


RAPPORT PROJET DE DEVELOPPEMENT MOBILE

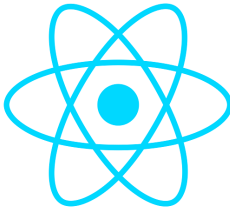
ENSC
2ÈME ANNÉE



Dash Chat



Firebase



React Native

Vous avez un compte

Email

Email...

Mot de passe

Mot de passe...

CONNEXION

Continuer en mode anonyme

Votre pseudo

ANONYME

Réalisation d'une application de chat "DASH CHAT"

BINZAGR Ahmad B. | MARTIN Nicolas

4 Mars 2018 - 11 Avril 2019

Table des matières

1	Introduction	2
1.1	Contexte	2
1.2	Objectif général	2
2	Les spécifications générales	2
2.1	Exigences fonctionnelles	2
2.1.1	Choix de son pseudo par l'utilisateur	2
2.1.2	Affichage de la liste des channels	4
2.1.3	Création d'un nouveau channel	5
2.1.4	Editer un channel plus affichage des messages sans limite	6
2.1.5	Envoi de messages texte	8
2.1.6	Mise à jour en temps réel des messages envoyés sur le channel	9
2.1.7	Amélioration de l'expérience utilisateur	9
3	Les Spécifications détaillées	10
3.1	Structuration et choix de stockage des informations	10
3.2	Choix de conception et de production du code source	11
3.3	Organisation du travail dans le groupe et gestion de projet	12
4	Bilan et perspectives	12
5	Annexes	12
5.1	Dictionnaires des librairies utilisées	12

1 Introduction

Les applications mobiles sont devenues les indispensables du smartphone. Gratuites ou payantes, elles sont pour la plupart téléchargeables sur les plateformes Android et IOS. Jouer sur son smartphone, commander de la nourriture de son restaurant préféré, rencontrer l'âme soeur... Désormais les applications mobiles répondent à de nombreux besoins créés par la société connectée d'aujourd'hui. En voilà de nombreuses idées pour se lancer dans le développement mobile.

1.1 Contexte

Le développement mobile au sens de la discipline permet d'appréhender un nouvel environnement de développement ainsi qu'une nouvelle technologie répondant aux grands besoins du marché actuel. Pour ce projet, nous travaillons avec le Framework "React Native" pour le côté FrontEnd et "FireBase" pour le BackEnd.

1.2 Objectif général

L'objectif de notre travail consistait à créer une application de chat(messagerie instantanée), permettant à deux ou plusieurs utilisateurs de s'échanger des messages via un channel de discussion, une version très simplifiée de Slack ou Discord.

2 Les spécifications générales

2.1 Exigences fonctionnelles

Afin de répondre aux fonctionnalités attendues du projet, nous nous sommes intéressés dans un premier temps aux fonctions de base que doivent répondre l'application et les avons priorisées en mettant l'accent sur les fonctions indispensables au bon fonctionnement de celle-ci.

Pour que l'application soit fonctionnelle il faut qu'elle réponde aux exigences suivantes, l'utilisateur puisse :

- Choisir son pseudo
- Afficher la liste des channels existants
- Créer un nouveau channel
- Editer un channel et remonter les messages dans la liste
- Avoir une mise à jour en temps réel des messages envoyés sur le channel
- Vivre une meilleure expérience utilisateur

2.1.1 Choix de son pseudo par l'utilisateur

Cette fonctionnalité a bien été implémentée et bien exploitée pour offrir à l'utilisateur l'opportunité de garder un certain anonymat. On lui demande de choisir son pseudo dès la création de son compte. Nous stockons ensuite le pseudo ainsi que les informations lui permettant de se connecter à notre application sur "Firebase". Rappelons que "Firebase" permet l'authentification sociale à travers différentes modes de connexion (par e-mail et mode passe, via Facebook, via Google etc.), dans notre application nous demandons à l'utilisateur une adresse mail et un mot de passe. Une connexion anonyme étant permise par "Firebase", nous avons le choix de l'exploiter. Un utilisateur lambda peut donc se connecter en indiquant juste son pseudo pour avoir accès au chat.

L'avatar de l'utilisateur est défini en fonction du pseudo choisi.



Dash Chat

Vous avez un compte

Email

Email...

Mot de passe

Mot de passe...

CONNEXION

Continuer en mode anonyme

Votre pseudo

ANONYME

Connexion


Inscription



A registration form with a light gray border. At the top right, there is a small green arrow icon. The form contains four input fields: 'Pseudo' with the value 'Momo', a pixelated avatar of a man with brown hair, glasses, and a beard, 'Email' with the placeholder 'Email...', and 'Mot de passe' with the placeholder 'Mot de passe...'. Below the inputs is a blue button with the text 'INSCRIPTION' in white capital letters.

Pseudo

Momo



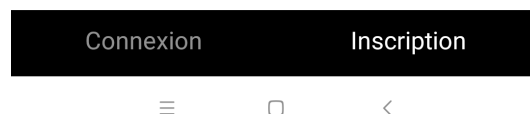
Email

Email...

Mot de passe

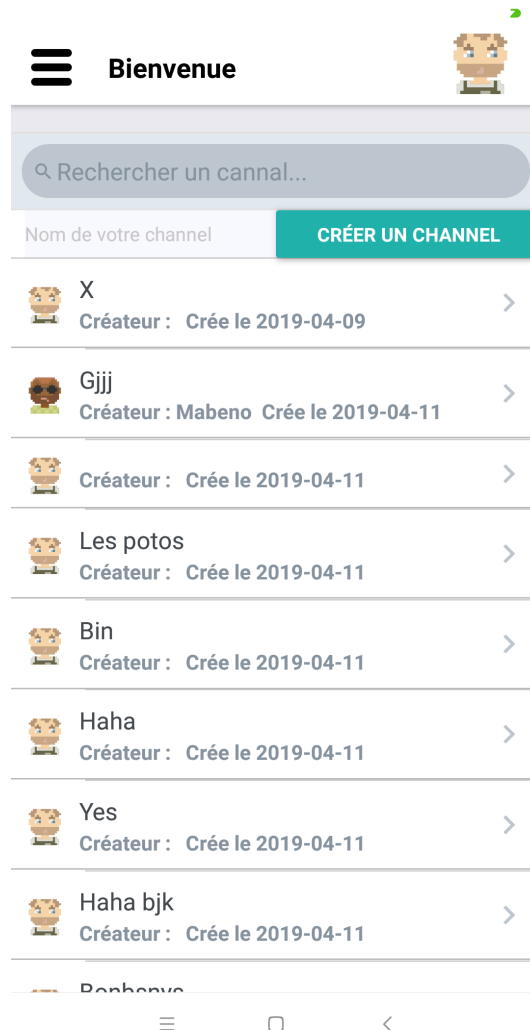
Mot de passe...

INSCRIPTION



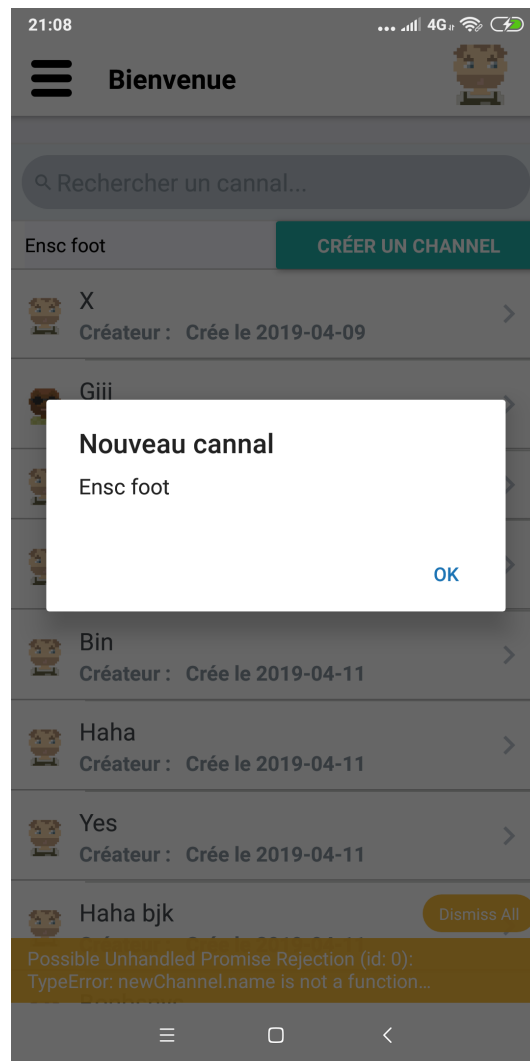
2.1.2 Affichage de la liste des channels

L'affichage de la liste des channels a bien été réalisé. Nous requêtons depuis notre base de données "Firebase" l'ensemble de channels stockés au noeud parent "channels", nous les stockons dans un tableau. Un tableau géré de façon globale grâce à l'intégration de "Redux" dans le store de notre application. A l'aide d'une ***FlatList*** les channels contenu dans notre tableau.



2.1.3 Création d'un nouveau channel

L'utilisateur a le libre droit de créer un channel en choisissant le nom de ce dernier. La création d'un nouveau channel se fait par l'intermédiaire d'une requête vers notre base de données grâce à la fonction *.push()* qui ajoute au noeud "channels" notre nouveau channel. Nous avons défini l'objet "channel" comme étant une structure avec trois champs : name (nom du channel), idChannel (l'id du channel), user (l'id de l'user ayant crée le channel).



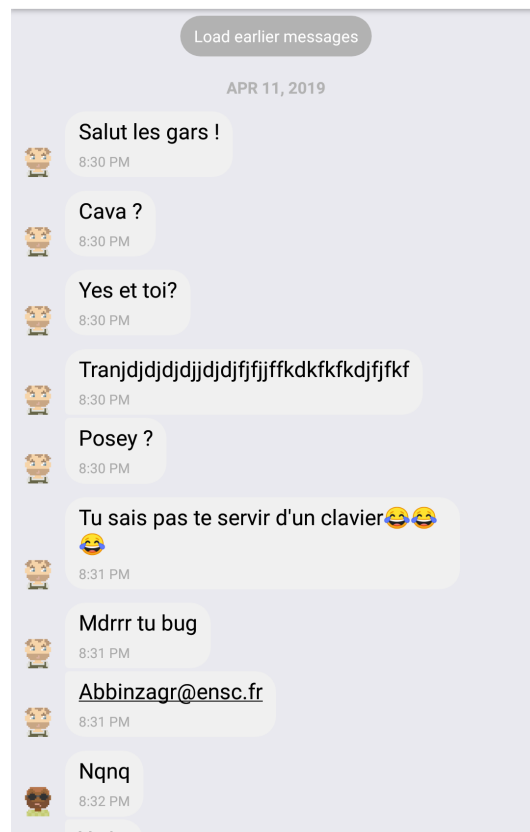
2.1.4 Editer un channel plus affichage des messages sans limite

Nous avons implémenté l'affichage des channels, à présent l'utilisateur peut éditer un d'un channel de la liste des channels et voir l'ensemble des messages déjà postés avec la possibilité de remonter les messages. Nous avons fait le choix de mettre en place un bouton *charger plus* qui requête depuis la base de données et stocks les vingt messages précédents les vingt derniers déjà visibles dans la discussion.

Nous avons fait ce choix parce que la librairie "UI" *GiftedChat* que nous utilisons pour le style d'affichage des messages nous offre pas la possibilité de pouvoir lui associer une action sur le "scroll" afin qu'on puisse remonter les messages juste au scroll.

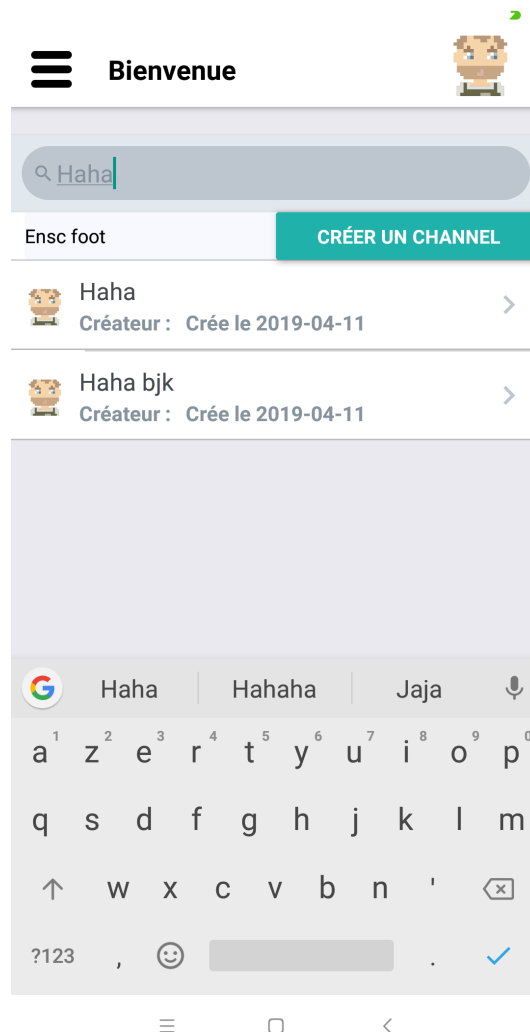
L'utilisateur peut également faire une recherche du channel qu'il souhaite éditer, grâce à notre fonction `searchRequest()` qui prend en paramètre le texte du channel à chercher et nous liste les channels correspondants au texte saisi dans le store Redux contenant les channels, on rappelle que la recherche est évolutive et dépend du texte saisi.

← Les potos



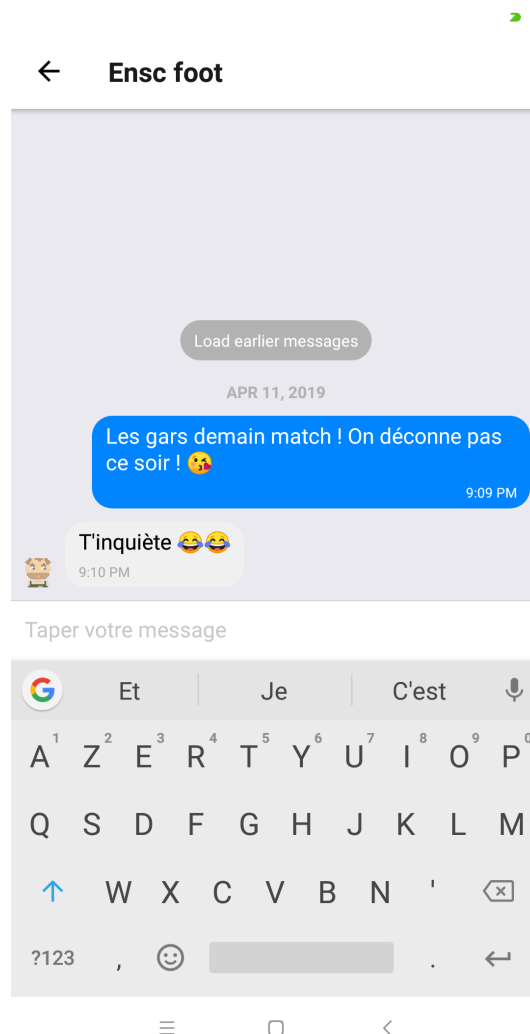
Taper votre message





2.1.5 Envoi de messages texte

A présent que l'utilisateur ait édité un channel, il peut envoyer un message. A l'aide de la librairie *GiftedChat* l'utilisateur peut saisir son message, ce dernier est ensuite envoyé grâce à une méthode `"sendMessageChannel(idChannel, messages)"` accessible depuis le store Redux qui prend en paramètre l'ID du channel ainsi que le contenu du message. La méthode ainsi implémentée concatène le message à la file des messages déjà existants. Au niveau structurel sur "Firebase" le message sera rattaché directement au noeud dont l'ID correspond au channel grâce à la méthode `.push()` qu'on a déjà vu précédemment dans la section création d'un channel. On appelle un message, une structure de donnée avec une chaîne de caractères (texte du message), l'ID de l'user, l'heure à laquelle le message a été posté et l'ID du channel auquel il appartient ainsi que son avatar.



2.1.6 Mise à jour en temps réel des messages envoyés sur le channel

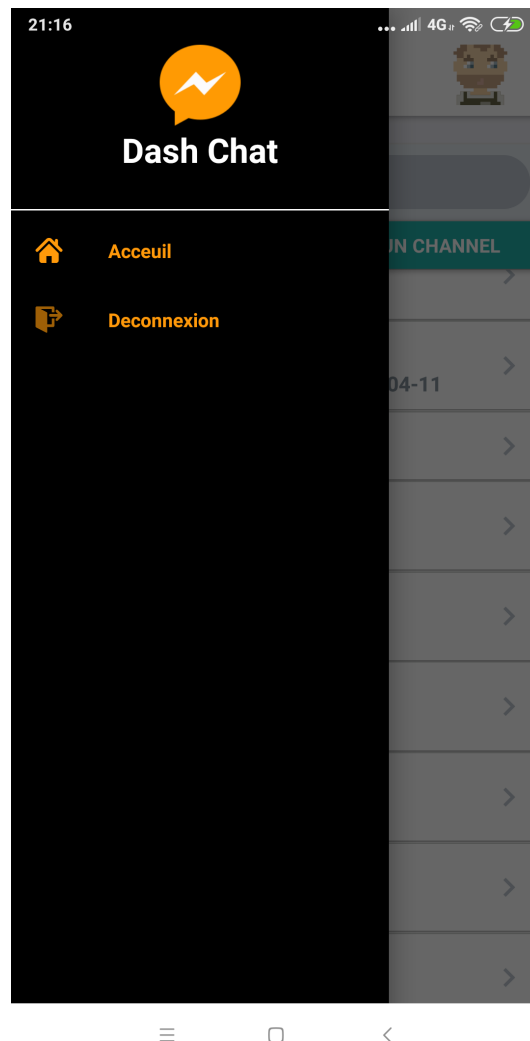
Nous nous sommes assurés que la mise à jour des messages envoyés sur un channel soit en temps réel. Grâce à la méthode "**sendMessageChannel(idChannel, messages)**" dont on a explicité le comportement précédemment une fois qu'un nouveau message est posté sur la base de données, cela entraîne un changement du state actuel qui "re-rend" la discussion avec les dernières modifications.

2.1.7 Amélioration de l'expérience utilisateur

Nous avons essayé d'offrir la meilleure expérience utilisateur dont nous étions capable de mettre en place en utilisant la librairie **GiftedChat** qui se charge de la mise en forme et du style d'affichage de nos messages, nous avons à droite de la conversation les messages que l'utilisateur a envoyé et à gauche ceux des personnes présentent dans la discussion. Un style d'affichage des messages Semblable à celui de **messenger**. Une fois l'utilisateur connecté nous affichons un message de bienvenu suivi de son pseudo en titre en début de notre page de gestion des conversations.

Nous avons aussi mis en place un "burger menu" qui permet à l'utilisateur de se déconnecter ou de se reconnecter à l'application. Grâce à l'intégration d'avatars via l'API **DiceBear Avatars**, chaque utilisateur se voit attribuer un avatar choisi aléatoirement dès qu'il aura saisi son pseudo. L'utilisateur verra apparaître dans la bulle de discussion les avatars des différents participants au channel.

Par ailleurs nous avons également essayé l'intégration d'emojis afin d'améliorer l'expérience utilisateur en reprenant le code source de la librairie **GiftedChat** dans le but de modifier le component **inputtoolbar** pour y intégrer les emojis, nous y sommes parvenus mais l'affichage dépassait de l'écran...

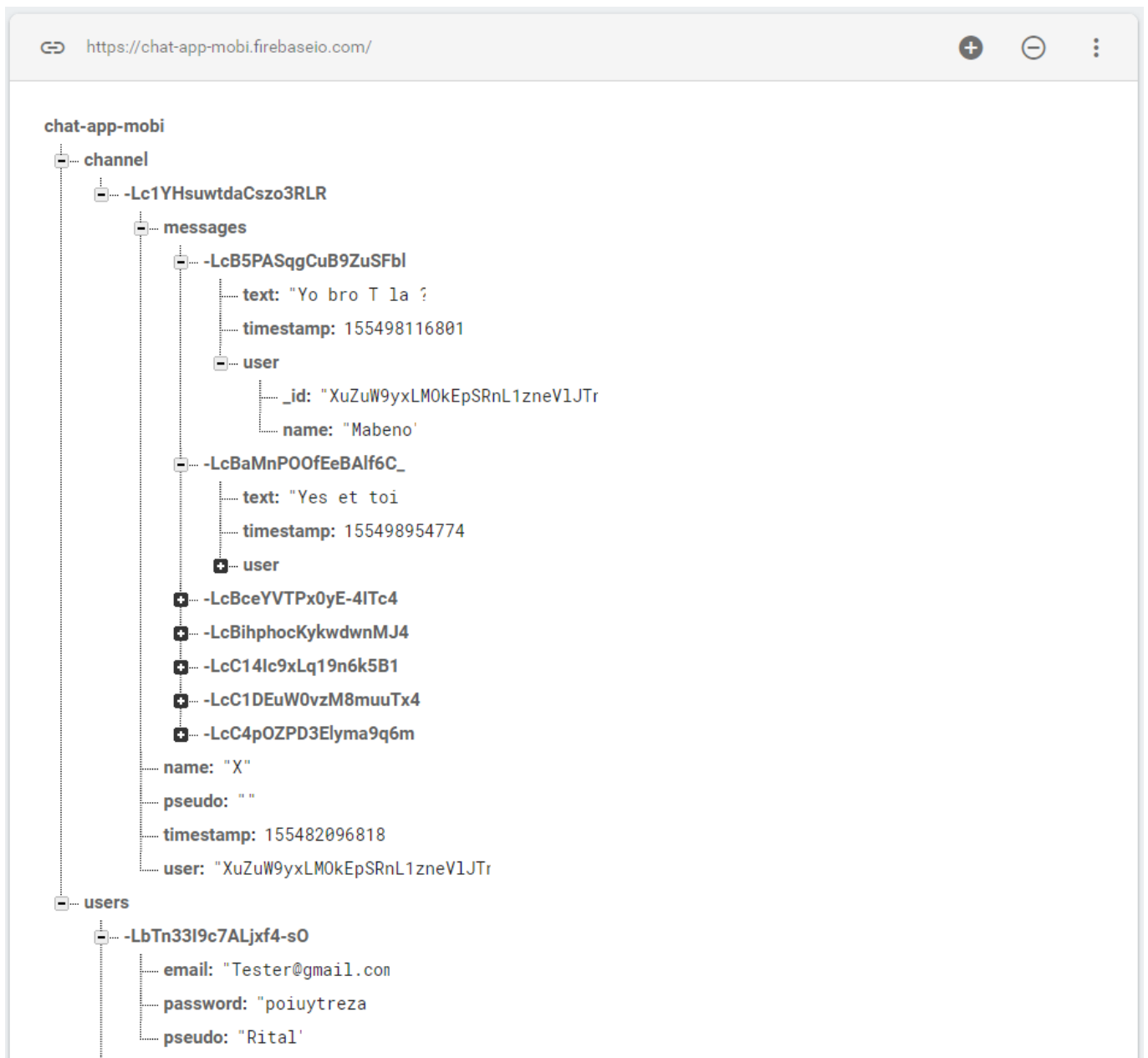


3 Les Spécifications détaillées

3.1 Structuration et choix de stockage des informations

Nous avons dans le cadre de ce projet, nous avons fait le choix de stocker l'ensemble des informations dans une base de données "non relationnelle" en utilisant "Firebase" qui propose d'héberger en NoSQL et en temps réel des bases de données. Nous avons fait ce choix parce qu'il nous semble être le meilleur pour le stockage de nos données, "Firebase" est simple d'utilisation mais en plus il nous offre la possible utilisation de l'authentification sociale.

Nous avons par la suite structurer nos données de la meilleure des manières afin qu'on puisse avoir une architecture à plusieurs niveaux. Avec des noeuds fils rattachés directement aux noeuds parents. Par exemple chaque channel contient les messages qui lui sont rattachés et le channel concerné est directement rattaché au noeud "Channels" qui lui contient l'ensemble des channels déjà existants. Nous nous sommes assurés de bien définir ce que c'est un channel et un message, ces derniers étant nos objets de classe.



3.2 Choix de conception et de production du code source

Le choix de conception et de production de notre code source a été guidé en grande partie par l'ensemble de technique de codage vu et réalisé dans le cadre des TP et tuto de ce module (structure des fichiers sources etc).

Tout d'abord l'application a été réalisé sous windows à l'aide des outils suivant : l'éditeur de texte Visual Code, Prettier pour formaliser le code, du framework "React Native" et "Firebase" pour le stockage et la gestion des données persistantes dans une base de données non relationnelle "NoSQL".

Nous avons utiliser le Redux store en y mettant deux reducers s'occupant respectivement des channels et de l'authentification. Celui responsable du channel contient un tableau de channel qui est chargé au moment de l'authentification. Tandis que celui responsable de l'authentification contient le pseudo de l'utilisateur, l'id de l'utilisateur ainsi que toutes les methodes firebase que l'on utilisent dans notre DataSource.

Par ailleurs nous avons suivi une logique répondant au principe de conception **DRY** (Don't Repeat Yourself). Nous avons combiné cette approche avec une autre **KISS** (Kepp It Simple, Stupid) pour que

notre code soit le plus simple possible et conservable à l'avenir. Nous avons bien veillé à ce que notre code respecte la convention camelCase, toute en essayant de choisir au mieux des noms de propriété, de méthode, de paramètre et de variable reflétant leur rôle. Tout en veillant à commenter notre code du mieux possible.

3.3 Organisation du travail dans le groupe et gestion de projet

Tout au long de ce projet, nous avons utilisé le **Pair programming**. C'est une méthode de travail dans laquelle deux personnes travaillent ensemble sur un même poste de travail. Une personne rédige le code, la seconde personne accompagne celui qui écrit en décelant les imperfections, en vérifiant que le code implémente correctement la fonctionnalité souhaitée et en suggérant des alternatives de développement. Il se charge également de consulter la documentation et les ressources à notre disposition pour le débogage du code. Les rôles se sont échangés très régulièrement.

Nous avons utilisés le logiciel de version **Git** ainsi que la plateforme **github** pour pouvoir gérer au mieux nos différentes versions. Nous avons également utilisé "Trello" comme outil de gestion de projet.

4 Bilan et perspectives

Ce projet fut l'occasion pour nous de prendre en main une nouvelle technologie qui est "React Native" ainsi que "Firebase" pour gérer nos données dans une base "non-relationnelle". Il a été aussi l'occasion de dénombrer de nombreux avantages qu'offre "React Native" dans la conception d'une application mobile. Comme on a pu le voir pendant les TP, "React Native" est Open Source. Une grande communauté s'est ainsi construite autour du framework et a permis son accroissement. Par ailleurs nous avons puis réaliser certaines choses qu'on pensait non réalisable au départ à cause des restrictions sur les librairies que nous avons pu exploiter. Mais contre toute attente nous avons eu accès même au code source des librairies pour nous en approprier le code ainsi le modifier à notre aise et faire des choses comme on le souhaitait.

Enfin nous sommes satisfaits d'avoir pu rendre une application aussi ergonomique et conviviale possible mais aussi fonctionnelle à défaut d'être parfaite.

5 Annexes

5.1 Dictionnaires des librairies utilisées

- "react-native-gifted-chat"
- "react-redux";
- "react-native-elements";
- "redux";
- "firebase";
- "react-navigation";
- "native-base";