KEEP IN TOUCH

Projet de Génie Logiciel

Amel Dussier, Bastien Clément, Antoine Drabble et Guillaume Serneels

Table des matières

Introduction 4
Analyse 5
Fonctionnalités 5
Partage des responsabilités entre le serveur et le client 6
Un diagramme d'activité général 7
Cas d'utilisation 8
Diagramme général de contexte 8
Description des acteurs 9
Description des cas d'utilisation 10
Modèles de domaine 19
Client 19
Serveur 20
Base de données 21
Modèle conceptuel 21
Conception du projet 22
Protocole d'échange entre le client et le serveur 22
Diagrammes de classes 23
Serveur 23
Base de données 24
Modèle relationnel 24
Ebauches des interfaces utilisateurs 25
Implémentation du projet 27
Technologies utilisées 27
Android 27
Scala 27
Client 28
Brève descriptions des packages et des liens entre eux 29
Gestion du projet 37
Rôle des participants 37
Représentant des utilisateurs 37
Chef de projet 37
Analyste 37

Software architect 37
Programmeurs 38
Responsable des tests 38
Responsable de la configuration 38
Plan des itérations initial 39
Itération 1 39
Itération 2 40
Itération 3 41
Itération 4 42
Itération 5 43
Itération 6 44
Bilan des itérations 45
Itération 1 45
Itération 2 47
Itération 3 49
Itération 4 51
Itération 5 53
Itération 6 55
Stratégie de tests 56
Fonctionnement d'Espresso 57
Problèmes rencontrés 57
Tests manuels effectués 58
Stratégie d'intégration du code 60
Etat des lieux 61
Ce qui fonctionne 61
Ce qu'il resterait à développer 61
Autocritique 62
Planification 62
Android 62
Conclusion 63
Améliorations possibles 63
Annexe 64
Table des illustrations 64
Manuel d'utilisation 65
Installation 65
Utilisation 65

Introduction

Dans le cadre de notre projet de Génie Logiciel, nous allons réaliser une application de chat pour clients mobiles, ressemblant à des produits comme WhatsApp ou Telegram.

Cette application s'appellera « Keep in touch », et proposera dans sa première version toutes les fonctionnalités permettant une communication rapide et conviviale : possibilité de trouver ses amis et de rester en contact avec eux, discussion avec un ou plusieurs contacts via des groupes, discussions publiques ouvertes à tout le monde, etc.

L'aspect sécurité et protection des utilisateurs sera également présente : possibilité de signaler des comportements inadéquats, de déléguer l'administration d'un groupe, ou encore de bloquer des contacts si nécessaire.

Le but de notre projet est d'arriver à développer un produit fini, utilisable et qui soit suffisamment modulaire pour être étendu facilement grâce à des ajouts de fonctionnalités à l'avenir. Bien sûr, nous aimerions également que le produit soit esthétique et agréable à utiliser, afin que les utilisateurs partagent l'enthousiasme que nous avons eu lors de ce projet.

Ci-dessous le logo de notre première version (réalisé par Michael Gaio), qui rappelle le nom de l'application grâce à son initiale, et qui symbolise également l'interaction sociale que « Keep in touch » veut promouvoir :

Figure 1 : Logo de l'application

Analyse

Fonctionnalités

La première fonctionnalité proposée à l'utilisateur sera bien sûr la création d'un compte, avec un login unique et un mot de passe. Ce compte lui permettra de s'authentifier dans l'application, et sera lié à toutes les données de l'utilisateur (contacts, messages, etc.).

Une fois connecté, l'utilisateur aura le choix entre trois types de chat différents :

- **Chat privé**
  Ce chat permettra de communiquer avec un contact de son choix. La première version permettra de communiquer par message texte puis, si le temps le permet, nous étudierons la possibilité d'implémenter des fonctionnalités supplémentaires (la communication audio, des discussions chiffrée de bout en bout avec un système utilisant des clés publiques/privées, etc.).

- **Chat groupe**
  Ce chat permettra de communiquer avec plusieurs contacts en même temps. Un utilisateur pourra créer un groupe et lui donner un nom. Il pourra y inviter les contacts de son choix. Toutes les personnes ajoutées pourront envoyer des messages, qui seront lisibles par tous les membres du groupe. Le créateur du groupe en sera l'administrateur et pourra y ajouter ou supprimer des membres. Il pourra également désigner d'autres administrateurs pour ce groupe.

- **Chat publique**
  Ce chat est similaire au chat de groupe, mais il a la particularité d'être ouvert à tout le monde. Les membres ne sont pas invités mais créent ou rejoignent eux-mêmes un chat public existant.

En cas de problème, une fonctionnalité de report de messages permettra aux utilisateurs de signaler un message abusif. Tous les reports seront envoyés à l'administrateur de l'application et à l'administrateur du groupe s'il ne s'agit pas d'un chat privé. Les reports envoyés à l'administrateur seront stockés dans une base de données et seront accessibles via une interface web.

Un utilisateur pourra également en bloquer un autre afin de ne plus recevoir de messages de sa part.

Partage des responsabilités entre le serveur et le client

Le serveur s'occupe de gérer l'ensemble des opérations, le client n'est qu'une vue des données du système. Il affiche les informations que l'utilisateur demande et transmet les actions qu'il souhaite effectuer. Le serveur s'assure qu'il en ait l'autorisation.

Le serveur fonctionne en permanence. Le client peut à tout moment s'y connecter en démarrant l'application et en fournissant ses informations de connexion (nom d'utilisateur et mot de passe).

Si l'application n'est pas lancée, l'utilisateur est considéré comme déconnecté. Après la première connexion, le client maintient un cookie (token) de session lui permettant de se reconnecter rapidement.

Un diagramme d'activité général

Le schéma ci-dessous montre l'architecture générale de notre solution, ainsi que la répartition des différentes responsabilités :

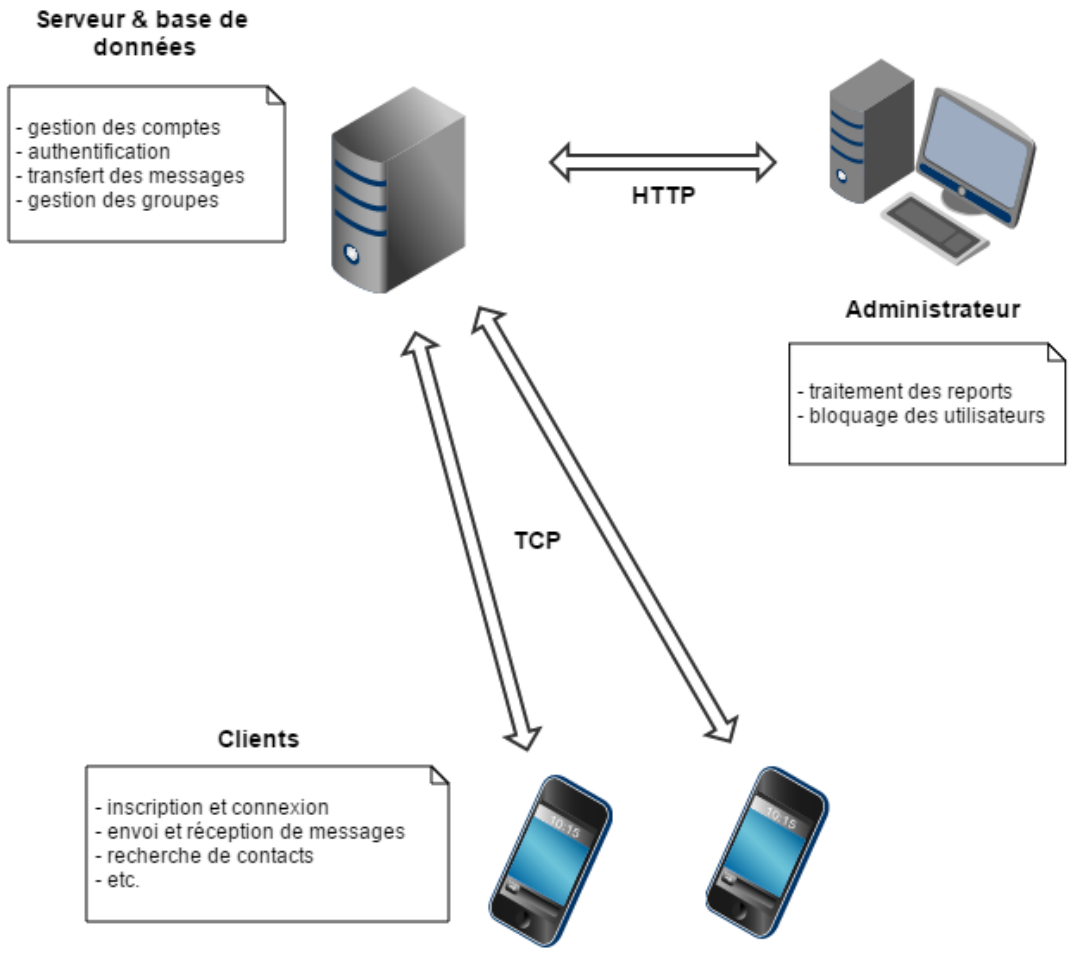

Figure 2 : Diagramme d'activité général

Le client permettra aux utilisateurs d'accéder à de nombreuses fonctionnalités. Seules les principales ont été mentionnées sur ce diagramme afin de ne pas le surcharger.

Cas d'utilisation

Diagramme général de contexte

Ci-dessous le diagramme général de contexte, avec les différents acteurs et cas d'utilisation que nous avons identifiés :

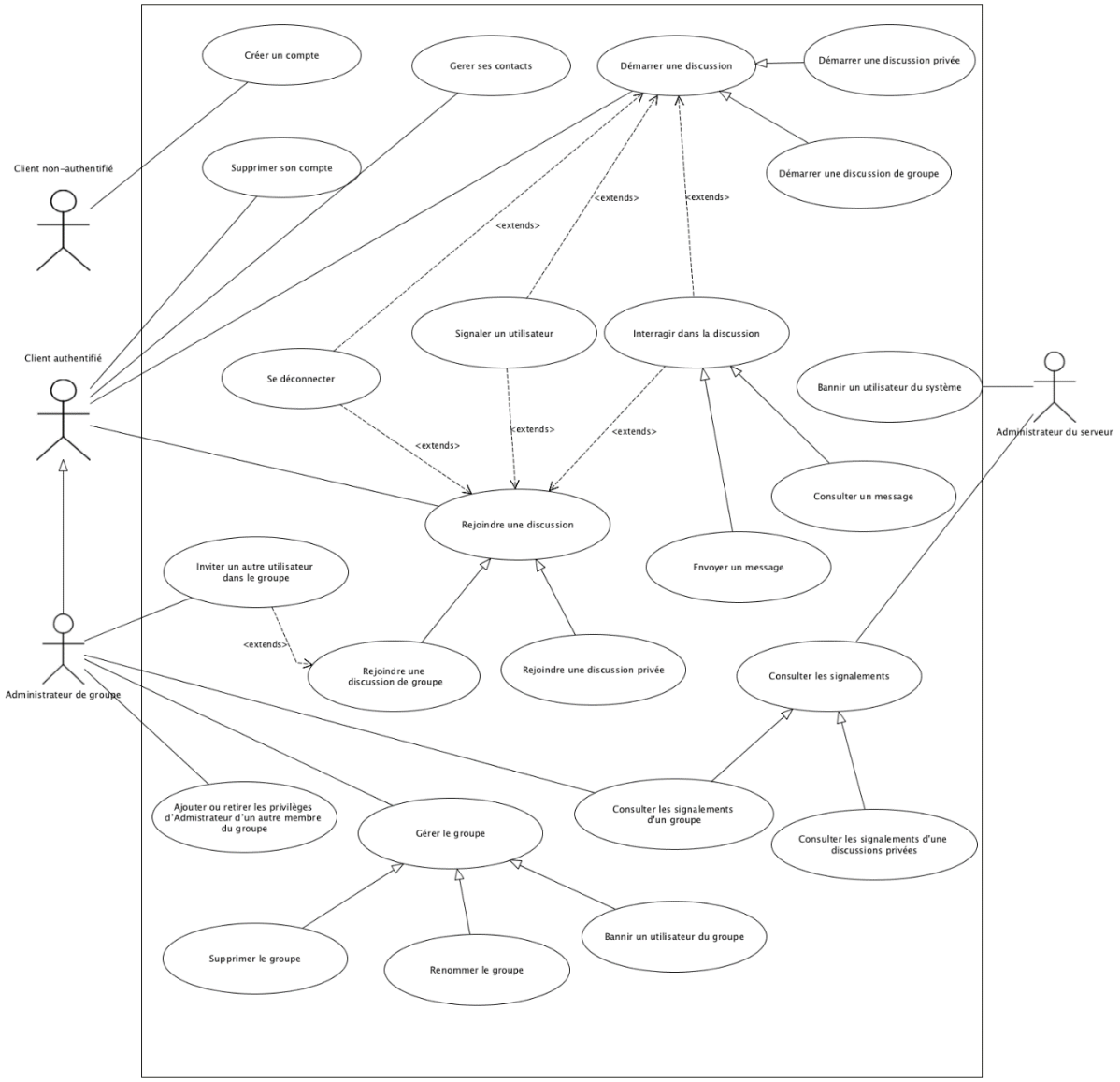

Figure 3 : Diagramme de contexte général

Description des acteurs

Acteurs principaux

- Client non-authentifié
  Il devra soit créer un compte soit s'authentifier pour se transformer en client authentifié.

- Client authentifié
  C'est l'acteur qui représente les utilisateurs qui possèdent un compte et qui sont connectés.

- Administrateur de groupe
  Il est notifié des reports des discussions de son groupe et peut gérer les membres du groupe. Le rôle d'administrateur de groupe étend le rôle de client authentifié.

Acteurs secondaires

- Administrateur du système
  Il est notifié des reports de groupes et de discussions privées et peut bannir des utilisateurs de l'application. Cet acteur n'a pas de rôle de client.

Description des cas d'utilisation

Prérequis : Authentification

Acteur(s) : Client non-authentifié

Description : Un client non-authentifié peut se connecter avec un compte existant

Scénario principal :

1. L'utilisateur ouvre l'application
2. L'utilisateur accède à l'interface de connexion
3. L'utilisateur fournit un pseudonyme et un mot de passe
4. Le serveur accepte les informations de connexion
5. L'application authentifie l'utilisateur

Scénarios d'échec :

a) Le serveur n'est pas disponible : l'application signale l'erreur et l'action est abandonnée
b) Les informations de connexion sont fausses : l'application signale l'erreur et l'action est abandonnée

Ajouter ou retirer les privilèges d'Administrateur d'un autre membre du groupe

Acteur(s) : Administrateur de groupe

Description : Un administrateur de groupe peut promouvoir ou dégrader un autre du groupe du rang d'administrateur de groupe

Scénario principal :

1. L'administrateur de groupe accède à une discussion de groupe qu'il administre
2. L'administrateur affiche la liste des membres du groupe
3. L'administrateur ouvre le menu contextuel du membre du groupe qu'il veut promouvoir ou dégrader
4. L'administrateur choisit « Promouvoir » ou « Dégrader»

Scénarios d'échec :

a. Le serveur n'est pas disponible : l'application signale l'erreur et l'action est abandonnée
b. Le pseudonyme de l'utilisateur à ajouter n'existe pas : l'application signale l'erreur et l'action est abandonnée

Ajouter un utilisateur dans un groupe

Acteur(s) : Administrateur de groupe

Description : Un administrateur de groupe peut ajouter un utilisateur au groupe

Scénario principal :

1. L'administrateur de groupe accède à une discussion de groupe qu'il administre
2. L'administrateur de groupe clique sur "Gérer la discussion"
3. L'administrateur de groupe clique sur "Ajouter un utilisateur"
4. L'administrateur de groupe choisit un de ces contacts enregistrés, ou entre le pseudonyme de l'utilisateur à ajouter
5. Le serveur accepte l'ajout de l'utilisateur

Scénarios d'échec :

a) Le serveur n'est pas disponible : l'application signale l'erreur et l'action est abandonnée
b) Le pseudonyme de l'utilisateur à ajouter n'existe pas : l'application signale l'erreur et l'action est abandonnée

Bannir un utilisateur du groupe

Acteur(s) : Administrateur de groupe

Description : Un administrateur de groupe peut supprimer un utilisateur d'une discussion de groupe

Scénario principal :

1. L'administrateur de groupe accède à une discussion de groupe qu'il administre
2. L'administrateur de groupe clique sur "Gérer le groupe"
3. L'application affiche la liste des membres
4. L'administrateur de groupe sélectionne un utilisateur membre du groupe
5. L'administrateur de groupe clique sur "Supprimer du groupe"
6. L'application demande confirmation
7. Le serveur autorise la suppression de l'utilisateur de la discussion publique
8. L'application met à jour la liste des utilisateurs membres du groupe

Scénarios d'échec :

a. Le serveur n'est pas disponible : l'application signale l'erreur et l'action est abandonnée

Bannir un utilisateur du système

Acteur(s) : Administrateur du serveur

Description : L'administrateur du serveur peut supprimer un utilisateur du système

Scénario principal :

1. L'administrateur du système accède à l'interface de gestion du système
2. L'administrateur clique sur « Bannir un utilisateur»
3. L'administrateur choisit parmi la liste des utilisateurs existants celui à bannir.
4. L'administrateur valide son choix

Scénarios d'échec :

a) Le serveur n'est pas disponible : l'application signale l'erreur et l'action est abandonnée

Charger les messages de l'historique

Description : Un client authentifié peut consulter les messages d'une discussion

Prérequis : Authentification

Scénario principal :

1. L'utilisateur accède à une conversation
2. L'utilisateur remonte (scroll) dans la discussion
3. L'application met automatiquement à jour, au fur et à mesure du scroll, le contenu de la conversation

Scénarios d'échec :

a. Le serveur n'est pas disponible : l'application signale l'erreur et l'action est abandonnée

Créer un groupe de discussion public, consulter les signalements d'un groupe

Acteur(s) : Administrateur du système, Administrateur de groupe

Description : Un administrateur peut consulter les reports d'un groupe

Scénario principal :

1. L'administrateur accède à une discussion de groupe qu'il administre
2. L'administrateur clique sur "Gérer la discussion"
3. L'administrateur clique sur "Voir les signalements"

Scénarios d'échec :

a. Le serveur n'est pas disponible : l'application signale l'erreur et l'action est abandonnée

Consulter les signalements d'une discussion privée

Acteur(s) : Administrateur du système

Description : Un administrateur du système peut consulter les reports d'une discussion privée

Scénario principal :

1. L'administrateur du système accède à l'interface de gestion du système
2. L'administrateur du système clique sur "Voir les signalements"

Scénarios d'échec :

a. Le serveur n'est pas disponible : l'application signale l'erreur et l'action est abandonnée

Consulter un message

Acteur(s) : Client authentifié

Description : Un client authentifié peut consulter les messages d'une discussion

Prérequis : Authentification

Scénario principal :

1. L'utilisateur accède à une conversation
2. L'application met automatiquement à jour le contenu de la conversation

Scénarios d'échec :

a. Le serveur n'est pas disponible : l'application signale l'erreur et l'action est abandonnée

Créer un compte

Acteur(s) : Client non-authentifié

Description : Un client non-authentifié peut créer un nouveau compte

Prérequis : Authentification

Scénario principal :

1. L'utilisateur fournit un pseudonyme et un mot de passe (avec confirmation
2. Le serveur accepte et crée le nouveau compte
3. L'application authentifie l'utilisateur

Scénarios d'échec :

a. Le serveur n'est pas disponible : l'application signale l'erreur et l'action est abandonnée
b. Le compte existe déjà : l'application signale l'erreur et l'action est abandonnée

Démarrer une discussion de groupe

Acteur(s) : Client authentifié

Description : Un client authentifié peut créer une discussion publique

Pré-requis : Authentification

Scénario principal :

1. L'utilisateur clique sur "Créer un groupe"
2. L'utilisateur entre le nom du groupe qu'il veut créer
3. Le serveur autorise l'utilisateur à créer la discussion
4. L'application ouvre la discussion de groupe pour l'utilisateur

Scénarios d'échec :

a. Le serveur n'est pas disponible : l'application signale l'erreur et l'action est abandonnée
b. Un groupe public avec ce nom existe : l'application signale l'erreur et l'action est abandonnée

Démarrer une discussion privée

Acteur(s) : Client authentifié

Description : Un client authentifié peut créer une discussion privée avec un autre utilisateur

Pré-requis : Authentification

Scénario principal :

1. L'utilisateur clique sur "Discussion privée"
2. L'utilisateur choisit un de ces contacts enregistrés, ou entre le pseudonyme de l'utilisateur avec qui il veut avoir une discussion
3. Le serveur autorise l'utilisateur à créer la discussion
4. L'application ouvre la discussion privée pour l'utilisateur

Scénarios d'échec :

a. Le serveur n'est pas disponible : l'application signale l'erreur et l'action est abandonnée
b. Aucun utilisateur avec ce pseudonyme n'existe : l'application signale l'erreur et l'action est abandonnée

Envoyer un message

Acteur(s) : Client authentifié

Description : Un client authentifié peut envoyer un message dans une discussion

Prérequis : Authentification

Scénario principal :

1. L'utilisateur accède à une conversation
2. L'utilisateur saisit un message
3. L'utilisateur clique sur "Envoyer"
4. Le serveur accepte l'envoi du message
5. L'application met à jour le contenu de la discussion pour chaque membre de la discussion

Scénarios d'échec :

a) Le serveur n'est pas disponible : l'application signale l'erreur et l'action est abandonnée

Gérer ses contacts

Acteur(s) : Client authentifié

Description : Un client authentifié peut gérer ses contacts (ajout, suppression, blocage)

Prérequis : Authentification

Scénario principal :

1. L'utilisateur peut cliquer soit sur "Ajouter un contact" ou sélectionner un contact, puis cliquer sur "Supprimer" ou "Bloquer"
   1.1. Si « Ajouter un contact » :
      1.1.1. L'utilisateur entre le pseudonyme d'un utilisateur
      1.1.2. Le serveur accepte l'ajout de contact
   1.2. Si « Supprimer ou Bloquer »:
      1.2.1. L'application demande confirmation
      1.2.2. Le serveur accepte la suppression ou le blocage du contact

Scénarios d'échec :

a. Le serveur n'est pas disponible : l'application signale l'erreur et l'action est abandonnée
b. Le pseudonyme de l'utilisateur à ajouter n'existe pas : l'application signale l'erreur et l'action est abandonnée

Inviter un autre utilisateur dans le groupe

Acteur(s) : Client authentifié

Description : Un client authentifié et membre d'un groupe, peut ajouter un utilisateur dans ce groupe

Prérequis : Authentification, Être membre du groupe

Scénario principal :

1. L'utilisateur clique sur le groupe dans lequel il veut ajouter un utilisateur dans sa liste de discussions
2. L'utilisateur clique sur « Ajouter un utilisateur »
3. L'utilisateur sélection un utilisateur depuis sa liste de contacts
4. L'utilisateur valide son choix

Scénarios d'échec :

a. Le serveur n'est pas disponible : l'application signale l'erreur et l'action est abandonnée
b. L'utilisateur est banni du groupe et ne peut pas être ajouté : l'application signale l'erreur et l'action est abandonnée

Rejoindre une discussion de groupe ou privée

Acteur(s) : Client authentifié

Description : Un client authentifié peut

1. rejoindre une discussion de groupe s'il y a été invité
2. rejoindre une discussion privée

Prérequis : Authentification

Si discussion de groupe :

Un autre utilisateur membre du groupe nous y a invité

Si discussion privée:

Un autre utilisateur a démarré une discussion privée avec nous

Scénario principal :

1. L'utilisateur constate l'apparition d'une nouvelle discussion dans sa liste de discussions
2. L'utilisateur clique sur la discussion pour la rejoindre
3. L'application affiche la discussion

Scénarios d'échec :

a. Le serveur n'est pas disponible : l'application signale l'erreur et l'action est abandonnée

Renommer le groupe

Acteur(s) : Administrateur de groupe

Description : L'Administrateur d'un groupe peut le renommer

Prérequis : Authentification

Scénario principal :

1. L'administrateur choisit le groupe qu'il désire supprimer parmi sa liste de discussions
2. L'administrateur de groupe clique sur "Gérer le groupe"
3. L'administrateur clique sur « Renommer le groupe »
4. L'administrateur saisit le nouveau nom du groupe
5. L'administrateur confirme le changement

Scénarios d'échec :

a. Le serveur n'est pas disponible : l'application signale l'erreur et l'action est abandonnée

Signaler un utilisateur

Acteur(s) : Client authentifié

Description : Un client authentifié peut signaler un autre utilisateur

Prérequis : Authentification

Scénario principal :

1. L'utilisateur sélectionne un ou plusieurs messages
2. L'utilisateur clique sur "Signaler"
3. L'application demande confirmation
4. Le serveur accepte le signalement

Scénarios d'échec :

a. Le serveur n'est pas disponible : l'application signale l'erreur et l'action est abandonnée

Supprimer le groupe

Acteur(s) : Administrateur de groupe

Description : L'Administrateur d'un groupe peut le supprimer

Prérequis : Authentification

Scénario principal :

1. L'administrateur choisit le groupe qu'il désire supprimer parmi sa liste de discussions
2. L'administrateur de groupe clique sur "Gérer le groupe"
3. L'administrateur clique sur « Supprimer le groupe »
4. L'administrateur valide son choix

Scénarios d'échec :

b. Le serveur n'est pas disponible : l'application signale l'erreur et l'action est abandonnée

Supprimer son compte

Acteur(s) : Client authentifié

Description : Un client authentifié peut supprimer son compte

Prérequis : Authentification

Scénario principal :

1. L'utilisateur clique sur "Gérer mon compte"
2. L'utilisateur clique sur "Supprimer mon compte"
3. L'application demande confirmation
4. Le serveur accepte la suppression du compte et supprime les infos du compte présentes sur la base de données
5. L'application déconnecte l'utilisateur

Scénarios d'échec :

c. Le serveur n'est pas disponible : l'application signale l'erreur et l'action est abandonnée

Modèles de domaine

Client

Ci-dessous le modèle de domaine représentant le côté client :

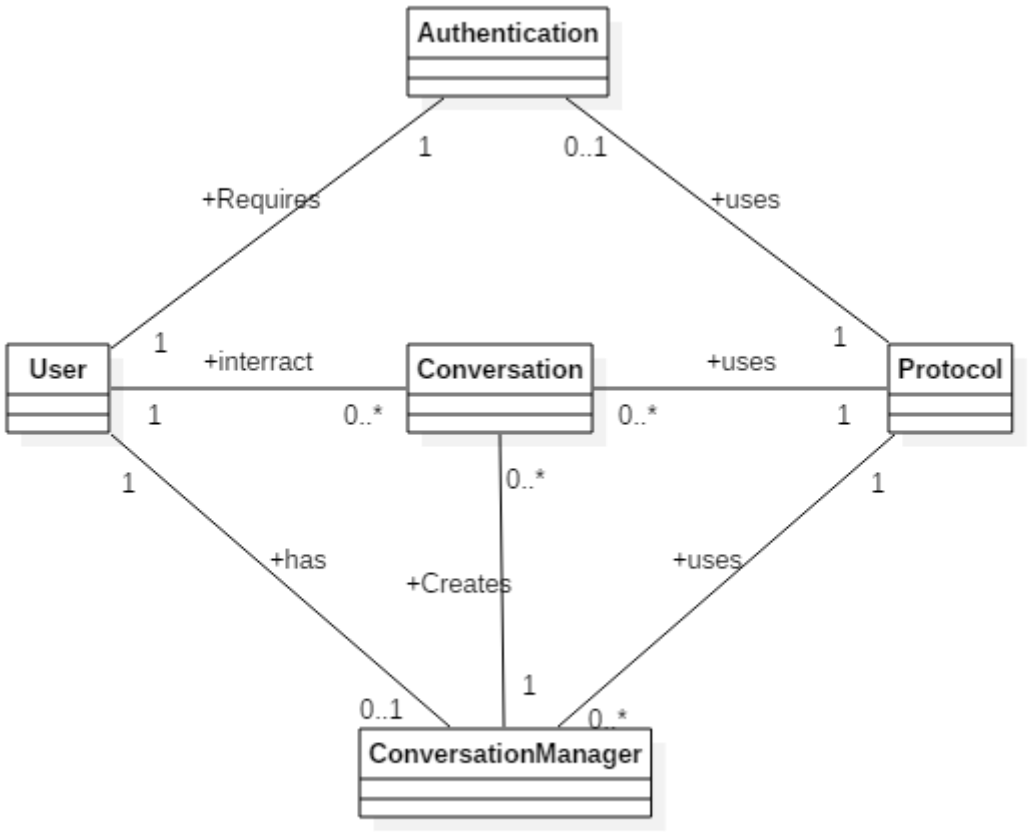

Figure 4 : Modèle de domaine client

Serveur

Ci-dessous le modèle de domaine représentant le côté serveur :

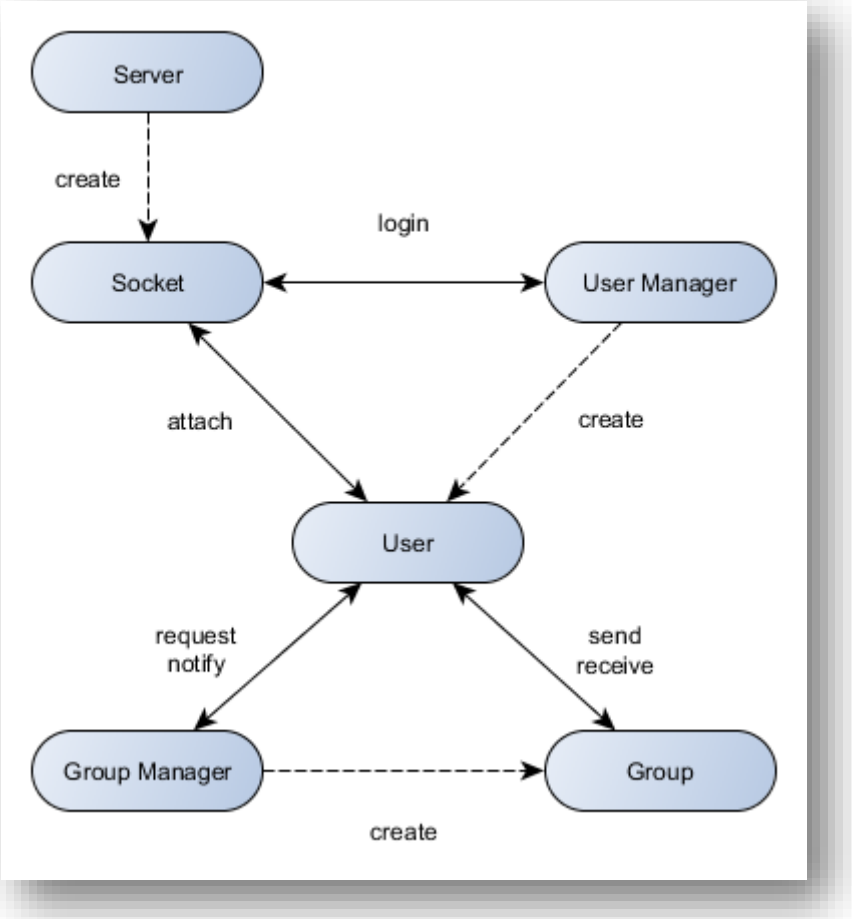

Figure 5 : Modèle de domaine serveur

Base de données

La base de données définit des utilisateurs qui peuvent être membre de conversations. Les conversations privées seront différenciées des conversations de groupe par l'attribut type dans conversation.

Une conversation contient plusieurs messages qui sont liés à un utilisateur et les messages peuvent être reportés. Seuls les administrateurs de groupes (membre dont l'attribut admin est à true) pourront voir les messages reportés.

Un utilisateur peut en bloquer un autre via l'association block.

Modèle conceptuel

Ci-dessous notre modèle conceptuel (ou entité-association) de notre base de données :

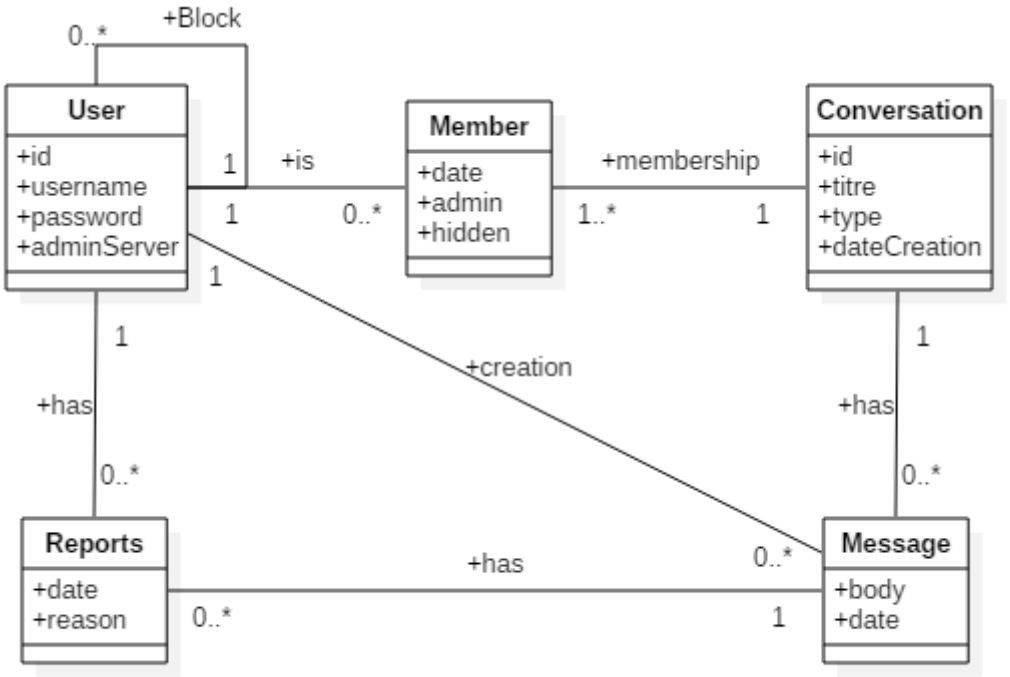

Figure 6 : Modèle conceptuel de la base de données

Conception du projet

Protocole d'échange entre le client et le serveur

Le protocole de communication client-serveur devra permettre les commandes suivantes :

Client -> Serveur

- Demande de création de compte
- Validation/Refus de création de compte
- Demande de tentative de connexion
- Validation/Refus de tentative de connexion
- Demande d'ajout d'un contact
- Validation/Refus d'ajout de contact (par exemple s'il est bloqué ou s'il y a une erreur) (Ajout d'un lien dans les deux sens entre les deux contacts, création de discussions vide)
- Demande de création de groupe
- Validation/Refus de création de groupe
- Envoi d'un message à un contact/groupe
- Validation/Refus d'envoi du message
- Demande de déconnection
- Validation/Refus de déconnection

Serveur -> Client

- Envoi d'un message au client
- Acceptation/Refus de réception
- Création d'un groupe dont le client est membre
- Acceptation/Refus de création

Une classe Java Protocole va définir les différentes commandes. Elle sera utilisée du côté serveur et client. Une commande sera représentée sur 3 bytes. Le premier byte va définir la commande à utiliser. On aura donc 255 commandes différentes possibles ce qui devrait suffire. Les 2 bytes suivants définiront la longueur des messages soit une longueur maximum de 65535. Le message sera situé directement après ces 3 bytes.

Diagrammes de classes

Serveur

Todo : Diagramme + description

Base de données

Modèle relationnel

Ci-dessous le schéma relationnel de notre base de données, qui se base sur le modèle conceptuel présenté au chapitre Analyse :

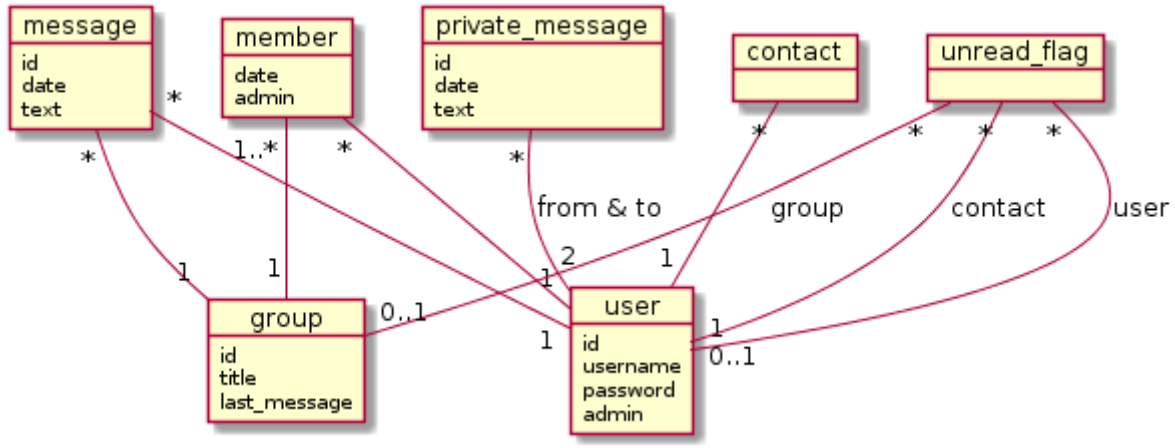

Figure 7 : Modèle relationnel de la base de données

Ebauches des interfaces utilisateurs

Cette interface représente l'écran d'acceuil, qui permet à l'utilisateur de se connecter :

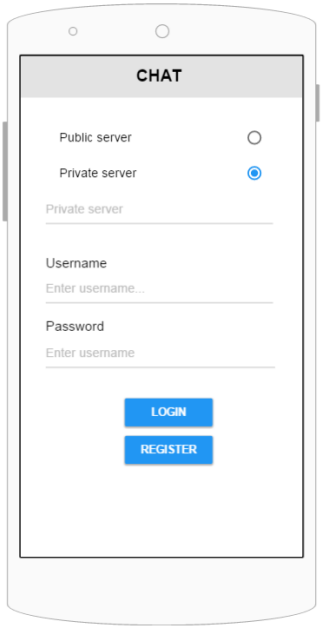

Figure 8 : Concept d'interface pour la fenêtre de login

Ci-dessous l'interface qui permet de gérer un groupe, avec ses membres, etc. :

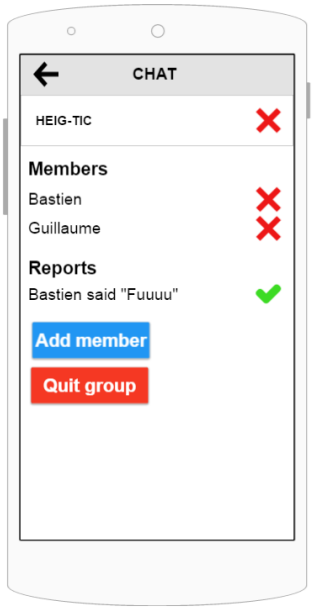

Figure 9 : Concept d'interface pour l'écran de gestion d'un groupe

Ici on peut voir ce que donnera l'afficahge d'une conversation, avec les messages reçus et envoyés :

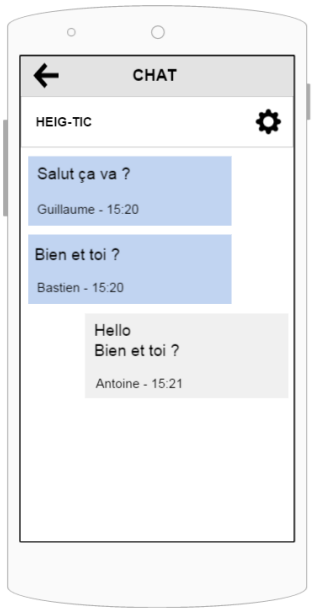

Figure 10 : Concept d'interface pour la fenêtre de discussion

Cette dernière image montre l'interface qui listera les groupes de discussions :

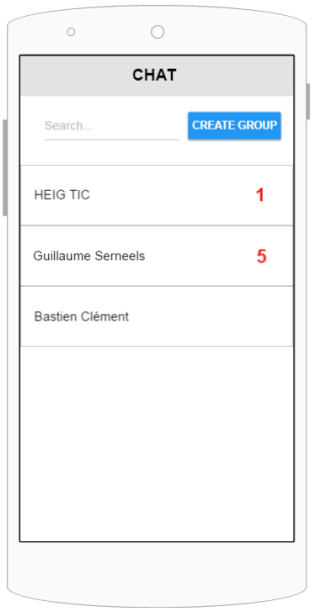

Figure 11 : Concept d'interface pour la fenêtre d'affichage de groupes

Implémentation du projet

Technologies utilisées

Android

Android est un système d'exploitation très utilisé avec environ 76% du marché des téléphones portables aujourd'hui. Il possède de très bon outil tel que l'IDE Android Studio qui est basé sur IntelliJ, et le développement se fait avec le langage Java.

La structure d'une application Android est la suivante :

- Un dossier nommé « java » contenant les différentes classes de l'application.
- Un dossier « res » contenant les différentes ressources. Par exemple les images dans plusieurs résolutions, les dispositions des vues (au format XML), les menus, les différentes valeurs utiles à l'interface telle que les couleurs, les dimensions, les chaine de caractères dans plusieurs langues si nécessaires, etc.
- Un fichier AndroidManifest.xml décrivant l'application. Par exemple son titre, la liste des vues, etc.

Les différentes vues de l'application sont gérées par un fichier XML et une classe qui hérite de la classe « Activity » et qu'on nomme donc activité. Le fichier XML définit les éléments présents dans la vue avec toutes leurs caractéristiques ainsi que la disposition de ces éléments. La classe activité va gérer toute la partie dynamique des éléments. De cette façon, on peut facilement séparer la vue du contrôleur.

Une activité va dialoguer avec le modèle afin de mettre à jour les éléments de la vue. Il peut utiliser des classes Android telles que la classe Adapter qui permet de gérer la mise à jour et les événements des listes.

Scala

Nous avons choisi Scala pour implémenter le serveur parce que c'est le langage le plus agréable pour travailler avec les framework Play et Akka.

À la base, nous souhaitions réaliser une architecture orientée acteurs côté serveur (en se basant sur Akka) accompagné d'un mini site web pour la gestion administrateur (avec Play). Ces deux frameworks fonctionnant très bien ensemble, cela semblait un choix adéquat.

Au final, nous avons choisi d'utiliser une API REST côté serveur pour la communication avec le client, nous avons donc abandonné l'utilisation d'Akka.

Client

Notre application est composée des paquets suivant :

- Activities
- Adapters
- Communications
- Fragments
- Interfaces
- Models
- Services
- Utilities

Les dossiers contenants des ressources importantes sont les suivants :

- Layout
  - Contient les layouts de toutes les vues
- Menu
  - Contient les éléments du menu
- Drawable
  - Contient le logo de l'application
- Values
  - Contient toutes les chaines de caractères affichées par les vues. Cette séparation permet de facilement modifier une chaine à plusieurs endroits et de pouvoir passer d'une langue à une autre sans problème.

Brève descriptions des packages et des liens entre eux

Activities

Toutes les vues de l'application sont contenues dans ce package. Elle utilise le package models (décris ci-dessous) afin de récupérer la liste des utilisateurs, groupes, messages, etc. Elle utilise le package adapter afin de gérer les listes de messages, contacts. Elle reçoit des notifications du package services afin de récupérer les événements du serveur.

La liste exhaustive des activités est la suivante :

- ContactAddActivity
  - Ajout de contact
- ContactDiscussionActivity
  - Affichage et envoi de message avec un contact
- ContactEditActivity
  - Modification d'un contact
- GroupAddMemberActivity
  - Ajout d'un membre dans un groupe
- GroupCreateActivity
  - Création d'un groupe
- GroupDiscussionActivity
  - Affichage et envoi de message dans un groupe
- GroupEditActivity
  - Modification d'un groupe
- LoginActivity
  - Connection à l'application
- MainActivity
  - Contient les deux fragments (décris ci-dessous) qui permettent d'afficher la liste des contacts et des groupes
- RegisterActivity
  - Création d'un nouveau compte

Pour la vue principale « MainActivity », il a fallu faire deux fragments afin de pouvoir faire un onglet discussion 1 à 1 et un onglet discussion de groupe. Un fragment est une vue spéciale pouvant être intégrée dans une autre vue, ça permet d'afficher deux vues dans une et de pouvoir « slider » (glisser) entre les deux.

Adapters

Les adaptateurs, « adapters » en anglais, permettent de personnaliser les listes. Ils héritent de « ArrayAdapter ». C'est une classe qui permet d'afficher un simple champ texte dans les listes et il doit être redéfini si on veut afficher autre chose dans les éléments de la liste.

Nous avons donc dû créer des adaptateurs pour les listes des contacts et des groupes afin d'afficher un texte « Message non lu » en cas de nouveau message. Pour la liste des membres d'un groupe afin de pouvoir afficher un bouton de suppression de membre. Et pour les listes de messages afin d'afficher l'heure d'envoi et le nom de l'utilisateur qui a envoyé le message (dans le cas des discussions de groupe).

Dans le diagramme ci-dessous on peut voir les adaptateurs et les activités qui les utilisent :

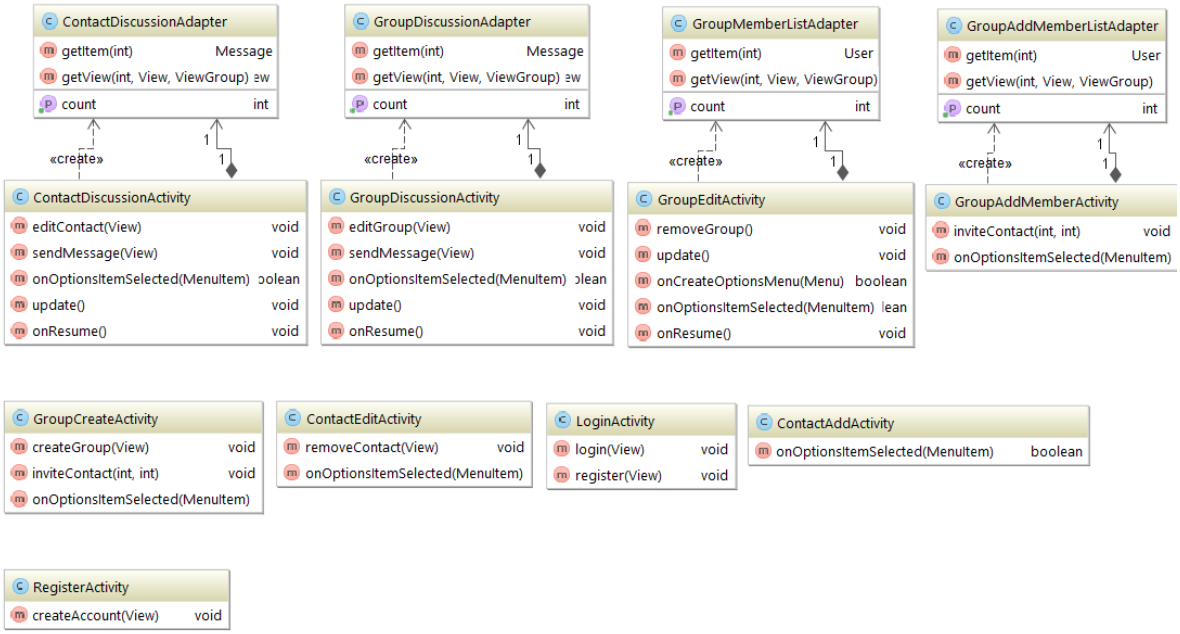

Figure 12 : Diagramme des activités et de leurs adapteurs

L'activité « MainActivity » n'est pas montrée ici car elle est liée aux fragments.

Fragments

Le paquet fragment contient, comme décrit plus haut, les deux fragments permettant d'afficher la liste des contacts et la listes des clients. Les 2 fragments sont intégrés dans l'activité « MainActivity » qui permet de passer de l'un à l'autre soit avec l'onglet soit avec un glissement de doigt.

Dans ce diagramme on voit les fragments qui sont utilisé par l'activité « MainActivity ». Ils ont également chacun un adaptateur pour leur liste :

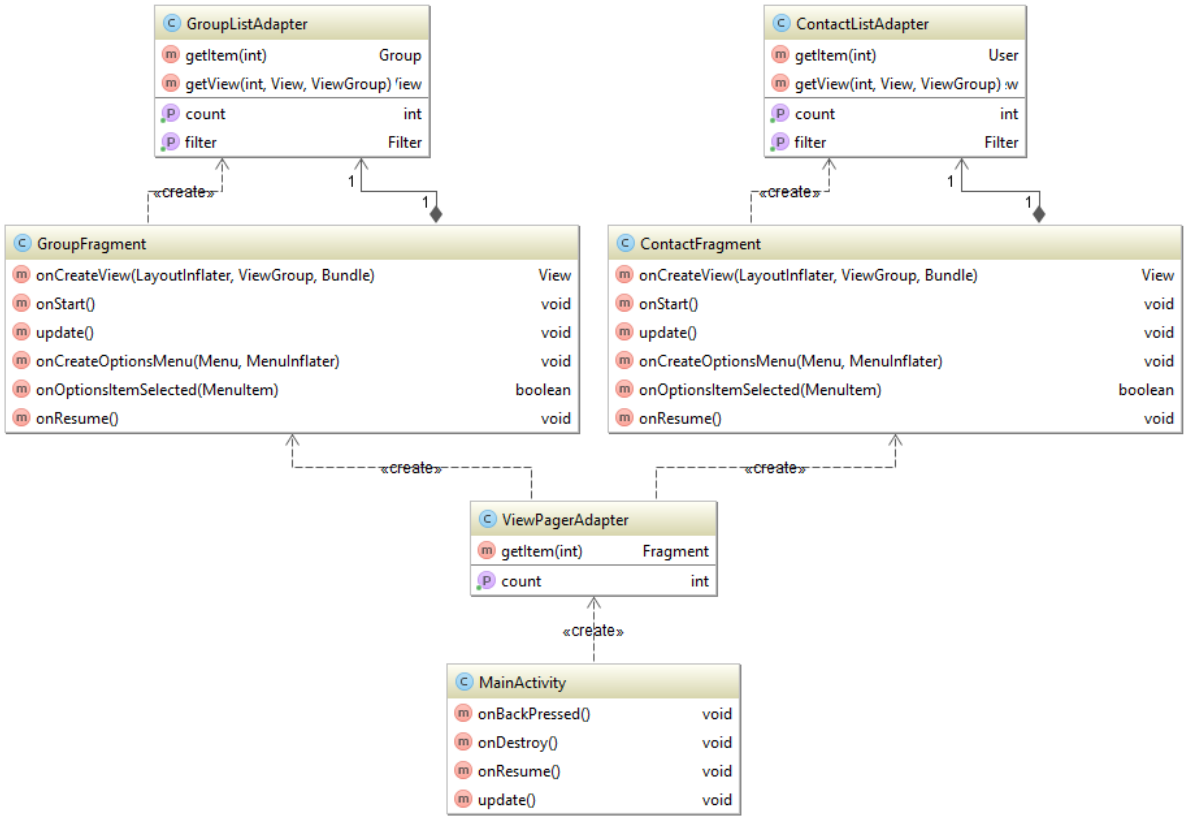

Figure 13 : Diagramme des fragments et de leur adapteur

Communications

Les paquets Communications permet d'envoyer des requêtes HTTP GET, POST, PUT et DELETE. Les classes « RequestDelete », « RequestPUT », « RequestGET » et « RequestPOST » héritent de la classe abstraite Communication qui hérite elle-même de « AsyncTask » (tâche asynchrone exécutée en fond). Cette classe prend on paramètre une classe anonyme héritant de l'interface ICallback et qui redéfinit les méthodes « Success » et « Failure » qui seront appelées respectivement en cas de succès ou d'échec.

Dans le diagramme ci-dessous on voit le paquet communications avec les 4 classes qui héritent de la classe « Communication » :

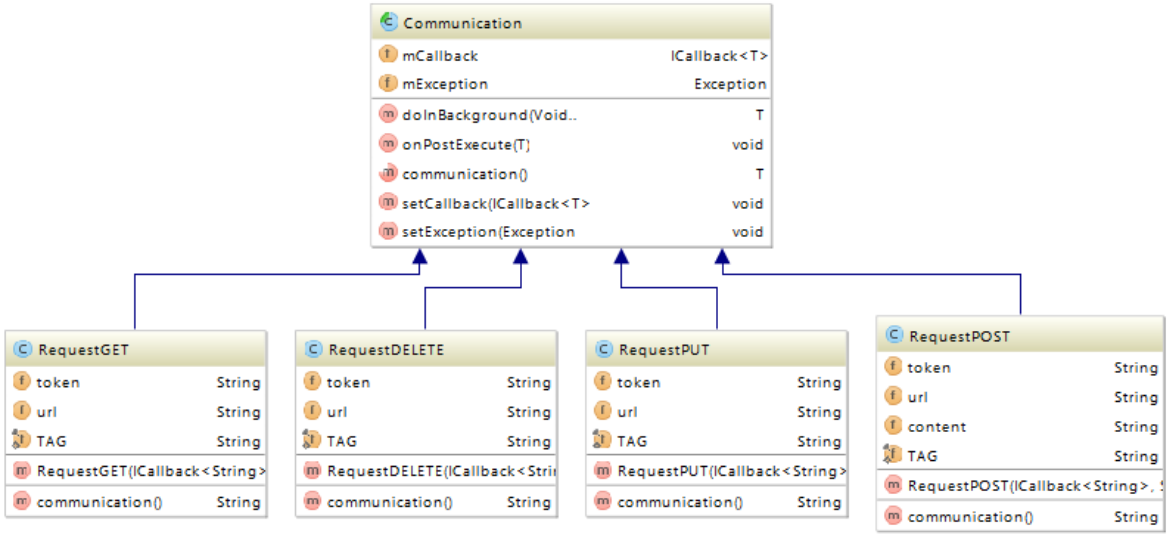

*Figure 14 : Diagramme du paquet communication**S***

Interfaces

Les interfaces utilisées dans l'application sont les suivantes :

- ICallback
  - Utilisé pour récupérer le résultat d'une requête http
- ICustomCallback
  - Utilisé pour gérer la mise à jour de l'activité courante lors d'un nouvel événement (les événements sont décrits dans la partie serveur)
- IJSONKeys
  - Définit toutes les clés JSON utilisée pour les requêtes http.
- IRequests
  - Définis toutes les url de l'api REST.

Dans le diagramme ci-dessous on voit les interfaces et toutes leurs variables et méthodes :

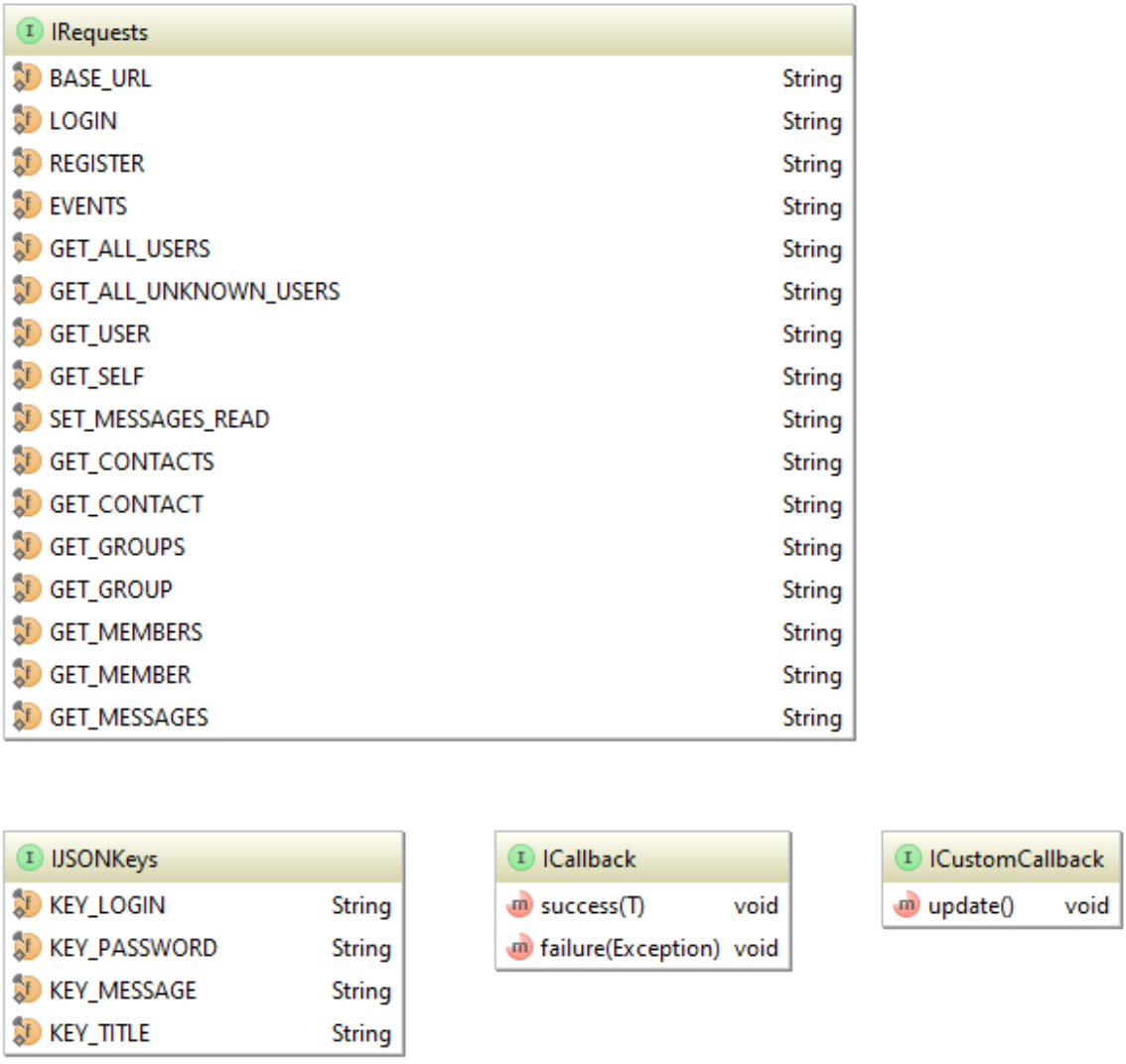

Figure 15 : Diagramme des interfaces

Models

Ce paquet contient les modèles de l'application (afin de compléter le modèle MVC). Il contient les classes groupe, message et utilisateur. Les classes groupes et utilisateurs contiennent une variable statique liste des contacts et liste des groupes afin de pouvoir accéder depuis n'importe où à ces listes.

Dans le diagramme ci-dessous on peut voir les trois classes du modèle :

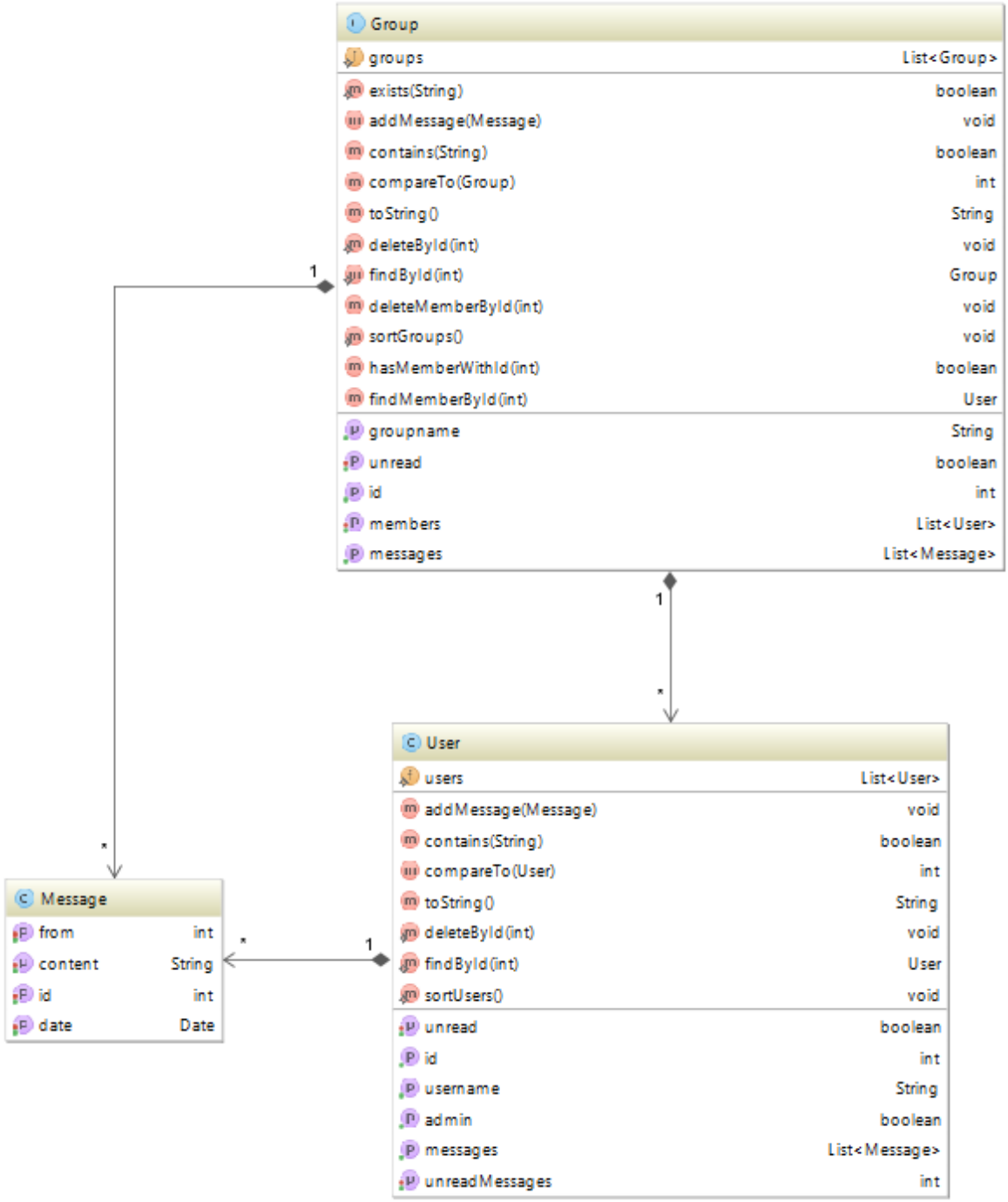

Figure 16 : Diagramme du package Model

Services

Le paquet service ne contient pas un service Android à proprement dit mais une classe qui va démarrer un thread responsable de récupérer tous les événements du serveur, d'en informer l'activité courante et de mettre à jour le modèle.

On peut voir ci-dessous toutes les méthodes qu'il utilise afin de gérer les différents événements. Cette classe est un singleton, elle a un constructeur privé et une méthode publique statique « getInstance » qui permet de récupérer l'instance de la classe (elle est créée si elle n'existait pas). De cette façon, nous aurons qu'un seul service d'événements ce qui réduit le risque d'erreurs et d'autre part l'instance sera accessible depuis n'importe où.

La variable « activity » contient l'activité courante et est utilisée par la fonction « updateCallbackActivity ». En cas d'événement, la méthode « updateCallbackActivity » sera appelée et signalera à la vue courante qu'il y a peut-être eu un changement dans le modèle.

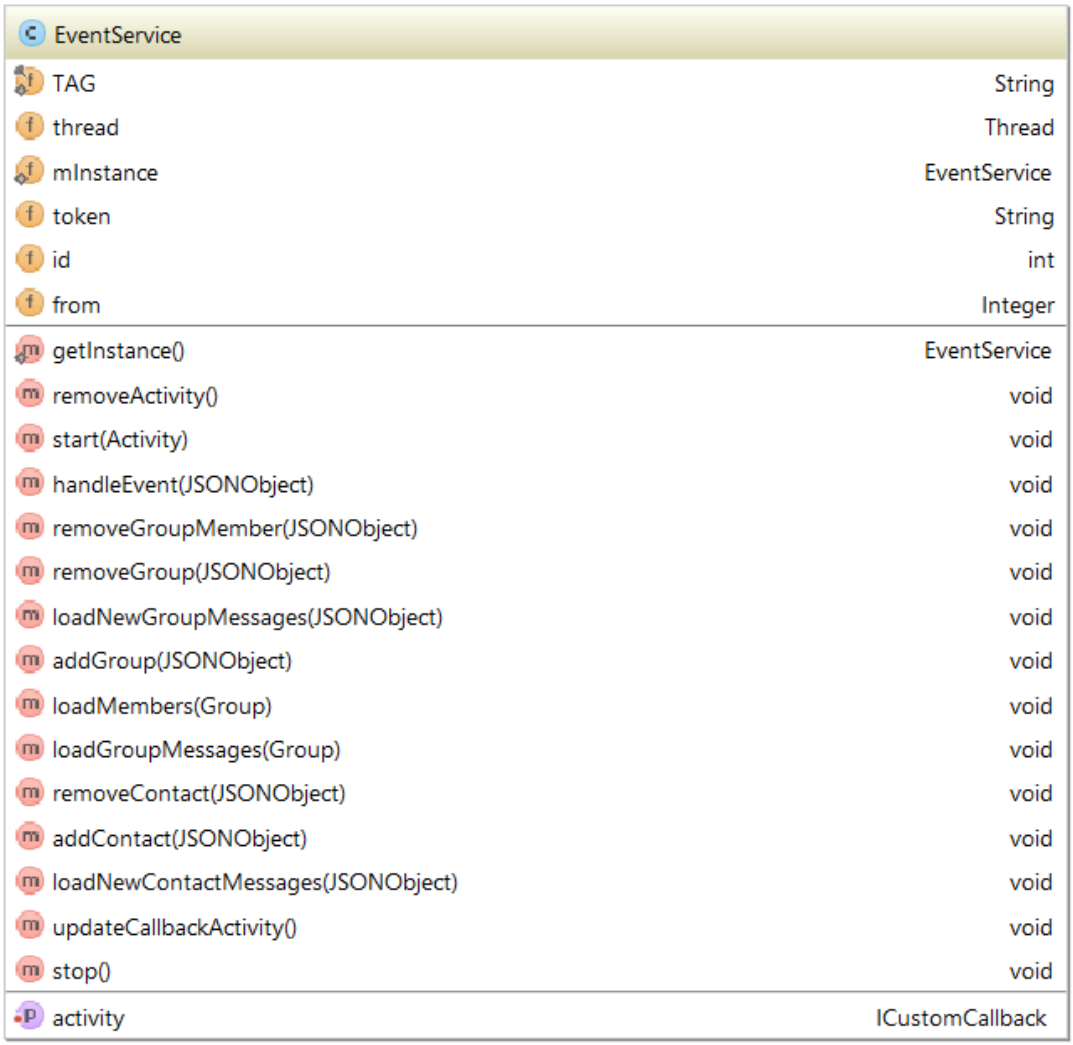

Figure 17 : Diagramme du service d'évènements

Utilities

Le paquet utilities contient une classe Utils qui permet différentes choses :

- Enregistré/Récupérer le token et l'identifiant de l'utilisateur connecté dans les préférences. Il peut-être accéder depuis n'importe quelle activité.
- Créer une chaine de caractère JSON pour l'envoi au serveur.
- Afficher une alerte en cas d'erreur dans le programme.

On peut voir dans le diagramme ci-dessous les différentes fonctions utilitaires proposée par la classe :

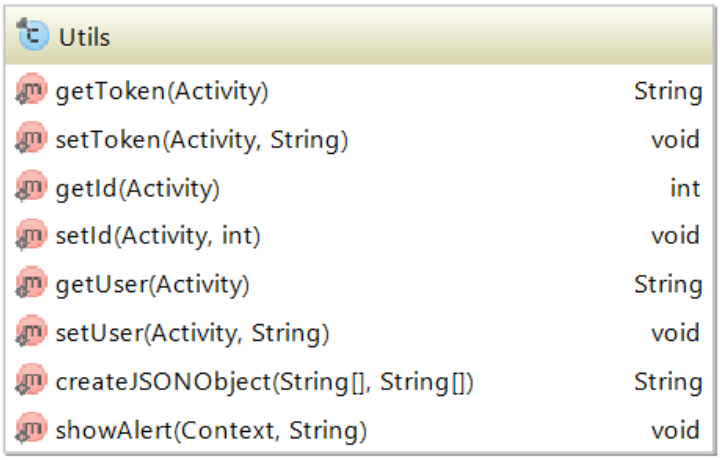

Figure 18 : Diagramme de la classe « Utils »

Gestion du projet

Rôle des participants

Représentant des utilisateurs

Amel Dussier

Responsabilités :

- Collecte des besoins
- Spécification des tests de fonctionnalités
- Explication des aspects métiers

Chef de projet

Amel Dussier

Responsabilités :

- Planification
- Coordination avec les utilisateurs

Analyste

Antoine Drabble

Responsabilités :

- Spécifications
- Collecte des demandes de changement

Software architect

Bastien Clément

Responsabilités :

- Conception de l'architecture du produit

Programmeurs

Bastien Clément, Antoine Drabble & Guillaume Serneels

Responsabilités :

- Participe à la conception du produit
- Ecriture des tests unitaires
- Codage

Responsable des tests

Guillaume Serneels

Responsabilités :

- Participe à l'intégration continue des composants
- Ecrit les tests fonctionnels
- Met en place l'architecture permettant de lancer régulièrement les tests fonctionnels

Responsable de la configuration

Bastien Clément

Responsabilités :

- Gestion de la base des artefacts du projet
- Gestion des releases
- Allocation des droits
- Responsable de la configuration logicielle & matérielle
- Intégration des changements

Plan des itérations initial

Itération 1

Objectif général : Création de la base de données et mise en place d'une première communication simple entre client-serveur

Objectifs détaillés :

- Apprendre les bases du développement Scala et Android (Gestion)
- Mettre en place la base de données (Développement de l'infrastructure)
- Définir une première version du protocole de communication (Gestion : Conception)
- Réaliser une première communication entre le client et le serveur (Développement de l'infrastructure)
    - Cet objectif est un prérequis de tous les cas d'utilisation. A l'issue de l'itération, il sera possible de vérifier qu'un message simple a bien été transmis du client au serveur.

Durée : 2 semaines

Date de début : vendredi 22 avril

Date de fin : vendredi 6 mai (Ascension)

Partage du travail, les heures sont indiquées par semaine :

- Antoine :
    - Apprendre les bases du développement Scala (3h)
    - Définition de la première version du protocole de communication (2h)
- Bastien :
    - Création de la base de données (2h)
    - Mise en place de la communication côté serveur (2h)
- Guillaume :
    - Apprendre les bases du développement Android (3h)
    - Mise en place de la communication côté client (2h)
- Amel :
    - Apprendre les bases du développement Android (3h)
    - Première ébauche de l'interface de connexion (2h)

Temps consacré : environ 19 heures par semaines (38 heures au total)

Itération 2

Objectif général : Mise en place des fonctionnalités de création / suppression de compte et de connexion

Objectifs détaillés :

- Ajouter les fonctionnalités de gestion de compte et de connexion au protocole de communication (Gestion : Conception)
- Interfacer l'application serveur avec la base de données (Développement de l'infrastructure)
- Implémenter la gestion des comptes et de connexion au niveau du serveur et du client (Développement des fonctionnalités)
    - Cas d'utilisation réalisés complètement : Création de compte, Connexion à l'application
    - Cas d'utilisation réalisés partiellement : Supprimer son compte, Se déconnecter
        - Ces fonctionnalités seront implémentées au niveau du code, mais leur intégration à l'interface utilisateur se fera lors des itérations suivantes.
          Il sera néanmoins possible de voir qu'une commande de déconnection est reçue coté serveur.
          Lors d'une suppression de compte, il sera possible de voir qu'une commande de suppression est reçue coté serveur et que les informations du compte sont effectivement supprimées de la base de données.
- Commencer la rédaction du rapport final, avec la structure des chapitres (Gestion : Rédaction)

Durée : 1 semaine

Date de début : mardi 3 mai

Date de fin : lundi 9 mai

Partage du travail :

- Antoine :
    - Ajouter les fonctionnalités de gestion de compte et de connexion au protocole de communication (3h)
    - Interfacer l'application serveur avec la base de données (2h)
- Bastien :
    - Implémenter la gestion des comptes et de connexion au niveau du serveur (5h)
- Guillaume :
    - Implémenter la gestion des comptes et de connexion au niveau du client (5h)
- Amel :
    - Finaliser l'interface utilisateur pour la création de compte et la connexion (3h)
    - Commencer la rédaction du rapport final (2h)

Temps consacré : environ 20 heures (5 heures par personne)

Itération 3

Objectif général : Mise en place des fonctionnalités de recherche et de gestion de contacts

Objectifs détaillés :

- Ajouter les fonctionnalités de recherche et de gestion de contacts au protocole de communication (Gestion : Conception)
- Implémenter la recherche et de gestion de contact au niveau du serveur et du client, ainsi que les interfaces utilisateur correspondantes (Développement des fonctionnalités)
    - Cas d'utilisation réalisés complètement : Gérer les contacts
    - Cas d'utilisation réalisés partiellement : Créer une discussion privée, Ajouter un utilisateur dans un groupe public
        - La fonctionnalité de recherche est utilisée dans plusieurs cas d'utilisation. La réalisation complète de ces cas d'utilisation se fera lors des itérations suivantes.
- Continuer la rédaction du rapport final (Gestion : Rédaction)

Durée : 1 semaine

Date de début : mardi 10 mai

Date de fin : lundi 16 mai

Partage du travail :

- Antoine :
    - Ajouter les fonctionnalités de recherche et de gestion de contact au protocole de communication
    - Continuer la rédaction du rapport
- Bastien :
    - Implémenter la recherche et de gestion de contact au niveau du serveur
- Guillaume :
    - Implémenter la recherche et de gestion de contact au niveau du client
    - Ajout de l'interface de recherche de contact
- Amel :
    - Ajout de l'interface de gestion de contact

Temps consacré : environ 20 heures (5 heures par personne)

Itération 4

Objectif général : Mise en place des discussions privées

Objectifs détaillés :

- Ajouter les fonctionnalités de création de discussion privée, d'envoi de messages et d'historique au protocole de communication (Gestion : Conception)
- Implémenter la création / suppression de discussion, l'envoi de messages et l'affichage de l'historique au niveau du serveur et du client, ainsi que les interfaces utilisateur correspondantes (Développement des fonctionnalités)
    - Cas d'utilisation réalisés complètement : Créer une discussion privée, Envoyer un message, Consulter les messages, Charger les messages de l'historique
    - Cas d'utilisation réalisés partiellement : Créer un groupe de discussion public
        - Une discussion publique est une extension d'une discussion privée, donc certaines fonctionnalités seront identiques. La réalisation complète de ce cas d'utilisation se fera lors des itérations suivantes.
- Continuer la rédaction du rapport final (Gestion : Rédaction)

Durée : 1 semaine

Date de début : mardi 17 mai

Date de fin : lundi 23 mai

Partage du travail :

- Antoine :
    - Ajouter les fonctionnalités de création de discussion privée, d'envoi de messages et d'historique au protocole de communication
    - Implémenter la gestion de l'historique de discussion au niveau du serveur
- Bastien :
    - Implémenter la création / suppression de discussion, l'envoi de messages au niveau du serveur
- Guillaume :
    - Implémenter la création / suppression de discussion, l'envoi de messages et l'affichage de l'historique au niveau du client
    - Continuer la rédaction du rapport
- Amel :
    - Ajout de l'interface de création et d'affichage de discussion
    - Ajout de l'interface de saisie de message

Temps consacré : environ 20 heures (5 heures par personne)

Itération 5

Objectif général : Création de discussion publique (discussion de groupe)

Objectifs détaillés :

- Ajouter les fonctionnalités de discussion publique (créer, supprimer, rejoindre), et de gestion des membres (ajouter, supprimer, promouvoir administrateur de groupe) d'une discussion publique au protocole de communication (Gestion : Conception)
- Implémenter toutes les fonctionnalités relatives aux discussions publiques au niveau du serveur et du client, ainsi que les interfaces utilisateur correspondantes (Développement des fonctionnalités)
    - Cas d'utilisation réalisés complètement : Rejoindre une discussion publique, Créer un groupe de discussion public, Supprimer un utilisateur d'un groupe de discussion public, Quitter un groupe, Ajouter un utilisateur dans un groupe
- Continuer la rédaction du rapport final (Gestion : Rédaction)

Durée : 2 semaines

Date de début : mardi 24 mai

Date de fin : lundi 6 juin

Partage du travail :

- Antoine :
    - Ajouter toutes les fonctionnalités relatives aux discussions publiques au protocole de communication
    - Aider Bastien et Guillaume pour implémenter les fonctionnalités au niveau serveur ou client
- Bastien :
    - Implémenter toutes les fonctionnalités relatives aux discussions publiques au niveau du serveur
- Guillaume :
    - Implémenter toutes les fonctionnalités relatives aux discussions publiques au niveau du client
    - Extension de l'interface d'affichage de discussion privée pour gérer les discussions publiques
- Amel :
    - Ajout de l'interface de gestion de groupes
    - Continuer la rédaction du rapport

Temps consacré : environ 40 heures (5 heures par personne et par semaine)

Itération 6

Objectif général : Signalement et blocage

Objectifs détaillés :

- Ajouter les fonctionnalités de signalement de message et de blocage d'utilisateur au protocole de communication (Gestion : Conception)
- Implémenter toutes les fonctionnalités relatives au signalement de message et de blocage d'utilisateur au niveau du serveur et du client, ainsi que les interfaces utilisateur correspondantes (Développement des fonctionnalités)
    - Cas d'utilisation réalisés complètement : Consulter les signalements d'un groupe public, Consulter les reports d'un groupe privé, Reporter un autre utilisateur
- Terminer la première version du rapport final (Gestion : Rédaction)

Durée : 1 semaine

Date de début : mardi 7 juin

Date de fin : lundi 13 juin

Partage du travail :

- Antoine :
    - Ajouter les fonctionnalités de signalement de message et de blocage d'utilisateur au protocole de communication
    - Terminer la première version du rapport final
- Bastien :
    - Implémenter toutes les fonctionnalités relatives au signalement de message et de blocage d'utilisateur au niveau du serveur
    - Finaliser l'application serveur
- Guillaume :
    - Implémenter toutes les fonctionnalités relatives au signalement de message et de blocage d'utilisateur au niveau du client
    - Finaliser l'application cliente et les différentes interfaces
- Amel :
    - Ajouter les options de signalement et de blocage aux différentes interfaces utilisateur
    - Finaliser les différentes interfaces

Temps consacré : environ 20 heures (5 heures par personne)

Bilan des itérations

Itération 1

Durée
2 semaines

Date de début
vendredi 22 avril

Date de fin
lundi 9 mai

Objectif
Création de la base de données et mise en place d'une première communication simple entre client-serveur.

Avancement
La base de données est créée et le processus de déploiement du logiciel serveur est prêt.

Pas encore de communication client-serveur suite à un changement de plan après discussion avec Jonathan. Nous utiliserons une API REST pour la majorité des opérations.

Nous pensons conserver une interface Socket lorsque l'application est ouverte uniquement pour permettre des notifications push au client Android (au lieu d'utiliser les services GCM). Le protocole du socket est donc grandement simplifié et sera défini lors d'une future itération.

Bilans personnels (heures effectives / heures planifiées)
Antoine (5h/5h)

- Apprendre les bases du développement Scala
    - J'ai mis en place mon environnement de développement. J'ai commencé à apprendre le langage Scala. Je vais devoir également étudier le framework Play que l'on va utiliser pour faire le serveur.
- Définition de la première version du protocole de communication
    - Nous avions d'abord prévu d'utiliser une connexion TCP ainsi qu'un protocole de communication binaire, mais après discussion avec l'assistant, nous allons mettre en place une communication REST en JSON et une communication TCP pour les notifications de type PUSH. Nous devons donc encore en parler avant de tout mettre en place.
    - J'ai également commencé à mettre en place une communication simple du côté client.

Bastien

- Aucun souci particulier à mentionner. La mise en place d'un hook GitHub pour automatiser le déploiement du serveur permettra d'avoir facilement une version « stable » du serveur accessible à tout moment pour le développement de l'application Android.

- La communication côté serveur a été développée avec en tête un protocole entièrement basé sur un socket bidirectionnel. Après discussion, ces fonctionnalités ne seront pas utiles puisque nous pouvons développer une grande partie de l'application en utilisant une API REST, très simple à mettre en œuvre avec Play.

Guillaume (5h/6h)

- Apprendre les bases du développement Android (3h) Début de l'apprentissage d'Android. Création du projet client avec une première Activity. Plusieurs interrogations concernant la gestion des IO sur Android et la mise en place de l'API Rest à clarifier avec l'assistant.
- Mise en place de la communication côté client Pas encore de communication effective coté client (cf. ci-dessus) Mise à jour du rapport (1h)

Amel

- Apprendre les bases du développement Android
    - Installation et configuration d'Android Studio, et d'un « device » pour tester et débugger les applications.
    - Prise en main de l'environnement de développement, de la structure des projets Android (ressources, contrôleurs en Java), etc. Un peu de peine à comprendre certaines notions, comme les fichiers de configuration « gradle » par exemple.
- Première ébauche de l'interface de connexion
    - Première version de l'activité Login, avec le placement de boutons, de champs textes et de boutons radio. Pas mal de recherche pour trouver les attributs nécessaires pour placer correctement les éléments (alignements, espaces entre les éléments, etc.).

Itération 2

Durée
1 semaine

Date de début
mardi 10 mai

Date de fin
lundi 16 mai

Objectif
Mise en place des fonctionnalités de création / suppression de compte et de connexion.

Avancement
-

Bilans personnels (heures effectives / heures planifiées)
Antoine

- Ajouter les fonctionnalités de gestion de compte et de connexion au protocole de communication
    - J'ai refactorisé le client Android et j'ai implémenté les boutons de logins et d'inscription afin qu'ils communiquent avec le serveur.
- Interfacer l'application serveur avec la base de données
    - J'ai créé la base de données et Bastien s'est occupé de l'interfacer avec la base de données.

Bastien

- Une gestion de compte relativement simple est disponible côté serveur. Les opérations de connexion et d'inscription sont disponibles.

Guillaume (5h/5h)

- Implémenter la gestion des comptes et de connexion au niveau du client
    - Envoi d'un register/login.
    - Pas encore de token retourné par le serveur
    - Planification: Si possible discuter vendredi 20 mai avec Jonathan Bischof et Bastien Clément au sujet des IO pour déterminer le protocole à utiliser.

Amel

- Finaliser l'interface utilisateur pour la création de compte et la connexion
    - L'activité de Login a été complétée avec quelques fonctionnalités :
        - l'affichage ou non du champ texte pour le nom du serveur privé est maintenant automatique, selon la sélection des boutons radio

        - un clic sur le bouton d'inscription lance maintenant l'activité de *Subscription*
    - Création de l'activité de Subscription :
        - placement des différents éléments de l'interface
        - mise en place des évènements lors des saisies de texte, par exemple pour vérifier que les champs ne sont pas vides ou que les deux mots de passe sont identiques
        - début de réflexion concernant la validation du nom de l'utilisateur: il faut qu'on se mette d'accord sur le format (commence par une lettre, pas d'espaces, lettres autorisées?) et sur la vérification de doublons (contact avec le serveur pour interdire l'utilisation d'un username déjà existant par exemple)
- Commencer la rédaction du rapport final
    - Pas encore eu le temps de commencer

Itération 3

Durée
1 semaine

Date de début
mardi 17 mai

Date de fin
lundi 23 mai

Objectif
Mise en place des fonctionnalités de recherche et de gestion de contacts

Avancement
Le projet avance bien, le refactoring de l'assistant nous a permis d'avoir un code plus propre côté client mais nous a fait perdre un peu de temps pour la fusion et l'adaptation à la nouvelle architecture. Nous avons un petit retard sur la gestion des contacts mais ça devrait être rapidement rattrapé maintenant que tout est en place.

Du côté serveur tout se passe bien.

Bilans personnels (heures effectives / heures planifiées)
Antoine (9h/5h)

- Ajouter les fonctionnalités de recherche et de gestion de contact au protocole de communication.
    - Comme l'assistant a refactorisé le client et Amel a fait des changements en même temps, j'ai dû fusionner les deux ce qui m'a pris pas mal de temps. J'ai également dû faire refonctionner le login et le register (itération 2).
    - J'ai créé les classes RequestPUT et RequestDelete pour l'envoi de requête HTTP PUT et DELETE.
    - J'ai créé l'activité ContactViewActivity qui permet de voir les messages envoyés avec un contact.
    - J'ai implémenté le bouton suppression d'un contact, mais je n'ai pas encore pu le tester
    - J'ai fait fonctionner la récupération du token pendant l'authentification/enregistrement.
    - J'ai fait fonctionner les fonctions GetToken et SetToken du client.
    - J'ai commencé à récupérer la liste des contacts afin de les afficher.
    - J'ai créé l'activité de recherche d'utilisateurs, mais je n'ai pas encore remplis la liste des utilisateurs.
- Implémenter la recherche et la gestion de client au niveau du client.

- Je n'ai pas eu le temps d'implémenter la recherche et la gestion car j'ai d'abord dû faire fonctionner le login/register et fusionner les deux projets.

Bastien (5h/4h)

- Implémenter la recherche et de gestion de contacts au niveau du serveur
  - La fonctionnalité a été implémentée sans difficulté. L'API REST est entièrement fonctionnelle pour les opérations de gestion de contacts.
  - Le temps supplémentaire relatif à la planification est lié à la mise de mécanisme de traitement d'erreur au niveau du serveur qui n'est pas directement liés à la gestion de contact. Il est maintenant plus aisé de communiquer un échec au client de l'API et le serveur devrait maintenant retourner les exceptions non-attrapées au consommateur de l'API en format JSON.
  - Par la suite, il sera possible de se baser sur le statut administrateur du client pour déterminer si l'exception doit ou non être détaillée.

Guillaume (4h/5h)

- Implémenter la gestion des comptes et de connexion au niveau du client
  - Compréhension et intégration du client v2 refactorisé par l'assistant.
  - Login / obtention du token fonctionnel
- Implémenter la recherche et la gestion de contact au niveau du client
  - Recherche fonctionnelle sur liste de contacts codés en dur
  - Suite à l'intégration du client v2, pas encore pu
- Ajout de l'interface de recherche de contact
  - Interface de recherche fonctionnelle

Amel (5h/5h)

- Ajout de l'interface de recherche de contact
  - Création d'une interface pour lister les contacts de l'utilisateur
  - Possibilité de faire une recherche avec un widget SearchView (passé un peu de temps à comprendre comment configurer la recherche avec un Adapter, et comment personnaliser l'affichage)
  - La sélection d'un contact permet d'accéder à l'interface de gestion du contact
  - Pas encore réussi à charger la liste de contacts directement depuis le serveur
- Ajout de l'interface de gestion de contact
  - Création d'une première version de l'interface de gestion d'un contact
  - Récupère le contact passé en paramètre depuis l'activité précédente
  - Pour l'instant l'interface contient seulement un bouton pour supprimer le contact (fonctionnel)

Itération 4

Durée
1 semaine

Date de début
mardi 24 mai

Date de fin
lundi 30 mai

Objectif
Mise en place des discussions privées

Avancement
Création d'une discussion privée coté client.

Bilans personnels (heures effectives / heures planifiées)
Antoine (9h/5h)

J'ai pu rattraper le retard de l'itération précédente. J'ai fait fonctionner la recherche et l'ajout de contacts, l'affichage de la liste des contacts, la suppression d'un contact et les boutons de retour en arrière. (4h)

J'ai également ajouté l'affichage des erreurs dans le client. (30min)

J'ai commencé à mettre en place la vue de discussion 1 à 1 avec Guillaume, et nous avons préparé l'adapter qui sera utilisé pour l'affichage de la conversation. (3h)

J'ai implémenté la réception des messages depuis le serveur et leur affichage dans la fenêtre de discussion que j'ai renommé en ContactDiscussionActivity. (1h)

J'ai implémenté le bouton d'envoi pour que les messages soient bien envoyés sur le serveur. (30min)

- Ajouter les fonctionnalités de création de discussion privée, d'envoi de messages et d'historique au protocole de communication.
    - Les messages fonctionnent mais il reste les notifications à faire
- Implémenter la gestion de l'historique de discussion au niveau du serveur.
    - Bastien s'en est occupé.

Bastien (7h/5h)

- Quelques mises à jour des API de l'itération 3
- Changement de la gestion des contacts côté serveur, au lieu d'avoir une paire d'entrée dans une table, il n'y a plus qu'une entrée avec un ordre précis des contacts
- Mise en place du système de push d'événement en utilisant la technique du long-polling HTTP
- Mise en place de l'API pour la gestion des discussions privées et des événements associés
- Mise en place de la gestion des messages lus / non-lus et des événements associés

- Amélioration du processus de mise à jour du serveur en utilisant Docker

Guillaume (5h/5h)

- Implémenter la création / suppression de discussion, l'envoi de messages et l'affichage de l'historique au niveau du client
    - Modification de ContactViewActivity, qui permet de modéliser une discussion contenant la liste de messages passé
    - La suppression de discussion correspond à la suppression du contact
    - En attente de l'implémentation des messages cotés serveur
- Continuer la rédaction du rapport
    - Ajout d'un fichier planification_iterations.md sur le git, avec la planification mise en forme. Mise à jour de bilan_iteration.md avec les dates.

Amel (5h/5h)

- Ajout de l'interface de création et d'affichage de discussion
    - Modifications de l'activité ContactViewActivity
    - Ajout d'un bouton "Envoyer message" dans l'interface de gestion de contact
- Ajout de l'interface de saisie de message
    - Intégrée à l'interface de saisie de message
- Correction de bugs dans l'interface de recherche de contacts

Itération 5

Durée
1 semaine

Date de début
mardi 31 mai

Date de fin
lundi 6 juin

Objectif
-

Avancement
-

Bilans personnels (heures effectives / heures planifiées)
Antoine (33h/5h)

J'ai dû refactoriser le code car on n'enregistrait pas la liste des utilisateurs et des messages et pour des questions de performance, il fallait la stocker dans l'application. J'ai également généré toute la javadoc des classes du client. 3h

J'ai mis la fonctionnalité du bouton de retour dans le fichier manifest comme l'assistant m'avait dit de faire mais ça change le comportement et du coup j'ai du rollback. (1h)

J'ai commencé à chercher comment faire le système de notification depuis Android. (Faire une classe qui permette de mettre à jour les adapteurs des différentes vues en allant chercher les évènements sur le serveur). J'ai trouvé une façon pas très propre et je cherche mieux. (3h)

J'ai changé la façon de faire la gestion des évènements en mettant en place un service comme vu avec Jonathan. (2h)

J'ai dû résoudre le problème que RequestGET est un task et pas un thread et bloque l'exécution des autres requêtes quand on veut récupérer les évènements. (2h)

J'ai fait marcher les évènements et la récupération du JSON. J'ai fait que la recherche n'affiche pas les utilisateurs déjà en contact et soi-même. (1h)

J'ai fait marcher les évènements d'ajout et suppression de contact. Et les évènements de nouveaux messages. (2h)

J'ai fait marcher le système de notification pour afficher une notification dans la liste des contacts quand un nouveau message a été reçu. (1h30)

On a eu le cours où on a fait la démo et avancer sur les évènements et la vue de création de groupe. (4h)

J'ai refactorisé un peu le code après les changements fait par Jonathan et j'ai résolu un problème avec les évènements. (1h)

J'ai ajouté les vues discussion de groupe et édition de groupe. J'ai implémenté l'envoi de message, la réception de message, la création de groupe, l'affichage de la liste des groupes... (4h)

J'ai ajouté le tri des contacts/groupes par dernier message reçu et j'ai ajouté le bouton de déconnection. (1h)

J'ai résolu quelques problèmes notamment la mise à jour des listes des fragments. (1h)

J'ai résolu des problèmes d'évènements (création de groupe et messages non lus). (1h)

Afficher la liste des membres d'un groupe et les boutons de suppression de membre. (1h30)

Ajouter la suppression et l'ajout de membres dans un groupe. Permettre de supprimer/quitter le groupe. Les fonctionnalités dépendent de si on est admin ou pas. (4h)

- Ajouter toutes les fonctionnalités relatives aux discussions publiques au protocole de communication.
    - J'ai ajouté une bonne partie des fonctionnalités de discussion publique.
- Aider Bastien et Guillaume pour implémenter les fonctionnalités au niveau serveur ou client.
    - Bastien c'est occupé de la partie serveur (je lui ai communiqué les quelques bugs).

Bastien (TODO/TODO)

- TODO

Guillaume (7h/5h)

- Ajout dans l'activité CreateGroup de l'affichage de la liste des contacts avec des cases à cocher. (2h)
- Refactorisation de l'activité principale sous la forme de deux fragments (contact/groupes) afin de pouvoir les afficher sous la forme d'ongles facilement navigables, avec l'aide de l'assistant. (3h)
- pour implémentation de l'onglet GroupFragment: création de l'adaptateur pour l'affichage des groupes et du modèle groupe. (2h)

Amel (5h/5h)

- Rédaction du rapport final
    - Définition de la structure de base
    - Intégration et mise en page du contenu du rapport intermédiaire
    - Complété la partie "Analyse"
    - Création du diagramme d'activité général
    - Premier état des lieux pour les illustrations manquantes et les sujets à éclaircir avec l'ensemble du groupe

Itération 6

Durée
1 semaine

Date de début
mardi 7 juin

Date de fin
lundi 13 juin

Objectif

Avancement

Bilans personnels (heures effectives / heures planifiées)

Antoine (3h/5h)

- Ajouter les fonctionnalités de signalement de message et de blocage d'utilisateur au protocole de communication.
    - Ces fonctionnalités ne seront pas intégrées car d'une part nous avons beaucoup trop de travail pour les 2 dernières semaines de cours et d'autre part car il est important de faire le rapport bien.
- Terminer la première version du rapport final
    - Nous avons commencé le rapport final

Bastien (TODO/5h)

- TODO

Guillaume (TODO/5h)

- TODO

Amel (TODO/5h)

- Rapport final

Stratégie de tests

Après une recherche d'informations sur le sujet, nous avons déterminé que le meilleur moyen de tester notre application serait de mettre en place des tests d'instrumentation. Ces tests permettent de vérifier certains comportements de l'interface graphique et de l'application en général.

Afin de réaliser cela, nous avons suivis les recommandations spécifiées dans la documentation Android (voir sources ci-dessous). Voici comment nous avons procédé :

- Téléchargement et installation de l'**Android Testing Support Library** qui fournit une API permettant d'implémenter facilement des tests unitaires JUnit ainsi que les tests d'instrumentation que nous avons implémentés pour notre application.

- Spécification du **testInstrumentationRunner** pour la configuration par défaut dans le fichier **build.gradle** :

```
android {
    compileSdkVersion 23
    buildToolsVersion "23.0.3"

    defaultConfig {
        applicationId "ch.heigvd.gen"
        minSdkVersion 23
        targetSdkVersion 23
        versionCode 1
        versionName "1.0"
        testInstrumentationRunner "android.support.test.runner.AndroidJUnitRunner"
    }
```

- Installation du framework **Espresso**, qui est l'API de test d'instrumentation fournie par l'Android Testing Support Library. Ce framework permet d'effectuer des actions sur les éléments de l'interface graphique (clics, écriture dans un champ, etc.) comme le ferait un utilisateur normal et de vérifier le résultat de ces actions.

- Ajout des dépendances d'Espresso dans le fichier **build.gradle** :

```
dependencies {
    compile fileTree(dir: 'libs', include: ['*.jar'])
    testCompile 'junit:junit:4.12'
    compile 'com.android.support:appcompat-v7:23.3.0'
    compile 'com.android.support:design:23.3.0'

    //Dependencies for testing
    //Solve conflict dependencies
    androidTestCompile 'com.android.support:support-annotations:23.3.0'

    androidTestCompile 'com.android.support.test:runner:0.5'
    // Set this dependency to use JUnit 4 rules
    androidTestCompile 'com.android.support.test:rules:0.5'

    //Intents
    androidTestCompile 'com.android.support.test.espresso:espresso-intents:2.2.2'
    // Set this dependency to build and run Espresso tests
    androidTestCompile 'com.android.support.test.espresso:espresso-core:2.2.2'
    // Set this dependency to build and run UI Automator tests
    androidTestCompile 'com.android.support.test.uiautomator:uiautomator-v18:2.1.2'


}
```

Le code source de nos tests est à placer dans un nouveau dossier **src/androidTest/java.**

Fonctionnement d'Espresso

Espresso permet d'accéder et d'interagir avec les éléments de l'interface graphique. Voici un exemple qui permet de remplir les champs username et password de notre application et de cliquer sur le bouton Login :

```
@Test
public void LoginActivity_To_MainActivity() {
    // Type login text and then press the button.
    onView(withId(R.id.login))
            .perform(typeText(mStringLoginToBetyped), closeSoftKeyboard());

    onView(withId(R.id.password))
            .perform(typeText(mStringPasswordToBetyped), closeSoftKeyboard());

    onView(withId(R.id.login_button))
            .perform(click());
```

Problèmes rencontrés

Nous n'avons malheureusement pas réussi à faire fonctionner correctement les tests avec Espresso, en effet seul un test vraiment trivial fonctionne :

```
@RunWith(AndroidJUnit4.class)
@LargeTest
public class LoginSuccessTest {

    private String mStringLoginToBetyped;
    private String mStringPasswordToBetyped;



    @Rule
    public ActivityTestRule<LoginActivity> mActivityRule = new ActivityTestRule<>(
            LoginActivity.class);
/*
    @Rule
    public IntentsTestRule<LoginActivity> mActivityRule = new IntentsTestRule(LoginActivity.class);
*/
    @Before
    public void initValidString() {
        // Specify valid strings for login.
        mStringLoginToBetyped = "holla";
        mStringPasswordToBetyped = "password";
    }

    @Test
    public void TrivialTest() {
        // Type login text and then press the button.
        onView(withId(R.id.login))
                .perform(typeText(mStringLoginToBetyped), closeSoftKeyboard());

        onView(withId(R.id.password))
                .perform(typeText(mStringPasswordToBetyped), closeSoftKeyboard());

        //onView(withId(R.id.login_button)).perform(click());

        onView(withId(R.id.login_button)).check(matches(withText("Login")));
```

Le problème est que nous n'arrivons plus à effectuer des vérifications sur les éléments de l'interface graphique après avoir changé d'activité, par exemple après le login.

Tests manuels effectués

N'ayant pas réussi à faire fonctionner les tests automatisés, nous avons effectués des tests manuels pour les fonctionnalités suivantes :

- Authentification, démarrer l'application puis saisir le nom d'utilisateur et le mot de passe dans les champs correspondants, puis valider en cliquant sur le bouton Login.

- Se déconnecter : clic sur le bouton en haut à droite, puis choisir Log off

- Créer une nouvelle discussion avec un Contact : depuis l'onglet Contact, cliquer sur le bouton + en haut à droite puis cliquer sur le nom du contact voulus dans la liste

- Accéder une discussion avec un contact : depuis l'onglet Contact, faire un clic sur le nom du contact

    - Envoyer un message dans la discussion, écrire le message puis cliquer sur la flèche d'envoi
    - Editer un contact, cliquer sur le nom du contact
        - Supprimer le contact en cours d'édition, clic sur le bouton REMOVE

    - Revenir à l'écran principal, clic sur le bouton back (la flèche en haut à gauche)

- Créer une nouvelle discussion de groupe, depuis l'onglet Groupe, cliquer sur le bouton + en haut à droite puis saisir le nom du groupe, cliquer sur les cases à cocher à côté du nom des contacts à ajouter dans la discussion. Puis valider la création du groupe en cliquant sur +.

- Accéder une discussion de groupe, depuis l'onglet Groupe, cliquer sur le nom du groupe

    - Envoyer un message dans la discussion, écrire le message puis cliquer sur la flèche d'envoi
    - Editer le groupe, cliquer sur le nom du groupe
        - Supprimer un membre du groupe, cliquer sur le bouton croix à côté du nom du membre
        - Ajouter un membre au groupe, cliquer sur le bouton + en haut à droite puis cliquer sur le nom du contact voulus dans la liste
        - Supprimer le groupe, cliquer sur le bouton croix en haut à droite
    - Revenir à l'écran principal, clic sur le bouton back (la flèche en haut à gauche)

Tous les tests fonctionnels manuels ont réussi, ils sont reproductibles en utilisant, par exemple, en les informations de login suivantes :

- username « Holla »
- password « password »

Sources pour le framework Espresso :

https://developer.android.com/training/testing/start/index.html#config-instrumented-tests

https://developer.android.com/training/testing/ui-testing/espresso-testing.html

Stratégie d'intégration du code

Comme nous étions plusieurs développeurs sur ce projet, nous avons utilisé un système de gestion de sources. Habitués à ces systèmes dans les autres cours et projets, nous avons utilisé Git et GitHub pour gérer notre code source et nos éléments de documentation.

Git permet de gérer les versions des fichiers de code source qui sont partagés au sein d'une équipe de développement. Chaque développeur peut récupérer, modifier et uploader les fichiers. Git dispose d'un système de gestion et de résolution de conflits pour les fichiers qui ont subi des modifications concurrentes.

Le site GitHub permet d'héberger le code source et les documentations en ligne. Il propose également de visualiser les fichiers ainsi que leur historique de modification directement sans passer par l'outil Git en ligne de commande.

Notre stratégie de développement a été de créer une nouvelle branche Git pour le développement de chaque nouvelle fonctionnalité de notre application. Cela permet de séparer au mieux le travail de chacun, et d'avoir un historique clair. Il est également plus facile de retrouver une ancienne version stable si les fonctionnalités ne sont pas ajoutées de façon partielle.

Nous avons également essayé de faire commits et merges fréquents, afin d'éviter au maximum les conflits et les pertes de temps que ça engendre.

L'ensemble des fichiers sources et des documentations de notre projet sont disponibles sur le repository GitHub suivant :

https://github.com/galedric/HEIG-GEN

Etat des lieux

Ce qui fonctionne

Toutes les fonctionnalités, sauf celles mentionnées dans le chapitre suivant, ont été implémentées et testées.

Nous avons également ajouté un système de notification de messages dans les discussions afin de ne pas avoir à ouvrir chaque discussion pour savoir s'il y avait un nouveau message. Ce n'était pas précisé dans le cahier des charges mais ça nous a paru essentiel pour une application de chat.

Nous avons aussi, grâce aux événements (requête HTTP GET qui attend les événements), pu faire en sorte que le client puisse être ouvert sur plusieurs appareils simultanément et que toutes les mises à jour apparaitront sur les deux appareils.

Ce qu'il resterait à développer

Ci-dessous la liste des fonctionnalités que nous n'avons pas pu implémenter dans les délais :

- Discussion publique ouverte à tout le monde
    - Cette fonctionnalité est similaire aux discussions de groupe. Il faudrait compter environ dix heures de travail, pour l'implémentation et pour les tests.
- Délégation de l'administration d'un groupe
    - Cette fonctionnalité est assez simple, car les modifications sont essentiellement au niveau du serveur (mis à part un bouton ou deux dans l'interface). Environ cinq heures de travail devraient suffire pour réaliser cette fonctionnalité.
- Bloquer les contacts
    - Cette fonctionnalité, comme la précédente, est également assez courte à réaliser. Nous estimons à environ cinq heures de temps la charge de travail nécessaire.

Autocritique

Planification

La charge définie dans le plan d'itération a été sous-évaluée.

Nous n'avons pas réalisé pendant l'élaboration du plan d'itération qu'à la fin de chaque itération, la fonctionnalité ajoutée devait être complétement implémentée (résolution des bugs, interface complète et finie, temps d'exécution optimisé), testée et refactorisée.

Nous n'avons pas pris en compte un délai supplémentaire pour les bugs, ou pour un changement de décision. Nous avons par exemple dû nous adapter et fusionner les deux projets quand l'assistant a fait une refactorisation du code. Ou quand nous avons décidé de gérer toutes les mises à jour de l'interface à l'aide d'événements envoyés par le serveur. Egalement quand des fonctionnalités ne marchaient plus après en avoir ajouté de nouvelles.

Nous aurions dû compter au moins le double de temps nécessaire à chaque itération et également enlever une ou deux fonctionnalités (telle que le report de messages).

Android

Le temps d'apprentissage a également pris beaucoup de temps, aucun de nous ne connaissait réellement le développement sur Android. Nous avons commencé par lire quelques tutoriels, puis nous avons appris sur le tas en implémentant les fonctionnalités. La vélocité du groupe a donc augmenté au fur et à mesure de la conception de l'application car nous comprenions de mieux en mieux les outils/langages.

Conclusion

… c'était trop cool

Améliorations possibles

Durant toute la phase de développement, nous avons remarqué des fonctionnalités manquantes, ou des parties de notre application qui pourraient être améliorés.

Ci-dessous une liste non-exhaustive des améliorations qu'on a imaginées :

- Cryptage
  - La sécurité des informations est un sujet d'actualité. Une future version de notre application devrait proposer un cryptage des données de bout en bout, afin de garantir la confidentialité des échanges des utilisateurs.

- Client IOS et Windows Phone
  - La première version de notre application est disponible uniquement sur le système Android de Google. Cela parce qu'il d'agit du système d'exploitation mobile le plus répandu sur le marché. Mais pour toucher un public plus large, il faudrait que nous soyons présents sur les autres plateformes.

- Client Web
  - Outre les clients mobiles, il y a aussi des clients potentiels sur les plateformes fixes. Accéder à ses messages depuis un site web pourrait être très pratique.

- Messages multimédias
  - Les messages textes sont essentiels mais très limités. Une amélioration possible de notre application serait de proposer l'envoi d'émojis, d'images, de contenu audio et vidéo, etc.

Annexe

Table des illustrations

Figure 1 : Logo de l'application 4
Figure 2 : Diagramme d'activité général 7
Figure 3 : Diagramme de contexte général 8
Figure 4 : Modèle de domaine client 19
Figure 5 : Modèle de domaine serveur 20
Figure 6 : Modèle conceptuel de la base de données 21
Figure 7 : Modèle relationnel de la base de données 24
Figure 8 : Concept d'interface pour la fenêtre de login 25
Figure 9 : Concept d'interface pour l'écran de gestion d'un groupe 25
Figure 10 : Concept d'interface pour la fenêtre de discussion 26
Figure 11 : Concept d'interface pour la fenêtre d'affichage de groupes 26
Figure 12 : Diagramme des activités et de leurs adapteurs 30
Figure 13 : Diagramme des fragments et de leur adapteur 31
Figure 14 : Diagramme du paquet communications 32
Figure 15 : Diagramme des interfaces 33
Figure 16 : Diagramme du package Model 34
Figure 17 : Diagramme du service d'évènements 35
Figure 18 : Diagramme de la classe « Utils » 36

Manuel d'utilisation

Installation

Serveur + client

Utilisation

Todo : captures d'écran