SWARMAPP PROJECT

Final Report

Thibault Viravau

Thibault.viravau@ensta-bretagne.org

Intentionally blank page.

Résumé

Ce document constitue le rapport final de mon projet d'Application Système (Unité de Valorisation 5.4) de troisième année de cursus d'élève Ingénieur des Études et Techniques de l'Armement à l'ENSTA Bretagne, dont la finalité est de confronter les élèves aux problématiques du métier d'ingénieur ou ingénieur chercheur, immédiatement avant leur Projet de Fin d'Études.

L'objectif du présent projet était le développement d'un logiciel permettant de suivre en temps réel le déroulement d'une compétition ; dans un premier temps il s'agit du contexte de compétitions de robotique marine, mais le concept peut être étendu à d'autres domaines de compétitions ou manifestations sportives. Ces notions d'extensions ne seront pas abordées exhaustivement

En particulier, l'intérêt du projet était de compléter les moyens de localisation des compétiteurs et les moyens de visualisation du déroulement de la compétition, qui se résumaient à un site internet à l'heure où le projet SWARMAPP a commencé. Les objectifs que doit remplir ce logiciel sont : premièrement, permettre aux compétiteurs d'émettre et visualiser leurs positions, deuxièmement, permettre aux organisateurs et spectateurs de visualiser le déroulement de la compétition sur ordiphone, et troisièmement, de proposer un mode déconnecté du réseau afin de permettre à l'utilisateur d'enregistrer des parcours personnels et d'enregistrer des points d'intérêts.

Un objectif s'est rajouté au projet durant son développement : développer une Application légère dont le rôle est uniquement d'envoyer la localisation courante au serveur ; l'ordiphone peut alors servir de balise pour la compétition.

Objectifs du projet :

- Réaliser une étude bibliographique
- Formaliser les fonctionnalités attendues du logiciel
- Établir l'architecture des Applications en concordance avec l'interface offerte par le serveur actuel
- Développer et intégrer les composants logiciels des deux Applications (légère et complète)

Objectifs atteints :

- ✓ Étude bibliographique
- ✓ Formalisation du besoin
- ✓ Établissement de l'architecture des Applications
- ✓ Développement de l'Application légère

Mots-clés :

- Suivi de compétition
- Suivi d'évènement sportif
- Compétition de robotique marine
- Développement Android
- API localisation Google

Abstract

During their curriculum at ENSTA Bretagne, third year student are asked to work on a short-term project, so as to prepare them for their last internship before they end their learning.

The project I chose to work on tackles with the development of an Android Application that would make it possible to monitor a competition from an Android smartphone. As in first approach, the context of such competitions is restricted to marine robotic competitions, but the concept can be virtually applied to every kind of competitions or sporting event. The present report will not deal with those possible extensions and we will mainly consider the World Robotic Sailing Championship as our context.

In particular, all the interest of the project lies in completing tools for monitoring and location transmitting: presently the only monitoring tool is the event website, developed and maintained by ENSTA Bretagne students, and the main location device is an electronic card on which ENSTA Bretagne's students worked as well. The three main objectives of the SWARMAPP Application are: first, to allow competitors to communicate their location to the server and visualize it; second, to permit organizers to monitor the competition from a smartphone; third, through a stand-alone mode, to make it possible for the user to record personal circuits or paths, possibly including Points Of Interests.

In the middle of the project an objective has been added during one of the weekly progress meeting: Android devices should be able to behave like beacons and complement the electronic cards currently under development.

Project's goals:

- Bibliographic searching
- Make out of the software's expected features
- Profile the Applications' architecture with regard to the present server's interface
- Develop and integrate the Applications' components (light and complete Applications)

Achieved goals:

- ✓ Bibliographic searching
- ✓ Making out of the features
- ✓ Profiling the architecture
- ✓ Development of the light Application

Key words:

- Competition monitoring
- Sport event monitoring
- Marine robotic competition
- Android Development
- Google Services API

Table of content

Introduction

Purpose of the document

This document finds its roots in the context of engineers' classes at ENSTA Bretagne, within the 5.4 course which aims at confronting students to industrial issues by asking them to design a solution according to a need, after a preliminary phase of state-of-the-art. This document is the final report of SWARMAPP project of which Thibault VIRAVAU is in charge, supervised by Olivier REYNET, teacher at ENSTA Bretagne. Its purpose is to summarize all the work done during this project.

Context of the project

SWARMAPP – SWARM tracking (Android) APPlication – is a project developed in connection with SWARMFRAME – SWARM tracking web FRAMEwork –, the goal of which was initially to offer a web User Interface to manage and supervise robots swarm. This aim has evolved into being able to offer to a sailing event administrator (the client) a user-friendly and easy-to-use interface to supervise its event, whereas before it was locked into one kind of event only, the World Robotic Sailing Championship. SWARMFRAME developers are Benoit BOURDON and Bastien DROUOT. In this document I will refer to "the event server" as "the server", either it is for the WRSC or any competition.

Purpose of the project

As stated in *Abstract*, SWARMAPP's role in this frame is to develop an Android Application to:

- Turn a competitor's Android mobile phone into a beacon able to send and receive data from the event's server (see *1.1* for more details) and more generally, access the Web of Things[1].
- If the user is not a competitor, make it possible for him to follow the event of his choice.
- Offer a possible stand-alone offline use where the user can record localization data and POIs[2], e.g. when hiking.

The light Android Application described too in Abstract has for objectives:

- Get the smartphone current localization
- Send it to the server
- If localization could not be send to server, offer a way to manually dump data

N.B.: The kind of client/server system we have to build is nothing *really* innovating: developing an Android Application is something quite common, like designing a house for an architect is not really innovating; but of course it still takes time, efforts and a good methodology to be correctly done.

[1] The **Web of Things** (WoT) is a term used to describe approaches, software architectural styles and programming patterns that allow real-world objects to be part of the World Wide Web. (http://en.wikipedia.org/wiki/Web_of_Things).
[2] Points Of Interest.

1 System Engineering phase

System Engineering is an important interdisciplinary course given at ENSTA Bretagne which aim is to focus on "how to design and manage complex engineering systems over their life cycles" [3]. The way I followed the S.E. process is the following:

- Stating the need
- Proceed to the state-of-the-art : catch the most problematic issues
- Analyze and refine the need : split it into several main themes
- Attribute technical solutions to needs
- Choose among several possible physical architectures

Once a candidate physical architecture has been chosen, the development begins. I did not explicitly write test specifications neither for unitary tests or interfaces tests, for the reason that good programming practices compels the developer to test every function he writes, step by step.

1.1 The need according to the client

One of the purposes of the Status Report, previously handed, was to state the need as the client expressed it. Here is a short summary of what was stated in the "Specifications" part:

- The Application must offer an online and an offline mode
- Online mode will be divided into competitor mode and spectator mode:
    - Competitor mode offers the possibility to send/receive data to/from the server and to display a map with the latest positions
    - Spectator mode offers the possibility to display a map with the latest positions of the chosen competitors, by requesting data to the server
- Offline mode is divided into replay mode and "Your own circuits" mode:
    - Replay mode allows a competitor to retrieve his data from the server (as long as the server can legally keep them) and display them as he would if he was spectating
    - "Your own circuits" mode offers the possibility to record paths and add POIs or geofences[4] to the circuit.

A latter reunion with the client/supervisor led to choose to develop a light Application turning a smartphone into a beacon, in priority.

1.2 State of the art

The Status Report also included the state-of-the-art at the moment when the need have been stated. We (SWARMAPP and SWARMFRAME teams) evoked the different technologies for client/server communication, and the different technologies for displaying the map in the Application. Please refer to the status report for further information about these technologies.

As stated in the Status Report, we chose JSON formatting for client/server communication, and Google Maps API to display maps, both for the website (server-based) and for the Application (Android-based).

---

[3] Wikipedia : System Engineering (http://en.wikipedia.org/wiki/Systems_engineering)
[4] See Wiktionary : geofence http://en.wiktionary.org/wiki/geofence

1.3 Forecasted schedule

The last purpose of the Status Report was to present the forecasted development schedule, which the "Rapport d'avancement" updated. The initial schedule was the one presented on "*Chart 1: Originally forecasted Gantt Diagram*". Currently, the light Application is totally done, but it triggered delay in the normal development schedule, which explains that the forecasted schedule is absolutely not respected[5].

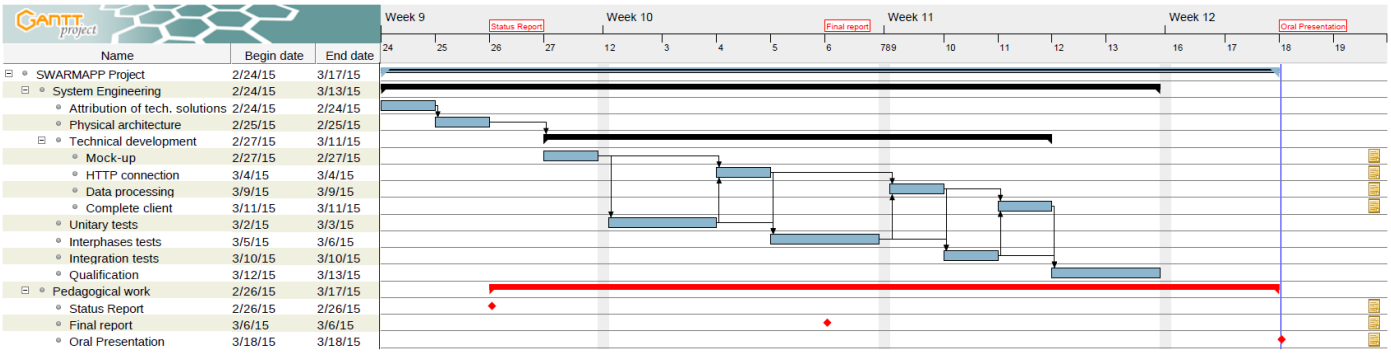


Chart 1: Originally forecasted Gantt Diagram

1.4 Classification of the need

After the redaction of the Status Report comes the step of analysis of the need: one category is allocated to each need; the result of this is presented in "*Appendix: Expression of the need*", in French. It presents the need as expressed, quite roughly, with a figure corresponding to the given category:

0. for miscellaneous
1. for security
2. for data treatment
3. for graphical interface
4. for message sending functions
5. for database operations
6. for optional functionalities

This classification permits to get to the second step: refining, which is the breaking down in sub-parts. The result of refining is presented in "*Appendix: Refinement of the need*": every sentence have been itemized into atomic function. These atomic functions are then associated with one or several technical solutions in the next step.

An example of an atomic break down is the following: (sentences begin with "the Application must")

2.
- 2.1. Receive information from server
  - 2.1.1. …
  - 2.1.2. …
  - 2.1.3. …
- 2.2. Send data to server
  - 2.2.1. Possess a way to connect to server
  - 2.2.2. Know the server's address
  - 2.2.3. Be able to pack data according to the server's unpacking method
- 2.3. (same as 2.2.)
  - 2.3.1.…

[5] As in any project?

1.5 Attribution of technical solutions

Once the need has been classified and broken down to atomic functions that the system must fulfill, the functions are given one or several technical solutions. The result of this process can be found in "*Appendix: Attribution of technical solutions to needs*" and constitutes the ingredients to build different candidate architectures.

For example one can pick any of the "vertical bar"[6]-separated technical solutions for each line and build a system with these technologies: the so-built system will fulfill the client's needs.

With regard to the different candidate architectures, I chose the following features:

"heavy" Application:

0. Two menus: one for choosing the working mode, one for authentication

2.1.1. HTTP[7] request
2.1.2. Wi-fi if available, mobile network elsewise
2.2.2. Ask the user as he registers the event
2.2.4. Google Services API for localization (more efficient)
2.4.3. SQLite Database featured by Android

3.1.1. Google Maps API

4.1.3. Vibrations + 30" timer before the message is sent + deactivation passcode

light Application:

The candidate architecture for the light Application was more straightforward than for the heavy one thus no choices were to be made. Moreover, the light and the heavy Application will share the same server, so they will proceed identically when requesting data; which imposes to the light Application several features of the heavy one.

[6] It seems that no word is clearly defined to designate "|" : http://en.wikipedia.org/wiki/Vertical_bar
[7] HTTP (HyperText Transfer Protocol) is a communication protocol used to format and transmit messages using the Internet

1.6 Physical Architecture

Once the features are chosen, the physical architecture phase can begin: it consisted in my case in drawing a mock-up Application. The physical architecture as I drew it represented the different Android Activities and transitions between Activities: such button here triggers such HTTP request which response is analyzed either to display a pop-up message or to open the next Activity.

"*Chart 2: Partial representation of the preliminary Physical Architecture*" shows a part of the drawing made to describe the mock-up Application which I began to create right afterward and which is described in part "*2.1. Mock-up*". It shows how the user is invited to select online or offline mode. In online mode a drop-down menu allows him to choose the event he will spectate or participate in, or to add a new one in the "Event adding Activity". Spectating doesn't need a {username, password} couple, but participating as a competitor does. Pressing [connection] takes the user to the main menu for competitors, where he can choose to display the map, send an emergency message, or start/stop the localization.

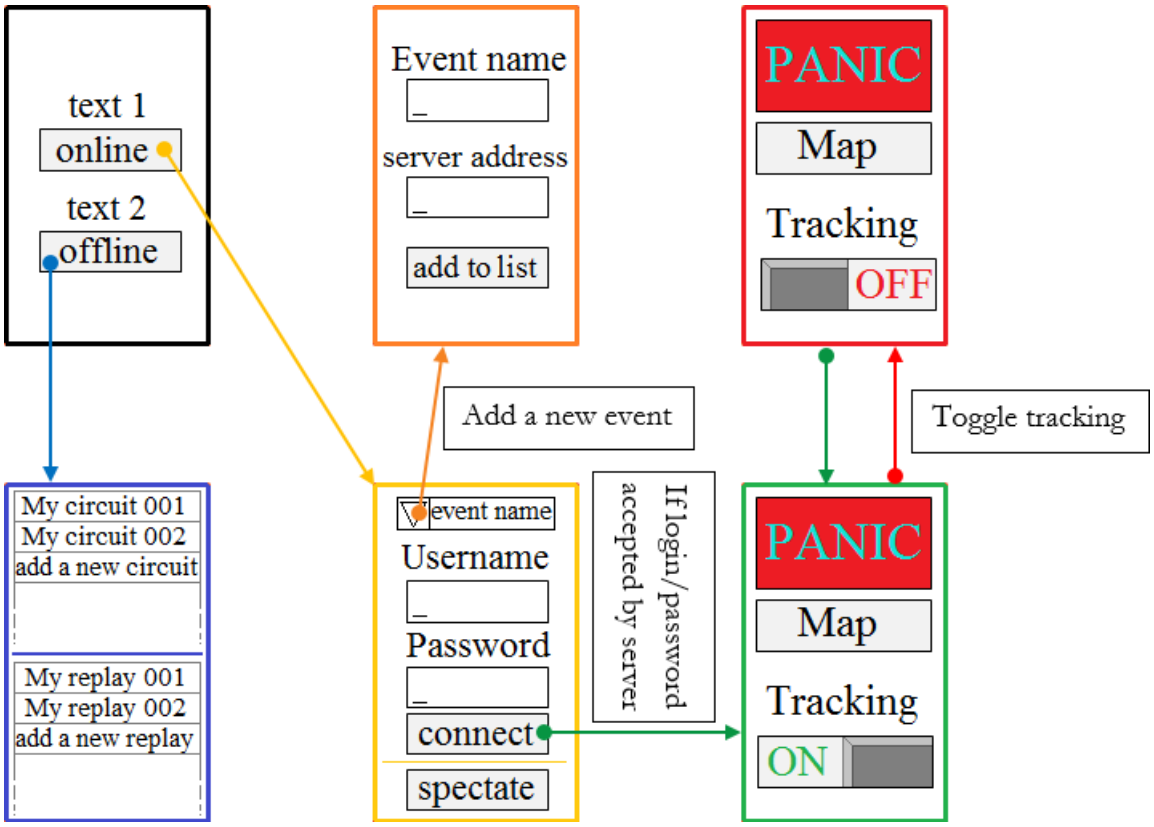


Chart 2: Partial representation of the preliminary Physical Architecture

2 Development phase

The development phase began with the creation of the mock-up Application even though there is no functional part in this version: it is a mere model of how should the final Application look. After the mock-up came the initial development of the heavy SWARMAPP Application. As core functionalities are developed, complex tools must be mastered, what requested to develop a prototype featuring only the core functionalities to learn to implement and use them.

From this prototype was born the Light Tracker which is, at the time when this report is written, totally operational.

2.1 Mock-up

The mock-up is a more detailed version of the physical architecture given that it is created as a real Android Application – i.e. with the same software, but carrying no runnable code within. It is purely visual, its graphical interface is shown in "*Chart 3: preliminary visuals of the final Application*". This version of the Application makes it possible to show the final rendered whereas no development has begun, and it constitutes the skeleton of the final Application, to which functionalities will be added.

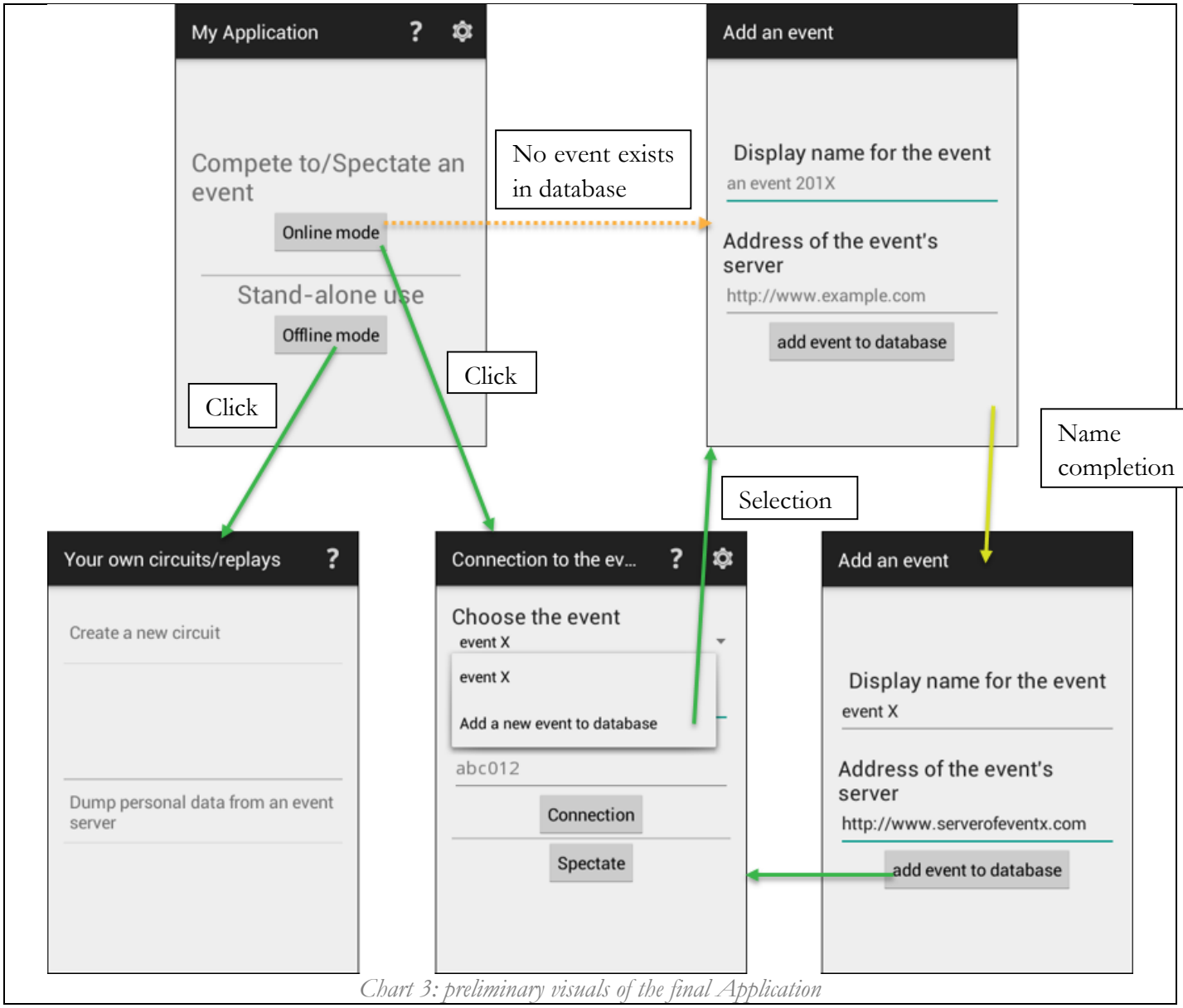


Chart 3: preliminary visuals of the final Application

2.2 Heavy Application

Once most of the mock-up is done, turning it into the heavy Application only consists in writing code to make the mock-up proceed to real data treatment, such as HTTP requests to switch from one Activity to another, or database queries to display dynamic information. At the moment of the writing of this report, the development of the complete Application have been paused to finish the light Application because of two elements:

- Several technologies needed to be mastered before being able to continue, thus calling for a testing Application
- During a meeting with the project supervisor, Olivier Reynet, we decided it would be positive to have a light Application acting like a beacon, which I talked about in the previous parts

At the moment, the heavy Application is only the mock-up enhanced with some database features, but the Light Tracker being totally operational, the development of the heavy Application can be resumed.

2.3 Testing Prototype

The different and various needs implies that the technologies used are unlikely to cover all the needs, this is why, in the Physical Architecture, only few features are related to another. This consequent number of different technologies to master imposed me to develop a test Application beforehand, for the geolocation acquiring, the database operations features and the HTTP requests with the SWARMFRAME server.

Some visuals of the prototype Application, called TestLocalisation, are visible on "*Chart 4: 3 screenshots of the prototype* Application". The Activity shown in the two last pictures is the one which gather location information, using Google Services API (Application Program Interface). The position indicated is a fake one – located near Sydney –, obtained thanks to a third party tool. (It allows the developer not to have to go outdoor each and every time he asks the testing device to display its location.)

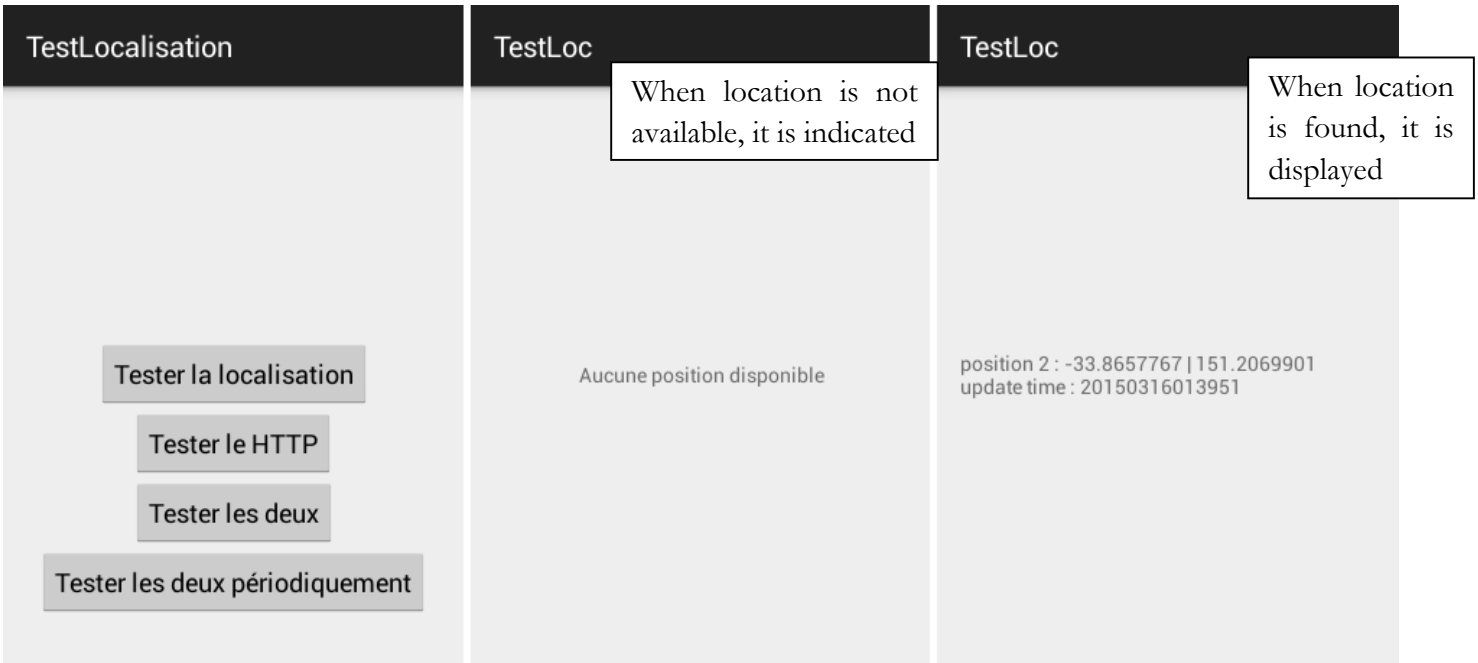


Chart 4: 3 screenshots of the prototype Application

2.4 Light Tracker

Once the prototype Application displayed satisfying results concerning location acquisition and data processing – including transfer to the server and storage in Android SQLite database – the light SWARMAPP Application could be started.

Light Tracker respects the good programming practices stated in "*Appendix: Good programming practices*" and features the following technologies:

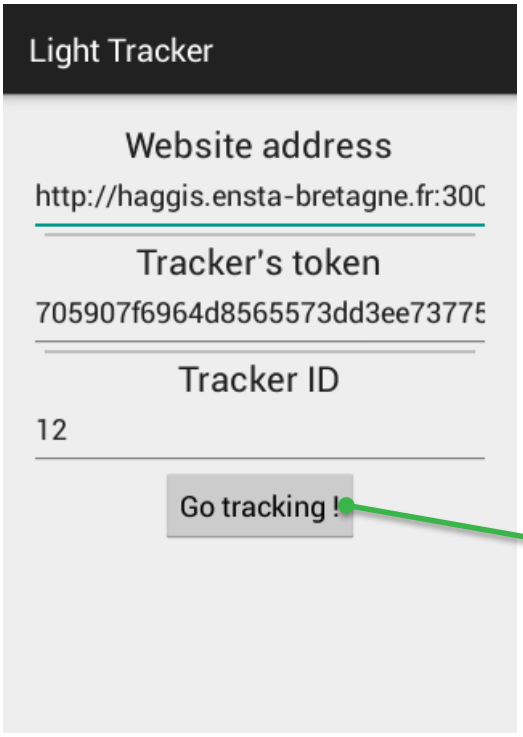
- Java
  - HttpURLConnection: a client for HTTP connections; it is recommended by Android as better than HttpClient which I used before.

- Android
  - Shared Preferences: a light way to keep simple information in record.
  - Toast: a simple way to display short, pop-up messages.
  - ActionBarActivity: a super-class allowing to display an action bar on top of the screen. On chart 3 for example one can see an action bar where the title of the Activity is written. Buttons can be added in an action bar such as the one with the question mark and the one with the gear.
  - Service: a running code with no graphical interface (foreground services in particular which have high priority and can keep running even if the Application is not visible).
  - Notification: they keep visible a running service, invisible otherwise, such as a MP3 player or a geolocation updater which have no graphical user interface.
  - SQLite: a light database library

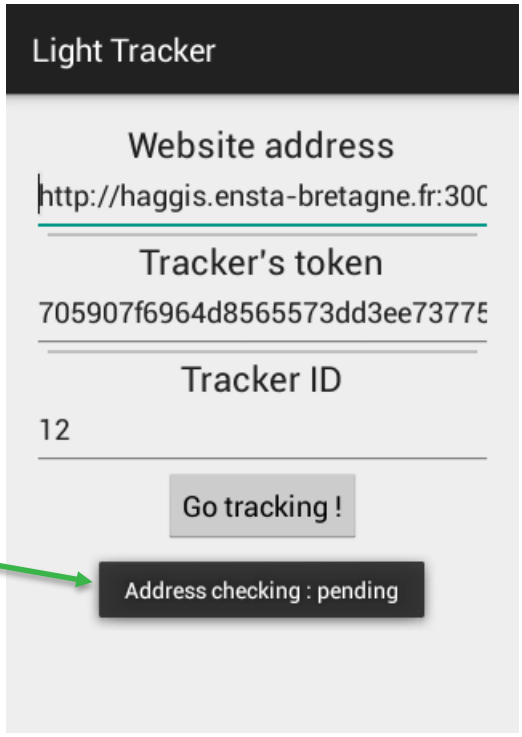
- Google Services API
  - GoogleApiClient: a general interface to access services offered by the Google Services library
  - LocationListener: an object defining callback functions when location is updated or no longer available

3 Results

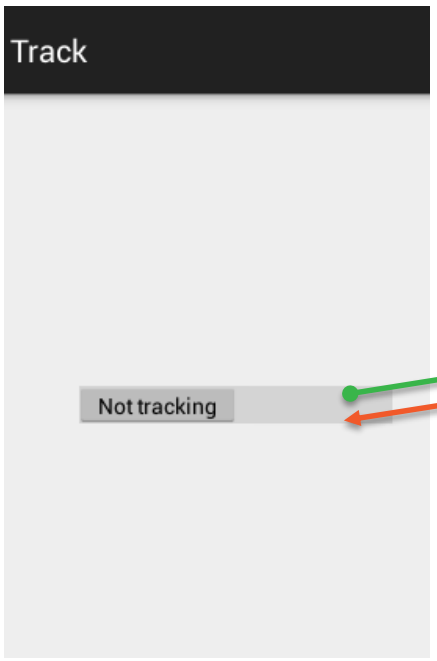
Light Tracker is fully operational, meaning that all needs expressed in "*Appendix: Expression of the need*" is fulfilled. "*Chart 5: screenshots of Light Tracker*" and "*Chart 6: more screenshots of Light Tracking*" are showing the architecture of the Application and the way the user can switch from one Activity to another.



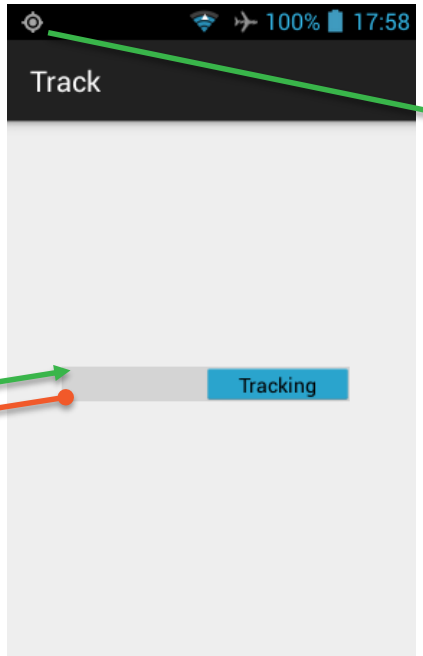
Screenshot 1: the opening menu



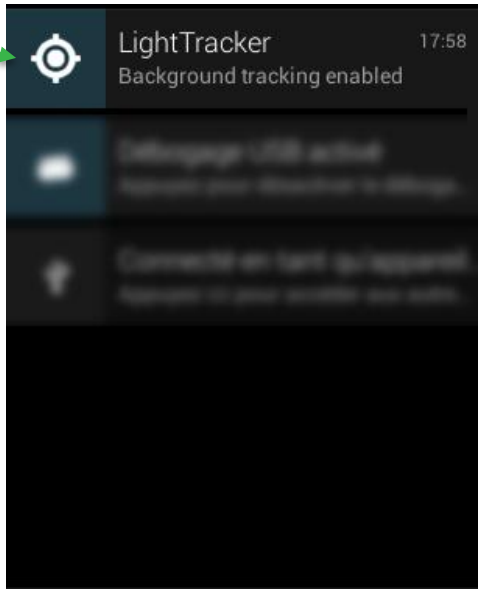
Screenshot 2: clicking "go tracking!" shows a Toast while the Application sends the server a request to make sure it is a correct "event server"



Screenshot 3: Once the address checking has been performed, the tracking Activity appears

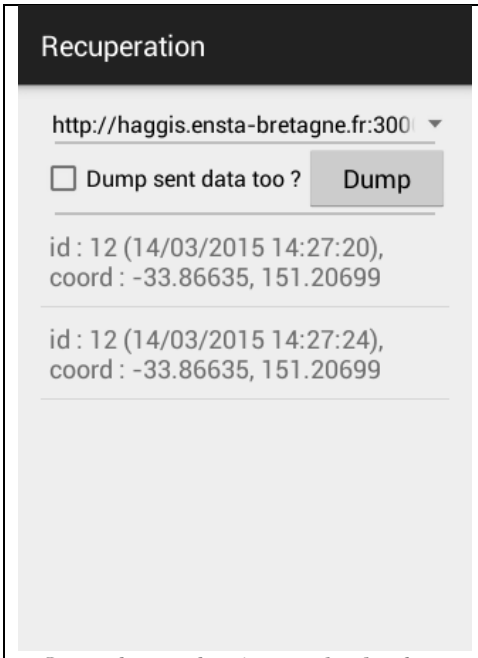


Screenshot 4: the toggle button has been switched on, a notification (folded) appears with the tracking icon

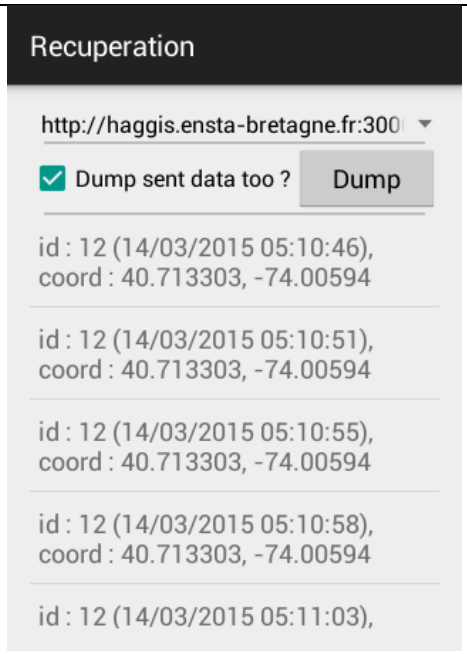


Screenshot 5: the notification (unfolded) when switching tracking on

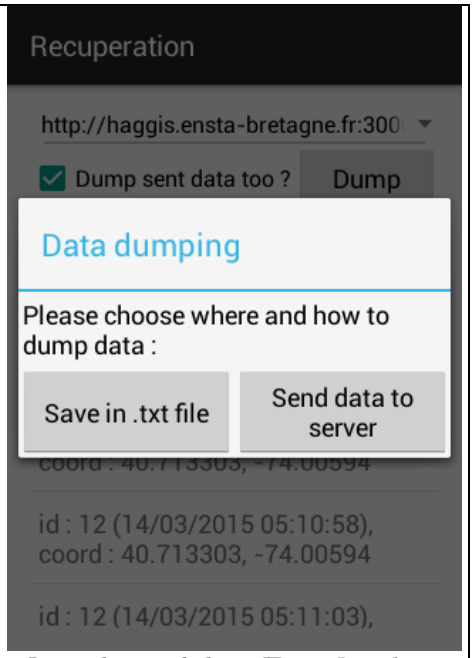
Chart 5: screenshots of Light Tracker



Screenshot 6: the Activity for database dumping



Screenshot 7: when checking "dump sent data too ?", all (old) data appear



Screenshot 8: clicking [Dump] makes a pop-up dialog appear with two buttons

Chart 6: more screenshots of Light Tracking

Screenshot 1 and *Screenshot 2* illustrate that when the user wants to use the device as a beacon, the Application checks the validity of the server's address, and of the {token, tracker id} couple given. If the data entered by the user passes the test, the Activity as shown on *Screenshot 4*, otherwise a Toast indicates the user that an error occurred somewhere, giving short details on the error in question.

Screenshot 3, *Screenshot 4* and *Screenshot 5* represent the tracking Activity: the switch on the screen offers the possibility to turn the update of the location on the server on or off. When data acquiring and sending is on, a persistent notification reminds the user that data are treated in background. The notification is unavoidable mainly for two reasons: to signal the user that the Application is working correctly, and to remind the user to switch tracking off if not necessary any more or if battery is low.

Screenshot 6, Screenshot 7 and *Screenshot 8* show the Activity for data dumping. It is accessible by pressing the "menu" button of the device. The drop-down menu on top of the screen allows the user to choose for which event dump data. The check box allows the user to choose either to dump only data which could not be sent to server or to dump all data including those already sent at the moment when they have been acquired. Clicking on [Dump] opens a dialog window where the user can choose to save data as a text file, or to try to send them again on the server, if an Internet connection is available. Once dumped, all data are marked "sent".

4 Analysis

At the time when I am writing this report, the development of the heavy Application is still paused, but will be resumed as soon as possible with the aim of being functional for March, 17th 2015, when tests will be done in real conditions (device on a boat on the sea, no GPS faker, no Wi-Fi connection).

From this point of view, the project SWARMAPP is not satisfying for the moment; but the late modification of the client's need counter-balances this, given that the need evolved to ask a light Application, which is already fully operational. The communication with the server seems stable and the location updating rate is around ¼ Hz, which is better than what could be obtained before.

From this point of view, the success of project SWARMAPP still relies a little on the development stage of the heavy Application when the tests will be done, but is clearly positive.

Furthermore, considering the good practices I followed (see Appendix: Good programming practices for further details) the code can be easily understood, and the development, resumed.

5 Conclusion

To conclude this report, let us recapitulate the stake of the project and the work done:

- The needed bibliographic searching has indeed been done and was presented in the Status Report handed on February the 26th. The technical issues in such a project boils down to simply assembling different libraries which have to be mastered beforehand, but no research is to be done to find a fulfilling solution to the need.

- The making out of the software's expected features was done and can be found in *Appendix: Expression of the need* and *Appendix: Refinement of the need*. It revealed that four main technical fields appears: graphical interface (map displaying), connection with the server, database storage, location sensors

- Profile the Applications' architecture with regard to the present server's interface: Light Tracker proved during laboratory tests that the interface between the Application and the server works correctly. Tests in real conditions will confirm or invalidate this.

- Develop and integrate the Applications' components (light and complete Applications): the heavy, complete Application is still under development, but the light one is operational.

The remaining work to fulfill the project is mainly to use the current heavy Application as a draft prototype and merge what has been developed in it (mainly database features) with Light Tracker to obtain 80% of the complete Application. The remaining 20% are learning to use: "Android Fragments" and "Google Maps API for Android"; once those two technologies are mastered, the complete SWARMAPP Application will be done.

APPENDICES

Appendix: Expression of the need

SWARMAPP "heavy" Application:

Mode connecté – évènementiel :

- Mode Compétiteur
  - Permettre à l'utilisateur de choisir son mode : participant (= objet connecté = tracker) (0.1 – 0.2)
  - Imposer de se connecter pour accéder aux Activités[8] (1.1)
  - Recevoir les infos relatives à la course, du serveur : missions, alarmes, notifications, réseaux sociaux (2.1)
  - Envoyer les infos au serveur : position, boussole, QR code (2.2)
  - Afficher le parcours/la zone et la progression (3.1)
  - Servir de dispositif d'alerte : « panic button » (4.1)
  - Proposer un raccourci à Instagram (6.1)
- Mode Spectateur
  - Recevoir les informations du serveur (2.3)
  - Afficher le parcours/la zone de l'épreuve, les informations de la course, … (3.2)
  - Proposer de rechercher un participant : nom, numéro, … (5.1)
  - Afficher la progression de 1/n/tous les participants (3.3)
  - Proposer d'envoyer un encouragement à un joueur (4.2)

Mode déconnecté (re-jeu) :

- Proposer à l'utilisateur de récupérer ses données personnelles (2.4)
- Afficher son parcours (3.4)
- Afficher 1/n/tous les parcours (idem 3.3) (3.5)

SWARMAPP "light" Application:

Fonctionnalités :

- Envoyer les données de géolocalisation au serveur enregistré par l'utilisateur (2.5)
- Proposer un fonctionnement en arrière-plan (3.6)

Traitement des données :

- Demander à l'utilisateur à quel serveur envoyer les données (5.2)
- Imposer que le serveur ait été reconnu pour proposer l'usage en tant que balise (1.2)
- Acquérir les données de géolocalisation par GPS (2.6)
- Enregistrer les données (2.7)
- Proposer à l'utilisateur deux moyens de récupérer les données de la Base De Données (BDD) : (2.8)
  - Envoi au serveur
  - Ecriture dans un fichier

[8] An Android Activity (Activité Android) is the base component of Android Applications, it is constituted of a graphical layout providing the interface and executing a given code to respond to the user's interactions.

Appendix: Refinement of the need

The Application must (/ must be able to):

0.
- 0.1. Permettre de choisir Online/offline
- 0.2. Permettre de choisir compétiteur/spectateur

1.
- 1.1. Imposer de se connecter pour accéder au menu participant
  - 1.1.1. Permettre de s'identifier auprès du serveur
  - 1.1.2. Empêcher d'accéder au menu « Participant » sans acceptation de la part du serveur
  - 1.1.3. Permettre de choisir la course à suivre

2.
- 2.1. Recevoir les infos relatives à la course
  - 2.1.1. Signaler son existence au serveur
  - 2.1.2. Disposer d'un moyen de connexion au serveur
  - 2.1.3. Pouvoir dépaqueter les informations envoyées par le serveur
- 2.2. Envoyer les informations au serveur
  - 2.2.1. Disposer d'un moyen de connexion au serveur
  - 2.2.2. Connaître l'adresse du serveur
  - 2.2.3. Disposer d'un moyen d'empaqueter les données telles que dépaquetées par le serveur
  - 2.2.4. Acquérir les données
- 2.3. (Idem 2.2.)
- 2.4. Proposer de récupérer les données sur le serveur
  - 2.4.1. Connaître l'adresse du serveur
  - 2.4.2. Pouvoir demander les données utilisateur inhérentes à une course
  - 2.4.3. Pouvoir enregistrer les données de façon réutilisable

3.
- 3.1. Afficher la progression, le parcours, les PDIs (compétiteur)
  - 3.1.1. Afficher une carte
  - 3.1.2. Récupérer les cartes (« tuiles »)
  - 3.1.3. Posséder l'information à afficher
  - 3.1.4. Proposer un code couleur spécifique parcours/progression/PDIs
  - 3.1.5. Permettre à l'utilisateur de comprendre le code couleur
- 3.2. (idem 3.1.) afficher la zone de l'épreuve/les informations (spectateur)
- 3.3. (idem 3.1.) afficher la progression de 1/n/tous les joueurs (spectateur)
- 3.4. (idem 3.1.) afficher la zone de l'épreuve/son parcours/les données publiques (rejeu)
- 3.5. Proposer à l'utilisateur de choisir quoi faire

4.
- 4.1. Servir de dispositif d'alerte (mode panique)
  - 4.1.1. Paramétrer les données à envoyer : quel message ? Quelles informations en plus de « nom, position » ?
  - 4.1.2. Permettre d'envoyer ces données en urgence
  - 4.1.3. Empêcher le déclenchement intempestif du mode panique
- 4.2. Permettre d'envoyer un encouragement
  - 4.2.1. Proposer d'envoyer un encouragement à un joueur dont on regarde le profil
  - 4.2.2. Disposer d'un moyen d'envoi des données au serveur (idem 2.2 sauf 2.2.4)

5.
- 5.1. Rechercher un participant : nom, numéro, …
  - 5.1.1. Permettre d'entrer des données
  - 5.1.2. Récupérer depuis le serveur les données à afficher
  - 5.1.3. Afficher les données récupérées

6.
- 6.1. Proposer un raccourci à Instagram sans perdre l'exécution de l'Application
  - 6.1.1. Proposer un raccourci ergonomique à Instagram
  - 6.1.2. Sauvegarder le contexte de l'Application

The light Application must :

1.2. (idem 1.1.2.)

2.5. (idem 2.2.)
2.6. (idem 2.2.4.)
2.7. (idem 2.4.3.)
2.8.
- 2.8.1. Permettre à l'utilisateur de choisir quelles données récupérer de la BDD
- 2.8.2. (idem 2.2.)
- 2.8.3. Pouvoir écrire les données choisies dans un fichier

3.6.
- 3.6.1. Proposer à l'utilisateur de basculer l'acquisition/envoi des données en tâche de fond
- 3.6.2. Afficher en permanence l'état de la tâche de fond (activée/éteinte)

5.2. (idem 5.1.1.)

Appendix: Attribution of technical solutions to needs

Hereinafter are presented different ways to fulfill the required functions:

For SWARMAPP "heavy" Application:

0.
- 0.1. Premier menu | « Toggle Button » Android[9]
- 0.2. Deuxième menu | deux boutons

1.
- 1.1.
  - 1.1.1. Activité d'authentification nécessitant un couple {identifiant, mot de passe}
  - 1.1.2. Conditionner le changement d'activité par la réponse du serveur
  - 1.1.3. Menu déroulant de l'activité d'authentification

2.
- 2.1.
  - 2.1.1. Requête http | ouverture d'un socket | communication Push
  - 2.1.2. Réseau téléphonique mobile | Wi-fi
  - 2.1.3. Données au format .json
- 2.2.
  - 2.2.1. Idem 2.1.2.
  - 2.2.2. Demander à l'utilisateur | écrire l'adresse en variable | consulter une liste de serveurs d'évènements donnant l'adresse associée
  - 2.2.3. Format .json
  - 2.2.4. API Google Services pour la localisation | API Android pour la localisation
- 2.3.
- 2.4.
  - 2.4.1. Idem 2.2.2.
  - 2.4.2. Requête HTTP
  - 2.4.3. Fichier | Base de données

3.
- 3.1.
  - 3.1.1. API Google Maps | … → cf. Status Report
  - 3.1.2. Serveur associé à 3.1.1.
  - 3.1.3. Idem 2.4.3.
  - 3.1.4. Code couleur
  - 3.1.5. Bouton « légende » menant à une activité « mode d'emploi »
- 3.2. (idem 3.1.)
- 3.3. (idem 3.1.)
- 3.4. (idem 3.1.)
- 3.5. Activité « Menu »

[9] Android Toggle buttons are buttons which offers two states: checked / not checked, they behave as we could expect from electrical switches to behave. They are the most intuitive way to offer the user a way to put on/put off a functionality.

4.
    4.1.
        4.1.1. Activité de paramétrage du bouton panique
        4.1.2. Bouton d'envoi rapidement accessible
        4.1.3. Vibrations + compte à rebours avant envoi du message d'urgence | demande de confirmation d'envoi du message | motif à dessiner pour annuler l'envoi

    4.2.
        4.2.1. Bouton « encourager » sur le profil
        4.2.2. Requête HTTP

5.
    5.1.
        5.1.1. Champ de texte dans l'activité « rechercher un compétiteur »
        5.1.2. Requête HTTP
        5.1.3. Activité d'affichage du profil du compétiteur

6.
    6.1.
        6.1.1. Bouton ergonomique
        6.1.2. Fichier | Bundle Android[10]

For SWARMAPP "light" Application:

1.2. (idem 1.1.2.)

2.5. (idem 2.2.)
2.6. (idem 2.2.4.)
2.7. (idem 2.4.3.)
2.8.
    2.8.1. Menu déroulant, cases à cocher
    2.8.2. (idem 2.2.)
    2.8.3. « FileWriter » Java[11]

3.6.
    3.6.1. « Toggle Button » Android
    3.6.2. « Notification » Android

5.2. (idem 5.1.1.)

[10] An Android Bundle is a map of <Key, Value> sets used to transfer data from an Application to another or to save and restore the state of an Activity when it is paused.
[11] Java's FileWriter is a convenient tool for writing data into files, for example to dump data from a server into a ".txt" file.

Appendix: Good programming practices

During all the project I applied myself on putting in practice good programming practices. Those one make it possible to understand the code more easily and to make corrections or evolutions if necessary.

Among these good practices are:

- Never freezing the User Interface Thread, which would result in a poor User Experience and a possible premature kill of the Application by Android's task manager.
- Using a "Const" static, final class, accessible by every other class. For example the name of the "shared preferences" file is kept in Const.PREFERENCES; error messages displayed are stored in the Const class, making it possible to use generic and specific error messages and to correct all of them in once, etc. … Almost no string is hard-coded.
- Using a Direct Access Object (DAO) as an intermediary to proceed to database operations.
- Using an object to represent the content of a database table (one object per table, i.e. one class "Position", one class "Evènement" …).
- Using a "singleton pattern" where and when possible. For instance, the service which updates the device's location on the server for Light Tracker respects this pattern: the service is obtained using the *getInstance* method and an error is thrown if another service of the same kind is already running because this situation must never occur.
- Respecting Android recommendations, in particular by refusing the use of deprecated classes or methods, and by trying to use the latest library (example: not use HttpClient but HttpURLConnection considering that HttpClient will probably not be supported for long).
- Using abstract classes and interfaces when possible and relevant. For example, all DAOs are extending BaseDAO (except DAOEvenement because it would not have been relevant).