

Dossier de projet professionnel

CONCEPTEUR D'APPLICATION WEB ET MOBILE

Réalisation de l'application Web et Mobile



Présenté par Ridha BOUGHEDIRI

SOMMAIRE

Dossier de projet professionnel	1
INTRODUCTION	6
Présentation personnelle	6
Présentation du projet en anglais	6
ORGANISATION ET CAHIER DES CHARGES	8
Analyse de l'existant	8
1. Applications de chat populaires	8
2. Points forts et points faibles de l'existant	8
3. Innovations récentes	9
Composition de l'application	9
1. Partie web pour les administrateurs	9
2. Partie mobile pour les utilisateurs	9
Les fonctionnalités attendues	10
Application mobile	10
1. Page de chat général :	10
2. Page de profil :	10
3. Page d'annuaire :	10
4. Page de connexion et d'inscription :	11
5. Page du listing des conversations d'un utilisateur :	11
6. Page de chat privé :	11
- gestion du channel	11
- Bannissement d'utilisateurs	11
- Droits d'administration	11
- Suppression du channel	12
- Possibilité de quitter le channel	12
Context technique	12
CONCEPTION DU PROJET	13
Choix de développement	13
1. Choix des langages	13
JavaScript en front-end :	13

JavaScript en back-end :	13
2. Choix des framework	15
1. React Native avec Expo (Frontend) :	15
2. Express (Backend) :	16
3. En résumé	16
3. Outils utilisés	16
Organisation du projet	17
1. Diagrammes de Gantt	17
2. Rôles et responsabilités	20
3. GitHub Kanban	21
Utilisation des Milestone (jalons)	21
Utilisation des Labels	22
Les différents type de label utilisé dans notre méthodes :	23
Evaluation des priorités	24
Estimation de la vitesse	25
Architecture logicielle	25
CONCEPTION FRONT-END	26
Arborescence du projet	26
Charte graphique	26
Design System	27
Maquettage	28
CONCEPTION BACKEND DE L'APPLICATION	29
La base de données	29
Mise en place de la base de données:	29
Conception de la base de données:	29
Modèle Conceptuel de donnée (MCD)	30
Modèle Logique de donnée (MLD)	30
Modèle Physique de données	31
DÉVELOPPEMENT DU BACKEND DE L'APPLICATION	32
Organisation	32

Arborescence	33
Fonctionnement de l'API	33
Architecture de l'API	34
Les différents statuts code utilisés dans le projets	34
Les différentes méthodes HTTP utilisées	35
Middleware	35
Routage	36
Sequelize	38
1. Object-Relational Mapping (ORM)	38
2. Prise en charge multi-base de données	38
3. Modélisation des données	39
4. Opérations CRUD	39
5. Migrations de base de données	39
6. Hooks et Validations	39
Connexion à la base de données	39
Controller	43
Model	44
Sécurité	44
les risques	45
Les solutions utilisées	45
Chiffrement des données sensibles :	45
JWT :	45
Gestion des Droits	46
Validation des données	47
Helmet:	48
DÉVELOPPEMENT DU FRONT-END	49
Arborescence	49
Pages et composants	50
Sécurités	52
Problématiques rencontrées	52
Exemple navigation imbriquées	53

CONCEPTION DE L'ESPACE ADMINISTRATEUR	54
Conception de la partie administration	54
User Story	54
Choix du Template, langage et framework	54
Conception du frontend de l'application Web	55
Charte graphique	55
Conception du back-end de l'application Web	55
CONCLUSION	55

INTRODUCTION

Résumé du dossier de projet

Ce dossier présente une partie du travail réalisé lors de ma formation à La Plateforme à Marseille, à savoir la conception et le développement d'un composant d'interface et une application multicouche répartie en intégrant les recommandations de sécurité ; regroupant un ensemble de compétences pour mon titre RNCP de niveau 7.

Vous y retrouverez donc un panel de compétences mises en avant lors de ce projet d'une durée de 5 mois, ainsi que son déroulement, les attentes attendues par la personne chargée de ce projet et sa réalisation finale.

Présentation personnelle

Bonjour je m'appelle Ridha BOUGHEDIRI j'ai 35 ans. J'ai intégré le cursus de la CodingSchool de la plateforme en Septembre 2021. J'ai obtenu mon titre de développeur web et mobile .

Aujourd' Je suis actuellement concepteur / développeur web / web mobile à La Plateforme _ Marseille .

Grâce à cette formation j'ai acquis et compris les concepts suivant :

- Concevoir et développer des composants d'interface utilisateur en intégrant les recommandations de sécurité
- Concevoir et développer une application multicouche répartie en intégrant les recommandations de sécurité.

Curieux et passionné, j'ai trouvé une reconversion professionnelle qui me convient parfaitement. Maintenant je souhaite en faire mon futur métier.

Présentation du projet en français et en anglais

Français :

Pendant nos périodes de formation, du temps a été consacré au développement d'une application mobile, J'ai décidé de partir sur une application de chat entre amis que j'ai appelé Chat us .

Chat us est une application qui permet à un utilisateur ou un groupe d'utilisateurs de communiquer via chat des meilleures équipes de football américain et également les pronostics de chaque match .

L'objectif est de mettre en place une application permettant à des groupes d'amis ou utilisateur solo sans groupe d'amis d'avoir la possibilité de lancer des conversations.

L'utilisateur pourra retrouver chaque conversation postée grâce à un feed qui permettra de consulter toutes les conversations .

Les utilisateurs peuvent à chaque match lancer des conversations .

L'application s'inspire à la fois des applications de tchats traditionnels et des applications de soirée entre amis.

Je voulais une application ludique, avec les mêmes codes que les applications de soirée entre amis sur lesquelles nous nous rendons souvent, tout en apportant une plus-value.

Anglais :

ChatSport is an innovative mobile application designed specifically for fans of American sports. Whether you are a fan of American football, basketball, baseball, or ice hockey, ChatSport provides a real-time discussion platform that allows you to connect with other enthusiasts from around the world.

The application offers a user-friendly and intuitive interface, allowing you to create your profile, select your favorite teams, and access dedicated thematic discussion rooms for each sport. You can also customize your notifications to ensure you don't miss any important news, live scores, expert analyses, and lively discussions.

ChatSport offers a multitude of features to enhance your sports chat experience. You can exchange private messages with other users, form discussion groups, share videos, images, and relevant articles. You can even organize live viewing events and invite your virtual friends to join you in watching the most exciting matches.

The application stands out for its dynamic and engaged community of American sports fans. You can chat with experts, players, coaches, and enthusiasts who share your love for the sport. Share your thoughts, opinions, and analyses, and enjoy a friendly environment where everyone is encouraged to actively participate.

Whether you want to discuss the latest happenings in the NFL, analyze the statistics of your favorite baseball team, or simply share your excitement for the NBA playoffs, ChatSport is the ideal mobile application to connect with other American sports fans and have an interactive and immersive discussion experience.

Download ChatSport now and join a passionate community of American sports fans ready to share their enthusiasm, knowledge, and passion with you. Don't let the games pass you by. Be at the heart of the action with ChatSport

Compétences couvertes par le projet

Ce projet couvre les compétences du titre suivantes:

- Maquetter une application
- Développer des composants d'accès aux données
- Développer la partie front-end d'une interface utilisateur web

-
- Développer la partie back-end 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

ORGANISATION ET CAHIER DES CHARGES

Analyse de l'existant

Dans le cadre de la conception de notre application de chat, il est essentiel de réaliser une analyse approfondie de l'existant afin de comprendre les fonctionnalités déjà disponibles sur le marché et d'identifier les opportunités d'amélioration. Cette analyse nous permettra de définir

les objectifs de notre application et de déterminer les fonctionnalités clés que nous devons inclure pour offrir une expérience utilisateur exceptionnelle.

1. Applications de chat populaires

Nous avons étudié les applications de chat les plus populaires et les mieux établies du marché, telles que WhatsApp, Facebook Messenger et Telegram. Ces applications offrent un large éventail de fonctionnalités, notamment la messagerie instantanée, les appels vocaux et vidéo, les groupes de discussion, les émoticônes, la possibilité de partager des fichiers et des images, etc. Il est important d'examiner attentivement ces fonctionnalités pour déterminer celles que nous devons intégrer dans notre application.

2. Points forts et points faibles de l'existant

L'analyse de l'existant nous a permis d'identifier les points forts et les points faibles des applications de chat existantes. Parmi les points forts, on peut citer :

- Interface utilisateur conviviale et intuitive.
- Capacité à envoyer des messages instantanés à des contacts individuels ou à des groupes.
- Possibilité de passer des appels vocaux et vidéo de haute qualité.
- Fonctionnalités avancées de confidentialité et de sécurité.
- Prise en charge multiplateforme (Web, iOS, Android).

Cependant, nous avons également identifié certains points faibles que nous devons prendre en compte pour améliorer notre application :

- Manque de personnalisation des conversations et des profils d'utilisateurs.
- Difficultés à trouver rapidement des messages ou des informations spécifiques.
- Limitations dans le partage de fichiers et de médias.
- Absence de certaines fonctionnalités de modération et de filtrage du contenu.

3. Innovations récentes

Nous avons également examiné les dernières innovations dans le domaine des applications de chat. Certaines tendances émergentes incluent :

-
- Intégration de chatbots pour une assistance automatisée.
 - Intégration de l'intelligence artificielle pour des suggestions de réponses rapides.
 - Utilisation de la réalité augmentée pour des conversations plus immersives.
 - Fonctionnalités de chiffrement de bout en bout pour une sécurité accrue.

Ces innovations peuvent servir de sources d'inspiration pour le développement de notre application de chat.

Composition de l'application

Notre application de chat est conçue pour répondre aux besoins spécifiques de deux catégories d'utilisateurs : les administrateurs et les utilisateurs. Afin de fournir une expérience optimale à chaque groupe, nous avons développé deux interfaces distinctes : une partie web dédiée aux administrateurs et une partie mobile destinée aux utilisateurs.

1. Partie web pour les administrateurs

La partie web de notre application est spécialement conçue pour permettre aux administrateurs de gérer et de superviser le fonctionnement global du système de chat. Les fonctionnalités clés de la partie web comprennent :

- Gestion des utilisateurs : Les administrateurs peuvent créer, modifier et supprimer des comptes utilisateur, ainsi que gérer les autorisations d'accès.
- Modération des contenus : Les administrateurs ont la possibilité de surveiller les conversations, d'appliquer des filtres de contenu et de prendre des mesures appropriées en cas de violation des règles d'utilisation.
- Statistiques et rapports : La partie web offre des fonctionnalités de génération de rapports et de visualisation des données pour permettre aux administrateurs de suivre l'activité des utilisateurs, les performances du système et d'obtenir des informations précieuses pour améliorer l'expérience globale.

2. Partie mobile pour les utilisateurs

La partie mobile de notre application est conçue pour offrir une expérience conviviale et intuitive aux utilisateurs finaux. Les fonctionnalités principales de la partie mobile comprennent :

-
- Messagerie instantanée : Les utilisateurs peuvent envoyer et recevoir des messages en temps réel, individuellement ou dans des groupes de discussion.
 - Appels vocaux et vidéo : L'application permet aux utilisateurs d'effectuer des appels vocaux et vidéo de haute qualité avec leurs contacts.
 - Partage de fichiers et de médias : Les utilisateurs peuvent partager des fichiers, des images, des vidéos et d'autres types de contenus avec leurs contacts.
 - Personnalisation du profil : Les utilisateurs ont la possibilité de personnaliser leur profil en ajoutant des photos, des informations personnelles et des préférences de confidentialité.

En séparant les fonctionnalités destinées aux administrateurs de celles destinées aux utilisateurs finaux, nous garantissons une interface adaptée aux besoins spécifiques de chaque groupe. Les administrateurs bénéficieront d'outils puissants pour gérer et superviser le système, tandis que les utilisateurs pourront profiter d'une expérience fluide et engageante sur leurs appareils mobiles.

Les fonctionnalités attendues

Application mobile

Notre application de chat vise à offrir une expérience conviviale et pratique pour les utilisateurs. Voici les principales fonctionnalités que nous prévoyons d'inclure :

1. Page de chat général :

Cette page constitue l'espace principal de conversation pour les utilisateurs. Elle permet aux utilisateurs connectés d'interagir avec d'autres membres de la communauté, d'échanger des messages en temps réel et de participer à des discussions de groupe. Les utilisateurs pourront consulter les messages précédents, envoyer des réponses, partager des médias et utiliser des fonctionnalités de modération appropriées.

2. Page de profil :

La page de profil permet à chaque utilisateur de créer et de gérer son propre profil.. Ils auront également la possibilité de gérer les paramètres liés à leur compte.

3. Page d'annuaire :

L'annuaire de l'application regroupe tous les utilisateurs inscrits. Cette page permet aux utilisateurs de découvrir de nouveaux contacts, de rechercher des utilisateurs inscrits.

L'utilisateur peut sélectionner un ou plusieurs membre(s) pour lancer une conversation privée.

4. Page de connexion et d'inscription :

Les pages de connexion et d'inscription offriront aux utilisateurs la possibilité de créer un nouveau compte ou de se connecter à leur compte existant. Les utilisateurs pourront fournir leurs informations personnelles, choisir un nom d'utilisateur et un mot de passe sécurisé. Si un utilisateur dispose déjà d'un token valide, il sera automatiquement redirigé vers la page du chat général lorsqu'il ouvrira l'application. Sinon, il sera dirigé vers la page de connexion pour se connecter ou s'inscrire.

5. Page du listing des conversations d'un utilisateur :

Cette page listera tous les canaux de conversation auxquels l'utilisateur est abonné ou auquel il a accès. L'utilisateur pourra visualiser les channels existants. Cette fonctionnalité permettra aux utilisateurs de lister ses interactions avec les autres utilisateurs.

6. Page de chat privé :

La page de chat privé permet aux utilisateurs d'engager des conversations individuelles ou collectives avec d'autres utilisateurs. Voici les fonctionnalités spécifiques de cette page :

- **gestion du channel**

Lorsqu'un utilisateur crée un channel de chat privé, il devient le super administrateur de ce channel. En tant que super administrateur, l'utilisateur a les droits spéciaux pour gérer le channel. Il peut bannir un utilisateur spécifique de la conversation, en lui interdisant l'accès au channel. De plus, le super administrateur a la possibilité de nommer un autre utilisateur en tant qu'administrateur du channel. Les administrateurs ont des droits étendus pour gérer les membres et les paramètres du channel. Enfin, le super administrateur a le pouvoir de supprimer complètement le channel si nécessaire.

- **Bannissement d'utilisateurs**

En tant que super administrateur, l'utilisateur a le pouvoir de bannir un utilisateur spécifique du channel de chat privé. Lorsqu'un utilisateur est banni, il perd l'accès au

channel et ne peut plus participer à la conversation. Cette fonctionnalité permet de maintenir la sécurité et la gestion appropriées des channels de chat privés.

- **Droits d'administration**

Le super administrateur a la possibilité de nommer d'autres utilisateurs en tant qu'administrateurs du channel. Les administrateurs ont des droits étendus, tels que la gestion des membres, la modification des paramètres du channel et la modération des messages. Cette fonctionnalité permet de déléguer la responsabilité de la gestion du channel à des utilisateurs de confiance.

- **Suppression du channel**

En tant que super administrateur, l'utilisateur a le pouvoir de supprimer complètement le channel de chat privé. Cette action entraînera la suppression de toutes les conversations, des fichiers partagés et des paramètres du channel. Cette fonctionnalité permet aux super administrateurs de gérer de manière appropriée ses canaux inactifs ou obsolètes.

- **Possibilité de quitter le channel**

Chaque utilisateur, y compris les super administrateurs et les administrateurs, a la liberté de quitter un channel de chat privé à tout moment. Lorsqu'un utilisateur quitte le channel, il n'aura plus accès aux messages et aux activités ultérieures du channel. Cette fonctionnalité donne aux utilisateurs la liberté de contrôler leur participation aux conversations privées.

En combinant ces fonctionnalités, notre application offre un environnement de chat privé flexible et personnalisé, où les super administrateurs peuvent gérer les membres, les droits et les paramètres des channels. Les utilisateurs peuvent ainsi engager des conversations privées tout en conservant le contrôle sur leur propre participation.

Nous cherchons à fournir une expérience de chat complète, alliant à la fois des conversations de groupe dynamiques et des échanges privés personnalisés. L'application offrira ainsi aux utilisateurs une plateforme conviviale.

Context technique

L'application mobile devra être accessible sur tous les systèmes d'exploitation Android et IOS.
L'application web devra être accessible sur tous les navigateurs.

CONCEPTION DU PROJET

Choix de développement

1. Choix des langages



JavaScript est un langage de programmation polyvalent qui peut être utilisé à la fois côté client (front-end) et côté serveur (back-end). Voici les avantages de l'utilisation de JavaScript dans ces deux domaines :

JavaScript en front-end :

1. Interaction avec l'interface utilisateur : JavaScript est principalement utilisé pour rendre les pages Web interactives. Il vous permet de manipuler le contenu HTML, de réagir aux actions de l'utilisateur (clics de souris, pressions de touche, etc.) et de mettre à jour dynamiquement l'interface utilisateur sans recharger la page entière.

-
2. Validation des données côté client : JavaScript peut être utilisé pour valider les données côté client avant de les envoyer au serveur. Cela permet de fournir une expérience utilisateur plus réactive en évitant des allers-retours coûteux avec le serveur pour valider les entrées.
 3. Amélioration des performances : JavaScript permet d'effectuer des calculs et des manipulations de données côté client, ce qui réduit la charge sur le serveur et améliore la performance globale de l'application.
 4. Manipulation du DOM : JavaScript offre des fonctionnalités puissantes pour manipuler le Document Object Model (DOM), qui représente la structure HTML d'une page. Cela permet de modifier dynamiquement le contenu, le style et le comportement des éléments de la page.

JavaScript en back-end :

1. Partage de code : En utilisant JavaScript à la fois côté client et côté serveur, vous pouvez partager certaines parties de votre code entre les deux. Cela permet une meilleure réutilisation du code et facilite le développement et la maintenance de l'application.
2. Gestion des requêtes et des réponses : JavaScript côté serveur, avec des environnements tels que Node.js, permet de gérer les requêtes HTTP entrantes et de générer des réponses dynamiques. Vous pouvez construire des API, gérer des sessions utilisateur, effectuer des opérations de base de données et d'autres tâches côté serveur.
3. Manipulation des données : JavaScript offre des bibliothèques et des frameworks robustes pour la manipulation des données côté serveur. Vous pouvez accéder à des bases de données, effectuer des opérations de lecture/écriture, analyser et formater des données, et même effectuer des calculs complexes.
4. Évolutivité et performances : L'utilisation de JavaScript côté serveur permet de tirer parti d'une architecture événementielle et non bloquante. Cela signifie que vous pouvez gérer simultanément de nombreuses connexions sans bloquer le thread principal. Cela améliore les performances de l'application, notamment lors de la gestion d'un grand nombre de requêtes en parallèle.

En utilisant JavaScript à la fois côté client et côté serveur, vous bénéficiez d'une expérience de développement plus homogène et d'une plus grande flexibilité dans la création d'applications Web modernes et interactives.



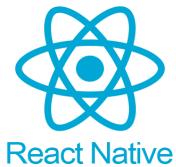
Node.js est un environnement d'exécution JavaScript côté serveur qui permet de développer des applications web et des services backend.

1. Événementiel et non bloquant : Node.js est construit sur le moteur JavaScript V8 de Chrome, qui utilise un modèle événementiel et non bloquant. Cela signifie que Node.js peut gérer un grand nombre de connexions simultanées sans bloquer le thread principal, offrant ainsi une scalabilité élevée et une meilleure réactivité de l'application. Dans un système de chat, où de nombreuses connexions en temps réel doivent être gérées, cette capacité est essentielle.
2. JavaScript côté serveur : En utilisant Node.js comme plateforme côté serveur, vous pouvez utiliser JavaScript à la fois côté client et côté serveur. Cela permet de partager du code entre le front-end et le back-end, ce qui facilite la réutilisation du code et améliore l'efficacité du développement. En outre, si vous avez déjà de l'expérience en JavaScript pour le développement front-end, vous pouvez appliquer vos compétences existantes pour le back-end, réduisant ainsi la courbe d'apprentissage.
3. Écosystème et modules : Node.js dispose d'un écosystème dynamique et d'une vaste bibliothèque de modules, disponibles via le gestionnaire de paquets npm. Ces modules prêts à l'emploi facilitent la mise en place rapide de fonctionnalités supplémentaires dans votre système de chat. Par exemple, vous pouvez utiliser des modules tels que Socket.io pour la communication en temps réel, Express pour la gestion des routes et des requêtes HTTP, ou encore MongoDB pour la persistance des données.

-
- 4. Performances : Node.js est connu pour ses performances élevées, grâce à son modèle de traitement asynchrone et à la gestion efficace des E/S. Cela permet d'optimiser le temps de réponse de votre système de chat, assurant une expérience fluide et réactive pour les utilisateurs.
 - 5. Communauté active : Node.js bénéficie d'une communauté active et engagée, ce qui signifie que vous pouvez trouver facilement du support, des tutoriels, des exemples de code et des solutions aux problèmes que vous pourriez rencontrer lors du développement. La documentation de Node.js est également bien entretenue, ce qui facilite l'apprentissage et la compréhension de la technologie.

En résumé, le choix de Node.js en tant que technologie pour le back-end offre une scalabilité élevée, une réactivité accrue, un partage de code avec le front-end, un écosystème riche en modules et des performances optimisées. Ces facteurs combinés font de Node.js un choix solide pour le développement d'applications en temps réel telles qu'un système de chat.

2. Choix des framework



1. React Native avec Expo (Frontend) :

React Native est un framework de développement d'applications mobiles multiplateformes. Il permet de créer des applications mobiles natives pour iOS et Android en utilisant JavaScript et React, une bibliothèque JavaScript pour la création d'interfaces utilisateur. React Native permet aux développeurs de créer une seule base de code et de la partager entre les plateformes, ce qui permet de réduire le temps et les efforts nécessaires pour développer et maintenir des applications pour différentes plateformes.

Expo est un ensemble d'outils et de services qui facilite le développement d'applications mobiles avec React Native. Il fournit des fonctionnalités supplémentaires et des composants prêts à

l'emploi, ainsi qu'un ensemble d'outils de développement, tels qu'un environnement de développement local, un système de build et un ensemble de services cloud pour faciliter le déploiement et la distribution des applications. Expo simplifie le processus de développement d'applications React Native en fournissant une expérience de développement plus fluide et en réduisant la complexité technique.

En utilisant React Native avec Expo pour le frontend de notre application, nous bénéficions d'un cadre de développement puissant et flexible qui nous permet de créer des applications mobiles multiplateformes efficacement, en tirant parti de la réutilisation du code et en offrant une expérience utilisateur native.

2. Express (Backend) :

Express est un framework web pour Node.js qui simplifie le développement d'applications web et d'API. Il fournit des fonctionnalités et des outils pour la création de routes, la gestion des requêtes et des réponses, le traitement des middlewares et la gestion des sessions, entre autres. Express est très populaire en raison de sa simplicité, de sa flexibilité et de sa légèreté, ce qui en fait un choix courant pour la création de services backend.

En utilisant Node.js avec Express pour le backend de notre application, nous pouvons créer des API robustes, gérer les requêtes et les réponses, interagir avec une base de données, gérer l'authentification et l'autorisation des utilisateurs, et fournir des fonctionnalités serveur nécessaires à notre application de chat.

3. En résumé

L'utilisation de ces frameworks nous permet de créer une architecture complète pour notre application, en reliant le frontend (React Native avec Expo) et le backend (Express JS) pour offrir une expérience utilisateur fluide et des fonctionnalités de communication en temps réel entre les utilisateurs.

Ces frameworks bien établis offrent une combinaison puissante pour le développement de votre application de chat, vous permettant de créer une interface utilisateur réactive et riche en fonctionnalités, tout en gérant les aspects du backend pour la communication et le stockage des données.

3. Outils utilisés

Dans le cadre de ce projet , j'ai utilisé différents outils:

- Visual Studio Code comme éditeur de code (IDE);
- Postman pour tester les requêtes API;
- GitHub pour le versionning de notre code;
- Milestone (jalons) sur GitHub pour organiser notre travail;
- NPM pour installer les paquets;
- Figma pour la création de nos maquettes;
- Draw.io pour la conception de notre base de données;
- Expo pour émuler mon application mobile sur mon téléphone;

Organisation du projet

Ce projet a été réalisé durant notre année de formation, la totalité du mois de janvier à été dédiée à la conception de notre application. Donc l'organisation de notre travail à été essentielle. Nous avons commencé le projet par un listing des tâches principales et essentielles . Pour ce faire , nous avons mis en place les procédés ci-dessous.

Ici seront definit les feutures par ordre de proriete de lapplication mobile &web

MUST V 1

- 1,Inscription**
- 2, Une connexion**
- 3, Un salon de chat général**
- 4, Une page profile**
- 5. Un annuaire de tous les utilisateurs**
- 6.page qui detaillle les informations d 'un user**
- 7,chatbot chatgpt**

SHOULD V2

- 1.Chat privé avec un utilisateur**
- 2,Barre de recherche pour les utilisateurs**
- 3,Envoyer des photos/videos**
- 4,Authentification double facteur**
- 5, Un utilisateur peut modifier son message**

COULD V3

- 1.connexion via GoogleLogin , Facebook**
- 2, Envoyer un message vocal**

WISH V4

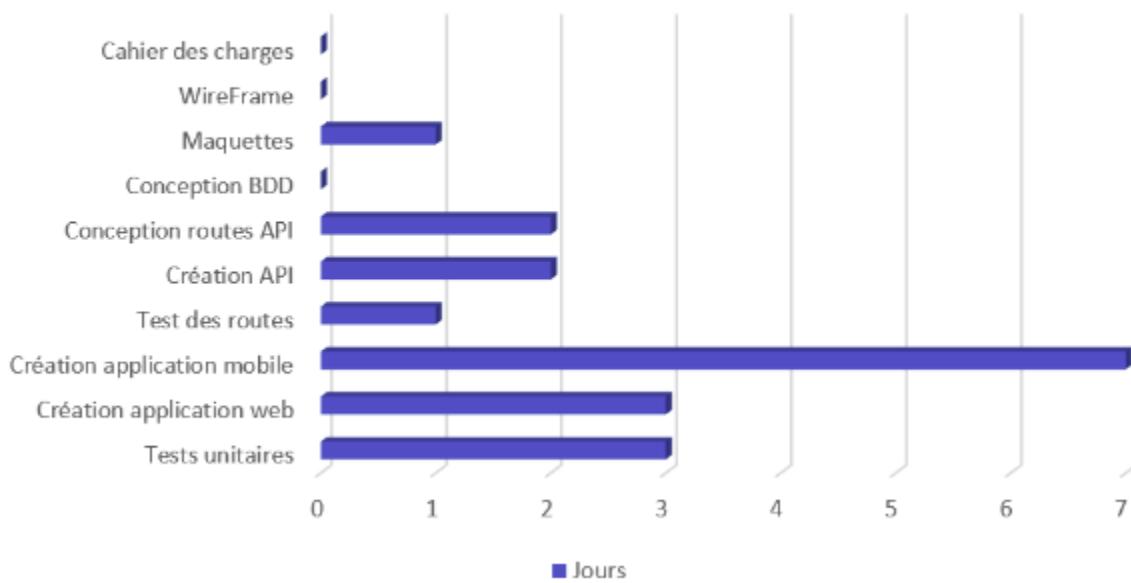
- 1,Chatcoin monnaier pour l application**

1. Diagrammes de Gantt

Le projet de développement d'une application de chat suit un plan structuré en utilisant des diagrammes de Gantt.



Diagramme de Gantt



- Il commence par une **première semaine** de conception où l'équipe évalue le périmètre, établit un cahier des charges et choisit la stack technique. En parallèle, des membres spécifiques de l'équipe travaillent sur la conception du wireframe et la modélisation de la base de données.

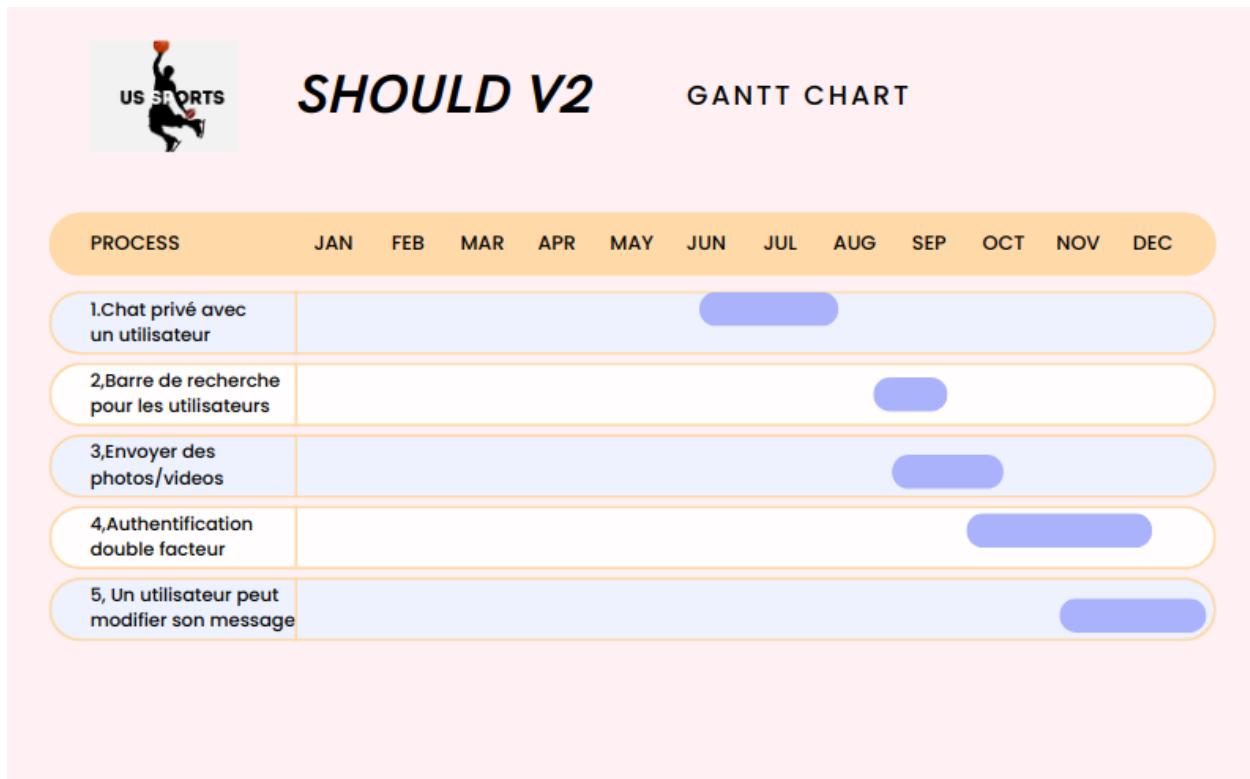


MUST V 1

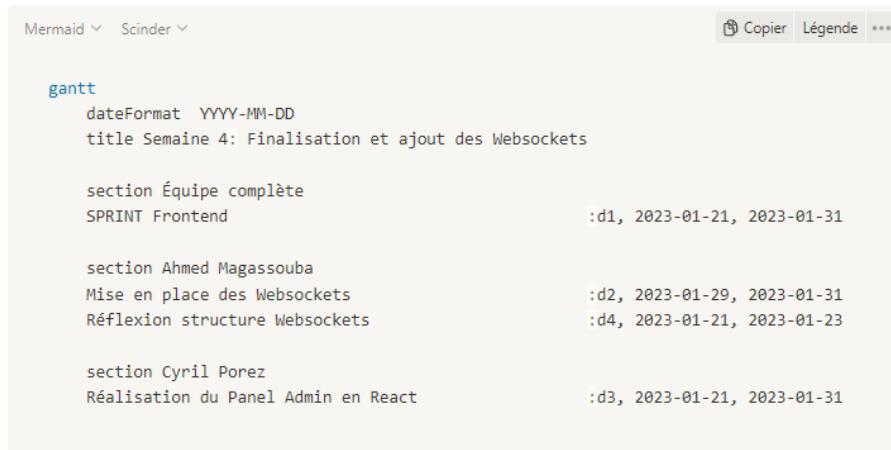
GANTT CHART



- La **deuxième semaine** se concentre sur la mise en place de la base de données, le développement du backend et la réflexion sur l'architecture du frontend. Des tests approfondis de l'API sont effectués, et une réunion de synchronisation est organisée.



- La troisième semaine est dédiée à l'évolution du frontend, avec un sprint spécifique.



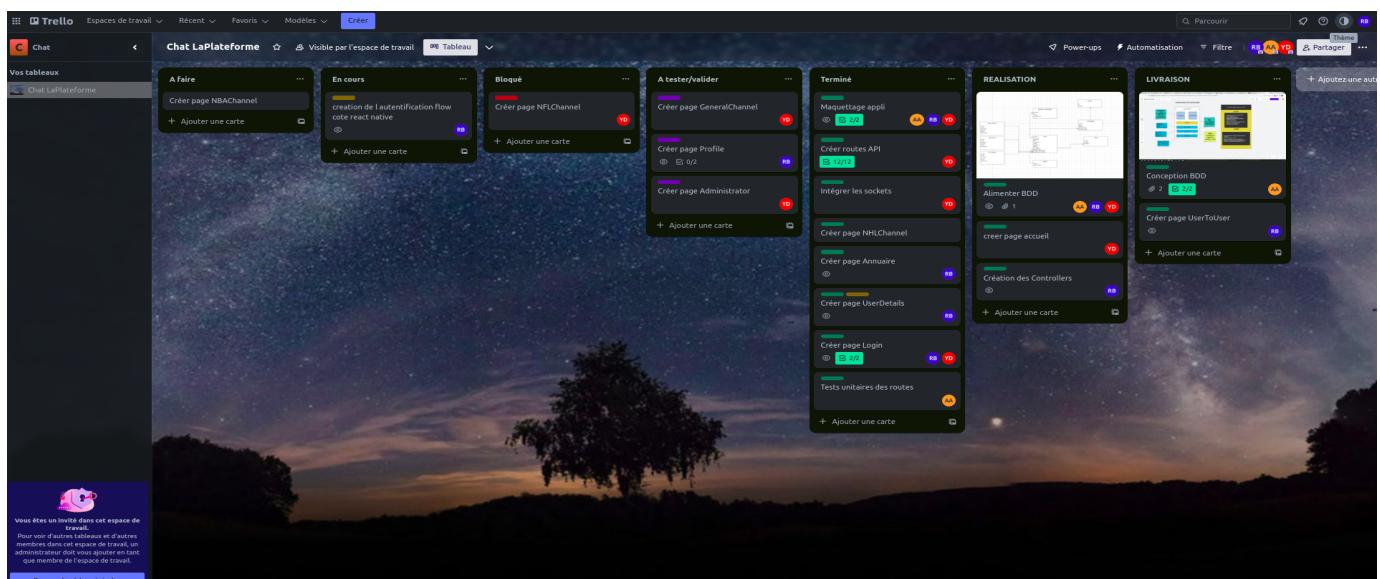
Dans l'ensemble, le projet suit une approche agile, où chaque membre de l'équipe a des tâches spécifiques à accomplir, conformément à la planification établie.

2. Rôles et responsabilités

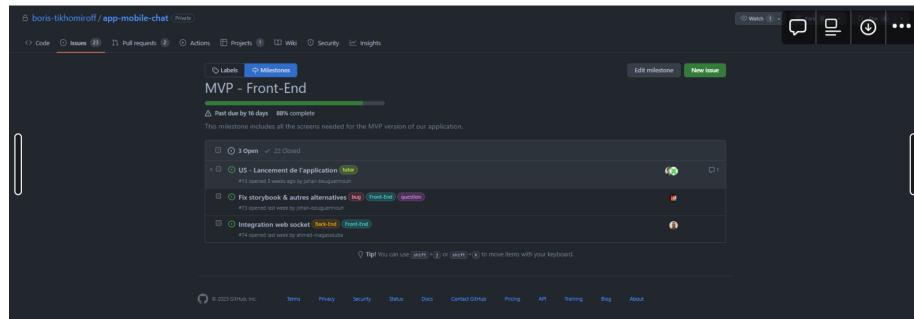
- @ridha boughediri & @Aurelien Adjami seront en charge de la conception et de la réalisation de la maquette de l'application WireFrame et figma
- @YONATAHN GARMON ET Ridha Boughediri seront en charge de la modélisation de la base de données et de sa mise à jour en cas de changements
- Ridha Boughediri , Yonathan DARMON , Aurélien Adjami seront en charge du développement de l'application mobile avec React Native
-

3. Trello

On a listé les tâches et les fonctionnalités :



En utilisant Github, les milestones sont créés sur les repository, et peuvent être associés à des issues pour les regrouper.



Utilisation des Labels

Les labels sur GitHub sont des étiquettes utilisées pour classer les tâches ou les demandes de fonctionnalités dans un projet. Ils permettent de catégoriser les tâches en fonction de leur statut, de leur priorité ou de leur type. Les utilisateurs peuvent créer des labels personnalisés en fonction des besoins de leur projet et les appliquer aux tâches ou aux demandes de fonctionnalités pour les organiser de manière efficace. Les labels permettent de filtrer les tâches et de les regrouper selon des critères spécifiques facilitant ainsi la gestion de projet.

Les différents type de label utilisé dans notre méthodes :

The screenshot shows a GitHub pull request interface for a pull request from `ridha-boughediri` to `admin`. The pull request has 15 commits and 17 files changed. The main commit message is "stat & bug". The pull request is still in progress. On the right side, there are sections for Reviewers (yonathan-darmon, aurelien-adjimi), Suggestions, Assignees (No one—assign yourself), Labels (None yet), Projects (None yet), and Milestone (No milestone). The bottom part of the screenshot shows the commit history with detailed commit messages like "test admin", "correction bug expo", "Merge branch 'master' into admin", etc.

Evaluation des priorités

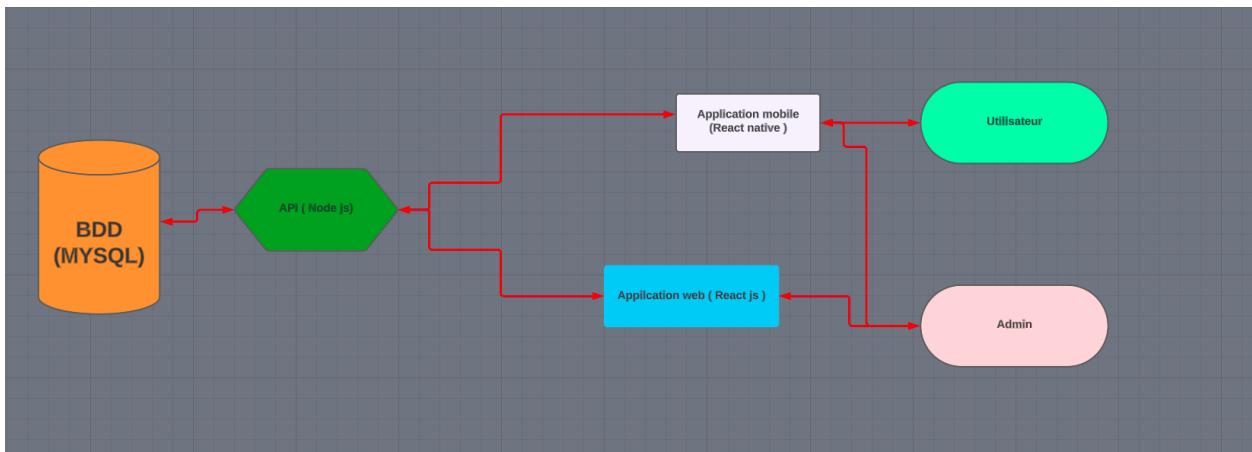
The screenshot shows a GitHub pull request interface for a pull request from `ridha-boughediri` to `admin`. The pull request has 3 commits: "dashboard fini", "delete user par admin", and "test statiq". The pull request is ready to merge. A modal window is open, showing merge requirements: "Require approval from specific reviewers before merging" (Branch protection rules ensure specific people approve pull requests before they're merged), "All checks have passed" (1 successful check), and "This branch has no conflicts with the base branch" (Merging can be performed automatically). Below the modal, there is a "Merge pull request" button. To the right, there are sections for Notifications (Unsubscribe, Customize, You're receiving notifications because you're watching this repository), and 3 participants. There is also a "Lock conversation" button.

Estimation de la vitesse

A la récupération d'une issue nous évaluons la vitesse de la tâche.

Architecture logicielle

Les utilisateurs qui ont un statut **USER** auront uniquement accès à l'application mobile. Les utilisateurs avec un statut **ADMIN** auront accès à la fois à l'application mobile et au site web. L'application mobile et le site web utilisent la même API. L'API communique avec ma base de données MySQL.

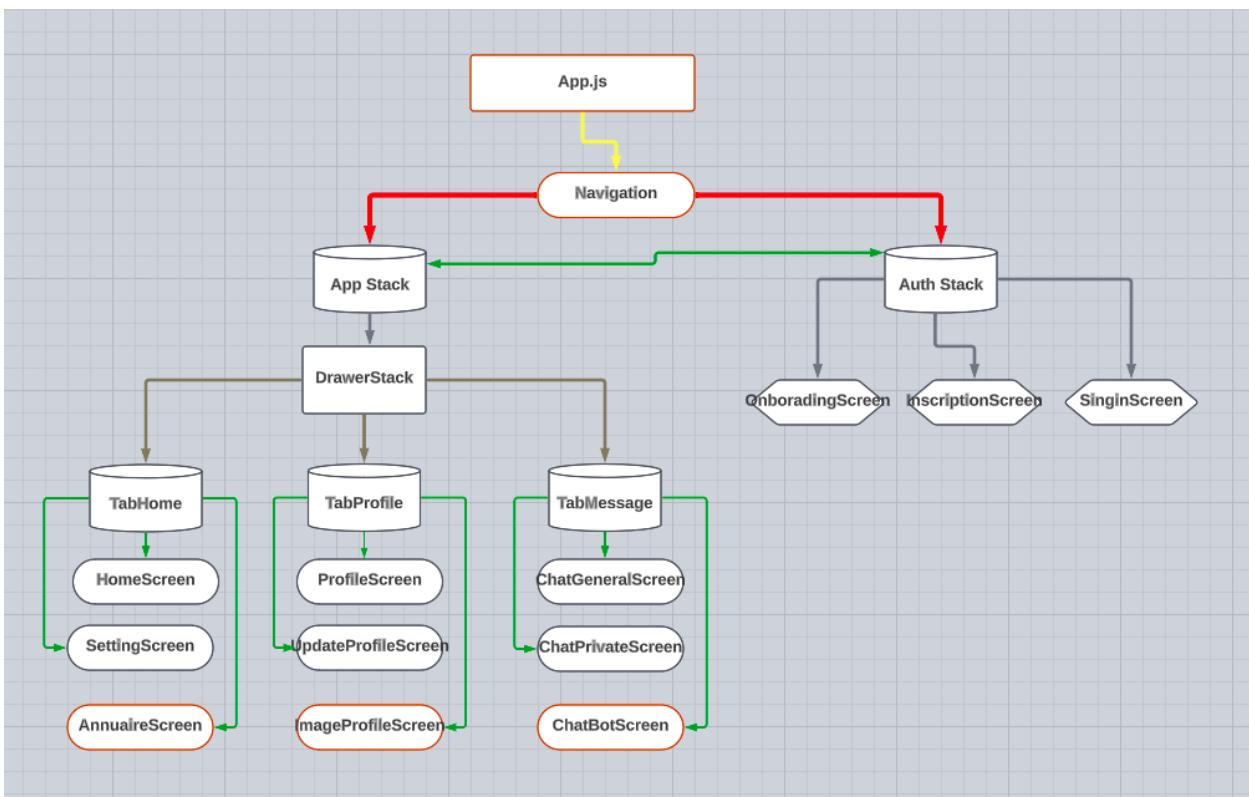


CONCEPTION FRONT-END

Nous avons choisi Figma pour créer la maquette et la charte graphique en raison de sa capacité à faciliter la collaboration en temps réel, de sa facilité d'utilisation et de son accessibilité multiplateforme. Grâce à ses fonctionnalités avancées de conception responsive, de

bibliothèques de composants et de styles réutilisables, ainsi que de prototypage interactif, Figma permet de créer des interfaces cohérentes, de simuler l'expérience utilisateur et d'obtenir des commentaires précieux avant le développement. En résumé, Figma offre une solution complète pour concevoir et collaborer efficacement tout au long du processus de création d'une application.

Arborescence du projet



Charte graphique

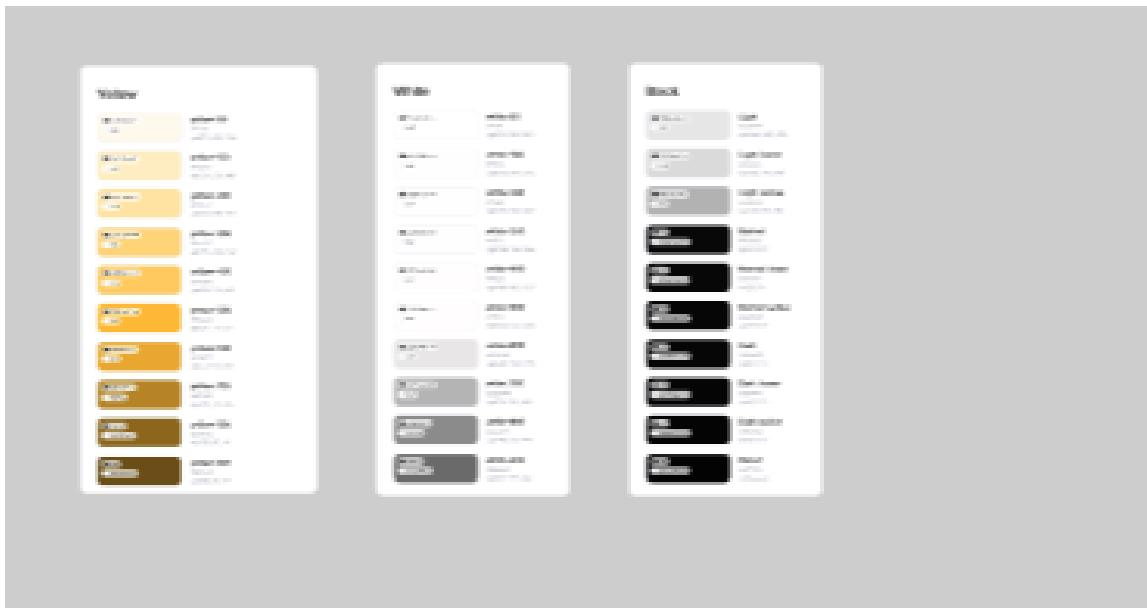
Notre charte graphique définit les principes visuels et les éléments graphiques qui donneront vie à notre application. Elle guide l'utilisation de couleurs, de typographies et de motifs, créant ainsi une identité visuelle cohérente et attrayante. Cette charte graphique reflète notre vision et nos valeurs, et servira de référence essentielle pour assurer une expérience utilisateur harmonieuse et mémorable.

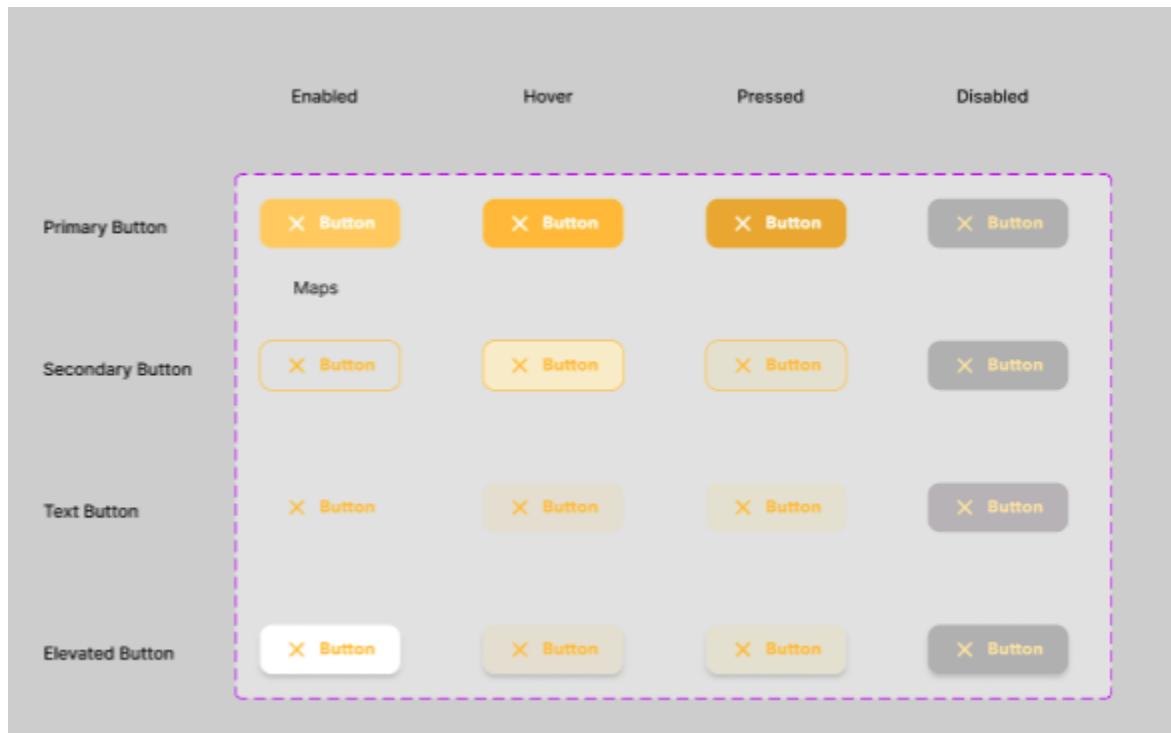
Chat_us



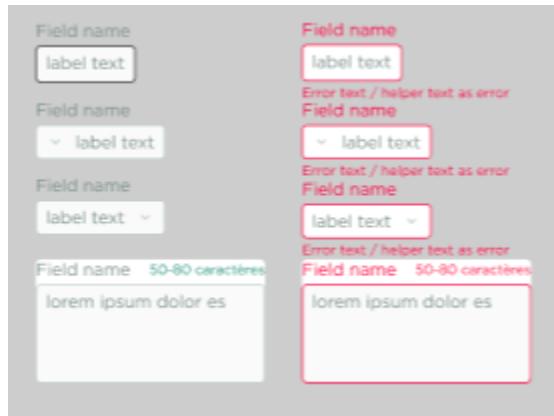
Design System

Notre design système est un ensemble cohérent de composants, de règles et de directives qui définit l'apparence et l'interaction de notre application. Il permet de créer une expérience utilisateur harmonieuse et conviviale en favorisant la réutilisation des éléments de conception et en garantissant une cohérence visuelle à travers l'ensemble de l'application. Le design système facilite la collaboration entre les membres de l'équipe, accélère le processus de





conception et renforce l'identité visuelle de notre marque



Typographies

Metropolis

Texte, titre, titre section, composant, label, contenu des pages statiques



Texte mobile

Ceci est un titre H1 - 24px

Ceci est un titre H 2- 20px

Ceci est un titre H 3 - 18px

Ceci est un titre H 4- 16px

Ceci est un titre H 5- 14px

Texte 16 px : Donec sed odio dui. Cras justo odio, dapibus ac facilisis in. Egestas eget quam. Maecenas faucibus mollis interdum maecenas faucibus. Cras mattis consectetur purus sit amet.

Texte 14 px : Donec sed odio dui. Cres justo odio, dapibus ac facilisis in. Egestas eget quam. Maecenas faucibus mollis interdum mæcenas faucibus. Cras mattis consectetur purus sit amet.

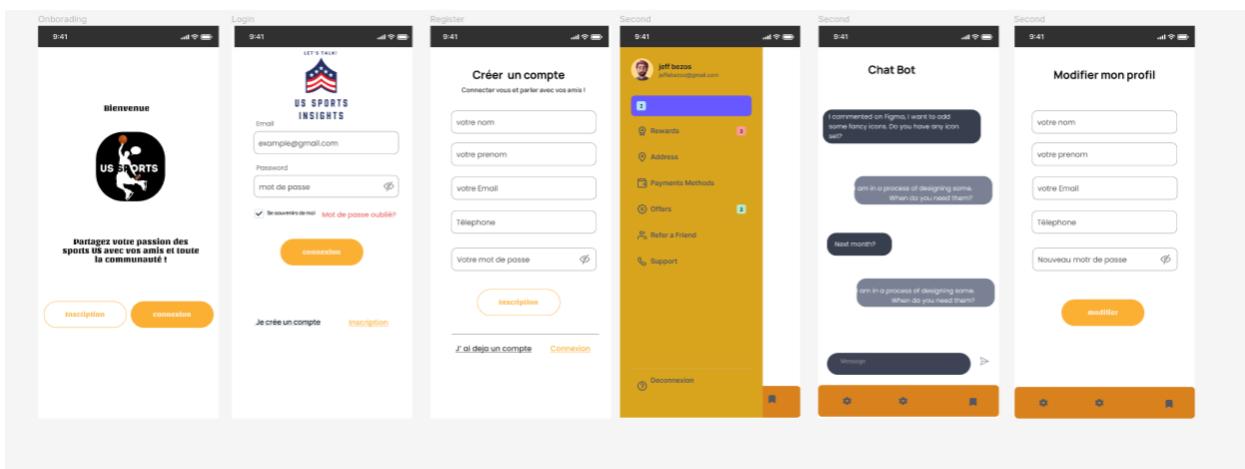
* Veuillez vous référer à l'annexe de ce dossier pour la totalité du design system.



Maquettage

Les maquettes ont été créées à l'aide de Figma .

* Veuillez vous référer à l'annexe de ce dossier pour la totalité du maquettage.



CONCEPTION BACKEND DE L'APPLICATION

La base de données

Mise en place de la base de données:

Pour ce projet , il est indispensable d'avoir une base de données, afin de garder de façon pérenne les données et la possibilité d' avoir un suivi. Mes recherches sur le framework express Js m'ont fait découvrir que les bases de données les plus utilisées avec node Js sont les bases de données NoSql. N'ayant aucune contrainte concernant le choix de celle-ci , j'ai choisi d'utiliser une base de données relationnelle(SQL) .

Durant mon apprentissage , j'ai beaucoup travaillé sur les bases de données MySQL , et j'aimerai pousser plus loin mes connaissances et je pense que ce projet peut m'aider.

Conception de la base de données:

Pour concevoir ma base de données j'ai utilisé la méthode **Merise**. Dans un premier temps , j'ai fait un recueil de données .

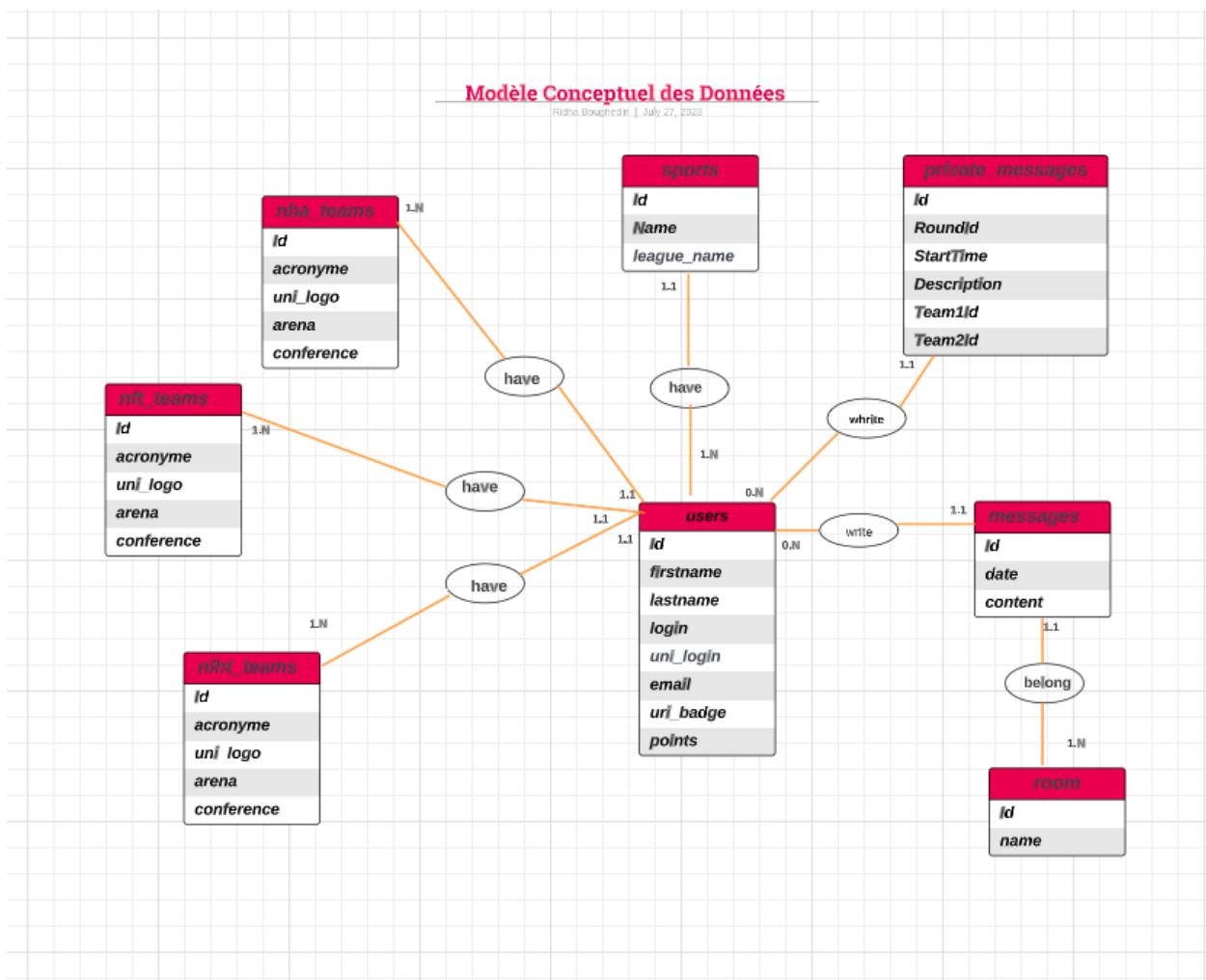
Voici un aperçu des données pertinentes qui m'ont aidé à construire une représentation claire des besoins:

- Pour les **utilisateurs** : j'ai besoin de stocker leurs noms , prénoms, mot de passe,numéro de téléphone, rôle.
- Pour les **channels** : j'ai besoin de stocker le nom du canal.
- Pour les **conversations** : j'ai besoin de stocker les messages,la date d'envoie du message et les participants.
- Pour les **participants** : j'ai besoin de stocker l'utilisateur , son rôle sur le channel et channel.

J'ai construis ma base de données , en procédant en trois grandes étapes:

Modèle Conceptuel de donnée (MCD)

La première étape a été la création du modèle conceptuel de données (MCD) . qui reflète la structure et les relations entre les différentes entités de notre système. Le MCD constitue un schéma essentiel qui nous permet de visualiser et de comprendre la logique de notre base de données. Il est composé de huit entités et du verbe de liaison, qui joue un rôle crucial dans la gestion des relations entre les utilisateurs, les canaux de conversation et les conversations elles-mêmes.

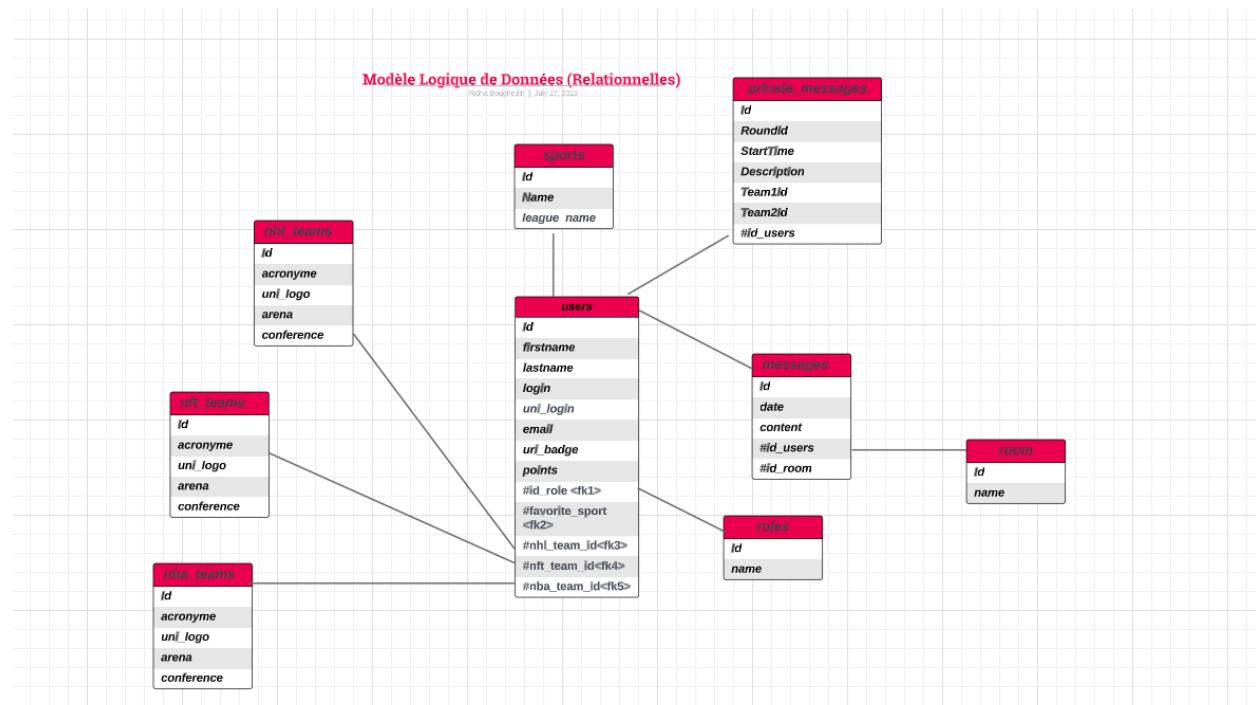


J'ai ensuite continué la création de ma maquette de base de données en créant le modèle logique de données. J'ai créé une clé primaire pour chaque entité qui permet d'identifier sans ambiguïté chaque occurrence de cette entité. Ensuite , j'ai rempli chaque entité avec des attributs en reprenant les informations de mon recueil de données. J'ai procédé à la création des cardinalités de chaque entité.

Exemple :

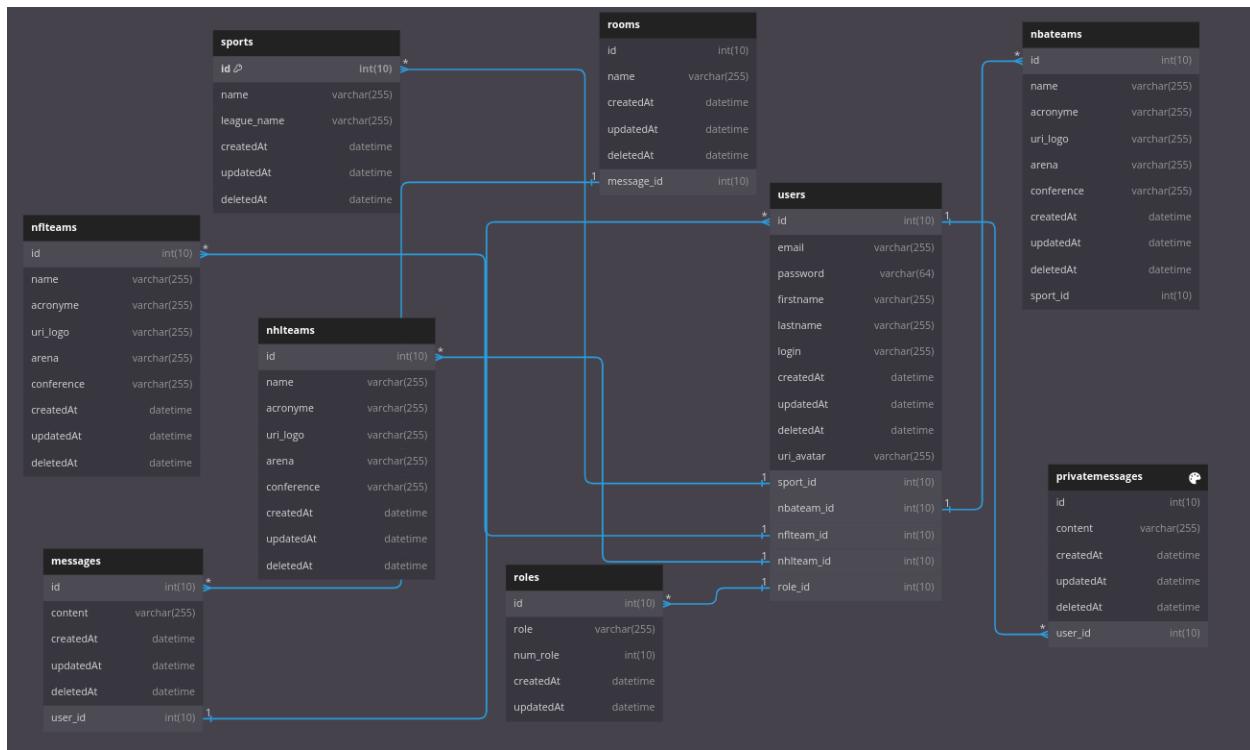
un utilisateur peut avoir au minimum à 0 ou au maximum N conversation et une conversation peut avoir au minimum 1 ou au maximum N utilisateur

Modèle logique des données (MLD)



Modèle physique des données (MPD)

consiste à implanter une base de données dans un SGBDR



DÉVELOPPEMENT DU BACKEND DE L'APPLICATION

Organisation

Mon back end a pour but d'être utilisé à la fois pour mon application web et mon application mobile. J'ai décidé de créer une API afin de ne pas coder deux fois ma logique métier.

Dans le but de rendre mon backend plus efficace , je me suis concentré sur la logique et l'optimisation de mon code.

J'ai donc fait des recherches dans ce sens.

Je divise donc mes programmes en différents modules, ainsi cela augmente la lisibilité du code et devient plus facile à maintenir pour les prochaines versions.

J'évite les répétitions en créant des fonctions et des services.

La logique de mon code est divisé en services et fichiers

Arborescence

J'ai suivi une architecture N-tier. Le principe de cette architecture est la séparation des préoccupations pour éloigner la logique métier des routes de l'API.

Les différentes couches de l'application sont séparées :

- Le routeur,
- La couche applicative : controller
- La couche data (les modèles).

Mon back end est donc composer des dossiers suivants :

-
- **Routes** : regroupant tous les fichiers de mes routes (un fichier par CRUD d'une table de base de données)
 - **Controllers** : Regroupant tous les controller (un par route)
 - **Middleware** : contenant des fichiers de middlewares des routes
 - **model** : contenant les modèles de toutes mes tables (un par table et par fichier)
 - **config** : contenant deux fichiers
 - **db.js** pour tous ce qui est configuration et création de la base de données
 - **mock.js** contenant des fixture avec lesquels on chargera notre base de données à sa création dans db.js
 - **test** : destiné aux test unitaire - Newman
 - **uploads** : images upload

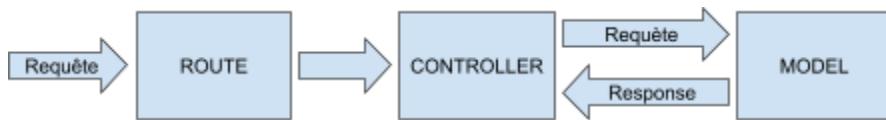
Fonctionnement de l'API

Lorsque le client envoie une requête sur mon API, le routeur analyse l'URL, en fonction de la route et de la méthode un controller est appelé.

Ce contrôleur via la méthode appelé par la route va communiquer avec le modèle afin de récupérer des données. Ensuite , ses données sont analysées par le controller puis une réponse est envoyée au format JSON avec un code statut.

Architecture de l'API

L'API suit une approche MVC (Modèle-Vue-Contrôleur) où chaque composant a un rôle spécifique



Architecture de l'API

Les différents statuts code utilisés dans le projets

Les différents statuts utilisé dans ce projet sont :

- **200** : OK
 - Indique que la requête a réussi
- **201** : CREATED
 - Indique que la requête a réussi et une ressource a été créé
- **204** : NO - CONTENT
 - Indique que la requête a bien été effectué et qu'il n'y aucune réponse à envoyer
- **400** : BAD REQUEST
 - Indique que le serveur ne peux pas comprendre la requête a cause d'une mauvaise syntaxe
- **401** : UNAUTHORIZED
 - Indique que la requête n'a pas été effectuée car il manque des informations d'authentification
- **403** : FORBIDDEN
 - Indique que le serveur a compris la requête mais ne l'autorise pas
- **404** : NOT FOUND
 - Indique que le serveur n'a pas trouvé la ressource demandée
- **405** : METHOD NOT ALLOWED
 - Indique que la requête est connue du serveur mais n'est pas prise en charge pour la ressource cible
- **500** : INTERNAL SERVER ERROR
 - Indique que le serveur a rencontré un problème.

Les différentes méthodes HTTP utilisées

Les différentes méthodes HTTP utilisées dans ce projet :

- **GET** Pour la récupération de données
- **POST** Pour l'enregistrement de données
- **PUT** Pour mettre à jour l'intégralité des informations d'une donnée
- **PATCH** Pour mettre à jour partiellement une donnée
- **DELETE** Pour supprimer une donnée

Middleware

Un middleware est une couche logiciel entre deux couches de logiciels. C'est une simple fonction qui a un rôle particulier.

Dans Express.js, un middleware est une fonction intermédiaire qui s'exécute entre la réception d'une requête HTTP et l'envoi d'une réponse. Il est utilisé pour traiter des tâches spécifiques liées à la requête, telles que l'authentification, la validation des données, la journalisation, etc.

De base, le framework possède quelques middleware mais il est possible d'en créer au besoin du projet.

Lorsqu'une requête est reçue par le serveur Express, elle passe par une série de middlewares avant d'atteindre la fonction de gestionnaire de route finale. Les middlewares peuvent effectuer des actions sur la requête, la réponse ou les deux, et ils ont également la possibilité de passer la main à l'étape suivante ou de terminer le traitement de la requête prématurément.

Un middleware peut être ajouté à une application Express à l'aide de la méthode `use`, fournie par l'objet `app` dans Express. Par exemple, supposons que nous ayons un middleware d'authentification qui vérifie si l'utilisateur est connecté avant de lui permettre d'accéder à certaines routes protégées. Voici à quoi cela pourrait ressembler :

```
const checkIsAdmin = (req, res, next) => {
  const { role } = req.decoded;
  if (role !== "ADMIN") {
    const message = `Vous n'avez pas les droits d'administration.`;
    return res.status(401).json({ message });
  }

  next();
};
```

```
router.put("/", checkToken, checkIsAdmin, updateAll);
```

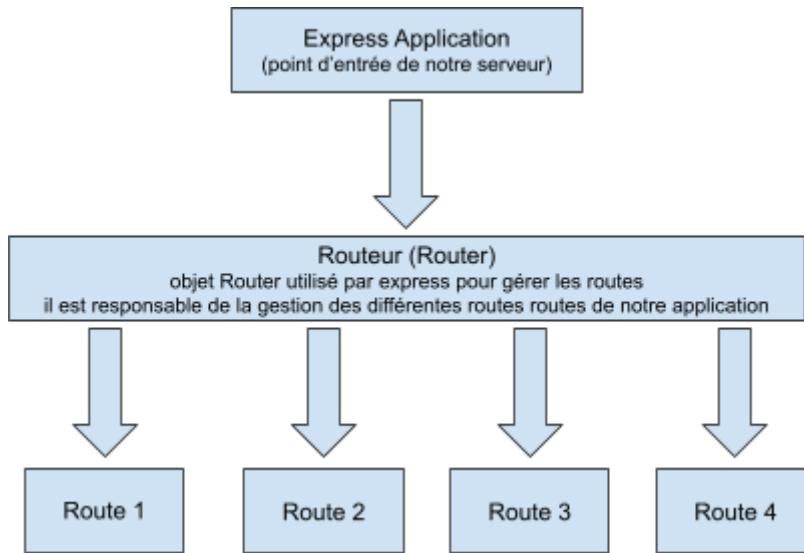
Dans cet exemple, **checkIsAdmin** est un middleware qui vérifie si l'utilisateur a le rôle admin . Si l'utilisateur a le rôle admin, la fonction `next()` est appelée pour passer au middleware ou à la fonction de gestionnaire de route suivante. Sinon, une réponse d'erreur est renvoyée immédiatement avec le statut 401.

Il est important de noter que l'ordre des middlewares est crucial, car ils sont exécutés séquentiellement dans l'ordre où ils ont été ajoutés. Ainsi, l'ordre dans lequel les middlewares sont définis peut avoir un impact sur le comportement de l'application.

En résumé, les middlewares dans Express.js sont des fonctions intermédiaires qui permettent de traiter les requêtes HTTP. Ils offrent une flexibilité et une modularité accrues en permettant d'ajouter des fonctionnalités supplémentaires à l'application, tout en gardant le code organisé et facile à gérer.

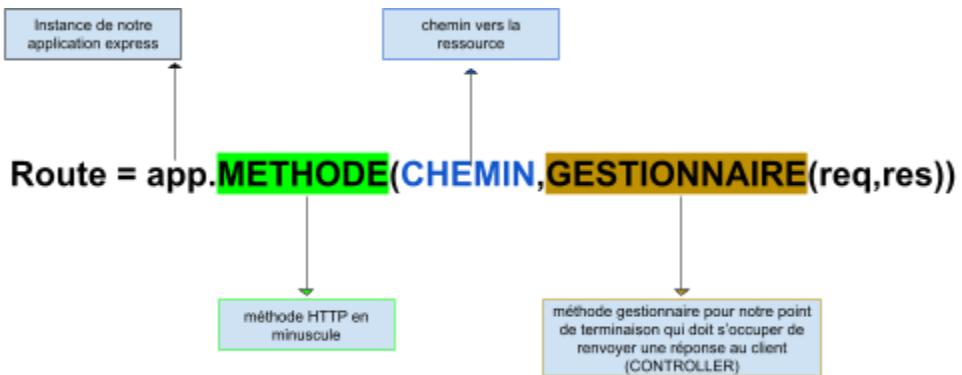
Routage

Afin de mettre en place le routage, j'ai utilisé le routeur de express js.



ExpressJs met à disposition un middleware qui gère facilement le routage du projet.

Pour déclarer une nouvelle route à express , on peut le résumer à cette exemple :



Comme expliqué plus haut mon but a été de rendre mon code facilement modifiable et maintenable j' ai donc diviser cette partie .

La première chose que j'ai faite a été la séparation des routes et le code d'implémentation de celles-ci.

```
app.use("/rooms", routeRooms);
app.use("/sports", routeSports);
app.use("/messages", routeMessages);
app.use("/nbaTeams", routeNba);
app.use("/nflTeams", routeNFL);
app.use("/nhlTeams", routeNhl);
app.use("/privateMessages", routePriveM);
app.use("/roles", routeRole);
app.use("/admin", routeAdmin);
app.use("/auth", routeLogin);
app.use("/users", router);
```

Le middleware use de cet exemple permet de spécifier le routeur qui doit être utilisé en fonction d'une route appelée.

Comme on peut le voir dans cet exemple, si l'URL '/api/messages est appelée , mon routeur routeMessages est utilisé.

App est une instance de express et la fonction use permet d'associer un routeur dans lequel on trouve toutes les routes pour cette ressource.

Dans les routeurs une classe express.Router est instancié. C'est un middleware niveau routeur.

Un middleware niveau routeur fonctionne de la même manière qu'un middleware.

Grâce à la méthode route() de celui-ci le schéma de la route est défini. Ainsi je peux utiliser les méthodes get , post, put, patch et delete afin de pouvoir effectuer différentes actions en fonction de la méthode choisie.

```
import { Router } from "express";  
  
const router = Router();  
  
router.get("/", getAll);
```

Dans les codes ci-dessus, j'ai importé la classe Router de express, grâce à cette classe, je peux faire appeler aux différentes méthodes HTTP.

Comme on peut le voir, la méthode utilisée est le get.

Le premier paramètre de la fonction est l'URL, le deuxième est la fonction de gestionnaire de la route qu'on trouvera dans le controller associé à la route.

Sequelize

Sequelize est une bibliothèque d'Object-Relational Mapping (ORM) pour Node.js qui facilite l'interaction avec des bases de données relationnelles, telles que MySQL, PostgreSQL, SQLite, et d'autres.

1. Object-Relational Mapping (ORM)

Sequelize est une solution d'ORM, ce qui signifie qu'elle permet de mapper les objets de votre application aux tables de la base de données et de faciliter la manipulation des données en utilisant des objets et des méthodes familières, plutôt que d'écrire des requêtes SQL directement. Cela simplifie le développement en vous permettant de travailler avec des entités et des relations plutôt qu'avec des requêtes et des manipulations de données brutes.

2. Prise en charge multi-base de données

Sequelize prend en charge plusieurs systèmes de gestion de bases de données relationnelles, ce qui vous permet de travailler avec différents fournisseurs de bases de données en utilisant une interface unifiée. Vous pouvez facilement passer d'une base de données à une autre sans avoir à réécrire tout votre code.

3. Modélisation des données

Sequelize vous permet de définir des modèles de données en utilisant des classes ou des fonctions constructeurs, qui représentent les tables de la base de données et leurs relations. Ces modèles vous permettent de définir des attributs, des associations, des validations et des méthodes pour travailler avec les données de manière plus orientée objet

4. Opérations CRUD

Sequelize fournit des méthodes et des opérations simples pour effectuer les opérations CRUD (Create, Read, Update, Delete) sur les données de la base de données. Vous pouvez créer de nouvelles instances d'objets, les récupérer depuis la base de données, les mettre à jour et les supprimer facilement en utilisant les méthodes fournies par Sequelize.

5. Migrations de base de données

Sequelize inclut également un mécanisme de migrations de base de données, qui vous permet de gérer les changements de structure de la base de données de manière contrôlée et reproductible. Vous pouvez créer des fichiers de migration pour décrire les modifications à apporter à la base de données, puis les exécuter séquentiellement pour mettre à jour la structure de la base de données.

6. Hooks et Validations

Sequelize offre des fonctionnalités supplémentaires telles que les hooks, qui sont des fonctions déclenchées avant ou après certaines opérations de base de données, et les validations, qui vous permettent de définir des règles de validation pour vos modèles.

Cela vous permet de contrôler et de valider les données avant qu'elles ne soient enregistrées dans la base de données.

Sequelize est largement utilisé dans l'écosystème Node.js pour simplifier la gestion des bases de données relationnelles. Il offre une couche d'abstraction puissante et pratique pour interagir avec les bases de données, en simplifiant le développement et en améliorant la productivité des développeurs.

Connexion à la base de données

Pour commencer j'ai installer le paquet npm `npm install sequelize`

Ensuite j'ai installé le paquet npm mysql2 `npm install mysql2` qui est un pilote spécifique à MySQL et permet à Sequelize de se connecter, de communiquer et de travailler avec une base de données MySQL. Il assure une compatibilité optimale avec les fonctionnalités et les spécificités de MySQL.

Il existe plusieurs paquets permettant une connexion à une base de données, j'ai choisi mysql. C'est la solution la plus simple et rapide à mettre en place pour interagir avec une base de données sql en nodeJs.

j'importe les modules nécessaire

```
import { Sequelize } from "sequelize";
```

je crée une instance sequelize en spécifiant les information de connexion: nom de la base de données,nom d'utilisateur et mot de passe.Ensuite on lui passe le host et le dialect

```
*****  
/* CONNEXION A LA BASE DE DONNEES */  
*****  
  
let sequelize = new Sequelize(  
  process.env.DB_NAME,  
  process.env.DB_USER,  
  "", //process.env.DB_PASSWORD_MACOS,  
  {  
    host: process.env.DB_HOST,  
    dialect: "mysql",  
    dialectOptions: {  
      charset: "utf8mb4",  
    },  
    logging: false,  
  }  
);
```

Avec l'orm sequelize on peut gérer les relation entre les tables de la base de données. Sequelize met à disposition des méthodes comme pour l'exemple ci dessous , j'utilise la méthode `hasMany` et `belongsToMany`

`const db = {};` : Ici, un objet db est créé pour stocker les différentes tables de la base de données.

`db.sequelize = sequelize;` : L'objet sequelize, qui représente la connexion à la base de données, est assigné à la propriété sequelize de l'objet db. Cela permet d'accéder à l'instance de Sequelize dans d'autres parties de l'application via db.sequelize.

```

// 1 relation user et sport

db.Sport.hasMany(db.user, {foreignKey: 'sport_id', onDelete: 'cascade'})
db.user.belongsTo(db.Sport, {foreignKey: 'sport_id'})

// 2 relation user et nba

db.nbaTeams.hasMany(db.user, {foreignKey: 'nbateam_id', onDelete: 'cascade'})
db.user.belongsTo(db.nbaTeams, {foreignKey: 'nbateam_id'})

// 3 relation user et nfl

db.nflTeams.hasMany(db.user, {foreignKey: 'nflteam_id', onDelete: 'cascade'})
db.user.belongsTo(db.nflTeams, {foreignKey: 'nflteam_id'})

// 4 relation user et nhl
db.nhlTeams.hasMany(db.user, {foreignKey: 'nhlteam_id', onDelete: 'cascade'})
db.user.belongsTo(db.nhlTeams, {foreignKey: 'nhlteam_id'})

// 5 relation private-message et user
db.userhasMany(db.privateMessage, {foreignKey: 'user_id', onDelete: 'cascade'})
db.privateMessage.belongsTo(db.user, {foreignKey: 'user_id'})

// 6 relation message et user
db.userhasMany(db.message, {foreignKey: 'user_id', onDelete: 'cascade'})
db.message.belongsTo(db.user, {foreignKey: 'user_id'})

// 7 relation user et role
db.role.hasMany(db.user, {foreignKey: 'role_id', onDelete: 'cascade'})
db.user.belongsTo(db.role, {foreignKey: 'role_id'})

// 8 relation room et message

db.message.hasMany(db.room, {foreignKey: 'message_id', onDelete: 'cascade'})
db.room.belongsTo(db.message, {foreignKey: 'message_id'})

| You, 6 months ago • folder back and frontend ...
| ****
| *** Synchronisation des modèles ***
| sequelize.sync(err => {
|   console.log('Database Sync Error', err)
| })
| db.sequelize.sync({alter: true})

module.exports = db

```

je charge mes fixtures dans le fichier mock.js dans la base de données avec la méthode create de sequelize pour avoir un jeux de données

```
/* ***** */  
/* SYNCHRINISATION DES MODELS */  
/* ***** */  
// force: true  
sequelize  
  .sync({ force: true }) //  
  .then(() => {  
    users.map((user) => {  
      db.User.create(user);  
    });  
    channels.map((channel) => {  
      db.Channel.create(channel);  
    });  
    participants.map((participant) => {  
      db.Participant.create(participant);  
    });  
    conversations.map((conversation) => {  
      db.Conversation.create(conversation);  
    });  
    console.log("Database is connected...");  
  })  
  .catch((err) => {  
    console.log("Error connecting to database...");  
    console.log(err);  
  });  
  
export default db;
```

je vérifie enfin si la connexion à la base de données peut être établie avec les informations fournies .

```
*****  
*** Import des modules nécessaires ***  
const { Sequelize } = require('sequelize')  
  
*****  
*** Connexion à la base de données ***  
let sequelize = new Sequelize(  
    process.env.DB_NAME, process.env.DB_USER, process.env.DB_PASS, {  
        host: process.env.DB_HOST,  
        port: process.env.DB_PORT,  
        dialect: 'mysql',  
        logging: false  
}  
  
/** Mise en place des relations */  
const db = {}  
  
db.sequelize = sequelize  
db.user = require('./models/user')(sequelize)  
db.message = require('./models/messages')(sequelize)  
db.nbaTeams = require('./models/nba_teams')(sequelize)  
db.nflTeams = require('./models/nfl_teams')(sequelize)  
db.nhlTeams = require('./models/nhl_teams')(sequelize)  
db.privateMessage = require('./models/private_message')(sequelize)  
db.room = require('./models/room')(sequelize)  
db.Sport = require('./models/sport')(sequelize)  
db.role = require('./models/role')(sequelize)
```

Controller

Cette couche est l'endroit où se trouve ma logique.

Ils agissent comme les gestionnaires des requêtes, les fonctions gestionnaire de chaque route y est traitée .

Ils sont responsables de l'extraction des données nécessaire des requêtes, de l'appel des fonctions appropriées dans les modèles pour effectuer les opérations crud correspondantes et de renvoyer les réponse appropriées au clients.

Dans l'exemple ci dessous , grâce à sequelize qui fournit des méthodes et des opérations simples pour effectuer les opérations CRUD (Create, Read, Update, Delete) sur les données de la base de données, on peut utiliser l'une de ces méthode le **findOne** qui comme son nom l'indique prend en paramètre un identifiant et return l'utilisateur qui correspond

```
// Sécurisé par authentification
const bcrypt = require("bcrypt");
const jwt = require("jsonwebtoken");

const DB = require("../db.config");
const User = DB.User;

exports.login = async (req, res) => {
  const { email, password } = req.body;
  // Validation des données reçues
  if (!email || !password) {
    return res.status(400).json({ message: "Bad email or password" });
  }
  try {
    // Vérification si l'utilisateur existe
    let user = await User.findOne({ where: { email: email }, raw: true });
    if (user === null) {
      return res
        .status(401)
        .json({ message: "This account does not exists !" });
    }

    // Vérification du mot de passe
    let test = await User.checkPassword(password, user.password);
    if (!test) {
      You, 1 second ago * Uncommitted changes
      return res.status(401).json({ message: "Wrong password" });
    }

    // Génération du token et envoi
    const token = jwt.sign(
      {
        id: user.id,
        nom: user.nom,
        prenom: user.prenom,
        email: user.email,
      },
      process.env.JWT_SECRET,
      { expiresIn: process.env.JWT_DURATION }
    );

    return res.json({ access_token: token });
  } catch (err) {
    if (err.name == "SequelizeDatabaseError") {
      res.status(500).json({ message: "Database Error", error: err });
    }
    res.status(500).json({ message: "Login process failed", error: err });
  }
};
```

Dans cet exemple on a la méthode de récupération d'un utilisateur (user), on y spécifie les données qu'on veut récupérer dans la table users et on traite le renvoie de la réponse au client selon le cas.

Model

Dans l'exemple ci-dessous on voit le model conversation créée grâce à l'ORM Sequelize en utilisant sa méthode define. Je définis les différents champs de ma table , je défini ma clé primaire ,l'auto incrémentation de celle-ci et lui disant que sa valeur ne peut pas être nulle.

Les DataTypes de Sequelize fournissent les différents types de champ.

Sequelize nous permet également de faire de la validation de data , nous verrons cette partie dans la sécurité

```
/** Import des modules nécessaires */
const { DataTypes } = require("sequelize");

module.exports = (sequelize) => {
  const User = sequelize.define(
    "user",
    {
      id: {
        type: DataTypes.INTEGER(10),
        primaryKey: true,
        autoIncrement: true,
      },
      email: {
        type: DataTypes.STRING,
        allowNull: false,
        validate: {
          isEmail: {
            msg: "Please provide a valid email address",
          },
        },
      },
      password: {
        type: DataTypes.STRING(64),
        allowNull: false,
        validate: {
          is: {
            args: /^[0-9a-f]{64}$/i,
            msg: "Password must be a valid SHA-256 hashed value",
          },
        },
      },
      firstname: {
        type: DataTypes.STRING,
        allowNull: false,
      },
      lastname: {
        type: DataTypes.STRING,
        allowNull: false,
      },
      login: {
        type: DataTypes.STRING,
        allowNull: false,
      },
      uri_avatar: {
        type: DataTypes.STRING,
        allowNull: true,
      },
      points: {
        type: DataTypes.INTEGER,
        allowNull: true,
      },
      uri_badge: {
        type: DataTypes.STRING,
        allowNull: true,
      },
    },
    {
      paranoid: true,
    }
  );

  return User;
};
```

Sécurité

Les API sont un moyen d'appeler des informations provenant de la base de données ainsi, il existe de nombreux risques.

les risques

Les différents risque sont :

- **les injections SQL** : Les injections consistent à insérer dans les codes un programme un autre programme malveillant permettant ainsi d'attaquer directement une base de données et y prendre le contrôle. L'attaquant peut alors se servir librement de toutes les informations récoltées.
- **Credential stuffing** : vol du login et password.
- **Attaque DDOS** : envoie de trafic en vue de surcharger le trafic d'une api
- **Man-in-the-middle** : consiste à pousser un utilisateur à se connecter a un service compromis, qui pourra alors s'emparer du jeton ou de la clés de l'utilisateur.

Les solutions utilisées

Pour rendre mon API sécurisé j'ai mis en place plusieurs actions que je vous décrirais dans ce chapitre.

Chiffrement des données sensibles :

Les mots de passe des utilisateurs ne sont pas stockés en dur dans la base de données. Pour hacher des mots de passe , j'utilise bcrypt.

JWT :

Dans le but d'effectuer une authentification sécurisée sur mon projet ainsi que stateless , j'utilise le Jeton JSON Web Token. Le JWT est un jeton qui permet l'échange des informations sur l'utilisateur de manière sécurisée. C'est une méthode de communication entre deux parties.

Ce jeton est composé de trois parties:

- Un header : identifie quel algorithme a été utilisé pour générer la signature

-
- Un payload : est la partie qui contient les informations de l'utilisateur, sous forme de chaîne de caractères hashé en base 64. Pour des mesures de sécurité , je n'insère aucunes données sensibles telles que des mots de passe ou des informations personnellement identifiables.
 - la signature : Elle est créée à partir du header et du payload générés et d'un secret. Une signature invalide implique systématiquement le rejet du token. La signature du jeton a une importance fondamentale , il sert à vérifier que les informations connues sont inchangées.

Lorsqu'un utilisateur essaie de se connecter à son espace , une demande est envoyée au serveur. Si les informations envoyées sont correctes , le serveur renvoie une réponse sous forme de JSON dans lequel s'y trouve le jeton . Celui-ci contient des informations concernant la personne connectée (son id , son mail et son rôle).

Le client enverra ce jeton avec toutes les demandes qui suivront. Ainsi, le serveur n'aura pas à stocker d'informations sur la session.

Gestion des Droits

Pour sécuriser les routes , j'ai développé un système de droits.

Pour ce faire , j'ai créé un middleware.

```

import dotenv from "dotenv";
import jwt from "jsonwebtoken";
import DB from "../config/db.js";

const User = DB.User;

dotenv.config();

const checkToken = (req, res, next) => {
  const event = new Date();

  console.log("AUTH TIME", event.toString());
  console.log(`[${event.toLocaleString()}] ${req.method} ${req.url}`);

  const authorizationHeader = req.headers.authorization;

  if (!authorizationHeader) {
    const message = `Vous n'avez pas fourni de jeton d'authentification. Ajoutez-en un dans l'en-tête de la requête.`;
    return res.status(401).json({ message });
  }

  const token = authorizationHeader.split(" ")[1];
  jwt.verify(token, process.env.JWT_KEY, (err, decodedToken) => {
    if (err) {
      if (
        err.name === "TokenExpiredError" &&
        err.expiredAt < new Date() &&
        err.message === "jwt expired"
      ) {
        const message = `Token expiré...`;
        return res.status(498).json({ status: 498, message, data: err });
      }
      const message = `L'utilisateur n'est pas autorisé à accéder à cette ressource.`;
      return res.status(401).json({ status: 401, message, data: err });
    }

    reqdecoded = decodedToken;
    console.log("reqdecoded", reqdecoded);
    next();
  });
};

```

Dans ce middleware , j'ai importé jsonWebToken qui est utilisé pour générer et vérifier un jwt.

Dans mon .env se trouve mes constantes tel que le JWT_KEY contient le secret key pour signer le JWT. Ce code ne doit jamais être partagé . Il est important que cette clé secrète soit complexe pour que l'application soit la plus sécurisée.

Ensuite , je vérifie si le token est valable et je récupère toutes les données contenues dedans. Je renvoie les informations du user dans la requête vers ma route.

Par exemple , pour la route 'api/users/:userId', qui permet la récupération d'un utilisateur. Cette action est réservée qu'aux personne connecté donc avec un token valide.

Pour pouvoir accéder à cette route il faut donc être en possession d'un token valide

```
router.get("/:userId", checkToken, getOne);
```

Ainsi à chaque appel de cette route le middleware de la route vérifie s'il y a un token valide pour permettre à l'utilisateur de voir les données.

Validation des données

Sequelize offre des fonctionnalités supplémentaires telles que les hooks, qui sont des fonctions déclenchées avant ou après certaines opérations de base de données, et les validations, qui vous permettent de définir des règles de validation pour vos modèles. Cela vous permet de contrôler et de valider les données avant qu'elles ne soient enregistrées dans la base de données.

```
email: {
  type: DataTypes.STRING,          You, il y a 5 mois • feat: app
  allowNull: false,
  unique: {
    msg: "Cet email est déjà pris.",
  },
  validate: {
    notNull: { msg: "L'email ne peut pas être nul." },
    notEmpty: { msg: "L'email ne peut pas être vide." },
    isEmail: { msg: "L'email n'est pas valide." },
  },
},
```

Dans l'exemple au dessus je prend le champ email de la table user où j'ai appliqué les règles de validations comme l'unicité des mail, la vérification du format mail et le fait que le mail ne doit pas être mail.

Helmet:

Helmet est une bibliothèque Node.js qui fournit des en-têtes HTTP de sécurité pour renforcer la sécurité des applications web. Elle aide à atténuer certaines vulnérabilités et attaques courantes en configurant automatiquement divers en-têtes de réponse HTTP liés à la sécurité.

Voici quelques fonctionnalités fournies par Helmet :

- Protection contre les attaques de cross-site scripting (XSS) : Helmet définit l'en-tête HTTP "X-XSS-Protection" pour activer le filtre XSS dans les navigateurs qui le supportent. Cela aide à prévenir les attaques XSS en indiquant au navigateur d'activer sa propre protection XSS.
- Protection contre les attaques de détournement de clic (clickjacking) : Helmet utilise l'en-tête HTTP "X-Frame-Options" pour empêcher les attaques de clickjacking en indiquant au navigateur comment afficher une page dans un frame ou un iframe. Vous pouvez configurer cet en-tête pour empêcher l'affichage de votre application dans des frames indésirables.
- Protection contre les attaques de sniffing MIME (MIME sniffing) : Helmet utilise l'en-tête HTTP "X-Content-Type-Options" pour empêcher le navigateur d'effectuer une détection de type MIME (MIME sniffing) sur les réponses de votre application. Cela aide à prévenir les attaques basées sur des types MIME incorrects ou malveillants.
- Protection contre les attaques de falsification de requêtes inter sites (CSRF) : Helmet fournit un mécanisme pour générer et valider les jetons anti-CSRF, qui peuvent être utilisés pour protéger les formulaires et les requêtes sensibles contre les attaques CSRF.
- Masquage de l'en-tête "Powered-By" : Helmet permet de masquer l'en-tête HTTP "X-Powered-By", qui par défaut expose des informations sur la technologie utilisée par le serveur. Cela peut aider à limiter les informations disponibles aux attaquants potentiels.

Il est important de noter que Helmet ne doit pas être considéré comme une solution de sécurité complète, mais plutôt comme une couche de sécurité supplémentaire pour renforcer la protection de votre application web. Il est recommandé de l'utiliser en combinaison avec d'autres bonnes pratiques de sécurité, telles que la validation des entrées, l'échappement des sorties, l'authentification et l'autorisation appropriées, et la gestion sécurisée des sessions.

Exemple de problématique rencontré:

Pendant le développement de notre application, nous avons rencontré une erreur "EADDRINUSE: address already in use" au moment de lancer votre application Node.js. Cette erreur indique que le port 8888, sur lequel votre application doit écouter les connexions entrantes, est déjà en cours d'utilisation par un autre processus sur votre machine. Cela peut se produire si vous avez déjà une autre instance de votre application en cours d'exécution ou si un autre programme utilise également le port 8888.

```
api-rest@i-0.0.0 dev
nodemon -r dotenv/config server.js

[nodemon] 2.0.22
[nodemon] to restart at any time, enter `rs`
[nodemon] watching path(s): ***!
[nodemon] watching extensions: js,mjs,json
[nodemon] starting `node -r dotenv/config server.js`
[ode_events:49]
    throw err; // Unhandled 'error' event
  ^

Error: listen EADDRINUSE: address already in use :::8888
    at Server.setupListenHandle [as _listen2] (node:net:1485:16)
    at listenInCluster (node:net:1533:12)
    at Server.listen (node:net:1621:7)
    at Object.<anonymous> (/home/bora/Bureau/Nouveau dossier/api-rest/server.js:58:10)
    at Module._compile (node:internal/modules/cjs/loader:1159:14)
    at Module._extensions..js (node:internal/modules/cjs/loader:1213:10)
    at Module.load (node:internal/modules/cjs/loader:1037:32)
    at Module._load (node:internal/modules/cjs/loader:878:12)
    at Function.executeUserEntryPoint [as runMain] (node:internal/modules/run_main:81:12)
    at node:internal/main/run_main_module:23:47
emitted 'error' event on Server instance at:
    at emitErrorNT (node:net:1512:8)
    at process.processTicksAndRejections (node:internal/process/task_queues:82:21) {
  code: 'EADDRINUSE',
  errno: -98,
  syscall: 'listen',
  address: '::',
  port: 8888
}

node.js v18.12.1
[nodemon] app crashed - waiting for file changes before starting...
```

Recherches Anglophones

J'ai pu résoudre cette erreur après des recherches sur des sites spécialisés et j'ai trouvé la solution adaptée à mon problème sur stackoverflow

how to set headers in axios patch request in react js

Asked 6 months ago Modified 6 months ago Viewed 638 times

Report this ad

Can someone tell me what mistake I am making or tell me how to set the header in axios patch request. when I am running the API through postman, everything is working fine but when I connect it with the front end, an error comes up saying that the JWT is not provided on the backend

here is the front end code :

```

import React, { useEffect } from 'react';
import { useParams } from 'react-router';
import axios from 'axios';

const Loader = () => {
  const parmas = useParams();
  const { id } = parmas;
  console.log(id);

  useEffect(() => {
    const fetchBags = async () => {
      try {
        const res = await axios.patch('http://localhost:4001/public/verify', {
          headers: {
            'Content-Type': 'application/json',
            Token: id,
          },
        });

        console.log(res);
        console.log('CBM', { res });
      } catch (error) {
        console.log(error);
      }
    };
  }, []);
}

```

The Overflow Blog

- What it's like to be on the Council (Ep. 5)
- How the Python language for AI is changing

Featured on Meta

- Colors update
- Stack Overflow World Congress
- Temporary policy: ChatGPT is back
- Launching 2 new NLP models
- Conclusions from question-contest experiments...

EXEMPLE ENVOIE DE DONNÉES

Dans cet exemple je vous montre comment j'ai procédé pour mettre en place l'inscription d'un utilisateur

Nous commençons par appeler notre route

```
52     router.post('/register', async (req, res) => [ You, 6 months ago • user ]
```

La route va faire appel à notre Controller, la première est de vérifier si l'utilisateur est déjà inscrit. En utilisant la méthode `findOne` de `sequelize` on vérifie si le mail de l'utilisateur (clé unique) n'existe pas déjà dans notre base de données. Si il est déjà existant, un statut code 409 est renvoyé. Avec un message disant que l'email est déjà utilisé.

```
try {
    // Vérification si l'utilisateur existe déjà
    const user = await db.user.findOne({ where: { email: email }, raw: true })
    if (user !== null) {
        return res.status(409).json({ message: `The user ${email} exists déjà !` })
    }
}
```

Une fois l'étape de vérification passée, on vérifie que le mot de passe entrée respecte notre regex.

```
const passwordRegex = /^(?=.*[a-z])(?=.*[A-Z])(?=.*[0-9])(?=.{8,})/;
```

Si la vérification du mot de passe ne correspond pas aux formats du regex, un statut code 400 est renvoyé. Avec le message disant Le mot de passe doit contenir au moins 8 caractères, une majuscule, une minuscule et chiffre.

Si le mot de passe correspond aux regex imposés. Le mot de passe est hashé à l'aide **BCRYPT**.

```
// Hashage du mot de passe utilisateur
let hash = await bcrypt.hash(password, parseInt(process.env.BCRYPT_SALT_ROUND))
req.body.password = hash
// Création de l'utilisateur
let userc = await db.user.create(req.body)
```

L'étape suivante est celle de la création de l'utilisateur, on appelle la méthode `create` de `sequelize`. Nous avons choisi d'inscrire tous les utilisateurs au `chat général` dès la création du compte. Nous avons fait le choix technique de connecter l'utilisateur à l'inscription. Nous créons le token et refresh token.

```

    Users.create({ ...body }).then(async (userCreated) => {
      const participantChatGeneral = await DB.Participant.create({
        userId: userCreated.id,
        channelId: 1,
      });

      // Cr ation du token
      const token = jwt.sign(
        { id: userCreated.id, role: userCreated.role },
        process.env.JWT_KEY,
        {
          expiresIn: process.env.JWT_EXPIRES_IN,
        }
      );

      //Creation du refresh token
      const refreshToken = jwt.sign(
        { id: userCreated.id, role: userCreated.role },
        process.env.JWT_REFRESH_KEY,
        { expiresIn: process.env.JWT_REFRESH_EXPIRES_IN }
      );
    });
  
```

Ceci met fin   l' tape de cr ation de donn es.

TEST

Nous souhaitons tester si l'API renvoie les donn es correctement . Nous avons test  nos URL d'API. On s'est servi de postman pour r aliser cette t che

The screenshot shows the Postman application interface. A GET request is being made to the URL `http://localhost:8888/messages`. The `Auth` tab is selected, and the `Bearer` option is chosen. Below the token input field, there is a placeholder text `Bearer Token`. The token itself is a very long string of characters: `eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6NiwibGFzdG5hbWUiOiJqZSIslmZpcnN0bmFtZSI6ImpliwiZW1haWwiOiJqZUBnbWFpbC5jb20iLCJyb2xlX2lkijpudWxsLCJpYXQiOjE2ODcxODI0MzQslmV4cCl6MTY4NzE4NjAzNH0.jW2zLS-4zdYL5ltSh12CbHnmU3NJSay2H9gRnHuo9rQ`.

The screenshot shows the Postman application interface. At the top, there is a header bar with tabs for "GET", "Headers 2", "Auth 1", "Body 1" (which is currently selected), "Tests", and "Pre Run". Below the header, there are tabs for "JSON", "XML", "Text", "Form", "Form-encode", "GraphQL", and "Binary". The "Body 1" tab contains a code editor with the following JSON payload:

```
1 {
2   "user_id": "6",
3   "content": "millionnaire"
4 }
```

Exemple d'url: **GET** **http://localhost:8888/messages**

Pour récuperer l'ensemble des messages envoyés sur notre app mobile. Ensuite on clique sur l'onglet "Tests" juste en dessous de la zone de requête.

Dans la section des scripts de test, nous avons écrit du code JavaScript pour effectuer des assertions sur la réponse de l'API.

```

// Vérification du code de statut de la réponse
pm.test("Statut de la réponse", function () {
    pm.response.to.have.status(200);
});

// Vérification du format de la réponse en JSON
pm.test("Format de la réponse", function () {
    pm.response.to.be.json();
});

// Vérification du contenu de la réponse
pm.test("Contenu de la réponse", function () {
    var jsonData = pm.response.json();

    pm.test("Vérification de la réponse", function () {
        pm.expect(jsonData).to.not.be.null;
        pm.expect(jsonData).to.not.be.undefined;
        // Autres assertions...
    });

    pm.test("Vérification des propriétés", function () {
        pm.expect(jsonData.data).to.have.property(@).that.is.not.null.and.not.undefined;
        // Autres assertions...
    });

    var expectedValue = "some value";
    pm.test("Vérification de la valeur", function () {
        pm.expect(jsonData.data[@].firstname).to.equal('Cyril');
        // Autres assertions...
    });

    // Vérification du nombre d'utilisateurs renvoyés
    pm.expect(jsonData.data).to.have.length(12);

    // Vérification des propriétés des utilisateurs
    pm.expect(jsonData.data[0]).to.have.property('id');
    pm.expect(jsonData.data[0]).to.have.property('firstname');
    pm.expect(jsonData.data[0]).to.have.property('lastname');
});

```

- Vérification de la réponse pour voir si elle est nulle ou pas .
- On vérifie si les propriété de la réponse ne sont pas vide ou undefined
- On vérifie si réponse correspond au résultat attendu
- qu'elle contient les détails corrects pour l'utilisateur avec l'ID 1

Test Results (5/6)			
All	Passed	Skipped	Failed
PASS	Statut de la réponse		
PASS	Format de la réponse		
PASS	Vérification de la réponse		
PASS	Vérification des propriétés		
FAIL	Vérification de la valeur Assertion Error: expected 'Ahmed' to equal 'Cyril'		
PASS	Contenu de la réponse		

```
var expectedValue = "some value";
pm.test("Vérification de la valeur", function () {
    pm.expect(jsonData.data[0].firstname).to.equal('Ahmed');
    // Autres assertions...
});
```

Test Results (6/6)			
All	Passed	Skipped	Failed
PASS	Statut de la réponse		
PASS	Format de la réponse		
PASS	Vérification de la réponse		
PASS	Vérification des propriétés		
PASS	Vérification de la valeur		
PASS	Contenu de la réponse		

Test Unitaire avec jest

Nous souhaitons tester si chaque unité fonctionne correctement de manière isolée. Nous effectuons des tests unitaires. Ils garantissent la stabilité et la fiabilité du code en détectant rapidement les erreurs et en facilitant la maintenance du système. Les tests unitaires contribuent ainsi à améliorer la qualité du logiciel et à assurer un développement itératif plus efficace. Pour ce faire nous utilisons le framework jest.

on voulait tester nos endpoint, voici l'exemple suivant effectué une route:`http://localhost:8888/users/me`. Nous testons le statut de la requête.

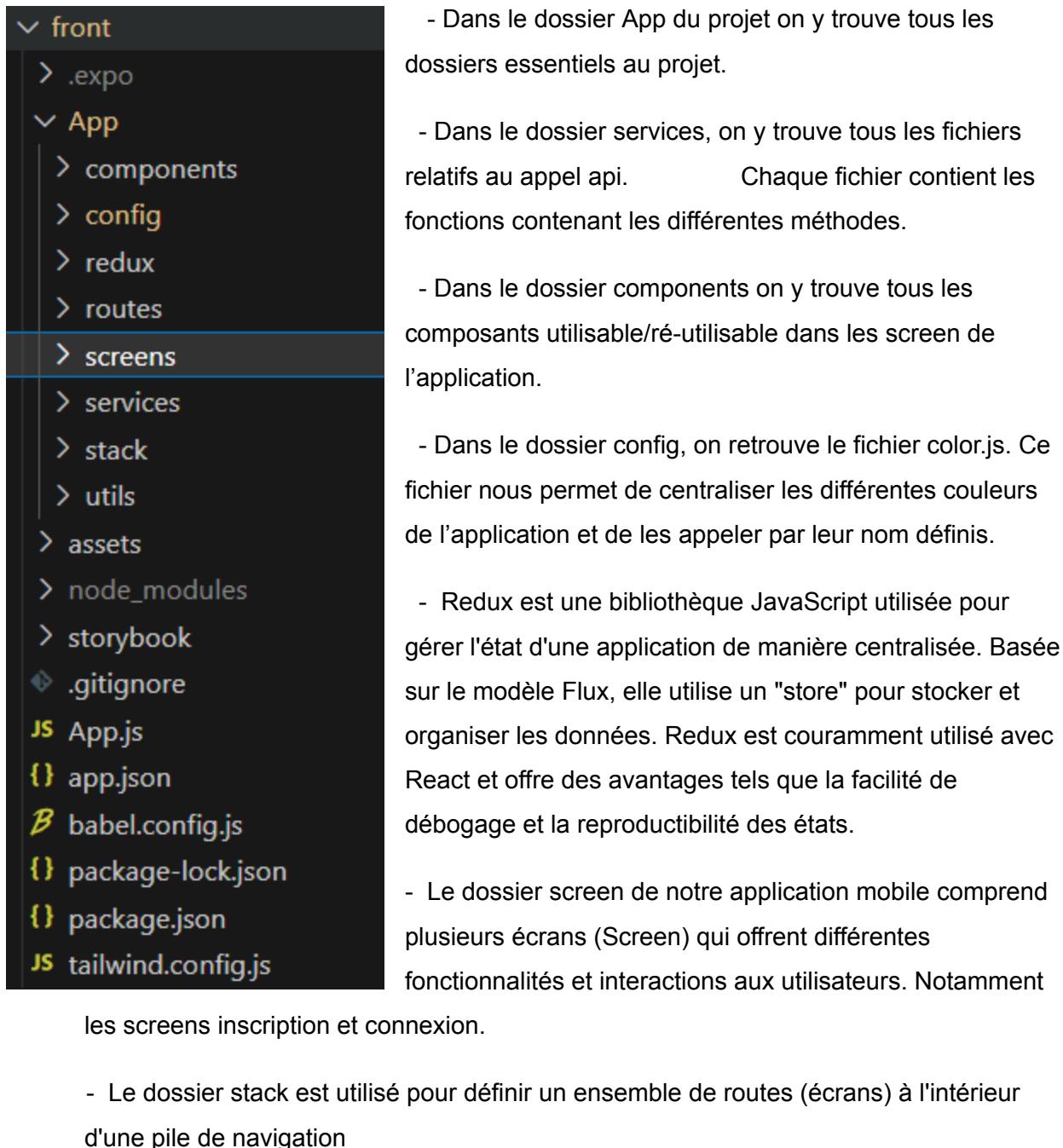
```
1 const express = require('express');      You, 6 months ago • folder back and frontend
2 const request = require('supertest');
3 const user = require('../routes/users');
4
5 describe('get all users', () => {
6   //Test pour la route qui nous retourne une liste de user
7   it('should return a status 200', async () => {
8     const response = await request(express).get('/');
9     expect(response.statusCode).toBe(200);
10    });
11  });
12
13
14 describe('get all users and show only first & last name', () => {
15   //Test pour la route qui nous retourne une liste de user
16   it('should return a status 200', async () => {
17     const response = await request(express).get('/firstlast');
18     expect(response.statusCode).toBe(200);
19    });
20  });
21
22
23 describe('get one user by its ID', () => {
24   //Test pour la route qui nous retourne un user par son ID
25   it('should return a status 200', async () => {
26     const response = await request(express).get('/:id');
27     expect(response.statusCode).toBe(200);
28    });
29  });
30
```

Voici le résultat obtenu, la validation des tests.

```
Test Suites: 2 passed, 2 total
Tests:       4 passed, 4 total
Snapshots:   0 total
Time:        1.656 s, estimated 2 s
Ran all test suites.
PS C:\Users\magas\Desktop\back> 
```

DÉVELOPPEMENT DU FRONT-END

Arborescence



Pages et composants

Un composant React est une unité de code réutilisable utilisée pour créer des interfaces utilisateur interactives dans les applications web. Il encapsule la logique et l'apparence d'une partie spécifique de l'interface. Les composants peuvent être de deux types : fonctionnels ou de classe. Les composants fonctionnels sont des fonctions JavaScript qui prennent des données en entrée et renvoient un rendu visuel. Les composants de classe sont des classes JavaScript qui étendent la classe de base de React et implémentent des méthodes spécifiques. Les composants React peuvent être imbriqués pour former une structure d'arbre modulaire. Ils gèrent leur propre état interne et mettent à jour automatiquement le rendu en cas de changement d'état. Les composants React permettent de créer des interfaces utilisateur réactives sans manipuler directement le DOM. Ils sont réutilisables, faciles à maintenir et favorisent le développement d'interfaces dynamiques et réactives.

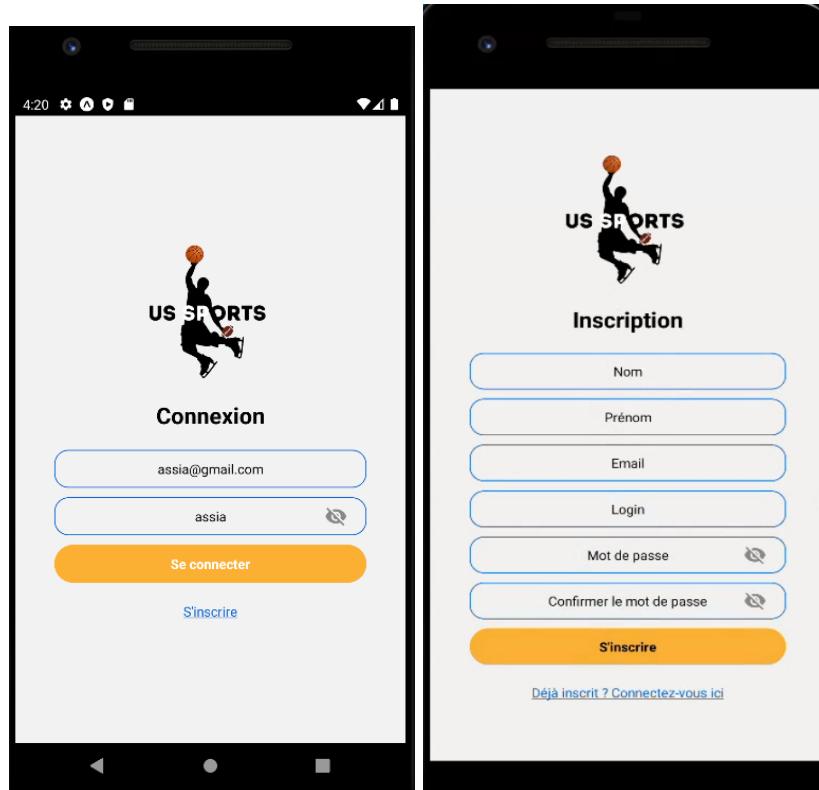
```
import React from "react";
import { TouchableOpacity, Text, View, StyleSheet } from "react-native";

const CtaComponent = (props) => {
  return(
    <View style={styles.container}>
      <TouchableOpacity
        title={props.title}
        color={props.color}
        onPress={props.onPress}
        style={props.style}
      >
        <Text style={props.style2}>{props.title}</Text>
      </TouchableOpacity>
    </View>
  )
}
```

```
const styles = StyleSheet.create({
  container: {
    width: "100%",
  }
})

export { CtaComponent }
```

Voici un exemple d'un composant réutilisable, il permet l'affichage d'un bouton. Dans mon projet , se trouvent de nombreux boutons avec des actions différentes . Mais ils sont composés des mêmes éléments , TouchableOpacity et un Text. Pour éviter les répétitions , j' ai créé un composant nommé Cta Component que j'appelle à chaque fois que j'ai besoin d'un bouton. Je lui passe en props toutes les différences comme par exemple le style(couleur) ou le nom du button.



Comme on peut le remarquer sur l'image ci- dessus , le composant ButtonCustom permet de créer les boutons de mon application.

Sécurités

Dans le l'objectif de rendre mon application la plus sécurisée possible en plus de la sécurité côté serveur, j'ai décidé de rajouter une couche sécurité sur le front de mon application. J'ai décidé d'utiliser les outils mis à disposition dans react native afin de me faire gagner du temps. J'ai décidé d'utiliser la bibliothèque formik combiné avec yup. L'avantage de ces bibliothèques est qu'elles possèdent une gestion des erreurs, après lui avoir paramétré les valeurs attendues de chaque champ. Elle se charge de vérifier les champs au clique sur le bouton de soumission du formulaire. Si un des champs ne correspond pas au pattern indiqué dans les paramètres, une erreur est affichée pour guider l'utilisateur et le bouton d'envois sera complètement désactivé permettant ainsi l'envois de données solides cotées serveurs.

Exemple navigation imbriquées

```
import { createStackNavigator } from "@react-navigation/stack";
import ChatListScreen from "../../screens/Channel/ChatListScreen.js";
import ChannelScreen from "../../screens/Channel/ChannelScreen.js";
import { ChannelOptionScreen } from "../../screens/Channel/channelOptionScreen";

const Stack = createStackNavigator();
.
```

```
const channelsStack = () => {
  return (
    <Stack.Navigator
      screenOptions={{ headerShown: false }}
      initialRouteName={"ChatList"}
    >
      <Stack.Screen name="ChatList" component={ChatListScreen} />
      <Stack.Screen name="Channel" component={ChannelScreen} />
      <Stack.Screen name="ChannelOptions" component={ChannelOptionScreen} />
    </Stack.Navigator>
  );
};

export { channelsStack };
```

Ce composant est chargé de la navigation dans mon application mobile. Pour qu'elle puisse fonctionner , il faut dans un premier temps importer react navigation ainsi que les différentes vues. Dans la constante Stack , je stock l'objet createStackNavigator permettant d'avoir accès à toutes les méthodes de cet objet dont le Stack.Navigator. Le Stack.Navigator permet le stockage des vues de l'application. L'ordre de celle-ci est important. Ma première Stack Screen permet d'avoir accès à mon menu drawer sur toutes les vues de cette Stack Navigator. Les autres Stack.Screen corresponds à un composant

CONCEPTION DE L'ESPACE ADMINISTRATEUR

Conception de la partie administration

Comme énoncé plus tôt, notre projet est composé d'un panel administration. Celui-ci est géré à l'aide d'une application Web. Pour développer cette partie du projet, nous avons choisi d'utiliser un template afin de bénéficier d'une base solide et optimiser notre temps de développement. Grâce à l'utilisation de celui-ci, nous avons pu gagner un temps précieux dans la phase de conception et de développement. De manière à rester cohérent dans le choix de développement de celui-ci, Nous avons choisi un template développé en react.

User Story

Une personne ayant le rôle **ADMIN**, est autorisée à se connecter. Pour ce faire, un formulaire de connexion est à sa disposition. Une vérification des identifiants est effectuée. En se connectant, l'administrateur a accès à un dashboard. Afin de pouvoir naviguer sur l'application Web. L'utilisateur peut utiliser le menu sur le côté avec les liens, qui le redirigera sur les pages souhaitées. En tant que rôle administrateur, l'utilisateur a accès à la liste des utilisateurs.

Choix du Template, langage et framework

Pour la conception du panel administrateur, nous avons fait le choix d'utiliser un template. Le template que nous avons sélectionné est spécifiquement conçu pour répondre aux besoins de notre projet. Il présente une structure claire et modulable, ce qui me permet de nous concentrer davantage sur le contenu et les fonctionnalités essentielles. Pour la partie front-end nous avons choisi un template utilisant **React Js**. Et pour le back-end nous réutilisons, l'api développée en Express Js.

Conception du frontend de l'application Web

Charte graphique

Dans l'optique de l'uniformisation, notre application Web utilise la charte graphique de l'application mobile présentée plus haut

Conception du back-end de l'application Web

Pour la partie back de ce site, j'utilise la même API que l'application.

Déploiement

Nous avons commencé par mettre en ligne notre base de données sur Pulseheberg.

Mon tableau de bord

Vos informations

- ridha boughediri
- 22 marseille, bouches du rhone, 13001 France
- Mettre à jour

Contacts

- Aucun contact trouvé
- Nouveau contact...

Raccourcis

- Commander un nouveau service
- Enregistrer un nom de domaine
- Quitter

Vos services actifs

- Univers Cloud - VPS Cloud (ACTIF mon-server)

Enregistrer un nom de domaine

Trouvez un nom de domaine!

Transfert Enregistrement

Dernières demandes

Aucun tickets récents Ouvrir un ticket

Actualités récentes

Paiement par virement bancaire 02/11/2022

Nous avons choisi Pulseheberg comme système de déploiement pour notre backend. l'image ci-dessous montre la simplicité de création l'app.

mon-server

2 vCPU, 8GB RAM, 30GB SSD (RAID 10) Online

Lausanne, Switzerland 193.168.144.167 Debian 11.3 QLS01

Lost task Rebuilding OS Done

VM details Billing details Graphs Snapshots Rescue ISO Firewall Reinstall Change resources

You can change the name of your VPS below: This will change the name of the VPS displayed on your customer area, as well as the hostname assigned during future reinstallations. The hostname currently in place in your OS will NOT be changed.

New hostname MySuperHostname Change

Some usecase (eg mail server) may need you to configure a specific DNS reverse (PTR record) on your VPS IP addresses. This option allows you to change your VPS IPv4 and IPv6 reverse DNS record:

Your reverse DNS record (PTR) unassigned.as62000.net Change

IPv4 address	193.168.144.167
IPv4 netmask	255.255.255.192 (2^6)
IPv4 gateway	193.168.144.129
IPv4 DNS servers	193.168.144.2, 193.168.144.3
IPv6 address	2a09:6384:1:167:193:168:144:167/48
IPv6 gateway	2a09:6384:1:1::1
IPv6 DNS servers	2a09:6384:4000:1::2, 2a09:6384:4000:1::3

On passe l'étape de config ou on définitie nos variables d'environnements

On pousse notre back sur le repo Git créer et fourni par heroku pour notre projet (voir l'image du dessous),

The screenshot shows a REST API documentation interface for 'Chat sport API' version 1.0.0. The top navigation bar includes links for Home, Documentation, and API. A sidebar on the left lists categories: default, rooms, sports, messages, and users. The main content area displays a list of endpoints under the 'default' category:

- GET /rooms
- POST /rooms
- PUT /rooms/{id}
- DELETE /rooms/{id}
- GET /sports
- POST /sports
- GET /sports/{id}
- GET /messages
- POST /messages
- PUT /messages/{id}

A green 'Authorize' button is located in the top right corner.

The screenshot shows a detailed view of the 'auth/login' POST endpoint. The top section shows the method (POST), URL (/auth/login), and a description: 'Obtenir la liste des salles'. Below this are sections for 'Parameters' (No parameters) and 'Responses' (application/json). The 'Responses' section includes a 'Curl' command, a 'Request URL' (http://ridhaboughediri.fr:8888/auth/login), and a 'Server response' section. The 'Server response' section shows a 400 Bad Request error with the message: "email ou mot de passe incorrect". It also displays the response headers and body.

```
ridha@mon-serveur:/$ tree -d -L 1
.
├── bin    -> usr/bin
├── boot
├── dev
├── etc
├── home
├── lib    -> usr/lib
├── lib32  -> usr/lib32
├── lib64  -> usr/lib64
├── libx32 -> usr/libx32
├── lost+found
├── media
├── mnt
├── opt
├── proc
├── root
├── run
├── sbin  -> usr/sbin
├── srv
├── sys
└── tmp
└── usr
└── var
```

```
ridha@mon-serveur:~/api/routes$ tree
.
├── app-mobile-chat.code-workspace
├── auth.js
├── messages.js
├── nbaTeams.js
├── nflTeams.js
├── nhlTeams.js
├── privateMessage.js
├── roles.js
├── rooms.js
├── sports.js
└── users.js

0 directories, 11 files
ridha@mon-serveur:~/api/routes$
```

/default/post_nhlTeams

```

curl
-X 'GET' \
http://ridhaboughediri.fr:8888/users \
-H 'accept: application/json'

```

Request URL
<http://ridhaboughediri.fr:8888/users>

Server response

Code	Details
200	<p>Response body</p> <pre>{ "data": [{ "id": 3, "email": "ma@gmail.com", "password": "\$2b\$10\$4QStJsqBxc91vNptcYc6.FglaSpoBKzW0gYZqBg.XOHmwkhGpnSW", "firstname": "ma", "lastname": "ma", "url_avatar": null, "points": null, "uri_badge": null, "createdAt": "2023-03-08T11:46:58.000Z", "updatedAt": "2023-03-08T11:46:58.000Z", "selectedAt": null, "sport_id": null, "nbteam_id": null, "nftteam_id": null, "nhlteam_id": null, "role_id": null }, { "id": 4, "email": "wadirida@gmail.com", "password": "wadirida", "firstname": "wadirida", "lastname": "wadirida", "login": "wad", "url_avatar": null, "points": null }] }</pre> <p>Response headers</p> <pre> connection: keep-alive content-length: 16417 content-type: application/json; charset=utf-8 date: Mon, 24 Mar 2023 14:34:59 GMT etag: 11111111111111111111111111111111 keep-alive: timeout=5 x-powered-by: Express </pre>

Responses

Code	Description
------	-------------

L'exemple ci dessus nous montre un exemple de requête avec notre app déployé en utilisant son url

Problème rencontré durant le déploiement

Durant le déploiement de notre application, nous avons rencontré un problème au niveau du port de déploiement. Pulseheberg impose une variable \$PORT à utiliser pour lancer l'app déployer.

```
Linux mon-serveur 5.10.0-8-amd64 #1 SMP Debian 5.10.46-4 (2021-08-03) x86_64
The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
Last login: Mon Jul 24 14:04:02 2023 from 37.26.187.6
ridha@mon-serveur:~$ ls
api chat_us_sql mysql-apt-config_0.8.20-1_all.deb mysql_pubkey.asc mysql_pubkey.asc.gpg server.js server1.js server3.js servertest.js
ridha@mon-serveur:~$ cd api/
ridha@mon-serveur:~/api$ ls
JWT controllers index.html nano.save package-lock.json public server.js server.js.save.1 server.js.save.3 tests
bdd db.config.js models node_modules package.json routes server.js.save server.js.save.2 server.js.save.4 tmp
ridha@mon-serveur:~/api$ 
```

The screenshot shows a Stack Overflow question page. The question title is "run compiled binary -> not found?". It was asked today, modified today, and viewed 18 times. The question text states: "I am trying to run a compiled binary on my debian server. The binary was build on another machine with the exact same OS and architecture (Debian 12, x64). When I try to run it, I get a "not found" back." Below the question is a code block showing terminal output:

```
/usr/src/app/bin # ls -la
total 25412
drwxr-xr-x 1 root root 4096 Jul 24 10:28 .
drwxr-xr-x 1 root root 4096 Jul 24 10:29 ..
-rwxr-xr-x 1 root root 26003232 Jul 24 10:27 extrude_skeleton
/usr/src/app/bin # chmod +x extrude_skeleton
/usr/src/app/bin # ./extrude_skeleton
/bin/sh: ./extrude_skeleton: not found
/usr/src/app/bin #
```

The question also includes a note: "what does this "not found" refer to?" and a response: "Thanks a lot!" with tags "bash" and "debian". A timestamp indicates the question was asked 3 hours ago by user Hannes F.

On the right side of the page, there's a sidebar with "The Overflow Blog" featuring posts like "Improving time to first byte: Q&A with Dana Lawson of Netlify" and "What's it like to be on the Python Steering Council (Ep. 592)". There's also a "Featured on Meta" section with links to "Colors update: A more detailed look", "Stack Overflow at WeAreDevelopers World Congress in Berlin", "Temporary policy: Generative AI (e.g., ChatGPT) is banned", "Launching 2 new collectives: PHP and NLP", and "Conclusions from title-drafting and question-content assistance experiments...".

Below the sidebar, there's a "Hot Network Questions" section listing various interesting questions from other sites in the network.

CONCLUSION

Pour conclure , ce projet m'a permis de découvrir de nouveaux frameworks et aussi de pouvoir monter en compétence sur javascript. Ce projet qui s'est déroulé sur l'année de formation m'a appris à organiser mon travail.

En effet ,

durant cette année je n'ai pas seulement développé ce projet , j'ai aussi collaboré sur le développement d'une autre application mobile avec des personnes de ma promotion. Une application sur la gestion des humeurs développée avec symfony pour l'API et react Native pour l'application. Celle-ci m'a permis de développer des compétences transverses comme le travail d'équipe. De plus, le projet présenté dans ce dossier m'a permis de voir que je pouvais m'adapter aux différentes situations.