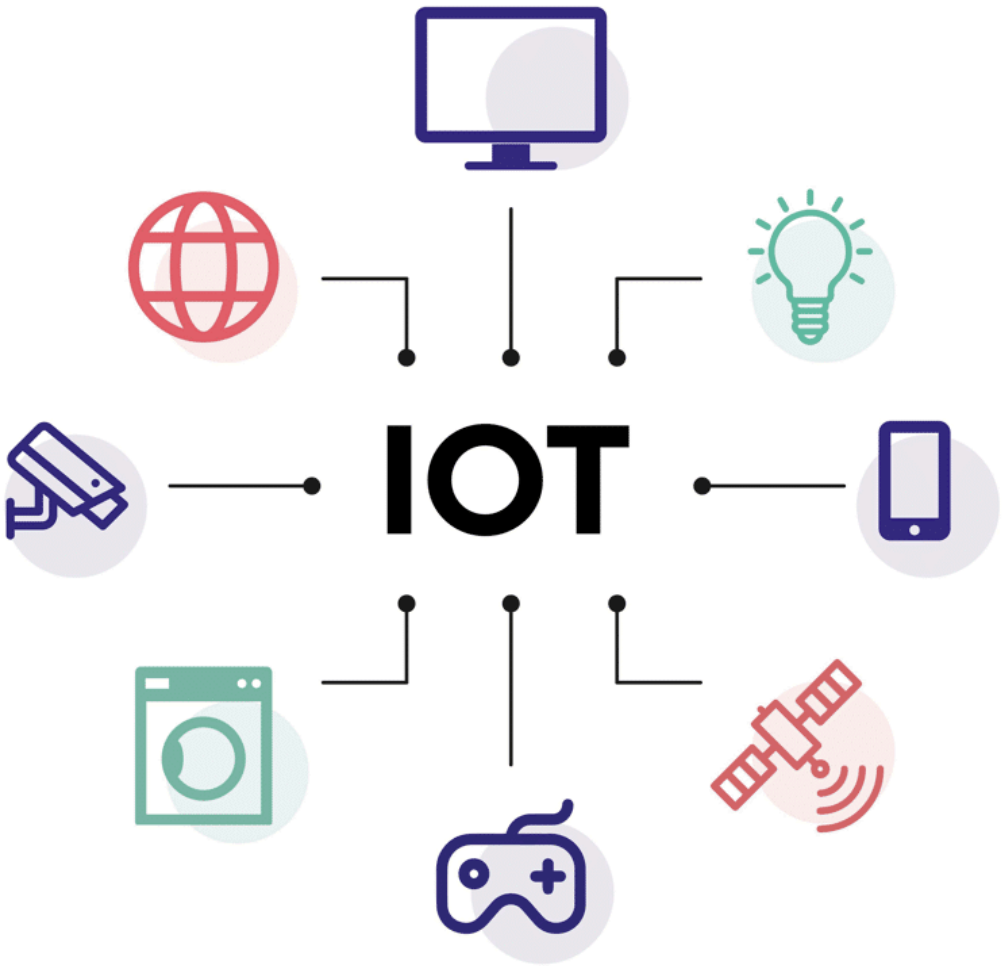
YANG Johan

Rapport De Projet



Détection WiFi

L’objectif est de scanner les réseaux WiFi environnants pour en extraire les adresses MAC. Ces informations serviront à la géolocalisation via la base de données. Cependant, pour garantir la précision de la localisation, le système doit impérativement distinguer les points d'accès fixes (Box Internet, routeurs) des points d'accès mobiles (Partage de connexion etc …).

C’est pourquoi, il y a un algorithme de filtrage qui a été établie avant l’enregistrement dans la base de données.

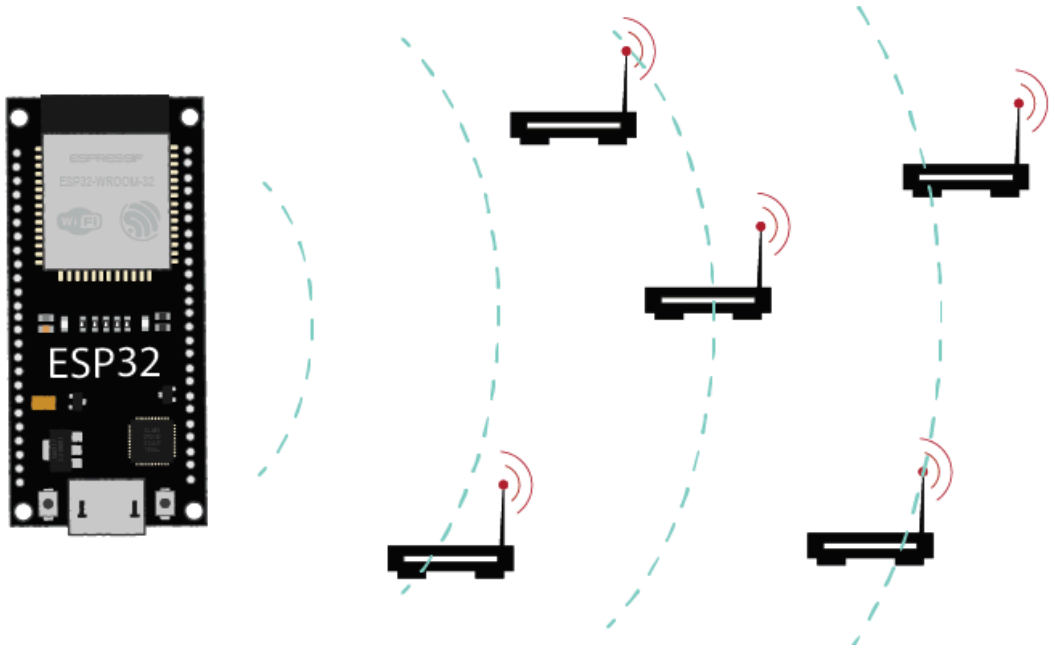
L’algorithme repose sur deux éléments principales :

- L’adresse MAC
- Le nom du réseau WiFi (SSID)

En regardant le deuxième quartet du premier octet de l'adresse MAC. Si celui-ci correspond aux valeurs 2, 6, A ou E, il s'agit d'une adresse administrée localement (MAC aléatoire), caractéristique typique des smartphones modernes pour protéger la vie privée. Ainsi, on peut donc ignorer ce point WiFi et ne pas l’ajouter dans la base de données.

On fait la même chose pour les noms du réseau WiFi, la plupart des réseaux mobiles commencent par “iPhone …”, “Galaxy …” , “Android …” etc … si l’on n’a pas changé ce nom manuellement. On peut alors les ignorer dans ce cas aussi.

Pour ce qui est de la détection WiFi, on utilise le scanneur de base de l’ESP32. En effet, l'ESP32 contient un mode **station passive** qui va détecter tous les réseaux aux alentours.



LoRaWAN

Après avoir scanné et filtré les différents points d'accès WiFi, on envoie via les données sur TTN via le module LoRaWAN. Toutefois, le réseau LoRaWAN ayant une bande passante limitée, les données ne sont pas envoyées en texte clair mais sous forme de trame binaire optimisée.

Le format étant le suivant :

Octet	**Taille**	**Description**
0	1 octet	Marqueur de début (Header : 0x55)
1-2	2 octets	Timestamp relatif (Secondes depuis démarrage)
3	1 octet	Nombre de points d'accès détectés (N)
4-9	6 octets	Adresse MAC du point d'accès 1
10	1 octet	Puissance du signal (RSSI)
11	1 octet	Canal WiFi
...	...	(Répétition pour les N suivants, max 3)

Le programme suit une machine à états simple pour assurer la connexion :

- Initialisation : Configuration LoRa et mise du WiFi en mode station.

- Gestion de la connexion: Tant que le module n'est pas enregistré sur le réseau LoRaWAN, il tente une procédure de Join toutes les 15 secondes.

- Scan et Transmission : Une fois connecté, un scan est déclenché toutes les 60 secondes. Les réseaux filtrés sont triés, puis les 3 meilleurs (les plus stables/fixes) sont encapsulés dans le payload et envoyés via la commande AT+MSGHEX.

Ensuite, dès que les données sont arrivées sur TTN, on utilise un algorithme en javascript afin de décoder la trame envoyée en hexadécimale.

```javascript
function decodeUplink(input) {
  var bytes = input.bytes;
  var data = {};
  var i = 0;

  // 1. Décodage de la Location
  data.location_char = String.fromCharCode(bytes[i++]);

  // 2. Timestamp (2 octets)
  var timeVal = (bytes[i++] << 8) | bytes[i++];
  data.device_time = timeVal;

  // 3. Nombre d'AP détectés
  var apCount = bytes[i++];
  data.ap_count = apCount;

  data.access_points = [];

  // 4. Boucle pour récupérer les APs
  // ATTENTION : On lit maintenant 6 octets pour la MAC + 1 RSSI + 1 Channel = 8 octets par AP
  for (var k = 0; k < apCount; k++) {
    if (i + 7 > bytes.length) break; // Sécurité

    // Lecture de la MAC COMPLÈTE (6 octets)
    var mac = "";
    for (var j = 0; j < 6; j++) {
        var b = bytes[i++];
        if (j > 0) mac += ":";
        mac += ("00" + b.toString(16).toUpperCase()).slice(-2);
    }

    // RSSI (int8 signé)
    var rssiRaw = bytes[i++];
    var rssi = (rssiRaw > 127) ? rssiRaw - 256 : rssiRaw;

    // Channel
    var channel = bytes[i++];

    data.access_points.push({
      mac: mac,
      rssi: rssi,
      channel: channel
    });

    // Pour Node-RED (facultatif, juste le premier AP)
    if (k === 0) {
        data.AP1_MAC = mac;
        data.AP1_RSSI = rssi;
    }
  }
  return {
    data: data,
    warnings: [],
    errors: []
  };
}
```

Node-Red

Le flux est divisé en quatre étapes majeures : la réception MQTT, le décodage binaire, le stockage en base de données et la géolocalisation via l'API WiGLE.

Les données arrivent de TNN via le protocole MQTT. Le payload reçu est encodé et doit donc être décodé afin d'être exploitable.

Nœud "Decode LoRa" : Ce nœud contient une fonction JavaScript qui réalise l'opération inverse du programme de l'ESP32. Il extrait le marqueur d'en-tête, le timestamp, et boucle sur les octets restants pour reconstruire les adresses MAC et convertir les valeurs de RSSI en entiers signés.

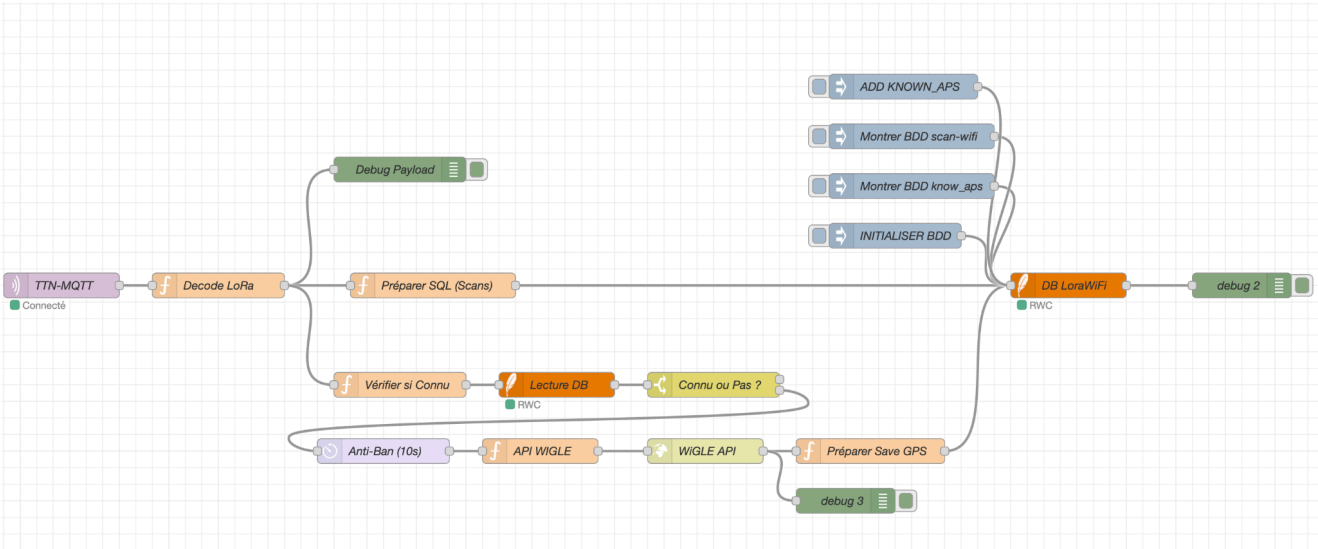
Pour assurer un historique des mesures, nous utilisons une base de données **SQLite**. Le traitement est séparé en deux flux :

- **Historique des scans :** Chaque point d'accès détecté est enregistré dans la table wifi_scans avec son adresse MAC, son RSSI et son canal. Cela permet de garder une trace de l'activité du capteur.
- **Base des points connus :** La table known_aps stock les coordonnées géographiques (Latitude/Longitude) associées à chaque adresse MAC.

L'API Wigle va ensuite nous servir afin de localiser les points WiFi dans la base de données wifi_scans et donc ensuite de pouvoir les mettres dans known_aps.

On a alors les étapes suivantes :

- **Vérification en base :** Pour chaque MAC reçue, le système interroge d'abord la base de données locale (Vérifier si Connu). Si l'adresse est déjà connue, on évite une requête API inutile.
- **Requête WiGLE :** Si la MAC est inconnue, le flux interroge l'API de **WiGLE**.
- **Anti-Ban & Optimisation :** Un nœud "Delay" (limité à 1 requête toutes les 10 secondes) est configuré pour respecter les quotas de l'API et éviter le bannissement de la clé API.
- **Ajout :** Une fois les coordonnées récupérées (Latitude/Longitude), elles sont enregistrées dans la table known_aps pour que la prochaine détection de ce même routeur soit instantanée.



FastAPI

Pour traiter les données stockées en base et les présenter à l'utilisateur, nous avons développé une API REST légère avec le framework Python **FastAPI**. Ce serveur agit comme une couche d'abstraction entre la base de données SQLite et l'interface web. Il expose trois endpoints principaux :

- GET /api/scans : Récupère l'historique brut des réseaux détectés.
- GET /api/ap_connus : Renvoie la liste des points d'accès géolocalisés.
- GET /api/position_estimee : Exécute l'algorithme de calcul de position en temps réel.

Avant de pouvoir trianguler, nous devons estimer la distance entre le capteur (ESP32) et chaque point d'accès WiFi détecté.

La fonction rssi_to_distance implémente la formule suivante :

$$d = 10^{A-RSSI/10*n}$$

Où :

- **d** : Distance estimée (en mètres).
- **RSSI** : Puissance du signal reçu (ex: -75 dBm).
- **A** : Référence de puissance à 1 mètre (fixée empiriquement à **-45 dBm**).
- **n** : Indice d'atténuation du signal, fixé à **3.5** pour un environnement urbain/intérieur dense.

Une fois les distances estimées pour les bornes détectées dans les 60 dernières secondes, le serveur calcule la position de l'objet. Nous utilisons la méthode du Barycentre Pondéré. Chaque borne connue "tire" la position estimée vers elle. La force de cette attraction (le poids w) est inversement proportionnelle au carré de la distance :

$$w_i = 1/d_i^2$$

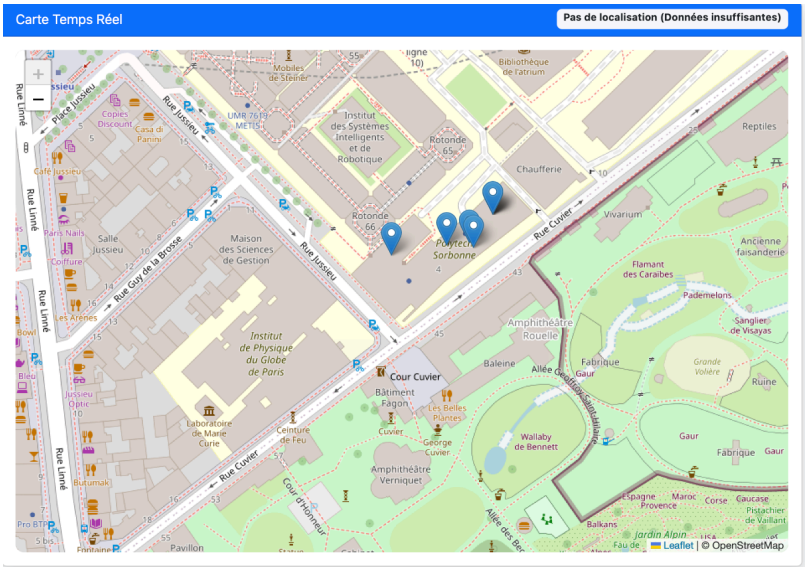
Les coordonnées finales $(Lat_{final}, Lon_{final})$ sont calculées ainsi :

$$(Lat_{final} = \sum Lat_i * w_i / \sum w_i \; ; Lon_{final} = \sum Lon_i * w_i / \sum w_i)$$

Cette méthode permet de privilégier les bornes très proches (signal fort) et de minimiser l'impact des bornes lointaines ou instables.

Une page HTML utilisant la bibliothèque **Leaflet.js** permet une interface graphique utilisant la cartographie interactive.

- **Mécanisme de mise à jour :** Le navigateur effectue du *Polling* (requêtes périodiques) toutes les 2 secondes vers l'API.
- **Visualisation :**
    - **Points Bleus :** Les points d'accès WiFi connus (Ancres).
    - **Marqueur Rouge :** La position calculée de l'ESP32.
    - **Tableau de droite :** Affiche en temps réel les trames LoRa décodées, avec un code couleur pour la qualité du signal (Vert/Orange/Rouge).



Derniers Scans LoRa

MAC	RSSI	Ch
B0:6E:BF:66:61:08	-74	3
E8:65:D4:72:F9:79	-77	6
24:0A:C4:60:DF:71	-63	6
E8:65:D4:72:F9:79	-77	6
B0:6E:BF:66:61:08	-72	3
24:0A:C4:60:DF:71	-61	6
B0:6E:BF:66:61:08	-74	3
E8:65:D4:72:F9:79	-79	6
24:0A:C4:60:DF:71	-67	6
58:97:BD:CD:93:E2	-85	1
B0:6E:BF:66:91:B8	-83	6
E8:65:D4:72:F9:79	-77	6
E8:65:D4:72:F9:79	-78	6
B0:6E:BF:66:91:B8	-83	6
B0:6E:BF:66:61:08	-74	3
E8:65:D4:72:F9:79	-76	6
B0:6E:BF:66:61:08	-74	3

(Fait chez mes parents donc pas localisé)

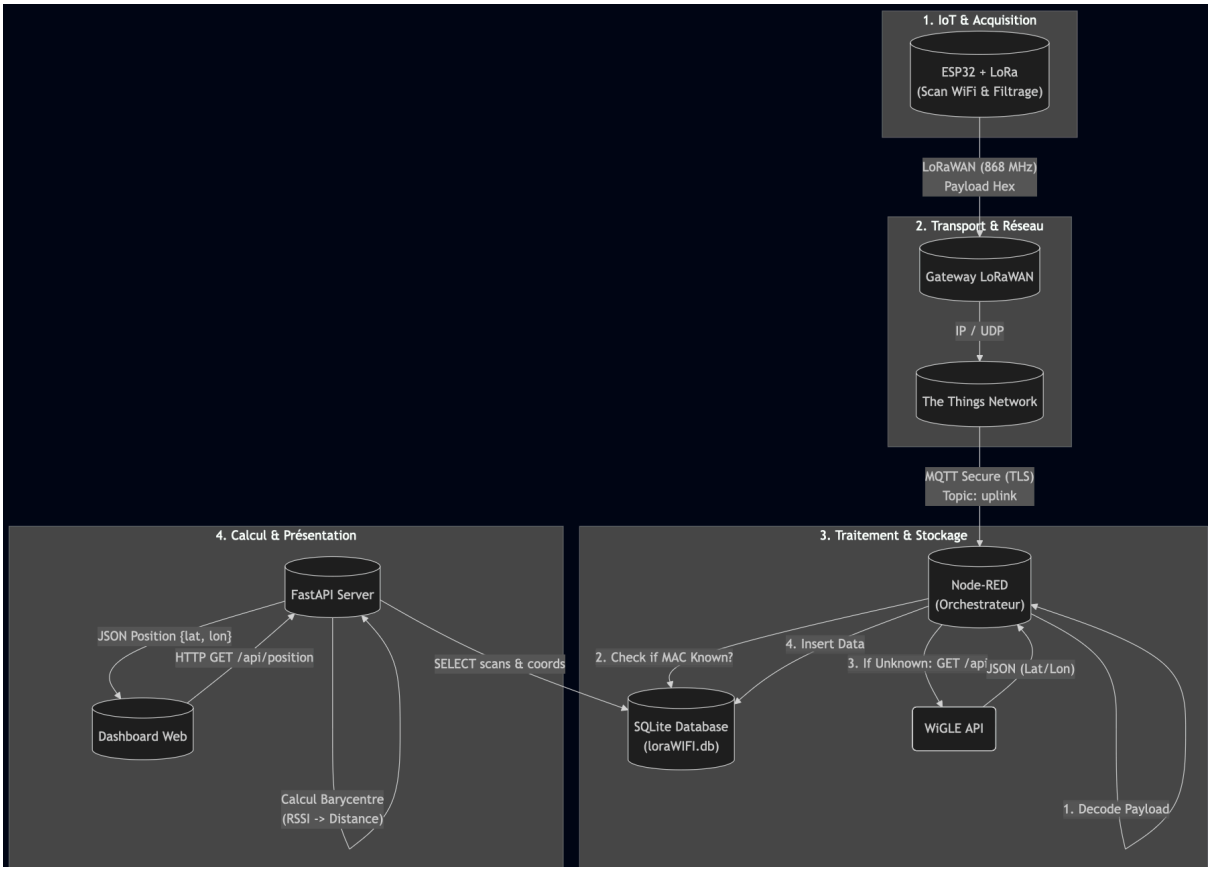


Schéma représentant l’architecture complète du GPS WiFi

Problèmes rencontrés et solutions

Une contrainte majeure a été rencontrée lors de l'implémentation : les quotas de requêtes journalières imposés par l'API WiGLE. Cette limitation a rendu impossible l'alimentation automatique de la base de données telle qu'initialement prévue. Pour contourner ce blocage, nous avons opté pour une phase de calibration manuelle. Cette procédure a consisté à identifier physiquement les points d'accès en positionnant l'ESP32 à proximité immédiate de la source (validé par une faible atténuation), puis à saisir manuellement leurs coordonnées GPS précises dans la base de données SQLite grâce au téléphone qui nous permettait d’avoir les coordonnées GPS.