Université De Montpellier
Faculté Des Sciences

Niveau : Master 2

Module : Développement mobile avancé, IoT et embarqué

HAI912I

Implémentation d'un programme pour ESP32 permettant de recommander les parkings de Montpellier 3M les plus proches

---

Supervisé par :
M. Alban Mancheron
M. Yassine Haddab

Réalisé par :
Kaci Ahmed
Allouch Yanis

2021/2022

Table des matières

1 Introduction

De nos jours, l'internet des objets est une technologie ayant une importance considérable dans la vie quotidienne de l'homme du 21ème siècle. En effet, la communication en toute fluidité entre les personnes, les processus et même les objets est devenu possible grâce au simple fait de pouvoir connecter les produits du quotidien (tels que les appareils électroménagers, les voitures, les thermostats, etc.) à Internet par l'intermédiaire de terminaux intégrés.

Dans ce rapport, nous allons décrire les différentes étapes de la démarche que nous avons suivies afin d'implémenter un programme pour ESP32 permettant à partir de données de géolocalisation, de trouver les parkings de Montpellier 3M les plus proches, ayant des places disponibles. Nous détaillerons également plusieurs aspects liés à la gestion de ce microcontrôleur tels que la gestion des erreurs et l'autonomie. Nous décrirons également les erreurs que nous avons rencontrées durant le développement ainsi que les solutions que nous avons établie pour leur remédier. Enfin, nous terminerons par une conclusion et des perspectives.

2 Présentation de l'approche générale de l'application développé

Pour la réalisation de ce projet, nous avons d'abord étudié un ensemble de code fournit par notre professeur M. *Alban Mancheron*, à savoir un fichier intitulé *parking.cpp* et le fichier *parking.h* qui nous ont servis de support pour construire l'architecture suivante :

2.1 Structures utilisées pour sauvegarder les données utilisées par l'application

Dans ce projet, nous avons implémentés les structures de données suivantes :

— Une liste de parkings dont chaque élément contient l'identifiant, le nom, la longitude et la latitude d'un parking,

— Une liste dont chaque élément contient le nom du parking qu'il représente, le nombre de places libre et la distance entre ce parking et l'emplacement de l'ESP32.

2.2 Démarche suivie

Pour implémenter cette application :

1. En premier lieu, on obtient les coordonnés GPS de l'ESP32,
2. Puis, on récupère la liste des parkings pour calculer la distance entre chaque parking avec la localisation du microcontrôleur. Nous avons utilisé les structures de données définies ci-dessus pour sauvegarder les résultats obtenus,
3. Ensuite, on trie la liste des parkings avec leurs distances par ordre croissant des distances,
4. Enfin, on affiche les résultats obtenus dans la console reliée au microcontrôleur.

2.2.1 Obtention des coordonnées géographique de l'ESP32

Pour réaliser cette partie, on définit une structure qui permet de représenter la longitude et la latitude dont laquelle il faut introduire les informations de la position du microcontrôleur. Pour ce faire, on a défini une fonction pour simuler l'accès a un capteur GPS connecté au microcontrôleur, qui retourne actuellement une position écrite en dur dans le code.

2.2.2 Récupération d'une liste de Parkings

Tout d'abord, on sauvegarde dans le code une liste contenant les identifiants des parkings de Montpellier cette dernière est statique, car elle ne risque pas de changer

régulièrement étant donné la nature des constructions.
Puis, Nous interrogeons l'API présente à l'adresse https://data.montpellier3m.fr/ pour obtenir le status et le nombre de places disponibles dans chaque parking selon son identifiant. En effet, On utilise notre liste d'identifiants pour interroger le serveur et traiter chaque réponse (lecture XML). Et cela, afin de pouvoir sauvegarder le status et le nombre de places disponibles d'un parking.

2.2.3 Calcule de la distance

Nous avons réalisé le calcule de la distance entre la localisation du microcontrôleur et chaque parking récupéré à l'étape précédente avec deux méthodes différentes qui sont les suivantes.

Tous d'abord, dans une version simplifier on utilise un calcul mathématique, pour obtenir une distance linéaire dite distance à vol d'oiseau. Cette mesure a l'avantage d'être rapide et simple à implémenter, vu qu'elle se base sur la formule suivante :

$$\sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$$

Où :

- A est le point de départ dans le plan,
- B est le point d'arrivée dans le plan,
- x_1 représente la longitude du point A,
- x_2 représente la longitude du point B,
- y_1 représente la latitude du point A,
- y_2 représente la latitude du point B.

Par ailleurs, on remarque qu'on peut facilement optimiser le calcule de l'opération arithmétique *puissance* en le remplaçant par une multiplication comme suit :

$$\sqrt{(x_1 - x_2) * (x_1 - x_2) + (y_1 - y_2) * (y_1 - y_2)}$$

Cependant, malgré le fait que cette façon de calculer la distance soit rapide et économe, elle n'est pas satisfaisante. En effet, elle ne reflète pas la distance réelle que l'utilisateur de l'application doit parcourir. Pour remédier à cela, on utilise un vrai système de cartographie reposant sur Open Street Map qui propose un ensemble de services via une API Restful.
En se documentant cet API, nous avons découvert qu'il existe de nombreuses solutions et services en ligne, dont une liste non exhaustive est disponible à cette adresse et cette adresse.
Dans le cadre de ce projet, nous avons opté pour l'utilisation du projet OSRM dont l'API est accessible a l'adresse suivante http://project-osrm.org.

L'avantage d'utiliser ce service est que nous avons des données en temps-réel et aucune configuration de backend à mettre en place. Assurément, la documentation de l'API est accessible à cette adresse.

Par ailleurs, la figure 1 permet de montrer un cas d'exemple d'utilisation de cet API via Postman pour récupérer les données d'un trajet (en voiture) entre deux coordonnées GPS.

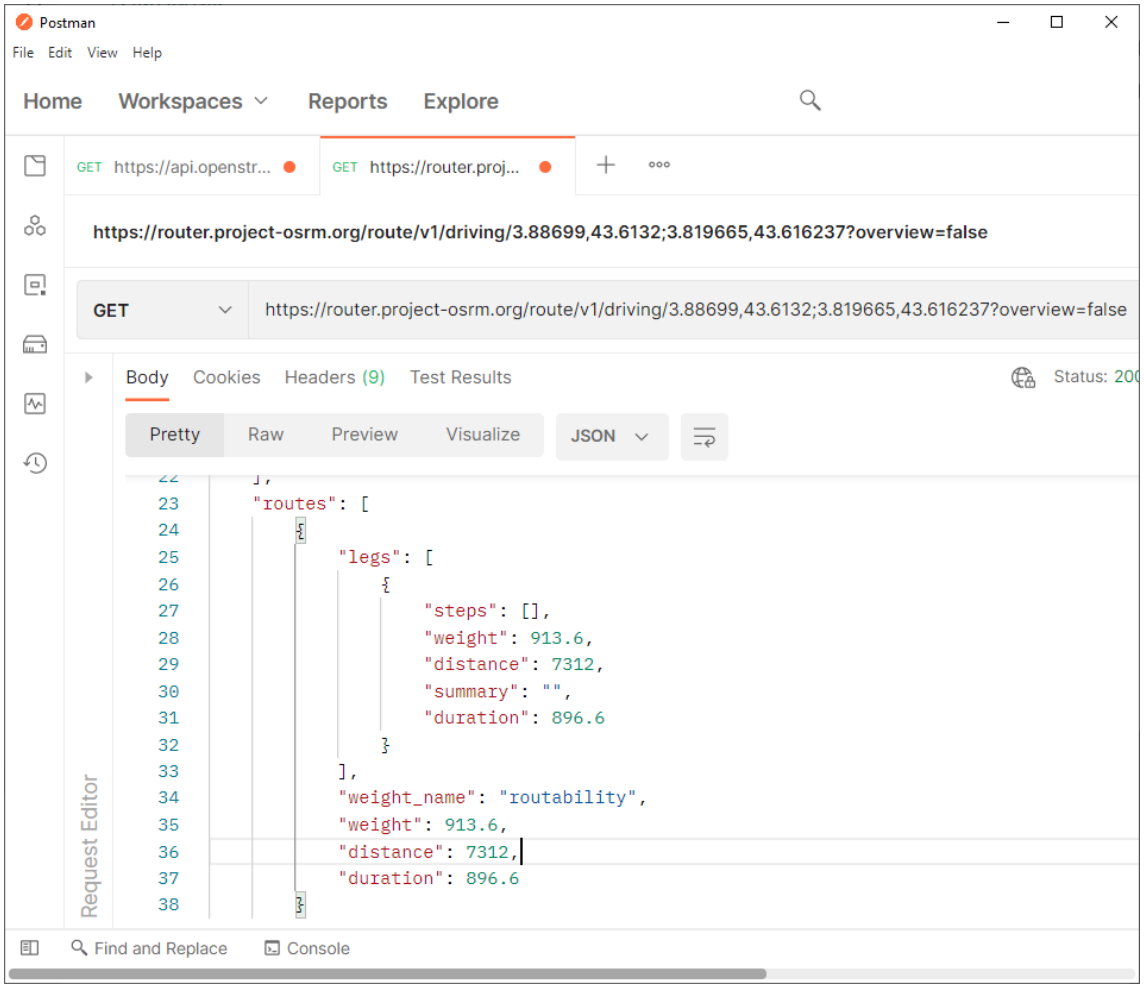

FIGURE 1 – Capture d'écran d'un cas d'exemple d'utilisation de l'API OSRM via Postman

Pour ce faire, dans la deuxième version de notre programme, on change l'implémentation de la fonction calculant la distance pour effectuer un appel HTTP GET vers l'API https://router.project-osrm.org/route/v1/driving/xA,yA;xB,yB?overview=false en remplaçant respectivement les valeurs 'xA', 'xB' par les coordonnées GPS courantes de l'ESP32 et 'yA', 'yB' par les cordonnés GPS d'un parking.

De surcroit, on utilise la libraire ArduinoJson pour désérialiser et manipuler avec aisance le résultat d'une requête JSON.

2.2.4 Trier les résultats

Pour pouvoir trier les résultats calculés jusqu'à présent, on implémente un algorithme de tri sélection basé sur les distances entre chaque parking et la localisation de l'ESP32. Et ce, pour pouvoir proposer à l'utilisateur, la liste des parkings les plus proches de sa position afin de lui permettre de choisir celui qu'il désire. En effet, nous avons discuté avec notre enseignant M. Alban Mancheron de certaines contraintes qui nous motive à employer cette solution. Voici une liste non exhaustive des contraintes abordées :

1. L'utilisateur préfère un parking à un autre par habitude,
2. L'utilisateur possède une voiture plus large que la moyenne, l'empêchant alors de se garer dans tous les parkings de Montpellier,
3. L'utilisateur préfère éviter le trafic en sortie des parkings du centre-ville (Antigone/Polygone).

2.2.5 Affichage des résultats

Pour afficher les résultats de la liste des parkings ordonnés dans l'ordre croissant des distances. On itère sur cette liste et on affiche sur la console le nom de chaque parking, le nombre de places libres qu'il contient et la distance qui le sépare de l'utilisateur.
Nous avons mis en annexe des exemples (Listing 2 et Listing 3) illustrant les affichages obtenus en utilisant les méthodes de calculs décrites précédemment.

3 Aspects liés à la gestion du microcontrôleur

Dans cette section, on va décrire différents aspects liés à l'ESP32 que nous avons traité à savoir : les erreurs et l'autonomie.
Dans notre application, nous avons implémenté une gestion des erreurs de manière a ne pas interrompre la boucle interactive *loop*. Cela, peut être matérialiser grâce aux points suivants :

— Dans le cas où une requête HTTP GET n'aboutit pas pour plusieurs raisons, le parking concerné est alors ignoré du calcul,

— Si le calcul de la distance par le service OSRM ne produit pas de bon résultat, le parking concerné est aussi ignoré du calcul.

De plus, nous avons fait en sorte que les erreurs graves soient automatiquement signalées dans la console.

Concernant l'autonomie, on a mesuré le temps d'exécution de la boucle interactive et on a abouti au fait qu'elle s'exécute en 1 min. Pendant ce temps-là, le WiFi

est activé en permanence. Selon la spécification fournie dans un des cours suivit dans ce module, cela correspondrait à une consommation de 250 mA. Par ailleurs, pour préserver l'autonomie de l'ESP32, nous avons aussi étudié qu'il est indispensable de sélectionner un certain temps de repos pour préserver cet ESP32. Ainsi, nous avons jugé que 10 minutes d'attente avant de relancer la boucle du programme est un temps raisonnable si on doit prendre en compte le temps d'attente moyen dans le trafic routier à Montpellier.

4 Erreurs rencontrées

N'étant pas familier avec l'écosystème Arduino, on va présenter dans cette section les erreurs rencontrées lors du développement ainsi que les solutions que nous avons mis en places dans ce projet.

La première erreur qu'on a rencontrée, relève du fait que nous sommes des développeurs avec une expérience C/C++ limités et de total débutant en Arduino. En effet, on apprend dans cette documentation que l'environnement Arduino possède ça propre classe *String*. Cette simple erreur de type, nous a coutés quelques heures de débogage et de recherche pour comprendre que nous avions affaire à une classe non implémentée dans la librairie standard C/C++ mais uniquement dans Arduino. Nous avons rencontrés cette erreur avec le client HTTPS de la libraire Arduino pour ESP32 *HTTPClient*, qui manipule des valeurs littérales encapsulé dans la classe String. Pour corriger cette erreur, on a implémenté la fonction de conversion string vers String suivante (Listing 1) :

```
/**
 * Convertit une string standard vers la classe String defini dans
   Arduino.h
 */
String convertToStringArduino(string aStr){
  String result;
  for(int i = 0; i < aStr.size(); i++){
    result += aStr.at(i);
  }
  return result;
}
```

Listing 1 – Fonction de conversion de string vers String écrite en C++

Nous avons rencontré une autre erreur au cours du développement de la version élaborer du calcul des distances. Cette erreur est en relation avec le protocole de communication utilisé. En effet, nous nous sommes rendus compte que si on ne faisait pas la requête sur le bon protocole (HTTP/HTTPS) sur le service OSRM, alors la requête n'aboutissait pas. On corrige cette erreur, en employant le protocole de communication HTTP. Puisque le serveur ne disposent pas de certificat SSL.

5 Conclusion

Dans le cadre de ce projet, nous avons implémenté un programme pour ESP32 permettant à partir de données de géolocalisation, de trouver les parkings de Montpellier 3M les plus proches, ayant des places disponibles.

Pour cela, nous avons défini des structures de données pour contenir les données manipulées durant l'exécution de cette application et implémentées grâce à une démarche dans laquelle on obtient les coordonnées GPS de l'ESP32 ; on récupère la liste des parkings pour calculer la distance entre chaque parking avec la localisation du microcontrôleur ; on trie la liste des parkings avec leurs distances par ordre croissant des distances. Enfin, on affiche les résultats obtenus dans la console reliée au microcontrôleur.

Nous avons aussi traité plusieurs aspects liés à la gestion de notre microcontrôleur, tels que la gestion des erreurs et l'autonomie, en plus d'avoir présenté dans ce rapport les différentes erreurs rencontrées lors du développement.

Pour terminer, ce projet nous a permis de découvrir ce qu'est l'internet des objets d'un point de vue pratique et d'améliorer nos connaissances concernant le protocole de communication HTTP et l'utilisation d'API RESTful.

6 Perspectives

Nous proposons les perspectives d'évolution de notre programme suivantes :

- Ajouter un serveur web sur le microcontrôleur pour l'augmenter avec les capacités suivantes :
  - Affichage des résultats en générant des pages webs,
  - Récupération d'une entrée utilisateur grâce à un formulaire HTML et les transmettre par une route GET/POST.
- Récupérer la position GPS courante de l'ESP32 soit :
  - via un capteur GPS connecté par les ports GPIO,
  - via une API accessible depuis le serveur WEB.
- Améliorer le backend en l'hébergeant dans un serveur doté d'un certificat SSL pour protégés les communications.

Nous aurions aimé pouvoir approfondir l'aspect des mises à jour OTA proposé par le framework Espressif brièvement abordé en cours ainsi que la gestion de la sécurité. Cependant, nous n'avons, malheureusement, pas pu le faire par manque de temps.

7 Annexe

Output obtenu en exécutant le projet calculant les distances linéaire (à vols d'oiseaux) :

```
14:55:18.199 -> Waiting for WiFi to connect...WiFi connected
14:55:21.342 -> Recuperation des coordonnees GPS...
14:55:21.342 -> Longitude : 3.88699, Latitude : 43.6132
14:55:21.342 -> Recuperation des donnees depuis Open-Data 3M...
14:55:50.339 -> Sort des parkings par distance
14:55:50.339 -> Affichage des parkings :
14:55:50.339 -> Parking 'Corum' (FreePlace : 158) Distance
    0.00477575
14:55:50.339 -> Parking 'Polygone' (FreePlace : 1448) Distance
    0.00532935
14:55:50.339 -> Parking 'Triangle' (FreePlace : 277) Distance
    0.0065037
14:55:50.339 -> Parking 'Europa' (FreePlace : 406) Distance
    0.00771693
14:55:50.339 -> Parking 'Comedie' (FreePlace : 36) Distance
    0.0085934
14:55:50.339 -> Parking 'Foch Prefecture' (FreePlace : 45) Distance
     0.0107035
14:55:50.339 -> Parking 'Saint Roch' (FreePlace : 337) Distance
    0.013026
14:55:50.339 -> Parking 'Arc de Triomphe' (FreePlace : 40) Distance
     0.0139618
14:55:50.386 -> Parking 'Gambetta' (FreePlace : 247) Distance
    0.0168214
14:55:50.386 -> Parking 'Pitot' (FreePlace : 344) Distance
    0.0168231
14:55:50.386 -> Parking 'Charles de Gaulle' (FreePlace : 111)
    Distance 0.0187364
14:55:50.386 -> Parking 'Arceaux' (FreePlace : 207) Distance
    0.0195528
14:55:50.386 -> Parking 'Garcia Lorca' (FreePlace : 289) Distance
    0.0225428
14:55:50.386 -> Parking 'Multiplexe (ouest)' (FreePlace : 113)
    Distance 0.0285713
14:55:50.386 -> Parking 'Circe Odysseum' (FreePlace : 585) Distance
     0.0319504
14:55:50.386 -> Parking 'Multiplexe (est)' (FreePlace : 103)
    Distance 0.033017
14:55:50.386 -> Parking 'Sabines' (FreePlace : 210) Distance
    0.0397446
14:55:50.386 -> Parking 'Occitanie' (FreePlace : 341) Distance
    0.0439242
14:55:50.386 -> Parking 'Euromedecine' (FreePlace : 151) Distance
    0.06461
14:55:50.386 -> Parking 'Saint-Jean-le-Sec' (FreePlace : 186)
    Distance 0.0648346
14:55:50.386 -> Parking 'Mosson' (FreePlace : 115) Distance
    0.0673883
```

```
14:55:50.386 ->
14:55:50.386 ->
14:55:50.386 -> Waiting 10min before the next round...
```

Listing 2 – Output du projet calculant les distances a vol d'oiseau

Output obtenu en exécutant le projet calculant les distances réels grâce à l'API OSRM :

```
18:16:15.327 -> En attente de la connexion au WiFi...WiFi connecte
18:16:18.965 -> Recuperation des coordonnees GPS...
18:16:18.965 -> Longitude : 3.88699, Latitude : 43.6132
18:16:18.965 -> Recuperation des donnees depuis Open-Data 3M et
   OSRM...
18:17:20.831 -> Trie des parkings par distance
18:17:20.831 -> Affichage des parkings :
18:17:20.831 -> Parking 'Europa' (FreePlace : 431) Distance 1104.4
18:17:20.831 -> Parking 'Corum' (FreePlace : 149) Distance 1151.7
18:17:20.831 -> Parking 'Antigone' (FreePlace : 124) Distance
   1312.1
18:17:20.831 -> Parking 'Polygone' (FreePlace : 1448) Distance
   1312.1
18:17:20.831 -> Parking 'Triangle' (FreePlace : 277) Distance
   1626.4
18:17:20.831 -> Parking 'Saint Roch' (FreePlace : 348) Distance
   2368.4
18:17:20.831 -> Parking 'Charles de Gaulle' (FreePlace : 104)
   Distance 3086.5
18:17:20.831 -> Parking 'Garcia Lorca' (FreePlace : 289) Distance
   3087
18:17:20.831 -> Parking 'Pitot' (FreePlace : 343) Distance 3124.1
18:17:20.878 -> Parking 'Arceaux' (FreePlace : 207) Distance 3167.7
18:17:20.878 -> Parking 'Arc de Triomphe' (FreePlace : 96) Distance
    3185.3
18:17:20.878 -> Parking 'Multiplexe (ouest)' (FreePlace : 51)
   Distance 3238.8
18:17:20.878 -> Parking 'Circe Odysseum' (FreePlace : 585) Distance
    3395.7
18:17:20.878 -> Parking 'Foch Prefecture' (FreePlace : 105)
   Distance 3458.7
18:17:20.878 -> Parking 'Multiplexe (est)' (FreePlace : 88)
   Distance 3491.5
18:17:20.878 -> Parking 'Gambetta' (FreePlace : 200) Distance
   3935.5
18:17:20.878 -> Parking 'Comedie' (FreePlace : 156) Distance 4140
18:17:20.878 -> Parking 'Occitanie' (FreePlace : 391) Distance 5560
18:17:20.878 -> Parking 'Sabines' (FreePlace : 216) Distance 7035.5
18:17:20.878 -> Parking 'Mosson' (FreePlace : 140) Distance 7308.2
18:17:20.878 -> Parking 'Euromedecine' (FreePlace : 157) Distance
   8330.9
18:17:20.878 -> Parking 'Saint-Jean-le-Sec' (FreePlace : 201)
   Distance 14038.7
18:17:20.878 ->
18:17:20.878 ->
```

```
18:17:20.878 -> Waiting 10min before the next round...
```

Listing 3 – Output du projet calculant les distances en utilisant OSRM