Projet Rady

Rapport
Projet de groupe
Champion, Loiseau, Rochat, Schubert
Resp. René Rentsch
HEIG-VD

11 janvier 2017

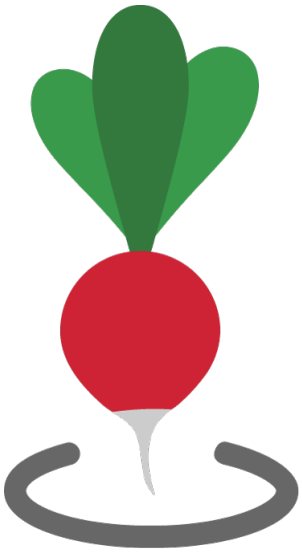


Table des matières

Remerciements

Nous tenons tout d'abord à remercier M. Prof. René Rentsch pour son suivi et ses conseils durant toute la période du travail. Nous voulons également à remercier M. Prof. Olivier Liechti pour son aide dans le choix des technologies.

1 Introduction

Dans le cadre du cours PDG en troisième année du cursus de Bachelor de la HEIG-VD, il a été demandé de réaliser un projet de semestre sur une période de 10 semaines. Après l'acceptation du cahier des charges à la semaine 4, le projet a pu débuter. Celui-ci porte sur la réalisation d'une application mobile permettant de se retrouver facilement entre amis notamment lors d'évènements. L'application se présenterait sous la forme d'une liste d'amis, laquelle permettrait de créer des groupes et de lancer une rencontre. L'utilisateur aurait alors le choix entre une carte en vue du dessus ou une boussole indiquant la direction à prendre. TODO : DECRIRE HIERARCHIE DU RAPPORT

1.1 Objectif

Les objectifs principaux de notre application peuvent être décomposés en trois grandes parties. Premièrement, la gestion des utilisateur, qui comprends la création de compte, la gestion du-dit compte, ainsi que la recherche et l'ajout d'amis. Ensuite, l'intégration de la géolocalisation dans notre application doit permettre, par des appels à une API, aux utilisateurs de se retrouver. Cela constitue la partie centrale de l'application. Enfin, étant donné que des informations sensibles transiteront sur le réseau par le biais de notre application, une attention particulière sera mise vis-à-vis de la sécurité.
Toutes les spécifications précises sont disponibles dans le cahier des charges fourni en annexe.

1.2 Architecture

L'application est composée d'une partie cliente (l'application mobile a proprement parlé), d'un serveur (hébergé sur une machine distante) communiquant avec une base de donnée et faisant appel au service FireBase Cloud Messaging (voir **figure 1**). Le serveur sera à même de gérer la connexion et l'interaction avec plusieurs clients simultanément de manière transparente pour les utilisateurs.
Les données de l'application seront stockées dans une base de données dont l'accès direct se fera uniquement par le serveur. Les clients devront passer par l'API mise à disposition par le serveur pour tout accès aux données, pour des raisons évidentes de sécurité. Enfin le serveur fera appel au service FireBase Cloud Messaging (FCM) pour envoyer les notifications de push aux utilisateurs de l'application.

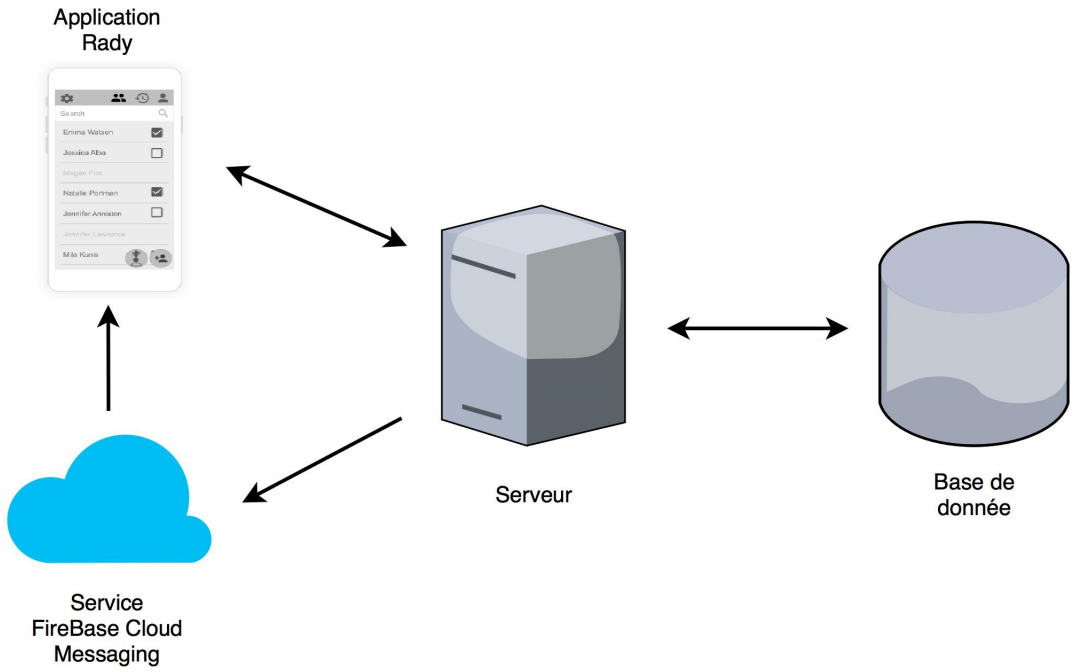


FIGURE 1 – Architecture simplifiée

La conception générale de l'application a été divisée dans ces mêmes différentes parties afin de réduire les dépendances au niveau du développement (temps d'attente sur le travail des autres) ainsi que le nombre de conflit sur le gestionnaire de version (git).

1.3 Choix des technologies

Le choix des technologie a été un point clé lors de l'élaboration du cahier des charges et les premières étapes du développement à proprement parlé du projet. Nous avons essayé de choisir les technologies de manière à ce qu'elles soient maintenables et performantes.

- **Interface :** Pour l'interface nous pensions tout d'abord à utiliser **MeteorJS** [3], étant donné qu'il s'agit d'un framework pemettant un developpement "cross-plateform" performant. Cependant notre choix s'est finalement porté sur un autre framework "cross-plateform" **Ionic2** [4], étant donné que **MeteorJS** est un framework dit "fullstack". Cela signifie qu'on aurait du développer avec ce framework de bout en bout, ce qui aurait pu compromettre le développement en cas de difficulté sur une des parties.
- **Serveur :** Concernant le serveur nous avons choisi d'utiliser **Django** [5] pour ses performances, sa sécurité, sa scalabilité et sa maintenabilité.
- **Base de données :** Le choix de la base de données s'est porté sur **PostgreSQL** car ce système de gestion de base de données (SGBD) est stable, fonctionne sur de nombreux OS et peut stocker des types de données dit "modernes". De plus ce SGDB fait parti des plus conforme aux norme ANSI SQL.
- **Serialisation :** Pour la sérialisation des données et leur transfert nous avons opté pour **JSON**, car en comparaison avec **XML**, le **JSON** est plus léger, plus simple et peut directement être utilisé par le TypeScript (Ionic2), ainsi qu'être stocké tel quel dans le SGBD.
- **Notifications :** Le systeme de notification permettant au serveur de maintenir les utilisateurs à jour concernant les interactions inter-utilisateur se fera via **Firebase Cloud Messsaging**[1]. FCM (anciennement Google Cloud Messaging) nous offre un service de transmission de message fiable, une plus grande simplicité de développement ainsi qu'une prise en charge multi-plateforme.

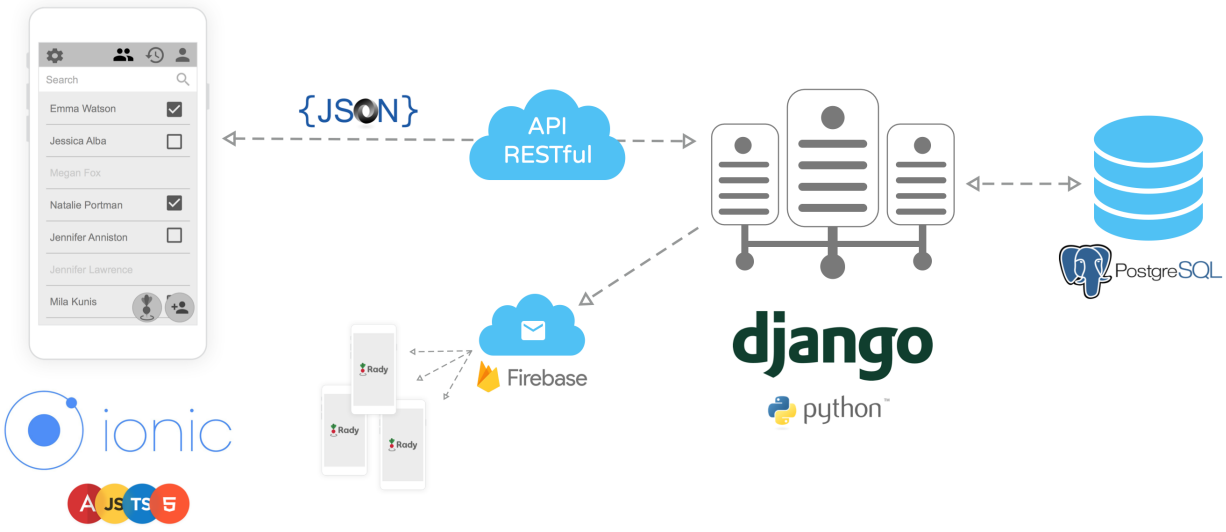


FIGURE 2 – Architecture et Technologies

2 Cadre de développement

Le client de l'application a été développé en TypeScript(2.0.9) et angular2(2.2.1), avec node.js(6.9.1) et le gestionnaire de paquet npm(3.10.9). Le serveur quand à lui a été développé avec python(3.5.2). Nous avons également utilisé un service web d'hébergement et de gestion de développement de logiciels : GitHub, avec le logiciel de gestion de version Git, afin de gérer la mise en commun du code et l'intégration des différentes fonctionnalités au projet. Par ailleurs nous avons aussi utilisé un serveur d'intégration continue (Travis) afin de pouvoir vérifier en temps réel le bon déroulement de notre projet. Enfin nous avons mis en place des séries de tests visant à faire en sorte que l'API fournie par le serveur soit utilisable et ne contienne pas de faille.

3 Application Client

Lors du développement de notre application, la première étape a été de nous familiariser avec les différentes technologies et "frameworks" nécessaire à la réalisation du produit final. Ensuite nous avons décidé de réaliser une maquette de l'interface que l'utilisateur final utilisera. Cette maquette a été conçue pour être simple et permettre une utilisation intuitive de l'application. Enfin il a fallut développer les menus et interfaces de l'application et lier les contrôleurs aux différentes fonctionnalités.

3.1 Frameworks

3.1.1 Ionic2

FIGURE 3 – Ionic2 logo

Notre application a été réalisée avec le framework Ionic2. Ce framework, disponible depuis septembre 2016, se distingue de sa version précédente Ionic en incluant la syntaxe ES6, une nouvelle structure de fichier et en intégrant une CLI plus perfectionnée permettant la génération automatique de pages. L'interface graphique est décrite par un fichier HTML utilisant des composants spécifiques à Ionic2 :

```
<ion-component>
```

Les balises permettent de placer les différents éléments de l'interface un peu à la manière dont on structure une page en HTML standard. La personnalisation des éléments se fait grâce à des styles CSS, spécifiés soit dans les attributs des éléments, soit via des feuilles de style externes dans les fichiers SCSS. Les attributs du SCSS diffèrent un peu des attributs CSS standards, ils permettent d'utiliser les SASS (Syntactically Awesome Style Sheets). Les fichier SCSS seront utilisés pour régénérer le CSS nécessaire à l'affichage des styles. A chaque fichier HTML et SCSS vient s'ajouter un fichier de code TypeScript, qui sert de contrôleur pour l'interface. Dans le HTML on définit les différents évènements de l'interface (par exemple un clic sur un bouton, ou un changement d'état d'un élément de type bascule) en déclenchant un appel de méthode au niveau du fichier contrôleur (le ficher TypeScript), qui gère ensuite l'évènement. Il est également possible d'ajouter et modifier des balises HTML directement depuis le contrôleur TypeScript afin de rendre l'interface plus dynamique, en la faisant par exemple réagir aux interactions de l'utilisateur ou aux réponses du serveur.

Voici la structure des sources de l'application mobile :

```
src/ ..... répertoire contenant les sources de l'application
├── app/ ..... répertoire contenant les liens des différents composants
│   ├── app.component.ts ..... fichier définissant la page d'accueil
│   ├── app.html ..... permet de lancer la page d'accueil
│   ├── app.modules.ts ..... permet de charger les différents modules et interfaces
│   ├── app.scss ..... styles portant sur toute l'application
│   └── main.ts ..... fichier permettant le lancement de l'application
├── assets/ ..... répertoire contenant les éléments utiles
├── lib/ ..... répertoire contenant les librairies utilisées
├── models/ ..... répertoire contenant les modèles(classes) utilisés
├── pages/ ..... répertoire contenant les différentes interfaces
│   ├── add-contact/ ..... répertoire contenant les éléments pour l'interface d'ajout de contact
│   │   ├── add-contact.html/ ..... code HTML définissant les composants de l'interface
│   │   ├── add-contact.scss/ ..... styles s'appliquant aux composants HTML
│   │   └── add-contact.ts/ ..... logique appelés par les composants HTML
│   ├── contact-list/ ..... répertoire contenant les éléments pour l'interface de gestion de contacts
│   │   ├── contact-list.html/ ..... code HTML définissant les composants de l'interface
│   │   ├── contact-list.scss/ ..... styles s'appliquant aux composants HTML
│   │   └── contact-list.ts/ ..... logique appelée par les composants html
│   └── <...> ..... autres fichiers d'implémentation
├── providers/ ..... répertoire contenant les services utilisés par l'application
├── theme/ ..... répertoire contenant les fichiers .scss appliqués sur toute l'application
├── declarations.d.ts ..... fichier de déclaration pour le compilateur TypeScript
├── index.html ..... première page chargée par ionic permettant le chargement du reste de l'application
├── manifest.json ..... données pour l'affichage de l'application
└── service-workers.js ..... fichier contenant un script qui tournera en tâche de fond
```

On observe que les interfaces sont dans le dossier pages avec leurs fichiers HTML, SCSS, et TS correspondants. Dans le dossier app/ on retrouve le lien entre les différentes pages et l'application en elle-même. Ionic2 propose une hiérarchie intuitive regroupant les éléments par nature puis par fonctionnalité (librairies, models ...) afin de simplifier le développement.

3.1.2 Angular2

FIGURE 4 – Angular2 logo

Angular2 est un framework JavaScript conçu pour simplifier le développement d'interfaces web. Ionic2 nous permet d'utiliser la version 2 de ce framework elle aussi sortie en septembre 2016. Angular2 nous offre la possibilité de réaliser du "data binding" bidirectionnel. C'est-à-dire que lorsque les données affichées dans la vue sont modifiées cela affecte directement le contrôleur qui s'est mis à jour. Ce framework nous donne aussi accès à des directives rendant le code plus extensible et modulable, et nous permet d'utiliser l'injection de dépendance pour l'appel à des services initialisés ailleurs dans le code.

3.1.3 Cordova

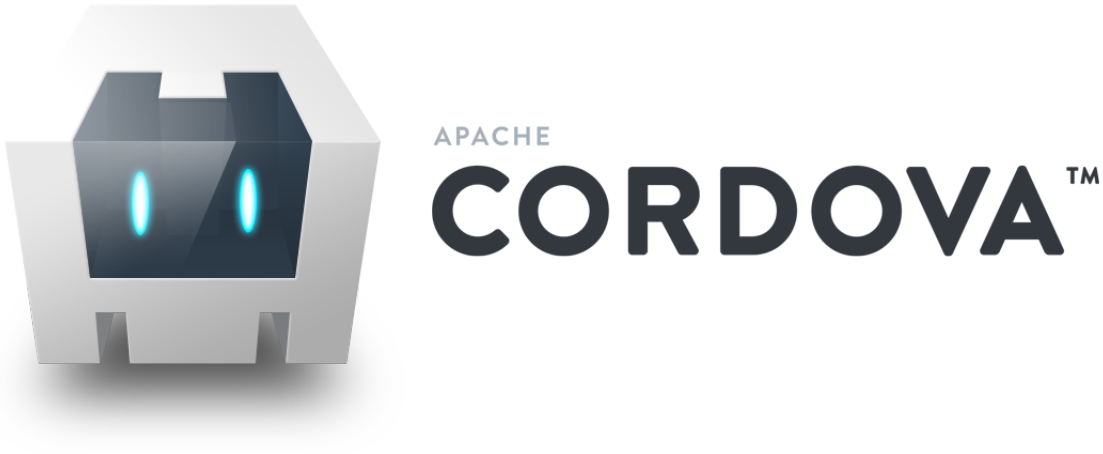


FIGURE 5 – Leaflet logo

Apache Cordova est un framework open-source qui permet de créer des applications pour différentes plateformes (Android, Firefox OS, iOS, Ubuntu, Windows 8...) en HTML, CSS et JavaScript. C'est lui qui sera appelé par Ionic pour "build" et déployer l'application directement sur le matériel. Ce framework permet d'avoir accès aux différentes fonctionnalités et éléments du téléphone tel que la boussole, l'accéléromètre, la géolocalisation ou encore l'appareil photo.

3.1.4 Leaflet

Leaflet est une librairie open-source JavaScript pour le développement d'application web de cartographie.

FIGURE 6 – Leaflet logo

Cette librairie nous permet non seulement d'afficher une carte, mais aussi des marqueurs que nous utilisons pour afficher les positions des utilisateurs, ainsi que les lieux que nous souhaitons localiser. De plus celle-ci est extrêmement configurable, on peut en effet choisir d'utiliser nos propres éléments à afficher tel que des marqueurs personnalisés ou encore utiliser divers générateurs de carte tel que mapbox, openstreetmap ou encore google map.

FIGURE 7 – Mapbox logo

3.2 Design

Il a tout d'abord fallu établir le design de l'application comprenant les différentes interfaces à afficher, l'ordre d'affichage des ces interfaces ainsi que les animations et écrans de chargement à afficher. Ainsi nous avons commencé le projet en établissant un flux d'utilisation :

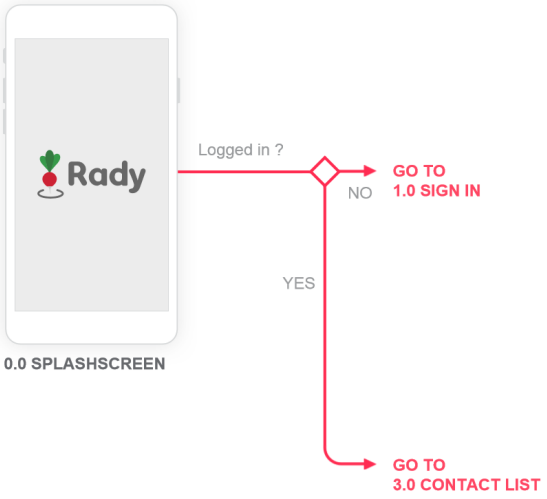


FIGURE 8 – User flow - Loading

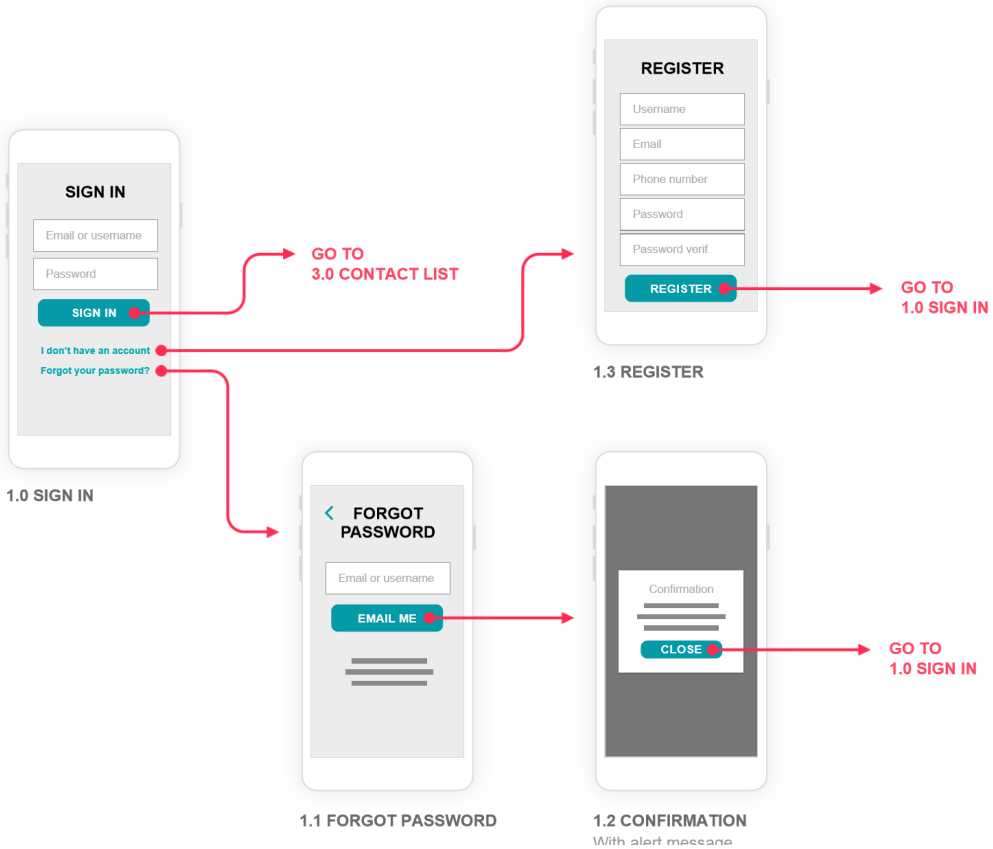


FIGURE 9 – User flow - Sign in

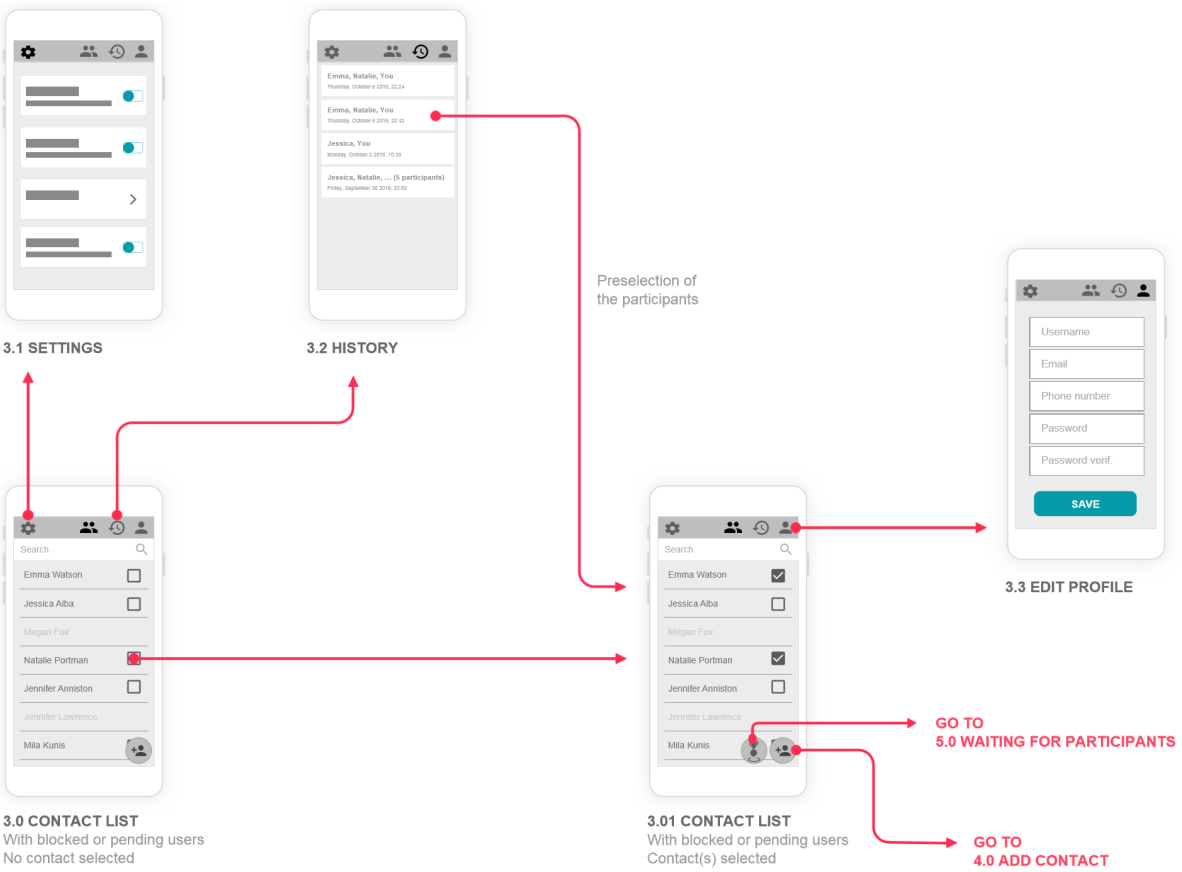


FIGURE 10 – User flow - Main app

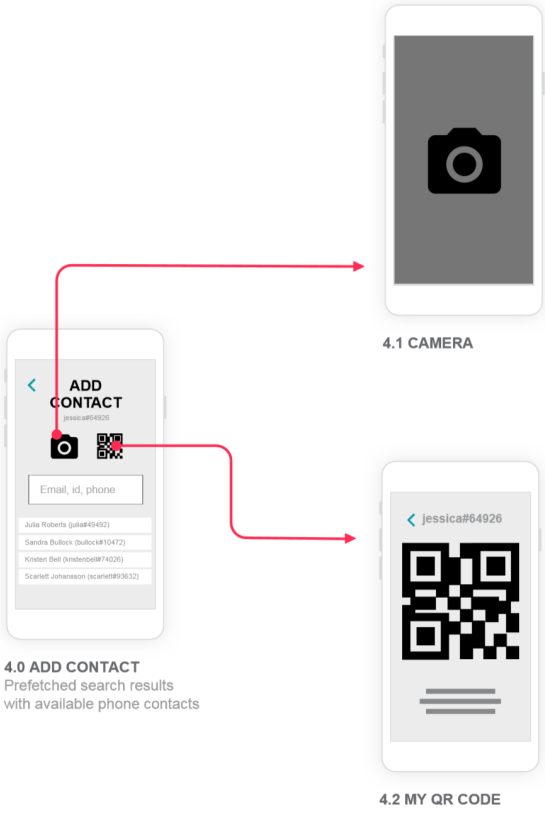


FIGURE 11 – User flow - New Contact

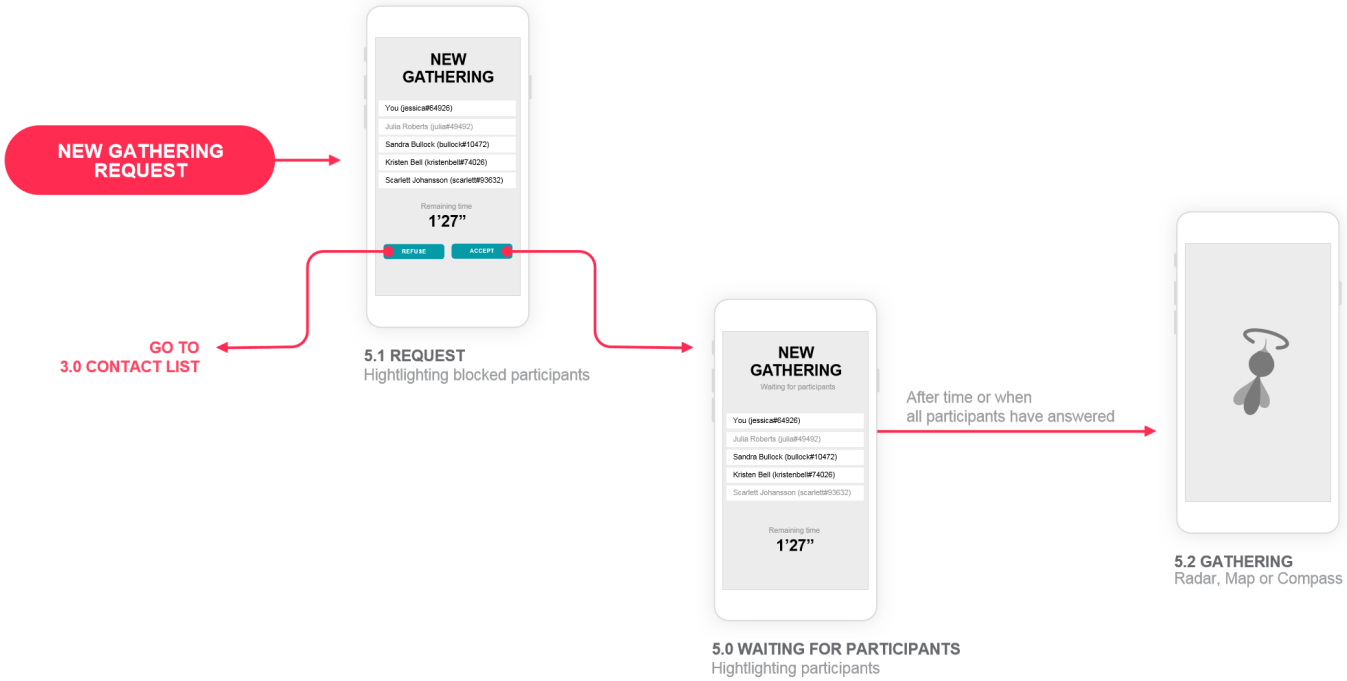


FIGURE 12 – User flow - Gathering

3.3 Développement de l'interface

3.3.1 Création des pages

Le développement des différentes pages a été réalisé en parallèle en répartissant les différents écrans constituant les interfaces finales de notre application. La construction des interfaces des menus constituaient la partie la plus simple du développement car ils ne font appel que à des composants simples définis par Ionic2 et seulement quelques directives Angular2. Nous nous sommes efforcés de réaliser des pages les plus proches possible du design réalisé précédemment afin de garder une simplicité de l'interface tant au niveau du développement que au niveau de l'utilisation finale.

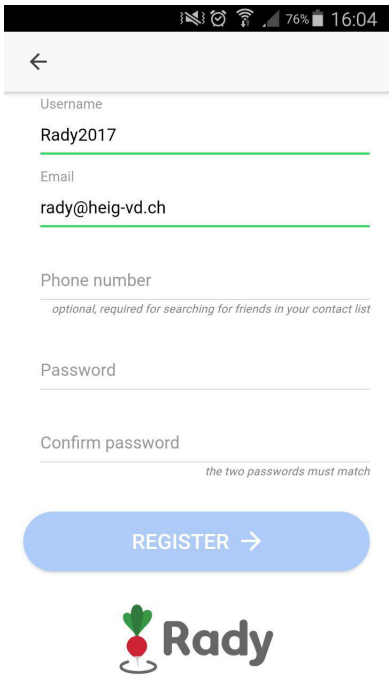


FIGURE 13 – Capture d'écran - Enregistrement

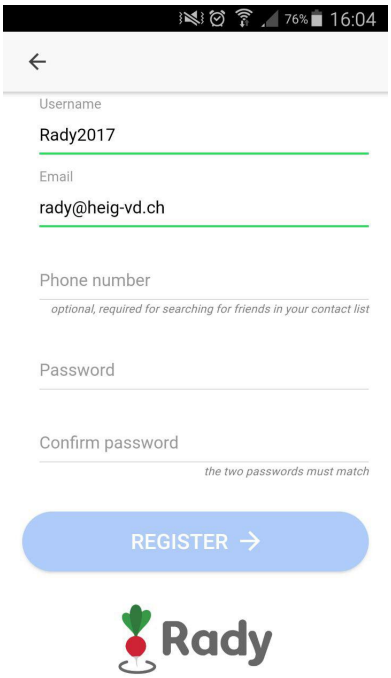


FIGURE 14 – Capture d'écran - Test

Ensuite est venu l'implémentation des différentes fonctions appelées par les éléments graphiques (vérification des champs, communication avec le serveur, changement de page). Pour ce faire nous avons du créer et utiliser différentes librairies et validateurs.

3.3.2 Validateurs

Pour éviter d'effectuer des appels serveur inutile nous avons choisi de faire des tests sur la forme des saisies utilisateur directement dans l'application avant de les envoyer au serveur via l'API afin de diminuer le nombre de réponses négatives. Pour effectuer ces tests nous avons donc eu besoin d'une librairie de validateurs pour nous permettre de définir si la saisie possède la bonne forme ou non. Étant donné que nous n'avons pas trouvé de validateurs correspondant à nos attentes nous avons décidé de les implémenter à la main.

```
1   /**
2   * Email validator
3   * Check if the control is a valid email
4   * @param emailcontrol
5   * @param msg
6   */
7   public static email(emailcontrol: string, msg: string) {
8     return (group: FormGroup): ValidatorResult => {
9       if(group.get(emailcontrol).value.length === 0)
10        return null;
11      let errors = {};
12      let regExp = /[a-z0-9!#$%&'*+/=?^_`{|}~-]+(?:\.[a-z0-9!#$%&'*+/=?^_`{|}~-]+)*@(?:[
            a-z0-9](?:[a-z0-9-]*[a-z0-9])?\.)+[a-z0-9](?:[a-z0-9-]*[a-z0-9])?/;
13      if(!regExp.test(group.get(emailcontrol).value))
14      errors[emailcontrol] = msg;
15      return Object.keys(errors).length === 0 ? null : errors;
16    }
17  }
```

Dans ce validateur pour les e-mail nous utilisons une expression régulière, afin de déterminer si l'e-mail entré par l'utilisateur possède une forme correcte. Lorsque le validateur détecte que l'email n'est pas valide, il renvoie une erreur à l'utilisateur qui est instantanément averti. Nous utilisons aussi un validateur personnalisé pour les numéros de téléphone afin de déterminer si le numéro est valide suivant l'indicatif du pays.

```
1   /**
2   * Phone validator
3   * Check if the control the a valid phone
4   * @param phonecontrol
5   * @param countrycontrol
6   * @param msg
7   */
8   public static phone(phonecontrol: string, countrycontrol: string, msg: string) {
9     return (group: FormGroup): ValidatorResult => {
10      if(group.get(phonecontrol).value.length === 0)
11        return null;
12      let errors = {};
13      try {
14        const phoneUtil = libphonenumber.PhoneNumberUtil.getInstance();
15        const phoneNumber = phoneUtil.parse(group.get(phonecontrol).value, group.get(
              countrycontrol).value);
16        if(!phoneUtil.isValidNumber(phoneNumber))
17          errors[phonecontrol] = msg;
18        } catch(e) {
19          errors[phonecontrol] = msg;
20      }
21      return Object.keys(errors).length === 0 ? null : errors;
22    }
23  }
```

Cette fois-ci nous n'utilisons pas d'expression régulière mais nous faisons appel à une librairie externe : libphonenumber. Cette librairie permet de passer le numéro de téléphone sous forme canonique, avant de vérifier la cohérence de celui-ci ce qui nous permet de garantir une uniformité de la nature des numéros de téléphone d'un utilisateur à un autre.

3.3.3 Rencontres

Les rencontres sont le point central de l'application. Leur conception tourne autour de plusieurs composants qui interagissent entre eux, afin de mener à bien tous les utilisateurs participants vers la destination choisie. Voici un schéma montrant ces différentes parties :

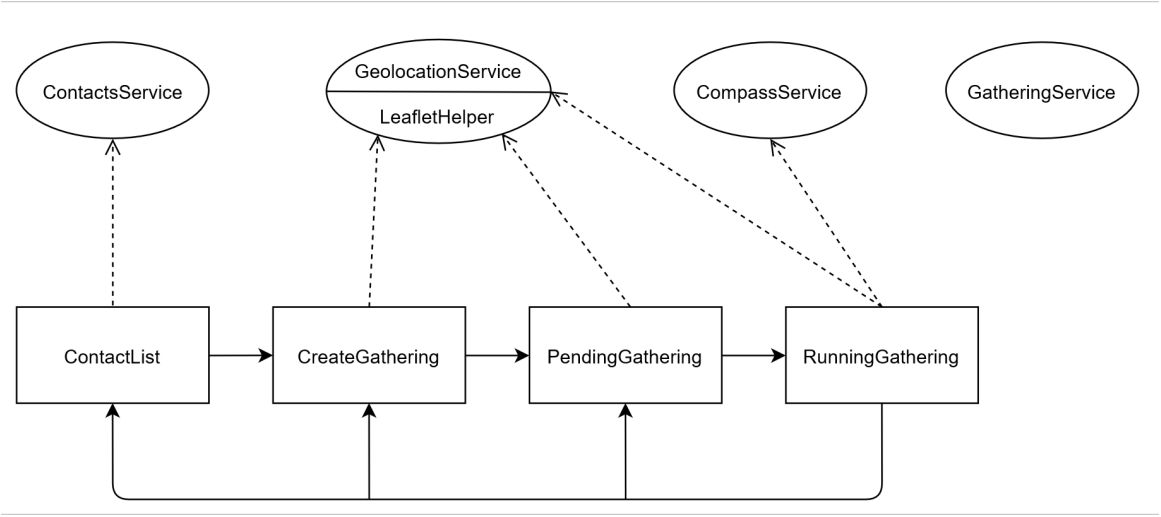


FIGURE 15 – Interactions des composants pour les rencontres

Sur ce schéma les rectangles sont les pages, avec les flèches pleines indiquant leurs successeurs et les flèches en trait-tillé mettant en évidance les services utilisés, qui eux sont les ovales. Les services flottants, comme `GatheringService` et `AuthService`, sont des services utilisés par toutes les pages.

Penchons-nous un peu sur `GatheringService`, ce service s'occupe de l'intégrité et des appels au serveur en ce qui concerne la rencontre en cours. Il est important de noter qu'il s'enregistre auprès de `NotificationService`, afin de recevoir les notifications *push* concernant les rencontres (qui elles-mêmes sont reçues de `PushService`).

Pour mieux comprendre ce schéma, nous allons faire un exemple des interactions dans le cas suivant :

— il y a deux utilisateurs dans la rencontre
  — le créateur de la rencontre, nommons-le **A**
  — l'invité de la rencontre, nommons-le **B**
— la destination est un point fixe sur la carte choisi par **A**
— la rencontre va à son terme, il n'y a pas d'interruptions/annulations/etc.

Voyons ce que ça donne :

A	**B**
lance l'application	
`ContactsService` récupère la liste des contacts	
se trouve sur `ContactList`	
choisi **B** dans sa liste	
lance la création de rencontre	
`GatheringService` est mis à jour	
se trouve sur `CreateGathering`	
choisi le mode et la destination	
lance la rencontre	
se trouve sur `PendingGathering`	reçoit une notification de nouvelle rencontre
	récupère la rencontre à l'aide de `GatheringService`
	accepte de voir la demande de rencontre
	se trouve sur `PendingGathering`
	accepte la demande de rencontre
reçoit une notification d'acceptation	va sur `RunningGathering`
`GatheringService` est mis à jour	*déplacement en cours*
va sur `RunningGathering`	
déplacement en cours	
⋮	⋮
arrive à destination	
	reçoit une notification que **A** est arrivé

reçoit une notification que **B** est arrivé	arrive à destination
reçoivent une notification de fin	
retour à `ContactList`	

Cette exemple est le cas de base, évidemment à chaque instant l'un et l'autre peuvent quitter la rencontre avant de l'avoir finie. Dans ce cas les notifications adéquates seront envoyées par le serveur et reçues par les utilisateurs.

Affichage de la carte Nous n'avons pas encore parlé des services `CompassService`, `GeolocationService` et `LeafletHelper`, voici un résumé :

`CompassService` :	Récupération de l'orientation du mobile par rapport au pôle nord.
`GeolocationService` :	Récupération de la position du mobile sur terre (latitude, longitude).
`LeafletHelper` :	Aide à la création de carte à l'aide de la bibliothèque *leafletJS*.

Les services de boussoles et surtout de géolocalisation sont gourmands en processus, et donc en batterie du mobile. À cause de cela il ne sont activés que lorsque c'est nécessaire.

Pour parler de la carte, à la base nous voulions trois vues différentes : radar, boussole et carte. Nous sommes arrivés à la conclusion qu'il était plus facile, intuitif et pratique de fusionner les trois en un :

— une carte avec notre position et la destination
— une flèche sur notre position indiquant notre orientation (boussole)
— la position des autres participants de la rencontre sur la carte (radar)

Ainsi tout les informations sont regroupées.

Notons encore que *leafletJS* ne permet pas d'opérer une rotation sur la carte, ainsi afin de savoir dans quel sens aller quand même, nous avons tirer une ligne droite entre notre position et la destination.

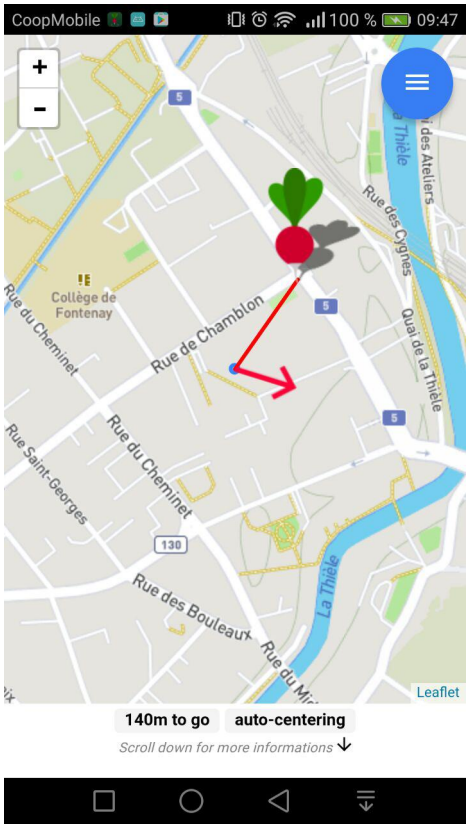


FIGURE 16 – L'utilisateur est trop orienté sur la droite

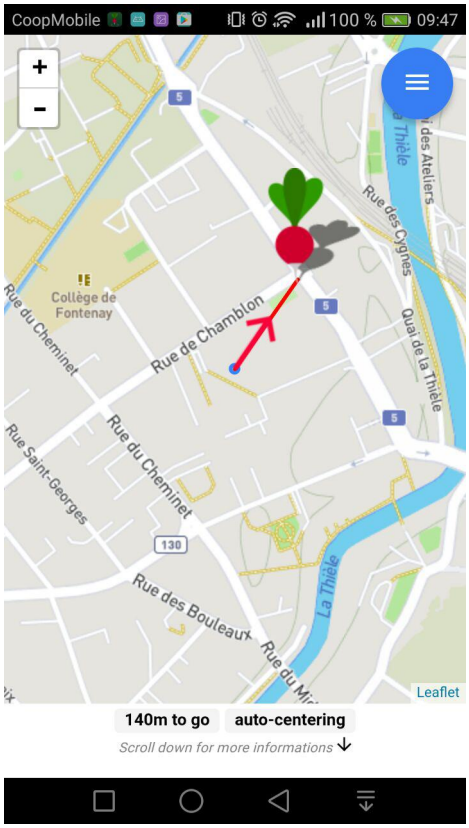


FIGURE 17 – L'utilisateur est en face

Avec ça, il suffit d'aligner la boussole et cette ligne afin de connaître la direction générale vers la destination :

4 Serveur et Communication

4.1 Fonctionnement Général

Afin d'établir le lien entre la partie serveur et la partie cliente, nous avons défini un protocole de communication qui s'occupe de gérer tous les échanges. Celui-ci définit les différents échanges effectués par les applications clientes et le serveur (autant les demandes de ressources que les transmissions de réponse), mais assure également le service de notification de FireBase Cloud Messaging.

Dans ce protocole nous avons définit que les clients ne communiqueraient jamais directement entre eux ni directement avec la base de donnée ou le service de notification. Toutes leurs requêtes doivent passer par le serveur avec lequel ils communiquent via une API REST. Celui-ci s'occupe de traiter avec la base de données et FireBase Cloud Messaging de manière complètement transparente pour l'utilisateur.

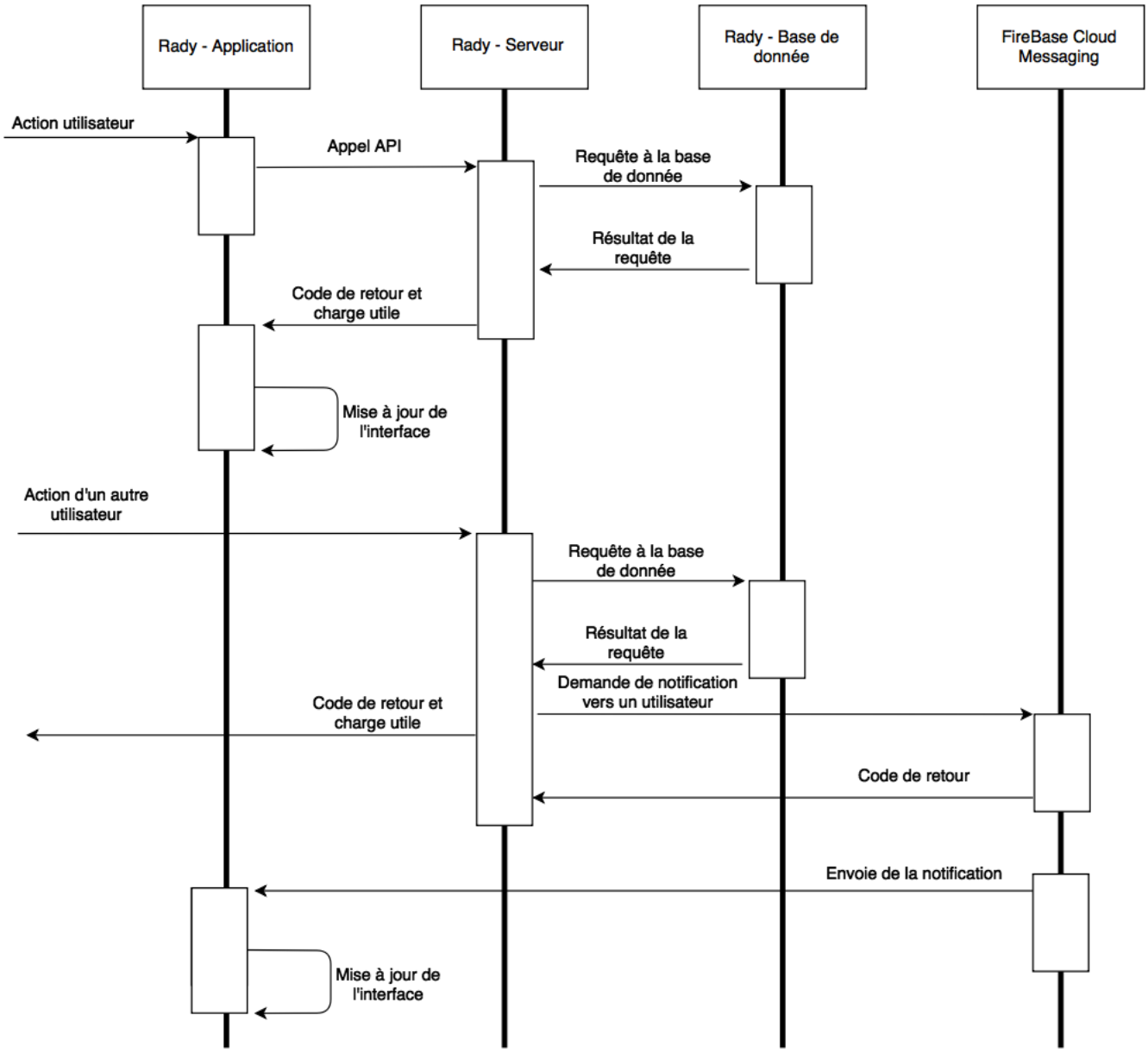


FIGURE 18 – Diagramme de séquence communication

4.2 Base de donnée

Au lancement du serveur, celui-ci fait appel aux services de Django pour créer et déployer la base de données. Un script permet de lancer le serveur et donne des instructions au framework depuis la ligne de commande afin de trouver les différents modèles nécessaires à l'instanciation des tables. Django nous permet également de nous rendre complètement indépendant du moteur de base de données que nous souhaitons utiliser. Grâce à l'utilisation de Fabric [6], le déploiement est entièrement automatisé.

4.2.1 Modèles

Les entités représentées par les modèles définissent la structure de notre base de données, mais sont aussi le coeur de notre application. En effet, celles-ci constituent les éléments sérialisables échangés à travers le réseau. Ces différents modèles sont définis à la fois en python du coté du serveur ainsi que dans le code TypeScript de l'application Ionic2. Cela permet à notre application de communiquer via un "dialecte" commun avec le serveur malgré une différence de langage de programmation.

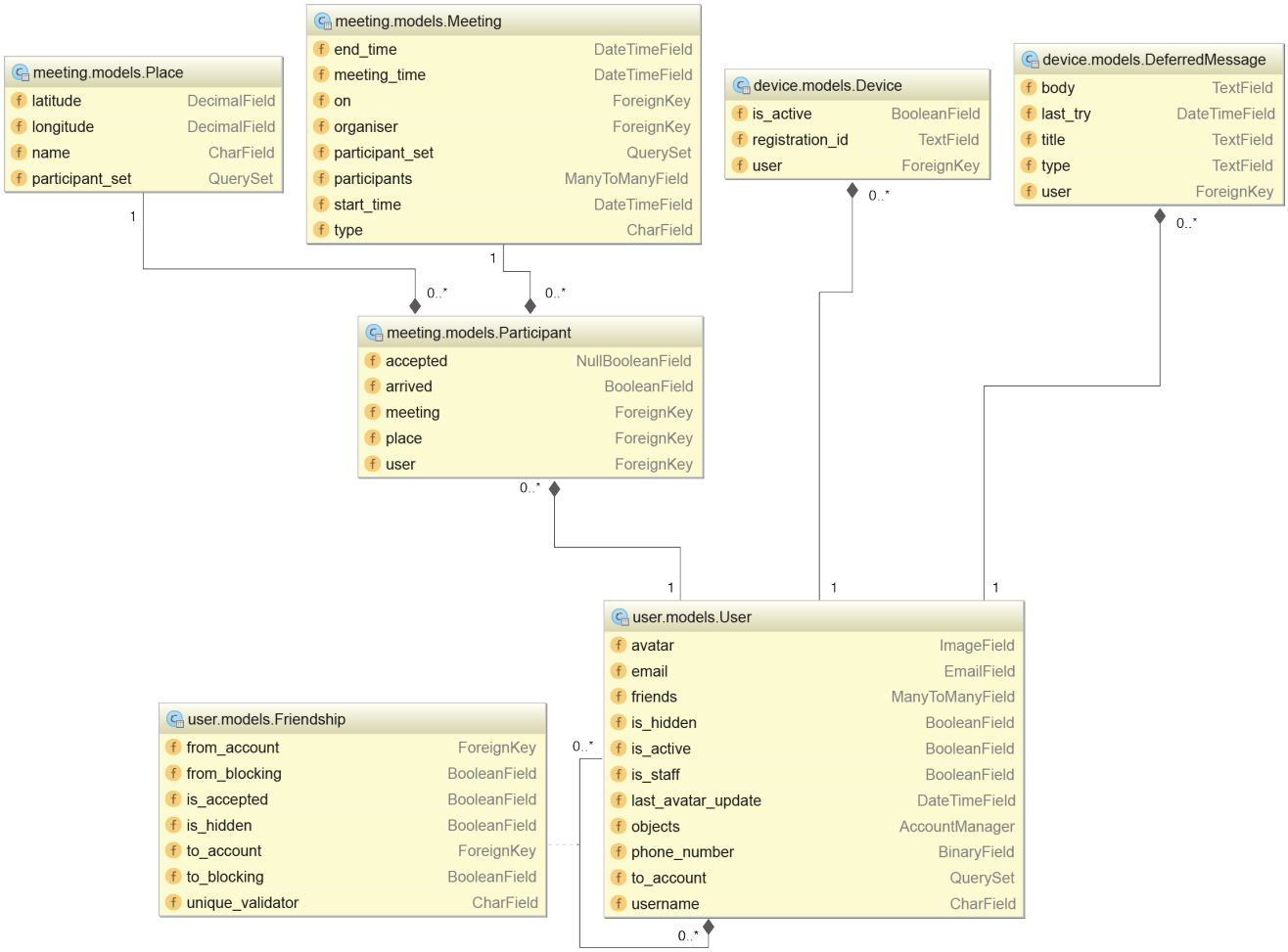


FIGURE 19 – Modele de domaine

4.2.2 Création avec Django

Tout d'abord il faut charger les service de Django puis lui demander d'effectuer la création des tables depuis les modèles. Cela s'opère via une ligne de commande lue et transmise à Django dans le fichier : manage.py

```
1     if __name__ == "__main__":
2         os.environ.setdefault("DJANGO_SETTINGS_MODULE", "rady.settings.debug")
3         try:
4             from django.core.management import execute_from_command_line
5         except ImportError:
6         # The above import may fail for some other reason. Ensure that the
7         # issue is really that Django is missing to avoid masking other
8         # exceptions on Python 2.
9         try:
10            import django # noqa
```

```
        except ImportError:
        raise ImportError(
            "Couldn't import Django. Are you sure it's installed and "
            "available on your PYTHONPATH environment variable? Did you "
            "forget to activate a virtual environment?"
        )
        raise
    execute_from_command_line(sys.argv)
```

Les paramètres entrés en ligne de commande indiquent à Django les opérations à effectuer, c'est à dire la création des différentes tables depuis les modèles pré-existants.

Django parcours alors nos applications par réflexion et en génère une représentation intermédiaire avant d'adapter celle-ci au moteur de base de données utilisé. La liaison entre Django et PostgreSQL se fait alors de manière toute simple, en spécifiant à Django les paramètres de connections. Par exemple :

```
    DATABASES = {
        "default": {
        "ENGINE": "django.db.backends.postgresql_psycopg2",
        "NAME": "rady",
        "USER": "rady",
        "PASSWORD": "CENSORED",
        "HOST": "localhost"
        }
    }
```

L'utilisation de Django nous permet donc de rester indépendant de la base de données, et changer de moteur revient à changer ces informations de connections.

4.3 API

L'API mise à disposition des usagers est une API REST. Cela signifie que le service fourni possède une interface uniforme, c'est à dire que chaque ressource est identifiée unitairement, que la manipulation de ces ressources se fait via des représentations et que les messages auto-descriptifs expliquent leur nature. De plus cette API met à disposition un implémentation des méthodes du CRUD pour les différents endpoints :

- /users/
- /users/me/
- /users/me/avatar/
- /users/friends/
- /users/friends/all/
- /users/friends/blocked/
- /users/friends/hidden/
- /users/friends/pending/
- /users/friends/<pk>/
- /meetings/
- /meetings/<pk>/
- /meetings/places/
- /meetings/places/<pk>/
- /meetings/<pk>/participants/
- /meetings/positions/
- /auth/login/
- /auth/refresh/
- /fcm/devices/
- /statistics/
- /admin/users/
- /admin/users/<pk>/

Nous pouvons observer que ces différents endpoints correspondent directement à des modèles (user, meetings, etc.) ou à des attributs des modèles (avatar, hidden, etc.) qui sont renvoyés ou reçus par l'API. La documentation complète se trouve en annexe.

4.4 Notifications

4.4.1 Fonctionnement

Le système de messages Push, à l'inverse des appels HTTP où le client demande une information au serveur qui lui répond, permet à ce dernier d'envoyer une information à ses clients.

Cette méthode, principalement utilisée sur les appareils mobiles, est asynchrone est ne nécessite pas que l'application cliente soit exécutée en tout temps. En effet, celle-ci peut se trouver en arrière-plan ou même être terminée, c'est un service dédié qui réceptionne les messages et les dispatche vers la bonne application.

Les notifications Push apparaissent comme une notification banale sur l'appareil de l'utilisateur, avec un titre, une description, éventuellement une icône et même un son lors de leur réception. Ce principe est désormais bien connu des utilisateurs de smartphones et tablettes. On peut citer un certain nombre d'exemples d'utilisation : les messageries instantanées, les rappels d'événements, les actions spéciales d'un magasins, une nouvelles news, etc.

Lorsque l'utilisateur clique sur la notification, l'application cible s'ouvre et en traite le contenu. Ces messages n'ont pas pour vocation d'envoyer des données, mais de notifier qu'une action est disponible. Par exemple, une information est disponible au téléchargement sur le serveur. Ceci permet des messages très légers.

Dans le cas où l'application est lancée, la notification n'est pas affichée et l'application est directement informée. Afficher une notification dans le cas contraire amène une forme de sécurité où l'utilisateur doit explicitement demander à ce que l'application soit ouverte.

TODO : schema-sequence-communication.png

L'envoi de messages s'effectue en passant au travers d'un serveur de Push. Les applications clientes doivent inscrire au près de ce serveur qui va leur retourner un token unique d'une certaine validité et correspondant à l'appareil. Ce token doit ensuite être transmis au serveur de l'application. Celui-ci pourra demander l'envoi d'un message au serveur de Push en utilisant ces tokens comme destinataires. Le serveur de Push informe répond ensuite en indiquant si l'envoi a réussi ou s'il a échoué.

4.4.2 Firebase Cloud Messaging

Durant ce projet, nous avons utilisé le serveur Push de Google, Firebase Cloud Messaging (FCM). Cette technique a l'avantage d'être simple a mettre en place, gratuite (bien que le nombre de message soit limité, mais amplement suffisant pour nous) et multi-plateforme. Les clients peuvent être sur iOS, Android ou même Web. En revanche, elle nécessite que les Google Play Services soient installés sur l'appareil (sous forme d'une application). Ceci n'est pas possible sur tous les appareils (vielles versions de l'OS et du matériel, smartphones "anti-Google" comme le Fairphone, désire des utilisateurs, etc.). Néanmoins, FCM permet de couvrir une large part de marché de manière rapide et efficace.

Le service de Google permet également de notifier des appareils par groupes (liste de tokens) ou encore par sujet (par exemple, toutes les personnes intéressées au football ou au jet-ski).

4.4.3 Concernant ce projet

Python possède un large éventail de librairies et PyFCM en est une qui permet de communiquer avec Firebase Cloud Messaging. Celle-ci nous a facilité l'envoi des messages push en s'occupant de toutes les communications réseaux. Concernant la partie mobile, les transactions avec FCM s'effectuent à l'aide d'une librairie pour Cordova.

Les clients s'enregistre auprès de FCM et reçoivent un **registration_id** qui est propre à l'appareil. Ils doivent ensuite le transmettre à notre serveur qui en a besoin pour la communication. Ceci via l'API Rest mise en place. Plus précisément à l'aide du endpoint **POST /fcm/devices/**.

Le serveur est plus complexe. Nous avons utilisé des envois à un appareil et des envois groupés. Les envois "solo" pour les demandes d'amitié et leur réponse, car elles ne concernent qu'une seule personnes. En ce qui concerne les rencontres, plusieurs personnes sont à chaque fois notifiées ce qui rend utile les messages par groupe.

Un point important est que dans le cas où un utilisateur se connecte à son compte sur l'application, qu'il se déconnecte et se reconnecte à l'aide d'un autre compte, il est possible que le registration_id généré par FCM soit le même, car il s'agit du même appareil. Il a donc fallu faire attention à ce que le token soit lié au dernier utilisateur qui l'a utilisé afin de ne pas transmettre des notification à la mauvaise personne.

Les messages contiennent peu d'information (limité à 4KB par FCM). Pour ce projet, nous transmettons un type d'événement, ainsi les identifiants nécessaires afin que le client puisse récupérer les bonnes données.

```
1     meeting.participants.send_message(
2         title="New meeting",
3         body="{} added you to a meeting".format(meeting.organiser.username),
4         data=dict(type="new-meeting", meeting=meeting.id)
5     )
```

Ci-dessus, un exemple d'envoi d'une notification d'une nouvelle rencontre à un groupe d'utilisateurs.

Nous avons mis en place un certain nombre de notifications. Celle-ci informent les utilisateurs lors de demandes d'amis et de rencontres.

Nouvelle demande d'ami

— Data : *{type="friend-request", friendship=id }*

— Envoyé à la personne demandée en ami

Demande d'ami acceptée

— Data : *{ type="friend-request-accepted", friendship=id }*

— Envoyé à la personne ayant effectué la demande d'ami

Nouvelle demande de rencontre

— Data : *{ type="new-meeting", meeting=id }*

— Envoyé aux participants de la rencontres, excepté l'organisateur et les personnes "hidden" qui refusent automatiquement les nouvelles demandes.

Participation acceptée (lorsqu'un utilisateur a accepté la rencontre)

— Data : *{ type="user-accepted-meeting", meeting=id, participant=id }*

— Envoyé aux participants de la rencontres, excepté ceux qui ont déjà refusé la rencontre.

Participation refusée (lorsqu'un utilisateur a refusé la rencontre)

— Data : *{ type="user-refused-meeting", meeting=id, participant=id }*

— Envoyé aux participants de la rencontres, excepté ceux qui ont déjà refusé la rencontre.

Participation annulée (lorsqu'un utilisateur a annulé sa participation à une rencontre après l'avoir acceptée)

— Data : *{ type="user-canceled-meeting", meeting=id, participant=id }*

— Envoyé aux participants de la rencontres, excepté ceux qui ont refusé la rencontre.

Participant arrivé (lorsqu'un utilisateur a signalé son arrivée à la rencontre après l'avoir acceptée)

— Data : *{ type="user-arrived-meeting", meeting=id, participant=id }*

— Envoyé aux participants de la rencontres, excepté ceux qui ont refusé la rencontre.

Meeting démarré

— Data : *{ type="meeting-in-progress", meeting=id }*

— Envoyé aux participants de la rencontres, excepté ceux qui ont refusé la rencontre.

Meeting terminé

— Data : *{ type="meeting-finished", meeting=id }*

— Envoyé aux participants de la rencontres, excepté ceux qui ont refusé la rencontre.

Meeting annulé

— Data : *{ type="meeting-canceled", meeting=id }*

— Envoyé aux participants de la rencontres, excepté ceux qui ont refusé la rencontre.

5 Sécurité

Le but de notre application étant de permettre aux utilisateurs d'enregistrer leurs informations personnelles puis de se retrouver, via les coordonnées GPS envoyés par leur téléphone, nous nous sommes préoccupé de la sécurité de ces informations. Nous avons donc établit différents protocoles (notamment de test) afin de garantir leur préservation.

5.1 Communication

Nous chiffrons toutes les communications entre les clients et le serveur, afin de garantir que les données de nos utilisateurs, comme le mot de passe ou le numéro de téléphone, ne tombent pas entre de mauvaises mains. Pour ceci, nous nous basons sur le protocole HTTPS, utilisant TLS 1.2. La configuration du serveur, un proxy Nginx, est la suivante :

```
1     listen 443 ssl;
2     ssl_certificate /etc/letsencrypt/live/benschubert.me/fullchain.pem;
3     ssl_certificate_key /etc/letsencrypt/live/benschubert.me/privkey.pem;
4
5     ssl_protocols TLSv1.2;
6     ssl_prefer_server_ciphers on;
7     ssl_dhparam /etc/ssl/certs/dhparam.pem;
8
9     ssl_ciphers 'ECDHE-RSA-AES128-GCM-SHA256:ECDHE-ECDSA-AES128-GCM-SHA256:ECDHE-RSA-AES256
          -GCM-SHA384:ECDHE-ECDSA-AES256-GCM-SHA384:DHE-RSA-AES128-GCM-SHA256:DHE-DSS-AES128-
          GCM-SHA256:kEDH+AESGCM:ECDHE-RSA-AES128-SHA256:ECDHE-ECDSA-AES128-SHA256:ECDHE-RSA-
          AES128-SHA:ECDHE-ECDSA-AES128-SHA:ECDHE-RSA-AES256-SHA384:ECDHE-ECDSA-AES256-SHA384
          :ECDHE-RSA-AES256-SHA:ECDHE-ECDSA-AES256-SHA:DHE-RSA-AES128-SHA256:DHE-RSA-AES128-
          SHA:DHE-DSS-AES128-SHA256:DHE-RSA-AES256-SHA256:DHE-DSS-AES256-SHA:DHE-RSA-AES256-
          SHA:AES128-GCM-SHA256:AES256-GCM-SHA384:AES128-SHA256:AES256-SHA256:AES128-SHA:
          AES256-SHA:AES:CAMELLIA:DES-CBC3-SHA:!aNULL:!eNULL:!EXPORT:!DES:!RC4:!MD5:!PSK:!
          aECDH:!EDH-DSS-DES-CBC3-SHA:!EDH-RSA-DES-CBC3-SHA:!KRB5-DES-CBC3-SHA';
10    ssl_session_timeout 1d;
11    ssl_session_cache shared:SSL:50m;
12    ssl_stapling on;
13    ssl_stapling_verify on;
```

Nous y listons les options TLS 1.2 acceptables par le serveur, pour une protection optimale.

5.2 Tests Unitaires

5.2.1 Tests API

Les tests unitaires de l'API se révèlent très simples et suivent la méthodologie suivante :

1. Vérifier les méthodes acceptables pour le point d'entrée
2. Vérifier les droits requis pour le point d'entrée
3. Vérifier les contraintes d'intégrités de nos modèles (valeurs uniques, etc)
4. Vérifier la validité des actions devant être valides

Cette méthodologie permet de s'assurer que notre implémentation est correcte par rapport à la documentation, qui est la référence lors de l'implémentation des tests.

De plus, afin de s'assurer que nos tests n'ont aucun effet de bord, la base de données est remise à zéro entre chaque test.

5.2.2 Tests du Push

Il est difficile de tester les conversations entre le serveur et les clients de manière automatique. De plus, les messages transitent par les serveurs de Google, ce qui n'améliore pas les choses. Nous avons décidé de simplement simuler l'envoi à ces derniers.

Nous avons implémenté des **mocks** des fonctions d'envoi du module PyFCM. Ceci nous permettait de les remplacer par d'autres fonctions et ainsi contrôler qu'elles étaient correctement appelées. Un autre point extrêmement utile était de pouvoir simuler les réponses de ces méthodes afin de tester le comportement de notre serveur en cas de d'erreur ou de succès des envois.

5.2.3 Coverage

Le serveur, pierre angulaire de notre application et gardien des données de nos utilisateurs nécessite beaucoup plus de vérifications que le côté client. En effet, une erreur sur le client risque de causer quelques désagréments à certains utilisateurs, mais une erreur côté serveur pourrait amener à la divulgation de données confidentielles ou à la perte de données. Nous avons donc testé de manière extensive le serveur, et nous présentons le coverage de nos tests dans le tableau 20 Nous y présentons, par fichier, le nombre de lignes exécutées, le nombre de lignes manquantes, le nombre de branches totales, le nombre de branches partiellement testées et finalement la couverture du fichier.

Name	Stmts	Miss	Branch	BrPart	Cover
admin/___init___.py	0	0	0	0	100%
admin/apps.py	4	0	0	0	100%
admin/serializers.py	8	0	0	0	100%
admin/urls.py	4	0	0	0	100%
admin/views.py	13	0	0	0	100%
auth/___init___.py	0	0	0	0	100%
auth/apps.py	4	0	0	0	100%
auth/permissions.py	15	0	4	0	100%
auth/urls.py	5	0	0	0	100%
auth/views.py	37	6	8	0	82%
device/___init___.py	1	0	0	0	100%
device/apps.py	3	0	0	0	100%
device/fcm.py	10	0	0	0	100%
device/models.py	50	1	24	3	95%
device/serializers.py	8	0	0	0	100%
device/urls.py	4	0	0	0	100%
device/views.py	25	0	2	0	100%
manage.py	14	6	2	1	56%
meeting/___init___.py	1	0	0	0	100%
meeting/apps.py	5	0	0	0	100%
meeting/models.py	34	0	0	0	100%
meeting/serializers.py	119	3	40	1	97%
meeting/signals.py	41	0	26	1	99%
meeting/urls.py	4	0	0	0	100%
meeting/views.py	62	1	4	1	97%
rady/___init___.py	0	0	0	0	100%
rady/settings/___init___.py	23	0	0	0	100%
rady/settings/debug.py	9	0	0	0	100%
rady/urls.py	4	0	0	0	100%
rady/wsgi.py	4	4	0	0	0%
stats/___init___.py	1	0	0	0	100%
stats/apps.py	5	0	0	0	100%
stats/models.py	5	0	0	0	100%
stats/signals.py	15	0	2	1	94%
stats/urls.py	4	0	0	0	100%
stats/views.py	17	0	0	0	100%
user/___init___.py	1	0	0	0	100%
user/apps.py	5	0	0	0	100%
user/models.py	82	10	14	2	83%
user/serializers.py	162	7	42	3	95%
user/signals.py	15	0	4	0	100%
user/urls.py	4	0	0	0	100%
user/views.py	71	0	14	0	100%
TOTAL	898	38	186	13	95%

FIGURE 20 – Test coverage du server

6 Conclusion

6.1 Application fournie

Par rapport au cahier des charges initial (disponible en annexe à ce document), l'application répond à toutes les fonctionnalités prévues hormis l'implémentation de certains algorithmes, dont voici la liste :

— Mode de rencontre sur une personne
— Mode de rencontre en un point optimal
— Radar pour guider les utilisateurs

Les fonctionnalités optionnelles n'ont pas pu être intégrées par manque de temps, cependant leur ajout est tout à fait réalisable. En effet, la conception générale de l'application a été prévue pour être ouverte aux extensions. De plus certaines extensions on déjà été implémentées du coté du backend :

— Le système de gestion de places
— Le partage des positions des utilisateurs
— La système de gestion des avatars

Concernant la stabilité du programme, la majorité des bugs/crashs découverts pendant le projet ont été corrigés. Cependant certains cas posent en ce moment toujours problème :

1. Les notifications lorsque l'application est en background sont effectivement reçue. Ensuite, en foreground, nous n'arrivons pas à être redirigé à l'endroit désiré de l'application.
2. Le SecureService est à null lors du premier lancement de l'application. Il faut donc la redémarrer une deuxième fois pour continuer.
3. Le retour à MainTabs ne fonctionne pas si l'on vient d'une page qui a été chargée depuis un handler d'une notification.

Le premier problème est du à Cordova. Nous avons tenté de le résoudre, sans succès. Le deuxième bug est lui aussi du à Cordova, une exception java est lancée lorsqu'un service est à null, uniquement au premier lancement de l'application. Enfin le dernier problème n'est actuellement sans solution non plus. Après maintes recherches, nous n'avons pas trouvé de réponse à ce soucis. Notons encore qu'il est actuellement impossible de compiler l'application en *release* car soit Ionic, soit Cordova, n'inclus pas tous les fichiers nécessaires.

Après ces 10 semaines intenses de projet, nous sommes relativement satisfaits des résultats obtenus et de notre apprentissage. Cependant, nous avons appris de nos erreurs. Travailler avec des technologies qui ne sont pas encore arrivées à maturité, c'est s'exposer à de nombreux bugs encore non résolus par la communauté, ainsi qu'à une surcharge de travail non seulement en terme d'effort et d'apprentissage mais également en terme de recherche sur ces technologies. Néanmoins nous sommes fiers d'avoir pu obtenir des résultats et une application fonctionnelle. Voici une liste non-exhaustive des améliorations possibles auxquelles nous avons pensé (en plus de la correction des bugs mentionnés) :

— Icônes personnalisées pour chacun (à l'aide de son avatar) indiquant sa position sur la carte lors d'une rencontre
— Affichage des avatars des utilisateurs dans les menus
— Gestion des places favorites
— Fonctionnalités optionnelles du cahier des charges

6.2 Planification

La planification initiale du projet (disponible en annexe) a fonctionné durant les premières semaines. Rapidement nous nous sommes rendu compte que certaines tâches avaient été grandement sous-estimées tandis que d'autre avaient été totalement oubliées. Cela peut s'expliquer par le fait que les technologies que nous avons décidé d'utiliser, bien qu'efficaces, ne disposaient pas d'une quantité d'information suffisante (ou à jour). En effet, puisque certaines, telles que Ionic2 et Angular2, sont sorties en septembre 2016. Elles ne disposaient que d'une documentation peu fournie et d'une faible quantité d'exemples. De plus, nous pouvions difficilement nous renseigner via des sources externes, car celles-ci avaient souvent été utilisées lors des phases de bêta.

Ensuite, certaines tâches se sont révélées plus complexes que prévu ou alors ont requis des sous étapes non spécifiés dans le rapport. Par exemple, réalisation du push pour les notifications qui n'avait pas été pris en compte lors de la planification initiale et qui s'est finalement avéré être un pilier de notre application.

Enfin, une source majeure du retard fût le temps des déploiements et des tests de nos différentes parties du projet. Le temps de déploiement du serveur se compte en minute tout comme celui de l'application mobile. Ajouté à cela le fait de devoir tester le GPS et la boussole nous à forcé à effectuer de nombreux déplacements chrono-phages.

6.3 Travail de groupe

Ce projet fut extrêmement enrichissant, autant du point de vue technique que logistique. Cela nous a à nouveau montré les difficultés liées à la planification et au travail en groupe. De ces problèmes, nous avons pu en retirer de bonnes conclusions :

- La première est que quelque soit nos difficultés lors du projet, il ne faut pas hésiter a en parler avec les autres personnes du groupe. Ces derniers offriront une nouvelle vision au problème et apporteront leurs propres connaissances afin de résoudre les difficultés.

- La seconde est qu'il est préférable de "surdécouper" le projet en petites tâche avec de nombreuses "deadlines", plutôt que d'avoir de longues itérations. En effet lorsque l'objectif est encore lointain, nous avons tendance à prendre du retard qui devra être rattrapé au dernier moment.

- Enfin, nous nous sommes rendus compte qu'une uniformité au niveau de la structure du travail fourni permet une gestion rapide et efficace des problèmes, ce que nous avons durement appris à nos dépends.

Cependant, nous sommes donc, au final, heureux d'avoir pu apprendre de nos erreurs et participer à l'élaboration de A à Z d'un projet complet et complexe. C'est une épreuve très enrichissant. Nous avons pu apprendre les uns des autres et mettre en commun nos forces.

Table des figures

Références

[1] Firebase Cloud Messaging (FCM) is a cross-platform messaging solution that lets you reliably deliver messages at no cost..
`https://firebase.google.com/docs/cloud-messaging/`

[2] Developping tool for mobile apps with HTML, CSS , JS Target multiple platforms with one code base free and open source
`https://cordova.apache.org/`

[3] Open source platform for web, mobile, and desktop developpement. ,
`https://www.meteor.com/`

[4] Web app developpement framework for mobile,
`http://ionic.io/`

[5] High-level Python Web framework,
`https://www.djangoproject.com/`

[6] Python library for streamlining the use of SSH for application deployment.
`http://www.fabfile.org/`

[7] JSON Web Token is a compact URL-safe means of representing claims to be transferred between two parties. ,
`https://en.wikipedia.org/wiki/JSON_Web_Token`

[8] Angular2 is a development platform for building mobile and desktop web applications.,
`http://www.angular2.com/`

7 Annexes projet

7.1 Cahier des charges

Projet Rady - Cahier des charges

Projet de groupe
Champion, Loiseau, Rochat, Schubert
Resp. René Rentsch
HEIG-VD

5 janvier 2017

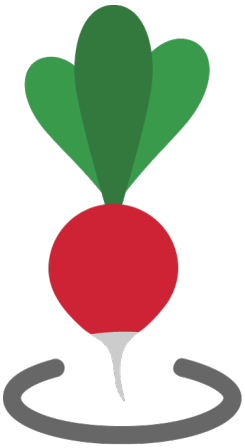


Table des matières

1 Motivation

Nous souhaitons réaliser une application mobile permettant de se retrouver facilement entre amis. L'idée nous est venue des festivals de cet été, dans lesquels il est parfois compliqué de se retrouver au milieu d'une grande foule. L'application se présenterait sous la forme d'une liste d'amis, laquelle permettrait de créer des groupes et de lancer une géolocalisation. L'utilisateur aurait alors le choix entre une carte en vue du dessus ou une boussole indiquant la direction à prendre.

2 But et réalisation

Afin de répondre à la problématique, nous avons imaginé une application qui se présenterait sous la forme d'une liste d'amis. Ceux-ci seront sélectionnables afin de lancer une demande de retrouvailles. Chaque ami sélectionné (plus le sélectionneur) pour une retrouvaille formeront un groupe. Un groupe est spécifique à une retrouvaille, c'est-à-dire qu'une nouvelle retrouvaille concernera un nouveau groupe, même si les membres sont les mêmes.

Idéalement l'application tournera sous Android et sous iOS, cependant si certains tests de faisabilité ne passent pas, l'attention sera porté sur Android (voir section 4.1).

Une ébauche de l'application se trouve à la section 4.3.

3 Fonctionnalités

3.1 Compte utilisateur

- Au premier lancement de l'application, chaque utilisateur sera invité à fournir un pseudonyme, un email, un numéro de téléphone ainsi qu'un mot de passe afin de **créer un compte**
- Chaque pseudonyme sera accompagné d'un numéro afin de permettre à plusieurs utilisateurs d'avoir le **même pseudonyme**
- Un utilisateur pourra **se déconnecter** un **supprimer son compte**
- Un utlisateur pourra **éditer** les informations relatives à son profile
- Si un utilisateur oublie son mot de passe, une **ré-initialisation** sera possible grâce à son email

3.2 Liste d'amis

- L'utilisateur pourra **ajouter un ami** à sa liste, lequel devra accepter la demande de son côté
- L'utilisateur pourra **supprimer un ami** ou **bloquer un ami** de sa liste
- Il sera possible de **rechercher des amis** depuis la liste de contacts du téléphone
- (optionnelle) Chaque utilisateur aura un **QR Code unique**, lequel pourra être scanné (via l'appareil photo) par un autre utilisateur afin d'effectuer un ajout rapide à la liste d'amis

3.3 Sécurité

- **Les champs de la base de données** ne devant pas être lus directement **seront hashés avec un sel** (par ex : mot de passe, numéro de téléphone)
- **Les communications** entre les clients et le serveur, ainsi qu'entre le serveur et la base de données, **seront chiffrées** à l'aide de SSL
- Les utilisateurs ne pourront pas demander au serveur des informations auxquels ils n'ont pas les **droits**
- Un utilisateur peut se mettre en **invisible** pour une certaine durée durant laquelle ses amis ne pourront pas lui envoyer des demandes de retrouvailles
- Un utilisateur pourra toujours **accepter ou refuser** une demande de retrouvailles, celle-ci lui indiquera également si un **ami bloqué** se trouve dans le groupe

3.4 Retrouvailles

- La création d'un groupe de retrouvailles sera simple et rapide pour l'utilisateur
- Une retrouvaille pourra se faire sur un **lieu fixé** à l'avance
- Une retrouvaille pourra se faire **sur une personne** du groupe, cela implique que la position peut se déplacer
- Une retrouvaille pourra chercher le **meilleur lieu de rencontre** (fixé) afin de minimiser la distance à parcourir par chacune des personnes d'un groupe

— L'application sera munie d'une partie **historique** pour lancer d'anciennes retrouvailles rapidement
— L'interface lors des retrouvailles pourra s'afficher sous différentes formes : **boussole, radar** ou encore **carte en vue du dessus**

3.5 Interfaces

— Une **interface administrateur** offrira la possibilité de modifier la base de données ainsi que des **statistiques** sur celle-ci
— L'**interface utilisateur** :
  — sera **claire, simple et rapide**, avec peu de menus et d'actions possibles
  — (optionnelle) aura une **gestion du jour et de la nuit** afin d'offrir les meilleurs contrastes possibles (l'utilisateur aura le choix via les options : jour, nuit, ou automatique)

4 Annexes

4.1 Technologies

Nous pensons utiliser les technologies suivantes :
— **Firebase Cloud Messaging** - Permet de pousser des messages sur les téléphones de manière efficiente.
— **MeteorJS** - Framework Crossplateform pour la création d'application mobile.
— **Open Street Map** - Service permettant l'accès aux données du projet du même nom, pour les données géographiques, cartes, etc.
— **Mobile** (gyroscope, géolocalisation) - Permet la création de boussole et la géolocalisation des utilisateurs.
— **Base de donnée SQL** - Permet le stockage de données liées aux comptes des utilisateurs.
— **Python/Django** - Framework pour la création de serveurs en tout genre, pour la gestion et le lien entre les utilisateurs.

Ces technologies nous sont majoritairement inconnues et seront donc un défi, tant au niveau de l'apprentissage que de la réalisation. La plupart feront l'objet d'un test de faisabilité, notamment les performances de MeteorJS.

Si MeteorJS devait se montrer sous-performant, nous partirons sur du développement Android pur (Java).

4.2 Logo et nom

Nous avons déjà réfléchi au nom ainsi qu'au logo :

Le nom vient du mot *radis* qui est proche de *radar*, puis nous avons simplement mis un *Y* pour *Yverdon*. Le côté drôle (et pratique) est que la forme d'un radis rappelle un peu les pointeurs de Google Maps.

4.3 Flux d'utilisation

Voir fichier en annexe : `heigvd-pdg-groupe1-rady-user-flow.pdf`

4.4 Planification initiale

Voir fichier en annexe : `heigvd-pdg-groupe1-rady-planification-initiale.pdf`

7.2 Journaux de travail

ANNEXE : Journaux de travail

Champion Patrick

Date	Heures	Tâche	Description
25.09.2016	4	Discussion de groupe	Proposition du projet fini
Total	**4**		
04.10.2016	2	Discussion de groupe	Ébauche du cahier des charges
05.10.2016	4	Discussion de groupe	Rédaction du CC, mise en place des journaux de travail
07.10.2016	1	Discussion de groupe	Cahier des charges fini
Total	**7**		
19.10.2016	4	Tests	Premiers essais avec Ionic2
Total	**4**		
27.10.2016	3	Mobile UI	Nettoyage des fichiers inutiles, déploiement sur un device ok, paramétrage de base du projet, création de la branche mobile-ui, création de docs/mobile-ui
28.10.2016	3	Mobile UI	Page Splashscreen ok, page SignIn canvas, docs/mobile-ui mis à jour
Total	**6**		
31.10.2016	3	Mobile UI	Page Register canvas, refactoring SASS
02.11.2016	3	Discussion du groupe	Modèles, API
04.11.2016	1	Mobile UI	Page ForgottenPassword canvas, mise en place des Validators customisés (mais qui ne font rien pour l'instant)
Total	**7**		
09.11.2016	3	Discussion de groupe	Vue sur l'avancement, choix de Firebase Cloud Messaging pour faire du Push, mardi 15.11.2016 réunion pour la présentation intermédiaire
Total	**3**		
14.11.2016	5	Mobile UI	Implémentation des custom validators OK
15.11.2016	1	Mobile UI	Partie NEW CONTACT commencée
15.11.2016	1	Discussion de groupe	Présentation intermédiaire OK
16.11.2016	3	Mobile UI	Partie NEW CONTACT canvas ok (avec navigation, mais aucune fonctionnalité), début de AuthService
18.11.2016	1	Mobile UI	AuthService en cours
Total	**11**		
22.11.2016	2	Mobile UI	AuthService en cours
23.11.2016	1	Discussion de groupe	Présentation intermédiaire
23.11.2016	2	Mobile UI	AuthService fini
Total	**5**		
30.11.2016	4	Mobile UI	Ajout de contact avec la recherche manuel OK, autres pages déclarées
Total	**4**		

07.12.2016	6	Mobile UI	Push sur le client OK
Total	**6**		
14.12.2016	4	Mobile UI	Gestionnaire de notifications OK
Total	**4**		
21.12.2016	4	Mobile UI	Corrections et discussion sur l'organisation
Total	**4**		
03.01.2016	3	Mobile	Liste de contact correctement organisé, création de son manager
04.01.2016	5	Mobile	Liste de contact OK, avec sync depuis le serveur
05.01.2016	6	Mobile	Création des rencontres en cours
06.01.2016	6	Mobile	Création des rencontres OK, mise à jour de Ionic
07.01.2016	6	Mobile	Géolocalisation OK, les cartes zooment automatiquement sur la position, ajout de l'icône et du splashscreen sur mobile, utilisation du logo Rady pour les marqueurs
08.01.2016	7	Mobile	Attente d'une rencontre OK, demande de rencontre bien reçue, distance vers la destination calculée
Total	**33**		
09.01.2016	6	Mobile	destination et direction affichées sur la carte, distance et participants (avec status) affichés, centrage automatique, FAB des actions OK
10.01.2016	10	Rush	gathering ok, prob des retour à MainTabs pas résolu mais origine trouvée, multiples corrections de l'app
11.01.2016	3	Fin	reliure, fin du rapport, ...
Total	**19**		
Total	**117**		

Loiseau Thibaut

Date	Heures	Tâche	Description
25.09.2016	4	Discussion de groupe	Proposition du projet fini
Total	**4**		
04.10.2016	2	Discussion de groupe	Ébauche du cahier des charges
05.10.2016	5	Discussion de groupe	Rédaction du CC, mise en place des journaux de travail
07.10.2016	2	Discussion de groupe	Cahier des charges fini
Total	**9**		
19.10.2016	3	Tests	Lecture de la documentation et prise en main d'Ionic2
21.10.2016	2	Tests	Début du diagramme de gantt
Total	**5**		
28.10.2016	3	Mobile UI	Apprentissage de Ionic2
30.10.2016	4	Administratif	Finalisation du diagramme de gantt et du tableau Heure/Personne

Total	**7**		
02.11.2016	3	Discussion du groupe	Modèles, API
05.11.2016	2	Mobile UI	Création des pages Historique / Edit Profile / Contact-list / Settings
06.11.2016	1	Mobile UI	Création des tabs et Ajout des FABs pour l'ajout de contact et pour le gathering
Total	**6**		
09.11.2016	3	Discussion de groupe	Vue sur l'avancement, choix de Firebase Cloud Messaging pour faire du Push, mardi 15.11.2016 réunion pour la présentation intermédiaire
11.11.2016	1	Mobile UI	Première implémentation des modèles coté client
Total	**4**		
14.11.2016	1	Mobile UI	Ajout de la fonction de recherche dans la contact list
14.11.2016	2	Mobile UI	Mise à jour des fab avec disparition et apparition du gathering
14.11.2016	1	Mobile UI	Premiere implémentation de l'edit profile
15.11.2016	3	Mobile UI	Résolution de bugs
19.10.2016	2	Administratif	Préparation de la présentation intermédiaire
Total	**9**		
23.11.2016	4	Mobile UI	Ajout de l'icone Rady sur le fab —> Doc officielle pas à jour
23.11.2016	1	Discussion de groupe	Présentation intermédiaire
Total	**5**		
30.11.2016	2	Mobile UI	Implémentation de la page de settings
30.11.2016	1	Mobile UI	Ajout de modèles
30.11.2016	2	Mobile UI	Ajout du bouton de logout et ajout de contenus sur historique
Total	**5**		
07.12.2016	3	Mobile UI	Début des pages de gathrering
10.12.2016	2	Mobile UI	Pages de fin de gathering et d'acceptation
Total	**5**		
14.12.2016	4	Administratif	Documentation de l'API avec SWAGGER
16.12.2016	1	Administratif	Mise à jour de l'api SWAGGER
Total	**5**		
21.12.2016	4	Discussion de groupe	Corrections et discussion sur l'organisation
22.12.2016	4	Administratif	Création du rapport
Total	**8**		
03.01.2016	4	Administratif	Amélioration de la documentation swagger
05.01.2016	5	Administratif	Ajout des premier chapitres et refactoring du plan
06.01.2016	2	Administratif	Mise à jour des plugins utilisé pour l'inclusion de code

07.01.2016	3	Administratif	Ajout de la documentation concernant les technologies de l'application
08.01.2016	5	Administratif	Ajout de la documentation concernant le serveur (début) et amélioration du plan
Total	**19**		
10.01.2016	12	Administratif	Finalisation de la documentation
Total	**12**		
Total	**103**		

Rochat Damien

Date	Heures	Tâche	Description
25.09.2016	4	Discussion de groupe	Proposition du projet fini
Total	**4**		
04.10.2016	2	Discussion de groupe	Ébauche du cahier des charges
05.10.2016	4	Discussion de groupe	Rédaction du CC, mise en place des journaux de travail
05.10.2016	2	Initialisation	Création du flux d'utilisation
07.10.2016	1	Discussion de groupe	Cahier des charges fini
Total	**9**		
05.10.2016	3.5	Initialisation	Discussion de groupe, rédaction cahier des charges, création du flux d'utilisation
05.10.2016	2.0	Initialisation	Création du flux d'utilisation
Total	**5.5**		
12.10.2016	0.5	Gestion	Discussion de groupe
	3.0	Apprentissage	Apprentissage de Django et de l'environnement Python
13.10.2016	3.0	Apprentissage	
Total	**6.5**		
19.10.2016	0.5	Gestion	Discussion de groupe
	3.0	Apprentissage	Apprentissage de Django et de Django Rest Framework
20.10.2016	3.5	Apprentissage	Apprentissage de Django et de Django Rest Framework
Total	**7.0**		
26.10.2016	0.0	Vacances	
02.11.2016	1.0	Gestion	Discussion de groupe
	2.5	Authentification	Apprentissage de JWT (Json Web Token)
	2.0	Authentification	Implémentation de l'authentification
06.11.2016	2.5	Authentification	Implémentation et test de l'authentification
	2.0	Authentification	Mise en place du système de réinitialisation du mot de passe
Total	**10.0**		
09.11.2016	1.0	Gestion	Discussion de groupe
	2.5	Apprentissage	Apprentissage Firebase Cloud Messaging (FCM)

10.11.2016	4.0	Push	Mise un place de l'enregistrement des devices client (avec token FCM)
Total	**7.5**		
15.11.2016	1.0	Présentation	Préparation de la présentation intermédiaire
16.11.2016	0.5	Gestion	Discussion de groupe
	3.0	Push	Implémentation et tests de l'envoi de notifications relatives aux demandes d'ami
18.11.2016	2.0	Push	Implémentation et tests de l'envoi de notifications relatives aux demandes d'ami
Total	**6.5**		
23.11.2016	1.0	Présentation	Présentation intermédiaire et discussions en groupe
	1.5	Push	Implémentation et tests de l'envoi de notifications relatives aux demandes d'ami
	4.0	Push	Développement d'un nouveau module de gestion des Device des utilisateurs et des notifications Push
Total	**6.5**		
30.11.2016	1.0	Gestion	Discussion de groupe
	3.5	Push	Mise en place de notifications différées
	2.0	Push	Mise à jour des notifications existantes
Total	**6.5**		
07.12.2016	1.0	Gestion	Discussion de groupe
	4.0	Push	Notifications relatives aux rencontres (nouvelle rencontre)
Total	**5.0**		
14.12.2016	1.0	Gestion	Discussion de groupe
	5.0	API Rest	Ajout du endpoint permettant de gérer sa participation à une rencontre
Total	**6.0**		
21.12.2016	1.0	Gestion	Discussion de groupe
	4.5	Push	Modification du système de notifications (correction de bugs, améliorations)
Total	**5.5**		
04.01.2017	3.0	Push	Modification du système de notifications et ajout de notifications de rencontres (acceptation/refus)
05.01.2017	4.0	API Rest & Push	Ajout de status sur les rencontres et notifications push liées
	3.0	API Rest & Push	Ajout du enpoint permettant de mettre à jour sa position et notifications push liées
06.01.2017	5.0	API Rest & Push	Ajout de la possibilité d'annuler une participation et une rencontre, et notifications push liées
07.01.2017	4.0	API Rest & Push	Tests et diverses modifications
08.01.2017	3.5	Documentation	

Total	**22.5**		
10.01.2016	4	Documentation	
	2	Tests corrections	
Total	**6**		
Total	**108.0**		

Schubert Benjamin

Date	Heures	Tâche	Description
25.09.2016	4	Discussion de groupe	Proposition du projet fini
Total	**4**		
04.10.2016	2	Discussion de groupe	Ébauche du cahier des charges
05.10.2016	4	Discussion de groupe	Rédaction du CC, mise en place des journaux de travail
07.10.2016	1	Discussion de groupe	Cahier des charges fini
Total	**7**		
19.10.2016	3	Tests	Lecture de la documentation de FCM et DDjango Rest Framework
21.10.2016	6	Préparation de l'environement de dévelopement	
Total	**9**		
23.10.2016	4	Installation du Travis pour les tests automatisés	
28.10.2016	5	Préparation des linters et de la buildchain	
Total	**9**		
02.11.2016	3	Discussion du groupe	Modèles, API
05.11.2016	4	Mise en place des modèles et du début de l'API	
Total	**7**		
09.11.2016	3	Discussion de groupe	Vue sur l'avancement, choix de Firebase Cloud Messaging pour faire du Push, mardi 15.11.2016 réunion pour la présentation intermédiaire
11.11.2016	3	Préparation du script de déployement automatique	
Total	**6**		
14.11.2016	3	Avancement de l'API côté serveur	
14.11.2016	2	Configuration du serveur pour gérer le HTTPS, sécurisation du serveur	
18.11.2016	3	Tests sur le serveur	
19.10.2016	1	Préparation de la présentation intermédiaire	
Total	**9**		
23.11.2016	3	Documentation de l'API	
23.11.2016	1	Discussion de groupe	Présentation intermédiaire
Total	**4**		
30.11.2016	3	Avancement de l'API et documentation	

Total	**3**		
07.12.2016	3	Préparation des outils pour la création du site web	
07.12.2016	1	mise à jour du serveur et du script de déployement pour le site web	
Total	**5**		
14.12.2016	4	Création de la page de login, ajout des fonctionnalités manquantes sur le serveur	
Total	**4**		
21.12.2016	4	Création de la page de gestion d'utilisateurs	
Total	**4**		
01.01.2016	4	Création de la page de statistiques, ajout du module statistiques pour le serveur	
02.01.2016	5	Aide pour l'architecture de l'application	
80 06.01.2016	4	Rapport	
06.01.2016	4	Ajout de la fonctionnalité "Hidden" pour l'utilisateur de l'application	
07.01.2016	4	Administratif	ajout de la documentation sur l'architecture du service
08.01.2016	5	Nettoyage de code	
Total	**26**		
10.01.2016	6	Administratif	Finalisation de la documentation
10.01.2016	4	Tests fonctionnels et corrections de bugs	
Total	**10**		
Total	**107**		

7.3 Planification initiale

Tâches

Nom	Ressources	Qte travail
Redaction Cahier Des charges	Benjamin , Patrick	12h
Redaction Diagramme de Gantt	Thibaut	6h
Design Logo	Damien	4h
Flux d'utilisation	Damien	4h
Maquette de l'interface	Benjamin , Patrick, Damien, Thibaut	6h
Conception Structure Application	Benjamin , Patrick, Damien, Thibaut	16h
Prise en main Gyroscope/Géolocalisation	Patrick, Thibaut	7h
Apprentissage HTML CSS	Patrick, Thibaut	7h
Apprentissage Meteor	Patrick, Thibaut	9h
Apprentissage FCM (Firebase Cloud Messaging)	Benjamin , Patrick, Damien, Thibaut	10h
Apprentissage Open Street Map	Benjamin , Patrick, Damien, Thibaut	10h
Apprentissage Django	Benjamin , Damien	10h
Test de faisabilité Django	Damien	8h
Test de faisabilité Meteor	Thibaut	8h
Infrastructure		
Set up d'un serveur d'integration continue	Patrick	8h
Set up Serveur	Benjamin	8h
Sécurité		
Hash des données sensibles	Benjamin	4h
Chiffrage de la communication Client/Serveur	Benjamin	2h
Création de droits	Benjamin	4h
Base de données		
Création du protocole	Benjamin , Patrick, Damien, Thibaut	16h
Structure de la BDD	Benjamin , Damien	6h
Création de la BDD	Damien	2h
Remplissage de la BDD	Damien	2h
Création de l'API	Benjamin , Damien	16h
Implémentation des modèles	Patrick, Thibaut	4h
Implémentation du Design		
Menus et interface	Patrick, Thibaut	16h
Boussole	Patrick	4h
Radar	Thibaut	4h
Map	Patrick, Thibaut	6h
Mode jour/nuit	Benjamin , Patrick, Damien, Thibaut	2h
Liaison Design - Models	Patrick, Thibaut	8h

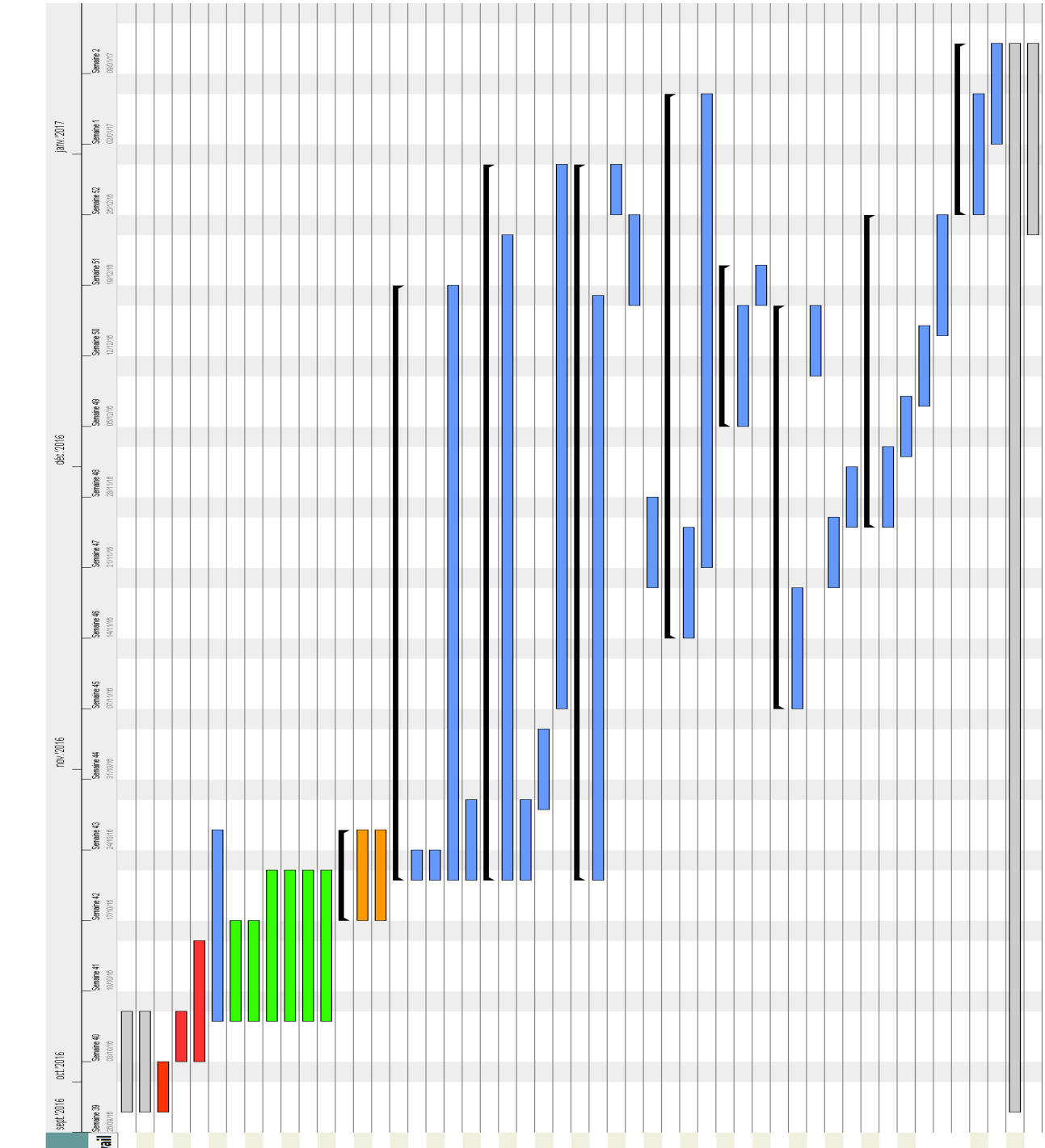
Tâches

Nom	Ressources	Qte travail
Interface admin		
Afiichage des statistiques	Benjamin	8h
Bloquage utilisateur	Benjamin	2h
Création de compte		
Génération d'un id unique	Damien	2h
Gestion de la déconnection	Damien	2h
Suppression du compte	Damien	3h
Changement de numro de telephone	Damien	4h
Gestion de la liste de contact		
Recherche dans la liste de contact	Patrick, Thibaut	8h
Ajout/Suppression/Bloquage	Benjamin , Damien	6h
Historique	Benjamin , Damien	12h
Lien avec la BDD	Benjamin , Damien	10h
Ajout par QR Code	Benjamin , Patrick, Damien, Thibaut	8h
Géolocalisation		
Point fixe	Patrick, Thibaut	12h
Sur utilisateur	Thibaut	8h
Point fixe Optimum	Patrick	8h
Documentation du projet	Benjamin , Patrick, Damien, Thibaut	45h
Redaction du rapport	Benjamin , Patrick, Damien, Thibaut	25h

GANTT project
sept. 2016
oct. 2016
nov. 2016
déc. 2016
janv. 2017
Semaine 39 26/09/16
Semaine 40 03/10/16
Semaine 41 10/10/16
Semaine 42 17/10/16
Semaine 43 24/10/16
Semaine 44 31/10/16
Semaine 45 07/11/16
Semaine 46 14/11/16
Semaine 47 21/11/16
Semaine 48 28/11/16
Semaine 49 05/12/16
Semaine 50 12/12/16
Semaine 51 19/12/16
Semaine 52 26/12/16
Semaine 1 02/01/17
Semaine 2 09/01/17
Nom
Ressources
Qte travail
Redaction Cahier Des charges Beniamin ,Patrick 12h
Redaction Diagramme de Gantt Thibaut 6h
Design Logo Damien 4h
Flux d'utilisation Damien 4h
Maquette de l'interface Beniamin ,Patrick,Damien,Thibaut 6h
Conception Structure Application Beniamin ,Patrick,Damien,Thibaut 16h
Prise en main Gyroscope/Géolocalisation Patrick,Thibaut 7h
Apprentissage HTML CSS Patrick,Thibaut 7h
Apprentissage Meteor Patrick,Thibaut 9h
Apprentissage FCM (Firebase Cloud Mes... Beniamin ,Patrick,Damien,Thibaut 10h
Apprentissage Open Street Map Beniamin ,Patrick,Damien,Thibaut 10h
Apprentissage Django Beniamin ,Damien 10h
Test de faisabilité Django Damien 8h
Test de faisabilité Meteor Thibaut 8h
Infrastructure
Set up d'un serveur d'integration cont... Patrick 8h
Set up Serveur Beniamin 8h
Sécurité
Hash des données sensibles Beniamin 4h
Chiffrage de la communication Client/... Beniamin 2h
Création de droits Beniamin 4h
Base de données
Création du protocole Beniamin ,Patrick,Damien,Thibaut 16h
Structure de la BDD Beniamin ,Damien 6h
Création de la BDD Damien 2h
Remplissage de la BDD Damien 2h
Création de l'API Beniamin ,Damien 16h
Implémentation des modèles Patrick,Thibaut 4h
Implémentation du Design
Menus et interface Patrick,Thibaut 16h
Boussole Patrick 4h
Radar Thibaut 4h
Map Patrick,Thibaut 6h
Mode jour/nuit Beniamin ,Patrick,Damien,Thibaut 2h
Liaison Design - Models Patrick,Thibaut 8h
Interface admin
Affichage des statistiques Beniamin 8h
Blocuage utilisateur Beniamin 2h
Création de compte
Génération d'un id unique Damien 2h
Gestion de la déconnection Damien 2h
Suppression du compte Damien 3h
Changement de numro de telephone Damien 4h
Gestion de la liste de contact
Recherche dans la liste de contact Patrick,Thibaut 8h
Ajout/Suppression/Blocuage Beniamin ,Damien 6h
Historique Beniamin ,Damien 12h
Lien avec la BDD Beniamin ,Damien 10h
Ajout par QR Code Beniamin ,Patrick,Damien,Thibaut 8h
Géolocalisation
Point fixe Patrick,Thibaut 12h
Sur utilisateur Thibaut 8h
Point fixe Optimum Patrick 8h
Documentation du projet Beniamin ,Patrick,Damien,Thibaut 45h
Redaction du rapport Beniamin ,Patrick,Damien,Thibaut 25h

Membres	Semaine 39	Semaine 40	Semaine 41	Semaine 42	Semaine 43	Semaine 44	Semaine 45	Semaine 46	Semaine 47	Semaine 48	Semaine 49	Semaine 50	Semaine 51	Semaine 52	Semaine 1	Semaine 2	Total par personne
Champion	4	7	7	7	5	6	6	7	6	6	6	7	6	6	6	6	98
Loiseau	4	7	7	7	6	6	5	8	6	7	6	5	5	6	7	6	98
Rochat	5	8	7	6	5	6	5	6	6	5	6	6	8	6	6	6	97
Shubert	4	7	7	6	6	8	6	5	8	6	6	5	6	6	7	6	99
																Total	
Total par semaine	17	29	28	26	22	26	22	26	26	24	24	23	25	24	26	24	392

7.4 Planification effective



7.5 Détail API REST

Rady API

This is the API used for Rady application

Version 1.0.0

Security

Bearer (API Key) Authenticate

Name	Authorization
In	header

Paths

`/users/`

`GET /users/`

Description

Returns all users registered in the application.

Parameters

Name	Located in	Description	Required	Schema
email	query	Filter the users by this email.	No	⇄ string
username	query	Filter the users by this username.	No	⇄ string
phone(list)	query	Filter the users by this/these phone number(s).	No	⇄ string

Responses

Code	Description	Schema
200	Return an array of registerd users.	▾ [⇄ ▸ PublicUser { }]
401	Could not authenticate the request.	

Security

Security Schema	Scopes
Bearer	

Try this operation

POST /users/

Description

Create a new user.

Parameters

Name	Located in	Description	Required	Schema
username	query	The username of the user.	Yes	⇄ string
password	query	The password of the user.	Yes	⇄ string
email	query	The email address of the user (must be unique).	Yes	⇄ string

Responses

Code	Description	Schema
201	The user has been created, return user details.	⇄ ▾ [▸ **PublicUser { }**]
400	The request is incorect, see the received error message.	

Try this operation

/users/me/

GET /users/me/

Description

Allows a user to get his own information.

Responses

Code	Description	Schema
200	Return the user information.	⇄ ▾ **UserProfil {** id: ▸ integer username: ▸ string email: ▸ string avatar: ▸ string is_hidden: ▸ boolean }
401	Could not authenticate the request.	

Security

Security Schema	Scopes
Bearer	

Try this operation

PUT /users/me/

Description

Allows a user to update his own information.

Parameters

Name	Located in	Description	Required	Schema
username	query	The new username.	No	⇄ string
email	query	The new email.	No	⇄ string
password	query	The new password.	No	⇄ string
phone_number	query	The new phone number.	No	⇄ string
country	query	The new country for the phone number. Required only if a new phone number is provided.	No	⇄ string
is_hidden	query	True if the user is hidden from the others, false otherwise.	No	⇄ boolean

Responses

Code	Description	Schema
200	Return the user information.	⇄ ▾ **UserProfil** { id: ▸ integer, username: ▸ string, email: ▸ string, avatar: ▸ string, is_hidden: ▸ boolean }
401	Could not authenticate the request.	

Security

Security Schema	Scopes
Bearer	

Try this operation

/users/me/avatar/

PUT /users/me/avatar/

Description

Allows a user to update his avatar.

Parameters

Name	Located in	Description	Required	Schema
avatar	formData	The new avatar image.	Yes	⇄ file

Responses

Code	Description
200	The avatar has been updated.
400	The request is incorect, see the received error message.
401	Could not authenticate the request.

Security

Security Schema	Scopes
Bearer	

Try this operation

DELETE /users/me/avatar/

Description

Allows a user to delete his avatar.

Responses

Code	Description
204	The avatar has been deleted.
400	The request is incorect, see the received error message.
401	Could not authenticate the request.

Security

Security Schema	Scopes
Bearer	

Try this operation

/users/me/friends/

GET /users/me/friends/

Description

Allows a user to retrieve his friends that have accepted the request and that are not blocked by the user.

Responses

Code	Description	Schema
200	Return an array of friends.	▾ [⇄ ▸ Friend { }]
401	Could not authenticate the request.	

Security

Security Schema	Scopes
Bearer	

Try this operation

POST /users/me/friends/

Description

Allows a user to request a new friendship relation.

Parameters

Name	Located in	Description	Required	Schema
friend	query	The id of the friend to request.	Yes	⇄ integer

Responses

Code	Description	Schema
201	The friend request is done, return the friendship details.	▾ [⇄ ▸ Friend { }]
400	An error has occured.	
401	The user isn't authenticated.	
403	You don't have permission to perform that action.	

Security

Security Schema	Scopes
Bearer	

Try this operation

/users/me/friends/all/

GET /users/me/friends/all/

Description

Allows a user to retrieve the list of his friends whether they already accepted the request or not, whether they are blocked or not.

Responses

Code	Description	Schema
200	Return an array of friends with informations if they are blocked, accepted or hidden.	▾ [⇄ ▸ **Friend** { }]
401	Could not authenticate the request.	

Security

Security Schema	Scopes
Bearer	

Try this operation

/users/me/friends/blocked/

GET /users/me/friends/blocked/

Description

Allows a user to retrieve the list of his friends that he blocked.

Responses

Code	Description	Schema
200	Return an array of friends with informations if they are blocked, accepted or hidden.	▾ [⇄ ▸ **Friend** { }]
401	Could not authenticate the request.	

Security

Security Schema	Scopes
Bearer	

Try this operation

/users/me/friends/pending/

GET /users/me/friends/pending/

Description

Allows a user to retrieve the list of his friends that where not yet answered.

Responses

Code	Description	Schema
200	Return an array of friends with informations if they are blocked, accepted or hidden.	▾ [⇄ ▸ **Friend** { }]
401	Could not authenticate the request.	

Security

Security Schema	Scopes
Bearer	

Try this operation

/users/me/friends/hidden/

GET /users/me/friends/hidden/

Description

Allows a user to retrieve the list of his friends that he hid.

Responses

Code	Description	Schema
200	Return an array of friends with informations if they are blocked, accepted or hidden .	▾ [⇄ ▸ **Friend** { }]
401	Could not authenticate the request.	

Security

Security Schema	Scopes
Bearer	

Try this operation

/users/me/friends/{pk}/

GET /users/me/friends/{pk}/

Description

Allows a user to retrieve a friendship relation.

Parameters

Name	Located in	Description	Required	Schema
pk	path	The id of the friend user.	Yes	⇄ integer (int64)

Responses

Code	Description	Schema
200	Return the friendship relation.	⇄ ▾ **Friend {** id: ▸ integer friend: ▸ **UserProfil { }** initiator: ▸ boolean is_hidden: ▸ boolean is_accepted: ▸ boolean is_blocked: ▸ boolean }
401	Could not authenticate the request.	
403	Could not autorize the request.	
404	The place doesn't exists.	

Security

Security Schema	Scopes
Bearer	

Try this operation

`PATCH /users/me/friends/{pk}/`

Description

Allows a user to update one of his friendship relation.

Parameters

Name	Located in	Description	Required	Schema
pk	path	The id of the friend user.	Yes	⇄ integer (int64)
is_blocked	query	True if the friend is blocked, false otherwise.	Yes	⇄ boolean
is_hidden	query	True if the friend is hidden, false otherwise.	Yes	⇄ boolean

Responses

Code	Description	Schema
200	The friendship relation has been modified, return the friendship details.	⇄ ▾ **Friend {** id: ▸ integer friend: ▸ **UserProfil { }** initiator: ▸ boolean is_hidden: ▸ boolean is_accepted: ▸ boolean is_blocked: ▸ boolean }
400	The request is incorect, see the received error message.	
401	Could not authenticate the request.	
403	Could not autorize the request.	
404	The place doesn't exists.	

Security

Security Schema	Scopes
Bearer	

Try this operation

`/meetings/`

GET /meetings/

Description

Allows a user to retrieve all the meetings which he was participant.

See the POST /meetings/ endpoint for more information about the types of meetings.

Responses

Code	Description	Schema
200	Return an array of meetings.	▾ [⇄ ▸ Meeting { }]
401	Could not authenticate the request.	

Security

Security Schema	Scopes
Bearer	

Try this operation

POST /meetings/

Description

Allows a user to create a new meeting.

There are 3 types of meeting :

- type = "shortest":

When the type is "shortest", then the server will compute an approximation of the shortest path between all participants and will use this as a meeting point. If this mode is chosen, then the `place` attribute will be displayed on `GET` requests, but may be null as long as the server didn't finish its computation.

- type = "place":

When the type is "place", then the server expects a `place` attribute on creation, which contains at least a `longitude` and `latitude` attributes, representing the coordinates of the place. It also accepts an optional `name` that has no influence on the place but allows to give a human readable name of it.

- type = "person":

When the type is "person", then the server expects a `on` attribute on creation, which contains the id of the user on which to make the meeting point. The user must be in the meeting.

Parameters

Name	Located in	Description	Required	Schema
type	query	The type of the meeting (place | shortest | person).	No	⇄ string
participants	query	The id of the participants.	No	▾ [⇄ integer]

Responses

Code	Description	Schema
201	The meeting has been created, return the meeting details.	▾ **Meeting {** id: ▸ integer organiser: ▸ **UserProfil { }** place: ▸ **Place { }** participants: ▸ **[]** meeting_time: ▸ string type: ▸ string status: ▸ string on: ▸ **UserProfil { }** }
401	Could not authenticate the request.	

Security

Security Schema	Scopes
Bearer	

Try this operation

/meetings/places/

GET /meetings/places/

Description

Allows the user to retrieve all his places.

Responses

Code	Description	Schema
200	Return an array of places.	▾ [⇄ ▸ **PublicUser { }**]
401	Could not authenticate the request.	

Security

Security Schema	Scopes
Bearer	

Try this operation

/meetings/places/{pk}/

GET /meetings/places/{pk}/

Description

Allows a user to retrieve a specific place.

Parameters

Name	Located in	Description	Required	Schema
pk	path	The id of the place.	Yes	⇄ integer (int64)

Responses

Code	Description	Schema
200	Return the place.	⇄ ▾ **Place {** id: ▸ integer latitude: ▸ number longitude: ▸ number name: ▸ string }
401	Could not authenticate the request.	
403	Could not autorize the request.	
404	The place doesn't exists.	

Security

Security Schema	Scopes
Bearer	

Try this operation

PATCH /meetings/places/{pk}/

Description

Allows a user to update one of his place.

Parameters

Name	Located in	Description	Required	Schema
pk	path	The Id of the place.	Yes	⇄ integer (int64)
name	query	The name of the place.	Yes	⇄ string

Responses

Code	Description	Schema
200	The place has been modified, return the place details.	⇄ ▾ **Place {** id: ▸ integer latitude: ▸ number longitude: ▸ number name: ▸ string }
400	The request is incorect, see the received error message.	
401	Could not authenticate the request.	
403	Could not autorize the request.	
404	The place doesn't exists.	

Security

Security Schema	Scopes
Bearer	

Try this operation

/meetings/{pk}/participants/

PATCH /meetings/{pk}/participants/

Description

Allows a user to update one of his participation.

Parameters

Name	Located in	Description	Required	Schema
pk	path	The Id of the place.	Yes	⇄ integer (int64)
accepted	query	True if the user accepted the meeting, false otherwise.	No	⇄ boolean
arrived	query	True if the user is arrived to the meeting, false otherwise.	No	⇄ boolean

Responses

Code	Description	Schema
200	The participation has been modified, return the participation details.	⇄ ▾ **Participant {** user: ▸ **UserProfil { }** accepted: ▸ boolean arrived: ▸ boolean }
400	The request is incorect, see the received error message.	
401	Could not authenticate the request.	
403	Could not autorize the request.	
404	The participation doesn't exists.	

Security

Security Schema	Scopes
Bearer	

Try this operation

/meetings/position/

POST /meetings/position/

Description

Allows a user to share his position.

Parameters

Name	Located in	Description	Required	Schema
latitude	query	The latitude.	Yes	⇄ number (DECIMAL(6))
longitude	query	The longitude.	Yes	⇄ number (DECIMAL(6))

Responses

Code	Description
201	The place has shared.
400	The request is incorect, see the received error message.
401	Could not authenticate the request.

Security

Security Schema	Scopes
Bearer	

Try this operation

/auth/login/

POST /auth/login/

Description

Allows the user to obtain a valide token, used to authenticate each of his requests.

Parameters

Name	Located in	Description	Required	Schema
username	query	The username of the user.	Yes	⇄ string
password	query	The password of the user.	Yes	⇄ string

Responses

Code	Description	Schema
200	The authentication is correct, return a valid token.	⇄ ▾ **Token** { token: ▸ string * }
400	The request is incorect, see the received error message.	

Try this operation

/auth/refresh/

POST /auth/refresh/

Description

Allows the user to refresh is token to extend his lifetime.

Parameters

Name	Located in	Description	Required	Schema
token	body	The old token	Yes	⇄ ▾ **Token** { token: ▸ string * }

Responses

Code	Description	Schema
200	The transmitted token is correct, return a new valid token.	⇄ ▾ **Token** { token: ▸ string * }
400	The request is incorect, see the received error message.	

Try this operation

/fcm/devices/

POST /fcm/devices/

Description

Allows the user to register a new device, related to his profile.

Parameters

Name	Located in	Description	Required	Schema
registration_id	query	Filter the users by this email.	No	⇄ string

Responses

Code	Description	Schema
201	The device has been registered, return the details of the device.	▾ **Device {** registration_id: ▸ string * }
400	An error has occured.	
401	Could not authenticate the request.	

Security

Security Schema	Scopes
Bearer	

Try this operation

Models

PublicUser

▾ **PublicUser {**
id: ▸ integer
⇄ username: ▸ string
avatar: ▸ string
}

UserProfil

▾ **UserProfil {**
id: ▸ integer
username: ▸ string
⇄ email: ▸ string
avatar: ▸ string
is_hidden: ▸ boolean
}

Friend

```
▾ Friend {
    id:           ▸ integer
    friend:       ▸UserProfil { }
    initiator:    ▸ boolean
    is_hidden:    ▸ boolean
    is_accepted:  ▸ boolean
    is_blocked:   ▸ boolean
  }
```

Meeting

```
▾ Meeting {
    id:            ▸ integer
    organiser:     ▸UserProfil { }
    place:         ▸Place { }
    participants:  ▸[]
    meeting_time:  ▸ string
    type:          ▸ string
    status:        ▸ string
    on:            ▸UserProfil { }
  }
```

Place

```
▾ Place {
    id:         ▸ integer
    latitude:   ▸ number
    longitude:  ▸ number
    name:       ▸ string
  }
```

Participant

```
▾ Participant {
    user:      ▸UserProfil { }
    accepted:  ▸ boolean
    arrived:   ▸ boolean
  }
```

Token

```
▾ Token {
    token:  ▸ string *
  }
```

Device

```
▾ Device {
    registration_id:  ▸ string *
  }
```