Dossier de projet professionnel

Chappy

Réalisation de l’application mobile et du client admin web

Créé par Criss Brierre

Présenté par Criss Brierre

Introduction

Présentation personnelle

Je m'appelle Criss Brierre, j'ai 21 ans. Passionné par l'informatique, j'ai décidé de m'orienter dans la programmation à la sortie du BAC pour effectuer un BTS Service informatique aux organisations. A l'aboutissement de celui-ci, j'ai décidé de faire un Bachelor IT - Web à l'école La Plateforme en alternance afin de me professionnaliser dans le développement Web et Mobile.

Présentation du projet en anglais

Today, there are a huge number of chat apps in the mobile ecosystem, but none of them have based their design on preserving a healthy, insult-free environment. chappy meets this need by incorporating an innovative concept: turning insults into "nice" words. This enables the application to reach a wider audience and, above all, transforms a constraint into a fun concept! The application allows users to create, modify and delete their profile, consult the list of connected users and send private and public messages in a general chat room accessible to all. There's also an admin web client enabling any admin to consult the list of users, modify/delete their profiles and similarly with private and public messages.

Compétences couvertes par le projet

Ce projet couvre les compétences du titre suivantes :
- Maquetter une application
- Développer des composants d'accès aux données
- Développer la partie front-end d'une interface utilisateur web
- Développer la partie back-end d'une interface utilisateur web
- Concevoir une base de données
- Mettre en place une base de données
- Développer des composants dans le langage d'une base de données
- Concevoir une application
- Développer des composants métier
- Construire une application organisée en couches

- Développer une application mobile

Organisation et cahier des charges

Analyse de l’existant

Comme dit précédemment dans la présentation en anglais, il existe énormément d'applications de chat sur mobile, pour ne pas les citer : WhatsApp, Facebook Messenger,Telegram,Signal,WeChat.. L’objectif principal de cette application est donc de se démarquer de celles-cis en y incorporant un concept innovant : le filtrage d’insulte avec des mots “gentils”. La modération des chats est une grosse contrainte pour les applications de chat traditionnelles, jouer avec celle-ci pour y proposer un concept fun permet donc de se démarquer.

Les utilisateurs du projet

Ce projet se décompose en deux parties : une application mobile et une interface web.

La partie application mobile qui sera utilisée par les utilisateurs qui pourront ajouter des messages dans le salon public et en message privés aux autres utilisateurs et les administrateurs qui eux auront la main sur la modification des profils des utilisateurs et pourront modérer le chat.

La partie site web est utilisée uniquement par les personnes ayant des droits administrateur. Sur cette espace , les administrateurs pourront avoir la liste des utilisateurs, chaques profils pourront être ajouter / modifier / supprimer par les administrateur et pareillement pour les messages privées / publiques.

Les fonctionnalités attendues

Application mobile

Page d’authentification :
Lorsque l’utilisateur lance l’application mobile , il est redirigé vers une page d’authentification sur laquelle il doit se connecter avec l’email et le mot de passe, il a donc la possibilité de se connecter ou alors de créer un compte.

Page d’accueil:
Quand l’utilisateur est authentifié, il arrive sur la page d’accueil lui proposant deux boutons, un bouton “chat” lui permettant d'accéder au chat général de l’application et le bouton “users” qui permet d'accéder à la page avec la liste des utilisateurs de l’application.

Page chat:
La page chat est le chat principal de l’application, chaque utilisateur de l'application est libre de l’utiliser et le filtre anti insulte est activé.

Page users :
La page users permet de consulter tous les utilisateurs de l’application, on y voit leurs états grâce à la colorimétrie de l’avatar (vert : connecté, rouge : déconnecté).
l’utilisateur peut rechercher un user grâce à la barre de recherche et il peut consulter le profil d’une personne.

Page consulter profil:
Accessible depuis la page users, elle permet d’afficher les informations du profil d’une personne (modifiable par l’admin), mais aussi de pouvoir afficher ses propres informations, qui sont modifiables.

Page option:
La page option, qui permet de se déconnecter, mais aussi de donner l’accès à la page de modification de son profil.

Contexte technique

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

Conception du projet

Choix de développement

Choix de langage

L’unique langage de programmation utilisé dans ce projet est le javascript qui couvre le back-end comme le front end. Avec pour environnement de dev node.js.

Nous avons choisi JavaScript car c’est l'un des langages de programmation les plus populaires, il n’a pas quitté le top 3 des langages les plus utilisés depuis sa création.

Les langages populaires ont une grande communauté de développeurs et de ressources disponibles pour nous accompagner.

Javascript est utilisé par de nombreux outils à la fois serveur, web et mobile. Nous pourrons donc centraliser la stack de langage utilisé pour l’ensemble du projet.
Enfin ce choix est aussi personnel car au vue de la demande dans ce langage, mon binôme et moi voulions monter en compétences grâce aux framework utilisées avec celui-ci.

Choix des frameworks :

Définition d’un framework :

Un framework est un ensemble d'outils, de bibliothèques, de conventions et de bonnes pratiques préétablies qui fournissent une structure et un cadre de développement pour faciliter la création d'applications logicielles. Il fournit une base sur laquelle les développeurs peuvent construire et organiser leur code de manière cohérente, réutilisable et efficace.

Back-end :

Pour la création de l'api qui sera consommée par l'application mobile et web, nous avons fait le choix d'utiliser **Express.js**

Express.js bénéficie d'une vaste communauté de développeurs, ce qui signifie qu'il existe de nombreuses ressources, tutoriels et modules complémentaires disponibles pour étendre les fonctionnalités de l'API. L'écosystème d'Express.js est bien établi et offre des solutions pour la gestion des bases de données, l'authentification, la validation des données, etc.
Express.js est conçu pour être évolutif et peut gérer efficacement des applications avec une charge de trafic élevée. Il offre des options de mise en cache, de gestion des sessions et de clustering pour améliorer les performances de l'API et permettre une évolutivité horizontale.

Nous avons utilisé un ORM nommé Sequelize.

Définition d'un ORM :

Un ORM (Object-Relational Mapping) permet de mapper les objets d'une application à des tables dans une base de données relationnelle. Il agit comme une couche d'abstraction entre l'application et la base de données, en permettant aux développeurs de manipuler les données en utilisant des objets.

Nous l'avons utilisé car ça permet d'avoir une logique générique pour la persistance et l'accès aux données ce qui assure une compatibilité avec toutes les plateformes de données, il est facile d'utilisation et fournit un système de validation de données avant de les persister en base.

dialectes supporté par sequelize : **mariadb, mysql, postgres, sqlite, mssql, snowflake, oracle**

Front-end :

Pour la création de mon application mobile, j'ai choisi le framework react native car il est écrit en javascript et il permet de déployer l'application pour IOS et pour Android, le fait de connaître aussi react.js en web nous a conforté à choisir ce framework.

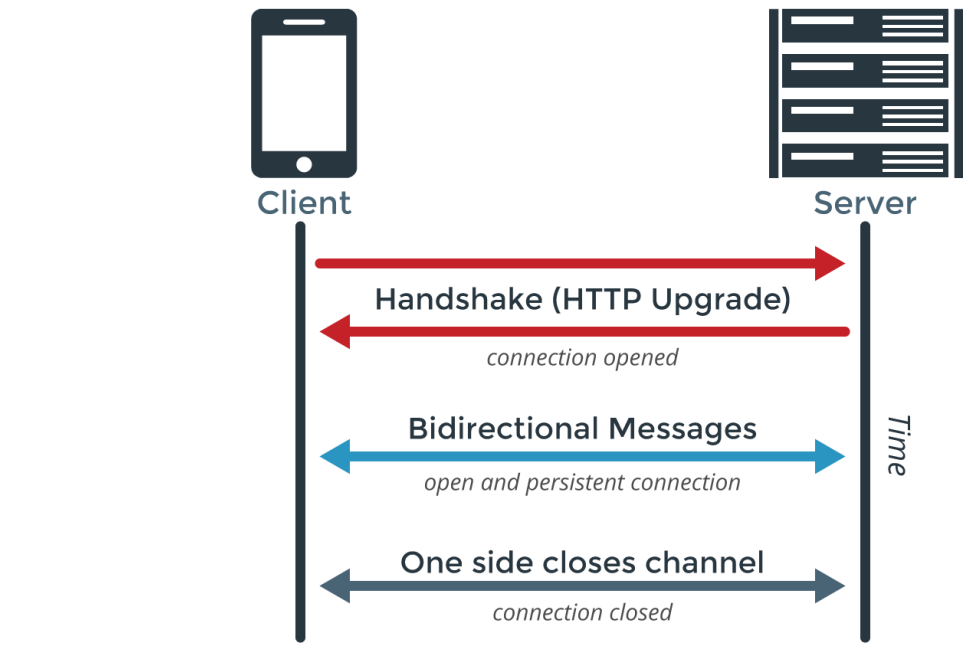
Et donc naturellement, on s'est aussi tourné vers react.js pour le client admin en web.

Logiciels et autres outils

- Visual Studio Code comme éditeur de code;
- Thunder client pour effectuer les requêtes API;
- Git et GitHub pour le versionning de mon code;
- Trello pour organiser mon travail;
- NPM pour installer les paquets;
- Figma pour la création de mes maquettes;
- LucidChart pour le maquettage de ma base de données;
- Expo pour émuler mon application mobile sur mon téléphone;

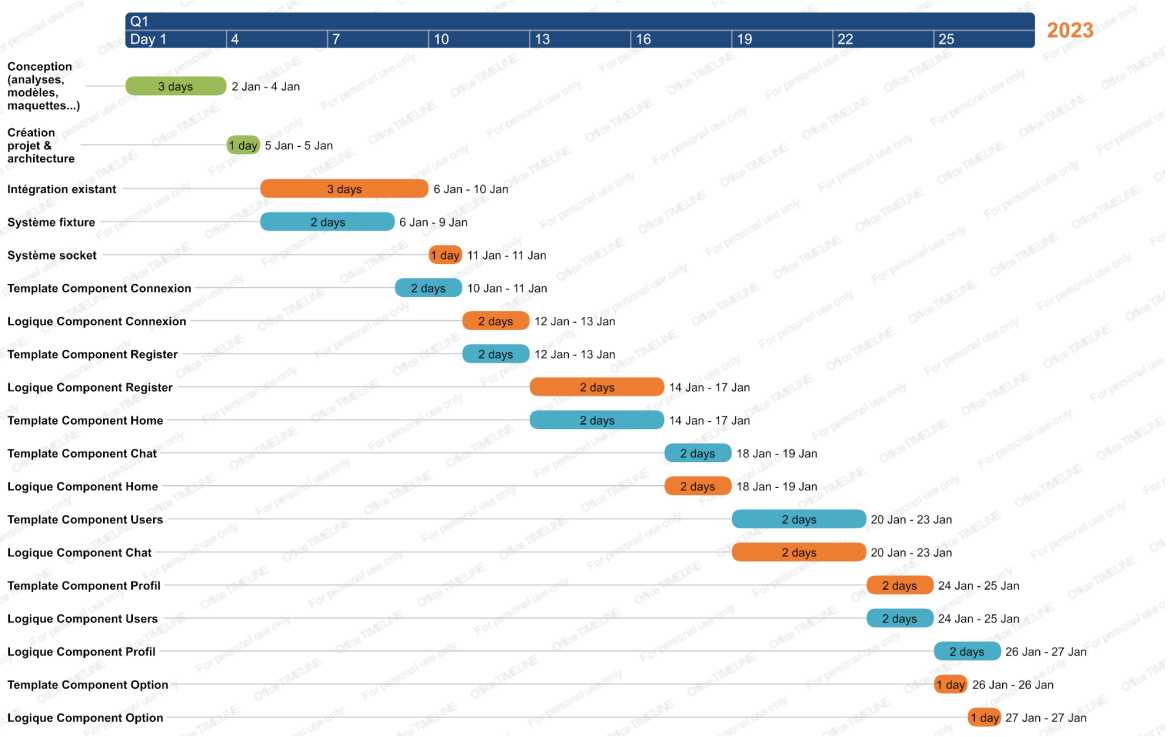
Socket

Les websockets permettent une communication bidirectionnelle en temps réel entre un client et un serveur. Initialement, le client envoie une demande de connexion WebSocket au serveur via une requête HTTP. Si la connexion est établie, le protocole passe du mode HTTP au mode WebSocket, permettant l'échange continu de messages. Les deux parties peuvent alors envoyer des données sous forme de messages sans avoir à envoyer de nouvelles requêtes, offrant une communication instantanée et efficace.

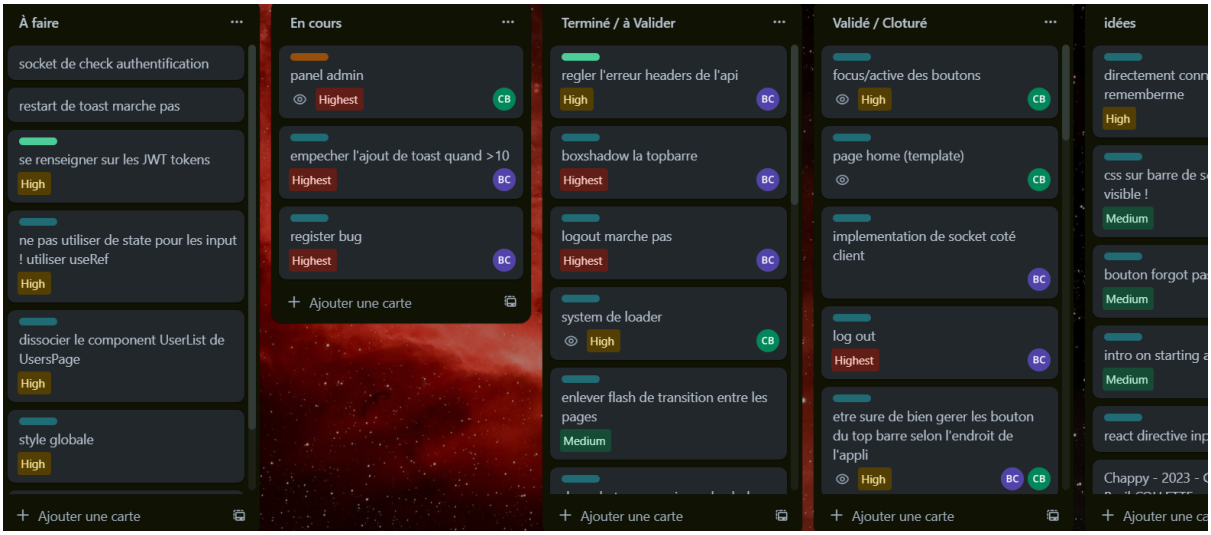


Organisation du projet

//parler de gantt

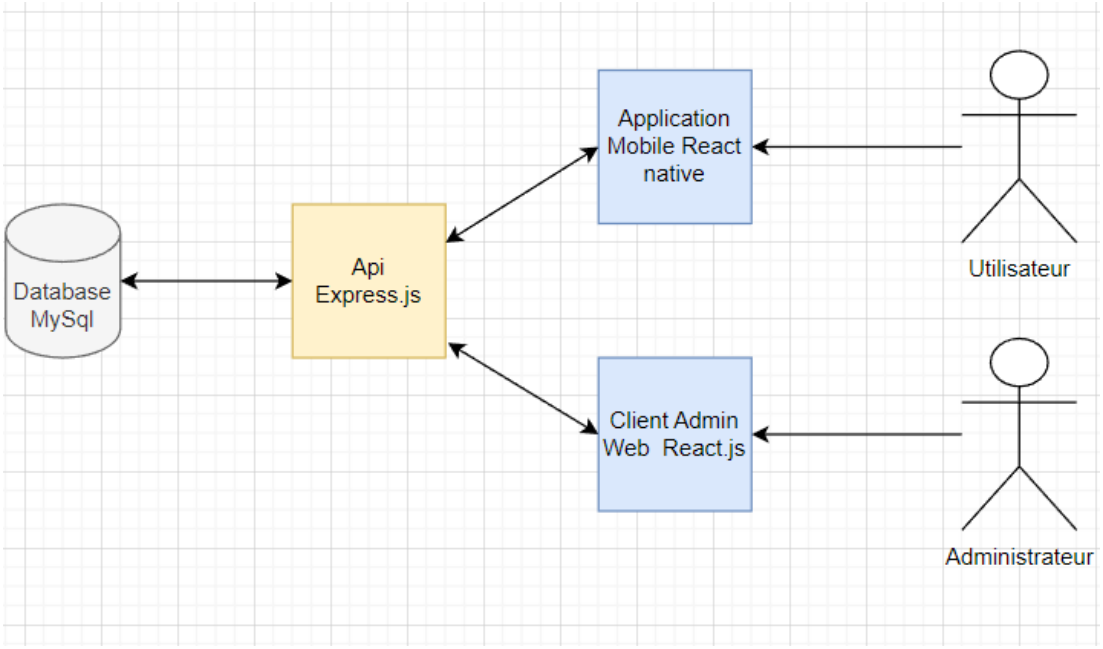


Avoir une bonne organisation de projet est primordial pour pouvoir mener son projet à bien. Moi et mon binôme avons commencé par lister toutes les tâches avec une deadline fixe, pour cela nous avons utilisé l'outil de gestion de projet en ligne Trello.



Nous avons choisis de ce basé à 100% sur le sujet demandé par notre centre de formation car c'était pour nous notre cahier des charges, nous y avons juste ajouté notre idée de filtrage d'insulte en plus afin d'avoir un concept concret d'application.

Architecture Logiciel

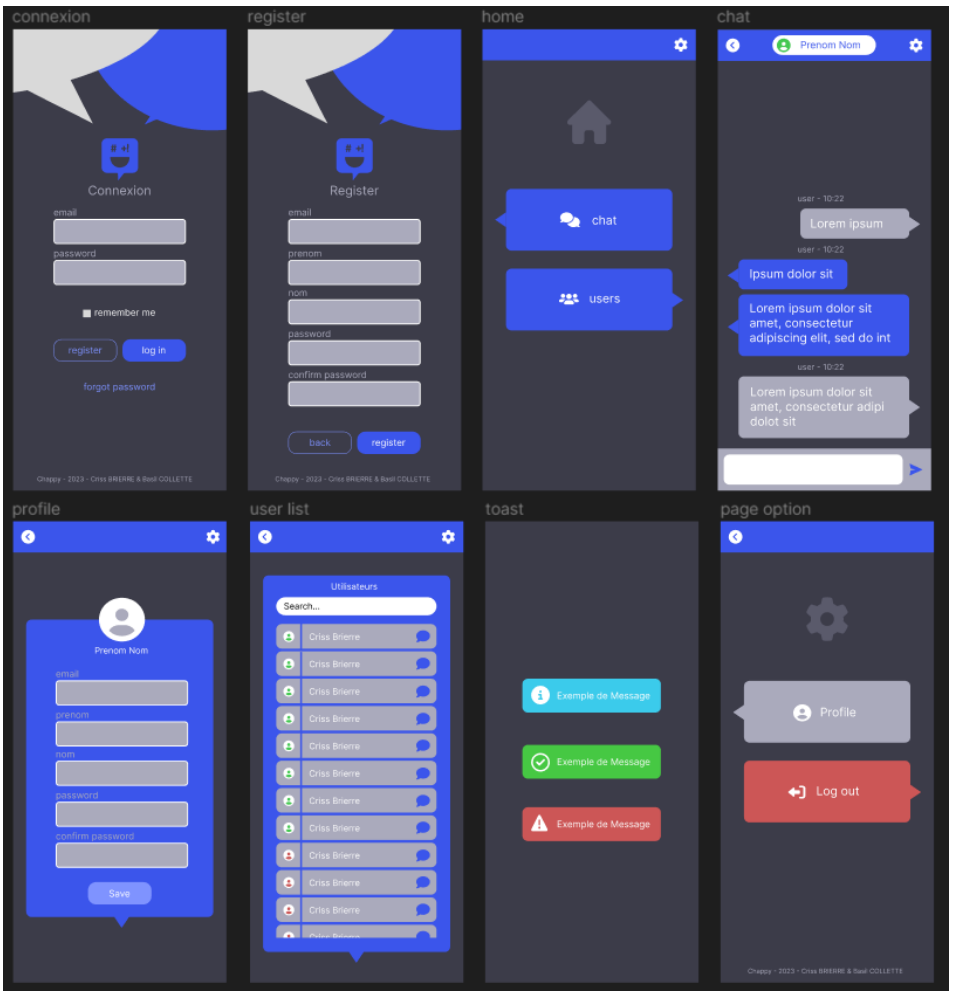


Conception de l'application

Front-end

Maquettage

Le maquettage a été réalisé avec Figma qui est éditeur de graphiques vectoriels et un outil de prototypage



Charte graphique

Back end

Mise en place de la base de données

il est indispensable d'avoir un moyen de persister les données dans ce projet et donc il faut faire le choix d'une base de données, nous avons réfléchis à quel SGBD utiliser.

Base de données relationnelle :

Une base de données relationnelle présente divers avantages. Elle offre une structure organisée basée sur des tables et des relations claires entre elles, facilitant la gestion des données. Les contraintes d'intégrité garantissent la cohérence et la validité des données.

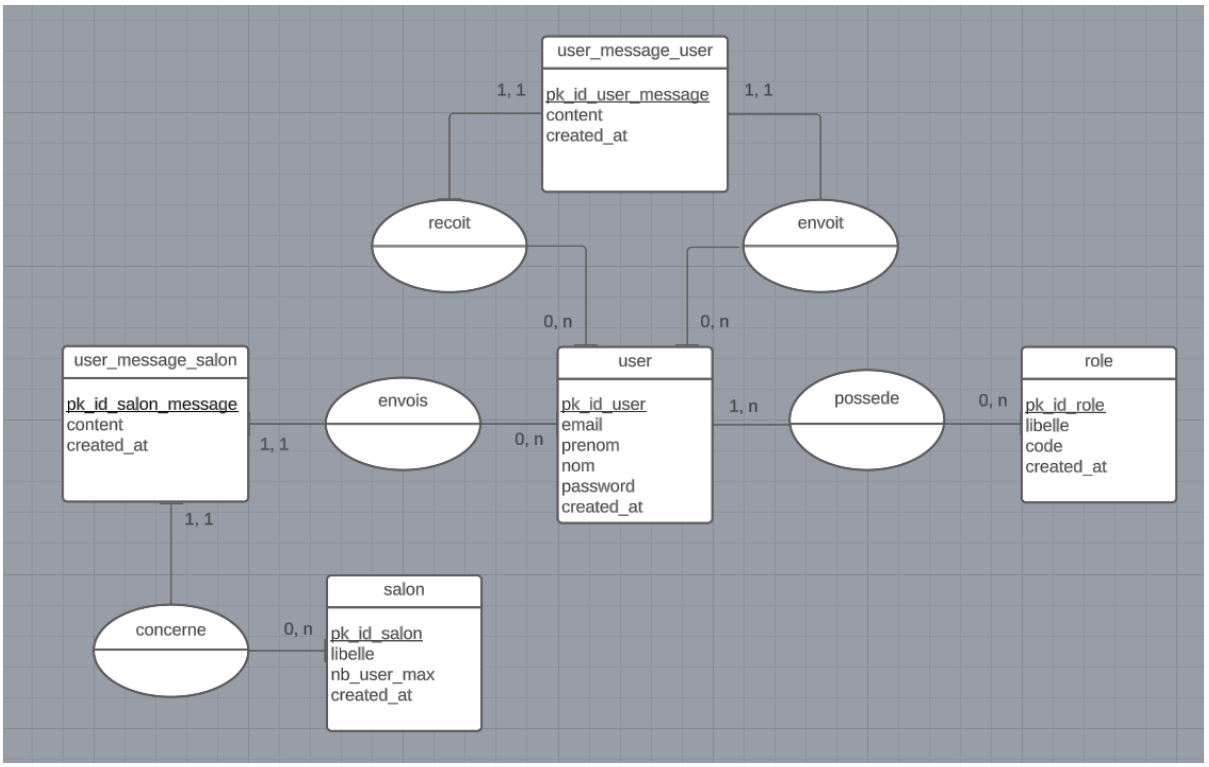
Les relations entre les tables maintiennent la cohérence des données. Elles facilitent également l'intégration et la jointure de données provenant de différentes sources. Les SGBD assurent la durabilité des données et permettent le contrôle de la redondance grâce aux contraintes d'unicité .

Ces avantages correspondent exactement au besoin dans notre contexte d'application de messagerie dans laquelle par exemple les messages sont liés aux utilisateurs .

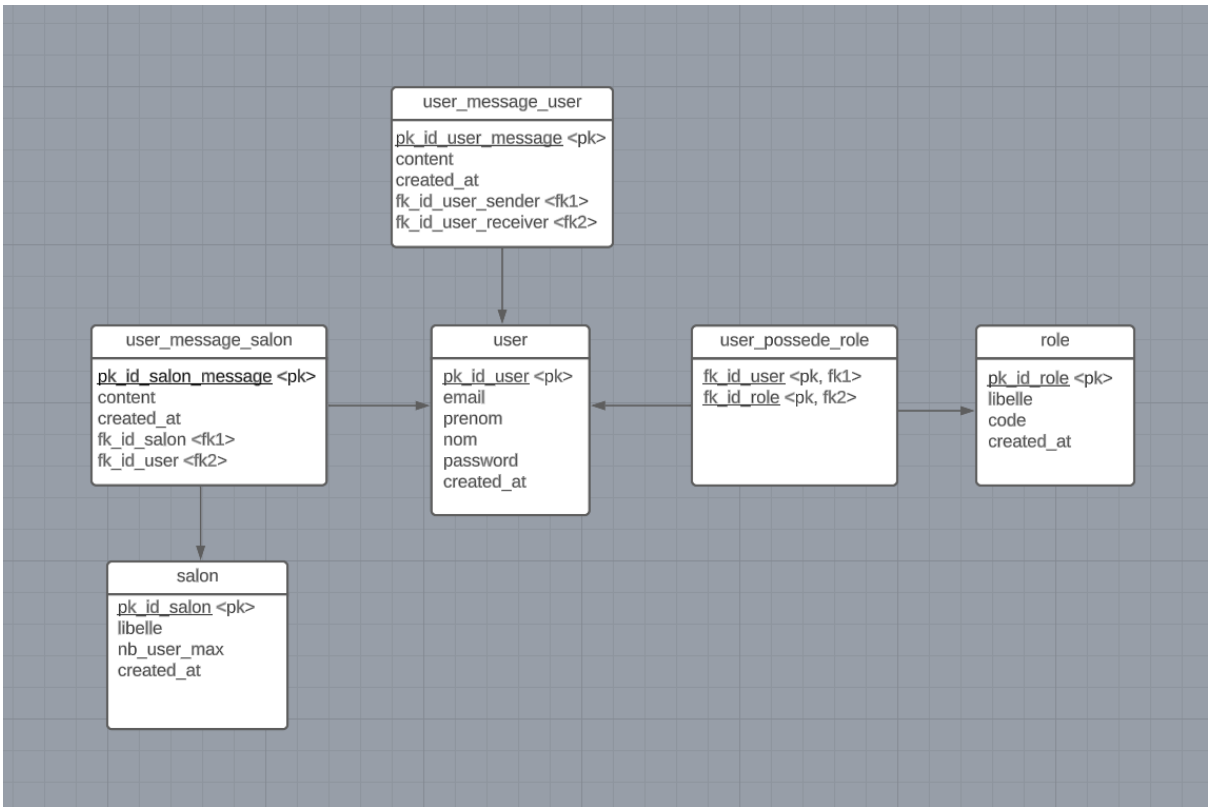
Conception de la base de données :

Pour définir les tables ainsi que leurs relations dans la base de données, il faut faire ce qu'on appelle un MCD (**Modèle Conceptuel de données**) qui va grâce à aux cardinalités définir le type de relations entres-elles, ensuite il faut faire un MLD (**Modèle logique de données**) qui va concrétiser cela en définissant les tables de la base de données.

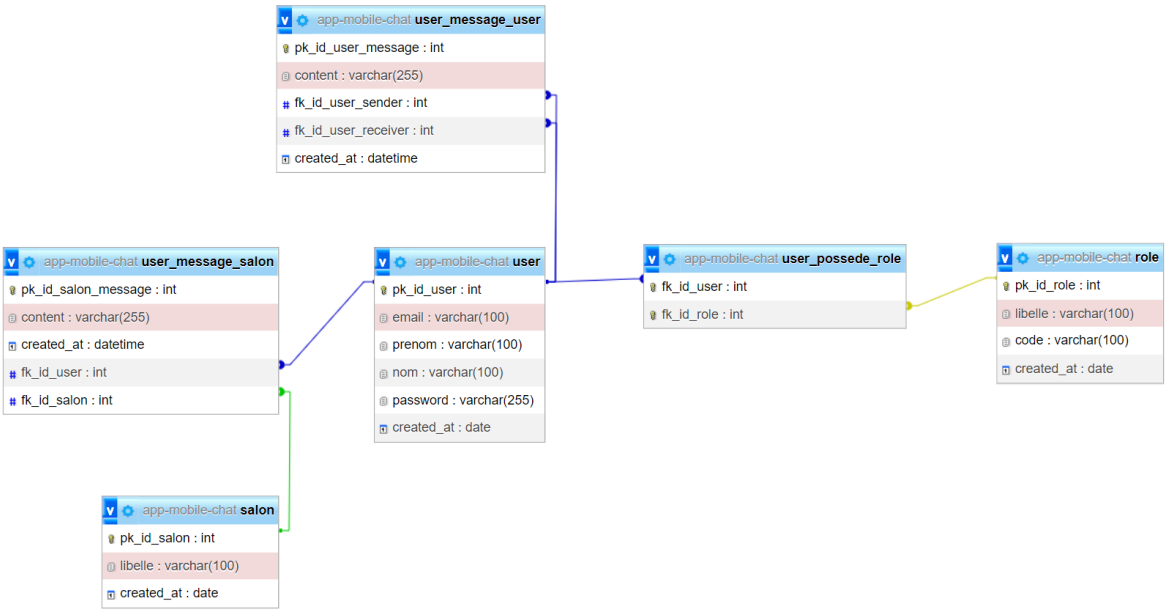
Modèle conceptuel de données



Modèle logique de données



Modèle physique de données

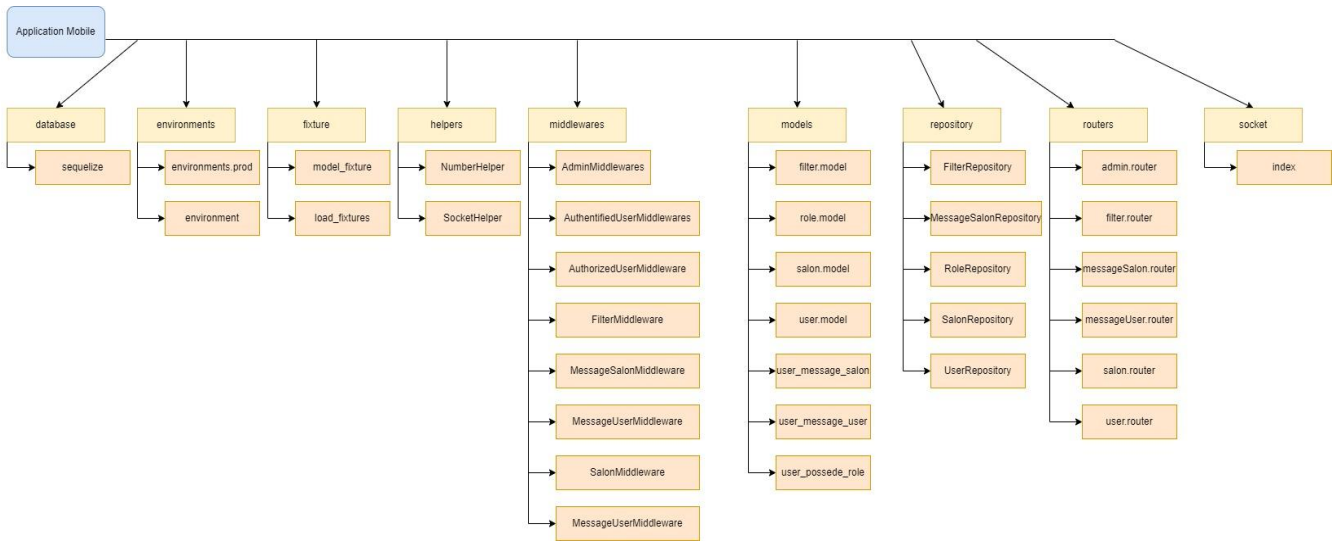


Développement du back end de l’application

Organisation

Pour garantir une utilisation efficace de mon backend à la fois pour mon application web et mobile, j'ai opté pour la création d'une API afin d'éviter la duplication de la logique métier. J'ai divisé ma logique en modules distincts, ce qui a amélioré la lisibilité du code et facilite sa maintenance pour les futures versions. J'ai également évité les redondances en créant des fonctions et des services. Ainsi, la structure de mon code est conçue pour être facilement maintenable et réutilisable.

Arborescence



J’ai suivi une architecture 3-tier. Le principe de cette architecture est la séparation des préoccupations pour éloigner la logique métier des routes de l’API.

Les différentes couches de l’application sont séparées :
- Le routeur,
- Middleware qui gère aussi la couche métier de l’application
- La couche data (les modèles, et repository).

Mon back end est donc composer des dossiers suivants :

- database: permet d'accéder au fichier database.js qui gère la connexion à la base de données, pour éviter des surcharges de mémoire nous avons décider

d’y incorporer à celui-ci un Singleton qui permet de créer une seul instance de connexion
- fixture: permet de créer les fixtures de l’application en utilisant sequelize donc en accord avec model.
- repository : permet d’accéder et d'insérer des données en base.
- helpers : permet d’accéder au fichier de helper qui héberge de la logique utilisé à divers endroit de l’application, donc ce fichier centralise de la logique utilisé plusieurs fois.
- Middleware : contient les fichiers avec les middlewares des routes, les middlewares contient aussi la logique métier de l’application.
- modèle : contenant les modèles de toutes les tables.
- routes : contenant toutes les routes de l’application.
- socket : gère le socket de l’application en y faisant la connexion et en y définissant des events sockets.

Fonctionnement de l’api

Lorsque le client envoie une requête sur mon API, le fichier server.js analyse l’url et renvoie sur le router correspondant,ces routes font appeler des middlewares qui vont exécuter la logique métier qui elle même appelle les repositories pour pouvoir insérer modifier et accéder à la données, les middlewares contiennent aussi un bloc try catch qui permet de renvoyer une réponse en fonction de l’état de la requête, cette réponse est au format JSON .

Les différents statuts utilisés dans se projets sont :

Les différents statuts utilisés dans se projets sont :
- 200 : OK
Indique que la requête a réussi
- 201 : CREATED
Indique que la requête a réussi et une ressource a été créé
- 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
- 406 : METHOD NOT ALLOWED
Indique que la requête est connue du serveur mais n'est pas prise en charge

pour la ressource cible
- 500 : INTERNAL SERVER ERROR
Indique que le serveur a rencontré un problème.

Les différentes méthodes HTTP utilisées 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
- DELETE - Pour supprimer une donnée

Middleware

Un middleware est une couche logiciel entre deux couches de logiciels. C'est une simple fonction qui a un rôle particulier. Dans le cas du framework Express c'est une fonction entre la requête et la réponse. Cette fonction a accès aux paramètres de la requête et de la réponse et donc il permet d'effectuer de nombreuses actions. Un de ses objectifs est la vérification des données envoyées par exemple dans le body des requêtes. Un middleware peut être appliqué à une unique route ou à plusieurs.
De base, le framework possède quelques middleware mais il est possible d'en créer au besoin du projet.

Routage

Le routage dans Express.js consiste à associer des URL spécifiques à des fonctions de gestion de requêtes, appelées "middlewares", qui sont exécutées lorsque ces URL sont atteintes. Cela permet de définir facilement les actions à effectuer en fonction de l'URL demandée par le client.

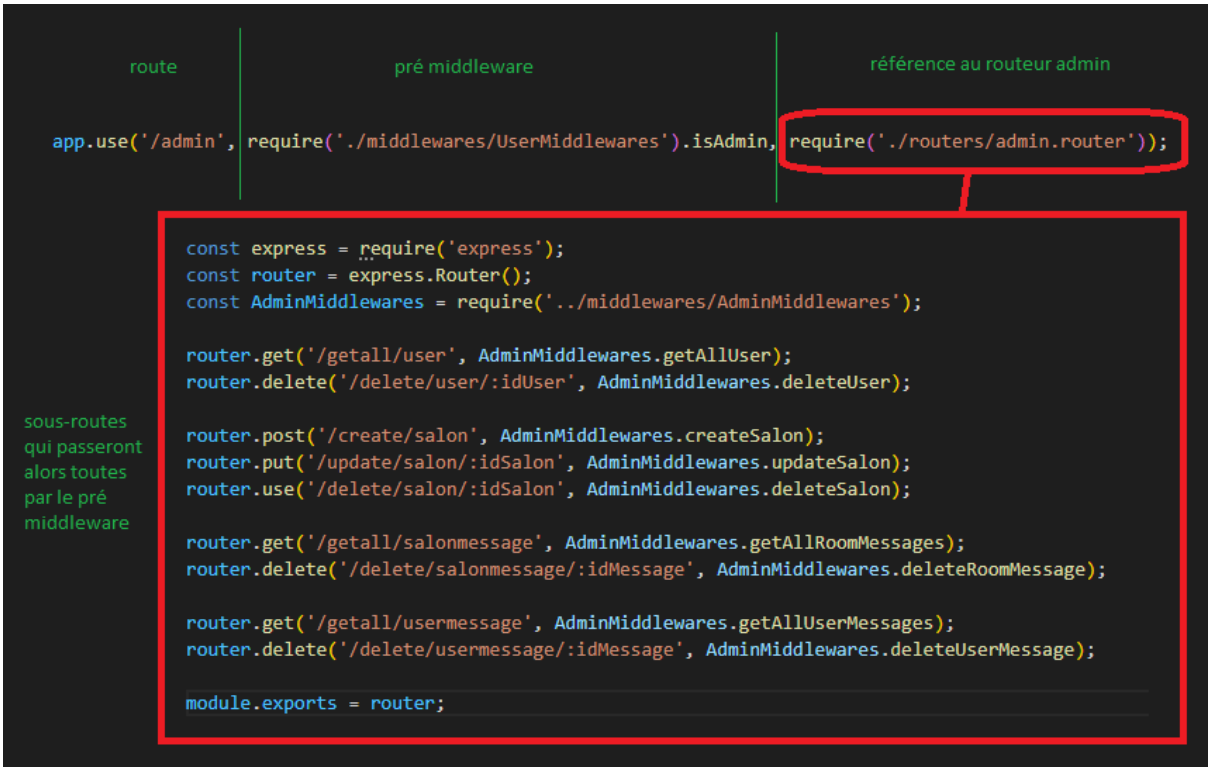
Pour effectuer la séparation du code, nous avons séparé les routes en fonction de leurs responsabilitées :

```
//USER
app.use('/user', require('./routers/user.router'));
//Salon
app.use('/salon', require('./routers/salon.router'));
//MessageSalon
app.use('/messagesalon', require('./routers/messageSalon.router'));
//MessageUser
app.use('/messageuser', require('./routers/messageUser.router'));
//AdminRoute
app.use('/admin', require('./middlewares/UserMiddlewares').isAdmin, require('./routers/admin.router'));
//FilterRoute
app.use('/translate', require('./routers/filter.router'));
```

une route user qui permet d'accéder à toutes les routes pour les données d'un user.
une route salon qui permet d'accéder à toutes les routes pour les données d'un salon.
une route messageuser qui permet d'accéder à toutes les routes pour les messages privées.
une route messagesalon qui permet d'accéder à toutes les routes pour les messages publics.
une route admin qui permet d'accéder à toutes les routes accessibles uniquement par l'admin.
une route translate qui permet de récupérer le filtrage du chat.

Certains middlewares référencent d'autres routeurs contenant elle-même des middlewares de routing pour centraliser certaines étapes.

Exemple :
La route admin référence un routeur contenant plusieurs middlewares routes qui vont alors passées par le pré processus d'authentification du middleware admin, on utilise ainsi le plein potentiel de centralisation des middlewares.



Les routeurs contiennent plusieurs routes d'une même catégorie, ces routes appellent des middlewares qui exécute la logique métier de l'application

Exemple :
La route du routeur admin '/getall/user', appelle le middleware getAllUser

Le voici :

```
/**
 * retourne tout les users nested
 * GET http://127.0.0.1:3000/admin/getall/user
 */
const getAllUser = async (req, res, next) => {
    try {
        let users = getUsersState(await UserRepository.getAll(parseInt(req.params.idMessage)));
        res.status(200);
        res.send(users);

    } catch (err) {
        console.log(err);
        res.status(500).send('error_getting_all_users');
    }
}
```

Application level Middleware

Ce type de middleware exécute la logique métier de l'application et accèdent au repositories au besoin, c'est aussi qui s'occupe de retourner une réponse au client. Elles sont appelées une fois la résolution de route effectuée.

Repositories

Il s'occupe d'initier la connexion à la base de données et définissent les opérations qu'il sera possible d'y effectuer de par ses fonctions. Ils utilisent des models pour retranscrire les données. Notre organisation des repository a été d'en attribuer un pour chaque table de notre base de données.

Models

Les modèles définissent les schémas des entités en base de données, ils sont donc utilisés par les repositories comme abstraction des données pour les utiliser en tant qu'entité.

Exemple model user :

```
const Sequelize = require("sequelize");

module.exports = (sequelize) => {

    const roleModel = require("./role.model")(sequelize);
    const user_possede_role = require("./user_possede_role.model")(sequelize);

    const userModel = sequelize.define("user", {
        idUser: {
            primaryKey: true,
            autoIncrement: true,
            type: Sequelize.NUMBER,
            field: 'pk_id_user',
            allowNull: false
        },
        email: {
            type: Sequelize.STRING,
            allowNull: false
        },
        prenom: {
            type: Sequelize.STRING,
            allowNull: false
        },
        nom: {
            type: Sequelize.STRING,
            allowNull: false
        },
        password: {
            type: Sequelize.STRING,
            allowNull: false
        },
        createdAt: {
            type: Sequelize.DATE,
            allowNull: false,
            field: 'created_at'
        }
```

Socket

Il référence les différents events listeners qui sont mis en place lorsqu'une connexion socket est établie.

Helper

Les helpers mettent à disposition des fonctionnalités qui pourront être utilisées n'importe où dans l'api, ce qui permet de centralisé de la logique.

Exemple :
Notre socket helper met à disposition le management des différents sockets gardé en mémoire, comme retrouver le user associé à un socket par exemple.

Sécurité

La sécurité d'une API est importante car elle protège les données sensibles et confidentielles des utilisateurs et des applications, pour cela nous devons sécuriser notre api contre les injections SQL et il faut assurer une gestion des droits.

L'injection SQL est une technique d'exploitation de vulnérabilités de sécurité qui consiste à injecter du code SQL malveillant dans une application web pour exécuter des commandes non autorisées sur la base de données sous-jacente.

Hachage de mot de passe

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

JWT

Le JWT (JSON Web Token) est important pour la sécurité d'une API car il permet d'authentifier et d'autoriser les utilisateurs de manière sécurisée. Il garantit l'intégrité des données en signant numériquement les informations contenues dans le token. De plus, il permet de stocker des informations supplémentaires dans le token lui-même, réduisant ainsi la dépendance sur les sessions serveur et offrant une approche stateless.

il est composé de 3 parties :

1. L'en-tête (header) spécifie le type de token (JWT) et l'algorithme de signature utilisé, tel que HMAC, RSA ou ECDSA.
2. La charge utile (payload) contient les informations supplémentaires que l'on souhaite transférer, telles que l'identité de l'utilisateur, des autorisations ou des métadonnées. La charge utile peut être divisée en trois types : les données réservées (claims) prédéfinies (ex : l'émetteur, le sujet, la date d'expiration), les données publiques (claims) personnalisées et les données privées (claims) spécifiques à l'application.
3. La signature est générée en utilisant l'en-tête, la charge utile, une clé secrète (symétrique) ou une paire de clés publique/privée (asymétrique). La signature

garantit l'intégrité des données et permet de vérifier si le token a été modifié depuis sa création.

Pour conclure le JWT permet d'assurer l'authentification des utilisateurs de l'application en toute sécurité et de savoir si le user est de connaître ses droits.

Identification

L'identification est un enjeu important et ce pour 3 raisons :

-Le facteur données : L'auteur d'un message est une donnée au même titre que le message lui-même, il participe à la compréhension de ce dernier.

-Le facteur traçabilité : Tout ce qui est attrait au dialogue peut rapidement basculer dans le cadre de problème juridique et donc pour cela, l'identification de l'auteur est un enjeu majeur.

-Le facteur sécurité : Empêcher à l'usurpation de l'identité participe à la sécurisation du facteur traçabilité vu précédemment et permet la gestion de droit comme par exemple : l'accès conditionné en lecture et en écriture à certaines données.

Pour cela nous avons mis en place 3 middleware, ils sont un passage forcé pour chaque opérations attrayant à la gestion ou à l'accès de données utilisateurs :

- Le middleware isAdmin : représente un rôle spécifique et restreint d'utilisateur qui sont chargés de la bonne digestion de l'application. Il permet l'accès à des fonctionnalités qui sortent du cadre de l'utilisateur classique.

- Le middleware isAuthorized : il conditionne les opérations à n'être disponible que pour les personnes concernées ou administrateur. Exemple :
  La gestion du profil ne peut être faite que par la personne elle-même ou par un administrateur.

- Le middleware isAuthentified : Toutes les opérations sauf évidemment l'inscription et la connexion passe par un middleware qui vérifie l'authentification. Ceci dans le but d'assurer le facteur **traçabilité,** les intéractions ne peuvent alors être faites seulement par des personnes connues de l'application.

```
const isAuthorized = async (req, res, next) => {
    try {
        if (!req.params.idUser) {
            res.status(406).send('idUser GET params is missing');
            return;
        }
        res.locals.userId = parseInt(req.params.idUser);

        if (res.locals.authentifiedUser.idUser != res.locals.userId
            && !UserRepository.isAdmin(res.locals.authentifiedUser)
        ) {
            res.status(406).send('not_authorized');
            return;
        }
        next();
    } catch(err) {
        console.log(err);
        res.status(500).send('error_during_authorization_check');
    }
}
```

Fixture

Les fixtures sont importantes lorsque l'on dev un back end car cela permet de créer des données de test cohérentes et réutilisables, facilitant ainsi le développement et le débogage, elles permettent également d'aider à simuler des scénarios réels et variés, permettant ainsi de valider le bon fonctionnement de l'application dans différentes situations.

Tests

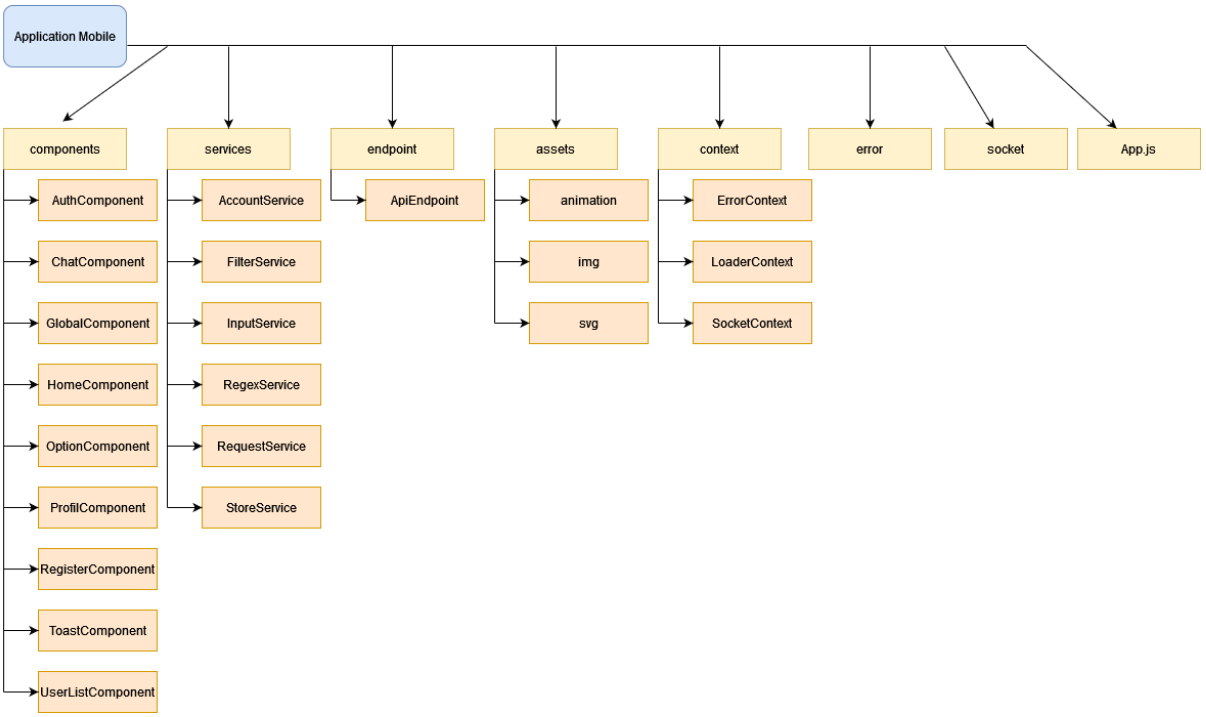

j'ai utilisé Thunder Client (extension pour Visual Studio Code) ainsi que postman, afin de tester mon API Express.js. Mon objectif était de vérifier si le token JWT était fonctionnel et si les routes renvoient les données correctement. J'ai effectué plusieurs requêtes à mon API en fournissant le token JWT dans les en-têtes appropriés. À ma grande satisfaction, toutes les requêtes ont été traitées avec succès et les données demandées ont été renvoyées conformément à mes attentes. Cela m'a permis de confirmer que le système d'authentification basé sur JWT fonctionnait correctement et que les routes étaient configurées de manière adéquate. Thunder Client s'est révélé être un outil pratique et efficace pour effectuer ces tests.

Développement du front end de l’application

Expo

Expo offre des fonctionnalités telles que le live-reload pour visualiser instantanément les modifications apportées à l'application, il permet le débogage directement sur mobile ainsi qu'un déploiement facile sur les appareils mobiles via une application Expo dédiée. Cela permet aux développeurs de se concentrer davantage sur la logique de l'application plutôt que sur les détails techniques de la configuration de l'environnement.
La facilité technique d’expo est un avantage mais aussi un inconvénient car il est difficile d’exclure expo d’un projet, expo n’offre pas toutes les fonctionnalités natives en mobile (bluetooth,map).

Arborescence



Dossier component

Nous nous sommes de l’architecture angular pour structurer nos composants dans le projet : chaque components est un dossier qui contient ses 3 parties élémentaires, à savoir

un fichier component.jsx contenant sa logique (state, fonction, evenement..),
un fichier template.jsx qui contient uniquement le contenu balistique et
le fichier style.jsx qui permet de mettre en forme les éléments visuels.

Dossier services

Les services permettent de fournir des fonctionnalités qui seront utilisées dans toutes l'application ce qui permet de centraliser de la logique et donc de ne pas se répéter.

Dossier endpoint

l'endpoint donne accès à l'entièreté de l'application à toutes les routes utilisées et utilisables par celle-ci.

Dossier Assets

Le dossier assets stockent toutes les ressources utilisées par l'application pour fournir du contenu visuel, auditif et interactif. Dans notre cas les animations, les images et les svg.

Dossier Context

Les providers sont un mécanisme pour créer un contexte dans lequel les données peuvent être fournies et consommées par les composants descendants sans avoir besoin de passer explicitement les données via les props.
Nous l'avons utilisé pour rendre accessible les fonctionnalités liées à la gestion d'erreur, de toaster (bulle informative), de loader et de socket.

Dossier Error

Ce dossier contient la classe ChappyError qui permet d'encapsuler la logique de gestion de l'application

Sécurité

Pour assurer l'authentification, chaque components permettant les interactions utilisateurs utilise cette fonction qui redirige l'utilisateur en fonction de s'il est connecté.

```
const init = async () => {
  try {
    let connected = await AccountService.isLogged();
    if (!connected) {
      props.navigation.replace("Login");
    }
```

Context

Les contexts que l'on a mis en place sont des enjeux majeurs pour l'expérience utilisateur : toaster,loader,error. Tout 3 ont un rôle dans l'information à l'utilisateur et garde une accessibilité des interfaces et des mécanismes.

Loader : informe de l'attente des informations avant de pouvoir interagir avec celles-ci.

Toaster : synthétise une information clé sur une opération qui vient d'être effectuée.

Error : Attrape les erreurs systèmes et formule d'une manière compréhensible pour l'utilisateur final le problème rencontré.

Voir annexe

Asynchrone

La programmation synchrone a pour faiblesse qu'une instruction ne puisse être exécutée qu'une fois la précédente terminée. Ainsi, si l'une d'entre elle prend un temps non négligeable à être résolu,l'ensemble des processus bloqués par exemple une requête si elle était synchrone rendrait l'interface de l'application gelée, les événement ne serait pas déclenché pendant tout ce temps. L'utilisation de l'asynchrone est donc une solution puisqu'elle permet d'attendre la résolution d'un processus en gardant en parallèle le programme fonctionnel. Par exemple : l'envoie d'un message.

Programmation Componentielle

Notre application est un projet react native qui est une extension de react, react étant une librairie dont la philosophie est le développement de composants. La programmation componentielle est un paradigme de programmation où les fonctionnalités sont organisées en composants réutilisables et autonomes. Les composants encapsulent leur propre logique, leur interface utilisateur et leurs dépendances, facilitant la maintenance et la modularité du code.

J'ai par exemple créé un component "Vertical Button" qui est utilisé à plusieurs endroit du projet, ce qui permet de centraliser en un seul endroit son style, son animation, son action au clique…

Template du component :

```jsx
import VerticalButtonStyle from "./vertical-button.style.jsx";

export default function VerticalButtonTemplate(props) {

    return (
        <TouchableOpacity
          onPress={() => {props.actionButton()}}
          underlayColor='#8093FF'
          activeOpacity={0.5}
          >
          <Animated.View style={[VerticalButtonStyle.bulle, props.animation]}>
            <SvgChat width={30} height={30} fill="white" />
            <Text style={{ color: "white", fontSize: 18, marginLeft: 12 }}>{props.text}</Text>
            <View style={props.right ? VerticalButtonStyle.triangleBulleDroite : VerticalButtonStyle.triangleBulleGauche}></View>
          </Animated.View>
        </TouchableOpacity>
    );
}
```

Instanciations du component:

```jsx
<View style={HomeStyle.containerBulle}>
  <VerticalButtonComponent actionButton={props.goRoom}
    text="Chat"
    animation={props.animationChatBtn}
    >
  </VerticalButtonComponent>
  <VerticalButtonComponent actionButton={props.goUserList}
    text="Users"
    animation={props.animationUserListBtn}
    >
  </VerticalButtonComponent>
```

Deux instanciations du component, avec des paramètres différents

Développement du client-admin de l’application

Un panel admin est une interface de gestion réservée aux administrateurs d'une application ou d'un site web. Il permet aux administrateurs d'accéder à des fonctionnalités avancées pour gérer et contrôler différents aspects de l'application.
Dans le cas spécifique de votre panel admin, il offre les fonctionnalités suivantes :

1. CRUD sur les utilisateurs : Vous pouvez créer, lire, mettre à jour et supprimer des utilisateurs de l'application. Cela vous permet de gérer les comptes des utilisateurs, leurs informations personnelles et leurs autorisations.
2. Suppression de messages : Vous pouvez supprimer des messages indésirables ou inappropriés dans l'application. Cette fonctionnalité vous permet de maintenir un environnement de communication sain et conforme aux politiques de modération.
3. Gestion des rooms publiques : Vous pouvez ajouter ou supprimer des rooms publiques dans l'application. Cela vous donne le contrôle sur les espaces de discussion accessibles à tous les utilisateurs, permettant de créer ou de supprimer des lieux de conversation thématiques.
4. Suppression de messages et de rooms privées : Vous avez la possibilité de supprimer des messages et des rooms privées dans l'application. Cela vous permet de gérer les conversations privées entre utilisateurs, en supprimant les contenus indésirables ou en réponse à des demandes de suppression.
5. Filtrage par propriété : Le panel admin vous permet de filtrer toutes les opérations mentionnées ci-dessus en fonction de certaines propriétés. Cela signifie que vous pouvez appliquer des filtres spécifiques pour cibler des utilisateurs, des messages ou des rooms en fonction de critères tels que le nom, l'identifiant, le statut ou d'autres attributs pertinents.