Munuera Romain
Bancarel Valentin
Janowiez Thomas

Local Sport Meetings

Tuteur de projet : Guillaume Imbert

IUT de Clermont-Ferrand Département informatique Année 2014-2015

UdA | Université d'Auverane

Nous autorisons les responsables de l'IUT à publier notre rapport sur l'intranet de l'établissement

Remerciements

Nous remercions M. Imbert pour nous avoir encadrés et répondus à nos interrogations tout au long du projet. Nous tenons également à remercier M. Delon pour le temps qui nous a consacré à l'installation des différents outils nécessaires au développement du projet. Finalement, nous remercions l'IUT des Cézeaux, pour le prêt du matériel et l'ensemble des enseignants, grâce auxquels, nous avons acquis les connaissances essentiels.

Table des matières

1) Introduction

Dans le cadre de notre licence professionnelle Génie logiciel, développement d'applications pour plateformes mobiles, nous avons eu la chance de réaliser un projet avec pour tuteur M. Guillaume IMBERT.

Que ce soit des sites responsifs ou des applications, de nombreux dispositifs permettent aujourd'hui de consulter les rencontres sportives dans diverses disciplines (football, rugby, …) depuis un smartphone ou une tablette. Or, la plupart du temps, celles-ci ne concernent que les occasions de « grande envergure » à défaut des rencontres régionales ou départementales.

L'objectif du projet a donc été de développer une application mobile qui tourne sous le système d'exploitation Android à la demande d'un client. Elle devait permettre à l'utilisateur de visualiser, sur son téléphone, la rencontre sportive du moment autour de Clermont-Ferrand. Ainsi, sous forme d'onglet, l'utilisateur peut cibler les rencontres qu'il souhaite consulter pour un sport, une catégorie ou une compétition. Toutes les informations sont stockées dans le datastore(5) de Google, un système de base de données (basé sur Google Big Table).
A cela s'ajoutait également la réalisation d'un site Web pour offrir une interface à l'administrateur lui facilitant l'organisation et la manipulation des données.

Nous allons, à travers ce rapport, présenter les principales étapes de la réalisation de ce projet, depuis l'analyse du sujet jusqu'à la mise en place de l'application et du site web. Nous verrons également l'organisation que nous avons mise en œuvre pour optimiser l'achèvement du projet.

2) Présentation du projet

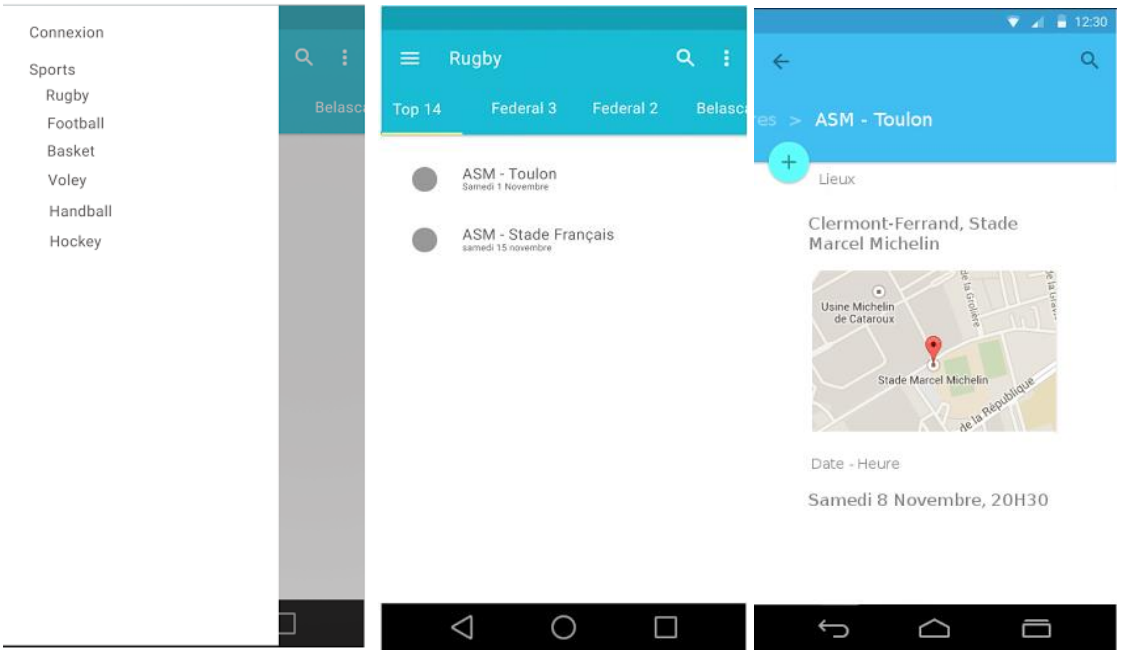
2.1) Cahier des charges

M. Imbert étant la seul personne en contact direct avec le client, c'est tout naturellement lui qui nous a communiqué les détails du sujet ainsi que les attentes du client.
L'idée principale du projet est la création d'une application qui va permettre aux utilisateurs de visualiser les rencontres sportives du moment autour de Clermont-Ferrand.

Celle-ci doit inclure trois niveaux de navigation sous forme d'onglets qui permettront ainsi à l'utilisateur d'ajuster ses recherches suivant différents critères. En effet, au démarrage du logiciel, ce dernier doit avoir la possibilité de spécifier les rencontres d'une discipline (football, rugby, handball …).Une fois celle-ci sélectionnée, l'utilisateur peut consulter les rencontres de ce sport et peut une nouvelle fois filtrer sa recherche par catégorie (senior, benjamin, U17 …).

Les attentes du client sont donc « simples », celui-ci nous donnant simplement une idée générale de ce qu'il souhaitait. Nous avions donc une grande liberté sur l'ajout de nouvelles spécifications à l'application.

A la suite de ces informations nous avons élaboré des maquettes visuelles de l'application afin de donner une idée générale de celle-ci au client et qu'il puisse de ce fait la valider.



Nous avons donc rajouté de nombreuses possibilités comme le filtrage des rencontres par compétition (maquette centrale), un écran affichant le détail d'une rencontre (maquette de droite). Nous avons pensé qu'il serait intéressant pour l'utilisateur de pouvoir partager une rencontre sur les réseaux sociaux, ajouter l'évènement à son calendrier, réaliser un itinéraire de sa position actuel jusqu'au lieu du rendez-vous, ajouter un sport à ses favoris pour optimiser la recherche d'une rencontre.

A la demande de notre tuteur, l'application devait être développée pour les téléphones munis d'un système d'exploitation Android.

Pour gérer les données que l'application mobile exploite, il était évidemment indispensable de créer une interface administrative pour la gestion des rencontres. Au départ, nous pensions développer sur le logiciel mobile.

Ainsi l'administrateur pourrait se connecter à partir de l'application (bouton « Connexion » visible sur la maquette de gauche). Mais cela a le désavantage de surcharger l'interface visuelle du mobile et serait totalement inutile pour les utilisateurs. Par conséquent, nous avons donc opté pour un site Web où l'administrateur aurait la possibilité d'ajouter, modifier ou supprimer des rencontres.

2.2) Organisation du travail

A la demande de notre tuteur du projet, M. Imbert, nous avons réalisé le projet en suivant le framework d'organisation Scrum. Cette méthode agile s'appuie sur le découpage d'un projet en « boîtes de temps » nommés « sprints ».

Pour faciliter la communication, nous avons utilisé le service de stockage Google Drive. Nous pouvions ainsi partager des fichiers entre les membres du groupe notamment les documents liés à l'analyse. C'est également sur cette même plateforme que nous déposions l'APK(31) de l'application à la fin de chaque sprint. M. Imbert pouvait ainsi présenter une version au client et lui exposer l'avancement du projet.

Nous avons également utilisé l'outil de gestion de développement Forge Clermont Université ce qui nous a permis de travailler ensemble sur un même projet.

Un sprint avait une durée d'une semaine. De ce fait, à la fin de chacun d'eux, une réunion était organisée pour présenter l'avancement du projet, les difficultés que nous avions rencontrées et la planification des tâches pour le prochain sprint.
De plus, tous les deux jours, nous envoyions un compte rendu à notre tuteur lui définissant les tâches qui avaient été réalisées.
Voici le diagramme de Gantt réalisé :

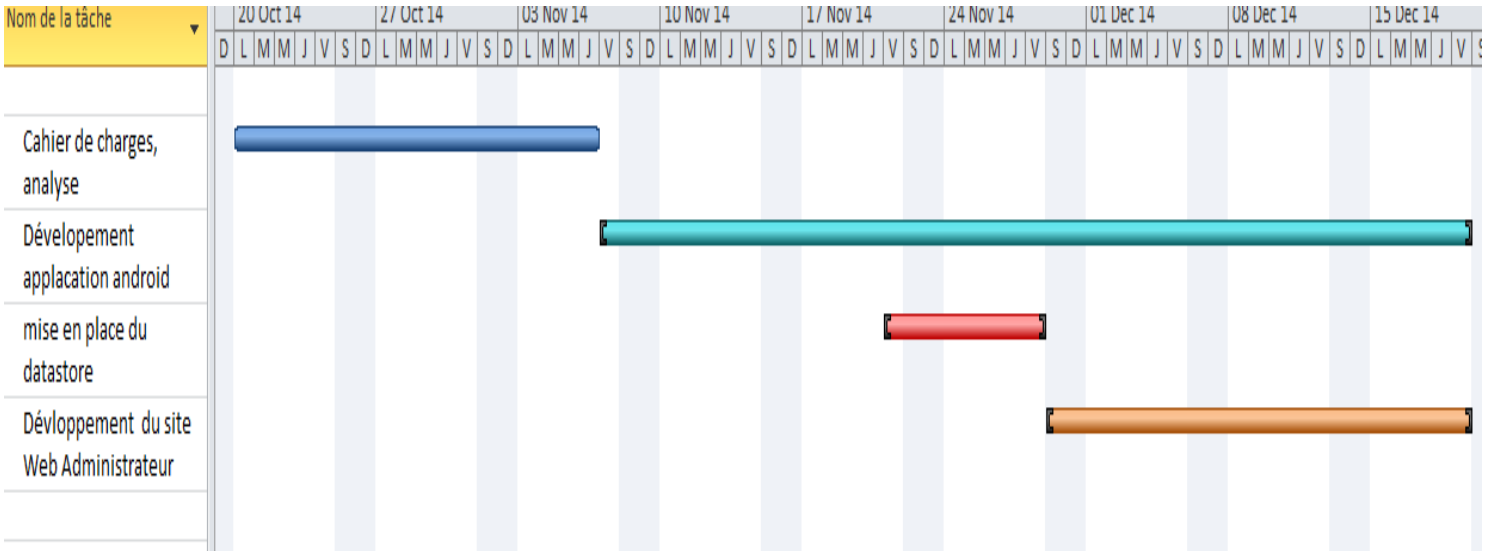


Figure 1 - Diagramme de Gantt réel

Le diagramme de Gantt donne l'impression qu'il n'y a pas beaucoup de tâches réalisées mais nous les avons regroupées pour des raisons de lisibilité. En effet, dans le développement de l'application Android, nous avons réalisé nombreuses tâches que nous nous sommes répartie. Par exemple, l'un de nous s'occupait du menu coulissant pendant qu'un autre mettait en place le système d'onglets.

De plus les tâches « développement application android », « mise en place du datastore » et « développement du site web administrateur » ont été réalisées en même temps. Nous avons mis en place le datastore ensemble en l'adaptant à l'application. Puis, une personne est restée sur le développement de l'application tandis que les deux autres se sont concentrées sur le site web.

3) Réalisation du projet

3.1) Analyse

Pour la réalisation du projet nous avons tout d'abord commencé par faire une phase d'analyse. En se basant sur le cahier des charges fournies, nous avons commencé par lister les différentes fonctionnalités qui doivent être présente dans l'application. Pour cela nous avons réalisé un premier diagramme d'utilisation afin de dégager toutes les actions utilisateur.

A partir de ce diagramme nous avons pu dégager les actions suivantes :

Action de filtrage :

- Consulter la listes de rencontres les plus proches
- Consulter la liste des rencontres en filtrant par sport
- Consulter la liste des rencontres en filtrant par catégories
- Consulter la liste des rencontres en filtrant par compétitions

Autres action :

- Consulter le détail d'une rencontre
- Consulter ses favoris
- Ajouter une rencontre au calendrier

Par la suite grâce à ce diagramme nous avons pu identifier les classes métier de notre application qui se compose donc de sport, de catégorie, de compétition, de rencontre et de participant à une rencontre. Nous avons donc réfléchi à la manière la plus simple pour hiérarchiser toutes ces classes. Nous sommes donc arrivés à la hiérarchie suivante :

Un sport est composé de catégories, les catégories (ex : senior) sont composé de compétitions, les compétitions (ex : TOP 14) sont composés de rencontres et une rencontre est associée à deux participants. On a donc le diagramme de classe suivant :

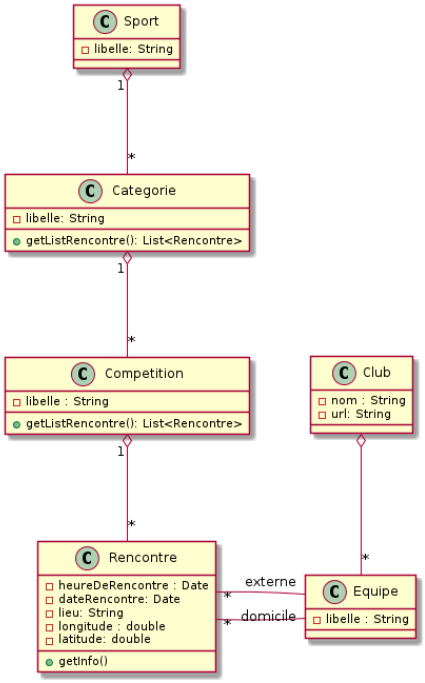


Figure 2 - Diagramme de classe

Base de données :

Pour stocker toutes ces données, nous devions utiliser un serveur(30) de base de données. Suite à le demande de notre tuteur M. Imbert nous avons choisi d'utiliser le Data Store de Google qui fonctionne sous le système d'un web services et fournit une API de bas niveau permettant d'accéder aux données à partir de l'application. En plus de cette API nous avons utilisé une API de plus haut niveau appelée Objectify(1) ce qui nous a simplifier l'accès aux données.

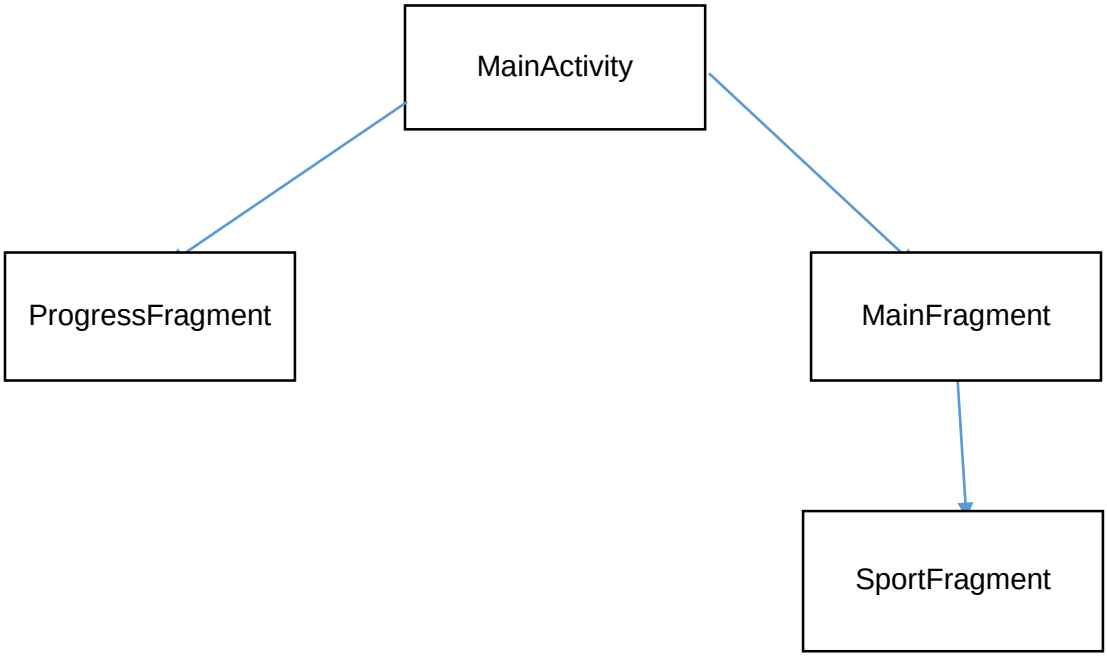
Notre application se décrit donc en deux modules : un module appelé app et un module appelé backend :

Module app : Ce module contient donc toutes les classes concernant l'application (Activity, Fragment, …).

Module backend : Ce module est un module générée automatiquement et contient donc toute la partie métier de l'application et accès au données sur le data store. A chaque classe métier est associée une classe avec des méthodes permettant d'accéder aux données liées à sa classe métier.

Hiérarchie de l'application :

Avant de nous lancer dans le code nous avons aussi défini la hiérarchie générale de notre application, c'est-à-dire la manière dont nous allons disposer l'affichage des données et l'enchainement des différentes pages. Afin de réduire au maximum le niveau de profondeur de l'application nous avons choisi d'implémenter le pattern(3) navigation drawer défini sur le site d'android developper. Grâce à ce pattern nous avons réussi à limiter à deux le niveau de profondeur de l'application. La hiérarchie de notre application peut se résumer par le schéma suivant :



Nous avons donc une activité principale MainActivity, dans laquelle nous plaçons un fragment, dans un premier temps le fragment de chargement puis dans un deuxième temps le MainFragement qui contient le drawer menu. Dans ce MainFragment on place un autre fragment qui représente le contenu affiché dans le drawer layout. Ce contenu est donc un sport fragment qui représente la page de consultation des rencontres d'un sport.

3.2) Application Android

3.2.1) Présentation

L'application que nous devions réaliser est donc une application permettant à un utilisateur de consulter les rencontres sportives autour de Clermont-Ferrand. Cette application est destinée à fonctionner sur des appareilles Android et elle a donc été réalisée en JAVA(2).

Pour faire cette application nous avons utilisé l'IDE Android Studio développé par Google et qui sert principalement à développer des applications Android.

Cette application fonctionne avec des appareilles disposant de l'API 16 au minimum et est prévue pour fonctionner sur l'API 21 avec Android 5. Pour réaliser l'application nous nous sommes basés sur les nouvelles spécifications de Google apparues avec Android 5 notamment tous ce qui concerne le Material Design. Ceci nous a permis d'utiliser des nouveaux composants graphiques qui à notre sens améliore l'expérience utilisateur de l'application.

3.2.2) Travail

Agencement de l'application :

Dans notre application l'utilisateur doit être capable de consulter les rencontres sportives selon plusieurs critères de filtrage comme le sport, les catégories et les compétitions. Pour cela il a fallu réfléchir dans un premier temps à la disposition des différentes pages de l'application. Nous avons essayé de réduire au maximum le niveau de profondeur de l'application pour ne pas détériorer l'expérience utilisateur afin que l'application soit la plus agréable à utiliser.

Pour permettre tous ces filtrages nous avons choisi de mettre en place le pattern(3) « Navigation drawer » défini sur dans la documentation d'Android. Ce pattern(3) nous a permis de

réaliser le filtrage par sport grâce au menu coulissant. Enfin pour chaque sport nous avons utilisé un système d'onglet de checkbox afin de filtrer les rencontres par catégories et compétitions.

Nous sommes donc parties des maquettes suivantes :

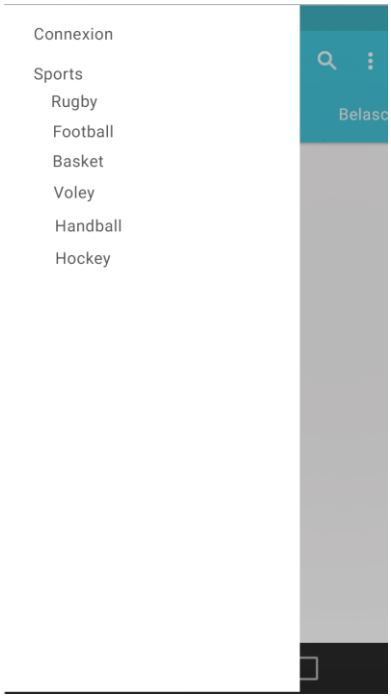
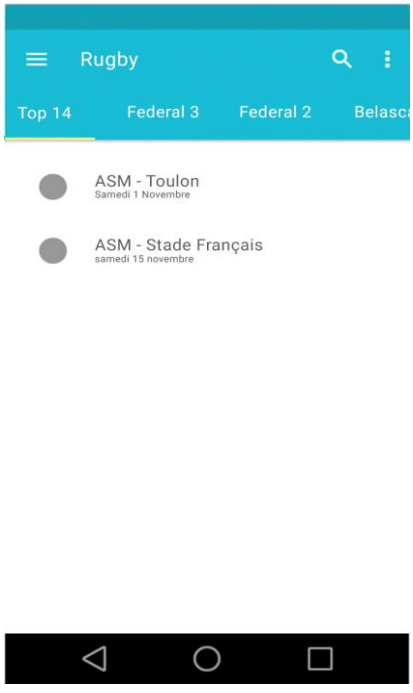


Figure 3 - Maquette Application Android

Cette maquette correspond donc au navigation drawer qui permet de filtrer les rencontres par sport. Ce menu est donc accessible directement, l'utilisateur peut donc changer de sport sans avoir besoin de faire des retours arrière sur les pages.



Enfin, cette maquette représente la vue d'un sport où on peut voir donc les onglets qui représentent les compétitions et les catégories sont accessibles depuis la toolbar(4) avec un bouton qui affiche une liste de check-box.

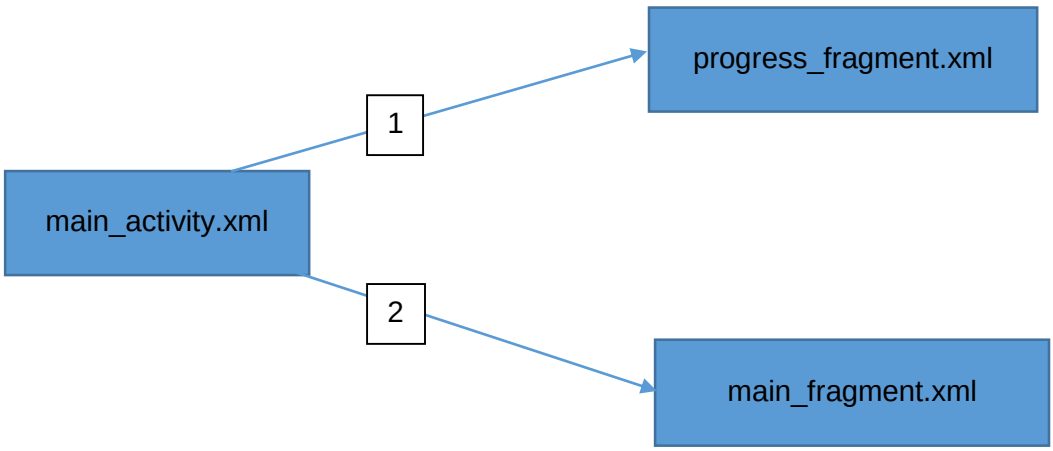
Travail réalisé :

- Page de chargement et d'accueil :

Au démarrage l'application a tout d'abord besoin d'aller chercher les données sur le datastore(5), nous avons donc au démarrage une page de chargement. Ceci nécessite bien sûr que le téléphone soit connecté à internet. Pendant ce chargement nous récupérons grâce à un système de thread(6) tous les sports ainsi que les 10 rencontres les plus proches dans le temps tout sport confondus. Le chargement des rencontres pour chaque sport s'exécute seulement lors du clic sur un sport. De cette manière on ne charge les données que lorsque l'on en a besoin. Lorsque ce premier chargement est terminé l'utilisateur peut donc consulter les 10 rencontres les plus proches ainsi que filtrer les rencontres par sport depuis le menu coulissant.

D'un point de vue plus technique nous avons donc une activité contenant un fragment. Lors du chargement on place dans cette activité un fragment affichant une barre de chargement. Puis lorsque le chargement est terminé on remplace le fragment par celui qui représente la page d'accueil.

Pour réaliser cela nous avons utilisé les layouts(7) suivant :



1- Lancement de chargement des données, on place dans main_activty.xml le fragment affichant le page de chargement

2- Fin du chargement on remplace le fragment de chargement par celui qui représente le navigation drawer.

On a donc les pages suivantes :

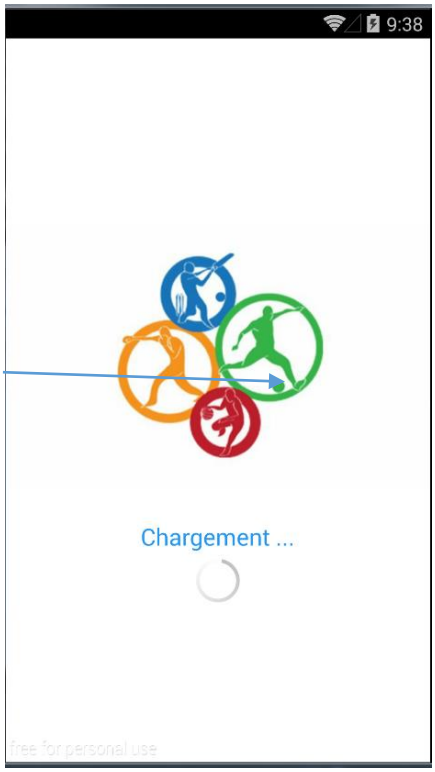


Figure 6 - Capture chargement des données

Fin du chargement

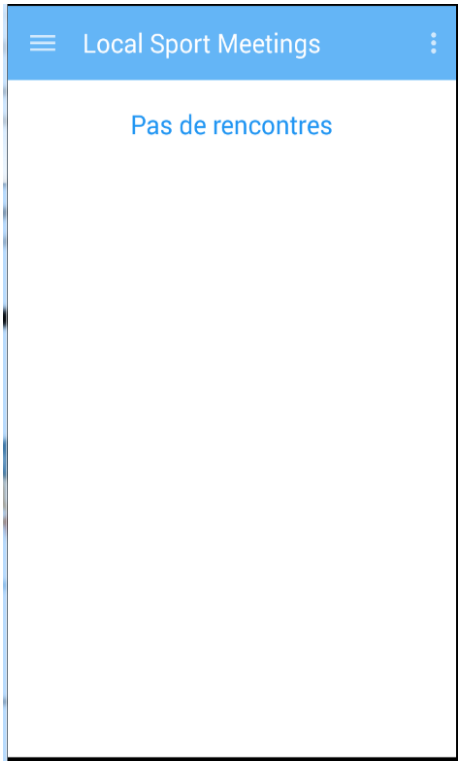


Figure 5 - Capture d'écran de l'application

- Page d'un sport :

A partir de l'accueil, l'utilisateur peut donc utiliser le menu coulissant pour sélectionner un sport. Lors du clic sur un sport une autre page de chargement s'affiche afin de charger les données relatives aux sports puisque l'on a décidé que l'on charger seulement les données dont on a besoin afin de limiter les appels réseaux et la mémoire utilisée par l'application. En revanche lorsque l'utilisateur change de sport, on garde en mémoire les données déjà chargées. De cette manière si l'utilisateur veut revenir sur un sport qu'il a déjà consulté il n'aura plus besoin de charger les données depuis la base.

Sur la page d'un sport, l'utilisateur utilise donc des onglets qui lui permettent de filtrer les rencontres par compétition. Exemple pour le rugby il pourra consulter seulement les rencontres du TOP 14. Pour changer d'onglets il peut sois cliquer directement sur l'onglet qu'il veut consulter ou bien alors il peut aussi faire glisser les onglets afin de consulter l'onglet suivant ou précèdent. Pour réaliser cela nous avons utilisé une classe défini dans la documentation d'Android appelé SlidingTabLayout. Ce layout(7) dérive de la classe HorizontalScrollView et représente donc la liste des compétitions sous forme d'onglet. Ensuite pour chaque onglet est associé une ViewPager qui contient donc les rencontres de chaque compétition. Le layout(7) du fragment représentant la page d'un sport contient donc deux éléments, le SlidingTabLayout et une ViewPager.

Dans les fonctionnalités demandées l'utilisateur doit aussi pouvoir filtrer par catégories. Si l'on reprend le diagramme de classe, un sport est composé de catégories, une catégorie est composée de compétition et une compétition est composée de rencontre. Pour gérer ce filtrage nous nous sommes là aussi inspirés d'une classe trouvée sur internet, à savoir MultipleSelectionSpinner. Cette classe hérite de la classe Spinner et permet de sélectionner plusieurs éléments sous la forme de checkbox. Pour afficher cette liste nous avons placé cet élément dans la toolbar(4) de l'application qui par défaut à une visibilité égale à « gone (8) ». On affiche cet élément seulement lorsque l'utilisateur se trouve sur la page d'un sport. Par défaut toutes les catégories sont sélectionnées, de cette manière lorsque l'utilisateur clique sur un sport il peut consulter les rencontres de toutes les compétitions d'un sport. Lorsque l'utilisateur sélectionne ou désélectionne une catégorie, le SlidingTabLayout affichant les onglets se met alors automatiquement à jour en enlevant ou en rajoutant des onglets. De cette manière l'utilisateur peut donc combiner le filtrage par catégories et compétition.

Nous sommes donc arrivés au résultat suivant :

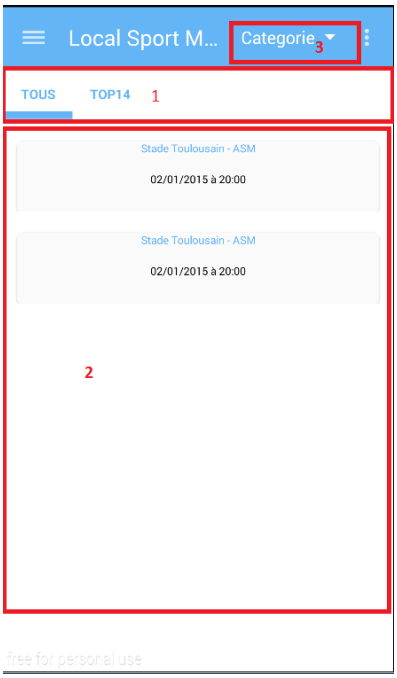


Figure 7 - Capture d'écran de la page d'un sport

1-SlidingTabLayout représentant les compétitions sous forme d'onglet.

2-ViewPager représentant les rencontres d'une compétition.

3-Toolbar(4) contenant le spinner permettant de sélectionner les catégories de la manière suivante :

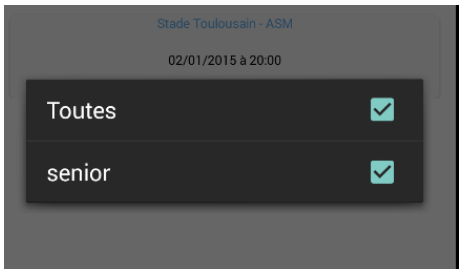


Figure 8 - Capture d'écran du spinner des catégories

- Page d'une rencontre :

Enfin la dernière page de notre application est celle permettant à l'utilisateur de consulter le détail d'une rencontre. Depuis cette page il peut consulter l'heure et le lieu de la rencontre ainsi que les participants. Pour représenter le lieu nous avons inclus une carte Google Maps(24) qui est centrée avec un repère placé sur le lieu de la rencontre. Cette page est représentée par une autre activité. Les pages précédentes représentent des fragments placés dans la même activité mais pour cette vue nous avons choisi de faire une nouvelle activité. C'est par conséquent le seul endroit où le menu coulissant n'est pas accessible et où l'utilisateur est contraint d'effectuer un retour en arrière.

Depuis cette page l'utilisateur peut aussi grâce à deux FloatingActionButton placé dans un FloatingActionMenu créer un événement dans son calendrier avec la date et l'heure de la rencontre, ou bien démarrer un itinéraire depuis sa position vers le lieu de la rencontre avec l'application Google Maps.

Enfin nous avons aussi utilisé la classe ShareActionProvider pour permettre à l'utilisateur de partager une rencontre via différents moyens de communication (messages, réseaux sociaux, …). Pour partager la rencontre nous avons créé une page web décrivant une rencontre et dans le contenu du partage nous plaçons seulement l'URL(9) du détail de la rencontre. Nous avons été contraint de faire ainsi car Facebook permet seulement le partage d'un lien avec la classe ShareActionProvider, si l'on veut partager seulement du texte avec Facebook nous sommes obligés d'utiliser leur propre api.

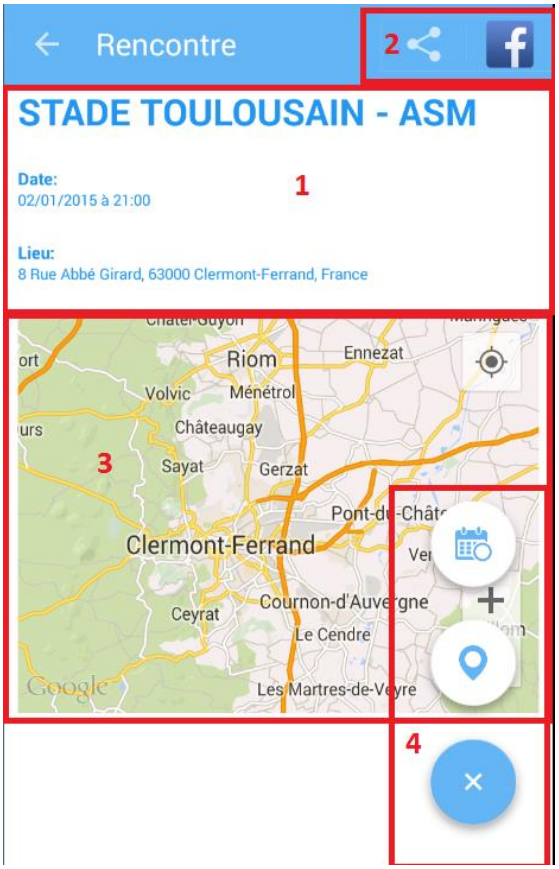


Figure 9 - Capture d'écran de la page d'une rencontre

On retrouve donc sur cette page les éléments suivants :

1- Détail de la rencontre. On retrouve les participants, la date et le lieu.

2- La partie gérée par la classe SharActionProvider permettant le partage.

3- La carte google maps permettant à l'utilisateur de situer le lieu de la rencontre sur une carte.

4- Les deux FloatingActionButton permettant de créer un événement au calendrier de l'appareil Android ou bien de démarrer un itinéraire vers le lieu de la rencontre avec l'application google maps.

3.3) **Interface web**

La deuxième partie du projet fut de mettre en place une solution afin que le client puisse ajouter, modifier ou supprimer des données de l'application, lors de l'analyse, nous avons envisagé deux options, la première était de lui permettre de se connecter directement depuis l'application mobile, cela aurait impliqué un formulaire de connexion visible par tous les utilisateurs de l'application bien que seul l'administrateur en aurait l'utilité, de plus l'insertion de données n'aurait pas été très "user friendly" étant donné que certaine données telle qu'une rencontre nécessite un formulaire de dix champs à remplir.

Nous avons donc choisi de développer une interface web type CRUD(11) afin de fournir au client une solution plus simple, plus pratique et qui ne serait pas visible par les utilisateurs non concernés. Cette solution étant des plus adaptée étant donc que nous avions la possibilité de réutilisé tous les classes métiers et du backend(12) développer pour l'appli mobile dans une application web utilisant les JAVA servlet(14) intégré dans le projet Android.

Ne disposant pas de la durée nécessaire pour développer un site web complet, nous avons opté pour l'utilisation d'une template graphique afin de ne pas devoir coder toutes la partie design et de nous focaliser sur le développement à proprement parler. Le backend étant également déjà codé nous y avons ajouté une classe métier utilisateur ainsi que son endpoint(15) associer afin de pouvoir gérer la connexion, nous avons ainsi pu nous focaliser sur la présentation des données a l'utilisateur.

Afin de limiter la taille des champs select(16) dans les formulaires, ceux-ci sont tous dynamique, c'est à dire qu'avant de sélectionner une entité dans un select, si celle-ci dépend d'une entité parente, le select de l'entité parente doit être renseigné, e.g, lors de l'ajout d'une rencontre, il est impossible de sélectionner une compétition si aucune catégorie n'est sélectionnée.

Figure 10 - Exemple de

Pour éviter la duplication de code, le header(17), le footer(18), le contenu redondant de la balise “head”(19), ainsi que toutes les inclusion de javascript se font a partir d’include .jsp(9). Le menu étant diffèrent en fonction de la page, (l’onglet actuel est mis en surbrillance), toutes URL doit contenir le nom de l’entité qu’il concerne être afin que le fichier d’import du header puisse mettre l’onglet correspondant en surbrillance.

Pour gérer les messages de retour à l’utilisateur, nous avons développé une méthode permettant de retourner différents types de message (succès, avertissement, erreur) a l’utilisateur. Cette méthode prend 2 paramètres, la page sur laquelle l’utilisateur doit être redirigé, le message à lui afficher et une constante déclarée dans le Contrôleur(INDICE 21) principal qui représente le type de message.

Figure 11 - Messages retournés à l'utilisateur

Via un lien présent dans le footer, l’administrateur peut également modifier son mot de passe en remplissant un formulaire.

Chaque action, que ce soit l’ajout, le listage, la suppression ou la modification se fait de la même façon pour chaque entité:

Ajout:

Le formulaire d’ajout est accessible via le menu du header en survolant l’entité désirée, il a un champ pour chaque attribut de l’entité concerne, hormis l’id, qui lui est géré par Objectify. Nous avons utilisé les nouveaux inputs disponibles avec HTML5(22) afin de proposer à l’utilisateur des champs text, nombre, mais également date et heure afin d’adapter les formulaires à l’entité concerne.

Algorithme d’ajout d’une entité:

DEBUT

→Renseignement des champs du formulaire

→Envoi du formulaire

→Vérification de tous les champs dans une servlet

SI le formulaire est invalide

→Redirection sur la page d'ajout

SINON

→L'entité est ajouté au Datastore

→Redirection sur la page d'affichage de l'entité

FIN SI

FIN

De plus, pour l'ajout d'une rencontre, nous avons utilisé un plugin jQuery(23) afin de faciliter la saisie du lieu ainsi que des coordonnées d'une rencontre en sélectionnant simplement un point sur une carte google maps(INDICE 24).

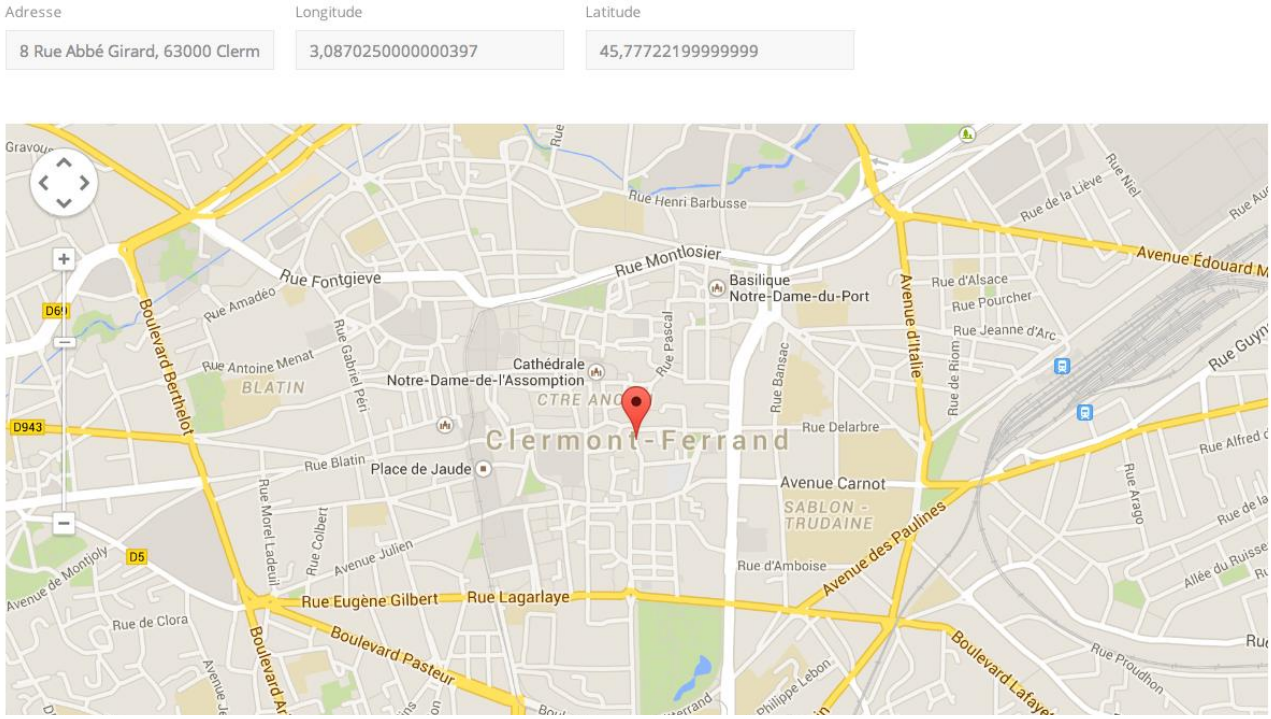


Figure 12 - Ajout des coordonnées de la rencontre

Listage:

La page d'affichage est accessible en cliquant sur l'entité désirée dans le menu du header, lors du clique, une servlet est appeler, celui-ci génère la liste de toutes les entités existante et les retournes a une page .jsp qui va à partir de cette liste différents tableaux javascript(25) utilisé pour le filtrage. C'est également à partir de la page de listage des rencontres que l'utilisateur peut

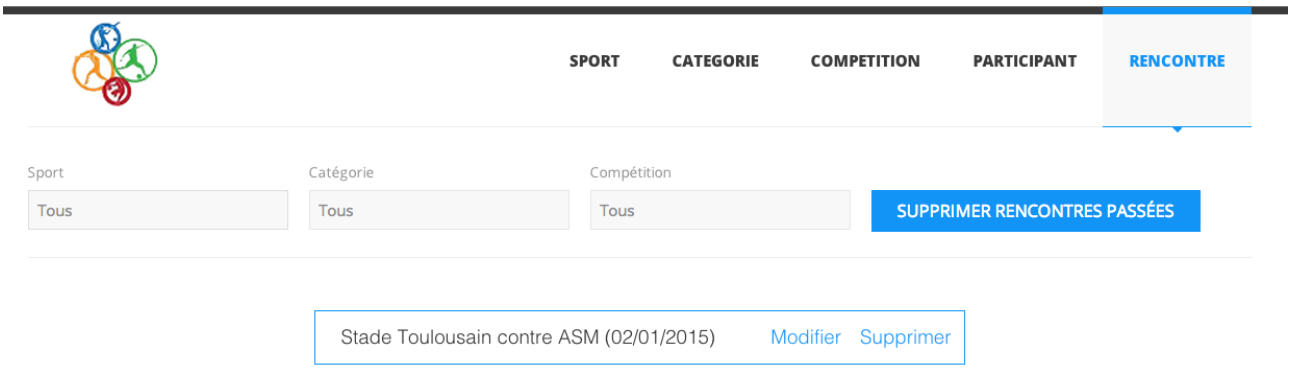


Figure 13 - Exemple listage des données

supprimer toutes les rencontres qui sont passés depuis plus d'une semaine.

Algorithme d'affichage d'une entité:

DEBUT

→Clique sur le nom d'une entité dans le menu

→Génération d'une liste d'entité par la servlet

→Génération de tableaux variable JavaScript associant chaque entité a ses entités parentes à partir de la liste passée depuis la servlet et affichage de la page

→Modification d'un filtre

→Suppression de toutes les entités présente sur la page

→Parcours du tableau adapté

→Affichage des entités concernées

FIN

Algorithme de suppression des entités passées:

DEBUT

→Clique sur le bouton de suppression des rencontres passées

→Redirection vers une servlet qui récupère toutes les rencontres dans le DataStore

→Suppression des rencontres passées depuis plus d'une semaine

→Redirection sur la page de listage des rencontres

FIN

Modification:

Le formulaire de modification d'une entité est disponible à partir de sa page de listage. Le formulaire d'affichage est conçu de la même façon que le formulaire d'ajout, a la seule différence que celui-ci est prérempli avec les valeurs actuelles de l'entité en cours de modification. Au niveau de la servlet, le principe est également le même.

Algorithme de modification d'une entité:

DEBUT

→Génération du formulaire pré-rempli avec les valeurs par défaut

→Modification des champs du formulaire

→Envoi du formulaire

→Vérification de tous les champs dans une servlet

SI le formulaire est invalide

→Redirection sur la page de modification

SINON

→L'entité est modifié dans le Datastore

→Redirection sur la page d'affichage de l'entité

FIN SI

FIN

Suppression:

La suppression se fait également sur la page de listage de l'entité, lors du clique sur le label supprimé, une pop-up de confirmation est affiché et si l'utilisateur valide, une servlet est appelée et supprime l'entité dans le DataStore.

Algorithme de suppression d'une entité:

DEBUT

→Listage des entêtées

→Clique sur le label "supprimer"

→Demande de confirmation

SI confirmation

→Redirection sur une servlet

→Suppression de l'entité dans le datastore

→Redirection sur la page de listage

FIN SI

FIN

3.4) Mise en place du Backend

Les smartphones font de plus en plus partie du quotidien. Cependant, ces dispositifs sont faibles en ressources et disposent de peu d'espace de stockage. Afin de pallier ce problème, les informations sont enregistrées dans une base de données hébergée sur un serveur. Le smartphone peut de cette façon accéder à distance aux données et délègue ainsi l'exécution des tâches complexes et le stockage des éléments à une plateforme mobile. C'est ce que l'on appelle un PaaS (Platform as a service).

L'application étant développée uniquement à partir du système d'exploitation mobile Android, c'est tout naturellement que M. Imbert nous a conseillé l'utilisation de la plateforme PaaS : App Engine. Celle-ci permet en effet de développer en toute simplicité des applications et possède et nombreuses fonctionnalités, notamment le stockage permanent.

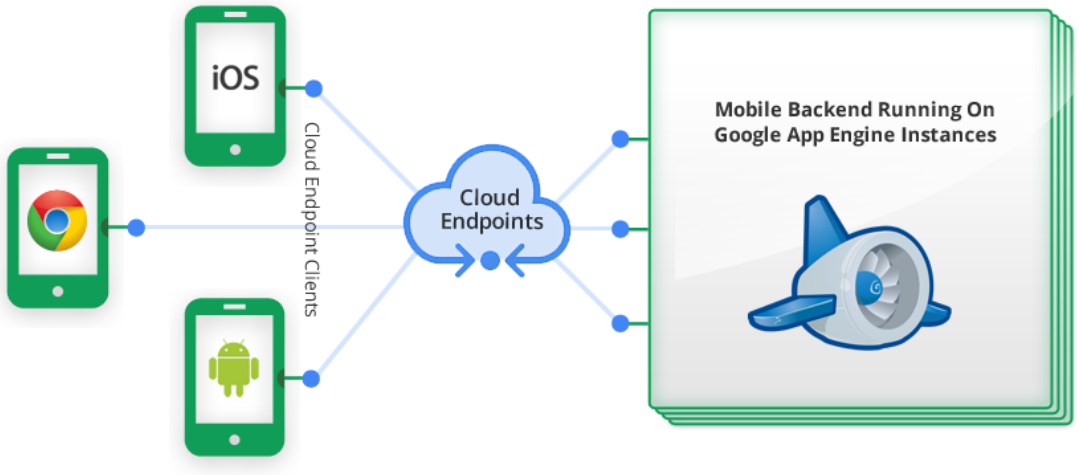


Figure 14 - Schema du cloud enpoints

Comme nous l'avons vu plus haut, nous avons à gauche de la figure les interfaces utilisateur (application et site web) qui se situent côté client. A droite, nous avons le backend qui sera hébergé

sur l'App Engine, donc côté serveur. En résumé, Cloud Endpoint permet de connecter notre application backend avec toutes nos applications frontend(13). Nous séparons ainsi notre application en deux parties logiques.

C'est ce backend que nous avons développé pour construire une liaison entre l'application (et le site Web par la même occasion) et le datastore.

Ainsi l'application Android sera constituée de différentes couches. Tout d'abord, nous avons l'IHM qui correspond à la partie visible de l'application. C'est à partir de cette interface que les utilisateurs vont pouvoir faire la demande de requêtes à la base de données. Puis nous avons le backend qui contient toutes les classes et méthodes qui servent à manipuler les objets de la base de données.

3.4.1) Le datastore

La base de données de Google App Engine appelée le datastore repose sur le SGBD Big Table(28) 0développé par Google. Tout d'abord, il est important de savoir que le datastore n'est pas une base de données relationnelle. C'est juste une Map ordonnée multidimensionnelle dans laquelle les données sont stockées sous forme de <clé, valeur>. Ainsi, l'avantage du datastore est qu'il n'utilise pas de tables donc pas de schéma SQL. Il n'est pas nécessaire de prédéfinir en avance sous quel format les données doivent être enregistrées.

3.4.2) Objecty

Nous venons de voir que le datastore est un système puissant de stockage. Afin de pouvoir le manipuler et ainsi effectuer des requêtes, nous avons utilisé la bibliothèque Objectify. Objectify(1) est un peu comme une sorte d'ORM(29) pour le Datastore. En effet avec le Datastore, Google fournit une API de bas niveau permettant de manipuler les entités du Datastore. Afin de simplifier le code et la manipulation des entités, nous avons utilisé une API de plus haut niveau appelé Objectify développé par une organisation extérieure à Google et qui permet de faire des requêtes sur le Datastore en nous cachant l'utilisation de l'API fourni par Google. Objectify permet donc la communication et l'échange d'informations entre les applications et le serveur.

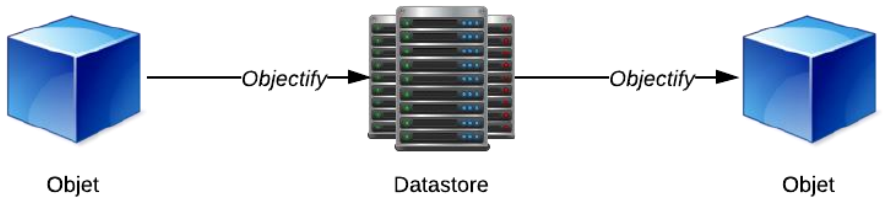


Figure 15 – Système de communication du datastore avec les objets

Objectify transforme donc nos entités Java créer dans l'application en objet destiné au Datastore et inversement elle transforme les entités du Datastore en objet Java utilisable dans notre application.

La mise en place de Objectify se fait à l'aide d'annotations. Nous allons d'abord voir comment créer une entité destinée à être stocké sur laDatastore :

```
@Entity
public class Categorie {

    @Id
    private Long id;
    private String libelle;
    @Index
    private Long idSport;

    ......

}
```

Figure 16 - Représentation d'une classe avec objectify

Tout d'abord, il y a l'annotation @Entity devant la définition de la classe pour qu'Objectfy sache qu'il s'agit d'une classe qu'il peut persister. Ensuite, chaque entité dispose d'une clé qui l'identifie de manière unique parmi toutes les entités du datastore. Cet id est renseigné via une propriété sur l'objet annoté @Id. Il est de type Long et va être généré automatiquement par le datastore lors de l'ajout d'une entité à ce dernier.

Enfin, l'annotation @Index permet d'indexer un attribut de l'objet en le précédant ce qui permet d'effectuer une requête sur cet attribut.

Ensuite il a fallu concevoir les classes et méthodes permettant d'effectuer ces requêtes sur ces entités.

```
@ApiMethod(
        name = "getBySport",
        path = "categorieBySport/{idSport}",
        httpMethod = ApiMethod.HttpMethod.GET)
public List<Categorie> getBySport (@Named("idSport")Long idSport) throws NotFoundException {
    List<Categorie> listCategorie = ofy().load().type(Categorie.class).filter("idSport", idSport).list();
    if (listCategorie == null || listCategorie.size() == 0) {
        throw new NotFoundException("Could not find Categorie with idSport: " + idSport);
    }
    return listCategorie;
}
```

On peut voir un extrait de code ci-dessus, dans cet extrait on peut voir la méthode qui nous sert à récupérer toutes les catégories d'un sport. On peut voir là aussi différentes annotations utilisées par Objectify . Par exemple chaque méthode possède l'annotation @ApiMethod qui permet de décrire le nom de la méthode avec aussi la méthode http utilisé.
Dans cet extrait on peut aussi voir l'utilisation de objectify avec les lignes suivantes :
List<Categorie> listCategorie = ofy().load().type(Categorie.class).filter(« idSport », idSport).list() ;
Cette ligne indique à Objectify que l'on veut charger des objets de type Categorie en appliquant un filtre sur le champ idSport. On récupère ainsi les catégories du sport qui possède pour id, l'idSport passe en paramètre.

Avec Objectify la récupération d'objets tient donc en seule ligne. Si on avait choisi d'utiliser l'API fourni par Google ceci aurait surement été plus compliqué.

4) Résultat final

Au final, nous avons pu livrer l'application Android(26) ainsi que le site d'administration fonctionnels, nous sommes arrivés à ce résultat en se répartissant les taches équitablement.

4.1) Application Android

L'application Android est fonctionnelle, cependant le cahier des charges n'est pas complètement validé étant donné que nous n'avons pas eu le temps de développer les 4 users story(INDICE 27) correspondant à l'ajout de 4 entités (un sport, une catégorie, une compétition ou un club) dans une liste de favoris qui aurait permis à l'utilisateur de retrouver les rencontres associées rapidement.

b) Interface web d'administration

Le site web est également fonctionnel et entièrement terminé. Le site offre à l'administrateur une interface pour gérer l'ensemble des données présente dans le DataStore.

4.2) Difficultés rencontrées

Manque d'information :

La première difficulté rencontrée a été le manque d'information à propos du projet au tout début de celui-ci. Nous n'avions pas encore les informations nécessaires pour commencer son étude. Quelles étaient les intentions du client vis-à-vis de l'application ? Quelles technologies devons-nous utiliser ? Sous quel système d'exploitation doit-elle être développée ?

Nous avions débuté une analyse en ayant une définition assez vague du sujet, de multiples interrogations se posaient et nous ne savions finalement pas dans quelle direction nous devions partir.

Nouvelles technologies :

Une des difficultés majeures a été l'utilisation de nouvelles technologies de développement. D'une part, Android est un système d'exploitation sur lequel nous n'avions peu voir aucune expérience. Son architecture de base, avec sa multitude de dossiers et de fichiers nous à forcer à les étudier avant de se lancer dans son développement.

Mise en place du DataStore :

La difficulté la plus chronophage fut la mise en place de tout le backend, en effet il nous a été imposé de travailler avec le DataStore de Google qui au final s'avère être un outil très puissant mais qui nous a demandé beaucoup de travail et de temps afin d'en comprendre le fonctionnement et de pouvoir le mettre en place au sein de notre application.

Partage d'une rencontre :

Le partage de l'application a également été un problème, car nous avions commencé à développer cette fonctionnalité avec seulement les applications installées de base sur les téléphones, lors du test après avoir téléchargé l'application Facebook, nous nous sommes rendu compte qu'il était impossible de partager du texte brute. Il nous a donc fallu opter pour une autre solution, à savoir le développement d'une page web dont l'URL serait partagée.

4.3) Améliorations possibles

Ajout des favoris :

Faute de temps, la gestion des favoris fut les seuls User Stories(28) que nous n'avons malheureusement pas pu implémenter cela serait donc l'amélioration la plus souhaitable.

Amélioration / optimisation des accès au DataStore :

En effet, malgré le fait que les accès au DataStore se font quasiment instantanément lors de la navigation sur l'interface web, il n'en est pas de même pour l'application Android où ils sont extrêmement lents. Malgré une optimisation du code et quelques recherches sur internet, nous n'avons pas trouvé à quoi était dû cette lenteur.

Gestion de différents utilisateurs :

Le site web dispose d'un utilisateur unique, il pourrait être intéressant pour l'administrateur d'ajouter d'autres utilisateurs via un formulaire avec éventuellement des droits restreint (accès à un seul club, sport, etc) ,ceci permettrait à différents clubs sportifs d'ajouter eux-mêmes leurs rencontres.

Adapter pour une tablette, optimisation des différents écrans :

Nous avons commencé à adapter l'application Android a différentes résolutions et taille d'écran afin que l'application reste lisible sur les écrans à haute résolution et qu'aucun élément ne dépasse de l'écran au contraire sur les écrans à faible densité de pixels. Cependant l'application n'est pas identique sur tous les écrans de smartphones. De plus un layout spécial pour les tablettes ainsi que pour le mode paysage pourrait être mis en place afin d'utiliser l'espace au mieux sur l'écran.

5) Conclusion

Au terme de ces 10 semaines de projet nous donc finalement réussit à développer une application opérationnelle. Nous avons beaucoup apprécié de travailler sur ce projet car il nous a permis de découvrir la réalisation d'application mobile et plus particulièrement d'applications Android.

Ce projet nous a donc permis de découvrir de nouvelles technologies comme Android mais aussi d'approfondir nos connaissances notamment en Java(2) ainsi que sur les JSP(10) pour la partie web de notre projet. Il nous a aussi appris de nouvelles méthodes d'organisation, car pour deux d'entre nous c'était la première fois que nous appliquions la méthode SCRUM(11) pour un projet.

En conclusion, ce projet est pour nous une réussite de par le fait que nous avons réussi à livrer une application fonctionnelle, et de par les connaissances qu'il nous a apportées. Malgré les difficultés rencontrées nous garderons donc un très bon souvenir de ce projet et mettrons le savoir-faire accumulé et nous retiendrons les erreurs commises pour ne pas les reproduire pour d'autre projets.

6) English summary

This ten-week project is part of our degree in mobile platforms at the University Institute of Technology in Clermont--Ferrand.

The objective of this project was to design and develop an application from scratch that would allow the user to quickly and easily see the nearest upcoming sports games. We also had to develop a website. For this task, we did not start from scratch as we used a graphic template and we just developed the content. This website is made only for the administrator to update the games, sports and teams available.

For this project, we worked for a real client but we never met him. M.Imbert, our supervisor acted as intermediary. M. Imbert went through the project specifications with us so that we could begin the analysis stage.

After the research & analysis stage, we could start the implementation stage and each Thursday until the end of the project we released a beta version and our supervisor presented it to the client, except the last Thursday where we release the final version for both, the application and the website.

We encountered some difficulties with the google datastore (which is where we retrieve data) . That was something that we had never seen before and it was hard to set up.
Another problem linked to the datastore, is the time the application needed to retrieve data, which impacted the user experience. We didn't succeed in solving this problem, because it's due to the datastore capacity.

If we had had more time, we would have developed and as a final feature to allow user to mark a sport or a team as a favorite to show related data in priority.

In conclusion, the application we created is over 90 % complete and the application is ready to be used by the users.

7) Webographie

Documentation général :

- http://stackoverflow.com/
- http://openclassrooms.com/courses/montez-votre-site-dans-le-cloud-avec-google-app-engine/manipuler-le-datastore-avec-objectify
- http://rominirani.com/category/cloud-computing/google-app-engine/
- **http://rdonfack.developpez.com/tutoriels/cloudendpoints/**

Documentation Android :

- http://developer.android.com/
- http://openclassrooms.com/courses/creez-des-applications-pour-android

8) Lexique

1) Objectify : Objectify est une API Java conçu pour le Datastore de Google App Engine

2) Java : Java est un langage de programmation orienté objet.

3) Pattern : Un pattern est la meilleur solution connue à un problème de conception récurent.

4) Toolbar : Barre d'outils d'Android qui se situe au-dessus de l'application.

5) Datastore : Le datastore est un outil créé par Google permettant de stocker des données sous forme de clé valeur.

6) Thread : Un thread est un tâche similaire à un processus à la différence que tous les threads d'un processus partage la même mémoire.

7) Layout : Un layout défini la structure de l'interface utilisateur. Il gère la manière dont les éléments graphique doivent s'afficher.

8) Gone : Visibilité qui cachent l'élément mais qui à la différence de la visibilité « Invisible » fais en sorte que l'élément ne prend pas de place sur l'interface utilisateur.

9) JSP : C'est une technologie basée sur Java permettant de générer des pages web HTML.

10) SCRUM : SCRUM est une méthode agile d'organisation de développement d'u projet

11) CRUD : Crud est l'abréviation de Create Read Update Delete, il s'agit d'un type de site gérant des données le plus basiquement possible, il permet l'ajout la suppression, l'affichage et la modification de ces données.

12) Backend : le backend est la partie qui n'est pas visible par les utilisateur.

13) Frontend : Le frontend constitue la partie visible de l'application. C'est l'opposé du backend.

14) Servlet : Une servlet est un programme exécuté sur un serveur permettant de gérer dynamiquement des fichiers au format html, par exemple en récupérant des données correspondant à certains critères.

15) Endpoint : Un endpoint est un point d'accès à une ou plusieurs données.

16) Champ select : Un champ select est le nom défini par la norme HTML5 pour désigner une liste déroulante dans un formulaire web.

17) Header : Un header ou entete en Francais est le nom défini par la norme HTML5 pour définir l'entete d'une page web.

18) Footer : Un footer ou pied de page en Francais est le nom défini par la norme HTML5 pour définir le pied de page d'une page web

19) Balise head : La balise head est l'élément HTML5 permettant de fournir des méta-informations sur une page web au navigateur par exemplequel style lui appliquer.

20) Include jsp : En jsp, un include permet de d'inserer le contenu d'un fichier dans un autre au moment de la génértion de la page, cela permet d'écrire une seul fois le code d'une partie de page web qui doit etre present sur differente page.

21) Controleur : cf definition d'une servlet.

22) HTML5 : format de données conçu pour représenter les pages web.

23) JQuery : bibliothèque JavaScript créée pour faciliter l'écriture de scripts côté client dans le code HTML des pages web.

24) Google maps : service gratuit de cartographie en ligne.

25) Javascript : langage de programmation de scripts principalement employé dans les pages web interactives mais aussi pour les serveurs

26) Android : systeme d'exploitation pour smartphones aujourd'hui développé par Google, il est également adapté pour être utilisé sur des téléviseur, montres connectés et voitures.

27) User story :

28) SGBD Big Table : système de gestion de base de données compressées, haute performance, propriétaire, développé et exploité par Google.

29) ORM : technique de programmation informatique qui crée l'illusion d'une base de données orientée objet à partir d'une base de données relationnelle en définissant des correspondances entre cette base de données et les objets du langage utilisé.

30) Serveur : Système informatique destiné à fournir des services à des utilisateurs connectés.

31) APK : un format de fichier utilisé pour distribuer et installer des logiciels (la plupart du temps des applications ou des jeux) au sein du système d'exploitation Android.