Mamadou Diouldé Diallo - MDD22049622
Romain Guillot - GUIR15079709

Remise préliminaire R1

Ce rapport prélimaire R1 a pour objectif de présenter notre méthodes de travail, le but de l'application developpée (cas d'utilisation, description des fonctionnalités).

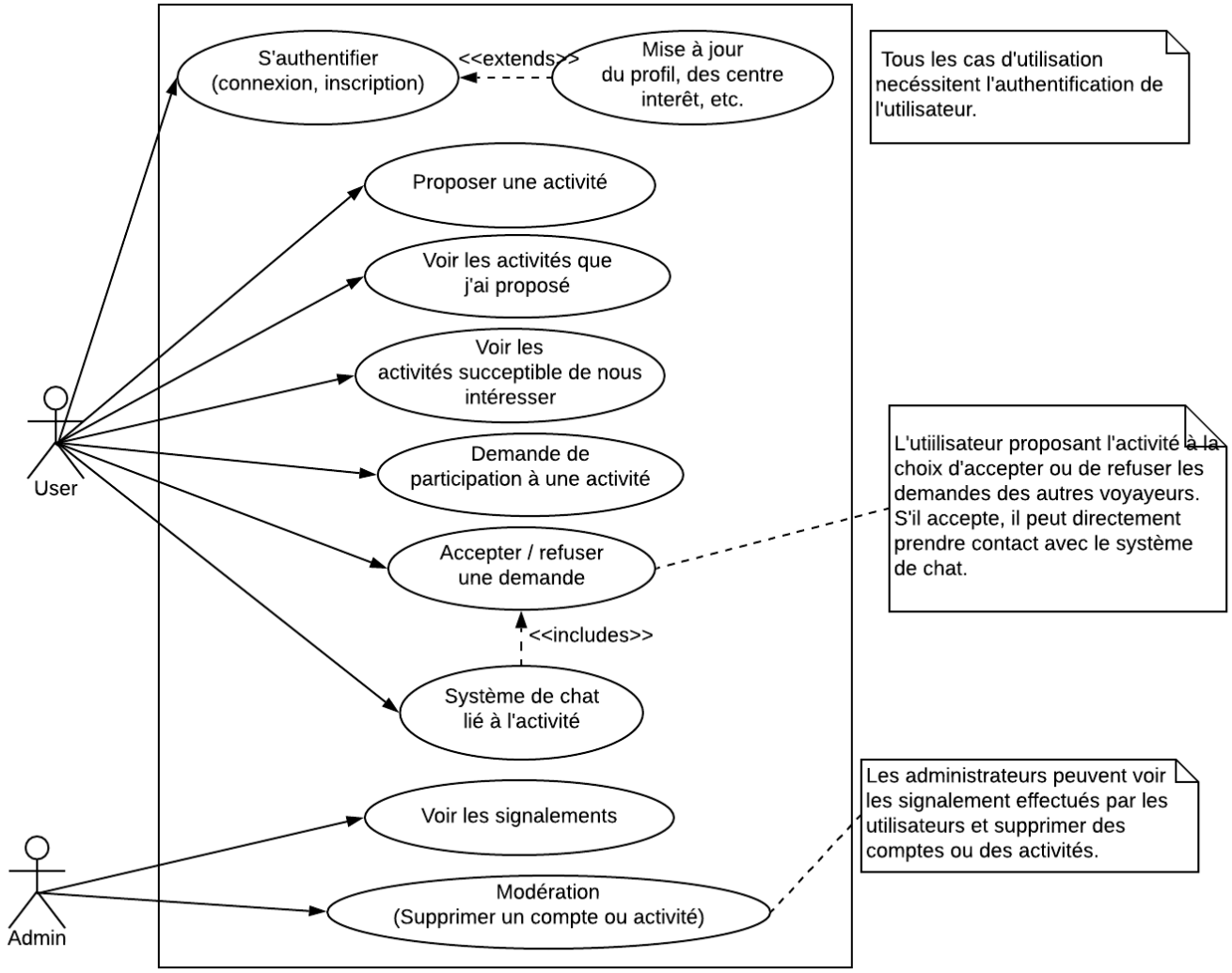
En fin de rapport, nous avons aussi joint le résumé de 4 premiers sprints.

Somaire

Description de l'application

Le projet à pour but de développer une application mobile multiplateforme (Android et iOS) permettant de mettre en relation des voyageurs se trouvant à proximité ayant des intérêts communs à faire des activités ensemble (ex : boire une bière dans un bar, faire une randonnée, aller en club, visiter un musée, etc.).

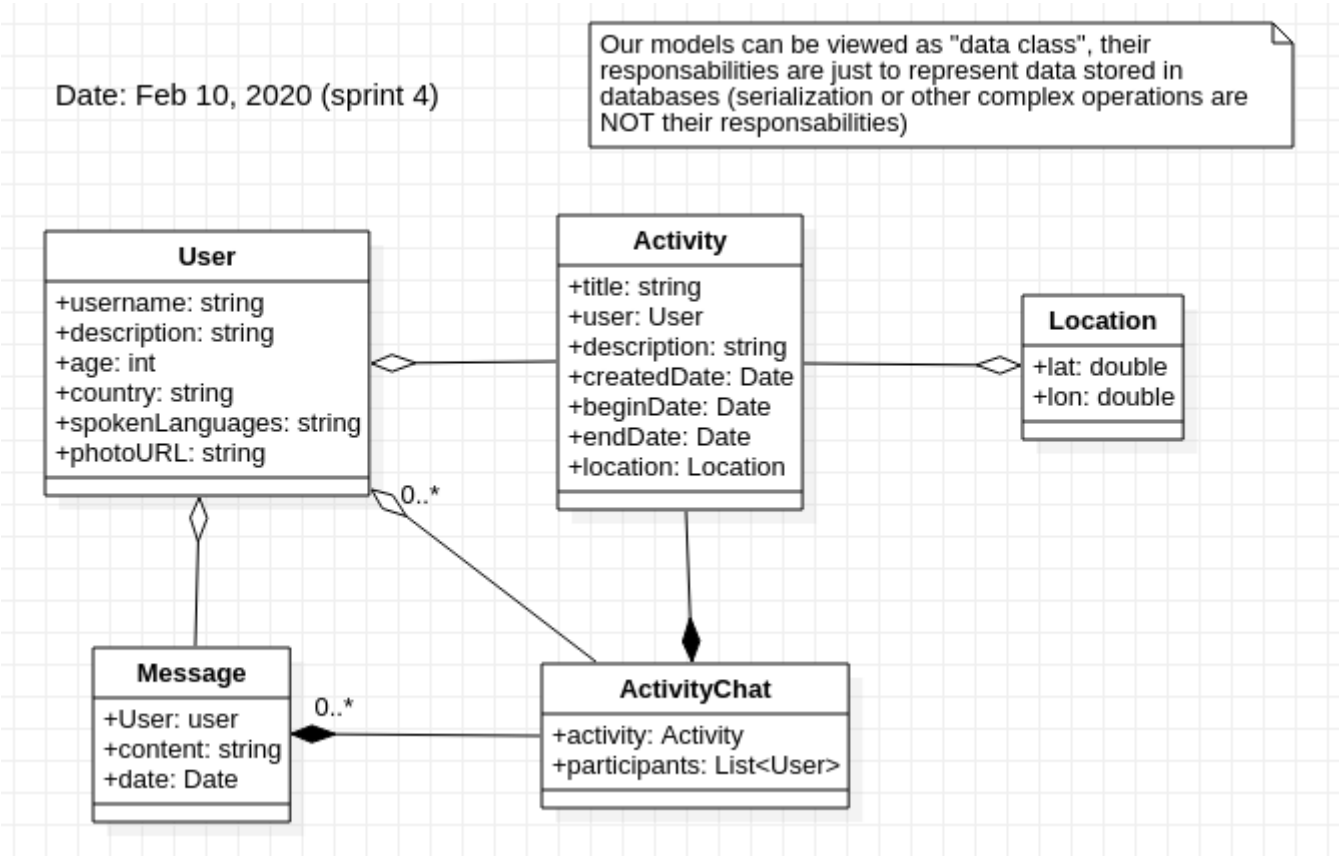
Étant nous même voyageurs nous avons déjà eu ce besoin mais nous n'avons trouvé aucun service répondant à cette problématique.



Architecture : Modèles

Dernière modification : 12 fev 2020

Voici le diagramme de classe à jour avec le **sprint#4** (le système de messagerie n'est pas encore spécifié, les classes `ActivityChat` et `Message` sont donc à l'état de suppositions pour l'instant)



User

TODO : fields constraints Issue#10

Propriétés	Contraintes	Remarques
username		
age		
description		
langues parlées		

Propriétés	Contraintes	Remarques
pays		
photos		

Activité

Propriétés	Contraintes	Remarques
date de création		utile principalement pour le tri
titre	6 < len < 50	
utilisateur		l'utilisateur proposant l'activité
description	20 < len < 500	
date de début	> date de création	date à partir de laquelle l'utilisateur est disponible pour l'activité
date de fin	> date de fin	date limite jusqu'à laquelle l'utilisateur n'est plus disponible pour effectué l'activité
localisation		localisation approximative de l'activités, des coordonnées GPS afin de pouvoir utiliser la position dans des algorithmes *de proximité*

Architecture : client-side

Dernière modification : 12 fev 2020

Technologies

Nous avons choisi de developper notre application avec **Flutter**, un framework open source developpé par Google (et par la communauté !).

Voici les raisons qui nous ont menées à chosir **Flutter** :

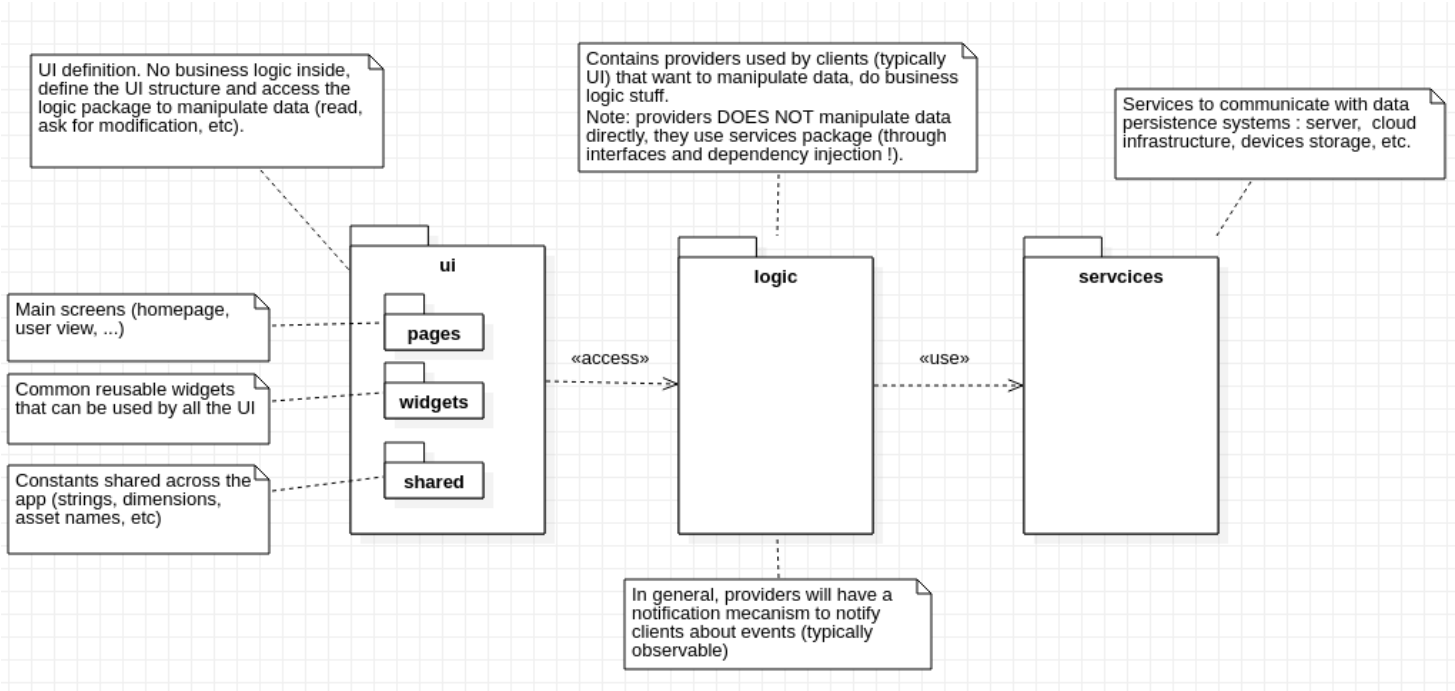
- **Popularité :** Flutter est un framework activement soutenu par sa communauté, avec de nombreuses resources nous permettant de nous documenter (voir à titre d'illustraction la figure ci-dessous)
- **Cross-platform :** permet le developpement sur Android et iOS avec le même *code base* (et aussi Web, Desktop en version bêta)
- **Rapidité de developpement** : Flutter se base sur le langage de programmation *Dart*. C'est un langage qui permet la compilation AOT (Ahead of time) et JIT (Just in time). Lors de developpement l'application est compilé JIT et tourne dans une machine virtuelle pour permettre d'appliquer les changements très rapidement. La version finale, elle, est compilé AOT en langage natif (x64/ARM) pour améliorer les performances et réduire la taille de l'application.
- **La philosophie** : Flutter utilise le *declarative style* pour developper les interface graphique où tout est widget, programmer de cette manière est très appréciable selon nous
- **Framework récent (mais mature)** : Flutter est un framework récent basé sur un langage récent, les architectes / ingénieurs ont donc fait des choix permettant de facilement produire du code clair, bien structuré, suivants des principes de conception élégants

	First release date	Stars (Github)	Contributors	# questions on Stackoverflow
Flutter	Dec 4th, 2018	86.6k	530	34k
React Native	Jan 28, 2015	84.k	2000	67k

Date: Feb 12, 2020

Architecture

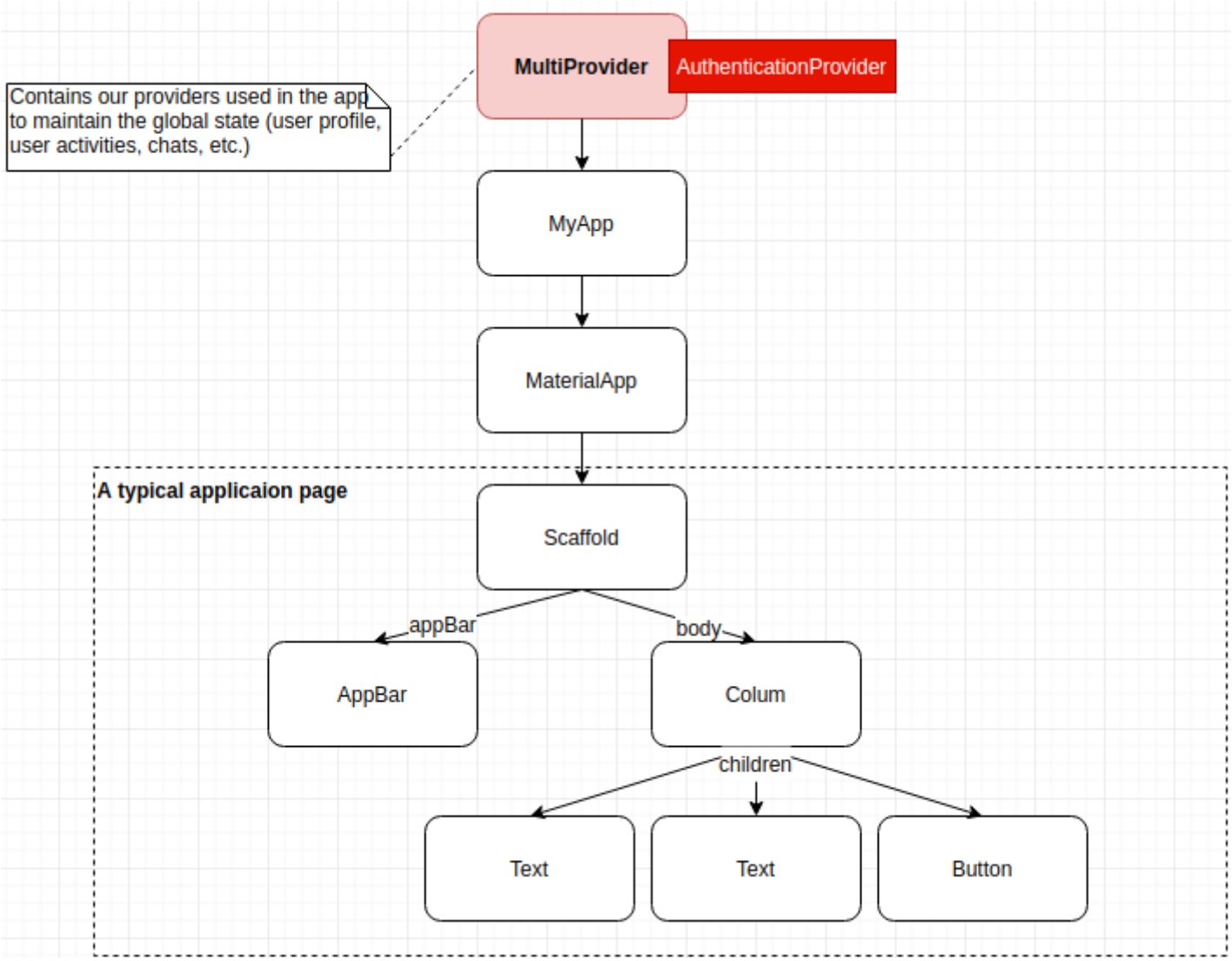
Voici le diagramme de package côté client :



UI

Est contenu dans le dossier `ui/` . Contient les différents Widgets et constants permettant de définir l'interface graphique. Communique avec les providers (contenus dans le package `logic`).

Typiquement voici un arbre représentant une page de notre application :



Les providers sont définis tout en haut de l'arbre, et donc accessible par les enfants.

Logic

Contient nos providers permettant de gérer l'état **global** de notre application, typiquement l'utilisateur. Ils sont définis à la création de l'application (*pas forcément créé à ce moment là, ils seront créés uniquement lorsque quelqu'un en aura besoin)*. Ils seront contenu dans un Widget pouvant les contenir, voir l'abre de widget ci-dessus.

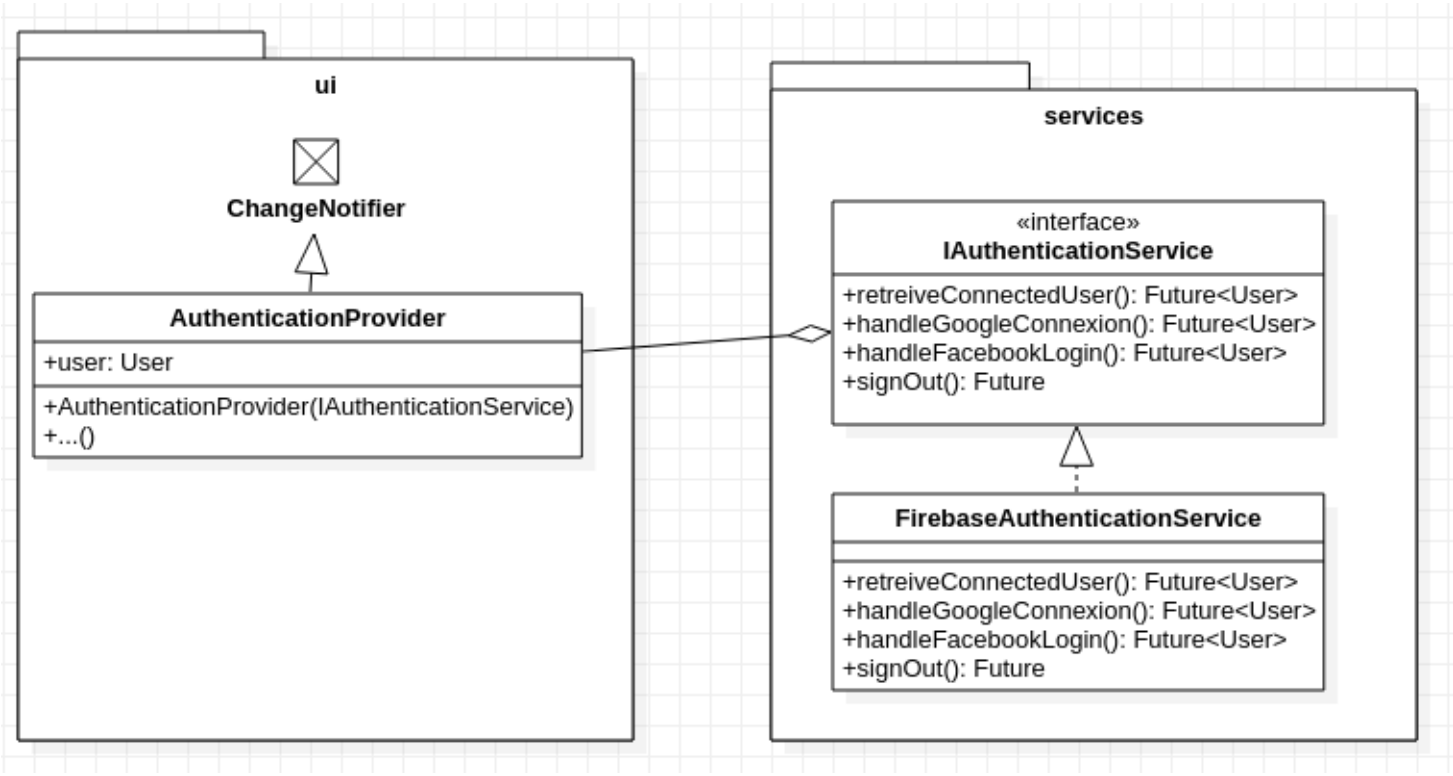
La plupart du temps ils extendront la classe `ChangeNotifier` pour notifier les clients (observer/observable pattern) :

> A class that can be extended or mixed in that provides a change notification API using VoidCallback for notifications.

Services

Permettent la communication avec les serveurs (manipulation de données : écriture, lecture, suppression).

Voici un diagramme de classe montrant l'architecture actuelle avec l' `AuthenticationProvider` et le service permetant l'authentification. Les providers ont connaissance des services via leurs interfaces, et sont injectés dans le constructeur (IoC)



Architecture : server-side

Dernière modification : 12 fev 2020

Technologies

Nous avons choisi de stocker nos données avec **Firebase**, notamment avec les services suivants :

- **Firebase Authentication:** gestion des utilisateurs
- **Cloud Storage:** stockage des fichiers (photos par exemple)
- **Cloud Firestore:** stockage des données (base de données)

Voici les raisons qui ont menées à choisir **Firebase** :

- **Très bien intégré à Flutter via la suite de plugin flutterfire.** Ces plugins qui permettent d'accéder aux services de Firebase sont maintenus par l'équipe de Firebase. Cela permet d'avoir des plugins mis à jour rapidement lors de changements de Firebase.
- **La simplicité d'utilisation** : nous avons une équipe réduite (2 personnes), aucun d'entre nous a déjà eu de l'expérience dans la mise en place / configuration de serveurs. Utiliser un système *serverless* permet de nous abstraire beaucoup de contraintes
- **La gratuité du service pour commencer** : pas besoin d'investir de l'argent, le plan gratuit permet de subvenir à nos besoins durant le developpement
- **Familiarité avec le service** : nous avons déjà quelques connaissances de base de certains services Firebase
- **Mesures de qualités** : Firebase ne permet pas uniquement le stockage de données mais fourni aussi des services permettant d'avoir des mesures de qualités de notre applications (performances, crash, app distribution, etc)
- **Évolution** : Firebase fourni aussi d'autres services qui peuvent être utiles pour la suite : Google Analytics pour analyser le comportement de nos utilisateurs, A/B testing, etc.

Données / structure

Utilisateur

Voir `archi_models.md > #Users` pour davantage information à propos des utilisateurs.

Connexion / inscription

Firebase Authentication est utilisé pour gérer la connexion des utilisateurs via les façons suivantes :

- email / mot de passe
- Facebook
- Google

Données

Voici la structure noSQL des données des utilisateurs :

```
col:users
    doc:uid#1
        - name: text
        - age: integer
        - description: text
        - country: text
        - languages: text
        - photo: text
    doc:uid#2
        ...
    ...
```

Note: l'attribut `photo` des utilisateurs est une URL vers la photo de profile de l'utilisateur. Cette URL correspond à la photo stocké sur *Cloud Storage* voir ci-dessous.

Photo

Les photos des utilisateurs sont stockées dans *Cloud Storage* dans le dossier `users_photos` .

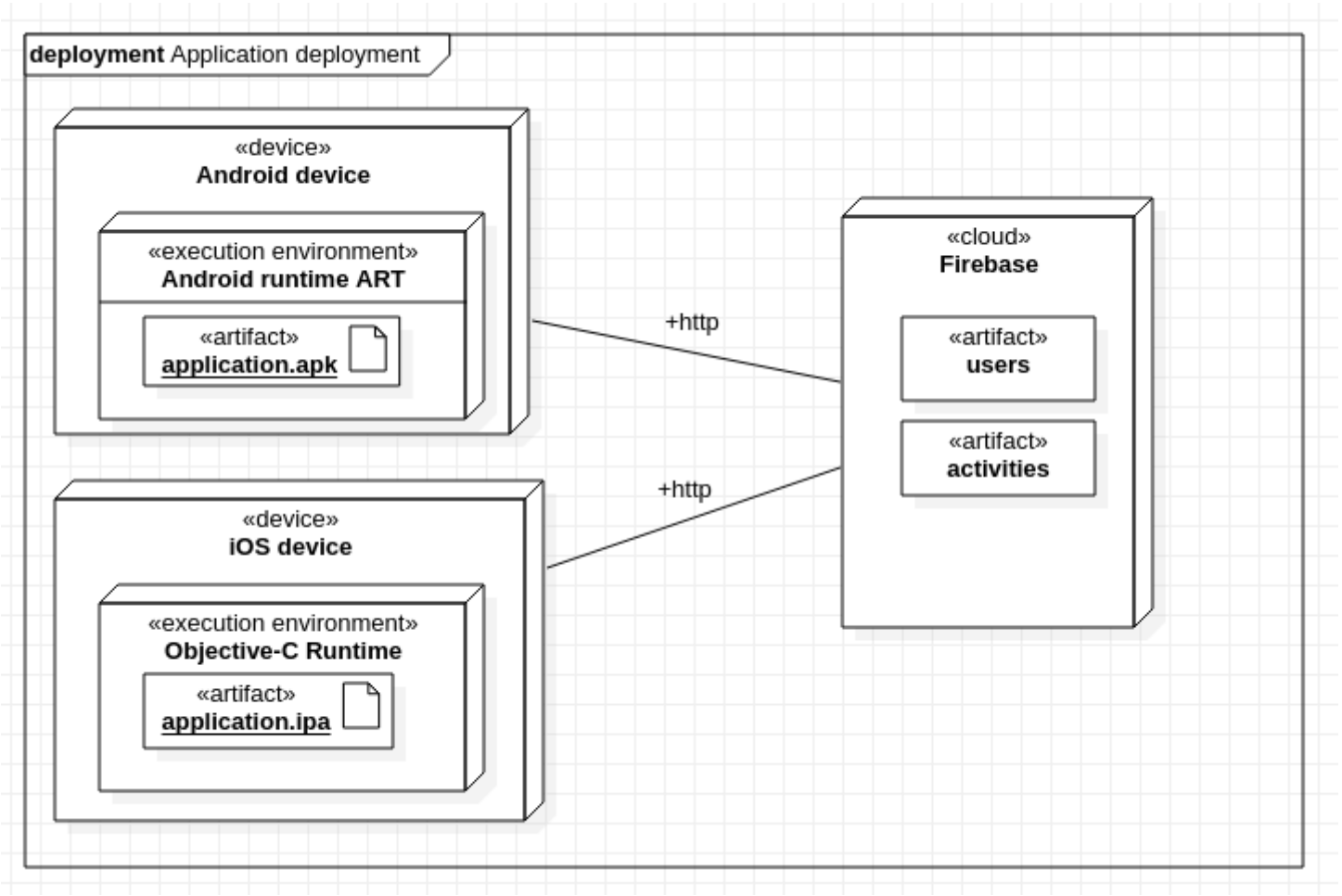
TBD: Définir les limites de tailles d'image / compression / format / etc.

Activité

In progress with the sprint#4

Déploiement

Voici le diagramme UML de deploiement :



La phase du *build* nous génère un fichier `.apk` pour être exécuté sur l'*Android Runtime environment* sur les appareils Android. À terme, cette phase de *build* générera aussi un fichier `.ipa` pour les appareils iOS.

L'application générée communique avec les serveurs Firebase pour accéder aux données / services.

Fonctionnalité : Authentification

Dernière modification : 8 fev 2020

Objectifs

L'objectif de cette semaine est d'implémenter l'authentification au sein de l'application. Toutes les fonctionnalités sont uniquement accessibles aux utilisateurs connectés sur l'application, cette fonctionnalité est donc primoridiale.

Détails

Le processus d'authentification contient les deux aspects suivants :

- vérifier si l'utilisateur est connecté;
- proposer à l'utilisateur de s'authentifier (s'il ne l'est pas).

Deux méthodes d'authentification sont condidérées :

- L'approche traditionnelle via email / mot de passe (plus longue, plus pénible pour l'utilisateur);
- Grâce aux réseaux sociaux tels que *Google* ou *Facebook*. Cette méthode est rapide (une seule action de l'utilisateur) et non contraignante pour la plupart des utilisateurs, elle est donc à privilégier.

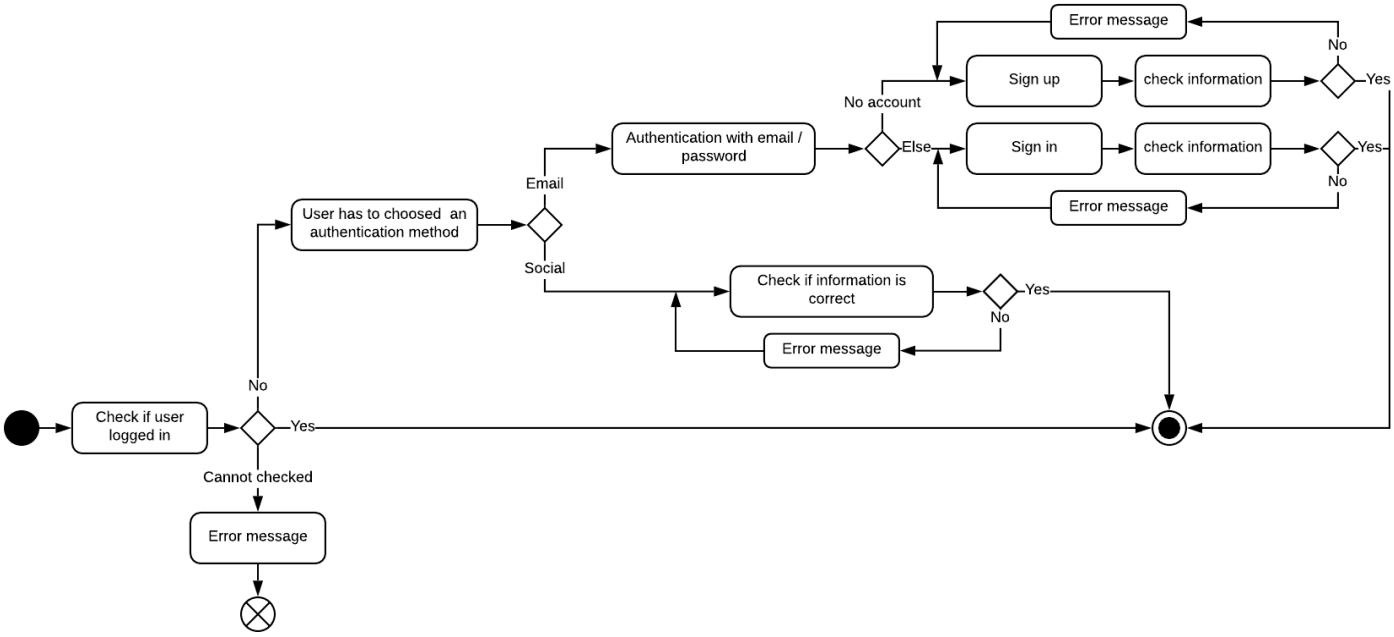
L'approche grâce aux réseaux sociaux ne fait pas la différence entre *connexion* et *inscription*. *Firebase* se charge de créer le compte utilisateur s'il n'existe pas déjà. Par contre, l'approche traditionnelle doit prendre en compte cette différence. Si l'utilisateur choisi la méthode traditionnelle il aura donc le choix entre s'identifier directement ou créer un nouveau compte.

Bien-sûr, si le processus d'authentification échoue, l'utilisateur devra en être informer et sera inviter à retenter son action (si possible). Voici différent scénarios d'échecs :

- Impossible de vérifier si l'utilisateur est connecté ou non (impossible de communique avec *Firebase* par exemple);
- Une erreur est survenue lors de la vérification des informations de l'utilisateur (inscription avec email déjà existant, mot de passe trop faible, adresse mail incorrecte, etc.).

Diagramme d'activités

Voici le diagramme d'activités récapitulant le processus d'authentification d'un utilisateur :



Fonctionnalité : Profil utilisateur

Dernière modification : 8 fev 2020

Voici les éléments que composent le profil utilisateur :

- Nom d'utilisateur (texte libre, 1 ligne)
- Age (nombre)
- Description (texte libre, multi lignes)
- Pays (texte libre, 1 ligne)
- Langues parlées (texte libre, 1 ligne)
- Photo

Visualisation

Consiste à créer la visualisation du profil utilisateur avec les éléments ci-dessus.

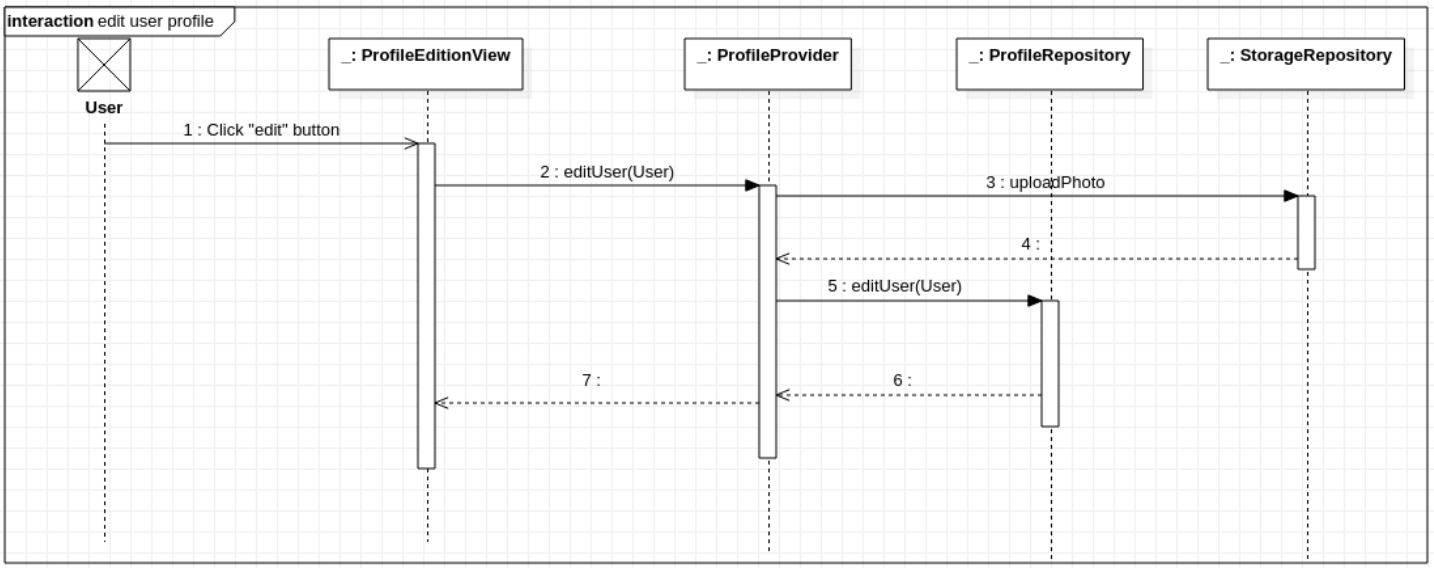
Doit proposer un bouton pour se deconnecter.
Doit proposer un bouton pour supprimer son compte.
Doit proposer un bouton pour éditer son profil (voir ci-dessous).

Edition

Consiste à créer une formulaire pour éditer son profil utilisateur.

Doit respecter les contraintes des différents éléments.
Les champs doivent être pré-remplies avec les informations courantes de l'utilisateur.

Selon l'architecture du projet, voici le diagramme de séquence pour mettre à jour le profil de l'utilisateur:



Références

Voir `archi_firebase.md` pour les détails *côté serveur* (stockage des données, photo).

Fonctionnalité : Explore

Dernière modification : 8 fev 2020

Explore fait partie des fonctionnalités principales de l'application. Une fois l'utilisateur connecté, la page *Explore* est affichée (lors de l'ouverture de l'application normale via le launcher d'application).

La page *Explore* a pour objectif d'afficher les activités pouvant intéresser l'utilisateur. La problématique majeure est donc de savoir **comment choisir ces activités**.

Dans un premier temps, l'algorithme de selection des activités sera basique :

- l'activité devra avoir une localisation *proche* de la localisation de l'utilisateur (proche = rayon d'acceptance - valeur fixée arbitrairement);
- les activités seront triées par ordre de date de création décroissante.

D'un point de vue UI, une liste des activités doit être affichée avec les informations suivantes pour chaque activité :

- Le titre
- Le lieu
- La plage de date (début, fin)
- La photo de l'utilisateur

Dans un premier temps, la page contiendra uniquement la liste des activités. Voici cependant quelques pistes d'amélioration :

- proposer de **changer la localisation** (par exemple si il veut prévoir un futur voyage dans une nouvelle destination)
- **Changer le rayon d'acceptance** des activités par rapport à la localisation effective (celle de l'appareil ou celle choisie)

Planning : Sprint 1

Planning

Durée : 1ère semaine (du 20 au 26 janvier)
Objectifs :

- [x] Mise en place de notre environnement de travail (Git, environnment de travail, système de communication, etc.);
- [x] Mise en place du projet Firebase (côté serveur);
- [x] Choix d'architecture côté client;
- [x] Développement de la fonctionnalité d'authentification (spéfication, implémentation et tests).

Résulats

Environnement de travail

- Système de versionning **Git** + hébergement sur **Github**.
- Environnement de travail : Flutter, Dart, Android SDK, IDE (vscode).
- Outils de *continuous integration* et *continuous delivery* : CodeMagic. Configuration pour automatiser la phase de test + la phase de build automatiquement lors de *push* sur la master origin.

Projet Firebase

- Création du projet Firebase
- Activation du service **Firebase Authentication** avec Google, Facebook, mail/psswd
- Configuration des nos environnement de travail dessus (signature SHA)

Architecture client

Voir `archi_appclient.md` .

Authentification

Voir `feature_authentification.md`

- Spécification
- Boutons d'authentification (UI) : Google, Facebook, Email ;
- Provider qui communique avec le service et notifie l'UI;

- Service qui permet l'authentification avec Google et Facebook

TODO

- **TODO:** Authentification (connexion / inscription) via email / mot de passe Issue#3
- **TODO:** configuration de la phase de deploiement sur Google Play. Issue#8 - Issue#9

Planning : Sprint 2

Planning

Durée: 2ème semaine (du 27 au 2 février)
Objectifs:

- [x] Mise en place de la BDD (configuration + structure)
- [x] Spécification profil utilisateur (visualisation + édition)
- [x] Renseignement du profil utilisateur:
  - [x] Formulaire (photo, nom d'utilisateur, age, description, langues parlées, pays)
  - [x] Visualisation du profil

Résulats

Mise en place de la BDD (configuration + structure)

Voir `archi_firebase.md` .

- Création de la BDD Cloud Firestore
- Strucutre des données
- Règles de sécurités

Renseignement du profil utilisateur:

Voir `archi_models.md`
Voir `feature_profilutilisateur.md` .

- Spécification du modèle "User"
- Spécifications des fonctionnalités de visualisation et d'édition
- Service : service permettant de récupérer les informations de l'utilisateur et de les modifier
- Provider : provider permettant de communiquer avec le service du profil et de notifier l'UI
- UI : affichage du profil utilisateur
- UI : formulaire d'édition du profil

TODO

- **TODO:** Photo de profil (upload / modification) Issue#12

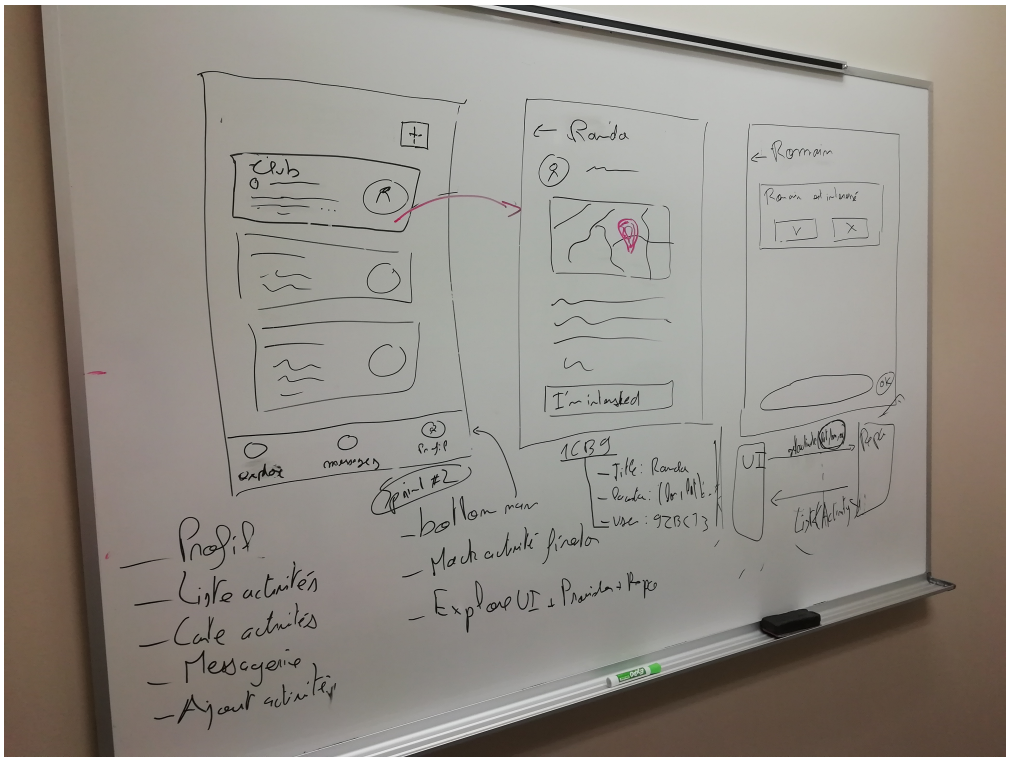
- **TODO:** Bouton de suppression du compte Issue#14

Planning : Sprint 3

Planning

Durée: 3ème semaine (du 3 au 9 février)
Objectifs:

- [x] Layout global de l'application (BottomNavBar)
- [x] Spécification de la page "Explore" + de la structure d'une activité
- [ ] **[WIP]** Page "Explore" (liste des activités à proximité)
    - Layout
    - Provider (avec données fictives)



Résultats

Layout global de l'application (BottomNavBar)

Bottom navigation bar avec les onglets :

- Explore : liste d'activités
- Chats : conversion liées aux activités

- My activities : liste des activités proposées
- Profile : visualisation / édition du profil utilisateur

Page "Explore" + structure d'une activité

- Voir `archi_models.md`
- Voir `feature_explore.md`
- Spécification d'une activités
- Spécification de la page *Explore*
- Service : mock service pour récupérer des activités
- Provider : provider communiquant avec le repository et notify l'UI
- UI : [WIP] page explore

TODO

- **WIP**: finir la page *Explore* (documentation, finitions)

Planning : Sprint 4

Planning

Durée: 4ème semaine (du 10 au 17 février)
Objectifs:

- [ ] Remise préliminaire R1 : diagrammes non terminés (classes, component, deployment) + assembler tous les éléments dans un rapport
- [ ] Finir *Explore* (UI)
- [ ] Vue détaillée d'une activité (ActivityView)
- [ ] Compléter le document d'architecture de la base de données `archi_firebase.md` avec le système d'activités
- [ ] Remplacer l'implémentation `MockActivitiesRepository` par une implémentation utilisant Firebase (`FirestoreActivitiesRepository`)
- [ ] Nom de l'application + configuration de CodeMagic (CI/CD) pour le deploiement automatiser sur le Play Store (pour les *internal testers*)

Résultats

In progress