



Réalisation de l'application web & mobile

-

US Sport Chat

1. Introduction.....	4
1.2. Résumé du projet « Chat us ».....	5
1.3. Summary of the "Chat us" project in English.....	6
1.4. Compétences couvertes par le projet.....	7
2. Cahier des charges.....	8
2.1. Définition des besoins.....	8
2.1.1. Les objectifs de l'application.....	8
2.1.2. Les cibles.....	9
2.1.3. Périmètre du projet.....	9
2.1.4. Fonctionnalités.....	9
2.1.5. Contraintes techniques.....	10
3. Conception.....	11
3.1. User Story.....	11
3.2. UX.....	12
3.2.1. User Experience.....	12
3.2.2. Définition du parcours utilisateur.....	15
3.3. UI.....	16
3.3.1. Côté Pote & charte graphique.....	16
3.3.2. Maquette & Prototype.....	18
3.4. Modélisation de la base de données.....	20
3.4.1. Méthode MERISE.....	20
3.4.2. Modèle conceptuel de données (MCD).....	20
3.4.3. Modèle Logique de Données (MLD).....	22
3.4.4. Modèle Physique de Données (MPD).....	24
3.5. Collaboration à la gestion de projet.....	25
3.5.1. Objectifs.....	25
3.5.2. Organisation.....	26
3.5.3. Versionning & Workflow.....	27
3.5.1. Gestion des dépendances.....	29
4. Environnement Technique.....	30

4.1. Technologies & Outils.....	30
4.2. Spécificités techniques : Backend.....	31
4.2.1.API REST.....	31
1. Installer Node.js :.....	34
2. Créez un nouveau projet :.....	35
3. Installez Express :.....	35
5. Configurez votre application Express :.....	35
Pour installer Express.js et Sequelize dans notre projet, voici les étapes :	37
4.2.4.Nodejs / API Swagger : Création d'une Table.....	41
Tout d'abord :.....	42
4.2.5.Node js / Sequelize ORM: Ajout des relations.....	44
1. Association Many-to-One (Plusieurs-à-Un).....	45
2. Association One-to-Many (Un-à-Plusieurs).....	45
3. Association Many-to-Many (Plusieurs-à-Plusieurs).45	45
4.3. Spécificités techniques : Frontend.....	46
4.3.1. React Native.....	46
4.3.2.Expo.....	47
4.3.3.React Navigation.....	49
4.3.4. Axios : Interaction avec la base de données.....	52
5. Présentation du jeu d'essai.....	54
5.1. Parcours utilisateur pour la connexion d'un user.....	54
Pour le style, ce n'est toujours pas du CSS :	59
6.Extrait d'une recherche à partir de site anglophone.....	60

1. Introduction

1.1 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.

2.1 Présentation personnelle

Bonjour je m'appelle Ridha BOUGHEDIRI j'ai 35 ans .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.

1.2. Résumé du projet « Chat us »

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.

1.3. Summary of the "Chat us" project in English.

Our mobile application is designed to meet the needs of passionate sports fans who want to stay informed about match scores and the latest news on players from each team.

With our application, you can easily track scores in real-time and get live updates on news from your favorite team.

You can also personalize your app experience by selecting the teams and leagues you want to follow, and receive instant notifications about match results, scores, news, and events. Our app also features advanced capabilities such as detailed graphs for each match and player, statistics, and analysis to help you make informed decisions about upcoming matches and player performances.

In summary, our mobile application is the perfect companion for any sports fan looking to stay informed and connected with their favorite teams.

Our application is designed for american sports enthusiasts who want to share their passion with other people and post their personal bets

1.4. Compétences couvertes par le projet

Voici les compétences du référentiel couvertes par le développement réalisé sur ce projet.

- Activité type 1 : Concevoir et développer des composants d'interface utilisateur en intégrant les recommandations de sécurité
 - Maquetter une application
 - Développer des composants d'accès aux données
- Activité type 2 : Concevoir et développer la persistance des données en intégrant les recommandations de sécurité
 - Concevoir une base de données
 - Mettre en place une base de données
 - Développer des composants dans le langage d'une base de données
- Activité type 3 : Concevoir et développer une application multicouche répartie en intégrant les recommandations de sécurité
 - 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

2. Cahier des charges

Avant de passer à la réalisation de la maquette ou une quelconque étape de développement, nous avons travaillé avec mon groupe qui était composé de 4 personnes, entre autres, Robin Papazian, Robin Arbona, Pierre Malardier et moi-même, à l'élaboration d'un cahier des charges afin de cadrer le projet, lister les fonctionnalités et se donner des objectifs.

2.1. Définition des besoins

2.1.1. Les objectifs de l'application

- Les objectifs qualitatifs

L'objectif est de mettre en place une appli permettant à des groupes d'amis ou utilisateur solo sans groupe d'amis d'avoir la possibilité de lancer des paris (défis peut-être par la suite).

Prévue pour être utilisée uniquement sur une app, un travail en amont sera

réalisé pour que l'app soit compatible sur tous les types d'OS que ce soit Android ou IOS.

- Les objectifs quantitatifs

L'application devra être capable de stocker les informations de chaque utilisateur de manière sécurisée.

L'identité visuelle doit être accrochante et dans l'air du temps avec un Flat Design.

La navigation doit être naturelle et intuitive et une attention particulière doit être portée sur l'UI/UX de manière générale pour pouvoir accrocher l'attention d'un public large.

2.1.2. Les cibles

L'application sera ouverte à toute personne souhaitant s'amuser avec ses amis dans le cadre de paris de toute sorte ce qui donne une liberté et permet de toucher un large spectre d'utilisateurs peu importe la provenance sociale ou le type de loisir visée.

La cible comprend toute personne qui cherche à identifier des changements d'humeur, des périodes de bien-être ou d'autres phases émotionnelles passagère... sur l'application il y aura que des utilisateurs inscrits, ce qui comprend les utilisateurs qui pourront participer à des paris, voir l'historique de tous les paris auxquelles ils ont participés, agrémenter ou modifier leurs profil (image, nom, prénom, date de naissance, num de tel, mail), avoir accès au défis de la semaine, créer des groupe de paris, invité des amis dans le ou les différent groupe de paris auxquels il participe.

2.1.3. Périmètre du projet

L'app sera disponible uniquement en français. Par ailleurs, il devra être adapté à tous les OS (Androïd, IOS) pour permettre à tous les utilisateurs d'avoir une expérience d'utilisation optimale. Celle-ci pourra comprendre l'utilisation de fonctionnalités natives des appareils mobiles, telles que l'appareil photo ou la géolocalisation. Les données personnelles des utilisateurs seront stockées dans une base de données.

2.1.4. Fonctionnalités

- L'app comprend une page d'accueil à partir de laquelle nous pouvons nous nous connecter/s'inscrire.

- Il sera également possible de consulter le fonctionnement de l'app après inscription de l'utilisateur.
-
- L'app comportera un espace utilisateur sécurisé par un mot de passe et hashé dans la base de données (en accord avec la réglementation du RGPD).
- Pour les personnes souhaitant modifier leurs informations, il sera mis à disposition sur l'app une page profil.
- Il y aura la possibilité de modifier, ajouter, supprimer un pari
- Il y aura la possibilité de participer à un pari
- il y aura la possibilité de rechercher un pari en particulier

2.1.5. Constraintes techniques

L'application devant être compatible de manière native avec les systèmes d'exploitation mobile iOS et Android, plusieurs choix techniques ont été fait pour ce choix.

Côté Front, nous avons décidé d'utiliser la librairie Expo React Native. Celle-ci permet de créer des applications mobiles multiplateformes sans avoir à apprendre les langages de chaque système, elle offre un gain de temps et de productivité énorme tout en offrant une certaine ergonomie. La base de données et l'API qui permettent les requêtes HTTP devront être sécurisées.

Les mots de passe de l'application devront être hashé en base de données et tout utilisateur pourra récupérer ou voir supprimer ses données à tout moment.

3. Conception

Afin de mieux appréhender nos étapes de conception et d'intégration, nous avons fait le choix d'utiliser une approche AGILE, celle-ci nous a paru idéale pour optimiser au mieux notre travail.

3.1. User Story

Avant de réaliser les étapes de conception concernant l'UX et l'UI, nous avons fait de la veille et testé plusieurs applications mobiles abordant le même thème comme winamax, picolo, betclic...

Cette démarche nous a permis de nous mettre à la place d'un utilisateur et de définir les fonctionnalités de l'application.

Les « User Stories » sont utilisées pour imaginer les besoins des utilisateurs, elles permettent de voir les caractéristiques, les fonctions et les exigences de l'application à créer.

Elles permettent d'appréhender la nécessité d'une fonctionnalité sur l'application et aide à faire comprendre l'utilité de la fonctionnalité et sa priorité.

Les user stories ont ainsi permis à notre équipe d'établir une arborescence représentant le parcours utilisateur.

3.2. UX

3.2.1. User Experience

Les premières minutes d'utilisation sont cruciales pour assurer l'évolution et l'utilisation d'une application sur la durée. L'expérience de l'application doit être agréable intuitive ainsi que tape à l'œil avec un esprit jeune et frais.

L'UX (User Experience) appliquée au mobile consiste en la conception d'applications à même de faire vivre une expérience ludique, drôle et humaine à l'utilisateur dans notre cas.

L'UX a été ainsi mobilisée au début du projet, lors de l'élaboration de la stratégie pour :

- Faciliter l'utilisation de l'application et son fonctionnement
- Contribuer à l'amusement de l'utilisateur dans la durée en faisant appel à sa créativité lors de la création de son pari
- Permettre un usage sur différents appareils et dans des contextes spécifiques (en soirée, en compétition...)

Pour savoir comment optimiser notre application mobile et apporter une expérience plaisante pour l'utilisateur, il a fallu effectuer des recherches

et porter notre attention sur les points suivants :

- Limiter les fonctionnalités au minimum puisque le but de l'application est l'amusement sans forcément compliquer la tâche de l'utilisateur.
- Simplifier le contenu au maximum, donc proposer une application fluide et rapide est un objectif majeur.
- Guider l'utilisateur par le design Afin d'aider l'utilisateur dans sa prise de décision, le travail autour des couleurs et des formes est important dans le but de hiérarchiser l'information et inciter à l'action est nécessaire. La charte graphique doit être déclinée sur l'ensemble de l'application afin d'être reconnaissable, rassurer l'utilisateur et obtenir un rendu harmonieux.
- Limiter le travail de l'utilisateur de par son format et les situations dans lesquelles il est utilisé, manipuler un smartphone n'est pas toujours une tâche facile.

Il est nécessaire de diminuer le plus possible le nombre d'actions à entreprendre côté utilisateur en limitant les étapes, supprimant un maximum de champs dans les formulaires, utilisant les données déjà fournies par l'utilisateur, affichant les bons claviers au bon moment, utilisant les fonctionnalités natives du téléphone tel que l'appareil photo etc.

- Travailler l'ergonomie, donc ne pas oublier que les utilisateurs ne sont pas toujours dans des conditions optimales de navigation. Chaque élément doit donc être de taille raisonnable et atteignable à

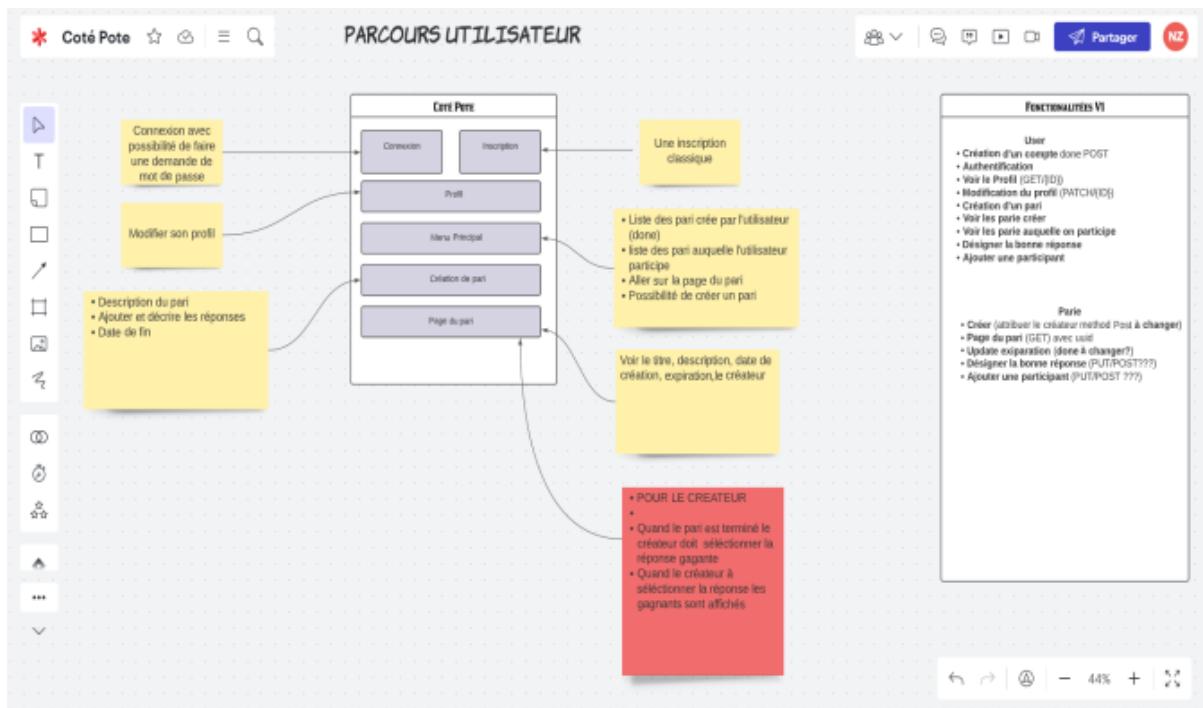
une main.

- Ne pas obliger à une quelconque action, réfléchir à l'usage et à la récurrence de l'envoi des notifications « Push ».
 - Prendre en compte la diversité humaine et concevoir des produits et services numériques ouverts à toutes et tous et qui ne nécessitent pas d'adaptation de la part des utilisateurs.
-
- Des conventions existent pour le mobile et les utilisateurs s'attendent à les retrouver dans l'application comme par exemple une « Tab Bar » pour la navigation, un « Burger Menu » plutôt que menu classique...
-
- Maximiser les chances de pérennisation de l'application en proposant le bon contenu au bon moment, le degré de personnalisation à implémenter est à estimer suivant les fonctionnalités du projet et des ressources.
-
- Les stores d'application possèdent leurs propres algorithmes de positionnement. L'ASO ou « AppStore Optimization » est un travail à part entière, rédiger de bonnes descriptions qui tapent à l'œil, sélectionner la bonne catégorie, ajouter des captures d'écran attrayantes, représentatives des fonctionnalités, obtenir de bons avis avec de bonnes notes.

3.2.2. Définition du parcours utilisateur

Pour définir le parcours utilisateur, nous avons identifié les étapes clefs de la navigation, et traduit à partir de celle-ci les possibilités et actions menant aux différentes pages de l'application.

À partir de cette réflexion, nous avons modélisé un schéma représentant l'ensemble du parcours utilisateur et des fonctionnalités disponibles



3.3. UI

3.3.1. Côté Pote & charte graphique

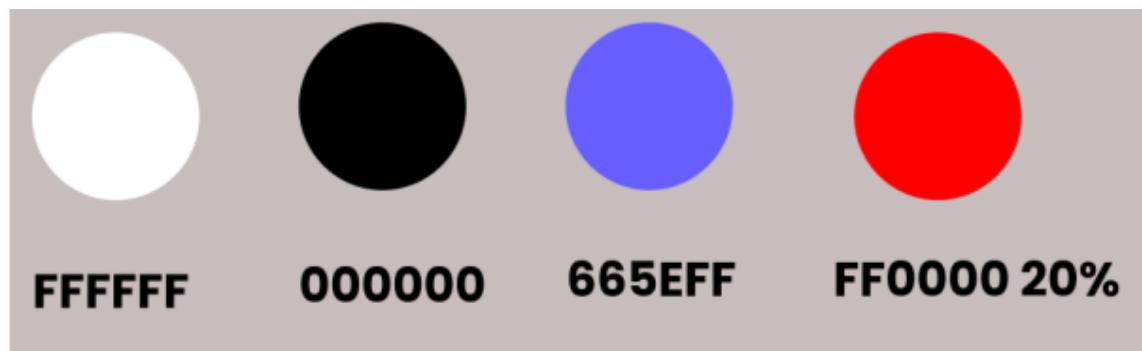
Pour réaliser la maquette, nous avons d'abord constitué une sélection de

designs inspirés d'applications mobiles existantes, entre autres du benchmarking.

Cela nous permet d'avoir plusieurs idées de ce vers quoi l'on pourrait se diriger pour réaliser la charte graphique et le design de l'application.

L'élaboration de la charte graphique, celle-ci reprend :

- Un logo visible sur l'écran d'accueil
- Une liste des polices à importer et utiliser sur l'ensemble de l'application
- Les couleurs principales de l'application
- Des icônes, formes et couleurs de boutons



3.3.2. Maquette & Prototype

Pour la création de la maquette, nous avons fait le choix d'adopter une approche de création selon les principes de l'Atomic Design. Cette méthodologie de conception d'interfaces graphiques permet de

réduire nos composants à de plus petites briques réutilisables, lesquelles peuvent ensuite être assemblées pour constituer des modules de plus grande taille.

Tous ces types de composants ont un but précis, et combiné entre eux, permettent de constituer un ensemble de pages cohérent et facilement maintenable de par la modularité qu'apporte ce type de conception.

C'est un concept que l'on peut également comparer à la programmation orientée objet et qui dans notre cas, est justifié par l'utilisation de la librairie React Native adoptant un design pattern basé sur la création d'application par composant.



Pour reprendre les principes de l'Atomic Design nous avons créé un Kit UI.

C'est un ensemble d'éléments graphiques, basés sur la hiérarchie citée précédemment (atomes, molécules, organismes) et qui seront réutilisés pour créer chaque page.

Logo



The image shows a mobile application interface for "US SPORTS INSIGHTS".

First Screen: A login screen with fields for "Email" (example@gmail.com) and "Mot de passe" (motdepasse). It includes a checkbox for "Se souvenir de moi" and a "Connexion" button. Below the form are links for "je crée un compte" and "Inscription".

Second Screen: A registration screen titled "Créer un compte" with fields for "votre nom", "votre prénom", "votre Email", "Téléphone", and "Votre mot de passe". It includes a "Veuillez valider" button. Below the form are links for "J'ai déjà un compte" and "Connexion".

Third Screen: A profile management screen titled "Modifier mon profil" with fields for "votre nom", "votre prénom", "votre Email", "Téléphone", and "Nouveau mot de passe". It includes a "Modification" button.

Fourth Screen: A navigation menu on the left with options: Profils, Adresses, Payment Methods, Offers, Refer a Friend, Support, and Documentation. The "Offers" section has a green checkmark. The main area shows a "Chat Bot" interface with a message from the bot: "I am currently in development, I want to add some new features. Do you have any ideas?" and a "Send message" button.

3.4. Modélisation de la base de données

3.4.1. Méthode MERISE

MERISE (Méthode d'Étude et de Réalisation Informatique pour les Systèmes d'Entreprise). Née à la fin des années 1970 en France, elle a pour objectif de définir une démarche permettant la modélisation et la conception de S.I. Parmi les ressources informatiques de ces S.I., figurent en particulier les fichiers de données, bases de données et système de gestion de bases de données (S.G.B.D.).

C'est sur ce dernier point que la méthode MERISE nous a été utile, car c'est en se basant sur ses principes de modélisation que nous avons conçu la base de données **chat_us_sport**.

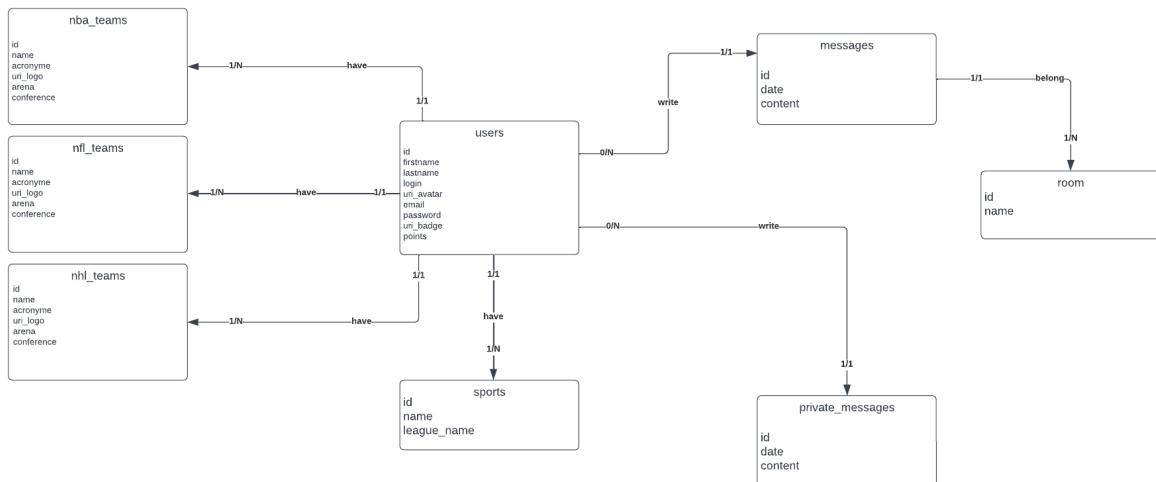
3.4.2. Modèle conceptuel de données (MCD)

c'est un modèle simplifié de la base de données, où les tables sont seulement au stade d'entités (c'est l'équivalent de la table en MCD), entre son second nom de schéma Entité/Association.

D'abord il a fallu imaginer tous les besoins des futurs utilisateurs de **chat_us_sport**. A partir de ces besoins, j'ai été en mesure d'établir les règles de gestion des données à conserver.

Ensuite il faut définir le dictionnaire des données, c'est-à-dire toutes les données élémentaires qui vont être conservées en base de données et définir certaines caractéristiques qui figureront dans le MCD. Parmi ces caractéristiques, on retrouve par exemple la référence d'une donnée et notamment un identifiant unique, sa désignation, son type etc...

Étape finale à sa conception, on a pu à partir des informations précédemment recueillies, créer chaque entité, unique et décrite par un ensemble de propriétés et leur associations permettant de définir les liens et cardinalités entre les entités.[reprendre](#)



3.4.3.Modèle Logique de Données (MLD)

Le modèle logique de données (MLD), est une étape intermédiaire entre le modèle conceptuel de données et le modèle physique de données.

Le passage d'un MCD en MLD s'effectue selon quelques règles de conversion précise :

Une entité du MCD devient une table. Dans un SGBD de type relationnel,

une table est une structure tabulaire dont chaque ligne correspond aux données d'un objet enregistré et où chaque colonne correspond à une propriété de cet objet.

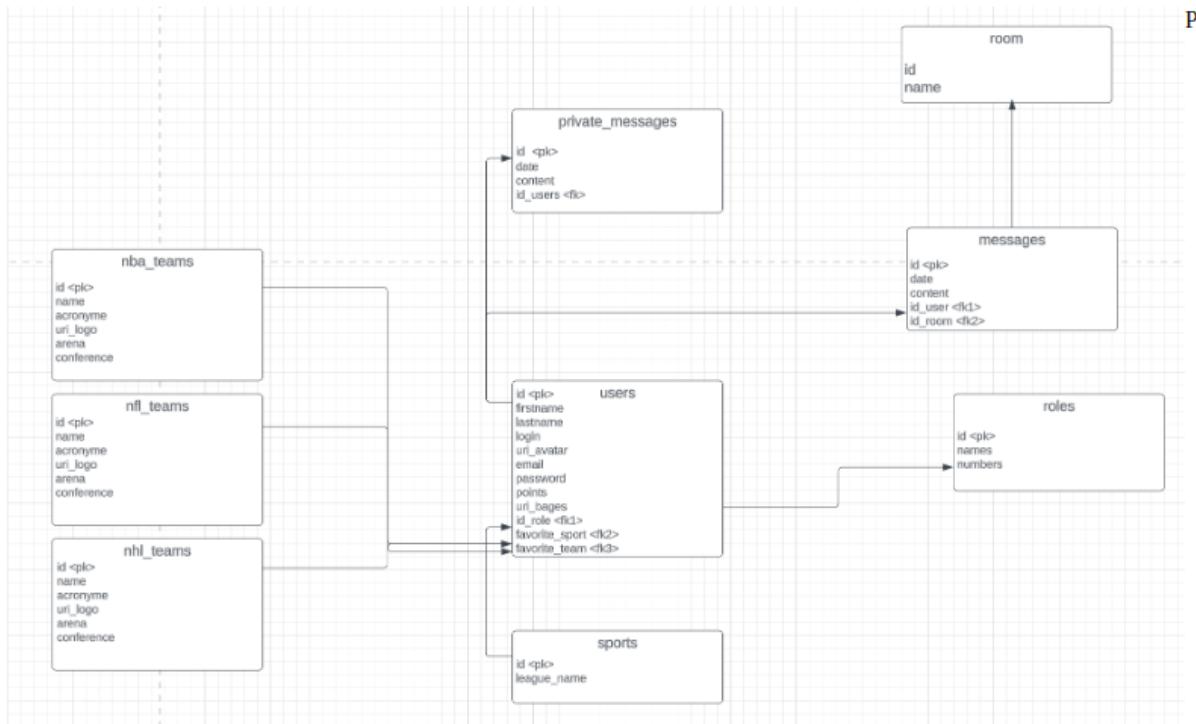
Ces colonnes font notamment référence aux caractéristiques définies dans le dictionnaire de données du MCD.

Les identifiants respectifs de chaque entité deviennent des clefs primaires et toutes les autres propriétés définies dans le MCD deviennent des attributs.

Les clefs primaires permettent d'identifier de façon unique chaque enregistrement dans une table et ne peuvent pas avoir de valeur nulle.

Les cardinalités de type "0:n" / "1:n" sont représentées à travers le référencement de la clef primaire de la table qui la possède, en clef étrangère au sein de la table auquelle elle est liée.

Si deux tables liées possèdent une cardinalité de type "0:n / 1:n", la relation sera traduite par la création d'une table de jonction, dont la clef primaire de chaque table deviendra une clef étrangère au sein de la table de jonction.reprendre

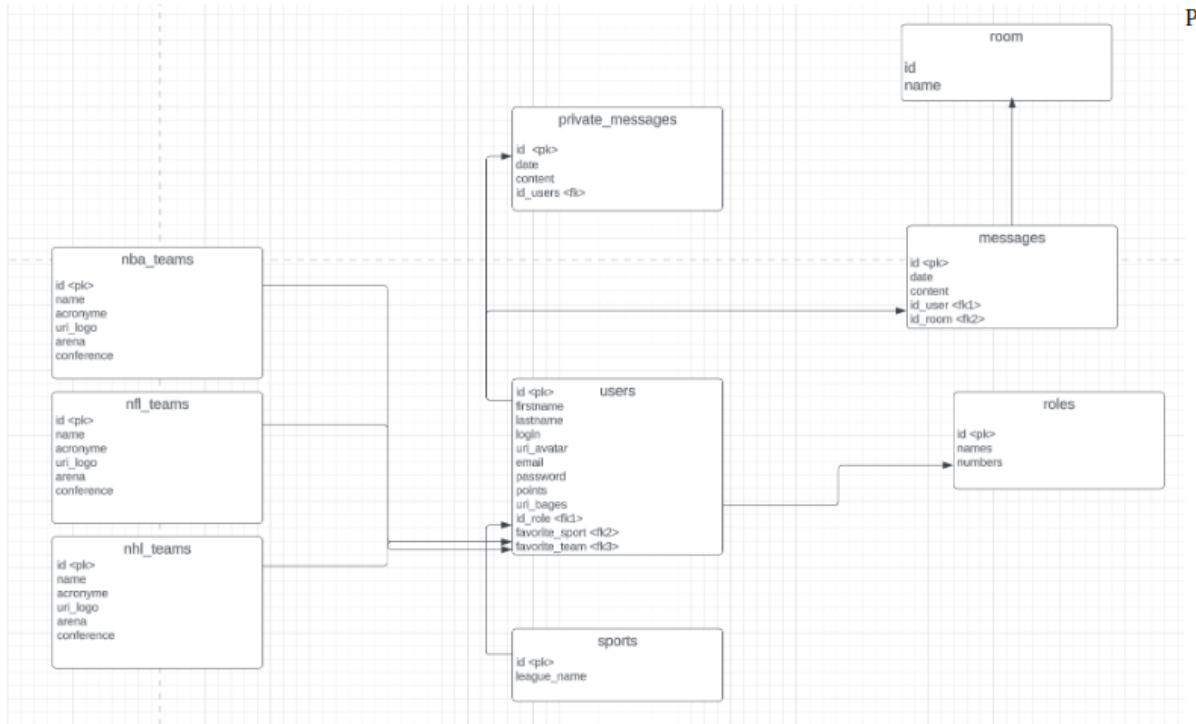


Ici, nous voyons bien que les clés primaires de chaque table associée deviennent des clés étrangères dans chacune des tables qui reçoivent les données.

3.4.4. Modèle Physique de Données (MPD)

Étant presque une formalité, le MPD consiste à l'implémentation des modèles générés précédemment dans le Système de Gestion de Base de Données (SGBD) utilisé.

Dans notre cas, ce sera le SGBD MySQL couplé au moteur de stockage InnoDB qui permet la gestion des contraintes des clefs étrangères en base de données. **reprendre**



3.5. Collaboration à la gestion de projet

3.5.1. Objectifs

Avant de démarrer le développement, il a été nécessaire de définir des objectifs clairs en termes de livrables.

Nous avons décidé de produire une version bêta de l'application.

Cette version présente les fonctionnalités principales afin de tester l'essentiel des fonctionnalités auprès d'un panel d'utilisateurs et de prendre des décisions sur l'évolution du projet en fonction de leurs retours.

Nous avons choisi de présenter une application sur laquelle un utilisateur

peut s'inscrire, se connecter, ajouter un pari, participer à un pari.

La page profil permet de consulter ses informations et les modifier.

Une Tab Bar en bas de l'application permet de naviguer de manière fluide

entre les différentes interfaces.

Pour atteindre ces objectifs dans un temps limité, c'est-à-dire pendant le rythme de l'alternance, nous avons fait des choix en termes d'organisation, de technologies et d'outils.

3.5.2. Organisation

En vue d'optimiser notre gestion de projet et le travail d'équipe, nous avons

utilisé divers outils de travail collaboratif permettant de répartir les tâches de développement.

Concernant la gestion des tâches, nous avons utilisé l'outil de gestion de projet Trello, il offre la possibilité de lister les tâches à accomplir et suivre leurs avancements par les différents membres de l'équipe.

Les utilisateurs ont accès à un tableau sous forme de cartes assignables et

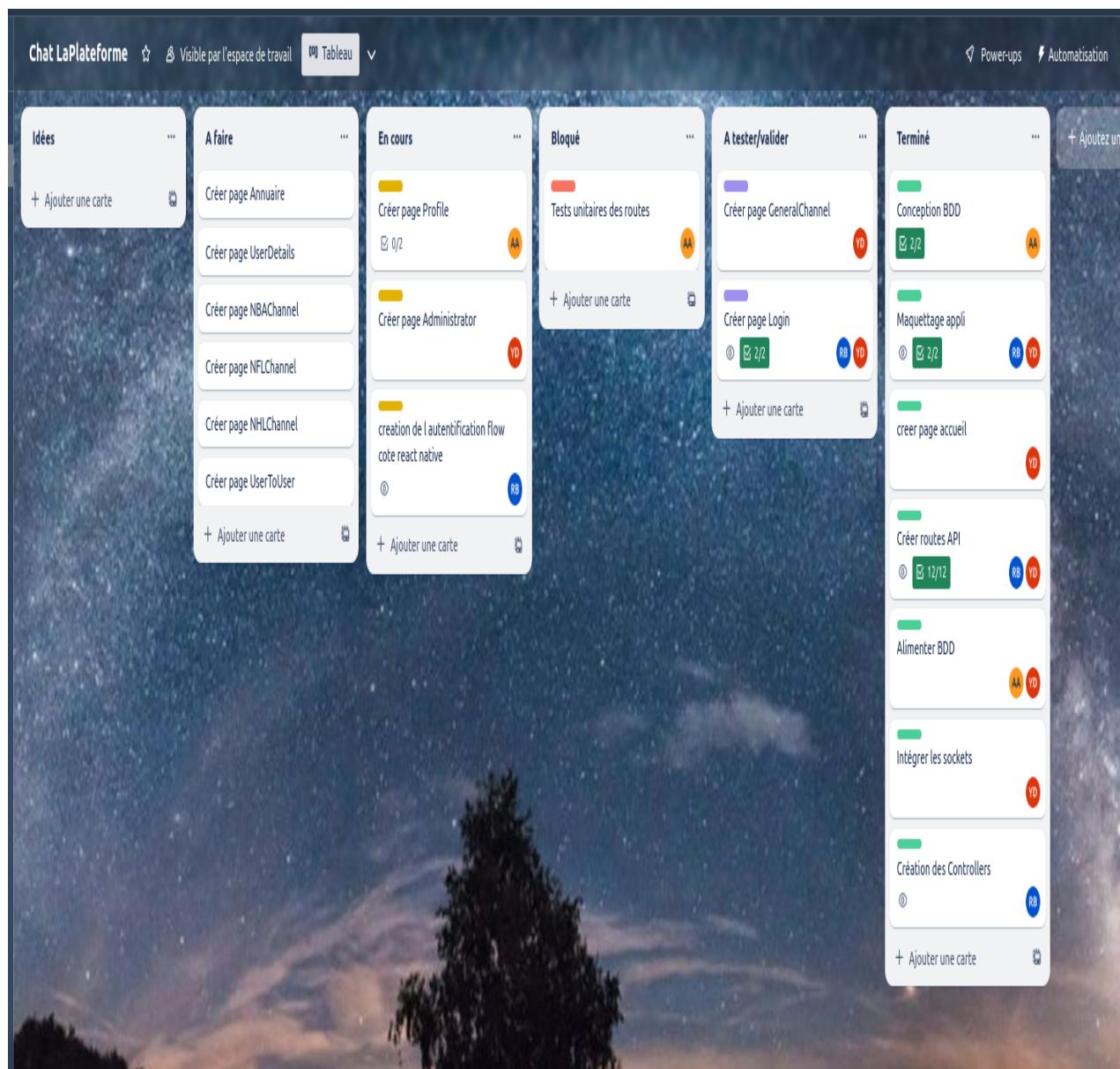
mobiles d'une planche à l'autre, traduisant leur avancement.

page 24Pour notre cas, plusieurs contraintes étaient à prendre en compte

concernant les délais de réalisation, le rythme de l'alternance ne nous permettant d'accorder que très peu de notre temps au projet, nous avons

fait le choix de ne pas se donner de délais impartis concernant la réalisation des tâches.

Pour s'adapter à notre rythme, nous avons ainsi tenté de suivre une approche « AGILE » pour collaborer et atteindre nos objectifs de développement.

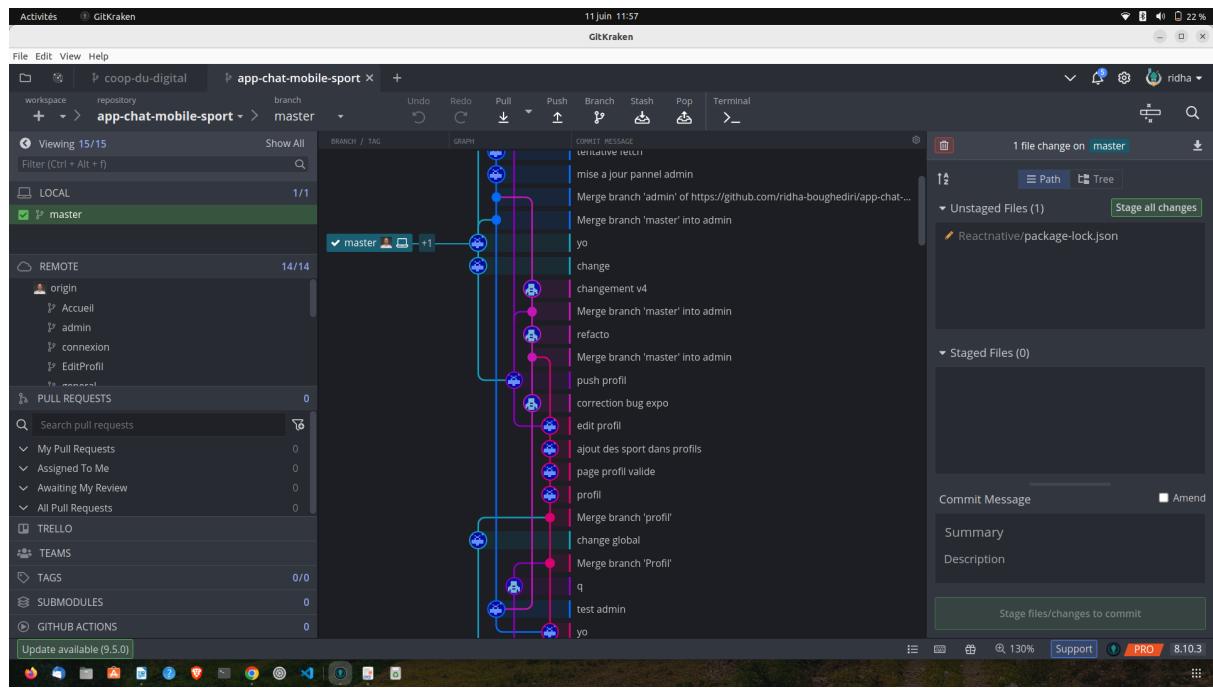


3.5.3. Versionning & Workflow

Pour gérer les différentes modifications apportées dans le code et sauvegarder l'ensemble du travail, pour ma part, j'ai utilisé le logiciel de versionning Git Kraken, associé à GitHub.

Git Kraken offre une interface graphique de gestion et de visualisation des

actions opérées sur Git et permet une vue d'ensemble des différentes branches, commits, etc.



Pour organiser les différentes parties de l'application j'ai créé 1 seul repositories avec 3 dossiers :

- Un dossier Backend qui contient la partie API développée avec **Node js**
- Un dossier Front-Native qui contient l'interface développée avec la **EXPO React Native**
- Un dossier Front qui contient l'interface développée avec **React js**

Sur les trois dossiers Frontends et Backend, nous avons choisi la méthode de travail GitFlow.

GitFlow permet de poser un cadre en standardisant la création de branches et ainsi modéliser un workflow selon le type de tâche que l'on va faire.

- **Branches principales :**

- Main : cette branche représente l'état du projet en production.
-

- **Branches secondaires :**

- feature : chaque nouvelle fonctionnalité sera développée sur une branche feature. La création d'une branche feature se fera toujours depuis la version la plus récente de la branche main et sera préfixée de cette manière

3.5.1. Gestion des dépendances

Parmi les différentes fonctionnalités développées, certaines dépendent de l'utilisation de librairies externes. Ces librairies permettent l'implémentation de fonctionnalités déjà créées par d'autres développeurs et nous évitent de perdre du temps.

Voici quelques exemples de librairies utilisées dans le projet :

- Axios qui est un Client HTTP simplifiant les requêtes en BDD
- React Navigation qui permet le routing et le partage de données entre plusieurs écrans

Pour gérer ces dépendances, nous utilisons l'outil **NPM**.

Ce gestionnaire pour Javascript permet à partir d'un fichier au format JSON "package.json" de répertorier tous les modules nécessaires au projet et leurs versions.

Grâce à cela, nous pouvons facilement les installer et les maintenir à jour.

4. Environnement Technique

4.1. Technologies & Outils

4.2. Spécificités techniques : Backend

4.2.1. API REST

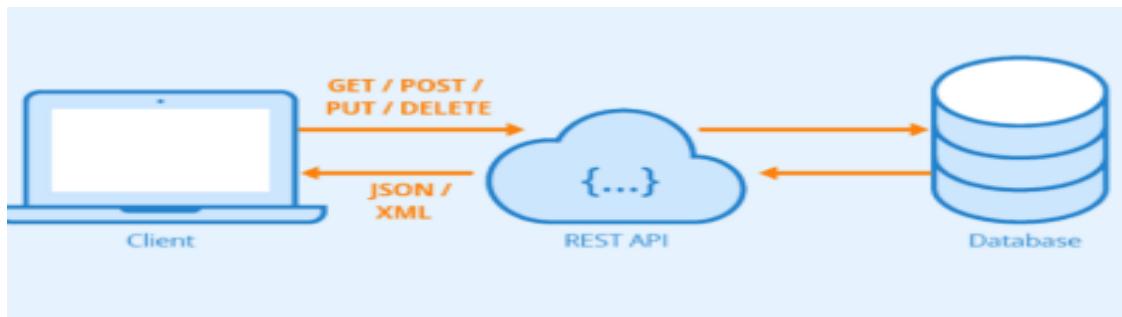
Pour l'application mobile, nous avons fait le choix concernant la gestion des données, de créer API, API Application Programming Interface ». Pour faire simple, une API est une interface qui permet d'échanger des données, de façon réutilisable et standardisée, entre différents composants d'une application ou entre une application et d'autres développeurs sans dépendre d'un langage en particulier.

Chaque donnée, sont regroupées au sein d'une « Collection » et peuvent être consultées par le client sous différents formats de données comme le JSON.

Plusieurs architectures d'API existent. Parmi elles, on retrouve l'architecture REST, Abréviation de Representational State Transfer,

c'est le modèle le plus utilisé pour la création d'API et c'est cela que nous avons choisi.

Elle met à profit l'utilisation des verbes http (GET, POST, PUT, DELETE, etc.) et permet une compréhension facile des différentes actions possibles sur une API.



La Client-serveur séparation, Cette séparation permet au client de s'occuper uniquement de la récupération et de l'affichage de l'information et permet au serveur de se concentrer sur le stockage et la manipulation des données.

Une mise en cache des données est nécessaire et permet de réduire le nombre de fois où un client et un serveur doivent interagir.

Il en résulte une accélération des temps de chargement pour l'utilisateur car pour une même requête exécutée plusieurs fois, le client pourra récupérer tout ou une partie des données directement sur la mémoire cache de sa machine / mobile sans avoir besoin de requêter le serveur.

Pour mener à bien le développement de notre back-end, nous avons fait de la veille afin de déterminer quelle solution serait la plus pertinente pour mettre en place notre API en fonction de nos contraintes.

Notre choix s'est tourné vers l'utilisation de Node.js avec Express.

parce que Express, qui est un framework minimaliste, rapide et flexible pour Node.js.

Express facilite le développement d'applications web en fournissant une abstraction de haut niveau pour les tâches courantes, tout en offrant une grande liberté et une flexibilité dans la façon dont vous concevez votre application.

Voici quelques raisons pour lesquelles je veux utiliser Express :

Gestion des routes simplifiée : Express vous permet de définir des routes pour gérer les différentes URL de votre application. Par exemple, je peux définir une route pour la page d'accueil ("/") et une autre pour le formulaire de contact ("/contact"). Cela facilite grandement la gestion des différentes requêtes HTTP.

Middleware puissant : Express propose un système de middleware qui me permet d'ajouter des fonctionnalités supplémentaires à l'application. Je peux utiliser des middlewares pour la gestion des sessions, l'authentification, la compression, la journalisation, la validation des données, etc.

Cela me permet de modulariser l'application et de réutiliser facilement des fonctionnalités communes.

Intégration aisée avec d'autres modules Node.js : Express fonctionne bien avec d'autres modules Node.js. Par exemple, je peux utiliser des moteurs de template tels que Handlebars ou EJS pour générer des vues dynamiques. Je peux également utiliser des modules tels que Sequelize pour interagir avec une base de données MySQL.

Flexibilité et extensibilité : Express est conçu pour être minimaliste et sans opinion, ce qui signifie que je peux l'utiliser comme base pour construire une application selon mes besoins spécifiques.

Il ne vous impose pas de structure ou de conventions rigides, me laissant libre de concevoir l'application de la manière qui me convient le mieux.

Grande communauté et écosystème : Express est l'un des frameworks web les plus populaires pour Node.js, ce qui signifie qu'il bénéficie d'une communauté active et d'un écosystème riche en modules complémentaires. Je peux trouver de nombreuses ressources, tutoriels et exemples en ligne pour m'aider dans votre développement.

En résumé, Express est un choix judicieux pour le développement d'applications web avec Node.js en raison de sa simplicité, de sa flexibilité, de sa puissance et de son intégration aisée avec d'autres modules. Il vous permet d'accélérer votre processus de développement tout en offrant la possibilité d'étendre votre application selon vos besoins spécifiques.

Tout d'abord, Express est un framework web pour Node.js qui facilite la création d'API REST en utilisant le modèle .

La partie serveur du framework est écrite en JavaScript, tandis que la partie client peut être développée en utilisant JavaScript (React Native et React.js).

Pour la persistance des données, nous utiliserons une bibliothèque d'accès aux données compatible avec Node.js, telle que Sequelize qui nous permettra de persister et d'interroger facilement nos données.

Ces bibliothèques sont optimisées pour la performance et offrent des fonctionnalités avancées telles que la génération automatique de requêtes SQL et la prise en charge de plusieurs systèmes de gestion de bases de données populaires tels que PostgreSQL, MySQL, . etc.

L'installation et la configuration d'Express et des bibliothèques de persistance peuvent être effectuées en utilisant le gestionnaire de packages npm, qui est inclus avec Node.js. Assurez-vous d'avoir Node.js et npm installés sur votre machine.

Voici les étapes générales pour installer et configurer Node.js avec Express :

1. Installer Node.js :

Téléchargez et installez la dernière version stable de Node.js à partir du site officiel (<https://nodejs.org/>). Suivez les instructions d'installation pour votre système d'exploitation.

2. Créez un nouveau projet :

Accédez à ce répertoire et initialisez un nouveau projet Node.js en exécutant la commande suivante : // ` ` // npm init // ` ` // Suivez les instructions pour configurer votre projet et générer le fichier package.json.

3. Installez Express :

Dans votre terminal, exécutez la commande suivante pour installer Express et enregistrer cette dépendance dans votre fichier package.json :

```
npm install express
```

4. Installez la bibliothèque de persistance de votre choix :

Selon votre préférence et les besoins de votre projet, installez Sequelize ou Mongoose en utilisant la commande npm appropriée. Par exemple, pour installer Sequelize, exécutez la commande suivante :

```
+ app-chat-mobile-sport git:(master) ✘ npm i express sequelize mysql2
```

5. Configurez votre application Express :

Créez un fichier JavaScript (par exemple, `app.js`) pour configurer votre application Express. Dans ce fichier, vous pouvez définir vos routes, vos modèles de données, vos contrôleurs, etc. Consultez la documentation d'Express et de la bibliothèque de persistance choisie pour des exemples de configuration et d'utilisation.

6. Démarrez le serveur :

Dans votre terminal, exécutez la commande suivante pour démarrer votre serveur Express : // ` ` // node app.js // ` ` //

Votre API basée sur Node.js avec Express est maintenant prête à être utilisée et vous pouvez commencer à développer vos fonctionnalités selon vos besoins spécifiques.

Voici ci-dessus sont une introduction générale à l'utilisation de Node.js avec Express pour créer une API. En fonction de vos besoins spécifiques, vous devrez peut-être effectuer des configurations supplémentaires et utiliser d'autres bibliothèques ou modules complémentaires.

les étapes 

Pour mener à bien le développement de notre back-end, nous avons fait des recherches afin de déterminer quelle solution serait la plus pertinente pour

mettre en place notre API en fonction de nos contraintes. Notre choix s'est tourné vers l'utilisation de Node.js avec Express.js et l'ORM Sequelize.

Tout d'abord, Express.js est un framework web minimaliste et flexible pour Node.js qui permet de créer des applications web et des API REST. Il suit le modèle de conception MVC (Modèle, Vue, Contrôleur) et facilite la création de routes, la gestion des requêtes et des réponses, ainsi que l'intégration de middlewares pour ajouter des fonctionnalités supplémentaires.

En ce qui concerne la persistance des données, nous avons choisi d'utiliser Sequelize, un ORM (Object-Relational Mapping) pour Node.js. Sequelize facilite l'interaction avec la base de données en permettant de manipuler les objets JavaScript plutôt que d'écrire directement des requêtes SQL. Il prend en charge plusieurs SGBD populaires tels que PostgreSQL, MySQL, SQLite, MSSQL, etc.

En utilisant Sequelize, nous pouvons définir des modèles pour nos entités et gérer les opérations de création, lecture, mise à jour et suppression (CRUD) de manière simplifiée. Il facilite également la création de requêtes complexes en utilisant des méthodes chaînées et fournit des fonctionnalités avancées telles que la validation des données, les associations entre les modèles, etc.

Pour installer Express.js et Sequelize dans notre projet, voici les étapes :

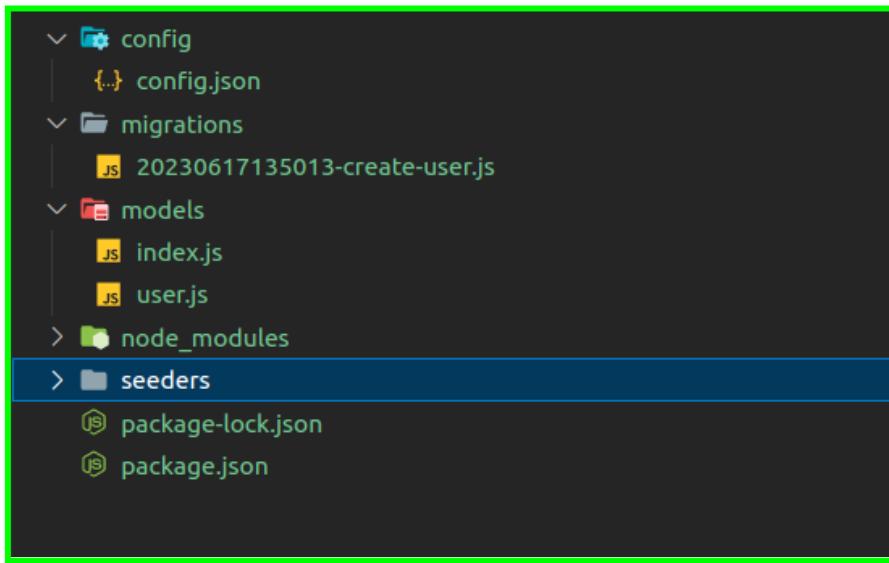
Il faut d'abord s'assurer d'avoir l'environnement adéquat sur sa machine, entre autres XAMPP et NPM pour ma part.

Pour créer un nouveau projet Node js :

```
→ app-chat-mobile-sport git:(master) ✘ npm init
```

ensuite installer les packages nécessaires au projet :

```
→ app-chat-mobile-sport git:(master) ✘ npm i express sequelize mysql2
```



Une fois cela fait, je n'ai plus qu'à créer ma base de données dans le SGBD et son schéma dans mon projet avec les lignes de commandes suivantes :

```
○ → apirest git:(master) ✘ npx sequelize-cli db:create  
npx sequelize-cli db:migrate
```

Le fichier "package.json" est utilisé par le gestionnaire de paquets de Node.js (NPM) pour gérer les dépendances du projet. Dans la section "dependencies", nous spécifions les librairies que nous utilisons dans notre projet, ainsi que leurs versions correspondantes. Lorsque nous exécutons la commande "npm install" à partir de la racine du projet, NPM télécharge et installe automatiquement toutes les dépendances listées dans le fichier "package.json":

```
You, 6 seconds ago | 1 author (You)
You, 5 months ago • folder back and frontend ...
{
  "name": "api-rest",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "start": "node -r dotenv/config server.js",
    "dev": "nodemon -r dotenv/config server.js",
    "test": "jasmine"
  },
  "author": "",
  "license": "ISC",
  "dependencies": {
    "bcrypt": "^5.1.0",
    "body-parser": "^1.20.1",
    "cors": "^2.8.5",
    "csurf": "^1.11.0",
    "dotenv": "^16.0.3",
    "express": "^4.18.2",
    "express-rate-limit": "^6.7.0",
    "express-session": "^1.17.3",
    "helmet": "^7.0.0",
    "jsonwebtoken": "^9.0.0",
    "mysql2": "^2.3.3",
    "nodemon": "^2.0.20",
    "sequelize": "^6.28.0",
    "sequelize-cli": "^6.5.2",
    "socket.io": "^4.5.4"
  },
  "devDependencies": {
    "jasmine": "^4.5.0"
  }
}
```

Cela facilite la gestion des librairies externes et des dépendances dans notre projet, car nous pouvons simplement partager le fichier "package.json" avec d'autres développeurs travaillant sur le même projet, et ils pourront ainsi installer toutes les dépendances requises en utilisant NPM.

phpMyAdmin

Structure SQL Rechercher Exporter Importer Opérations Privileges Procédures stockées Événements Plus

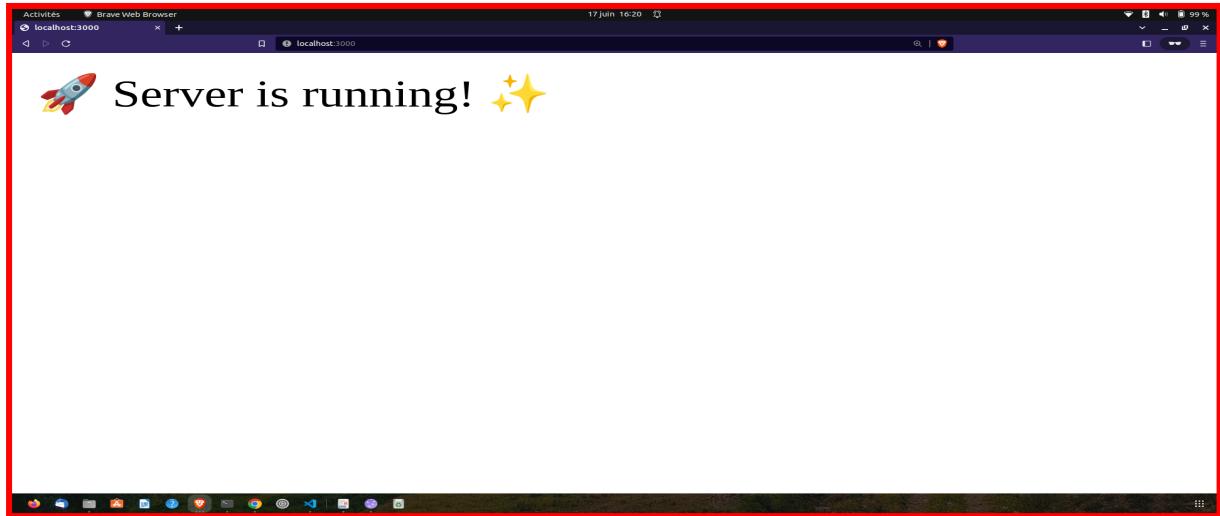
Filtres

Contenant le mot :

Table	Action	Lignes	Type	Interclassement	Taille	Perte
messages	Parcourir Structure Rechercher Insérer Vider Supprimer	1	InnoDB	utf8mb4_general_ci	32,0 kio	-
nbateams	Parcourir Structure Rechercher Insérer Vider Supprimer	0	InnoDB	utf8mb4_general_ci	32,0 kio	-
nfiteams	Parcourir Structure Rechercher Insérer Vider Supprimer	0	InnoDB	utf8mb4_general_ci	16,0 kio	-
nhiteams	Parcourir Structure Rechercher Insérer Vider Supprimer	0	InnoDB	utf8mb4_general_ci	16,0 kio	-
privatemessages	Parcourir Structure Rechercher Insérer Vider Supprimer	0	InnoDB	utf8mb4_general_ci	32,0 kio	-
roles	Parcourir Structure Rechercher Insérer Vider Supprimer	0	InnoDB	utf8mb4_general_ci	16,0 kio	-
rooms	Parcourir Structure Rechercher Insérer Vider Supprimer	0	InnoDB	utf8mb4_general_ci	32,0 kio	-
sports	Parcourir Structure Rechercher Insérer Vider Supprimer	0	InnoDB	utf8mb4_general_ci	16,0 kio	-
users	Parcourir Structure Rechercher Insérer Vider Supprimer	31	InnoDB	utf8mb4_general_ci	96,0 kio	-
9 tables	Somme	32	InnoDB	utf8mb4_general_ci	288,0 kio	0 o

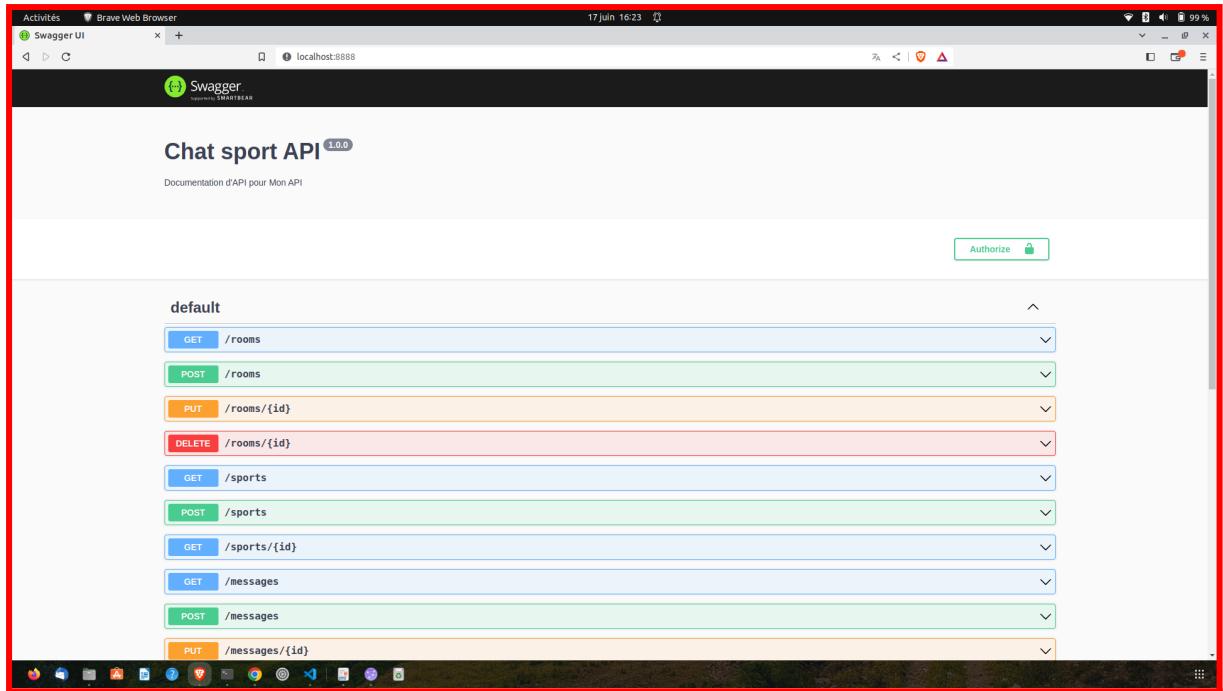
Tout cocher Avec la sélection :

Pour vérifier si cela à bien fonctionné, j'ouvre <https://localhost:8000>
Si cela fonctionne bien, il y aura cette vue sur le navigateur :



Lors de l'installation de swagger de cette manière, l'API sera exposée en tant que "/api".

Donc j'ouvre <http://localhost:8000/api> pour voir la documentation de l'API.



Swagger expose une description de l'API au format OpenAPI
Il intègre également une version personnalisée de Swagger UI , une belle interface restituant la documentation OpenAPI.
Il faut cliquer sur une opération pour afficher ses détails. et je peux également envoyer des requêtes à l'API directement depuis l'interface utilisateur.

Mon projet API Platform est désormais 100% fonctionnel, il ne reste plus qu'à implémenter les différentes entités et y mettre les annotations nécessaires pour le bon fonctionnement de mon api.

4.2.4.Nodejs / API Swagger : Création d'une Table

Une fois l'étape de la mise en place de mon projet en Node - Swagger réalisé, la première étape que j'ai effectué est la création de mes entités, c'est là où **Sequelize ORM** rentre en jeu : Sequelize facilite grandement la création de mes entités ainsi que mes champs, il prend tout en compte pour la création / modélisation de ma base de données.

A l'aide de quelques lignes de commande je crée mon modèle de données

et gère la persistance (la sauvegarde) dans une table.

Voici les commandes utiles pour rendre la création d'un modèle plus fluide, création d'une entité, création de la table correspondante, opération nommée migration.

Tout d'abord :

```
○ → backnode git:(main) ✘ npx sequelize-cli model:generate --name User \
--attributes firstName:string,isActive:boolean
```

Cette ligne de commande me sert à créer une entité, l'étape suivante est de choisir le nom de cette entité (ou nom de l'entité que je souhaite modifier).

tout d'abord le type (si c'est une string,, un booléen, un datetime) dans mon cas c'est une string et par là suite je dois choisir la longueur de la chaîne de caractère et enfin si elle peut être nul.
je n'ai plus qu'à répéter l'opération pour chaque champ de mon entité.

Une fois la création de mon entité effectué je n'ai plus qu'à faire une migration pour que ma base de données dans mon SGBD (Système de Gestion de Base de Données) se met à jour : cela se fait en une ligne de commande :

```
○ → backnode git:(main) ✘ npx sequelize-cli db:migrate
```

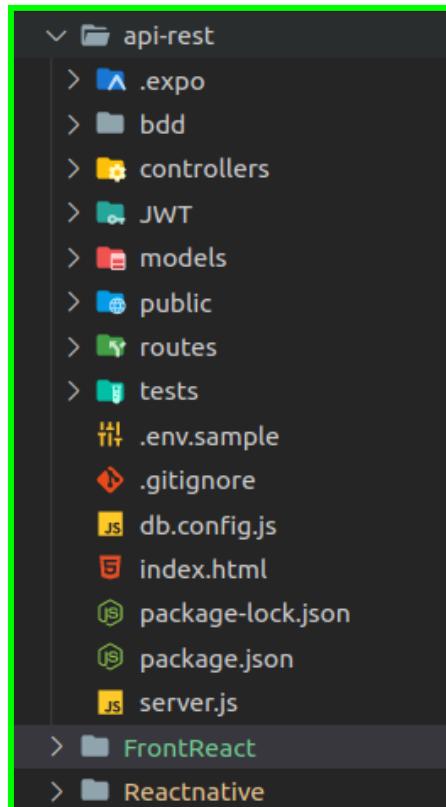
maintenant je retrouve ma table en base de données :

The screenshot shows the phpMyAdmin interface with the following details:

- Database:** chat_us_sport
- Table:** users
- Structure:**
 - Columns:**

#	Nom	Type	Interclassement	Attributs	Null	Value par défaut	Commentaires	Extra	Action
1	id	int(10)			Non	Aucun(e)		AUTO_INCREMENT	Modifier Supprimer Plus
2	email	varchar(255)	utf8mb4_general_ci		Non	Aucun(e)			Modifier Supprimer Plus
3	password	varchar(64)	utf8mb4_general_ci		Non	Aucun(e)			Modifier Supprimer Plus
4	firstname	varchar(255)	utf8mb4_general_ci		Non	Aucun(e)			Modifier Supprimer Plus
5	lastname	varchar(255)	utf8mb4_general_ci		Non	Aucun(e)			Modifier Supprimer Plus
6	login	varchar(255)	utf8mb4_general_ci		Non	Aucun(e)			Modifier Supprimer Plus
7	createdAt	datetime			Non	Aucun(e)			Modifier Supprimer Plus
8	updatedAt	datetime			Non	Aucun(e)			Modifier Supprimer Plus
9	deletedAt	datetime			Oui	NULL			Modifier Supprimer Plus
10	url_avatar	varchar(255)	utf8mb4_general_ci		Oui	NULL			Modifier Supprimer Plus
11	sport_id	int(10)			Oui	NULL			Modifier Supprimer Plus
12	nbteam_id	int(10)			Oui	NULL			Modifier Supprimer Plus
13	nftteam_id	int(10)			Oui	NULL			Modifier Supprimer Plus
14	nhteam_id	int(10)			Oui	NULL			Modifier Supprimer Plus
15	role_id	int(10)			Oui	NULL			Modifier Supprimer Plus
 - Actions:** Parcourir, Structure, SQL, Rechercher, Insérer, Exporter, Importer, Privileges, Opérations, Suivi, Délécheurs.

une fois toutes ces étapes réalisées, je vois que dans l'architecture de mes fichiers est mis également à jour, un fichier s'ajoute dans le dossier migrations et également un fichier sera ajouter dans le dossier pour les entités :



4.2.5.Node js / Sequelize ORM: Ajout des relations

Pour ajouter les cardinalités entre nos Table, Sequelize met à disposition lors de la création de ma table des possibilités de les paramétrer. Celui-ci nous permet d'établir les liens entre nos modèles en choisissant le type de cardinalité correspondant à mon MCD.

Pour créer des relations entre les collections (tables) avec Sequelize ORM, je peux utiliser les associations disponibles dans Sequelize. Voici comment vous pouvez définir différentes types de relations :

1. Association Many-to-One (Plusieurs-à-Un)
2. Association One-to-Many (Un-à-Plusieurs)
3. Association Many-to-Many (Plusieurs-à-Plusieurs)

Une fois les relations faite en passe à l'étape suivante : l'exécution du fichier **db.config.js**

```
You, 27 seconds ago | 1 author (You)
1  *****/
2  /** Import des modules nécessaires */
3  const { Sequelize } = require("sequelize");
4
5  *****/
6  /** Connexion à la base de données */
7  let sequelize = new Sequelize(
8    process.env.DB_NAME,
9    process.env.DB_USER,
10   process.env.DB_PASS,
11   {
12     host: process.env.DB_HOST,
13     port: process.env.DB_PORT,
14     dialect: "mysql",
15     logging: false,
16   }
17 );
18
19 /** Mise en place des relations */
20 const db = {};
21
22 db.sequelize = sequelize;
23 db.user = require("./models/user")(sequelize);
24 db.message = require("./models/messages")(sequelize);
25 db.nbaTeams = require("./models/nba_teams")(sequelize);
26 db.nflTeams = require("./models/nfl_teams")(sequelize);
27 db.nhlTeams = require("./models/nhl_teams")(sequelize);
28 db.privateMessage = require("./models/private_message")(sequelize);
29 db.room = require("./models/room")(sequelize);
30 db.Sport = require("./models/sport")(sequelize);
31 db.role = require("./models/role")(sequelize);
32
33 // relation equipe nba avec chaq sport
34
35 db.Sport.hasMany(db.nbaTeams, { foreignKey: "sport_id", onDelete: "cascade" });
36 db.nbaTeams.belongsTo(db.Sport, { foreignKey: "sport_id" });
37
38 // 1 relation user et sport
39
40 db.Sport.hasMany(db.user, { foreignKey: "sport_id", onDelete: "cascade" });
41 db.user.belongsTo(db.Sport, { foreignKey: "sport_id" });
42
43 // 2 relation user et nba
44
45 db.nbaTeams.hasMany(db.user, { foreignKey: "nbateam_id", onDelete: "cascade" });
```

comme dans l'exemple ci dessous, il nous demande d'abord quelle est

l'entité à modifier, ensuite le nom du champs que l'on souhaite attribué, par la suite définir que c'est un type ManyToOne, ensuite lui définir un nom, si le champs peut être nul et si je souhaite ajouter une nouvelle propriété.

4.3. Spécificités techniques : Frontend

4.3.1. React Native

De nos jours, plusieurs types d'applications mobiles existent. D'un côté, on retrouve les applications mobiles natives, conçues pour être exécutées sur un système d'exploitation spécifique, elles requièrent l'apprentissage d'un langage propre à la plateforme. Cela nous aurait amené à faire un gros travail en codant plusieurs fois l'application selon le nombre d'OS avec lesquels elle doit être compatible. À l'inverse des applications natives, on a les applications cross platform. Leur développement requiert l'utilisation de frameworks tels que Ionic ou Cordova, ces derniers vont à partir d'une codebase pouvoir rendre une application compatible avec les différentes plateformes en ne la codant qu'une seule fois.

Cela représente un gain de productivité non négligeable, cependant ce modèle d'application présente quelques inconvénients, ce type de framework ne va pas donner accès aux composants natifs du téléphone, limitant ainsi les possibilités de développement.

De plus, les performances seront plus basses de ce qu'apporte une application native.

Heureusement, d'autres solutions existent pour remédier aux problèmes cités précédemment.

Ainsi, pour ne pas avoir à apprendre deux langages spécifiques mais garder l'avantage d'une application native, nous avons fait le choix de développer Côté Pote grâce à la librairie EXPO React Native.

EXPO React Native permet de développer des applications iOS, Android à partir de la même codebase Javascript / Typescript et ses concepts sont pratiquement identiques à ceux React, à la différence que EXPO React Native ne manipule pas le DOM.

Les applications mobiles construites avec EXPO React Native donnent des applications entièrement natives, qui utilisent les mêmes API que si elles étaient développées dans Xcode ou Android Studio.

4.3.2.Expo

Pour pouvoir mettre en place l'environnement de développement pour React Native où plus généralement pour la grande majorité des Framework Front-End, je sais que j'ai besoin de NodeJS qui est une plateforme logicielle libre en JavaScript, orientée vers les applications réseau événementielles.

Il y a également NPM ou Node Packet Manager, écrit en grande partie en JavaScript, est indissociable du succès de node.

Il permet de gérer ou bien de publier de nouveaux logiciels au sein de l'écosystème NodeJS.

Pour installer React Native, nous avons utilisé la librairie Expo-CLI. Expo fournit un SDK permettant la création et le déploiement d'applications mobiles avec React Native en fournissant un ensemble d'outils et de services construits autour de ce dernier.

Par exemple, il nous permet à l'aide de l'utilitaire « watchman » de traquer nos fichiers et compiler notre code à la volée pour visualiser les résultats sur différents simulateurs iOS / Android ou directement sur notre appareil

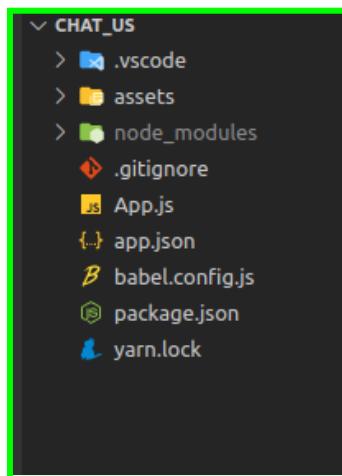
mobile personnel ce qui apporte un confort supplémentaire pendant le développement.

Pour créer un nouveau projet géré par Expo, il faut exécuter la commande Expo init qui va créer l'environnement nécessaire au développement de notre application.

```
expo init my-app
```

```
TERMINAL
○ → reactnative git:(main) ✘ expo init ■
```

Une fois cette opération faite, je me retrouve avec une architecture de ce genre :



Ce qui est pratique une fois de plus avec ce type de framework, je peux aisément utiliser et télécharger des librairies pour pouvoir s'en servir par la suite sur mon projet, je peut retrouver simplement dans le fichier "composer.json" toutes les librairies et dépendances que j'ai sur mon projet :

```
git commit -m "Initial commit"
You, last month * 2nde commit ...
{
  "name": "doe",
  "version": "1.0.0",
  "main": "node_modules/expo/AppEntry.js",
  "scripts": {
    "start": "expo start",
    "android": "expo start --android",
    "ios": "expo start --ios",
    "web": "expo start --web"
  },
  "dependencies": {
    "@expo/google-fonts/roboto": "^0.2.3",
    "@expo/vector-icons": "^13.0.0",
    "@expo/webpack-config": "^18.0.1",
    "@react-native-async-storage/async-storage": "1.17.11",
    "@react-navigation/bottom-tabs": "^6.5.7",
    "@react-navigation/drawer": "^6.6.2",
    "@react-navigation/native": "^6.1.6",
    "@react-navigation/native-stack": "^6.9.12",
    "@react-navigation/stack": "^6.3.16",
    "axios": "^1.4.0",
    "expo": "~48.0.15",
    "expo-asset": "~8.9.1",
    "expo-image-picker": "~14.1.1",
    "expo-secure-store": "~12.1.1",
    "expo-status-bar": "~1.4.4",
    "jwt-decode": "^3.1.2",
    "react": "18.2.0",
    "react-dom": "18.2.0",
    "react-native": "0.71.3",
    "react-native-dropdown": "^0.0.6",
    "react-native-elements": "^3.4.3",
    "react-native-gifted-chat-expo": "^1.1.0",
    "react-native-paper": "^5.8.0",
    "react-native-reanimated": "~2.14.4",
    "react-native-safe-area-context": "4.5.0",
    "react-native-screens": "~3.20.0",
    "react-native-vector-icons": "^9.2.0",
    "react-native-web": "~0.18.10",
    "socket.io-client": "^4.6.1"
  },
  "devDependencies": {
    "@babel/core": "^7.20.0"
  },
  "private": true
}
```

4.3.3.React Navigation

Maintenant il faut savoir que la navigation entre différente vue est très différente de celle du web classique, il n'y a pas de "href" qui ramène d'un lien à l'autre.

Il y a React Navigation qui est une bibliothèque populaire pour le routage et la navigation dans une application React Native. Cette bibliothèque aide à résoudre le problème de la navigation entre plusieurs écrans et du partage des données entre eux.

Pour naviguer entre les écrans, j'utilise StackNavigator, il fonctionne exactement comme une pile d'appels.

Chaque écran vers lequel je navigue est poussé vers le haut de la pile. Chaque fois que j'appuie sur le bouton Retour, les écrans se détachent du haut de la pile.

tout d'abord je commence par installer la librairie avec cette commande sur mon terminal :

```
npm install @react-navigation/native
```

Ensuite, j'installe `@react-navigation/stack` et ses dépendances

Pour Côté Pote, il est possible de naviguer au travers de l'application grâce à un composant TabBar et Drawer, situé au bas de l'écran, qui permet de basculer entre les écrans principaux de l'application.

Pour que ces composants soient fonctionnels, ils doivent être les enfants du composant `<NavigationContainer>`, fourni par React Navigation, et que l'on va appeler à la racine de notre application dans le composant App.js. `<NavigationContainer>` est la base qui va nous permettre de gérer notre arbre de navigation ainsi que ses différents états et fournir différentes fonctionnalités propres à la plateforme où il sera exécuté.

Après avoir créé mon `<Navigation Container>` nous appelons la fonction `createBottomTabNavigator` qui va nous créer notre TabBar et retourner un objet contenant deux propriétés, `Navigator` et `Screen`. Chaque composant `<Tab.Screen>` correspond à une Tab au bas de l'écran et nous permet de changer d'écran.

-

Une prop `name`, qui va nous permettre de naviguer vers mon écran grâce au nom qui lui est attribué

```

const Tab = createBottomTabNavigator();

const ProfileStack = () => {
  return (
    <Tab.Navigator
      screenOptions={{
        activeTintColor: "blue",
        inactiveTintColor: "gray",
      }}
    >
      <Tab.Screen
        name="Compte"
        component={ProfileScreen}
        options={{
          tabBarIcon: ({ color, size }) => (
            <MaterialCommunityIcons
              name="face-man-profile"
              size={size}
              color={color}
            />
          ),
        }}
      />

      <Tab.Screen
        name="photo"
        component={UploadImageScreen}
        options={{
          tabBarIcon: ({ color, size }) => (
            <EvilIcons name="image" size={size} color={color} />
          ),
        }}
      />

      <Tab.Screen
        name="Modifier"
        component={EditProfileScreen}
        options={{
          tabBarIcon: ({ color, size }) => (
            <Entypo name="edit" size={size} color={color} />
          ),
        }}
      />
    </Tab.Navigator>
  );
};

```

Ces derniers doivent tous être contenu dans un composant parent `<Drawer.Navigator>` et contenir au minimum les deux tabbar suivantes pour permettre la transition entre les écrans :

- Une prop `component`, correspondant au functional component de l'écran qui doit être rendu 

Après avoir créé nos Tabs, nous avons créé les écrans qui seront affichés au sein de ces derniers.

La création d'un écran se fait de la même manière que la création d'une Tab à l'exception de la fonction que l'on va appeler qui est `createNativeStackNavigator`.

Grâce à cette fonction, nous pouvons créer une Stack contenant plusieurs écrans et établir une navigation entre eux grâce à la gestion événementielle des composants fournis par React Native.

```

const Tab = createBottomTabNavigator();

const ProfileStack = () => {
  return (
    <Tab.Navigator
      screenOptions={{
        activeTintColor: "blue",
        inactiveTintColor: "gray",
      }}
    >
      <Tab.Screen
        name="Compte"
        component={ProfileScreen}
        options={{
          tabBarIcon: ({ color, size }) => (
            <MaterialCommunityIcons
              name="face-man-profile"
              size={size}
              color={color}
            />
          ),
        }}
      />

      <Tab.Screen
        name="photo"
        component={UploadImageScreen}
        options={{
          tabBarIcon: ({ color, size }) => (
            <EvilIcons name="image" size={size} color={color} />
          ),
        }}
      />

      <Tab.Screen
        name="Modifier"
        component={EditProfileScreen}
        options={{
          tabBarIcon: ({ color, size }) => (
            <Entypo name="edit" size={size} color={color} />
          ),
        }}
      />
    </Tab.Navigator>
  );
};

```

4.3.4. Axios : Interaction avec la base de données

Pour interagir avec mes données j'ai utilisé la librairie Axios.
 Cette librairie est un client HTTP nous permettant de faire des requêtes en

BDD et de recevoir nos réponses sous forme de Promesse. Il fournit plusieurs méthodes facilitant l'écriture des requêtes et contrairement à Fetch, il permet un meilleur support des réponses renvoyées par le serveur car elles n'ont pas besoin d'être converties au format JSON et peuvent être exploitées lorsque la promesse est résolue.

Pour effectuer une requête avec Axios, nous devons importer le module axios au sein de notre fichier en charge de la requête.

Ce module comporte plusieurs fonctions dans lesquelles nous devons renseigner en paramètre les informations liées à la requête.

Chaque fonction est un raccourci lié à la méthode d'un verbe http et pour chaque requête il faut passer en paramètre l'URL de notre ressource et un objet contenant notre body dans le cas d'une écriture en BDD.

```
1 import axios from "axios";
2 import * as SecureStore from "expo-secure-store";
3 import { BASE_URL } from "../config";
4 import { set } from "react-native-reanimated";
5 import jwtDecode from "jwt-decode";
6
7 export const AuthContext = createContext();
8
9
10 const AuthProvider = ({ children }) => {
11   const [userToken, setUserToken] = useState(null);
12
13   const [isLoading, setIsLoading] = useState(false);
14   const [userInfo, setUserInfo] = useState(null);
15   const [test, setTest] = useState("je suis un developpeur millionnaires");
16
17   const connexion = (email, password) => {
18     setIsLoading(true);
19     axios
20       .post(`${BASE_URL}/auth/login`, { email, password })
21       .then(async (res) => {
22         await SecureStore.setItemAsync("userToken", res.data.access_token);
23         const user = jwtDecode(res.data.access_token);
24         setUserInfo(user);
25         setUserToken(res.data.access_token);
26       })
27       .catch((e) => {
28         console.error("erreur de connexion", e);
29       })
30       .finally(() => {
31         setIsLoading(false);
32       });
33   };
34 }
35 
```

5. Présentation du jeu d'essai

5.1. Parcours utilisateur pour la connexion d'un user

Je vais vous présenter les extraits de code retracant les étapes du processus de la connexion d'un utilisateur.

Le point d'entrée de l'application se faisant au sein du fichier App.js,

c'est à partir de du composant **AppNav** que l'**authentication flow** la navigation se fait :

Si je suis connecté j'accède à l'**App-Stack** qui contint l'éléments de l'application sinon je suis redirigé vers l'**Auth-Stack** qui contient les éléments de connexion et l'inscription et reset password .

j'ai un drawer dans le cas ou un utilisateur est connecté ou une stack de navigation avec 3 vues si l'utilisateur n'est pas authentifié.

L'onglet Connexion est la première de la pile et donne accès à la première page du parcours utilisateur pour la connexion.

```
You, last month | 1 author (You)
import { ActivityIndicator, View } from "react-native";      You, last month • 2nd
import AppStack from "./AppStack";
import AuthStack from "./AuthStack";
import React, { useContext } from "react";
import { NavigationContainer } from "@react-navigation/native";
import { AuthContext } from "../Context/AuthContext";

function AppNav() {
  const { isLoading, userToken } = useContext(AuthContext);
  if (isLoading) {
    return (
      <View style={{ flex: 1, justifyContent: "center", alignItems: "center" }}>
        <ActivityIndicator size={"large"} />
      </View>
    );
  }

  return (
    <NavigationContainer>
      {userToken !== null ? <AppStack /> : <AuthStack />}
    </NavigationContainer>
  );
}

export default AppNav;
```

et également

```
...
import React from "react";
import AppNav from "./src/navigation/AppNav";
import AuthProvider from "./src/Context/AuthContext";

const App = () => {
  return (
    <AuthProvider>
      <AppNav />
    </AuthProvider>
  );
};

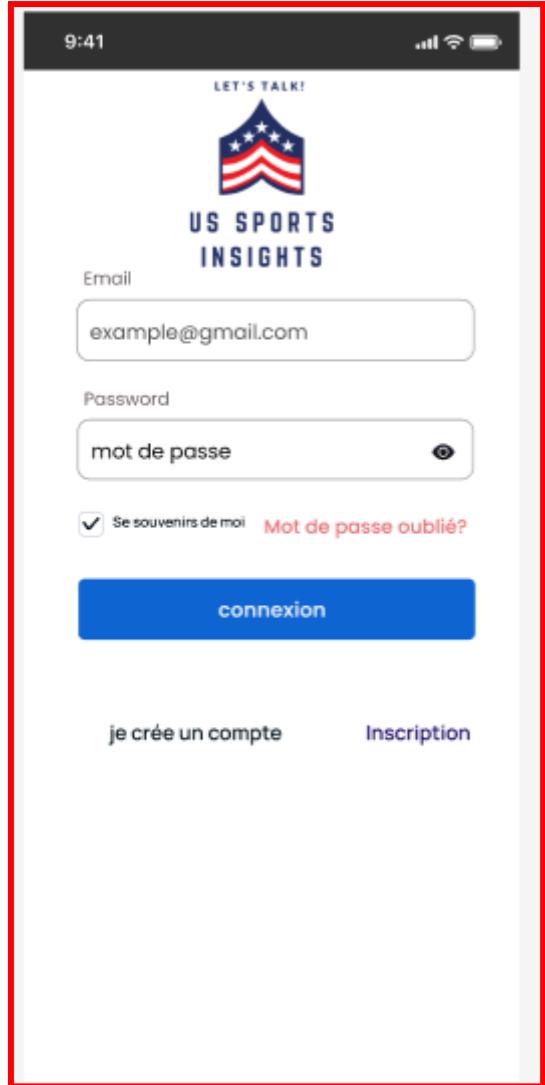
export default App;
```

Cette page, ainsi que les suivantes sont les enfants d'une Stack d'écran créée grâce à la fonction createStackNavigator au sein de l'application.

Lorsque cette fonction est appelée, elle met à disposition deux propriétés, Navigator et Screen, j'importe également NavigationContainer qui va englober Navigator et les Screens.

<Screen> dispose de plusieurs props dont deux importantes, "name" qui permet de naviguer vers le composant et ensuite "component" qui va effectuer le rendu de ce que je veux afficher.

Le screen qui sera affiché par défaut sera "Connexion" Le rôle de cet écran est d'afficher les inputs mail et mot de passe, c'est input vont me permettre de faire un call API avec leurs valeurs en paramètre pour interroger mon API.



```

import React, { createContext, useEffect, useState } from "react";      You, last month * inscri
import axios from "axios";
import * as SecureStore from "expo-secure-store";
import { BASE_URL } from "../config";
import { set } from "react-native-reanimated";
import jwtDecode from "jwt-decode";

export const AuthContext = createContext();

const AuthProvider = ({ children }) => {
  const [userToken, setUserToken] = useState(null);

  const [isLoading, setIsLoading] = useState(false);
  const [userInfo, setUserInfo] = useState(null);
  const [test, setTest] = useState("je suis un developpeur millionnaires");

  const connexion = (email, password) => {
    setIsLoading(true);
    axios
      .post(`${BASE_URL}/auth/login`, { email, password })
      .then(async (res) => {
        await SecureStore.setItemAsync("userToken", res.data.access_token);
        const user = jwtDecode(res.data.access_token);
        setUserInfo(user);
        setUserToken(res.data.access_token);
      })
      .catch((e) => {
        console.error("erreur de connexion", e);
      })
      .finally(() => {
        setIsLoading(false);
      });
  };

  const logout = async () => {
    setIsLoading(true);
    try {
      await SecureStore.deleteItemAsync("userToken");
      setUserToken(null);
      // Autres opérations de déconnexion ici, telles que supprimer le jeton de l'état global
    } catch (error) {
      console.log(error);
    }
    setIsLoading(false);
  };
}

```

J'effectue cette requête dans un hook `useEffect` qui me sert à renvoyer une valeur mémorisée, que je passe ensuite en paramètre à la méthode `React.createContext()` avec lequel je vais englober tous les composants dans celui-ci pour pouvoir y avoir accès de manière global sans me soucier de comment faire passer de l'enfant au parent ou inversement des données, des méthodes ou des variables de manière global dans mon application.

```

6
7  const isLoggedIn = async () => {
8    try {
9      setIsLoading(true);
10     let userToken = await SecureStore.getItemAsync("userToken");
11     setUserToken(userToken);
12     setIsLoading(false);
13     const user = jwtDecode(userToken);
14     setUserInfo(user);
15   } catch (error) {
16     console.log(`isLoggedIn in error ${error}`);
17   }
18 };
19
20 useEffect(() => {
21   isLoggedIn();
22 }, []);
23
24 return (
25   <AuthContext.Provider
26     value={{
27       test,
28       userToken,
29       userInfo,
30       isLoading,
31       isLoggedIn,
32       connexion,
33       logout,
34     }}
35   >
36     {children}
37   </AuthContext.Provider>
38 );
39 };
40 export default AuthProvider;
41

```

Je stocke le résultat de la réponse dans le storage interne du mobile de l'utilisateur grâce à aux méthodes mises à disposition dans la librairie "SecureStore".

Les hooks sont des fonctions faisant partie de l'écosystème React et apportent au sein des composants de type fonction, des fonctionnalités auparavant accessibles seulement aux composants de type classe.

- Le hook useEffect permet de gérer les cycles de vie d'un composant et déclencher des actions lorsque le rendu de celui-ci est terminé.
- Le hook useState il permet de gérer des états au sein du composant et mettre à jour ce dernier si un changement survient sur le state, ayant pour effet de lancer un nouveau rendu pour mettre à jour l'écran.

Pour le style, ce n'est toujours pas du CSS :

Avec React Native, je stylise mon application en utilisant JavaScript. Tous les composants de base acceptent un accessoire nommé style. Les noms et les valeurs de style correspondent généralement au fonctionnement de CSS sur le Web, sauf que les noms sont écrits en utilisant la camelCase, par exemple backgroundColor plutôt que background-color...

Il faut commencer par importer "StyleSheet" depuis la librairie react-native et ensuite déclaré une constante où l'on va stocker la méthode "StyleSheet.Create({})" et à l'intérieur déclarer toute les classes dont j'ai besoins :

```
3
4  const styles = {
5    button: [
6      padding: 10,
7      backgroundColor: "#0E64D2",
8      borderRadius: 5,
9      marginRight: 10,
10     ],
11    buttonText: [
12      color: "#fff",
13      fontWeight: "bold",
14     ],
15  };
16
17  export default WelcomeScreen;
18
```

```
14  <Text
15    style={[
16      fontSize: 18,
17      marginTop: 20,
18      marginStart: 10,
19      marginEnd: 10,
20      textAlign: "center",
21    ]}
22  >
23    Le Premier tchat pour discuter Pronostique et suivi de ses équipes de
24    sport américain favorites
25  </Text>
```

6.Extrait d'une recherche à partir de site anglophone

Durant le développement de ce projet, j'ai rencontré de nombreuses situations qui ont nécessité une recherche d'informations.

En grande majorité, les solutions recherchées trouvaient réponses sur des sites anglophones (surtout StackOverFlow).

Pour décrire une de ces situations, je me suis tourné vers **StackOverFlow et Expo** pour savoir comment utiliser le **secure store** que j'avais avec le context api que j'ai utilisé dans ce projet

