# Manuel développeur

par Jaton David, Naddeo Eddy & Villarejo Maxime

Voici le manuel développeur contenant toutes les informations utiles pour le développement et le maintien de T-Vault. Vous y trouverez des informations concernant les versions des outils utilisés ainsi que des dépendances. Ce manuel contient également des informations relatives à l'implémentation.

# Table des matières

# Framework

React Native : v0.74.5

# Dépendance

Toutes les librairies et dépendances utilisées sont listées dans le fichier “package.json”.

# Configuration du projet

Tout d’abord, clonez le projet via git localement sur votre ordinateur. Installez ensuite Node.js sur votre ordinateur. Une fois cela fait, ouvrez le répertoire du projet dans votre IDE et, dans un terminal, utilisez la commande “npm i” pour installer les dépendances et les diverses librairies utilisées dans ce projet.

Une fois cela fait, pour tester l’application, la commande `expo start` permet de lancer le serveur de test Expo. Il faudra ensuite utiliser l’une des solutions proposées par Expo afin de tester l’application.

# Implémentation

Cette section explique les points cruciaux de l’implémentation de T-Vault.

## Firebase

Firebase est un outil d’aide au développement pour les applications mobiles et web. Il propose un ensemble de fonctionnalités permettant de stocker des données, d'envoyer des notifications, de remonter des erreurs liées aux clics des utilisateurs, etc. Dans le cadre de T-Vault, nous l'utilisons pour la gestion des utilisateurs et le stockage des données.

## Gestion des utilisateurs

Firebase offre un outil de gestion des utilisateurs et des méthodes de connexion. T-Vault utilise cet outil pour la création des comptes et le stockage de ces derniers. Pour la création des comptes, dans le fichier `Signup.jsx`, la librairie `firebase/auth` est importée et c’est elle qui permet la gestion des méthodes d'authentification. Dans le cadre de ce projet, les utilisateurs doivent s’inscrire avec une adresse e-mail et un mot de passe.
La librairie `firebase/firestore` permet quant à elle de stocker des données dans la base de données. Dans la gestion des utilisateurs, elle nous sert à stocker les noms d’utilisateur uniques pour chaque compte créé ainsi qu’un UID pour chaque utilisateur.

# Base de données

Firebase offre également une base de données NoSQL. Elle est utilisée pour stocker les noms d'utilisateurs et les UID des comptes. Elle est également utilisée pour stocker les données liées aux voyages et aux étapes de ces derniers. Chaque voyage et chaque étape possède un ID unique de 6 caractères. Les images postées par les utilisateurs sont stockées sur le cloud de Firebase. Pour stocker les données, les librairies `firebase/storage` et `firebase/firestore` doivent être importées.

# Arborescence

L'arborescence de l'application est divisée comme suit :

## app

Contient tous les fichiers de l'application. Chaque fichier présent dans `app`, hormis ceux nommés `layout`, est une page visible et accessible par l'utilisateur. Les fichiers se trouvant à la racine du dossier `app` implémentent les pages n'ayant pas besoin d'être authentifiées pour y accéder. `Index.jsx` est la page permettant de s'identifier et `Signup.jsx` la page permettant de créer son compte. Le fichier `_layout.jsx` sert de composant de layout principal pour ces pages. Il gère les choses suivantes :

- Gestion de l'authentification
- Redirection en fonction de l'état de l'utilisateur
- Affichage d'un indicateur de chargement

## Structure de l'application

### (auth)

Le dossier `(auth)` se trouve dans le dossier `app`. Il regroupe toutes les pages nécessitant d'être authentifié pour y accéder. Le fichier se trouvant à la racine de `(auth)` sert de composant de layout principal pour les pages nécessitant une authentification.

### (tabs)

Le dossier `(tabs)` se trouve dans le dossier `app`. Il regroupe toutes les pages accessibles via la barre de navigation en bas de l'application.

### account

La page `account` sert à consulter les informations du compte de l'utilisateur. Sur cette page, il est également possible de les modifier et de se déconnecter.

### discover

La page `discover` permet de consulter les voyages que d'autres utilisateurs ont partagés avec nous sans nous donner les droits d'écriture.

### map

La page `map` permet de consulter une carte du monde avec les différents étapes des voyages à noter dessus. Il est possible de choisir si l'on veut voir uniquement ses voyages ou ceux qui ont été partagés avec nous.

### trip

La page `trip` permet de voir les voyages que nous avons créés ou ceux qui ont été partagés avec nous avec les droits d'écriture. Cette page permet également de créer un voyage via une fenêtre modale.

### step

Le dossier `step` se trouve à la racine du dossier `(auth)`. Il contient un fichier `[id].jsx`. Ce fichier implémente la page permettant de visualiser un step avec les photos et commentaires. L'ID du step et du voyage est passé à cette page via le routeur pour pouvoir récupérer les bonnes informations sur la base de données. La page s'appelle `[id]` pour des raisons de routage dynamique.

### trip

Le dossier `trip` se trouve à la racine du dossier `(auth)`. Il contient un fichier `[id].jsx`. Ce fichier implémente la page permettant de visualiser un voyage. Les steps y sont affichés ainsi que les informations entrées à la création du voyage. L'ID du voyage est passé à cette page via le routeur pour pouvoir récupérer les bonnes informations sur la base de données. La page s'appelle `[id]` pour des raisons de routage dynamique.

### updateStep

Le dossier `updateStep` se trouve à la racine du dossier `(auth)`. Il contient un fichier `[id].jsx`. Ce fichier implémente la page permettant de modifier les informations contenues dans un step. À la modification d'un step, les images sont téléchargées en local et supprimées de la base de données jusqu'à la validation de la modification afin d'éviter un conflit si deux personnes modifient un step en même temps. L'ID du step et du voyage est passé à cette page via le routeur pour pouvoir récupérer les bonnes informations sur la base de données. La page s'appelle `[id]` pour des raisons de routage dynamique.

## Composants

Plusieurs composants ont été créés pour ce projet.

### AddDiscoverTrip

Ce composant permet d'ajouter un voyage dans sa page de découverte via un code généré par l'utilisateur voulant nous partager un voyage.

### DatePickerModal

Ce composant implémente une fenêtre modale pour le composant `DateTimePicker` de React Native Community. Il permet au `DateTimePicker` sur iOS d'apparaître directement sans requérir d'actions supplémentaires de l'utilisateur.

### FirestoreListenerContext

Ce composant sert à gérer de manière centralisée les listeners de Firestore.

### Header

Ce composant implémente le header des pages principales de l'application.

### LocationPicker

Ce composant implémente la fonction de scope sur la carte. Il permet à l'utilisateur de choisir la localisation en la sélectionnant sur la carte plutôt que de la spécifier à la main.

### ShareTripModal

Ce composant implémente la fenêtre modale permettant de partager un voyage avec un autre utilisateur. Il spécifie la logique pour ajouter l'utilisateur et lui administrer des droits de lecture ou d'écriture.

### StepCard

Ce composant implémente les cartes permettant d'afficher les informations principales d'un step à l'intérieur d'un voyage.

### TabBar

Ce composant implémente la barre de navigation permettant d'accéder aux pages principales de l'application.

### TripCard

Ce composant implémente les cartes permettant d'afficher les informations principales d'un voyage.

### TripModal

Ce composant implémente la fenêtre modale permettant d'ajouter un voyage.

# Tests

## Header

Les tests du composant Header servent à vérifier si le composant se rend correctement en fonction des propriétés qui lui sont données.

## Signin

Les tests de la page Signin « moque » les fonctions de Firestore. Il vérifie que les bons arguments sont passés dans le bon ordre pour garantir le bon fonctionnement du login.

## Tabbar

Les tests du composant Tabbar servent à vérifier si le composant s’affiche correctement et au bon endroit. Ils vérifient également si les événements sont déclenchés correctement lorsque les boutons sont pressés.