Université d'été - Rapport

Ibanez Thomas, Lovino Maxime, Tournier Vincent

16 septembre 2016

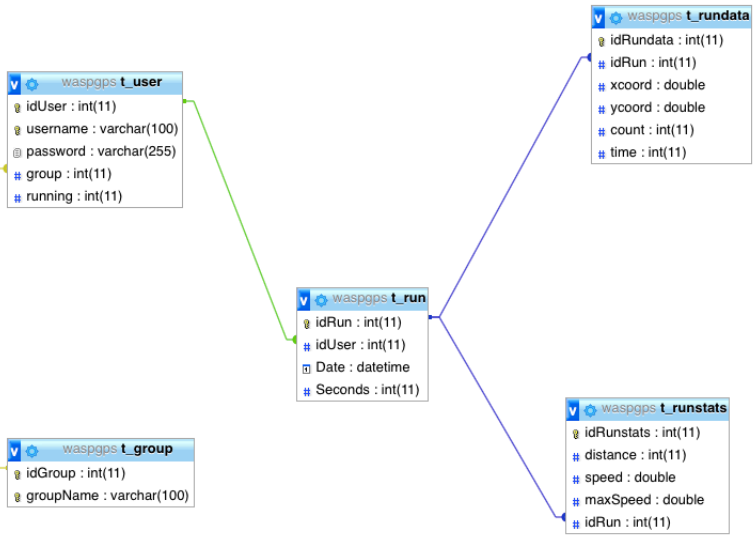
1 Introduction

WaspRunner est un projet miniature et compact de traceur GPS pour joggeurs soucieux de travailler leur VMA.
WaspRunner se compose de trois éléments : un module traceur à emporter avec soi durant la course (fort heureusement le module est léger et compact - il peut facilement être transporté dans une valise à roulette), un serveur web sur Raspberry Pi, ainsi que d'une application Android.
Le traceur est lui-même fait d'une carte waspmote équipée d'un module GPS et GPRS, d'une carte wifi dont la simplicité n'a d'égal que la fiabilité, d'une batterie faisant aussi office de lest pour que le coureur ne dépasse pas les limitations de vitesses du code de la route, ainsi que d'une carte microSD pour sauvegarder les courses.
Le serveur web récupère les données GPS du traceur, les traite et les stocke. Il fournit également la possibilité de consulter les rapports de courses sur Internet.
L'application Android permet quant à elle de recevoir les rapports de course directement sur son smartphone.

2 Réalisation

2.1 Base de données

La base de données se présente selon le schéma suivant :



2.2 Récupération et envoi des données GPS

La première étape pour notre application est de récupérer la position actuelle via GPS, le module GPRS+GPS de la Waspmote s'occupe de nous donner la position en degrés [-180 ;180].
Nous appliquons ensuite un code correcteur d'erreurs afin d'éviter qu'un problème de communication n'entraine l'insertion de fausses données dans la course.
Une fois la position définie, la waspmote, si une carte SIM et une connexion sont disponibles, va envoyer une requête GET sur l'application web (page run.php), en définissant des flags dans l'url qui seront interprétés par le code php.

- uid : l'identifiant de l'utilisateur courant, (par défaut 1)
- start : doit être défini en cas de début de course
- x : longitude (en degrés)
- y : latitude (en degrés)
- cnt : numéro du point envoyé
- time : temps (en secondes) écoulé depuis le début de la course
- end : envoyé en fin de course

Voici un exemple de communication entre la waspmote et l'application web

- /run.php ?uid=1&start
- /run.php ?uid=1&x=<longitude>&y=<latitude>&time=12&cnt=1

— /run.php ?uid=1&x=<longitude>&y=<latitude>&time=18&cnt=2
— /run.php ?uid=1&x=<longitude>&y=<latitude>&time=23&cnt=3
— ...
— /run.php ?uid=1&time=2704&end

2.3 Traitement des données

L'application web s'occupe du lien avec la base de données, au départ de la course un fannion running dans la ligne de l'utilisateur est défini à l'id de la course en cours.
Ensuite à chaque fois qu'un point est envoyé, les données du point (x,y,cnt et time) sont ajoutés dans la table t_rundata.
Lorsque l'application reçoit le fannion end, la case running de l'utilisateur est remise à 0 et les statistiques de course sont calculées. Le calcul de la distance au sol est fait par la formule d'Haversine.

$$a = sin^2(\frac{\varphi_1 - \varphi_2}{2}) + cos(\varphi_1) * cos(\varphi_2) * sin^2(\frac{\lambda_1 - \lambda_2}{2})$$

$$d = 2r * arctan\left(\frac{\sqrt{a}}{\sqrt{1-a}}\right)$$

Où
φ_1 *et* φ_2 sont les longitudes des deux points
λ_1 *et* λ_2 sont les latitudes des deux points
r est le rayon de la terre (6371 Km)

2.4 Stockage des données en cas de problème de réseau

Dans le cas où le module est incapable, pour une raison ou une autre, de se connecter au réseau GPRS avant qu'une course ne soit démarrée, il nous a fallu trouver un moyen de ne pas rendre le module inutilisable. Pour cela, nous avons fait le choix de permettre à l'utilisateur de sauvegarder les données de la course sur une carte SD.
Dans le cas où le réseau GPRS est inaccessible au moment où l'on doit démarrer une course, les données GPS et tous les points sont sauvegardés sur un fichier texte présent dans la carte SD, sous la forme de lignes de commandes. La course se déroule alors normalement, chaque point étant sauvegardé sur la carte SD.
Lorsque la course est stoppée, le fichier est rendu disponible à l'envoi. Immédiatement, la carte WiFly s'active et essaye de se connecter au réseau WiFi sauvegardé dans sa configuration. De la même manière, si la carte s'allume et qu'un fichier de course est présent en mémoire, elle essayera d'abord d'envoyer le fichier avant de laisser l'utilisateur démarrer une course.
Sitôt que la carte parvient à se connecter à un réseau WiFi, elle ouvre

une connexion vers un socket java présent sur le serveur web. Lorsque la connexion TCP est établie, la carte envoie ligne par ligne les données présentes sur le fichier, puis se déconnecte et, si les données ont été correctement envoyée, détruit le fichier. Une nouvelle course peut alors être lancée.
Dans le cas où la carte est incapable d'obtenir une connexion WiFi ou TCP, et ce afin d'éviter un comportement bloquant pour l'utilisateur, un timeout a été mis en place. Globalement, une tentative de connexion prend 5 secondes, aussi nous avons limité le nombre de tentatives à 12 afin que la carte essaye de se connecter pendant environ une minute avant d'abandonner et de laisser l'utilisateur démarrer, s'il le souhaite, une nouvelle course. Par ailleurs, si cette nouvelle course doit être enregistrée sur la carte SD, elle sera simplement ajoutée à la suite de celle déjà présente.

2.5 Application Android

Une application Android a également été réalisée, il s'agit de WaspDroid.

2.5.1 Recupération des données et stockage

Pour récupérer les données de course depuis le serveur, un hook php dédié a été mis en place, en effet :
`/android.php?uid=1&listrun`
permet de récupérer la liste des courses dans un format CSV de la forme `runID;date;temps`. Ensuite, nous pouvons utiliser
`/android.php?uid=1&rundata&idRun=X`
pour récupérer la liste des points pour la course X. Ses données sont en CSV de la forme `x;y;count;temps`

Les données sont stockées dans une base de données locale SQLite qui sert de cache en cas de perte de connexion de l'appareil et permet de rendre l'application plus rapidement fonctionnelle, étant donnée que les données sont déjà chargées sont devoir envoyer des requêtes au serveur. Les classes `RunDBContract` et `RunDBHelper` s'occupent de cette base de données. La classe `PHPConnector` prend en charge l'intéraction avec notre serveur, elle hérite de `AsyncTask` pour lui permettre de charger les données dans un thread séparé. (obligatoire sur Android)

Dans les réglages de l'application, il est possible de définir l'URL et le port du serveur à utiliser (par défaut sampang.internet-box.ch et port 8080) ainsi que de remettre à zéro la base de données de cache.

2.5.2 Interface graphique et API Maps

L'interface graphique de l'application est assez simple, nous avons deux activités principales. `MainActivity` qui affiche la liste des différentes courses et qui lance `DetailView` lorsqu'on clique sur l'une d'entre elles.

Dans la vue détaillée, nous affichons les différentes statistiques de la course, ainsi qu'un fragment Google Maps avec le parcours de celle-ci. L'application demande la permission à l'utilisateur à ce moment-là pour utiliser sa localisation et pouvoir l'afficher sur la carte également.

2.5.3 Classes implémentées

Pour gérer les courses, deux classes ont été implémentées, il s'agit de `DataPoint` et de `Run`.

La classe `DataPoint` désigne un point de notre parcours, c'est-à-dire une position GPS (classe `GPSCoordinates` que nous avons réalisé) ainsi que le "count" qui est un numéro de séquence qui nous permet d'identifier l'ordre des points et la valeur du chronomètre lors de l'envoi du point.

La classe `Run` elle, comprend une list de `DataPoint` représentant ses points ainsi que des attributs d'instance pour sa date, son ID, l'ID de l'user et le temps total de la course.

`GPSCoordinates` est une petite classe qui désigne la latitude et longitude d'un point ainsi qu'une méthode pour calculer la distance de l'instance à une autre coordonnée (formule d'Haversine, voir plus haut)