Plateforme RandoVanGo

PLANIFICATION DE VOYAGES RANDONNEES VAN ITINERANTS

Nathalie BEDIEE
Développeur IA - ISEN Brest
Novembre 20025

Bloc de compétences 1 :
Réaliser la collecte, le stockage et la mise à disposition des données

ISEN OUEST

RandoVanGo

Table des matières

1. PRÉSENTATION DU PROJET

1.1. Genèse du projet

Le projet RandoVanGo est né d'une frustration : l'impossibilité d'organiser un séjour de randonnée en van sans jongler constamment entre cinq à six applications différentes (Météo France, Park4Night, MaRando, etc.). L'outil a été conçu pour centraliser ces informations vitales et résoudre cette dispersion des outils. Il fonctionne comme un planificateur de randonnées (Gestion Planning Rando) permettant à l'utilisateur, soit d'importer une trace GPX personnelle et de l'enrichir, soit de construire son voyage à partir des parcours intégrés à l'application. RandoVanGo offre ainsi une solution unique et complète aux randonneurs vanlifers.

Acteurs : Utilisateurs finaux (randonneurs, vanlifers), contributeurs qui peuvent fournir des nouveaux tracés de randonnées, des points d'intérêt (restaurants, commerces, points d'eau, stations-services, ...) ou des spots (lieu de stationnement et de nuitées)

1.2. Objectifs et spécifications techniques

Le projet s'articule autour d'un pipeline ETL (Extract-Transform-Load) automatisé, dont Architecture technique global intègre plusieurs sources de données :

- **APIs REST publiques :** Données météo (Open-Meteo), Points d'intérêt (POI) issus d'OpenStreetMap (via Overpass API), et données de patrimoine culturel (Wikidata via requête SPARQL).
- **Web scraping :** Spots pour des nuitées en van (Park4Night) en utilisant Selenium/Chromium.
- **Fichiers de données :** Traces de randonnées au format GPX (XML géospatial).

Les données extraites sont transformées en appliquant des règles métier (nettoyage, normalisation, agrégation). Elles sont finalement Chargées dans deux bases de données complémentaires :

- **MySQL** pour les données relationnelles structurées (villes, randonnées, spots, POI, météo).
- **MongoDB** pour les documents JSON géospatiaux (traces GPX complètes

L'exposition des données est assurée par une API REST (FastAPI) documentée selon le standard OpenAPI. Cette API est consommée par un frontend développé en Flask.
Enfin, une base de données **SQLite** gère l'authentification et les rôles des utilisateurs (administrateurs et contributeurs) pour sécuriser l'accès et les actions spécifiques (comme l'upload de fichiers GPX).

Stack Technique :
- Backend & API : Python, FastAPI, Uvicorn
- Authentification : JWT, Passlib (Bcrypt)
- Frontend : Flask, Jinja2, HTML/CSS/JS, Leaflet.js (cartographie)
- Bases de données: MySQL (relationnelle via mysql-connector), MongoDB (NoSQL via PyMongo), Sqlite
- ETL & Traitement : pandas, selenium/chromium, gpxpy (gestion GPX), geopy (Géocodage), requests (appels HTTP), threading (exécution parallèle), logging (suivi), Nominatim (OSM Geocoding)
- Conteneurisation : Docker, Docker Compose

Contraintes : Conformité RGPD (hachage bcrypt, traçabilité auth_logs) Diagramme de séquence d'authentification
Budget : Projet sans budget, utilisant que des outils open-source et APIs gratuites.

Méthodologie : Développement itératif avec cycles courts
- Versionnage Git avec branches : GitHub/natbediee/randovango
- Tests manuels via Swagger UI (FastAPI)
- Arborescence structurée
- logs structurés (backend/logs/)

2. AUTOMATISATION EXTRACTION DES DONNÉES

2.1. Architecture du pipeline ETL

Le pipeline orchestré par `etl_pipeline.py` (`backend/etl/`) coordonne : extraction GPX → identification ville → extractions complémentaires (météo, spots P4N, POI OSM, patrimoine Wikidata) → transformation → chargement MySQL/MongoDB. Diagramme de flux ETL Pipeline

Point de lancement : Automatique lors import GPX via frontend, ou manuel : `docker compose exec backend python3 -m etl.etl_pipeline.`

Initialisation dépendances : Conteneurisation Docker, Dockerfile.backend installe bibliothèques depuis requirements.txt (requests, selenium, gpxpy, pymongo, mysql-connector-python, pandas, geopy).

Gestion erreurs : Logging structuré utils/logger_util.py (fichiers rotatifs + console), try-catch permettant continuation si échec partiel, fallback 3 serveurs Overpass, validation cohérence avant chargement, fin traitement et sauvegarde résultats. Exemples de logs

2.2. Extraction des sources

Fichiers locaux - GPX Diagramme de flux ETL des fichiers GPX
Module gpx.py lit depuis data/in/gpx/ → sélectionne premier .gpx → parsing XML immédiat → extraction (points trace lat/lon/élévation, waypoints, métadonnées) → transformation calcule distance 3D haversine, dénivelé positif, durée Naismith (4km/h + 600m/h montée), géocodage ville du point de départ pour transmettre aux autres ETL.

API REST - Open-Meteo Diagramme de flux ETL de gestion des données météo
Module : api_meteo.py géocode ville (Nominatim)
Requête : GET api.open-meteo.com/v1/forecast (coordonnées, timezone Europe/Paris, variables météo quotidiennes, forecast_days=7)
Retour : parsing JSON → tuples SQL, logs horodatés

Web scraping - Park4Night Diagramme de flux ETL des données Park For Night
Module : scraper_p4n.py utilise Selenium/Chromium en mode headless
Automate : gestion bandeau cookie automatique → navigation URL construire avec les coordonnés (https://park4night.com/fr/search?lat=48.34151&lng=-4.71924&z=15) → attente jusqu'à 40s que l'élément `«<ul class="listmap-list" id="searchmap-list-results">` soit présent, indiquant le chargement complet de la liste des spots→ extraction liens fiches détaillées (https://park4night.com/fr/place/225996) →vérification que le spot n'est pas déjà présent en base si non collecte (nom, description, type, coordonnées, note, services) → consolidation dans un DataFrame pandas.

API REST - Overpass API Diagramme de flux ETL des données Open Street Map
Module : api_osm.py géocode ville, bounding box rayon 5km
Requête : Overpass QL, langage spécifique à OSM syntaxe : nwr["tourism"="attraction"] récupération attractions, eau, toilettes, restaurants, supermarchés, fuel
Fallback sur 3 serveurs (overpass-api.de, overpass.kumi.systems et lz4.overpass-api.de) (timeout 90s)
Même avec cette solution, les requêtes peuvent échouer, un etl_rattrapage a donc été créé, qui relance sur toutes les villes n'ayant pas de données.
Réponse : JSON (ID OSM, coordonnées, tags)

API REST – Wikidata Diagramme de flux ETL des données Wikidata
Module : api_wikidata.py.
Requête : SPARQL (via le service query.wikidata.org/sparql). Récupère les POI depuis les identifiants wikidata dans un rayon de 5 km autour de la ville.(wd:Q570116 # Phare, wd:Q23397 # Plage, ...)
Réponse : JSON (Id QID Wikidata, coordonnées, url image, type poi)

3. AGRÉGATION ET NETTOYAGE

3.1. Agrégation OSM

Module transform_osm.py convertit JSON Overpass → DataFrame pandas.
Suppression corrompus (sans tags/coordonnées),normalisation des noms.
Mapping des POI par catégories métier (restaurant, hébergement, services, patrimoine, nature, autre) pour homogénéisation.
L'agrégation wikidata suit le même concept.

3.2. Agrégation Park4Night

Module transform_p4n.py traite DataFrame scraping.
Parsing coordonnées "(48.3714, -4.7587)" → colonnes séparées
Split de la chaine service → mapping des types de services avec un dict commun à OSM et wikidata
Validation notes (to_numeric errors='coerce', notes >5 ou absente → NaN)

3.3. Agrégation GPX

Module transform_p4n.py prend un fichier GPX brut
Parse avec gpxpy, extrait les points de trace et waypoints
Calcule la distance et le dénivelé, récupère la description et identifie la ville de départ via les coordonnées.
Il retourne un dictionnaire normalisé prêt à être chargé en base, ou None si la ville n'est pas trouvée.

4. BASE DE DONNÉES

4.1. Base de données MySQL 🖇 MCD BDD MySQL, données structurées

La structure des tables MySQL du projet Randovango est organisée autour des entités principales suivantes, elle stock toutes les données nécessaires à la planification d'une randonnée :

- **cities** : Contient les villes (nom, coordonnées, département, région, pays).
- **hikes** : Randonnées, liées à une ville (city_id), avec infos GPX, distance, dénivelé, source, etc.
- **sources** : Référence des sources de données (ex : auteur GPX).
- **spots** : Points d'intérêt (POI) géolocalisés, liés à une ville ou une randonnée.
- **poi** : Services ou POI spécifiques, avec type, coordonnées, etc.
- **weather** : Prévisions météo, liées à une ville (city_id) et une date.
- **trip_plans** : Plans de voyage créés par les utilisateurs, liés à une ville et à un utilisateur ou un token invité.
- **trip_days** : Jours associés à un plan, chaque jour pouvant référencer une randonnée (hike_id) et un spot (spot_id)
- **trip_days_pois** : sur un jour d'itinéraire l'utilisateur peur choisir plusieurs POI

Exemple de relations:

- hikes.city_id → cities.id (une randonnée appartient à une ville)
- trip_plans.user_id → users.id (ou user_token pour invités)
- trip_days.trip_plan_id → trip_plans.id

Cela forme un schéma relationnel centré sur la ville, avec des liens forts entre randonnées, plans, météo et POI. Les relations sont assurées par des clés étrangères, garantissant la cohérence des données.
🖇 Schéma de la base de données MySQL

4.2. Base de données SQLite MCD de la BDD SQLITE, authentification

Le module d'authentification de Randovango repose sur une base SQLite légère, dédiée à la gestion des utilisateurs, des rôles et à l'audit des accès API.

- **users**: stocke les comptes utilisateurs (identifiant, nom d'utilisateur, mot de passe hashé, etc.).
- **roles** : liste des rôles disponibles (admin, contributeur et user).
- **user_role** : table d'association permettant d'attribuer plusieurs rôles à un utilisateur (clé composée user_id, role_id).
- **auth_log** : journalisation des connexions et actions pour l'audit de la sécurité.

4.3. Base de données Mongo

MongoDB est utilisé pour stocker les traces GPX brutes. Ce choix permet de conserver l'intégralité des fichiers d'origine, de faciliter les recherches géospatiales et d'assurer la flexibilité sur les formats de données importés. A terme le front pourra proposer ces fichiers directement aux utilisateurs pour qu'ils les exploitent lors de leur randonnée.

4.4. Requêtes SQL et optimisations Exemples de requêtes

Extraction villes enrichie :

Le module step1.py dans backend/api/routers/ implémente les requêtes SQL pour la sélection des villes. La requête principale extrait l'identifiant, le nom, le département, la région, le pays, la latitude et la longitude depuis la table cities, triés par ordre alphabétique de nom.

Cette requête de base est ensuite enrichie dynamiquement avec trois comptages distincts effectués pour chaque ville :

- récupère le nombre de randonnées vérifiées dans un rayon de 5 kilomètres. Le filtrage sur « verifie> garantit que seules les randonnées validées par les modérateurs soient présentées.
- récupère le nombre de spots van vérifiées dans un rayon de 5 kilomètres
- récupére l'ensemble des points d'intérêt toujours dans un rayon de 5 kilomètres

PS : A terme le filtrage sur 5 km, pourra être un paramètre côté front.

Extraction météo :

Le module load_meteo.py gère l'insertion des prévisions météorologiques avec une logique anti-doublons. Avant chaque insertion, une requête de vérification contrôle l'existence d'une prévision pour le couple (city_id, date. Si la requête retourne un résultat, l'insertion est ignorée.

La récupération des prévisions pour affichage frontend utilise la requête : `SELECT * FROM weather WHERE city_id = %s ORDER BY date ASC`. Le tri chronologique est essentiel pour l'affichage sous forme de timeline météo sur 7 jours.

PS : Un etl_meteo spécifique a été créé pour rafraichir les données côté front pour toujours présenter 7 jours de données par rapport à la date du jour.

5. FASTAPI (API REST) Documentation Swagger UI

FastAPI est utilisé comme serveur principal pour exposer l'API REST du projet : il gère les routes pour la gestion des villes, randonnées, plans, utilisateurs, l'authentification JWT, et orchestre les appels aux modules ETL et aux bases de données. Il assure la rapidité, la sécurité et la documentation interactive de l'API Randovango.

Exemples de routes FastAPI

- **GET /cities** : liste les villes disponibles avec statistiques et météo.
- **POST /create_plan** : crée un nouveau plan de voyage (trip plan) pour un utilisateur ou un invité.
- **GET /hikes/{city_id}** : retourne les randonnées vérifiées pour une ville donnée.
- **POST /auth/login** : authentifie un utilisateur et retourne un token JWT.
- **DELETE /gpx/{gpx_id}** : supprime une trace GPX sur Musql et stockage MongoDB pour les traces non validées par admin.

Ces routes illustrent la diversité des fonctionnalités exposées par l'API Randovango (gestion des données, planification, sécurité, personnalisation).

6. CONCLUSION

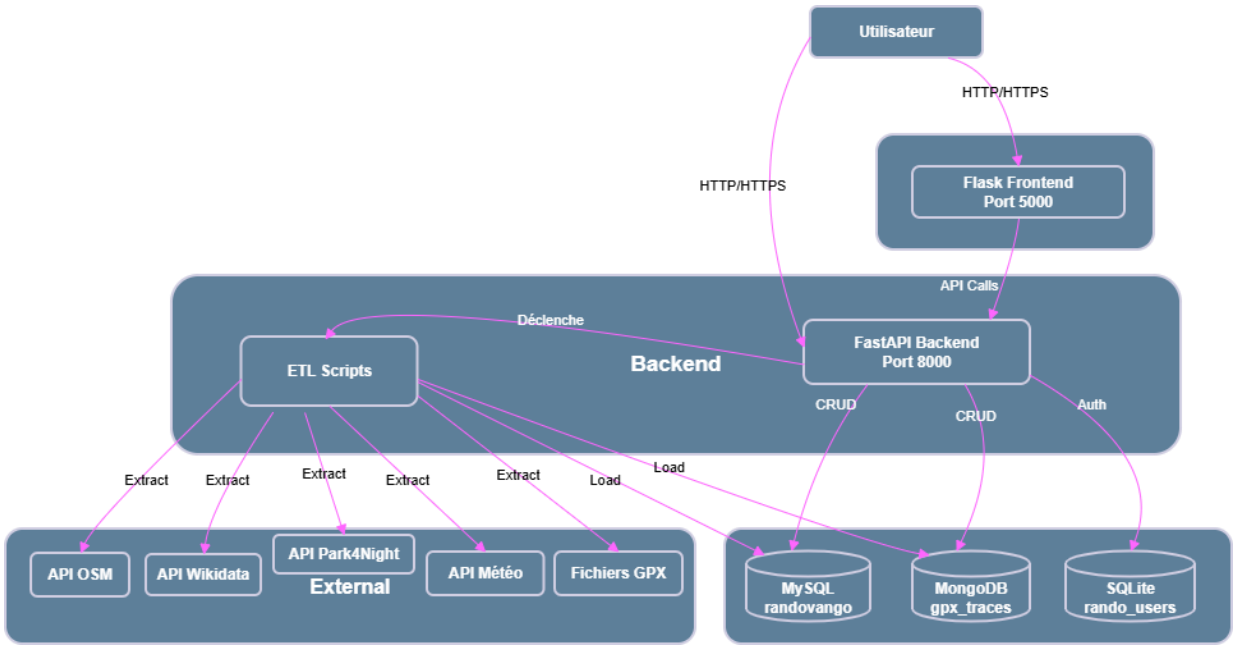
Le projet RandoVanGo pose les fondations d'une plateforme évolutive grâce à son architecture modulaire et ses flux de données automatisés. Plusieurs axes d'amélioration sont identifiés pour enrichir l'expérience utilisateur et renforcer la valeur ajoutée. L'interface utilisateur finale, accessible via le frontend Flask, concrétise l'ensemble de cette chaîne de traitement en proposant aux randonneurs en van une expérience complète de planification de séjours itinérants. 🔗 Vues du front

L'intégration de modules d'intelligence artificielle constitue une évolution naturelle du projet : recommandations personnalisées d'itinéraires basées sur l'historique utilisateur et les préférences déclarées, analyse prédictive des conditions météorologiques combinant données historiques et tendances saisonnières, génération automatique de plannings optimisés selon le profil physique et les contraintes temporelles. Ces fonctionnalités s'appuieraient sur les données déjà collectées et stockées, minimisant l'impact technique de leur implémentation.

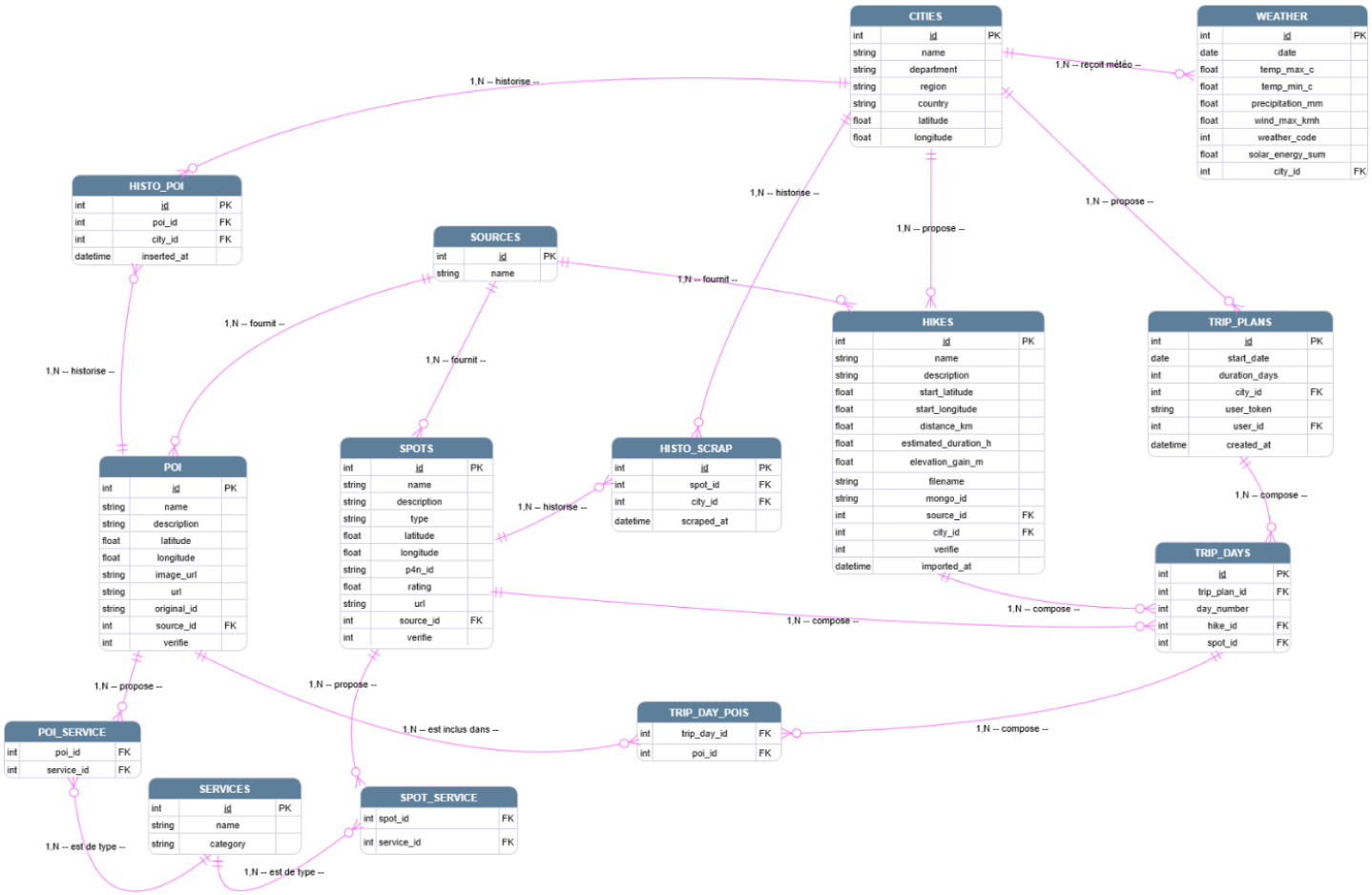
La conception modulaire du pipeline ETL facilite l'adaptation à d'autres pratiques itinérantes (bikepacking, trekking sac à dos). L'ajout de nouvelles sources de données ou de critères de filtrage spécifiques (difficulté technique, accessibilité PMR, présence de refuges) ne nécessiterait que l'ajout de modules ETL supplémentaires sans remise en cause de l'architecture existante.

7. ANNEXES

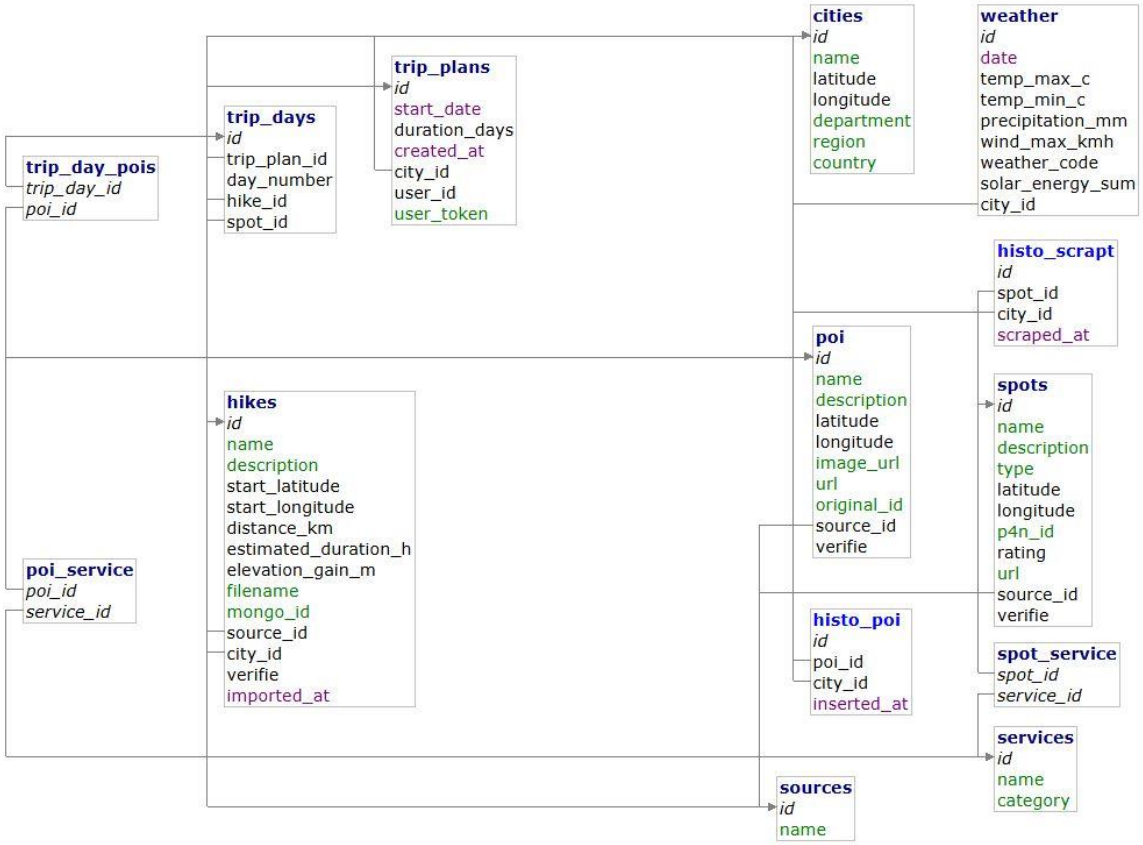
7.1. Architecture technique global Retour



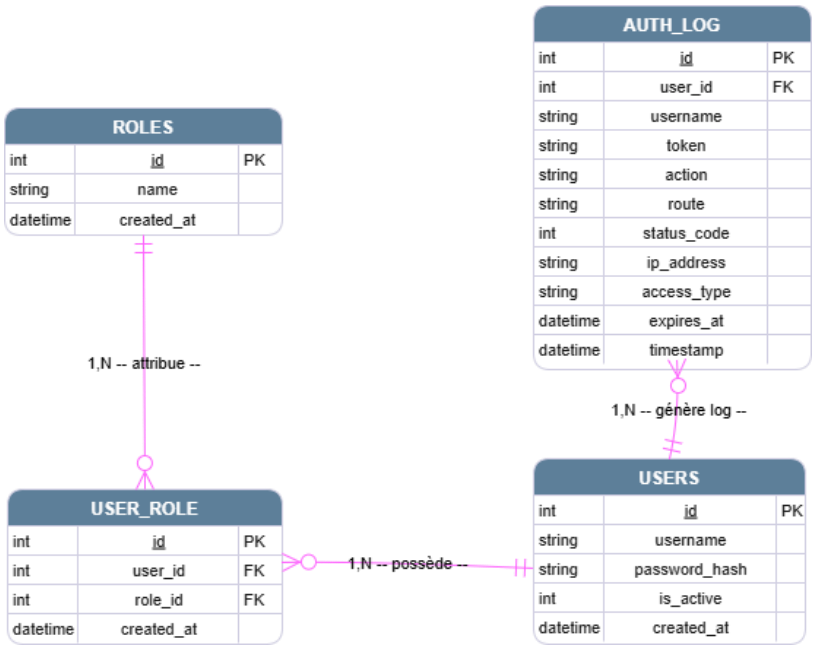
7.2. MCD BDD MySQL, do<<nnées structurées nécessaire à la planification Retour



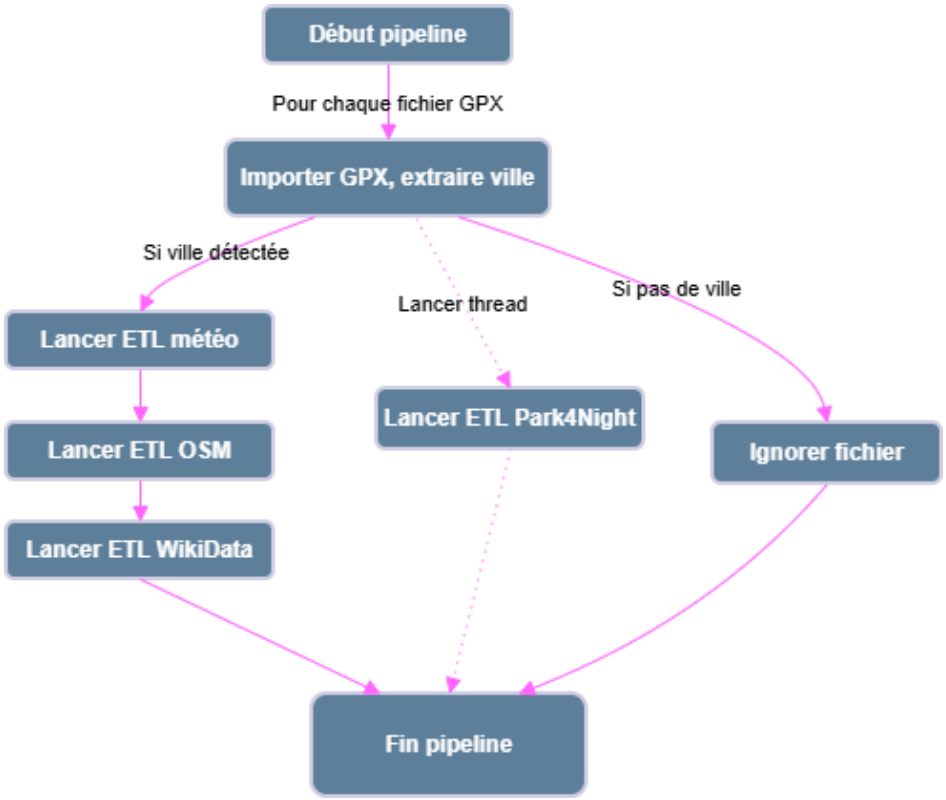
7.3. Schéma de la base de données MySQL Retour



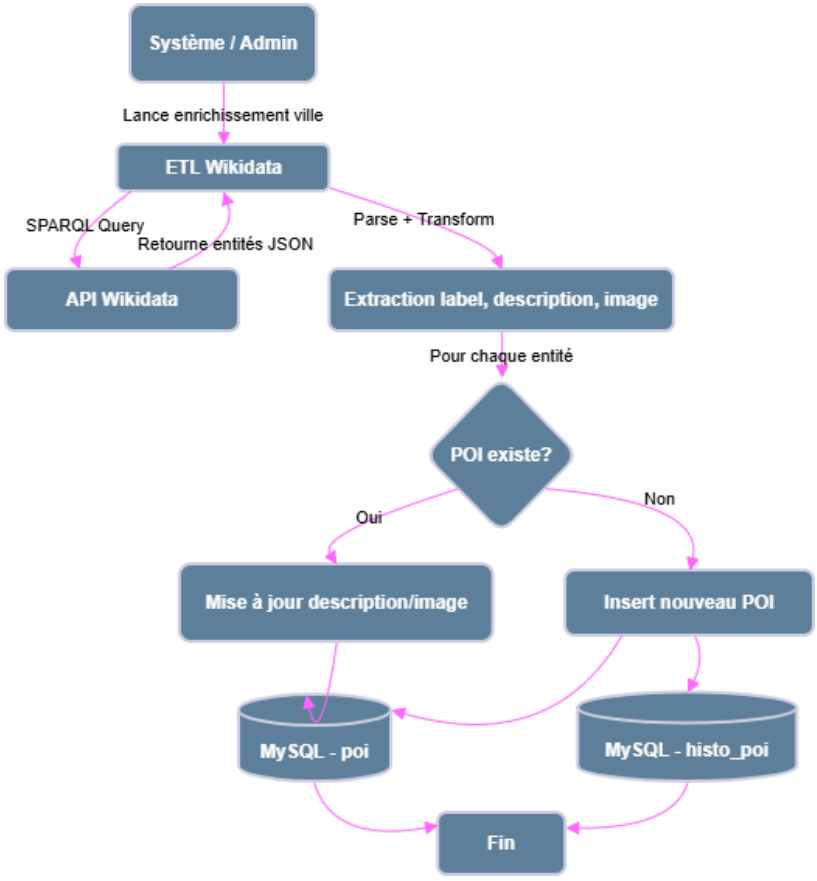
7.4. MCD de la BDD SQLITE, authentification Retour



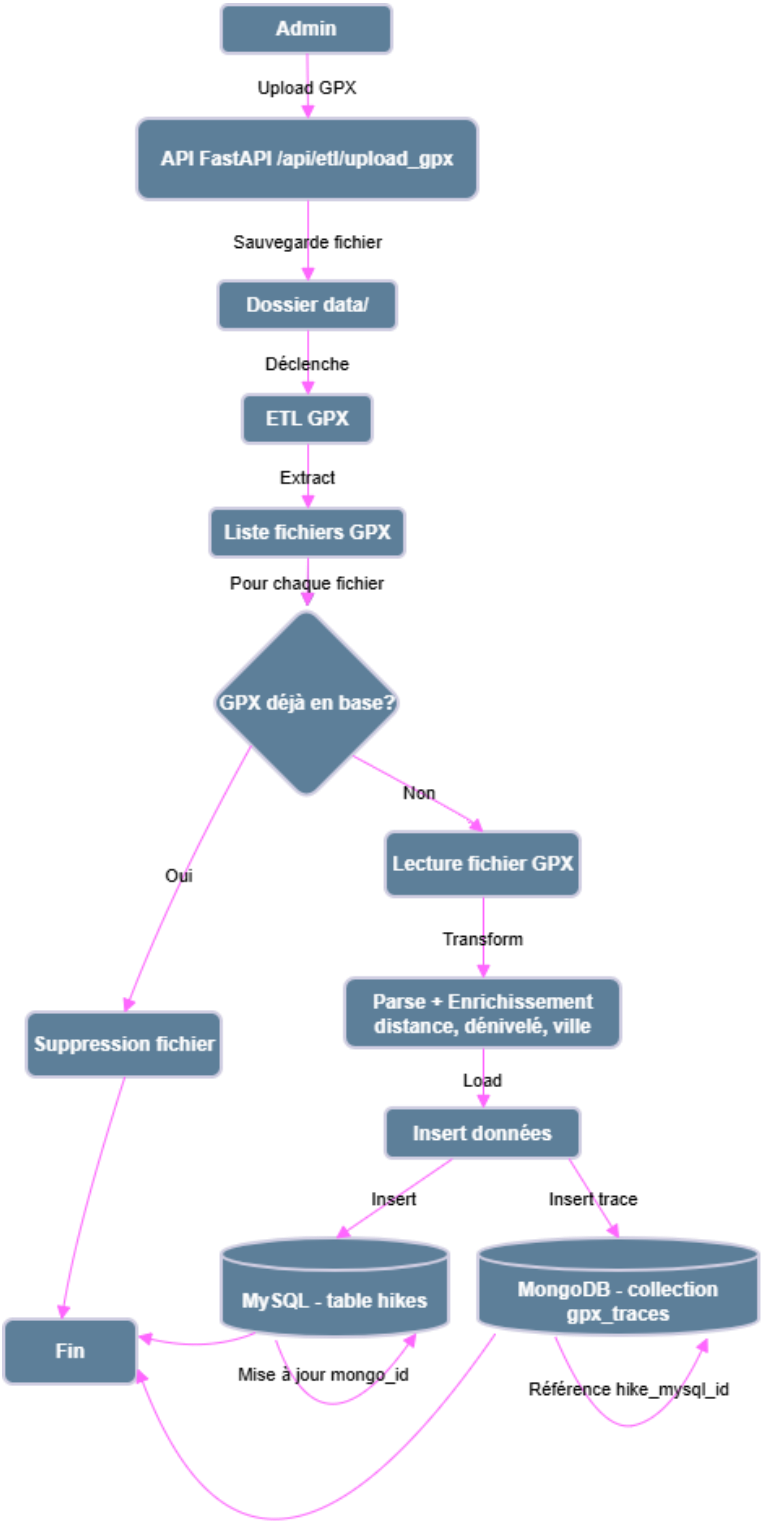
7.5. Diagramme de flux ETL Pipeline Retour



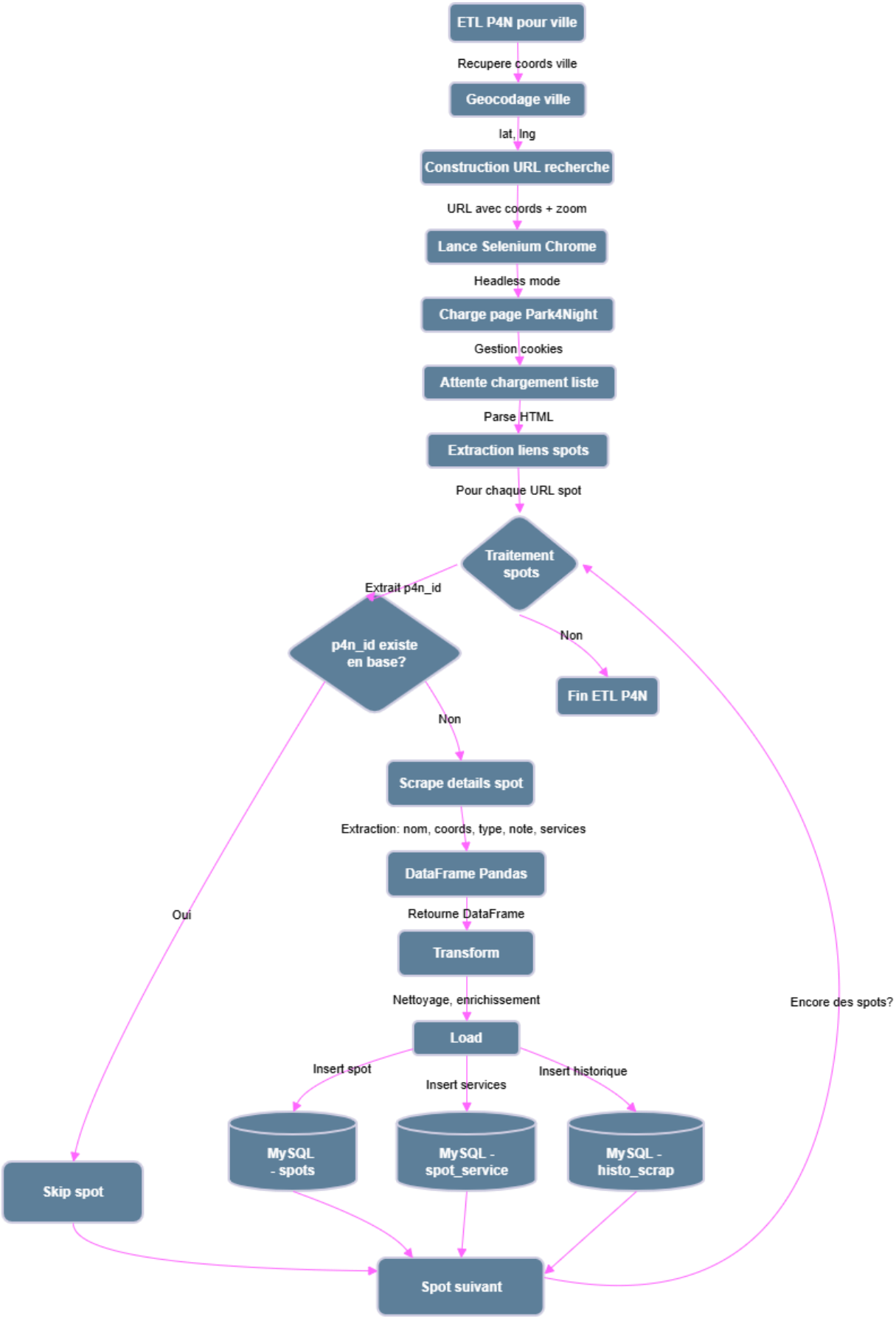
7.6. Diagramme de flux ETL des données Wikidata Retour



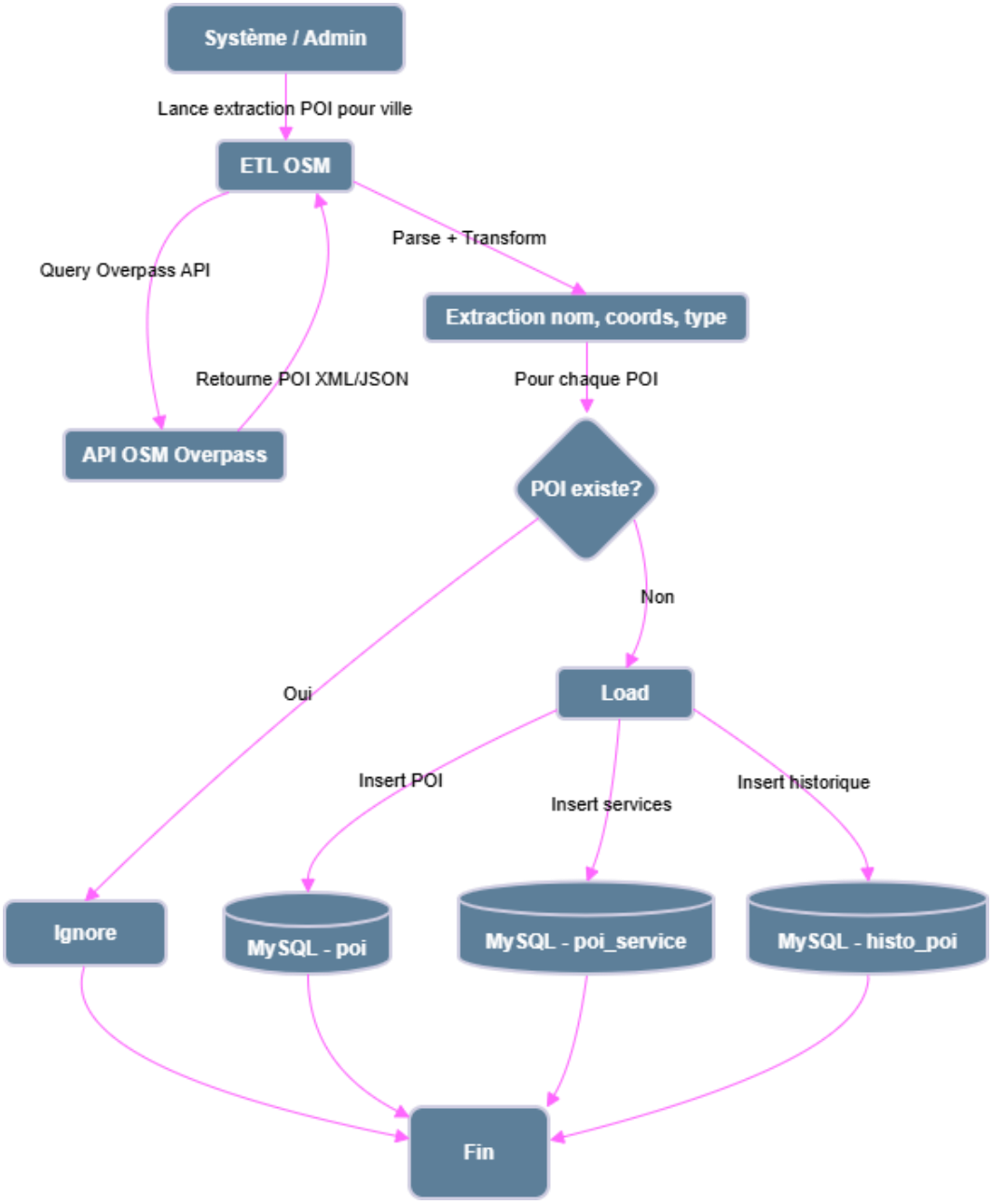
7.7. Diagramme de flux ETL des fichiers GPX Retour



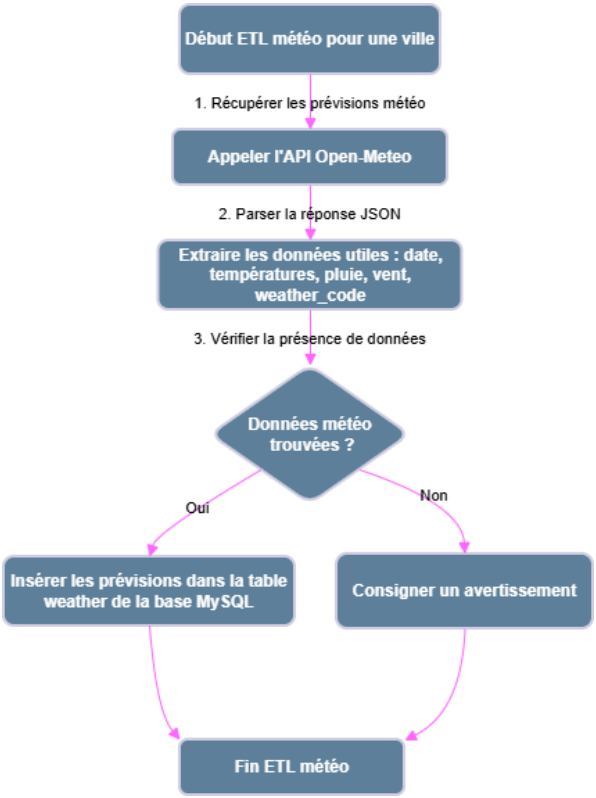
7.8. Diagramme de flux ETL des données Park For Night Retour



7.9. Diagramme de flux ETL des données Open Street Map Retour



7.10. Diagramme de flux ETL de gestion des données météo Retour

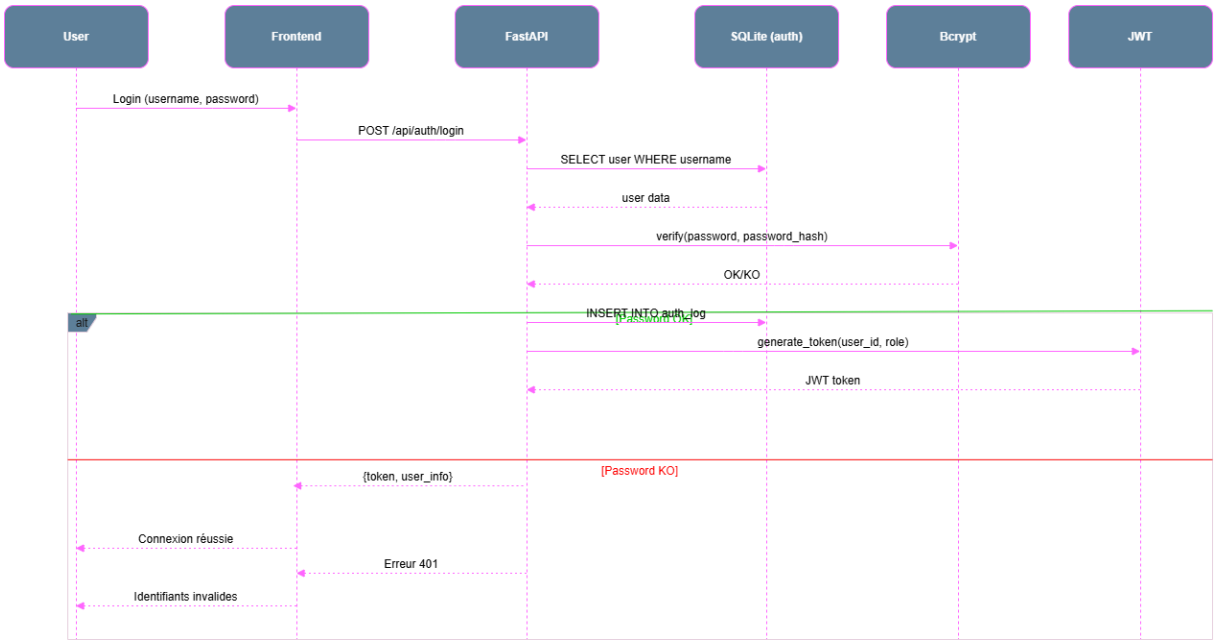


7.11. Exemples de requêtes Retour

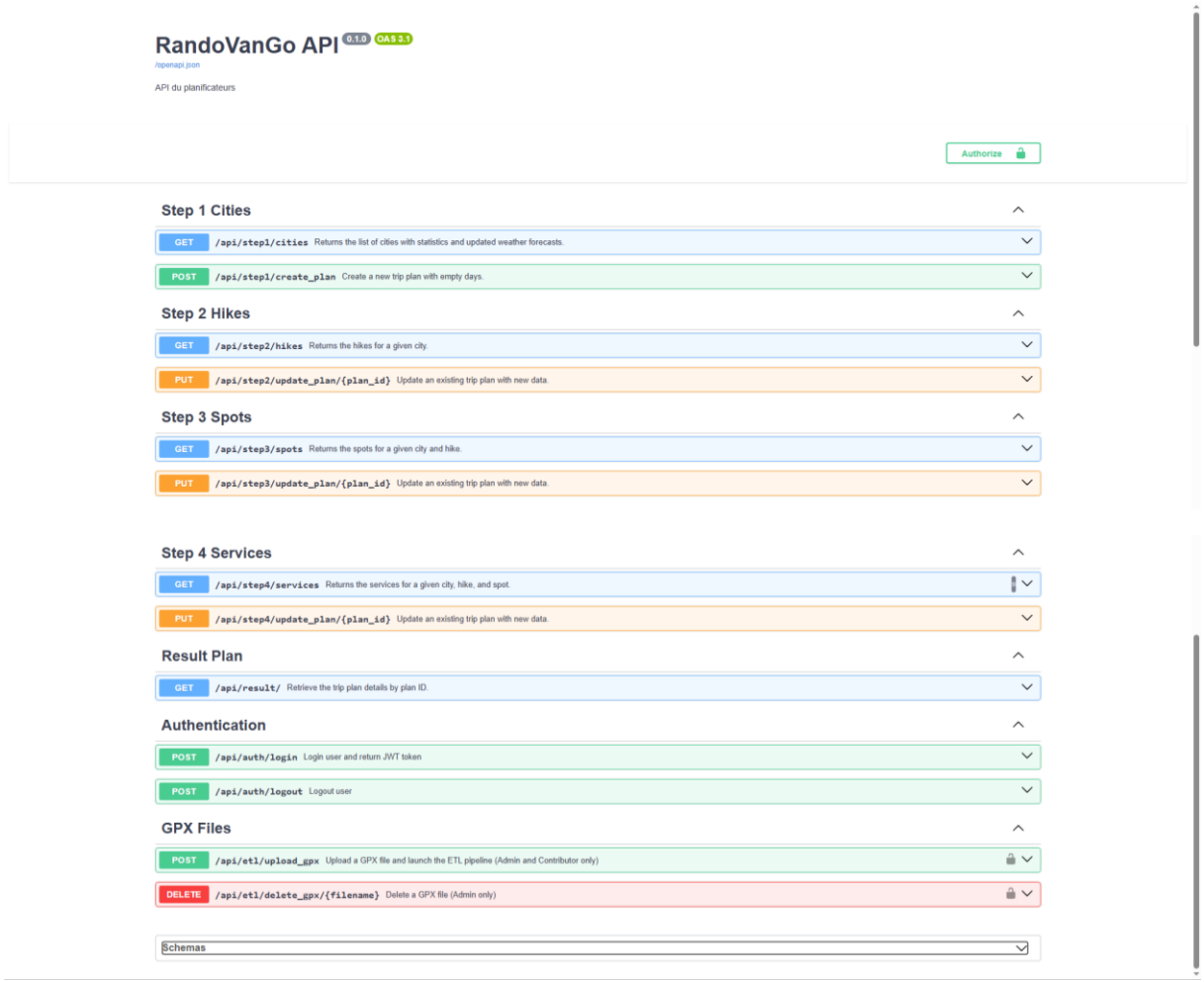
docs > requetes.sql

```sql
-- Annexe - Requêtes SQL réelles utilisées dans le projet Randovango (avec références)

-- 1. Sélection du nombre de randonnées vérifiées dans un rayon autour d'une ville
-- Fichier : backend/api/routers/step1.py
SELECT COUNT(*) as count FROM hikes
WHERE verifie = 1 AND start_latitude BETWEEN :min_lat AND :max_lat AND start_longitude BETWEEN :min_lon AND :max_lon;

-- 2. Vérification de l'existence de prévisions météo à J+6
-- Fichier : backend/api/routers/step1.py
SELECT COUNT(*) as count FROM weather WHERE date >= :date_j6;

-- 3. Sélection des villes (pour affichage)
-- Fichier : backend/api/routers/step1.py
SELECT id, name, department, region, country, latitude, longitude FROM cities ORDER BY name ASC;

-- 4. Sélection des prévisions météo pour une ville
-- Fichier : backend/api/routers/step1.py
SELECT * FROM weather WHERE city_id = :city_id ORDER BY date ASC;

-- 5. Création d'un nouveau plan de voyage (trip_plans)
-- Fichier : backend/api/routers/step1.py
INSERT INTO trip_plans (start_date, duration_days, city_id, user_token, user_id, created_at)
VALUES (:start_date, :duration_days, :city_id, :user_token, :user_id, NOW());

-- 6. Création des jours associés à un plan (trip_days)
-- Fichier : backend/api/routers/step1.py
INSERT INTO trip_days (trip_plan_id, day_number, hike_id, spot_id) VALUES (:trip_plan_id, :day_number, NULL, NULL);

-- 7. Mise à jour du champ mongo_id dans une randonnée (GPX)
-- Fichier : backend/etl/load/load_gpx.py
UPDATE hikes SET mongo_id = :id_mongo WHERE id = :gpxtrace_id;

-- 8. Sélection des rôles d'un utilisateur (jointure roles/user_role)
-- Fichier : backend/services/authentification.py
SELECT r.name FROM roles r JOIN user_role ur ON ur.role_id = r.id WHERE ur.user_id = :user_id;

-- 9. Sélection du nombre de scrapings pour une ville (jointure histo_scrap/cities)
-- Fichier : backend/etl/etl_pipeline.py
SELECT COUNT(*) FROM histo_scrap hs JOIN cities c ON hs.city_id = c.id WHERE c.name = :city_name;
```

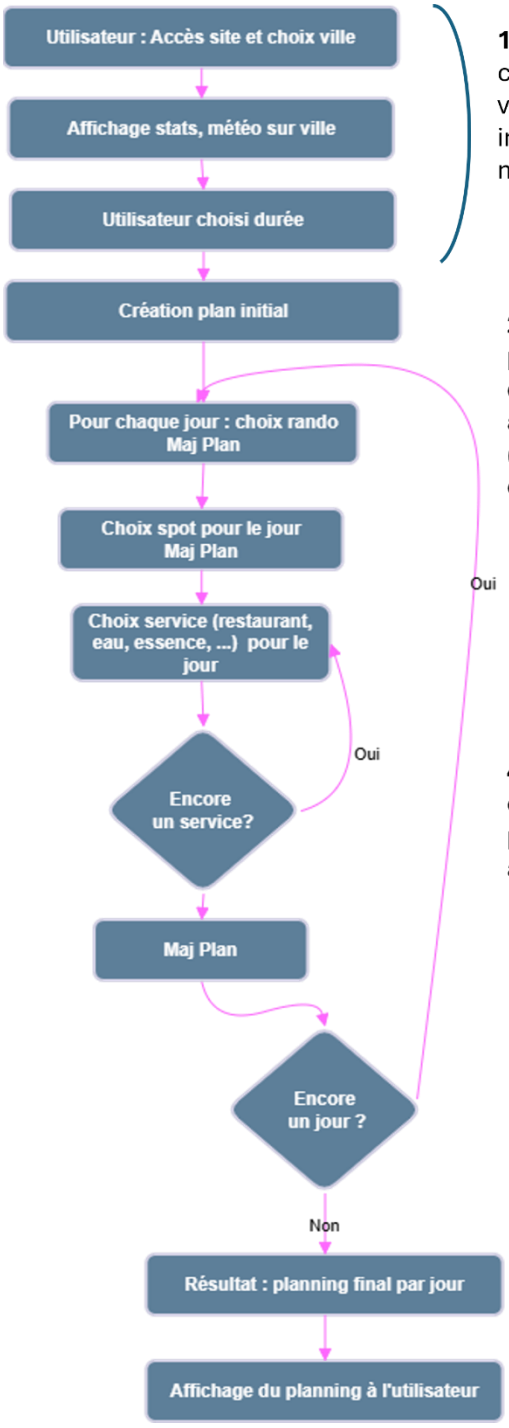
7.12. Diagramme de séquence authentification Retour



7.13. Documentation Swagger UI Retour



7.14. Gestion Planning Rando Retour



1. **Sélection d'une ville** comme point de départ avec visualisation cartographique interactive et prévisions météorologiques sur 7 jours

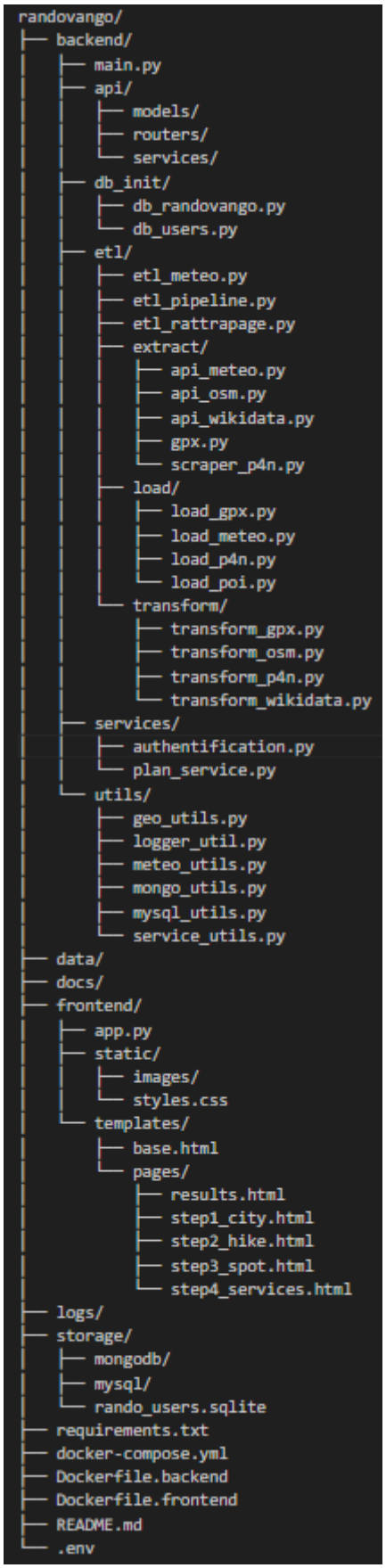
2. **Choix d'une randonnée** parmi les traces GPX disponibles autour de la ville avec informations détaillées (distance, dénivelé, durée estimée)

3. **Identification de spots de nuit** pour van autour de la ville.

4. **Localisation des services essentiels** (points d'eau potable, toilettes publiques, alimentation, stations-service)

5. **Génération d'un planning multi-jours** exportable et modifiable selon les besoins

7.15. Arborescence projet Retour



7.16. Exemples de logs Retour

logs > scraper_p4n.log

```
6332  2025-10-29 14:08:03,203 - INFO - Extrait: (14170) Le Billot - D39
6333  2025-10-29 14:08:04,241 - INFO - --- Scraping de l'URL : https://park4night.com/fr/place/353250 ---
6334  2025-10-29 14:08:04,927 - INFO - Extrait: (61120) Vimoutiers - 10 Avenue du Général de Gaulle
6335  2025-10-29 14:08:05,967 - INFO - --- Scraping de l'URL : https://park4night.com/fr/place/2016 ---
6336  2025-10-29 14:08:07,275 - INFO - Extrait: (61120) Vimoutiers - Rue du Docteur Dentu
6337  2025-10-29 14:08:08,295 - INFO - --- Scraping de l'URL : https://park4night.com/fr/place/440818 ---
6338  2025-10-29 14:08:09,218 - INFO - Extrait: (61250) Saint-Nicolas-des-Bois - 81 L'Ermitage
6339  2025-10-29 14:08:10,264 - INFO - --- Scraping de l'URL : https://park4night.com/fr/place/451754 ---
6340  2025-10-29 14:08:13,007 - INFO - Extrait: (61250) Écouves - 3 Route du Salut
6341  2025-10-29 14:08:14,043 - INFO - --- Scraping de l'URL : https://park4night.com/fr/place/205015 ---
6342  2025-10-29 14:08:14,948 - INFO - Extrait: (61120) Camping Municipal La Campière - Vimoutiers
6343  2025-10-29 14:08:15,983 - INFO - --- Scraping de l'URL : https://park4night.com/fr/place/289282 ---
6344  2025-10-29 14:08:16,751 - INFO - Extrait: (14140) Val-de-Vie - D179A
6345  2025-10-29 14:08:17,772 - INFO - --- Scraping de l'URL : https://park4night.com/fr/place/78531 ---
6346  2025-10-29 14:08:18,624 - INFO - Extrait: (14140) Cave Cibricole Gautar
6347  2025-10-29 14:08:19,663 - INFO - --- Scraping de l'URL : https://park4night.com/fr/place/113959 ---
```

logs > api_osm.log

```
26  2025-10-29 09:37:31,147 - INFO - Données OSM extraites pour Saint-Renan (302 POI)
27  2025-10-29 09:50:03,627 - INFO - Lancement de l'extraction OSM via Overpass pour 'Saint-Renan'...
28  2025-10-29 09:50:03,628 - INFO - Recherche des coordonnées pour 'Saint-Renan'...
29  2025-10-29 09:50:03,858 - INFO - Coordonnées trouvées : 48.4328451, -4.622982
30  2025-10-29 09:50:03,859 - INFO - Bounding box : {'south': 48.382845100000004, 'north': 48.4828451, 'west': -4.672982, 'east':
31  2025-10-29 09:50:03,859 - INFO - Tentative de connexion au serveur : https://overpass-api.de/api/interpreter
32  2025-10-29 09:50:05,986 - INFO - Lancement de l'extraction OSM via Overpass pour 'Portsall'...
33  2025-10-29 09:50:05,986 - INFO - Recherche des coordonnées pour 'Portsall'...
34  2025-10-29 09:50:06,694 - INFO - Coordonnées trouvées : 48.5578742, -4.6982378
35  2025-10-29 09:50:06,696 - INFO - Bounding box : {'south': 48.5078742, 'north': 48.6078742, 'west': -4.7482378, 'east': -4.6482
36  2025-10-29 09:50:06,698 - INFO - Tentative de connexion au serveur : https://overpass-api.de/api/interpreter
37  2025-10-29 09:50:10,824 - INFO - Nombre d'éléments trouvés : 302
38  2025-10-29 09:50:10,825 - INFO - Succès : 302 POI trouvés pour 'Saint-Renan'
39  2025-10-29 09:50:10,826 - INFO - Données OSM extraites pour Saint-Renan (302 POI)
40  2025-10-29 09:50:13,186 - WARNING - Erreur de connexion au serveur https://overpass-api.de/api/interpreter : 504 Server Error:
41  2025-10-29 09:50:13,186 - INFO - Tentative de connexion au serveur : https://overpass.kumi.systems/api/interpreter
42  2025-10-29 09:51:04,122 - INFO - Nombre d'éléments trouvés : 307
43  2025-10-29 09:51:04,123 - INFO - Succès : 307 POI trouvés pour 'Portsall'
44  2025-10-29 09:51:04,123 - INFO - Données OSM extraites pour Portsall (307 POI)
45  2025-10-29 09:58:00,473 - INFO - Lancement de l'extraction OSM via Overpass pour 'Tal ar Groas'...
46  2025-10-29 09:58:00,473 - INFO - Recherche des coordonnées pour 'Tal ar Groas'...
47  2025-10-29 09:58:01,825 - INFO - Coordonnées trouvées : 48.2482858, -4.4182903
48  2025-10-29 09:58:01,826 - INFO - Bounding box : {'south': 48.1982858, 'north': 48.298285799999995, 'west': -4.4682903, 'east':
49  2025-10-29 09:58:01,827 - INFO - Tentative de connexion au serveur : https://overpass-api.de/api/interpreter
50  2025-10-29 09:58:03,595 - INFO - Nombre d'éléments trouvés : 158
51  2025-10-29 09:58:03,596 - INFO - Succès : 158 POI trouvés pour 'Tal ar Groas'
52  2025-10-29 09:58:03,596 - INFO - Données OSM extraites pour Tal ar Groas (158 POI)
```

7.17. Vues du front Retour