Architecture Logique

Table des matières

1 Modules

On définit ici les modules du projet.

1.1 Planning

Le module planning contient la réalisation du planning côté **`core`** et la réalisation de l'interface graphique du gestionnaire d'évènements.

1.1.1 Core

On a besoin ici du package **`fr.upem.geoplan.core.planning`** qui contiendra la déclaration des classes représentants le planning ainsi que de ses interfaces de gestion.

La classe **`Planning`** définie les composants suivants :

- Une liste des évènements créés
- Des méthodes pour passer d'un évènement à l'autre (suivant/précédent)
- Des méthodes de recherche d'évènements
- Des méthodes d'ajout/suppression/édition d'évènements
- Des méthodes de filtre sur les évènements (prochains, en cours, manqués, priorité, participants, etc…)

La classe **`Event`** définie les éléments suivants :

- Une liste de propriétés exploitables :
    - Nom
    - Date et durée (date et heure début, date et heure fin)
    - Localisation (adresse, lieu-dit, coordonnées GPS, etc…)
    - Participants (ID user)
    - Organisateurs (voir Participants)
    - Description (détails)
    - Importance (probablement une échelle numérique 1-4, à déterminer)
    - Type (évènement, Rendez-vous, etc…)
    - Rappels (Temps avant début, répétitions, type (notifications, e-mail, etc…))
    - Coût
- Des méthodes d'accès conditionnées par les droits de l'utilisateur (i.e. organisateurs, participants)

Une classe **`Loader`** permettant de synchroniser les évènements avec ceux des gestionnaires déjà existants (i.e. Facebook, Google Calendar, etc…) Cette fonctionnalité est optionnelle pour le moment.

1.1.2 GUI

Au niveau de l'interface on peut utiliser une activité, qui représente la liste des évènements. Les interfaces de gestion du planning utiliseront des fragments. Ces fragments serviront à :

- Créer un évènement
- Supprimer un évènement (peut-être pas nécessairement un fragment)
- Editer un évènement

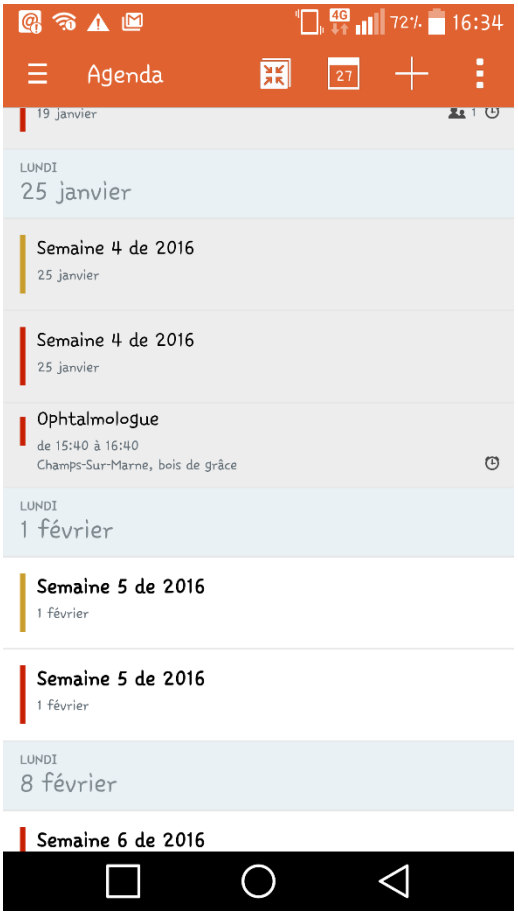
- Filtrer les évènements

Après chaque fonctionnalité, on revient sur l'activité principale (event list).

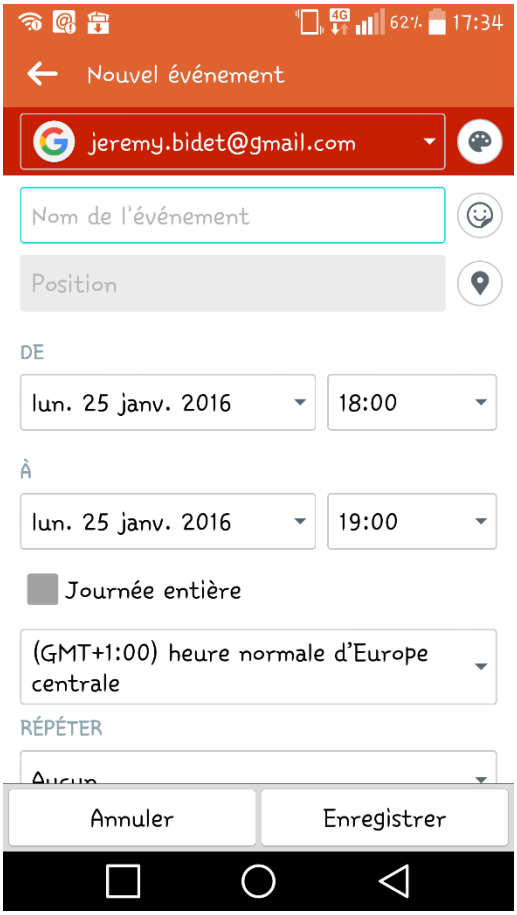
L'activité principale intègre, en plus de la liste, des actions permettant :

- D'accéder aux fonctionnalités (charger les fragments) :
  - Créer
  - Supprimer
  - Editer
  - Filtrer
- D'afficher les évènements manqués ou à suivre (sûrement un scroll top-bottom)
- D'afficher une barre de recherche (autre fragment, tout petit)

Voici une idée de ce à quoi pourrait ressembler l'activité principale :



Et voici une idée de ce à quoi pourrait ressembler les fragments :



1.2 Server

1.2.1 Hébergement

Jeremy, dispose d'un serveur dédié up 24/7 (modulo les crises d'ados de Numéricable) et est d'accord pour héberger cette partie et gérer les communications.

Si l'équipe possède d'autres solutions, il est d'accord pour les entendre ☺.

1.2.2 Database

- Registered user table
    - Id (primary key – autoincrement – unique)
    - User name (e-mail)
    - Password (hashcode : sha-1 ou autre)
    - Display name (« pseudo » | « prenom nom » | etc…)
    - Mobile phone (optionnellement optionnel)
- Planning table
    - Id (primary key – autoincrement – unique)
    - Name
    - Localization
    - Owners (foreign keys)

    - Guests (foreign keys)
    - Start date
    - Start time
    - End date
    - End time
    - Description
    - Level (importance niveau)
    - Type (évènement, rendez-vous, …)
    - Cost (euros)

Le champs Rappel du planning n'apparaît pas, car il est défini par chaque utilisateur lui-même.

1.2.3 Communication

1.2.3.1 Requêtes

Définitions des requêtes et méthodes d'interrogation avec la base de données.

- Création d'un utilisateur (compte)
- Modification des informations
    - Mobile phone
    - Display name
    - Pasword
- Création d'un event
- Modification d'un event
    - Owners
    - Guests
    - Date – Time
    - Description
    - Localization
    - Name
    - Level
    - Type
    - Cost
- Suppression d'un event (seulement les Owners)

1.2.3.2 Protocol

Les protocoles de communication (hors requêtes) entre le serveur et les terminaux mobiles seront codés en Java.

Définir si on utilise UDP ou TCP protocole de transfert.

- Pour le système de message push/pull (émission/réception), nous utiliseront des paquets légers :
    - ID sender (4 bytes) (static)
    - ID receiver (4 bytes) (static)
    - Message (152 bytes) (max)

Cette longueur est dû à la taille originelle des SMS (160 bytes) (voir aussi Twitter). Cela permet d'accélérer la transmission même sur des réseaux mobiles faibles (Edge, GPRS, etc…).
Le format est libre mais doit être défini une fois pour toute : BIG_ENDIAN, UDP, etc…

- Pour le système de push/pull position, nous utiliserons encore des paquets légers :
  - ID sender (4 bytes) (static)
  - ID event (4 bytes) (static)
  - GPS position (sûrement 2 float) (Ah pardon : surement, y'a plus de circonflexe)

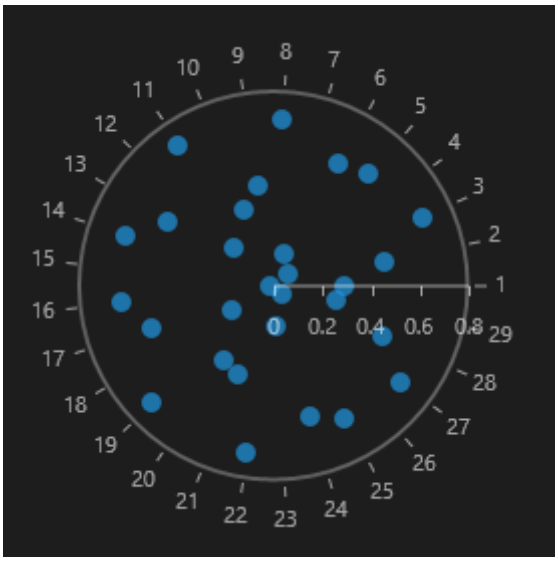
1.3 Radar / Maps

Ce module définit la méthode graphique employée pour afficher les informations de l'application.

Si le temps le permet, nous développerons les deux interfaces et intégreront une option pour switcher de l'une à l'autre, sinon priorité est donnée à… pas encore définie : le radar est plus sympa à développer, mais l'API Google exciterait peut-être un peu plus le prof.

1.3.1 Radar

On définit l'interface graphique du radar, voici un aperçu échantillon :



Il affiche, en son centre, un point qui représente la position de l'utilisateur (celui qui tient le téléphone), on l'appellera le récepteur, et autour du centre, des points représentants les autres utilisateurs, qu'on appellera les émetteurs. La distance et la position du point par rapport au centre indiquent la distance réelle et la direction de l'utilisateur pointé par ce marqueur.

Il serait intéressant d'intégrer dans ce radar, pour

D'une part représenter la distance :

- Une échelle de distance comme indiquée sur l'image

ou

- Fixer l'échelle et n'afficher que les utilisateurs à portée

D’autre part représenter la direction :

- Les points cardinaux : Nord, Est, Ouest, Sud (pas comme sur l’image)

ou

- Une flèche indiquant dans quelle direction regarde le récepteur

ou

- Fixer l’orientation du récepteur sur le haut du radar et ainsi orienter les autres points en fonction

Encore d’autre part, informatiser les émetteurs :

- Rendre cliquable les marqueurs afin d’obtenir des informations (distance précise, nom, message court, etc…)

ou

- Afficher autour les informations essentielles (nom) plus sous forme de pop-up des messages courts

Pour déterminer ces données, les émetteurs envoient au récepteur leur coordonnées GPS (latitude/longitude). Le récepteur calcul grâce à une (ou plusieurs) formule(s) les données (distance & direction). Cette (ces) formule(s) est (sont) à déterminer par celui qui s’occupera de ce module, un peu de recherche Google devrait suffire.

1.3.2 Maps

On utilisera ici l’API Google Maps. Sur le même principe que le radar, cette fonctionnalité va afficher les positions des émetteurs sur une carte, avec au centre de la carte, à l’initialisation, le récepteur.

Le récepteur à la possibilité, intrinsèque à l’API, de déplacer, zoomer, dézoomer cette carte.

Les détails d’implémentation de l’API sont dans les docs de Google, encore un peu de recherche ici.