node.js et Express pour créer une solution complète et cohérente.

L'application mobile offre une expérience conviviale aux utilisateurs pour se connecter, interagir et échanger des messages. J'ai utilisé des concepts tels que les composants, la navigation, les services et les API pour créer une interface réactive et intuitive.

La partie administration sur le site web permet aux administrateurs de gérer les utilisateurs, de visualiser les statistiques, de gérer les conversations et d'accéder à d'autres fonctionnalités avancées. J'ai utilisé des technologies comme React JS et Material-UI pour créer une interface élégante et conviviale pour les administrateurs.

J'ai accordé une attention particulière à la sécurité en utilisant des pratiques de sécurité recommandées, notamment l'authentification basée sur JWT, la validation des données et la protection contre les attaques courantes. Le projet a été organisé en utilisant des méthodologies de développement agiles telles que les user stories, les sprints et les tests. J'ai également utilisé des outils tels que Postman pour tester l'API, Jest pour les tests unitaires et la documentation pour assurer la qualité du code et la facilité de maintenance.

Dans l'ensemble, ce projet professionnel m'a permis de mettre en pratique mes compétences en développement web et mobile, ainsi que d'acquérir de nouvelles connaissances et compétences. Il m'a également familiarisé avec les bonnes pratiques de développement, la gestion de projet et la collaboration en équipe. Je suis satisfait du résultat final et confiant dans la valeur ajoutée de cette application de chat dans un contexte professionnel.

Annexe

Annexe 1 : Maquette de l'application *TissApp*

