Réalisation de l’application web & mobile

—

US Sports Chat

Sommaire

Table des matières

Introduction

- Présentation personnelle

Je m'appelle Yonathan Darmon, j'ai 39 ans et je vis à Marseille. Je me suis initié à la programmation en Septembre 2021 lorsque j'ai commencé une formation de l'école La Plateforme avec laquelle j'ai obtenu le titre RNCP 5 « Développeur Web & Mobile ».

M'étant pris de passion pour ce domaine j'ai décidé de poursuivre cette année avec le cursus Concepteur d'Applications et même de poursuivre sur le cursus Master IT and Business toujours avec la même école.

Aujourd'hui je prépare donc l'obtention du titre « Concepteur et Développeur d'Applications » et ce en alternance avec l'entreprise «Aparkate».

- Présentation du projet en anglais

My project for the "Concepteur & Développeur d'Applications" professional title is a mobile chat application focused on American sports and their major leagues like NBA, NFL or NHL. The project was proposed by my school, and I chose this theme because I have been a big fan of this sports, especially the NFL, for a long time.

The application will serve as a chat platform for sports enthusiasts to discuss various matches and share predictions. The target audience is anyone interested in American sports, particularly those who want to interact with other passionate fans.

To develop this application, I have chosen to work with NodeJS and Express for the API, React Native for the application front-end, and ReactJS for the administration panel. These technologies were chosen for their reliability and effectiveness in building mobile applications with dynamic and interactive user interfaces.

Overall, my project aims to provide a platform for sports enthusiasts to connect and share their passion for American sports, while also demonstrating my skills and expertise in application development using cutting-edges technologies.

- Compétences couvertes par le projet

Ce projet couvre les compétences du titre suivantes :

- Maquetter une application
- Développer une interface utilisateur de type desktop
- 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
- Collaborer à la gestion d'un projet informatique et à l'organisation de l'environnement
- de développement
- 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

Il existe déjà de nombreuses applications de type chat en direct, les plus connues étant Whatsapp ou encore Telegram qui se sont fait connaître principalement pour le chiffrement des conversations et donc pour leurs sûretés. Le but de notre chat n'était donc pas de révolutionner le marché de la

conversation en direct mais de se mettre au pli sur ce qui existe déjà et comprendre le fonctionnement des ces applications de manière à améliorer et renforcer nos compétences.

- Les utilisateurs du projet

Ce projet est découpé en deux parties : une application mobile et une interface web.

La partie application mobile est destinée à des utilisateurs de tout horizon, qui s'inscrivent eux même sur l'application et peuvent gérer leurs profils.

La partie site web est adressée uniquement aux personnes ayant des droits d'administrateur. De ce panel ils pourront modérer des messages, supprimer ou modifier des conversations, envoyer des informations sur les différents chat mais aussi gérer les utilisateurs.

- Les fonctionnalités attendues

Application mobile :

Page d'accueil :

La page d'accueil est adaptative, si l'utilisateur n'est pas inscrit ou connecté, sur la page d'accueil il aura les formulaires pour ce faire. Si l'utilisateur est connecté, au lancement de l'application il sera directement redirigé vers la page de chat général.

Page inscription : Cette page permet à un utilisateur de s'inscrire. Il doit renseigner lors de l'inscription son mail, son login et son mot de passe.

Page connexion : Cette page permet à un utilisateur de se connecter.

Page profil : Sur cette page l'utilisateur peut voir l'ensemble de ses informations, à savoir : Prénom, Nom, Login, Email, Avatar, Sport favori, Equipe favorite.

Page modification profil : Sur cette page l'utilisateur peut modifier l'ensemble des informations vues précédemment ainsi que son mot de passe.

Page chat général : Cette page est celle ou l'ensemble des utilisateurs peuvent discuter.

Page chat NBA : Cette page est dédiée aux utilisateurs ayant choisi le basket comme sport favori.

Page chat NHL : Cette page est dédiée aux utilisateurs ayant choisi le hockey comme sport favori.

Page chat NFL : Cette page est dédiée aux utilisateurs ayant choisi le football américain comme sport favori.

Page amis : Sur cette page l'utilisateur peut retrouver la liste des autres utilisateurs avec lequel il est ami.

Page annuaire : Sur cette page l'utilisateur peut retrouver l'ensemble des utilisateurs inscrits sur l'application.

Application web :

Page de connexion : Cette page est la page d'accueil du panel d'administration où l'administrateur doit se connecter.

Page de statistiques de l'applications : Après connexion l'administrateur arrive directement sur une page où se trouvent l'ensemble des statistiques de l'application comme par exemple le nombre d'utilisateurs enregistrés, le nombre de conversations créées ou le nombre de message envoyés.

Page CRUD utilisateurs : Sur cette page un administrateur peut gérer l'ensemble des utilisateurs.

Page CRUD conversations : Sur cette page l'administrateur peut gérer l'ensemble des conversations, comme par exemple en créer une, supprimer des messages…

- ### Contexte technique

L'application mobile devra être accessible sur tous le systèmes d'exploitation Androïd et iOS

L'application web devra être accessible sur tous les navigateurs.

Conception du projet

- ### Choix du développement

  - #### Choix des langages

Pour réaliser ce projet j'ai pris la décision d'utiliser uniquement JavaScript avec NodeJS comme environnement de développement.

J’ai fait le choix d’utiliser JavaScript et NodeJS pour différentes raisons.

Pour JavaScript parce que c’est l’un des langages de programmation les plus utilisés et les plus populaires dans le monde du développement web et mobile. Il offre une grande flexibilité, des performances élevées et une vaste gamme de bibliothèques et frameworks pour le développement de l’interface utilisateur et de la logique de l’application.

Node.js, quant à lui, est un environnement d’exécution JavaScript côté serveur qui permet de développer des applications web et mobiles évolutives et performantes. Il offre une architecture événementielle, un modèle de traitement non bloquant et une grande efficacité dans la gestion des entrées/sorties, ce qui le rend particulièrement adapté pour le développement de serveurs de données, de services web et d’API pour les applications mobiles.

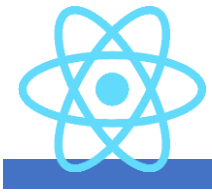
En combinant les deux, je m'assurais de créer une application mobile performante, rapide et efficace avec une interface utilisateur dynamique et interactive. De plus, il a été facile de réutiliser le code entre le front-end et le back-end, ce qui à considérablement réduit le temps de développement.

- Choix des frameworks

Express.js est un framework Node.js populaire pour la création d’applications web et mobiles évolutives et performantes. Il offre une grande flexibilité, une syntaxe simple et une grande variété de fonctionnalités, ce qui facilite grandement le développement et la maintenance d’applications complexes.

Personnellement, j’ai choisi Express car il est facile à prendre en main et qu’il permet de créer une API ainsi que gérer les demandes http et les routes ou encore effectuer des tâches courantes telles que l’authentification ou la gestion des sessions. Il offre aussi une grande variété de middleware ce qui permet d’ajouter un grand nombre de fonctionnalités à l’application.

Ayant eu à réaliser ce projet sur une durée de temps assez courte, Express.js m’a permis de développer de façon rapide et efficace une API.



Quant au front-end de mon application mobile, j'ai choisi d'utiliser le framework React Native car il est écrit en JavaScript, qu'il est open source et qu'il permet de réaliser une interface utilisateur dynamique et interactive, une navigation fluide, des performances élevées et une expérience utilisateur immersive. De plus, il offre une vaste bibliothèque de composants prêt à l'emploi ce qui m'a permis de gagner du temps lors de la réalisation de ce projet.

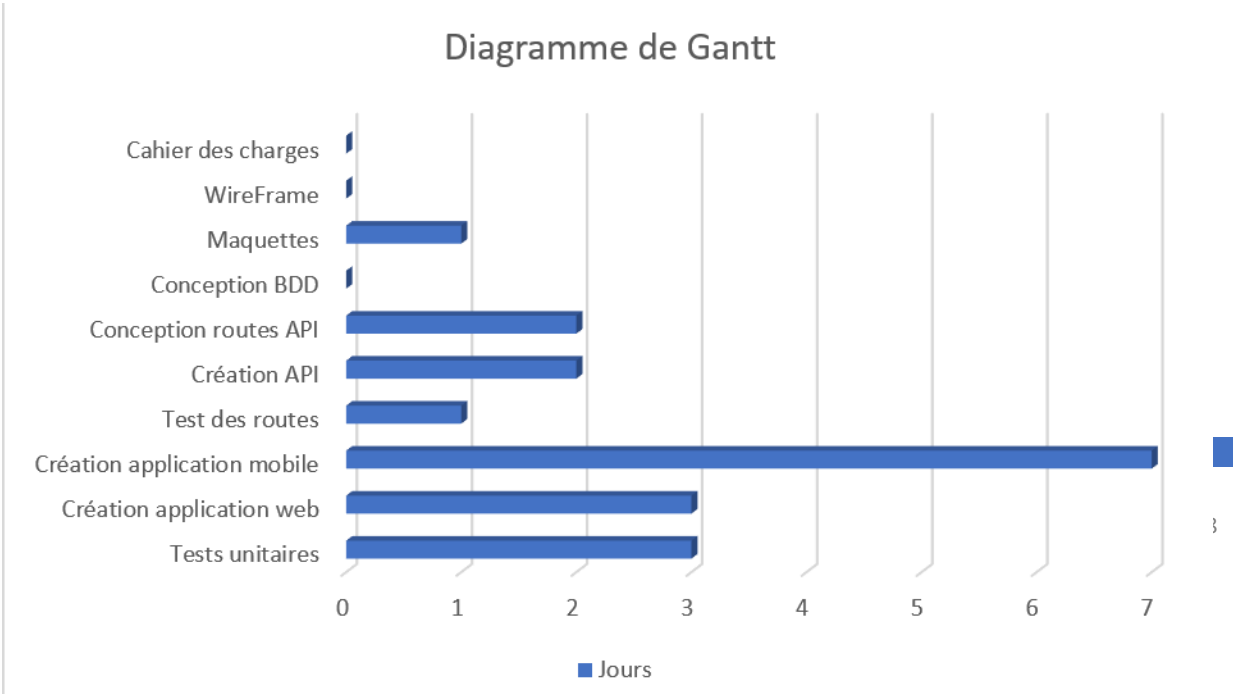
○ Logiciels et autres outils

- Visual Studio Code pour écrire mon code,
- Postman pour effectuer les requêtes API,
- GitHub et GitHub Desktop pour le versionning,
- Trello pour organiser mon travail,
- NPM pour installer les paquets,
- Figma pour la création de mes maquettes front,
- LucidChart pour le maquettage de ma base de données,
- Expo pour émuler mon application mobile sur mon téléphone,
- Canva pour la création de ma charte graphique et du logo.

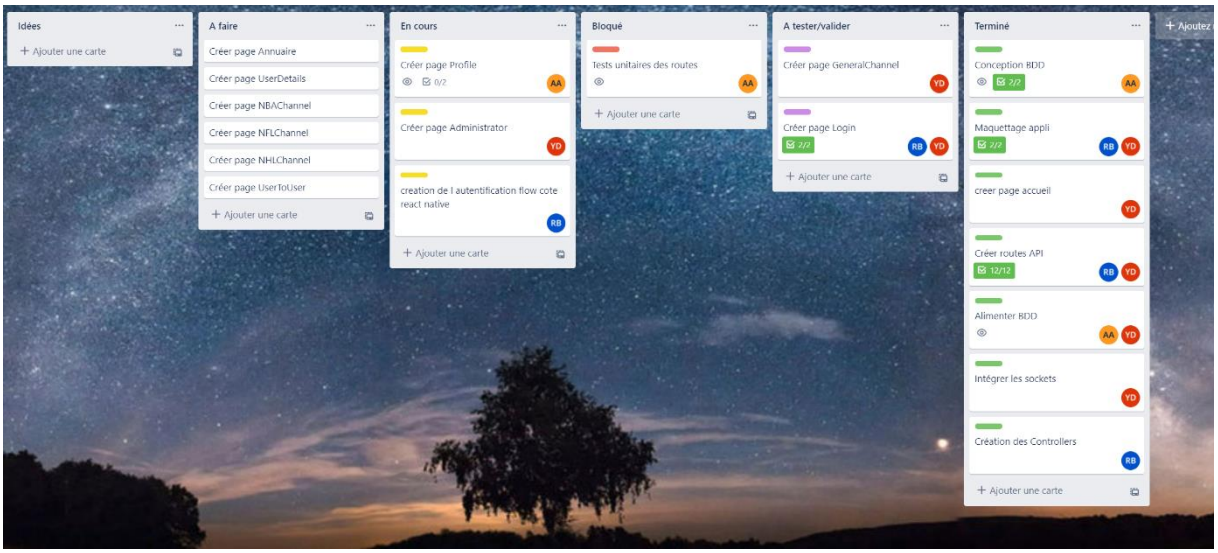
Organisation du projet.

Ce projet ayant été réalisé durant mon année de formation avec des temps en entreprise et d'autres projets à rendre, l'organisation de ce projet a été primordiale.

Pour commencer nous avons listé toutes les fonctionnalités de notre application puis nous avons réalisé un diagramme de Gantt après avoir réfléchi aux tâches globales à réaliser pour ce projet.



Le diagramme de Gantt nous a permis de décomposer les tâches globales pour avoir une vue d'ensemble sur le déroulé du projet ainsi que son avancée. Il nous a permis aussi de faire un Trello après décomposition des grosses tâches.



Architecture logicielle

Les utilisateurs qui ont un rôle « user » auront uniquement accès à l'application mobile. Les utilisateurs qui ont un rôle « admin » auront accès à la fois à l'application web et l'application mobile. Les deux applications utilisent la même base de données MySQL.

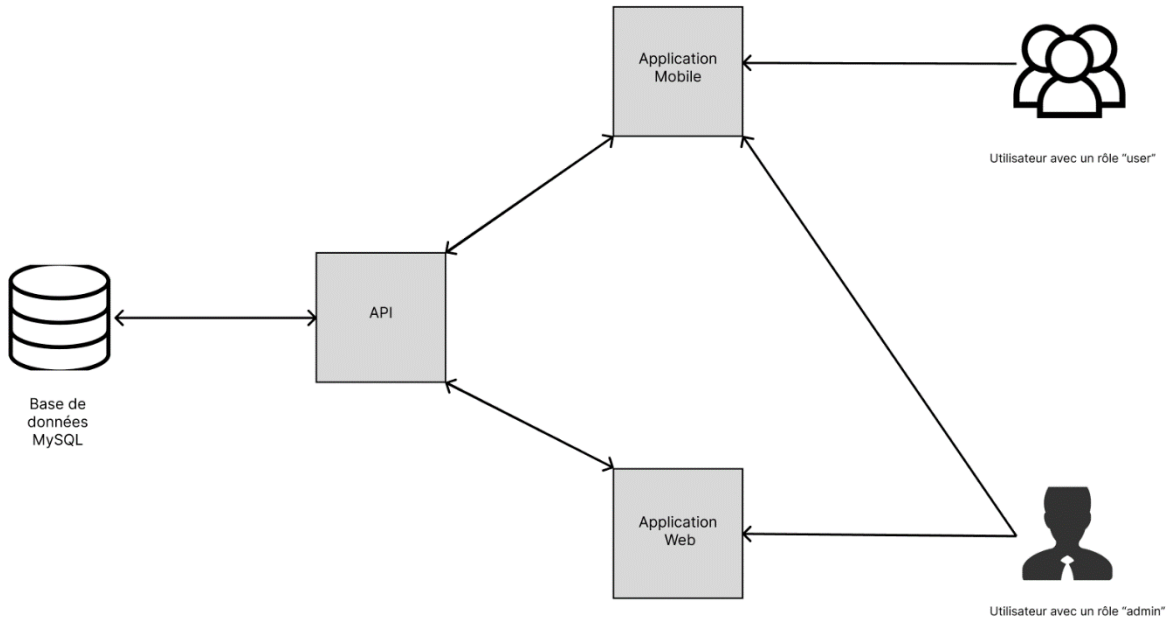
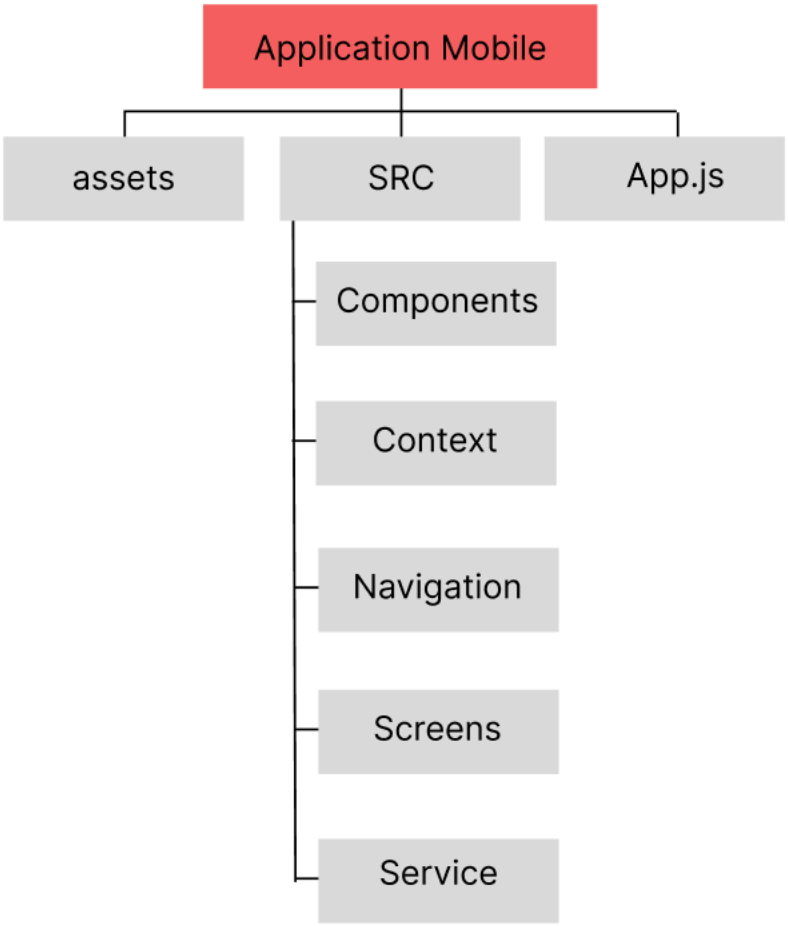


Schéma de l'architecture logicielle

Conception du front-end de l'application

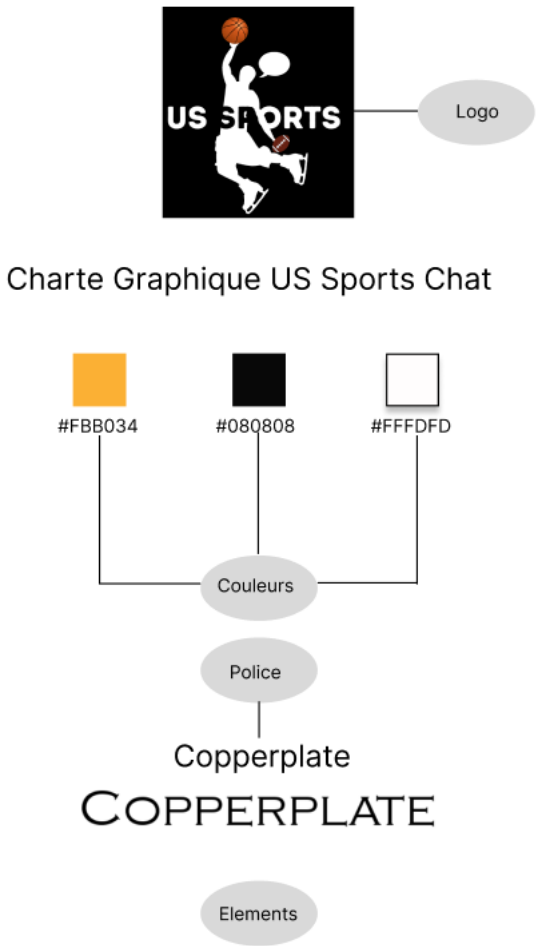
Pour concevoir ce projet et créer la maquette et la charte graphique j'ai décidé d'utiliser Figma. J'ai fait ce choix car j'ai l'habitude de travailler avec Figma puis il permet une collaboration en temps réel. De plus son interface est intuitive et facile à utiliser et le prototypage est puissant.

Arborescence du projet



Charte graphique

Après avoir fait l'arborescence du projet, je me suis attaqué à la création du logo. Ce projet étant lié avec un des sides projects d'un membre de notre équipe, Nous avions déjà une base de logo. Je suis ensuite passé à la charte graphique de l'application.



Maquettage

J’ai également utilisé Figma pour le maquettage Wireframe et Highrez. Je mets une capture d’écran de la page d’accueil et vous pouvez retrouver l’ensemble des maquettes en Annexe de ce dossier.

Conception Back-End de l'application

La base de donnée

Mise en place de la base de données.

Pour ce projet j'avais bien entendu besoin d'une base données qui permet de stocker les informations des utilisateurs ainsi que les messages.

Utilisant le framework Express.Js, j'ai appris que les bases de données les plus utilisées avec Node.Js sont les bases de données NoSQL.

Par soucis de temps et d'habitude de travail j'ai fait le choix d'utiliser une base de données relationnelle (SQL), ce qui m'a permis de perfectionner mes connaissances sur ce sujet.

Conception de la base de données.

Pour concevoir ma base de données, j'ai utilisé la méthode Merise pour plusieurs raisons. La première est que la méthode Merise permet une structuration du processus de conception.

Elle définit des étapes claires et un ensemble d'outils pour analyser les besoins, modéliser les données et concevoir la structure de la base de données. Cette approche permet d'organiser efficacement le processus de conception et de faciliter la compréhension et la communication entre les membres de l'équipe de développement.

De plus, la méthode Merise met l'accent sur l'analyse des besoins des utilisateurs et la modélisation des données en fonction de ces besoins. Elle favorise une approche centrée sur l'utilisateur, en veillant à ce que la structure de la base de données reflète fidèlement les processus métier et les règles de l'entreprise. Cela permet de concevoir une base de données qui répond spécifiquement aux exigences fonctionnelles et aux contraintes du système à développer.

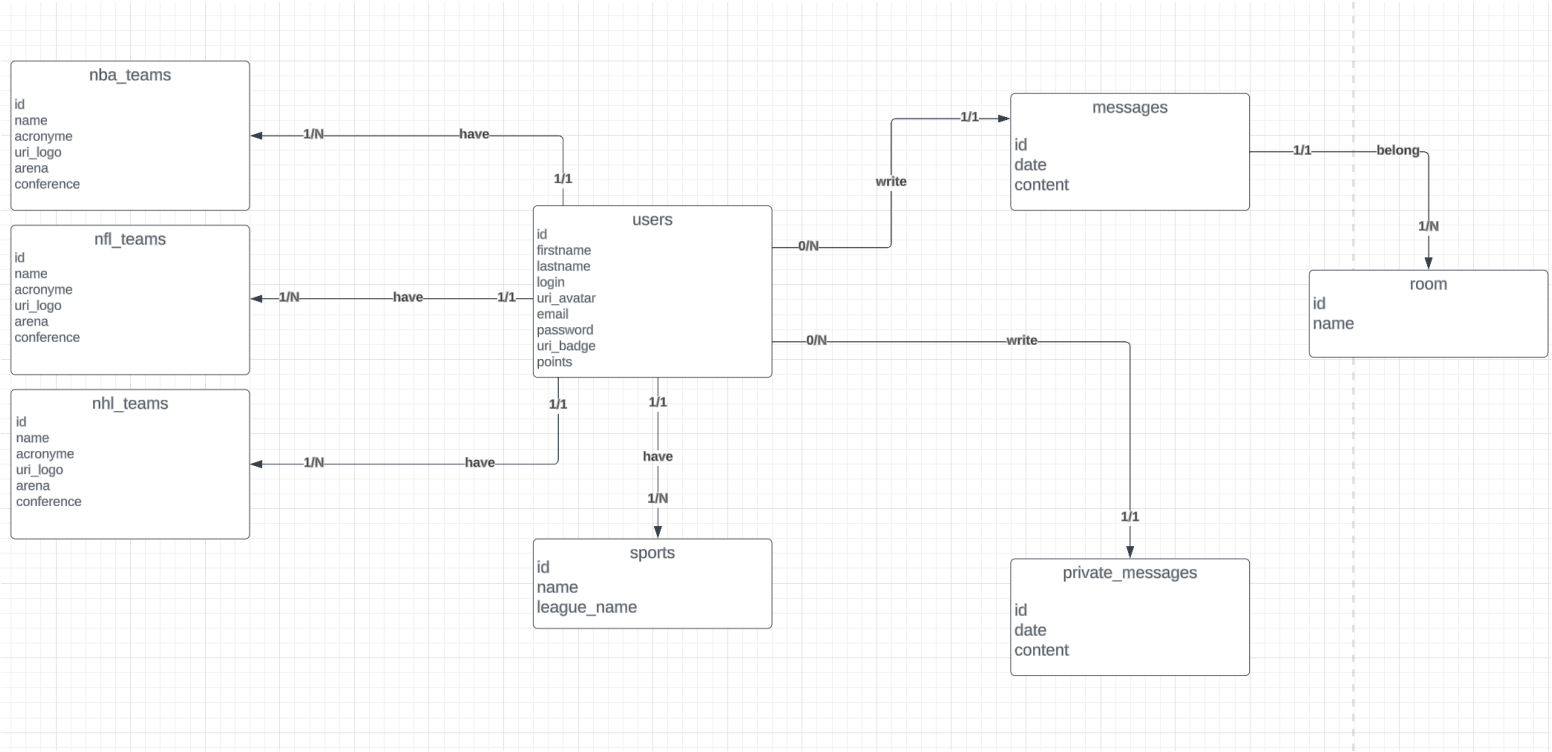
Avec son approche en plusieurs étapes elle distingue la modélisation conceptuelle (MCD), la modélisation logique (MLD) et la modélisation physique (MPD) ce qui permet une séparation claire entre la logique métier et l'implémentation technique, facilitant la maintenance et l'évolutivité de la base de données.

Dans un premier temps, j'ai réuni les données nécessaires dont voici un exemple.

- Pour les utilisateurs : j'ai besoin de stocker un login, un nom, un prénom, une date de naissance, un mot de passe, un rôle ainsi qu'un sport et une équipe favorite.
- Pour les messages : j'ai besoin de stocker l'heure, l'utilisateur qui a posté le message et le contenu du message.

Modèle Conceptuel de Données

Ensuite, en suivant la méthode Merise j'ai commencé à faire le MCD. J'ai crée des entités en fonction du dictionnaires de données récoltées et j'ai dessiné mes entités en leur donnant un nom.



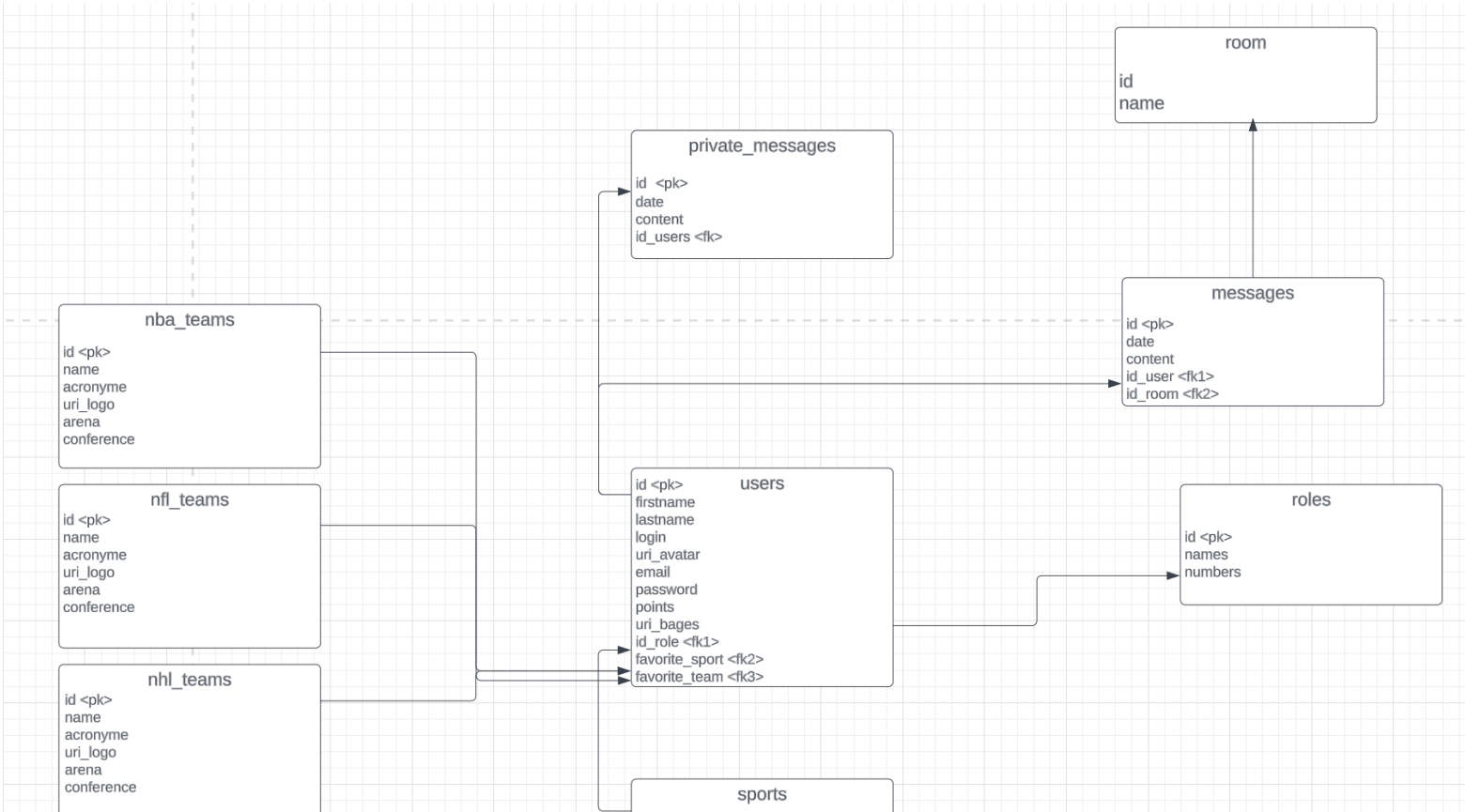
Modèle Logique de Données.

Pour poursuivre la conception de ma base de données et ayant mon MCD, j'ai fait le MLD.

J'ai donc créé une clé primaire pour chaque entité qui permet d'identifier sans ambiguïté chaque occurrence de cette entité.

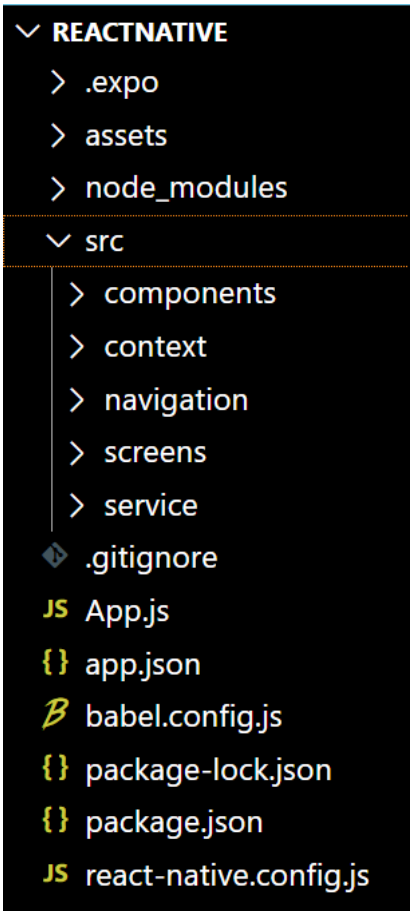
J'ai ensuite naturellement ajouté les attributs à chaque entité toujours en me basant sur le dictionnaires de données récoltées vu précédemment puis j'ai procédé à la création des cardinalités entre chaque entité.

Par exemple : 1 utilisateur peut envoyer 1 ou plusieurs (N) messages mais un message n'appartient qu'à 1 et 1 seul utilisateur.



Développement du Front-End de l’application.

Arborescence



Dans le dossier SRC du projet on retrouve tous les dossiers essentiels au projet, à savoir :

- Component : on y retrouve tous les composants réutilisables du projet.
- Context : on y retrouve le paramétrage qui nous permet d'utiliser des contextes tels que UseEffect par exemple.
- Navigation : on y retrouve le paramétrage de notre Drawer de navigation
- Screens : contient tous les écrans (pages) de l'applications.
- Service : contient les fichiers dans lesquels se trouve la logique métier.

Page et composants

Les composants jouent un rôle essentiel dans React et React Native en permettant de découper une page ou une interface utilisateur en éléments plus petits et autonomes, appelés composants. Ces composants sont conçus pour être réutilisables et peuvent être considérés comme des fonctions JavaScript. Ils prennent des entrées, appelées "props" (propriétés), et renvoient des éléments React qui décrivent ce qui doit être affiché à l'écran.

Le concept de composants dans React Native est de favoriser la réutilisation du code. Plutôt que d'écrire une logique ou une interface utilisateur spécifique à une seule partie de l'application, vous pouvez créer des composants réutilisables qui encapsulent cette logique et cette interface. Ces composants peuvent ensuite être utilisés dans différentes parties de l'application, ce qui permet d'économiser du temps et des efforts de développement.

En créant des composants réutilisables, vous pouvez développer un ensemble de blocs de construction qui peuvent être combinés pour créer des interfaces utilisateur complexes. Par exemple, vous pourriez créer un composant de bouton réutilisable avec des styles et des fonctionnalités spécifiques, puis l'utiliser dans différentes parties de votre application sans avoir à réécrire le code du bouton à chaque fois.

Cette approche de conception basée sur les composants permet de simplifier la maintenance et les mises à jour de l'application, car les modifications apportées à un composant se répercutent automatiquement sur toutes les instances où il est utilisé. Cela favorise également une meilleure organisation du code, une plus grande cohérence et une facilité de compréhension pour les développeurs travaillant sur le projet.

```
import React from 'react';
import {TouchableOpacity, Text} from 'react-native';
import Layouts from '../constants/Layout';
const Button = ({title, onPress}) => {


// console.warn("jesuis la ")

  return (
    <TouchableOpacity
      onPress={onPress}
      activeOpacity={0.7}
      style={{
        height: 55,
        width: '100%',
        backgroundColor: Layouts.blue,
        marginVertical: 20,
        justifyContent: 'center',
        alignItems: 'center',
      }}>
      {
      // console.warn("jesuis la aussi ")

}
      <Text style={{color: Layouts.white, fontWeight: 'bold', fontSize: 18}}>
        {title}
      </Text>
    </TouchableOpacity>
  );
};

export default Button;
```

Ci-dessus nous avons un composant qui nous permet de réutiliser le même bouton. Plusieurs boutons sont présents sur l'application et ce composant permet d’éviter la duplication de code et de gagner du temps de conception.

Chaque bouton sur le site correspond à des actions différentes mais sont composés des mêmes éléments : TouchableOpacity et Text.

Je peux donc faire appel à ce composant où je veux dans mon code et uniquement changer les props pour changer par exemple le style ou le nom du bouton.

Sécurité

Dans le développement du front-end de l'application, un aspect essentiel est la sécurité, qui vise à protéger les données sensibles et à prévenir les attaques potentielles. Pour garantir une sécurité front-end robuste, une bibliothèque populaire à considérer est "react-hook-form".

React Hook Form est une bibliothèque légère et performante qui facilite la gestion des formulaires dans les applications React. Elle offre une approche simplifiée pour la validation des données de formulaire, la gestion des erreurs et la collecte des valeurs saisies par l'utilisateur.

L'une des principales raisons de choisir react-hook-form pour renforcer la sécurité front-end est sa capacité à gérer la validation des données côté client. Avec cette bibliothèque, il est possible de définir des règles de validation personnalisées, telles que des validations de format d'e-mail, de mot de passe fort, ou de contraintes spécifiques à votre application. Cela permet de garantir que les données entrées par les utilisateurs sont conformes aux attentes, réduisant ainsi les risques d'injection de code malveillant ou de saisie de données incorrectes.

React-hook-form, permet également de gérer facilement les erreurs de saisie utilisateur. La bibliothèque fournit des mécanismes pour afficher des messages d'erreur dynamiques et des indications visuelles lorsqu'une validation échoue. Cela permet aux utilisateurs de comprendre rapidement les erreurs de saisie et d'effectuer les corrections nécessaires, améliorant ainsi l'expérience utilisateur globale.

Problématique rencontrée

Une des difficultés que j’ai rencontré durant le développement front-end de l’application à été d’imbriquer les navigations.

Mon souhait était d’avoir deux navigations distinctes pour l’application.

- Une dans le Drawer qui permet de naviguer dans toute l’application
- Une en bas de page qui permet de naviguer entre les conversations

Pour résoudre ce problème j’ai dû détailler comment fonctionne les différentes navigation puis j’ai utilisé la bibliothèque React Navigation.

Cette bibliothèque aide à faciliter la navigation entre les pages et le transfert de données entre celles-ci.

[Mettre screenshot ici]

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

[Mettre screenshot appli]

Exemple de formulaire de mise à jour du profil.

L'utilisateur peut modifier les informations de son profil. Par exemple il peut changer son login ou choisir un autre sport favori ou une autre équipe favorite

Voici les différentes étapes expliquées pour ce faire :

```
11  const EditProfilScreen = () => {
12    const [image, setImage] = useState(null);
13    const [login, setLogin] = useState('');
14    const [firstname, setFirstname] = useState('');
15    const [lastname, setLastname] = useState('');
16    const [email, setEmail] = useState('');
17    const [password, setPassword] = useState('');
18    const [passwordVisibility, setPasswordVisibility] = useState(true);
19    const [token, setToken] = useState('');
20    const [data, setData] = useState([]);
21
22    useEffect(() => {
23      const getInfo = async () => {
24        const res = await SecureStore.getItemAsync('access_token');
25        const decoded = jwt(res)
26        request('users/' + decoded.id, 'get', '')
27          .then(response => {
28            setFirstname(response.data.firstname)
29            setLastname(response.data.lastname)
30            setEmail(response.data.email)
31            setLogin(response.data.login)
32
33          })
34      }
35      getInfo()
36    }, []);
```

1) Tout d'abord on définit le composant qui est déclaré comme une fonction qui renvoie du JSX qui est une syntaxe similaire à du HTML. (Ligne 11)
2) On initialise les différents états à l'aide du hook « useState ». Ces états sont utilisés pour stocker les valeurs des champs de formulaire. (Ligne 12 -> 20)
3) On utilise le hook « useEffect » pour exécuter une fonction au chargement initial du composant. Il récupère les informations de l'utilisateurs à partir de SecureStore en utilisant un jeton d'accès (access_token), le décode à l'aide de jwt-decode puis utilise une requête HTTP avec la fonction request pour obtenir les détails de l'utilisateur à partir de l'API. Les détails récupérés sont ensuite utilisés pour initialiser les différents états.

```
80          const body = {
81            lastname: lastname,
82            firstname: firstname,
83            login: login,
84            email: email,
85            password: password
86          }
87          request('users/update' + decoded.id, 'patch', body)
88          .then(response => {
89            console.log(response.data)
90            setLogin(response.data.login)
91            setFirstname(response.data.firstname)
92            setLastname(response.data.lastname)
93            setEmail(response.data.email)
94            setPassword(response.data.password)
95          })
96          .catch(err => {
97            alert(err.response.data.message)
98            if(err.response.status == 401) {
99              console.log('Tu pues !')
100           }
101         })
102
103       } catch (error) {
104         console.error(error);
105       }
106     }
```

4) On envoie une requête http à l'API après avoir crée un objet avec les différentes valeurs insérées dans les champs qui ont été extraites des états correspondants.
5) On fait appel à 'request' avec les paramètres suivants : L'URL de l'API pour mettre à jour les informations de l'utilisateur, la méthode http Patch pour indiquer une mise à jour partielle des données et l'objet body contenant les nouvelles valeurs.
6) On gère la réponse de la requête http en utilisant une promesse 'then'. Lorsque la requête est réussie, la fonction de rappel reçoit la réponse de l'API dans le paramètre response. Ensuite, les états login, firstname, lastname, email, et password sont mis à jour avec les nouvelles valeurs reçues de la réponse. Cela permet de refléter les modifications dans l'interface utilisateur.

Développement du Back-End de l'application

Arborescence

Pour le développement du Back-End de l'application j'ai décidé d'adopter une architecture N-tier.

Le but de cette architecture est de séparer les différentes couches de l'application pour séparer la logique métier des routes.

Les différentes couches de l'application sont :

- Le routeur qui contient les routes.

- Le controller qui contient la logique.

- La couche data qui contient les modèles.

Mon Back-End est donc composé des dossiers suivants :

- Controllers qui regroupe les controller de l'application.

- Models qui regroupe tous les modèles des tables de la base de données.
- Routes qui regroupe tous les fichiers de routes de l'API (un par CRUD d'une table).

Fonctionnement de l'API

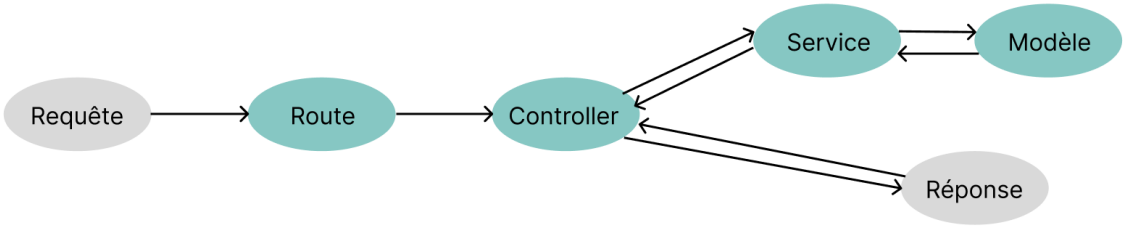
Lorsqu'un client envoie une requête à mon API, le routeur prend en charge cette requête et analyse l'URL. En fonction de la route spécifiée et de la méthode HTTP utilisée, le routeur détermine le contrôleur approprié à appeler. Le contrôleur est responsable de la gestion de la logique métier liée à la requête.

Le contrôleur interagit ensuite avec un service dédié, qui agit comme une couche intermédiaire entre le contrôleur et le modèle de données. Le service communique avec le modèle pour récupérer les données nécessaires à la requête.

Une fois les données récupérées, elles sont analysées et traitées par le service en appliquant les règles métier spécifiques. Le service prépare ensuite la réponse en format JSON, qui contient les données nécessaires, et associe un code de statut à cette réponse.

En résumé, le processus suit les étapes suivantes : analyse de l'URL par le routeur, appel du contrôleur approprié, communication avec le service pour interagir avec le modèle, récupération et traitement des données, préparation de la réponse au format JSON avec un code de statut.

Cette approche de développement basée sur un flux de travail en couches permet une séparation claire des responsabilités et favorise la modularité et la réutilisabilité du code. Elle facilite également la maintenance et l'évolution du système, car chaque composant est spécialisé dans sa tâche spécifique.



Architecture de l'API

Les différents statuts utilisés 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

- 404 : NOT FOUND

Indique que le serveur n'a pas trouvé la ressource demandée

500 : INTERNAL SERVER ERROR

Indique que le serveur a rencontré un problème.

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 fonction qui s'exécute entre la réception de la requête HTTP et l'envoi de la réponse. Il agit comme une couche intermédiaire qui peut effectuer des opérations de traitement ou de modification sur la requête et/ou la réponse avant de les transmettre à la prochaine fonction du pipeline.

Dans un projet React JS avec une API utilisant Express, les middlewares sont utilisés pour gérer diverses tâches telles que l'authentification, la gestion des erreurs, la journalisation des requêtes, la compression des données, etc. Ils offrent une flexibilité et une modularité accrues en permettant d'ajouter des fonctionnalités supplémentaires à l'application sans modifier directement les routes ou les contrôleurs.

Routage

Je me suis servi du routeur de Express.Js pour mettre en place mon routage.

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

Afin de facilement modifier et maintenir le code, j'ai décidé de séparer les routes et l'implémentation de celles-ci.

Donc, dans un fichier séparé que j'ai nommé server.js on peut retrouver ce code :

```
12  const routeRooms = require('./routes/rooms')
13  const routeSports = require('./routes/sports')
14  const routeMessages = require('./routes/messages')
15  const routeTeams = require('./routes/teams')
16  const routePriveM = require('./routes/privateMessage')
17  const routeLogin = require('./routes/auth')
18
19  const routeRole = require('./routes/roles')
20  app.use(cors({
21      origin: '*'
22  }));
23  app.use(bodyParser.json())
24
25  app.use('/rooms', routeRooms)
26  app.use('/sports', routeSports)
27  app.use('/messages', routeMessages)
28  app.use('/teams', routeTeams)
29  app.use('/privateMessages', routePriveM)
30  app.use('/roles', routeRole)
31  app.use('/auth', routeLogin)
32  app.use('/users', router)
```

Tout d'abord je stock les chemin d'accès aux routes dans des variables puis je les utilise avec le middleware « use » qui permet de spécifier le routeur qui doit être utilisé en fonction de la route appelée.

Par exemple, à la ligne 25, si l'URL '/rooms' est appelée, le routeur routeRooms est utilisé.

Pour expliquer en détail cette ligne :

- app = instance Express.
- use associe le routeur ou on retrouve les routes pour cette ressource.

La classe express.Router() est un middleware au niveau du routeur et est instanciée dans les différents routeurs.

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 méthode get , post, put, patch et delete afin de pouvoir effectuer

différentes actions en fonction de la méthode choisie.

```
33  let router = express.Router()
34
35  router.get('/', checkTokenexist, (req, res) => {
36      db.user.findAll()
37          .then(users => res.json({ data: users }), res.status(200))
38          .catch(err => res.status(500).json({ message: 'Database Error', error: err }))
39  });
```

Dans cet exemple, j'ai instancié la classe Router de express dans la variable router, puis grâce à cette classe, je fais appel aux différentes méthodes http dont j'aurais besoin.

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

Le premier paramètre de la fonction est l'URL, le deuxième est un middleware permettant la vérification de mon JWT token et le troisième est un gestionnaire qui est une fonction callback.

Controller

Dans les controllers on y trouve uniquement les tâches de validations comme par exemple la validation des droits. Aucune logique ne se trouve dans le controller .

Service

Cette couche est l'endroit où se trouve toute ma logique . L' objet request et response ne sont pas envoyés au service , seuls les paramètres de la requête sont envoyés après vérification du controller.

Model

Dans les modèles on retrouve toutes les informations des tables de la base de données.

Dans un fichier à part qui se donne db.config.js nous retrouvons la connexion à la base de données ainsi que la mise en place des relations entre les tables.

```
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, process.env.DB_USER, process.env.DB_PASS, {
9           host: process.env.DB_HOST,
10          port: process.env.DB_PORT,
11          dialect: 'mysql',
12          logging: false
13      }
14  )
15
16  /*** Mise en place des relations */
17  const db = {}
18
19  db.sequelize = sequelize
```

Tout d'abord, le code importe la classe Sequelize à partir du module 'sequelize'. Cela permet d'utiliser les fonctionnalités de Sequelize pour interagir avec la base de données.

Ensuite, un objet Sequelize est créé en utilisant les informations de connexion fournies par les variables d'environnement (process.env). Les informations de connexion incluent le nom de la base de données (DB_NAME), le nom d'utilisateur (DB_USER), le mot de passe (DB_PASS), l'hôte (DB_HOST), le port (DB_PORT) et le dialecte de la base de données (mysql dans cet exemple). La propriété logging est définie sur false pour désactiver l'affichage des journaux de requêtes dans la console.

Ensuite, des relations entre les tables de la base de données sont établies en créant des modèles Sequelize pour chaque table. Les modèles sont importés à partir de fichiers spécifiques (user.js, messages.js, teams.js, etc.) qui définissent la structure et les associations des tables. Les modèles sont associés à l'instance de Sequelize en passant l'objet sequelize créé précédemment.

Enfin, un objet db est créé et exporté. Cet objet contient l'instance de Sequelize (sequelize) ainsi que les modèles associés aux différentes tables de la base de données (user, message, teams, etc.). Cela permet d'accéder facilement à l'instance Sequelize et aux modèles dans d'autres parties de l'application.

En résumé, ce code configure la connexion à la base de données et établit les relations entre les tables en utilisant Sequelize. Il crée également un objet db contenant l'instance Sequelize et les modèles, ce qui facilite l'accès et la manipulation des données de la base de données dans le reste de l'application.

J’ai utilisé Sequelize car c’est un ORM (Object-Relational Mapping) populaire et largement utilisé dans l'écosystème JavaScript. Il offre une abstraction puissante pour interagir avec la base de données et facilite le développement d'applications robustes et évolutives.

Exemple de relation entre tables :

```
34   // 5 relation private-message et  user
35   db.user.hasMany(db.privateMessage, {foreignKey: 'user_id', onDelete: 'cascade'})
36   db.privateMessage.belongsTo(db.user, {foreignKey: 'user_id'})
```

La ligne 35 définit une relation "un-à-plusieurs" entre la table "user" et la table "privateMessage". Cela signifie qu'un utilisateur peut avoir plusieurs messages privés. La clé étrangère utilisée pour établir cette relation est "user_id" dans la table "privateMessage". L'option {onDelete: 'cascade'} spécifie que lorsque l’on supprime un utilisateur, tous ses messages privés associés seront également supprimés.

La ligne 36 définit une relation "plusieurs-à-un" entre la table "privateMessage" et la table "user". Cela signifie que chaque message privé appartient à un seul utilisateur. La clé étrangère utilisée pour établir cette relation est également "user_id" dans la table "privateMessage".

En définissant ces relations, on peut utiliser les méthodes Sequelize pour accéder et manipuler les données associées entre ces tables. Par exemple, si on a un objet "user" récupéré de la base de données, on peut accéder à tous ses messages privés en utilisant la bonne méthode.

Ces définitions de relations aident à structurer et à organiser les données dans votre base de données relationnelle, en facilitant la récupération et la manipulation des données liées entre les tables.

Sécurité

Chiffrement des données sensibles

J’ai utilisé Bcrypt pour chiffre les mots de passes des utilisateurs en base de données de manière à ce qu’ils soient illisibles en base lors du stockage.

Le facteur qui m’a fait choisir Bcrypt réside dans sa fonction de hachage adaptative. Elle utilise un algorithme appelé Blowfish pour effectuer le hachage, et il est possible de contrôler le nombre de tours de l'algorithme à effectuer, ce qui permet d'augmenter la complexité de calcul du hachage et de rendre les attaques par force brute plus difficiles et plus lentes.

```
87          let hash = await bcrypt.hash(password, parseInt(process.env.BCRYPT_SALT_ROUND))
88          req.body.password = hash
```

La ligne 87 utilise la fonction bcrypt.hash() pour générer un hachage du mot de passe. La fonction bcrypt.hash() prend deux arguments :

- le mot de passe à hacher et le nombre de rounds de hachage à effectuer.
- parseInt(process.env.BCRYPT_SALT_ROUND), indique le nombre de tours de hachage à effectuer. Ce nombre de tours détermine la complexité de calcul du hachage et peut être configuré en fonction des besoins de sécurité de l'application. La valeur est généralement stockée dans une variable d'environnement (process.env) pour permettre une flexibilité dans la configuration.

Une fois que le hachage est généré, il est stocké dans la variable hash. Le hachage résultant est une chaîne de caractères sécurisée et unique qui représente le mot de passe d'origine.

A la ligne 88 le mot de passe d'origine dans l'objet req.body (correspondant aux données envoyées par la requête) est remplacé par le hachage sécurisé. Cela permet de stocker le hachage du mot de passe dans la base de données plutôt que le mot de passe en clair, renforçant ainsi la sécurité des données utilisateur.

JWT

Dans le cadre de mon projet, j'ai mis en place une authentification sécurisée et sans état en utilisant JSON Web Token (JWT).

Le JWT est un type de jeton qui permet l'échange d'informations sur l'utilisateur de manière sécurisée. Il sert de méthode de communication entre deux parties. Le jeton est composé de trois parties distinctes :

- L'en-tête (header) : Il identifie l'algorithme utilisé pour générer la signature du jeton.
- La charge utile (payload) : Cette partie contient les informations de l'utilisateur, encodées en base64. Cependant, par souci de sécurité, je veille à ne pas inclure de données sensibles telles que des mots de passe ou des informations personnellement identifiables.
- La signature : Elle est créée à partir de l'en-tête et de la charge utile, en utilisant un secret. Une signature invalide entraîne automatiquement le rejet du jeton. La signature joue un rôle crucial car elle permet de vérifier l'intégrité des informations contenues dans le jeton.

Lorsqu'un utilisateur tente de se connecter à son espace, une requête est envoyée au serveur. Si les informations fournies sont correctes, le serveur renvoie une réponse au format JSON contenant le jeton. Ce jeton contient des informations sur l'utilisateur connecté, telles que son identifiant, son adresse e-mail et son rôle.

Par la suite, le client enverra ce jeton avec chaque requête ultérieure. Ainsi, le serveur n'a pas besoin de stocker d'informations sur la session de l'utilisateur. L'utilisation du jeton permet de maintenir l'état de l'authentification sans avoir recours au stockage de session côté serveur.

En résumé, j'ai implémenté l'authentification sécurisée et sans état en utilisant JSON Web Token (JWT). Le jeton contient les informations de l'utilisateur, est signé et vérifié à chaque demande, permettant ainsi de maintenir l'authentification tout en évitant le stockage d'informations de session côté serveur.

Exemple de problématique rencontrée