INSA INSTITUT NATIONAL DES SCIENCES APPLIQUÉES TOULOUSE

Projet de fin d'études
présenté par
Nell PARTY
Elève Ingénieur de l'INSA Toulouse
Spécialité AE
Filière SIEC
2024-2025

CRÉATION DE MÉTHODE D'IMPORT DE DONNÉES DANS LA LOGICIEL OPEN-SOURCE : GEOTREK.

Lieu du Projet de Fin d'Études
Makina Corpus
Tuteur du Projet de Fin d'Études
Joaquim NALLAR
Correspondant pédagogique INSA
Pascal ACCO

PFE soutenu le 00/00/20XX

MAKINA CORPUS

RESUME

Dans le cadre de mon stage de fin d'études réalisé chez Makina Corpus, j'ai travaillé à l'amélioration de l'interopérabilité entre Geotrek, une suite logicielle open-source de gestion des territoires, et OpenStreetMap, une base de données cartographique collaborative. L'objectif principal du stage était de concevoir une méthode d'import automatique des données OSM vers Geotrek, afin de réduire la saisie manuelle pour les utilisateurs et d'enrichir la base de données existante.

Après une analyse approfondie des deux structures de données, j'ai développé une série de parsers modulaires basés sur l'API Overpass. Ces outils permettent d'extraire, filtrer, transformer et intégrer automatiquement les objets OSM dans Geotrek, tout en gérant les cas particuliers liés aux géométries, aux champs absents et aux traductions. Un système de gestion des fichiers liés a également été intégré.

En complément, j'ai conçu un outil dédié à la génération de tronçons à partir d'OSM. Disponible via API ou en ligne de commande, cet outil produit des fichiers GeoJSON prêts à l'import. L'ensemble des développements a été mené dans une logique open source, avec tests, documentation et publication sur GitHub. Ce stage m'a permis d'allier des enjeux techniques concrets à une approche orientée qualité et réutilisabilité.

REMERCIEMENTS

Au terme de ce stage de cinq mois au sein de l'entreprise Makina Corpus, je tiens à adresser mes plus sincères remerciements à toutes les personnes qui m'ont accompagné, soutenu et guidé tout au long de cette expérience enrichissante.

Je remercie tout d'abord mes tuteurs, Joaquim Nallar, Jean-Etienne Castagnede et Justine Fricou, pour leur encadrement attentif, leur disponibilité et leur aide précieuse chaque fois que je rencontrais des difficultés. Leur accompagnement régulier a grandement contribué à ma progression tout au long du stage.

Je souhaite également remercier chaleureusement toute l'équipe Geotrek, Bastien Potiron, Marine Faucher, Florian Sommariva, Bastien Alvez, ainsi que de nouveau Joaquim Nallar, Jean-Etienne Castagnede et Justine Fricou, pour leur accueil, leur esprit d'équipe et la qualité des échanges que nous avons pu avoir autour du projet. Leur expertise et leur enthousiasme m'ont permis de mieux comprendre les enjeux et les objectifs de Geotrek, tout en m'intégrant dans une dynamique collective stimulante.

Je remercie aussi Lise Benazeth du service Ressources Humaines pour m'avoir accordé sa confiance en m'acceptant en stage, malgré mes connaissances limitées au départ dans le domaine. Cette opportunité m'a permis de monter en compétence et de mieux cerner mes aspirations professionnelles.

Je tiens à exprimer ma gratitude à Makina Corpus dans son ensemble pour m'avoir accueilli et intégré comme un membre à part entière de l'équipe. J'ai pu évoluer dans un environnement bienveillant et motivant, propice à l'apprentissage et à l'autonomie.

Enfin, je souhaite remercier les personnes extérieures à Makina Corpus qui ont participé au projet auquel j'ai contribué : Camille Monchicourt (Parc national des Écrins), Amandine Sahl (Parc national des Cévennes), Raphaël Doisy (coordinateur Geotrek aux Écrins) ainsi que Jean-Christophe Becquet (cofondateur d'APITUX). Leur expertise, leur implication et la qualité de nos échanges ont joué un rôle essentiel dans l'avancement du travail autour de l'intégration des données Geotrek dans OpenStreetMap.

TABLE DES MATIERES

GLOSSAIRE

Terme	Définition
API	Application Programming Interface. Interface de communication permettant à deux logiciels d'échanger des données ou des services.
BBox	Bounding Box (boîte englobante). Rectangle défini par deux points opposés, utilisé pour restreindre une zone d'analyse géographique.
Centroid	Point représentant le centre géométrique d'un objet spatial, souvent utilisé pour simplifier la localisation d'une entité complexe.
CI/CD	Continuous Integration / Continuous Deployment. Ensemble de pratiques visant à automatiser les tests et le déploiement du code dans un projet logiciel.
Docker	Technologie de virtualisation légère permettant d'exécuter des applications dans des conteneurs isolés, portables et reproductibles.
Django	Framework web Python basé sur le modèle MVT (Modèle, Vue, Template), utilisé pour développer des applications web robustes et structurées.
Django REST Framework	Extension de Django facilitant la création d'API RESTful pour exposer ou consommer des données de manière standardisée.
EPCI	Établissements Publics de Coopération Intercommunale. Structures administratives françaises regroupant plusieurs communes pour gérer des compétences partagées.
Flake8 / Ruff	Outils d'analyse statique du code Python, utilisés pour vérifier la qualité et la conformité du code aux standards.
GeoJSON	Format de fichier basé sur JSON, utilisé pour représenter des données géographiques (points, lignes, polygones) de manière structurée et lisible.
Git	Système de gestion de versions décentralisé permettant de suivre l'historique des modifications d'un projet et de collaborer efficacement à plusieurs.
Mock (test)	Objet simulé utilisé lors des tests unitaires pour remplacer des composants externes (comme une API) et en contrôler le comportement.
Multipolygon	Objet géographique complexe composé de plusieurs polygones, typiquement utilisé pour représenter des zones administratives ou naturelles dans OSM.
OpenStreetMap (OSM)	Base de données cartographique libre et collaborative, alimentée par des contributeurs du monde entier.
Overpass API	API spécifique à OpenStreetMap permettant de filtrer et extraire des objets cartographiques selon des critères précis (tags, zone géographique, etc.).
Parser	Composant logiciel qui permet d'interpréter et de convertir des données d'un format source vers un format cible compatible avec une application donnée.
POI	Point Of Interest (Point d'intérêt). Lieu remarquable à référencer dans une carte, souvent associé à une information touristique ou patrimoniale.

SIG	Systèmes d'Information Géographique. Outils informatiques permettant la capture, le stockage, l'analyse et la visualisation de données géolocalisées.
SIT	Systèmes d'Information Touristique. Plateformes destinées à centraliser, gérer et diffuser les informations liées à l'offre touristique d'un territoire.
SRID	Spatial Reference System Identifier. Identifiant numérique d'un système de coordonnées géographiques (ex : 4326 pour WGS84, 2154 pour Lambert-93).
TMA	Tierce Maintenance Applicative. Prestation externalisée de maintenance et d'évolution d'un logiciel par une société tierce.
WKT	Well-Known Text. Format textuel standardisé permettant de représenter des objets géographiques (points, lignes, polygones, etc.).

CHAPITRE 1. INTRODUCTION

Les Systèmes d'Information Géographique (SIG) sont des logiciels permettant aux institutions publiques, telles que les parcs naturels nationaux ou régionaux, les conseils départementaux, les communautés de communes ou encore les régions, de mieux organiser la gestion et la maintenance de leur territoire. Ces outils offrent une vision globale et précise de l'état des infrastructures (passerelles, bancs, signalétiques, etc.) en les localisant et en renseignant leur état. Cette capacité de suivi facilite la planification des interventions et contribue à répondre à des enjeux variés, allant de la sécurité à la valorisation touristique, en passant par l'accessibilité des sites.

Cependant, les SIG tels que QGIS ou ArcGIS restent complexes à prendre en main. La richesse fonctionnelle de ces outils, bien que précieuse, exige une certaine expertise technique. Leur utilisation nécessite souvent une formation approfondie, voire des compétences spécifiques, notamment en SQL. Ce niveau d'exigence constitue un frein pour les agents non spécialistes qui ont pourtant besoin de manipuler des données géographiques dans leur travail quotidien.

Dans cette optique, en 2012, le Parc national des Écrins, le Parc national du Mercantour et le Parco delle Alpi Marittime ont lancé un appel d'offres pour la création d'un outil plus accessible. L'objectif était de concevoir une solution à la fois intuitive et adaptée aux besoins concrets de gestion et de valorisation du territoire. C'est ainsi qu'est née Geotrek, une application développée par Makina Corpus, entreprise spécialisée dans le développement d'applications cartographiques web et mobile, open-source.

Geotrek est aujourd'hui une suite applicative open source composée de quatre modules. Le cœur du dispositif est Geotrek-admin, destiné aux gestionnaires de territoire. Cette interface regroupe deux grandes fonctionnalités. D'une part, la gestion du territoire : les utilisateurs peuvent y renseigner les aménagements présents sur le terrain, comme les panneaux ou les équipements de signalisation, en précisant leur localisation et leur état. Il est par exemple possible de signaler la dégradation d'une lame sur un panneau d'information ou de planifier une intervention dans une zone donnée. Cette fonctionnalité s'apparente à un SIG, mais dans une version simplifiée et plus accessible.

D'autre part, Geotrek-admin permet également de valoriser le territoire auprès du public. Les gestionnaires peuvent y créer des itinéraires de randonnée, définir des sites d'activités de plein air, ou encore intégrer des points d'intérêt. Ces données peuvent inclure, entre autres, le profil altimétrique des parcours, les points remarquables situés à proximité, ou les zones écologiquement sensibles, comme des aires de nidification. Ces éléments sont ensuite diffusés via trois autres modules de la suite Geotrek : Geotrek-rando, Geotrek-widget et Geotrek-mobile, tous dédiés à la mise en avant de ces contenus auprès du public.

Aujourd'hui, près d'une centaine institutions publiques utilisent Geotrek. La majorité sont françaises, mais la solution commence à s'exporter à l'international, notamment au Canada, dans la région de la Gaspésie.

C'est dans ce contexte que s'est inscrit mon stage, réalisé au sein de Makina Corpus. Ma mission principale portait sur la création d'une communication de données entre Geotrek et OpenStreetMap, une base de données cartographique libre, collaborative et constamment actualisée grâce à la contribution de plus de dix millions d'utilisateurs. Jusqu'alors, aucun lien direct n'existait entre ces deux systèmes.

L'objectif du stage était donc de concevoir des mécanismes permettant de tirer parti des données d'OpenStreetMap afin d'enrichir automatiquement la base de données de Geotrek. L'enjeu était principalement de limiter la saisie manuelle pour les utilisateurs de Geotrek.

Ce rapport s'organise en plusieurs chapitres permettant de retracer de manière progressive l'ensemble du travail effectué durant mon stage. Il commence par une présentation du cadre du stage, des objectifs qui m'ont été fixés ainsi que de la place qu'occupait ce projet au sein de l'entreprise Makina Corpus. Sont également abordés les domaines scientifiques mobilisés, en lien avec les problématiques d'interopérabilité des données et de gestion de données géographiques.

La suite du rapport est consacrée aux réalisations techniques. Elle décrit en détail le développement d'un système d'import des données issues d'OpenStreetMap dans Geotrek, la création d'un outil de génération automatique de tronçons, la gestion des projets en contexte open source, ainsi que les premières réflexions sur un outil de contribution vers OpenStreetMap depuis Geotrek. Chaque réalisation est replacée dans son contexte, en précisant les choix techniques, les solutions mises en œuvre et les résultats obtenus.

Enfin, un chapitre est dédié à une analyse réflexive de mon expérience, afin d'en tirer un bilan tant sur le plan technique que personnel. Le rapport se conclut par une synthèse des principaux apports du stage et des perspectives d'évolution pour les outils développés.

Cette introduction a permis de poser le cadre général du stage ainsi que les enjeux liés à Geotrek et OpenStreetMap. Nous allons désormais revenir plus en détail sur le contexte du stage, ses objectifs, et le positionnement scientifique du sujet.

CHAPITRE 2. CONTEXTE DU STAGE, OBJECTIFS ET POSITIONNEMENT TECHNIQUE

1. Contexte du stage et rôle du projet au sein de Makina Corpus

Mon stage s'inscrivait dans la thématique du partage de données cartographiques entre deux applications open source : Geotrek et OpenStreetMap.

Geotrek est une application de gestion et de valorisation des territoires naturels, développée à la suite d'un appel d'offres public en 2012. La société Makina Corpus a été chargée de son développement initial, mais le projet a depuis évolué. Aujourd'hui, Geotrek est utilisé par de nombreuses institutions publiques, chacune l'employant pour leurs propres objectifs. Les parcs nationaux et régionaux, comme le Parc national des Écrins ou le Parc national des Cévennes, s'en servent principalement pour la gestion de sentiers et la sensibilisation du public à la préservation de la biodiversité. Les départements, par exemple le département du Jura, emploient Geotrek pour optimiser la gestion des chemins, réduire les coûts d'entretien et valoriser les espaces naturels. Les offices de tourisme, tels que l'Office de tourisme du Grand Carcassonne, utilisent la plateforme pour promouvoir les activités de plein air et les randonnées. Enfin, les Établissements Publics de Coopération Intercommunale (EPCI), comme la Région Auvergne-Rhône-Alpes, s'appuient sur Geotrek pour coordonner les actions entre les différentes structures publiques de leur territoire.

Geotrek est composé de quatre modules principaux : Geotrek-admin, destiné à la gestion des données ; Geotrek-rando, un portail web personnalisable ; Geotrek-widget, un module léger intégrable sur tout site web ; et Geotrek-mobile, une application mobile également personnalisable. Le module Geotrek-admin constitue le cœur du système. Il est divisé en seize sous-modules, répartis en deux grandes catégories : la gestion du territoire et la valorisation du territoire.

Le tableau suivant présente les modules de gestion du territoire (cf Table 1).

Module	**Définition**
Tronçon	Réseau de sentiers et chemins structurant le territoire.
Sentier	Groupement de tronçons associé pour des raisons de maintenance.
Statut	Informations complémentaires sur les tronçons : statut foncier, type physique, organisme gestionnaire, etc.
Aménagement	Équipements et mobiliers installés sur le territoire (par exemple : bancs, passerelles).
Signalétique	Panneaux d'information, de guidage ou réglementaires à destination des randonneurs.
Intervention	Travaux d'entretien programmés, en cours ou achevés.
Chantier	Regroupement d'interventions assorti de données administratives.

Tableau 1 – Modules de gestion du territoire

La valorisation du territoire repose sur d'autres modules présentés dans le tableau suivant (cf Table 2).

Module	Définition
Itinéraire	Randonnées destinées au grand public.
Point d'intérêt (POI)	Lieu remarquable naturel ou historique situé à proximité d'un itinéraire.
Service	Informations pratiques liées aux randonnées (points d'eau, passages délicats, etc.).
Contenu touristique	Services touristiques commerciaux (restaurants, artisans, etc.).
Événement touristique	Animations ponctuelles organisées sur le territoire (expositions, conférences).
Signalement	Problèmes signalés par les usagers sur le territoire (par exemple, un arbre tombé sur un sentier).
Zone sensible	Zones protégées pour des raisons environnementales ou patrimoniales.
Site outdoor	Sites de pratique d'activités de plein air (sites d'escalade, zones de vol libre, etc.).
Parcours outdoor	Parcours spécifiques au sein de sites outdoor (voie d'escalade, parcours de kayak, etc.).

Tableau 2 – Modules de valorisation du territoire

Les données saisies dans Geotrek-admin alimentent les modules de communication publique, permettant la publication des informations touristiques et environnementales auprès des utilisateurs.

Plusieurs modules de Geotrek reposent sur un système de référencement dynamique pour la localisation des objets géographiques. Plutôt que de stocker des coordonnées géographiques fixes, le référencement dynamique définit les objets en fonction de leur position relative aux tronçons. Cette méthode consiste à associer un objet à un ou plusieurs tronçons, à préciser le pourcentage de début et de fin de son positionnement sur ces tronçons, ainsi qu'un éventuel décalage (offset) (Makina Corpus, 2023). La figure 1 ci-dessous illustre le fonctionnement de la segmentation dynamique.

Figure 1 - Illustration du fonctionnement du référencement dynamique

Tronçon	Début	Fin	Offset
P1	10%	80%	0m
P1	95%	95%	30m

Tableau 3 - Exemples de définition d'emplacement avec du référencement linéaire

Le référencement dynamique présente des avantages importants, notamment la réduction du volume de stockage des données et la facilité de recherche d'objets voisins. Par exemple, il devient aisé de repérer les points d'intérêt proches d'un itinéraire de randonnée. Toutefois, cette approche présente également plusieurs limites. Toute modification d'un tronçon entraîne le déplacement des objets qui lui sont associés, ce qui peut poser problème pour des entités fixes telles qu'un château. De plus, l'importation de données linéaires provenant d'autres sources est rendue difficile par l'absence d'un référentiel de tronçons commun. De la même manière, l'agrégation de données entre différentes instances de Geotrek peut devenir complexe si les réseaux de tronçons divergent. L'usage du référencement dynamique est donc sujet à débat au sein de la communauté Geotrek.

Enfin, Geotrek-admin est conçu pour interagir avec de nombreux systèmes d'information tiers. Certaines données sont importées automatiquement depuis des Systèmes d'Information Touristique (SIT) tels qu'Apidae ou Tourinsoft, afin de limiter la ressaisie par les utilisateurs. Geotrek peut également exporter ses données vers des plateformes de diffusion comme Cirkwi, IGNrando ou VisoRando. Ces échanges sont rendus possibles par l'utilisation de parsers, outils chargés de convertir les formats de données pour garantir leur compatibilité entre applications. Deux types de parsers coexistent : les parsers d'importation, qui permettent d'intégrer des données externes dans Geotrek, et les parsers d'exportation, qui assurent la publication des données Geotrek vers des plateformes tierces. Cette capacité d'interconnexion est essentielle pour accroître la visibilité des territoires et mutualiser les efforts de valorisation touristique.

Créée en 2004 par Steve Coast, OpenStreetMap (OSM) est une base de données cartographique collaborative et open source. Son objectif est de permettre à chacun de contribuer à la cartographie du monde entier, en ajoutant des informations géographiques telles qu'un banc, un restaurant, un sentier ou encore une frontière administrative. Avec plus de 10 millions de contributeurs à travers le monde, OpenStreetMap est aujourd'hui la plus grande base de données cartographique libre existante.

Le fonctionnement d'OpenStreetMap repose sur une structure de données simple, composée de trois types d'objets principaux : les *nodes*, les *ways* et les *relations*. Un *node* est un point défini par une paire de coordonnées géographiques (latitude et longitude). Il peut représenter un élément ponctuel comme un arbre, une fontaine ou un sommet. Plusieurs *nodes* peuvent être assemblés pour former un *way*, qui désigne une ligne ou un polygone. Un *way* linéaire peut représenter une route, une rivière ou un sentier, tandis qu'un *way* fermé (dont le premier et le dernier *nodes* sont identiques) permet de définir des surfaces comme un bâtiment, une forêt ou une région. Enfin, les *relations* sont des structures qui regroupent plusieurs *nodes* et *ways* ayant un lien logique. Elles servent notamment à représenter des objets complexes ou de grande taille, comme les limites administratives, les itinéraires de transport ou les zones protégées.

Chaque objet dans OSM est enrichi à l'aide de *tags*, qui sont des paires clé-valeur décrivant ses caractéristiques. Par exemple, le tag building=house indique qu'un polygone représente une maison, tandis que tourism=alpine_hut signale un refuge de montagne. Un même objet peut comporter plusieurs tags afin de décrire au mieux sa nature. Même si l'usage des tags est libre,

la communauté s'accorde sur des conventions partagées pour favoriser l'harmonisation des données à l'échelle mondiale.

Pour consulter ou modifier les données d'OpenStreetMap, plusieurs interfaces de programmation (API) sont disponibles. L'API principale, dite API v0.6, permet d'ajouter, de modifier ou de supprimer des objets. Pour l'extraction de données, l'API Overpass est la plus couramment utilisée. Elle permet de filtrer les données en fonction de critères comme une zone géographique, des tags ou des identifiants. Ces interfaces rendent possible l'intégration d'OpenStreetMap dans de nombreuses applications, notamment des outils de navigation comme Komoot ou Valhalla (OpenStreetMap, 2024).

Une fois le rôle du projet clarifié au sein de Makina Corpus, il convient à présent de s'intéresser aux objectifs précis qui ont guidé les travaux réalisés durant ce stage.

2. Objectifs initiaux

Au départ, les objectifs de mon stage n'étaient pas précisément définis. Le sujet qui m'a été confié par Makina Corpus portait sur une thématique générale : faciliter les échanges de données entre Geotrek et OpenStreetMap. Cette mission, initiée en interne, répondait à un besoin exprimé de longue date par l'entreprise, mais resté en suspens faute de ressources disponibles.

Mon premier objectif a donc été d'effectuer un travail de recherche documentaire approfondi sur les deux outils. Il s'agissait d'identifier les objets manipulés dans Geotrek et d'en analyser la structure en base de données, puis d'étudier les principes de modélisation des données dans OpenStreetMap. L'objectif final de cette phase exploratoire était de définir, pour chaque type d'objet dans Geotrek, les correspondances possibles avec les éléments d'OpenStreetMap en identifiant les tags appropriés.

À l'issue de cette phase d'analyse et après plusieurs échanges avec mes tuteurs de stage, deux objectifs principaux ont été formalisés.

Le premier objectif consistait à permettre l'import des données depuis OpenStreetMap vers Geotrek. Ce système devait être capable à la fois de peupler la base de données lors de la première utilisation et de proposer un mécanisme de mise à jour en cas de modification dans OpenStreetMap. Pour répondre aux besoins variés des utilisateurs de Geotrek, l'import devait être personnalisable et adapté à chaque type d'objet. En effet, contrairement à OpenStreetMap qui repose sur un modèle unique, Geotrek utilise différentes structures selon les objets. L'import devait donc être pensé de manière modulaire et flexible.

Le second objectif, plus exploratoire, visait à permettre l'export des données de Geotrek vers OpenStreetMap. Cet objectif a émergé au cours du stage, à la suite d'un sondage mené auprès de la communauté Geotrek. Il ne s'agissait pas d'un cahier des charges figé, mais plutôt d'un projet à cadrer.

Ces objectifs, bien que définis progressivement, s'inscrivent dans des enjeux techniques plus larges, qu'il est important de mieux comprendre avant d'aborder les développements réalisés.

3. Enjeux techniques et positionnement scientifique du sujet

Mon stage se positionne autour de deux domaines scientifiques : l'interopérabilité des données et les données géographiques.
L'interopérabilité des données désigne la capacité de deux systèmes à fonctionner ensemble et à s'échanger des informations. Il existe différentes familles d'approches selon que l'échange de données soit manuel ou automatique, ponctuel ou récurrent. Le choix de l'approche dépend de l'usage et de la philosophie des systèmes concernés. Par exemple, des fichiers tels que GeoJSON, ShapeFile ou XML permettent d'importer ponctuellement des données dans un logiciel de manière manuelle. À l'inverse, l'utilisation d'une API permet de récupérer un flux de données de façon automatisée et récurrente. La quantité de données à échanger peut aussi varier considérablement, allant de quelques dizaines à plusieurs milliers d'éléments.
Dans le cadre de Geotrek, les données sont importées à l'aide de *parsers* qui prennent en entrée soit un fichier, soit une API, et adaptent ensuite les données au format attendu par Geotrek. Par ailleurs, une API permet également d'exposer les données à l'extérieur de l'application.

Le second domaine scientifique abordé concerne les données géographiques, qui représentent des objets localisés sur la surface terrestre. Ces données nécessitent la gestion de leur position et de leur géométrie. Deux grandes approches existent : les géométries absolues et les géométries relatives.
Les géométries absolues stockent les coordonnées de chaque point constituant l'objet, comme dans le modèle « Simple Features » (point, linestring, polygon, multipolygon, etc.). À l'inverse, une géométrie relative est définie en fonction d'un autre objet, comme c'est le cas dans le référencement linéaire.
On distingue également deux manières de définir un point sur la Terre : les systèmes de coordonnées, comme WGS84 (utilisé par OpenStreetMap), qui utilisent la latitude et la longitude, et les systèmes de projection, comme Lambert-93 (utilisé en France, notamment dans Geotrek), qui convertissent la surface de la Terre en un plan. Les systèmes de projection permettent des calculs de distance plus précis, mais introduisent des déformations et sont donc valables uniquement localement. Chaque système est identifié par un SRID (Spatial Reference ID), tel que 4326 pour WGS84 et 2154 pour Lambert-93.

Durant mon stage, j'ai travaillé sur Geotrek_admin, une application dédiée à la gestion et à la valorisation des territoires. Ses utilisateurs sont principalement des structures publiques : parcs régionaux et nationaux, communautés de communes, régions, offices de tourisme… Comme toutes les organisations, ces structures font face à des contraintes de temps. Ainsi, l'une des premières problématiques est de réduire le temps de saisie des informations, tant pour la gestion que pour la valorisation du territoire.
Par ailleurs, la mise en valeur du territoire passe par sa visibilité sur plusieurs plateformes. L'export des données vers OpenStreetMap permettrait ainsi de diffuser les itinéraires de randonnée et les points d'intérêt non seulement dans l'application, mais aussi dans des services tiers. L'import de données vers OpenStreetMap permet de bénéficier des nombreuses applications utilisant sa base de donnée comme Komoot, une application de randonnée utilisée par près de 40 millions de personnes.
Enfin, les applications de gestion territoriale manipulent souvent un grand volume de données. L'un des enjeux est donc l'initialisation de l'application avec des flux de données de qualité, une opération généralement longue et chronophage, car elle nécessite de trouver des flux adaptés pour chaque type d'objet à intégrer.

Geotrek-admin est une application web développée avec le framework Django. Django, basé sur Python, repose sur le modèle MVT : *Modèle, Vue, Templates*. Les *modèles* correspondent aux tables de la base de données, les *vues* aux pages web de l'application, et les *templates* aux

pages HTML pouvant être alimentées dynamiquement. Ce modèle permet de bien séparer les données et leur traitement, renforçant ainsi la sécurité. Django propose également de nombreux outils facilitant la création d'une application web : interface d'administration, système d'authentification, tests unitaires… Il intègre aussi un framework dédié à la création d'API : Django REST Framework.
La base de données repose sur PostgreSQL, un système de base de donnée relationnel open-source, enrichi par l'extension PostGIS pour le traitement des données géographiques. Les pages HTML sont mises en forme avec CSS et JavaScript. L'application est versionnée via GitHub. Pour garantir une intégration continue, des *workflows* GitHub ont été mis en place pour effectuer une analyse statique du code et vérifier l'exécution des tests unitaires. Enfin, l'application est découpée en plusieurs conteneurs Docker synchronisés avec Docker Compose. L'utilisation de Docker permet une installation facilitée et compatible avec de nombreux environnements.

Ce stage m'a permis de mobiliser plusieurs notions vues lors de ma scolarité à l'INSA. Tout d'abord, étant donné que Geotrek-admin est un projet open-source, les tests unitaires sont essentiels pour garantir que les modifications apportées n'introduisent pas de dysfonctionnements ou de régressions. Il est donc indispensable de couvrir l'ensemble du code avec des tests, ce qui m'a conduit à en écrire pour chaque nouvelle fonctionnalité. Pour cela, j'ai mobilisé les notions abordées en cours de vérification de programme en 5e année. Ensuite, le cours de base de données en 3e année m'a permis de comprendre rapidement la structure de la base de données de Geotrek-admin ainsi que les diagrammes de classes associés. Il m'a également été utile pour explorer les données directement dans la base. Enfin, les compétences acquises en Python lors des cours de machine learning (4e et 5e années) ainsi qu'en robotique humanoïde (5e année) m'ont permis de prendre rapidement en main le framework Django.

Ce positionnement scientifique a orienté l'ensemble des choix techniques. Nous allons maintenant entrer dans le détail des solutions développées pour répondre concrètement aux besoins identifiés.

CHAPITRE 3 : DEVELOPPEMENTS REALISES POUR L'INTEGRATION OSM–GEOTREK

1. Développement d'un système d'import des données OpenStreetMap

La première tâche de développement qui m'a été confiée fut de créer des outils permettant d'importer des objets issus d'OpenStreetMap dans Geotrek.
Lors de l'étude documentaire que j'ai menée au cours du premier mois de stage, j'ai recensé les différents objets de Geotrek, analysé leur modèle en base de données et étudié leur correspondance avec les entités d'OpenStreetMap.

La table 4 ci-dessous résume les principales informations concernant les objets Geotrek. Le détail complet des modèles est disponible en annexe B.

Objet Geotrek	Géométrie Geotrek	Géométrie OpenStreetMap	Référencement linéaire	Problématiques
Aménagement	point, linestring	node, way, relation	oui	Conversion des géométries de OSM en point
Signalétique	point	node	oui	___
Point d'intérêt (POI)	point	node, way, relation	oui	Conversion des géométries de OSM en point
Contenu Touristique	point, linestring, polygon	node, way, relation	non	Manipulation des relations
Territoire administratif	multipolygon	relation	non	Manipulation des relations
Commune	multipolygon	relation	non	Manipulation des relations
Zone restreinte	multipolygon	relation	non	Manipulation des relations
Bureau d'information	point	node, way, relation	non	Conversion des géométries de OSM en point
Site outdoor	geometrycollection(point, linestring)	node, way, relation	non	Manipulation des relations, Gestion des hiérarchies

Tableau 4 - Structure des objets Geotrek et correspondances OSM

On constate dans ce tableau quatre principales difficultés le référencement linéaire, la conversion des géométries (nodes, ways, relations), la manipulation des relations OSM, et la versatilité des tags OSM.

Cette dernière problématique est commune à l'ensemble des objets. En effet, la structure souple d'OpenStreetMap permet à un même type d'objet d'être défini de manière sommaire, parfois avec un seul tag. Par exemple, un point d'information peut être identifié uniquement par le tag information=office, ou au contraire contenir un grand nombre de métadonnées sur l'établissement. Cette hétérogénéité complique le traitement automatisé, notamment lorsque

des champs obligatoires dans Geotrek ne peuvent pas être complétés faute d'informations dans OSM.Ma première mission fût donc de trouver l'outil permettant de résoudre toutes ces problématiques.

Conception des parsers d'import OpenStreetMap pour Geotrek

Dans Geotrek, un *parser* est un outil permettant d'importer des données depuis une source externe vers un modèle de données interne. Il a donc semblé naturel d'utiliser ce mécanisme pour intégrer les objets d'OpenStreetMap dans Geotrek.

Tous les parsers de Geotrek héritent d'une classe mère Parser. Celle-ci est conçue pour être très générique : elle prend en charge des sources variées (fichiers XML, Shapefiles, API externes, etc.) grâce à un système de paramètres de configuration (cf. annexe D) qui permettent d'adapter le comportement du parser à chaque usage.

Le fonctionnement de cette classe suit une séquence définie (cf. annexe C). Après récupération des données (fichier ou API), chaque objet est traité individuellement via la méthode parse_obj. Celle-ci initialise un objet Python (self.obj) correspondant à une instance du modèle ciblé : soit existante (selon l'identifiant externe eid et le fournisseur provider), soit nouvellement créée.

La méthode parse_fields est ensuite appelée. Elle s'appuie sur deux dictionnaires de correspondance (fields et constant_fields). Les champs constants sont renseignés directement, tandis que les autres sont extraits de la source via la fonction get_val. Celle-ci peut retourner une ou plusieurs valeurs.

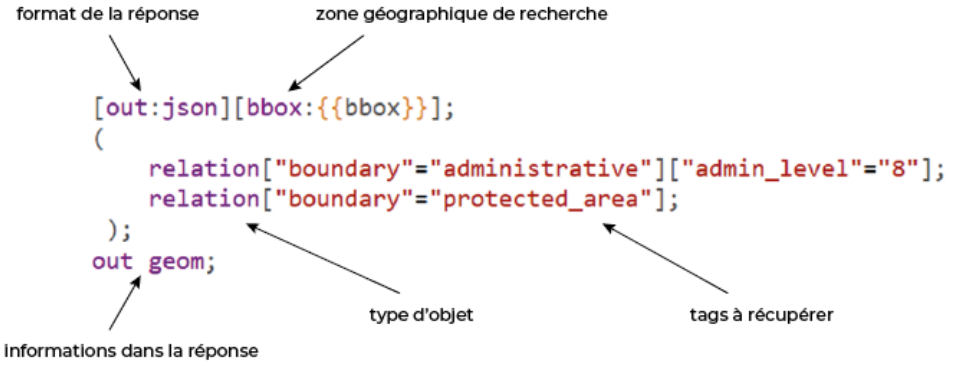
Une fois les valeurs extraites, elles sont transformées pour correspondre au format attendu. Les filtres utilisés permettent soit d'associer l'objet aux éléments existants en base lorsqu'il s'agit de clés étrangères, soit d'adapter plus finement les données via des fonctions personnalisées propres à chaque champ.

Après traitement, les données sont affectées à l'objet, qui est sauvegardé. Enfin, les relations complexes (clés étrangères, relations many-to-many, champs non standards) sont renseignées.

Pour intégrer les objets d'OSM, j'ai créé une classe de base OpenStreetMapParser, chargée de l'interrogation de l'API Overpass. Cette API est spécifiquement conçue pour l'extraction de données depuis la base OSM. Elle utilise un langage de requête dédié : Overpass QL, qui permet de filtrer les objets selon leurs tags, leur type (node, way, relation), une zone géographique…

Voici un exemple de requête Overpass (figure 2), permettant d'extraire toutes les relations possédant soit les tags boundary=administrative **et** admin_level=8, soit le tag boundary=protected_area.

Figure 2 - Commande Overpass QL



Pour rendre la configuration des requêtes accessible, j'ai mis en place une structure simple et lisible dans les parsers. La zone géographique est automatiquement définie à partir du paramètre SPATIAL_EXTENT de l'instance Geotrek, avec la possibilité d'ajouter une marge. L'attribut query_settings permet de préciser le type d'objet à interroger, le format de sortie attendu, ainsi que la marge éventuelle. L'attribut tags, quant à lui, permet de définir les conditions de filtrage sous forme d'une liste. L'attribut tags permet de définir les filtres à appliquer sur les objets OpenStreetMap interrogés via l'API Overpass. Il s'agit d'une liste dont chaque élément peut être soit un dictionnaire, représentant un filtre unique sur un, soit une liste de dictionnaires, exprimant une conjonction logique (ET) entre plusieurs tags . Une disjonction logique (OU) est ensuite appliquée entre tous les éléments de la liste principale. La figure 3 montre comment exprimer la requête de la figure 2 avec l'attribut tags. Cette structure rend la formulation des requêtes plus intuitive tout en restant fidèle à la logique d'Overpass QL.

Figure 3 - Spécification de tags pour la requête de la figure 2

```
tags = [
    [{"boundary": "administrative"}, {"admin_level": "8"}],
    {"boundary": "protected_area"}
]
```

Une méthode dédiée se charge de convertir cette structure Python en requête Overpass QL valide. Cette approche rend l'écriture des conditions compréhensible et modifiable sans connaissance du langage Overpass.

Une fois la requête exécutée, la réponse est récupérée au format JSON. La méthode next_row retourne ensuite chaque objet un à un à la classe Parser pour traitement.

À partir de la classe OpenStreetMapParser, j'ai développé un parser spécifique pour chaque type d'objet Geotrek concerné : Aménagement, Signalétique, POI, Contenus touristiques, Territoires administratifs, Zones restreintes, Bureaux d'information et Sites outdoor. Chaque parser est défini dans l'application correspondante, associe les bons champs source et cible, et applique les filtres personnalisés nécessaires.

Enfin, pour valider le bon fonctionnement de l'ensemble, j'ai effectué des tests en conditions réelles et écrit des tests unitaires couvrant tous les cas d'usage.

La mise en place des parsers a constitué la base de l'import depuis OpenStreetMap. Il a ensuite été nécessaire d'adapter leur comportement pour faire face à la variabilité et à l'incomplétude des données OSM.

Gestion des valeurs manquantes et attribution de noms par défaut

L'un des défis majeurs posés par les données OpenStreetMap réside dans l'hétérogénéité de leur niveau de définition. Un même type d'objet peut comporter des informations très variables selon les contributeurs. Par exemple, un parking à vélo peut se limiter à un simple tag amenity=bicycle_parking ou détailler le type de fixation et le nombre de places. Cette variabilité complique l'import automatique vers Geotrek, notamment lorsque certains champs du modèle sont obligatoires, comme le nom pour les points d'intérêt, alors que le tag name est souvent absent dans OpenStreetMap.

Pour remédier à cela, une fonctionnalité de **valeurs par défaut** a été ajoutée à la classe Parser, utilisée comme base par tous les parsers de Geotrek. Elle permet d'attribuer automatiquement une valeur fixe à un champ si celui-ci n'est pas renseigné dans les données sources. Cette valeur est définie via l'attribut default_fields_value, un dictionnaire associant à chaque champ concerné une valeur par défaut. Ce mécanisme s'applique uniquement aux champs classiques déclarés dans fields ou constant_fields, car les champs de type relationnel (many-to-many ou non_fields) ne sont pas obligatoires et ne nécessitent pas de valeur par défaut.

L'attribution de cette valeur est effectuée dans la méthode parse_real_fields, juste après l'extraction des données de la source mais avant l'application des filtres. Ce choix permet de garantir que la valeur par défaut sera elle aussi soumise à d'éventuelles transformations par les filtres, comme s'il s'agissait d'une donnée normale.

Cependant, pour fonctionner correctement avec les données OpenStreetMap, il a fallu adapter le comportement du parser en cas d'absence d'un tag. Alors que les autres parsers de Geotrek s'appuient sur des schémas de données fixes et lèvent une erreur (KeyError) lorsqu'un champ est manquant, cette réaction n'est pas adaptée à OSM, où l'absence d'un tag est courante et non anormale.

Pour cela, un attribut flexible_fields a été introduit. Lorsqu'il est activé (valeur à True), la méthode get_part utilise la méthode .get() sur les réponses API au lieu de crochets, ce qui évite de lever une exception. Cela permet de continuer le traitement normalement même en l'absence de certaines données, et de recourir aux valeurs par défaut si besoin.

Cette approche a permis de rendre l'import plus robuste face à la flexibilité des données OSM, notamment en garantissant la création d'objets valides même lorsque des champs requis sont absents dans la source.

Ce mécanisme a permis de sécuriser l'import en garantissant la validité des objets créés, même en l'absence d'informations complètes. Il restait toutefois à résoudre les problématiques liées à la gestion des géométries.

Conversion des géométries OSM en points compatibles avec Geotrek

Comme le montre la table 4 de l'introduction, plusieurs modèles de Geotrek tels que les points d'intérêt, la signalétique, les bureaux d'information et les aménagements s'appuient sur des géométries ponctuelles. Cela s'explique notamment par l'usage du référencement dynamique dans Geotrek, qui ne permet pas d'associer simplement des géométries linéaires ou surfaciques importées à des objets. Pour cette raison, les objets OpenStreetMap sont systématiquement convertis en points, ce qui suffit à localiser les entités avec précision, sans nécessiter de géométrie complète.

L'API Overpass permet de récupérer la géométrie des objets OSM. Les nodes contiennent directement leur latitude et leur longitude. Les ways sont représentés comme des listes ordonnées de points. Les relations sont des structures plus complexes composées de nodes et de ways ; elles peuvent modéliser des géométries de grande taille ou de forme composite, comme des multipolygones. Tous ces objets sont exprimés dans le système géographique WGS84.

À l'inverse, Geotrek repose sur les objets GEOSGeometry fournis par Django, qui prennent en charge différents types de géométries (Point, LineString, Polygon, etc.) et utilisent une projection métrique définie dans la variable SRID de l'instance. Toutes les géométries importées sont donc reprojetées à l'aide de la méthode transform pour correspondre à ce système.

Pour les nodes, la transformation est directe car un objet Point peut être créé à partir des coordonnées.

Pour les ways, deux cas sont distingués. Si le way est fermé (premier et dernier point identiques), il est interprété comme un polygone, et son centroïde est utilisé comme point de référence. Sinon, le way est considéré comme un linéaire, et un point est extrait à l'aide de la méthode interpolate_normalize(0.5), qui renvoie un point situé à 50 % de la longueur du segment. Ce choix garantit une position stable et reproductible, à la différence de la fonction point_on_surface, dont le comportement peut varier selon les versions de GEOS installées.

Pour les relations, dont la structure est plus hétérogène, l'approche retenue consiste à utiliser la bounding box renvoyée par l'API. Le centroïde de cette boîte est alors utilisé comme point géographique. Bien que ce point puisse se situer en dehors de la géométrie réelle, cette méthode a l'avantage d'être robuste et reproductible quel que soit l'environnement d'exécution.

Tous les parsers concernés, POI, signalétique, bureau d'information et aménagement, appliquent cette logique. Le cas des aménagements constitue une exception notable : bien qu'ils puissent, en théorie, stocker des linéaires, la contrainte du référencement dynamique impose également ici une réduction à un simple point.

Cette conversion systématique permet de garantir l'intégration homogène des objets OSM dans Geotrek, tout en s'adaptant à la diversité des géométries décrites dans OpenStreetMap.

Une fois les géométries adaptées aux contraintes de Geotrek, il a fallu s'atteler à la gestion des cas les plus complexes : les relations multipolygonales dans OpenStreetMap.

Traitement des relations complexes dans OpenStreetMap

Dans OpenStreetMap, les objets géographiques complexes, comme les zones avec plusieurs contours ou des trous, sont représentés par des relations de type *multipolygon*. Chaque relation est composée de chemins (ways) auxquels sont assignés des rôles : *outer* pour les contours externes et *inner* pour les zones à exclure (par exemple, une clairière au sein d'une forêt). Ce modèle permet de décrire des entités géographiques complexes, mais pose un défi important lors de la reconstruction des géométries dans un système comme Geotrek, qui utilise GEOS, le moteur géométrique de Django (Boeing, 2017). GEOS exige des géométries valides, ce qui suppose d'associer correctement chaque *inner* à un *outer*, une information qu'OSM ne fournit pas explicitement. Il faut alors analyser la géométrie elle-même, ce qui rend l'opération coûteuse en calcul et parfois incertaine.

Pour éviter cette reconstruction manuelle, deux services externes ont été explorés : Nominatim et Polygon.openstreetmap. Nominatim, bien que principalement utilisé pour la géocodification d'adresses, offre une fonctionnalité de récupération de géométrie à partir d'un identifiant OSM via l'API lookup. Cette API permet d'obtenir directement la géométrie au format WKT, ce qui est compatible avec GEOS. Cependant, elle ne couvre essentiellement que les zones administratives, ce qui la rend adaptée uniquement aux parsers des villes, des territoires administratifs et des zones restreintes, où une géométrie de type *MultiPolygon* est strictement requise.

Pour les contenus touristiques et les sites outdoor, qui utilisent des géométries plus variées, c'est l'API Polygon.openstreetmap qui a été retenue. Conçue pour fournir les géométries des relations OSM, elle retourne les données dans plusieurs formats, dont WKT et GeoJSON. Le format WKT a été privilégié car il est plus léger et directement interprétable par GEOS sans transformation supplémentaire. Par exemple, pour un même polygone, un fichier WKT pèse environ 600 octets contre plus de 1,4 ko pour sa version GeoJSON, ce qui permet de limiter la charge des échanges.

Cependant, l'API Polygon.openstreetmap étant collaborative, elle présente parfois des géométries incorrectes ou incomplètes. Pour les contenus touristiques et les sites outdoor, cette limite est contournée en traitant individuellement chaque membre de la relation : les *nodes* sont convertis en points, les *ways* ouverts en lignes et les *ways* fermés en polygones. Cette approche garantit une récupération partielle mais fiable des géométries, sans bloquer l'import. En revanche, pour les objets administratifs, où la géométrie complète est obligatoire, seul Nominatim est utilisé afin de garantir la validité des *MultiPolygons* requis par la base Geotrek.

Ce traitement a permis d'intégrer correctement les géométries structurées. D'autres adaptations ont été nécessaires, notamment pour modifier certains modèles de données internes à Geotrek.

Refonte du modèle de données des communes dans Geotrek

Dans Geotrek, le modèle des villes reposait initialement sur un champ code servant de clé primaire. Ce champ, censé contenir le code INSEE en France, posait deux problèmes majeurs. D'une part, son utilisation comme identifiant unique imposait qu'il soit toujours renseigné, ce qui n'est pas garanti par les données OpenStreetMap, notamment hors de France où aucun équivalent universel n'existe. D'autre part, cette structure empêchait toute modification ultérieure du code, limitant la souplesse du modèle et freinant l'ouverture internationale de Geotrek.

Afin de permettre l'import de données OSM et de rendre le modèle compatible avec des contextes internationaux, une refonte du modèle a été engagée. Le champ code a été conservé mais rendu facultatif, tandis qu'un nouveau champ id auto-incrémenté a été introduit en tant que nouvelle clé primaire. Cette évolution a nécessité une migration Django, outil standard permettant de transformer la structure des modèles et de propager ces changements sur toutes les instances Geotrek. Sept étapes ont été nécessaires : création du champ id en tant qu'entier, remplissage de ce champ pour les objets existants, suppression de la contrainte de clé primaire sur code, ajout des contraintes unique, blank et null à ce même champ, suppression de l'index automatique lié à l'ancienne clé primaire, conversion du champ id en AutoField avec attribution de la clé primaire, puis mise à jour de la séquence d'auto-incrémentation pour éviter les collisions avec les données déjà présentes.

Ce changement de structure a naturellement impacté les API de Geotrek. Pour l'API mobile, seule l'ajout du champ id au endpoint city a été nécessaire. En revanche, l'API v2, utilisée par Geotrek-rando et Geotrek-widget, a dû être largement revue. Les filtres basés sur le champ code ont été remplacés par des filtres sur l'id, notamment pour les endpoints city, outdoor_course, outdoor_site, tour, touristiccontent, touristicevent et trek. De plus, ces mêmes endpoints retournaient auparavant le code des villes dans leurs réponses, qui ont été converties pour inclure désormais l'ID, assurant une cohérence dans l'identification des villes.

Ces modifications côté API ont eu un effet en cascade sur les clients Geotrek-rando et Geotrek-widget. Ces outils s'appuyaient jusque-là sur le code pour des fonctionnalités comme l'affichage du widget météo. Pour préserver la compatibilité avec les anciennes versions, un mécanisme conditionnel a été introduit : si Geotrek-admin fournit un champ code, celui-ci est utilisé, sinon l'ancien champ id (contenant le code dans les anciennes versions de l'API) est conservé. Ce compromis permet d'assurer une transition fluide entre les versions, tout en posant les bases d'un modèle plus souple et internationalisé.

Cette refonte a ouvert la voie à une plus grande flexibilité et à une compatibilité internationale. Elle a également nécessité des ajustements au niveau des API et de leur intégration avec les clients.

Prise en compte des traductions dans les champs importés

Geotrek est conçu pour s'internationaliser, c'est pourquoi un système de traduction des champs a été mis en place. Lors de la création d'une nouvelle instance, les clients peuvent choisir les langues supportées ainsi que la langue par défaut, configurées via les paramètres MODELTRANSLATION_LANGUAGES et MODELTRANSLATION_DEFAULT_LANGUAGE. À la création de la base de données, des champs de traduction sont automatiquement générés pour les champs concernés, portant le nom du champ suivi d'un underscore et du code langue. Un alias sans indication de langue est créé pour la langue par défaut, afin d'améliorer la lisibilité du code.

OpenStreetMap intègre également un système de traduction : certains tags peuvent porter un suffixe avec un code langue, comme name:fr ou name:en, tandis que le tag sans suffixe est censé être dans la langue locale, même si cette convention n'est pas toujours respectée. Pour garantir la bonne prise en compte des traductions, notamment pour la langue par défaut, Geotrek récupère soit la valeur avec un tag explicite correspondant à la langue, soit la valeur du tag par défaut.

Un système automatique d'association des champs traduits se déclenche au début du processus d'import. Il récupère la liste des champs traduits du modèle, les langues configurées et la langue par défaut. Pour chaque champ traduit, il associe les tags OSM correspondants en ajoutant le suffixe du code langue. Pour la langue par défaut, l'association se fait à la fois avec le tag traduit et le tag non traduit, garantissant ainsi la priorité à la bonne langue tout en permettant de récupérer une valeur même si la traduction manque, ce qui est fréquent dans les données OSM.

Enfin, comme le champ de la langue par défaut peut contenir une liste de valeurs, une surcharge de la fonction get_val a été mise en place pour retourner la première valeur non nulle de la liste lorsque aucun filtre spécifique n'est appliqué. Cette approche évite la création de filtres dédiés pour chaque champ et répond aux besoins de flexibilité propres aux tags OSM.

Ce système fonctionne exclusivement pour les parsers OpenStreetMap et assure une gestion cohérente des traductions directement à partir des données sources.

La gestion multilingue renforce la portée internationale de l'outil. Il restait enfin à enrichir les objets avec des fichiers médias, pour offrir une valorisation plus complète des contenus.

Import d'images et enrichissement des objets avec fichiers liés

Les objets de Geotrek peuvent être associé à des fichiers comme des photographies afin d'étayer la description. Pour les parsers il existe une classe AttachmentsParserMixin dont les parsers peuvent hérités pour importer des fichiers liés. Cette classe possède une fonction save_attachments qui est appelé dans la fonction parse_non_field de la classe Parser. Pour résumer son fonctionnement, la fonction save_attachments récupère l'url de téléchargement du fichier ainsi de l'auteur, la légende et le titre si ceux-ci sont disponibles. Ensuite, elle télécharge le fichier si l'attribut download_activate est activé. Pour finir, elle sauvegarde les données récupérées dans le modèle Attachments et lie l'objet créé à l'objet importé.

OpenStreetMap possède plusieurs tags permettant de lier des images au objet. Il y a par exemple, wikimedia_commons, image, panoramax et mapillary. Dans le cadre de mon stage nous avons choisit de n'exploiter que wikimedia_commons et image. En effet, panoramax et mapillary proposent des fonctionnalités proche de google street view, ce qui rend la récupération de l'image complexe.

Les tags image doivent contenir un url de l'image à lier à l'objet. Cependant, de nombreux lien sont soit inaccessibles, soit le lien ne mène pas vers une image mais vers une page HTML par exemple. De plus, aucune informations complémentaires n'est présente. Afin, de n'importer que des images il est donc nécessaire de prendre en charge les erreur de téléchargement mais aussi vérifier le type du fichier téléchargé.

Au contraire les tags wikimedia_commons contiennent le nom d'un fichier ou d'une catégorie présent dans wikimedia_commons. Le type de contenu est identifié par un mot clé présent avant le nom, File : pour un fichier et Category : pour une catégorie. Afin de retrouver le lien de téléchargement il est nécessaire de faire appel à une première API. Cette API est accessible

via le lien https://api.wikimedia.org/core/v1/commons/file/ suivit du nom du fichier renseigné dans le tag. Cette API retourne à la fois le lien de téléchargement de l'image en différent format mais aussi l'auteur du fichier et son titre.

Pour le parser d'OpenStreetMap j'ai créé une classe OpenStreetMapAttachementsdParserMixin contenant l'ensemble des configurations et des modification nécessaire au bon fonctionnement de la classe AttachmentsParserMixin pour OpenStreetMap. Comme pour les champs géométrie il a été nécessaire de surcharger la fonction filter_attachments afin de récupérer les informations nécessaire à l'import des images. C'est-à-dire en retournant le lien de téléchargement et si disponible, l'auteur, le titre et la légende. OpenStreetMap possédant deux tags pour associer des images il a fallut séparer les processus de récupération des données. Ainsi, dans un cas un appel API est réalisé pour récupérer les informations tandis que dans l'autre elles sont disponibles de suite.

Pour finir, tous les fichiers référencé dans OpenStreetMap sont sont la licence Creative Commons Attribution Share-Alike 4.0 ou CC BY-SA 4.0. Ainsi, étant donné qu'il existe un champ pouvant stocké la licence mais qu'il n'existait pas de système dans AttachmentsParserMixin pour la renseigner, nous avons décidé d'ajouter un système de licence par défaut. Ainsi, si l'attribut default_license_label est renseigné alors si aucune valeur de licence n'est renseigné pour un fichier alors celle-ci est attribuée.

Grâce à cette fonctionnalité, les objets importés gagnent en expressivité. Il devenait alors pertinent de tester l'ensemble du système en conditions réelles, sur une instance complète.

Validation du système avec une instance test sur le parc du Triglav

Durant mon stage j'ai pu tester mes outils sur une instance Geotrek test. Cette instance couvrait le parc des Triglav en Slovénie.

A partir d'un réseau de tronçon obtenu grâce à l'application que j'ai développé et dont le détail est donné dans la prochaine partie, j'ai put recréer une instance Geotrek complète seulement à partir des données de OpenStreetMap.

Pour cela j'ai écrit des parsers personnalisés héritant des parsers OpenStreetMap. Ces parser permettent notamment de choisir le type des objets importés mais aussi de choisir quels sont les tags de filtres. L'intégralité des parsers personnalisé pour le parc des Triglav en Slovénie est donnée en annexe E.

Comme on peut le voir tous les parsers définissent un label, un provider, une liste de tags et un dictionnaire de valeur par défaut. Ensuite, en fonction de l'objet un type, une catégorie, ou bien une pratique peut être définit.

Ces parsers on permit d'importer 4185 objets. Le tableau 5 ci-dessous détail le nombre d'objet importé pour chaque catégorie d'objet Geotrek.

Type d'objet	**Nombre d'objet importé**
Aménagement	1444
Signalétique	467
POI	1325
Contenu touristique	329
Lieu de renseignement	6
Territoire administratif	19
Zone restreinte	1

Tableau 5 - Répartition des objets importé pour le parc des Triglav

On constate dans ce tableau que les parsers OpenStreetMap sont particulièrement efficace pour les aménagements et les POI. La catégories communes n'est pas présents dans le détails. En effet, ayant modifié l'API v2 et mobile, il me faut attendre que celle-ci soit modifiées avant de pouvoir valider ma pull request sur la modification du modèle City. La modification doit être intégré au produit avant de pouvoir entamer le développement du parser City car la modification de la base de donnée est nécessaire. Ce tableau montre également que 88% des parsers ont été réalisé lors de mon stage.

Les chiffres du tableau montre que l'import de données fonctionne pour la majorité des objets de Geotrek. Cependant, de nombreuse fonctionnalité mise en place ne peuvent pas être visualiser dans ces chiffres. C'est pourquoi les figures 4 et 5 suivantes présentent plus en détails les POI importés. L'instance ne pouvant pas afficher le slovène celle-ci a été mise en anglais afin d'avoir plus de données remontant d'OpenStreetMap.

Figure 4 - Liste des POI importé avec le parser OpenStreetMap

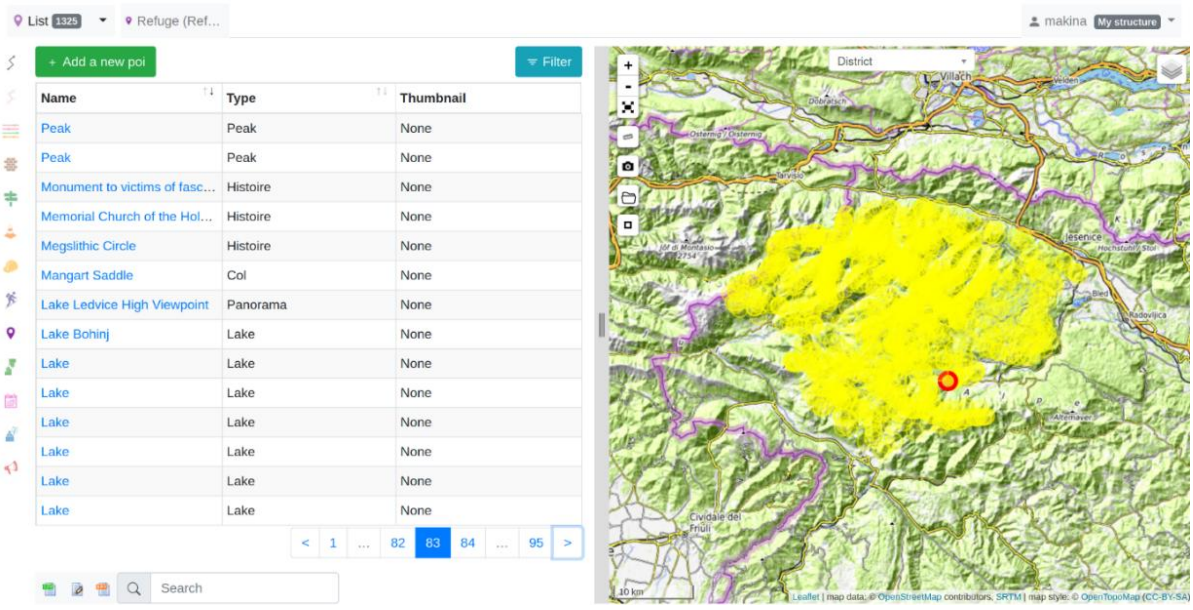
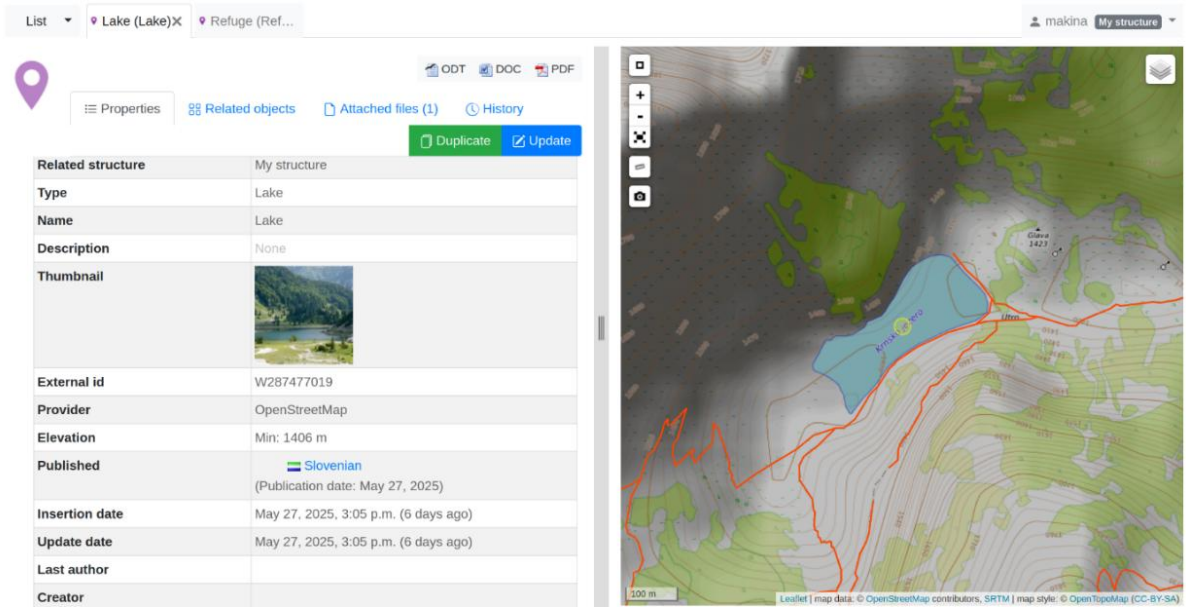


Figure 5 - Fiche détail d'un POI importé avec le parser OpenStreetMap



Sur la figure 4, on peut voir que certain possède le nom par défaut attribué dans les parsers personnalisés (cf Annexe E), tandis que d'autre comme le lac Bohinj possède leur vrai nom. Cela montre l'importance de la mise en place de nom par défaut mais aussi que le parser est en capacité de récupérer les informations des objets d'OpenStreetMap.

Ensuite, la figure 5 met en avant la capacité du parser à lier un fichier au objets Geotrek lorsque celui-ci est disponible. On peut également voir l'identifiant externe « W287477019 ». Cette objet est initialement de type way formant un polygone. Pour l'image associée l'objet possédait un tag image.

Cette exemple avec le parc des Triglav montre qu'il est possible de créer une instance Geotrek à partir des parsers OpenStreetMap.

Ces tests grandeur nature ont confirmé la robustesse des parsers et leur adaptabilité. L'import des objets étant opérationnel, il restait à traiter un cas spécifique : les tronçons, essentiels au fonctionnement du système.

2. Création d'un outil de génération de tronçons à partir d'OpenStreetMap

Contrairement aux autres objets de Geotrek, les tronçons présentent quelques contraintes supplémentaires. En effet, plusieurs règles spécifiques s'appliquent à cet objet afin de garantir le bon fonctionnement du référencement linéaire. Premièrement, les tronçons ne doivent pas s'intersecter. Ainsi, si deux tronçons se croisent, il est nécessaire de les sectionner. Ensuite, les tronçons doivent être soumis à une vérification humaine pour plusieurs raisons. Tout d'abord, afin de supprimer les tronçons inutiles comme les voies sans issue ou les routes non piétonnes, qui risqueraient de surcharger la base de données et l'interface graphique. Pour ces raisons, il a été décidé de ne pas créer un parser comme pour les autres objets.

La problématique majeure de cette mission était de trouver un moyen de convertir les « ways » d'OpenStreetMap en LineString, tout en garantissant que les segments ne s'intersectent pas. Nous nous sommes alors tournés vers OSMnx, un package Python développé par Geoff Boeing pour l'analyse urbaine à partir des données d'OpenStreetMap. L'avantage de cet outil est qu'il permet de convertir les routes et chemins d'OpenStreetMap en graphe. Les nœuds représentent les intersections, et les arcs représentent les routes entre deux intersections. De plus, les graphes conservent la géométrie des arcs sous forme de LineString. Ce package résout ainsi deux problèmes : d'une part, il convertit les « ways » en LineString, et d'autre part, il segmente les routes au niveau des intersections.

Cependant, OSMnx présente deux limites. Premièrement, si deux routes s'intersectent à des niveaux différents, par exemple au niveau d'un pont, OSMnx considère qu'il n'y a pas d'intersection. Or, Geotrek ne gère pas les différences de niveaux, il faut donc segmenter les tronçons manuellement. Cela nécessite une vérification humaine avant l'import dans Geotrek. Il est donc nécessaire de créer un outil dédié aux tronçons plutôt que de l'intégrer à un parser. De plus, OSMnx a près de six dépendances et son installation via pip n'est pas officiellement maintenue. L'intégration dans Geotrek serait donc trop coûteuse en maintenance. Étant donné que le réseau de tronçons est généralement importé uniquement lors de la création d'une instance Geotrek, il a été décidé de créer un outil séparé, générant des fichiers GeoJSON au format requis pour l'import dans Geotrek. Les fichiers GeoJSON sont une variante du format JSON, spécialisée pour les données géospatiales. Cette solution permet non seulement d'alléger la maintenance de Geotrek, mais aussi d'inciter l'utilisateur à vérifier les données générées avant leur import.

Ma mission a donc consisté à concevoir une application avec le framework Django permettant à l'utilisateur de définir une zone géographique et de récupérer les sentiers d'OpenStreetMap présents dans cette zone sous forme d'un fichier GeoJSON. L'application devait proposer deux méthodes de récupération des données :

- Une API REST avec une interface web, permettant d'installer l'application sur un serveur pour mettre à disposition l'API.
- Une commande Django admin, à exécuter en local, pour générer les fichiers GeoJSON directement depuis le terminal.

Pour répondre à cet objectif, j'ai commencé par écrire un script Python prenant en entrée un polygone et le type de sentier souhaité. Comme on peut le voir sur la figure 6 ci-dessous, le script est divisé en quatre étapes :

Figure 6 - Script de conversion des tronçons d'OpenStreetMap en GeoJSON

```python
def osm_paths_to_geojson(polygon, network_type="walk"):
    """
    download paths from OpenStreetMap as a graph, simplify the graph and convert the data into a geojson file
    """
    if not isinstance(polygon, Polygon) or not polygon.is_valid:
        raise TypeError("Invalid polygon")

    # Simplify the boundary with a tolerance of 0.00001°
    simplified_polygon = polygon.simplify(0.00001)

    network_graph_directed = ox.graph_from_polygon(
        simplified_polygon, network_type=network_type, truncate_by_edge=True
    )

    # convert to undirected MultiDiGraph to avoid double path
    network_graph_undirected = ox.convert.to_undirected(network_graph_directed)

    gdf_edges = ox.graph_to_gdfs(network_graph_undirected, nodes=False)

    geojson = gdf_edges.to_json(indent=4)

    return geojson
```

La première étape consiste à formater le polygone d'entrée en vérifiant et simplifiant sa géométrie. La simplification est réalisée avec la fonction simplify du package Shapely. Cette fonction réduit le nombre de points pour qu'ils soient espacés au minimum d'une distance définie, ici 0.00001°, soit environ un mètre. Cela permet de conserver une bonne précision tout en limitant le nombre de points et donc le temps de traitement d'OSMnx.

La deuxième étape est la récupération d'un graphe du réseau de tronçons sur la zone géographique validée. La fonction graph_from_polygon prend en argument un polygone, un type de réseau et un booléen truncate_by_edge. Le type de réseau peut être : all, all_public, bike, drive, drive_service ou walk. Dans le cadre de Geotrek, seuls les types all, bike, drive et walk sont pertinents. Étant donné que Geotrek est principalement utilisé pour la randonnée, le type walk est utilisé par défaut. Lorsque truncate_by_edge est activé, les sentiers partiellement présents dans la zone sont conservés, ce qui permet de ne pas exclure des sentiers dont l'origine est dans la zone mais la fin à l'extérieur.

Le graphe retourné est dirigé : ainsi, si un sentier est à double sens, deux arcs existeront entre les mêmes nœuds. Il faut donc convertir ce graphe en graphe non dirigé pour supprimer les doublons. Ensuite, le graphe est converti en GeoDataFrame puis en fichier JSON.

Pour faciliter l'utilisation de ce script, j'ai mis en place une API. Pour cela, Django propose une extension dédiée : Django REST Framework. Une API (Application Programming Interface) est une interface logicielle qui permet à une application d'exposer ses données à l'extérieur. Une API REST (Representational State Transfer) suit certaines conventions spécifiques. Pour la créer avec Django, il faut définir les paramètres d'entrée dans un serializer, puis créer une vue API qui récupère les données, les traite, puis renvoie le résultat. Le serializer comprend deux champs d'entrée :

- polygon : polygone de la zone géographique souhaitée au format WKT (Well-Known Text)
- network_type : type de réseau parmi all, drive, bike, walk (par défaut walk)

Pour le champ polygon, une vérification supplémentaire est effectuée pour s'assurer que le format WKT est valide, ainsi que la géométrie, grâce à Shapely. La vue API récupère ensuite les données du serializer, les traite, puis gère les verbes HTTP. Ici, seul le verbe POST est utilisé, afin de permettre l'affichage d'un formulaire d'entrée sur l'interface web. Le fichier GeoJSON est automatiquement téléchargé en cas de succès grâce à l'en-tête Content-Disposition. Pour

rendre l'interface web de l'API accessible, le moteur de rendu BrowsableAPIRenderer de Django REST Framework est utilisé. Il faut également associer la vue à une URL.

Figure 7 - Page web de l'API

La figure 7 montre deux sections : en haut, les résultats de la requête, et en bas, les paramètres d'entrée. Malgré une interface intuitive, l'API présente une contrainte : le temps de traitement. Pour des zones importantes, le téléchargement et le traitement peuvent durer plusieurs minutes. Une réponse asynchrone serait donc idéale, avec un traitement en arrière-plan et une notification par mail à l'utilisateur. Cette fonctionnalité n'a pas été implémentée durant le stage car elle n'était pas prioritaire. En effet, une commande Django admin a également été mise en place et n'est pas soumise aux mêmes contraintes temporelles.

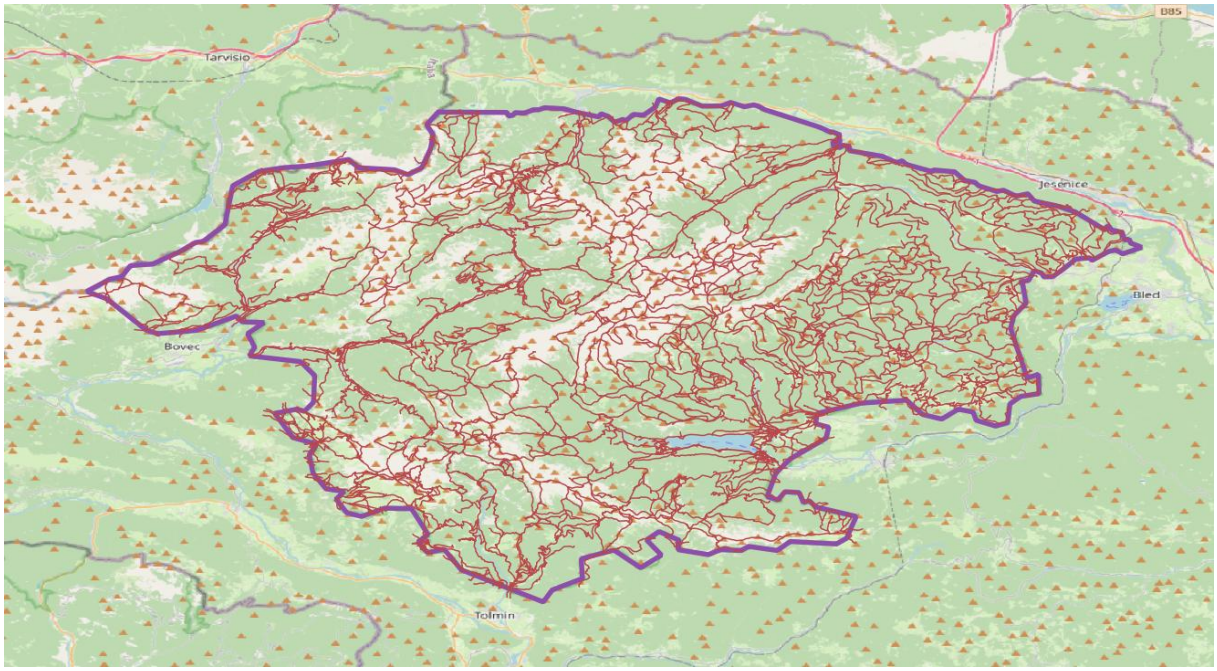
Les commandes Django admin sont des commandes personnalisées exécutables dans le terminal. Pour en créer une, il faut ajouter un fichier dans le dossier management/commands de l'application, définir une classe Command héritant de BaseCommand, puis définir les arguments via add_arguments, et enfin le traitement via handle. La commande créée accepte quatre arguments :

- filename : chemin de sauvegarde du fichier GeoJSON
- --bbox / --polygon : géométrie de la zone à traiter, respectivement boîte englobante ou fichier GeoJSON/WKT
- --network_type : type de réseau souhaité

Le choix d'accepter deux formats pour la géométrie permet de s'adapter aux utilisateurs. La boîte englobante est simple à utiliser, tandis que le polygone permet une définition plus précise mais nécessite plus de compétences.

Voici un exemple avec le parc national du Triglav en Slovénie. Pour extraire le réseau de tronçons présent sur la figure 8, la commande suivante a été utilisée : ./manage.py download --polygon triglav.wkt --network_type walk.

Figure 8 - Visualisation du réseau de tronçons du parc du Triglav sur QGIS



Sur la figure 8, les tronçons générés apparaissent en rouge et la géométrie d'entrée en violet. Le réseau contient 7463 tronçons et couvre l'ensemble du territoire, ce qui permettra un bon référencement linéaire. On constate que certains tronçons dépassent la zone délimitée, ce qui est un comportement attendu avec l'option truncate_by_edge activée.

Ensuite, pour mettre l'application à disposition des utilisateurs, une image Docker a été créée. Elle permet une installation facilitée, quel que soit l'environnement. L'image est basée sur python:3.12-bookworm, car Python 3.12 est utilisé par l'application et le tag bookworm correspond à la dernière version de Debian. Une image Python a été choisie pour la simplicité, mais une optimisation aurait été possible en partant d'une image Debian minimale et en installant uniquement les dépendances nécessaires. Un volume Docker a été mis en place pour permettre les échanges de fichiers entre l'hôte et le conteneur.

Les dépendances à installer sont listées dans un fichier requirements.txt disponible en annexe A. On y retrouve 22 packages, bien que seulement cinq soient explicitement listés, ce qui montre que l'intégration directe dans Geotrek aurait alourdi l'application. L'image Docker lance le serveur via Gunicorn. Bien que Django dispose d'un serveur intégré, celui-ci n'est pas prévu pour un usage en production. Gunicorn est donc utilisé pour les requêtes HTTP, mais ne gère pas les fichiers statiques (CSS, JavaScript). C'est pourquoi il doit être associé à un serveur comme Nginx. Cette association n'a pas été mise en place, l'API n'étant pas encore considérée comme suffisante pour une mise en production.

Enfin, dans le cadre de la mise en ligne du projet en open-source sur GitHub, j'ai été amené à mettre en place une démarche de qualité logicielle afin de garantir la maintenabilité du code. Pour cela, j'ai rédigé des tests unitaires couvrant l'ensemble des fonctionnalités développées. J'ai également configuré des workflows GitHub Actions exécutés automatiquement à chaque *pull request*. Le premier workflow avait pour objectif de lancer les tests unitaires afin de s'assurer que le code restait fonctionnel. Le second vérifiait la qualité du code et son respect des règles de style via l'outil *flake8*. Par la suite, lors de la reprise du projet, mon tuteur a opté pour l'outil *ruff*, écrit en Rust, qui offre une exécution beaucoup plus rapide tout en assurant le même type de vérifications.

L’outil développé répond aux besoins spécifiques de Geotrek pour l’import de tronçons depuis OpenStreetMap, tout en évitant les contraintes d’intégration directe dans l’application. Il permet une génération efficace de fichiers GeoJSON via une API ou une commande terminale, facilitant ainsi le traitement des données géographiques. Ce projet m’a permis de mobiliser et renforcer mes compétences en Python, Django, données spatiales, et Docker. Il pose également les bases pour des évolutions futures comme le traitement asynchrone ou l'amélioration de la validation des données. L’approche modulaire retenue garantit une maintenance légère et une plus grande autonomie pour les utilisateurs. Le projet est conçu pour être facilement accessible et réutilisable. Enfin, il est publié en open source sur GitHub, offrant un point de départ pour d'autres cas d’usage ou contributions : https://github.com/makinacorpus/osm-paths.

Grâce à cet outil complémentaire, il est désormais possible d’initier efficacement une base Geotrek à partir d’OSM. Il est cependant essentiel que ces développements s’inscrivent dans une démarche de qualité propre à un projet open source.

3. Développement en open source : méthodes et outils de qualité

Le projet Geotrek est hébergé sur GitHub et suit une méthodologie rigoureuse pour garantir la qualité et la pérennité du code, dans une logique open-source. Toute nouvelle fonctionnalité doit répondre à plusieurs exigences avant d'être intégrée au projet.

Premièrement, l'intégralité du code développé doit être couverte par des tests unitaires. Cela a constitué un défi particulier dans le cadre de mes développements, car ceux-ci faisaient souvent appel à des API tierces comme Overpass, Nominatim ou Polygon OpenStreetMap. Or, un test unitaire ne doit jamais dépendre d'un service externe, sous peine de produire des résultats instables. Pour contourner cette difficulté, j'ai utilisé les *mocks* du module Python `unittest`. Ces *mocks* permettent de simuler le comportement d'un appel externe en définissant à l'avance la réponse et le code de retour attendu. Cela rend possible le test de tous les cas de figure, sans dépendre d'un fonctionnement réel de l'API. Une fois les tests en place, l'outil Codecov est utilisé pour mesurer leur couverture et s'assurer que l'ensemble du code a bien été pris en compte, même si cela ne garantit pas que tous les scénarios sont testés.

Ensuite, la documentation de Geotrek doit être mise à jour pour chaque nouvelle fonctionnalité. Cette documentation, à la fois à destination des utilisateurs (collectivités, parcs naturels, etc.) et des développeurs, constitue un guide d'utilisation aussi bien qu'un manuel de déploiement ou de personnalisation. Par exemple, elle recense l'ensemble des commandes nécessaires à la mise en place d'une instance Geotrek.

Une fois les tests et la documentation finalisés, la fonctionnalité peut être soumise via une *pull request*. Celle-ci déclenche automatiquement plusieurs *workflows* GitHub Actions. Parmi eux, des tests sont exécutés sur les trois versions de Debian actuellement supportées par le projet : Focal, Jammy et Noble. Ces *CI* (intégrations continues) permettent également de valider la qualité du code grâce à des outils comme `ruff`. Lorsque toutes les vérifications automatiques sont validées, le code est ensuite relu et approuvé par un ou plusieurs développeurs avant d'être fusionné dans la branche principale.

Enfin, le client peut bénéficier des nouvelles fonctionnalités lors de la publication d'une *release* officielle incluant les dernières modifications.

La principale difficulté rencontrée durant ce processus a été la gestion de Git, notamment la coordination entre plusieurs branches interdépendantes. À chaque modification de la branche principale (`master`), je devais rebaser mes branches dans un ordre précis pour maintenir leur cohérence. J'ai donc dû m'autoformer pour maîtriser ces manipulations et éviter de devoir recommencer à zéro à chaque mise à jour.

Cette expérience m'a permis d'approfondir concrètement plusieurs notions, notamment la mise en œuvre des tests unitaires avec des *mocks* et la gestion avancée de Git. La nécessité de simuler des appels à des API externes m'a offert une meilleure compréhension des principes d'isolation des tests, tandis que le travail en branches et les opérations de *rebase* m'ont permis de progresser significativement dans l'usage quotidien de Git, outil fondamental dans tout projet collaboratif.

4. Réflexion autour d'un outil de contribution OSM depuis Geotrek

La deuxième mission qui m'a été confiée durant ce stage consistait à réfléchir à une solution pour encourager les utilisateurs de Geotrek à contribuer à OpenStreetMap, dans une logique de mutualisation des données. L'objectif n'était pas de faire un import massif, contraire à la philosophie d'OpenStreetMap, mais de favoriser une démarche manuelle, contrôlée et respectueuse des règles de la communauté OSM.

En effet, OpenStreetMap repose sur une vérification humaine des données. Chaque contribution doit être validée individuellement afin d'en garantir la qualité. De plus, il est essentiel d'éviter les doublons. Un même objet peut exister dans Geotrek et OpenStreetMap, avec des géométries différentes ou des positionnements légèrement décalés.

La première étape de cette mission a été de présenter différentes propositions de solution à la communauté Geotrek, afin de trouver un consensus. En tant que projet open-source, Geotrek repose sur la validation collective de ses évolutions. J'ai participé à plusieurs réunions en visio (via Jitsi) avec des membres clés de la communauté, notamment Camille Monchicourt, Raphaël Doisy (coordinateurs de Geotrek) et Jean-Christophe Becquet (membre actif de la communauté OSM). Mon rôle a été de formuler des propositions concrètes répondant aux contraintes évoquées, de les présenter, et de recueillir les retours.

La proposition principale reposait sur l'intégration d'un tableau comparatif entre un objet Geotrek et son équivalent dans OpenStreetMap, accessible directement depuis la fiche détail de l'objet dans Geotrek. Ce tableau permettrait à l'utilisateur de visualiser les correspondances de champs entre les deux bases, de sélectionner les données à transférer, puis de valider la contribution via un bouton dédié. Le lien entre les deux objets serait assuré par l'ID OpenStreetMap. Cette méthode offrait l'avantage de garantir une validation humaine et d'éviter les doublons. J'ai accompagné cette proposition d'une maquette interactive réalisée sur Penpot pour illustrer l'expérience utilisateur.

Cependant, cette solution a été jugée trop fastidieuse, notamment en raison du volume de données et de la difficulté d'accès à chaque fiche détail. D'autres idées ont alors émergé, comme le développement d'un outil externe dédié, la mise en place d'un tableau de bord global pour visualiser les objets à synchroniser, ou l'intégration dans Osmose, un outil de détection et de correction d'erreurs dans OpenStreetMap.

Aucune décision n'ayant encore été prise, j'ai réalisé un questionnaire via Framaforms à destination de la communauté Geotrek. Il vise à recueillir leur avis sur leur usage actuel d'OpenStreetMap, les outils qu'ils utilisent déjà, et leur préférence entre deux solutions proposées (comparaison directe ou flux Osmose). Les réponses sont en cours de collecte.

Par ailleurs, même si le développement de l'outil de contribution directe n'a pas été lancé, j'ai mis en place une fonctionnalité facilitant l'accès aux objets sources. J'ai créé un modèle Provider contenant un nom et un template HTML. En Django, un template est un fichier HTML enrichi de balises dynamiques permettant d'afficher des variables ou de générer du contenu en fonction du contexte. Le template HTML stocké dans le modèle sert à construire un lien vers l'objet source, par exemple sur OpenStreetMap. Pour afficher ce lien dans la fiche détail d'un objet, j'ai créé un template tag personnalisé. Il s'agit d'une fonction Django réutilisable dans les templates, qui permet ici de générer dynamiquement le lien en injectant les bonnes valeurs dans le template HTML. Grâce à cela, chaque objet lié à un provider affiche un lien cliquable vers sa source, facilitant la vérification des données et encourageant la contribution sur OpenStreetMap.

CHAPITRE 4. RETOUR D'EXPERIENCE : APPORTS TECHNIQUES, HUMAINS ET METHODOLOGIQUES

Au début de mon stage chez Makina Corpus, j'ai été confronté à une phase d'observation relativement longue, d'environ quatre semaines. Ce temps a été consacré à la lecture de documentation sur Geotrek et OpenStreetMap, afin de comprendre leur structure et d'envisager leur interconnexion. Même si cette période m'a permis de mieux cerner l'environnement technique, j'ai eu initialement du mal à me sentir pleinement utile, mon sujet de stage étant en périphérie des projets en cours. Ce n'est qu'en entrant dans une phase plus active de développement que j'ai commencé à mieux percevoir mon rôle dans l'équipe.

Le tournant s'est produit lorsque j'ai développé des parsers pour importer automatiquement des données issues d'OpenStreetMap dans Geotrek. Ces outils ont notamment servi à alimenter une nouvelle instance en Gaspésie. J'ai accompagné une personne de l'équipe dans leur utilisation, et cette implication m'a permis de mesurer l'impact direct de mon travail. Ce projet m'a également permis de contribuer au dépôt open source de Geotrek, ce qui m'a familiarisé avec le processus de collaboration publique, de revue de code, et de publication dans une communauté active. Cette expérience m'a donné une vision concrète du fonctionnement d'un projet libre, bien au-delà de l'écriture de code.

Mon intégration dans l'équipe a réellement commencé le deuxième jour de stage, le premier ayant été occupé par un tournage « Welcome to the Jungle ». Dès mon arrivée, mon poste de travail était prêt, et une brève présentation de l'équipe m'a été faite par la responsable RH. Un point initial avec mes tuteurs a permis de cadrer les objectifs du stage. J'ai ensuite bénéficié d'un encadrement très régulier, avec un point hebdomadaire chaque lundi pour discuter des avancées, des difficultés rencontrées et du programme à venir, ainsi qu'un suivi quotidien pour échanger avec mes encadrants sur les aspects techniques. Ce cadre m'a permis de monter rapidement en compétence, tout en gardant une vision claire de mes objectifs.

Durant le stage, mes échanges ont principalement eu lieu avec les développeurs backend et frontend de l'équipe Geotrek, ainsi qu'avec le chef de projet et la responsable support, également cartographe. Les interactions étaient fluides, souvent spontanées, mais certaines nécessitaient des réunions spécifiques, comme pour discuter des implications d'un changement d'API. Ces échanges m'ont permis de mieux comprendre les différents rôles au sein d'un projet technique, même si je n'ai pas découvert de métiers totalement nouveaux.

Dès le début, j'ai ressenti que Makina Corpus portait des valeurs fortes, notamment en matière d'open source, d'éthique et de transparence. L'ensemble des projets repose sur une logique ouverte, dans laquelle chaque contribution est publique, documentée et soumise à l'examen collectif. Cette culture m'a sensibilisé à une manière différente de concevoir le développement, centrée sur le partage, la durabilité et la qualité. Même si l'éco-conception n'a pas été un axe direct de mon stage, j'ai été sensibilisé aux enjeux liés à la sobriété numérique et à l'importance de limiter l'impact environnemental des services développés.

Sur le plan de l'égalité, j'ai observé un équilibre notable dans la répartition des rôles, avec une présence féminine d'environ 35 % dans un secteur souvent très masculin. L'ambiance de travail était respectueuse, inclusive, et les responsabilités étaient partagées équitablement. Je n'ai pas

été confronté à de dilemme éthique particulier, mais les pratiques de l'entreprise, notamment la transparence vis-à-vis des clients et l'engagement dans des projets à vocation publique, montrent que ces questions sont prises au sérieux.

Ce stage a été l'occasion de mettre en perspective mes valeurs personnelles. Je suis arrivé avec la volonté de travailler sur un projet ayant du sens, aligné avec mes convictions écologiques. Makina Corpus, par son positionnement sur des projets liés à la préservation des territoires, ses choix technologiques responsables et sa gouvernance ouverte, répondait à ces attentes. Cette expérience a aussi réorienté mon projet professionnel. Intéressé initialement par les systèmes embarqués, je souhaite désormais m'orienter vers le développement web backend, un domaine qui me semble plus cohérent avec mes valeurs, tout en offrant un terrain d'apprentissage riche et concret.

Au-delà des résultats techniques, ce stage m'a offert une véritable montée en compétences et une immersion dans les pratiques professionnelles du développement. Il est temps de conclure cette expérience et d'en dégager les perspectives futures.

CHAPITRE 5. CONCLUSION ET PERSPECTIVES D'EVOLUTION DU PROJET

Ce stage de fin d'études au sein de Makina Corpus m'a permis de mener à bien une mission à fort enjeu pour l'entreprise et la communauté Geotrek : concevoir et développer un système d'import de données issues d'OpenStreetMap afin d'enrichir automatiquement les bases de données de Geotrek. Le travail accompli a permis d'initier une avancée significative dans l'interopérabilité entre deux outils open source majeurs, avec des bénéfices concrets à plusieurs niveaux.

Pour Makina Corpus, les outils développés facilitent désormais considérablement l'initialisation d'une instance Geotrek. De plus, pour les utilisateurs, l'apport d'un parser OSM est particulièrement stratégique. En effet, il repose sur une base de données gratuite, complète et mondiale, constamment enrichie par la communauté. Il permet ainsi aux utilisateurs de disposer d'une source de données immédiatement exploitable, sans coût supplémentaire, tout en couvrant des territoires bien au-delà du périmètre national. L'ensemble des parsers nécessaires a pu être conçu et intégré au produit durant la durée du stage. Ce travail a également ouvert de nouvelles perspectives, notamment en initiant des discussions autour d'un outil de contribution vers OpenStreetMap. Ce projet, amorcé durant le stage, pourrait se poursuivre dans les mois à venir.

D'un point de vue personnel, ce stage m'a permis d'acquérir de nombreuses compétences techniques nouvelles, notamment l'apprentissage de Django, l'exploitation d'API REST, la manipulation de données géographiques (SIG), l'utilisation de Docker, ainsi que le développement de tests unitaires poussés. J'ai également renforcé ma maîtrise de Git. Plus encore, j'ai découvert un réel intérêt pour le développement back-end, que je n'avais pas encore exploré durant ma formation. Cette appétence m'a donné envie de m'orienter professionnellement vers un poste de développeur back-end, si possible dans un environnement open source.

Enfin, bien que le domaine de la géomatique et le développement web ne corresponde pas directement à mes spécialisations à l'INSA, j'ai choisi ce stage pour élargir mes horizons et sortir de ma zone de confort. Ce choix s'est révélé très enrichissant, me permettant de mettre à l'épreuve mes capacités d'adaptation, d'analyse et de structuration, tout en découvrant un champ d'application concret de l'ingénierie logicielle dans le domaine de la gestion territoriale. Si aucune embauche immédiate n'est envisagée chez Makina Corpus en raison du contexte interne, cette expérience a consolidé mes compétences et mes choix professionnels pour la suite de mon parcours.

ANNEXES

Présentation de l'Entreprise

Makina Corpus est une société de services en ingénierie logicielle, dont l'activité principale repose sur la conception, le développement et l'intégration d'applications web et mobiles innovantes. Elle se distingue par un positionnement résolument tourné vers les logiciels libres, qu'elle utilise de manière exclusive. Ce choix, stratégique et assumé, s'inscrit dans une logique de transparence, de pérennité et d'indépendance pour ses clients, qui restent pleinement maîtres de leurs outils et de leur évolution. En refusant les modèles propriétaires, Makina Corpus garantit à ses partenaires une autonomie technique renforcée et une meilleure maîtrise de leurs coûts.

L'entreprise est établie en France, avec deux implantations principales : un site à Nantes, au 11 rue du Marchix, et un second site à Toulouse, au 49 Grande Rue Saint-Michel. Aujourd'hui, Makina Corpus compte environ 46 salariés, répartis entre ces différentes localisations. Cette équipe, à taille humaine, fonctionne sur un mode collaboratif fort, ce qui permet une grande réactivité et une circulation fluide de l'expertise au sein de l'organisation.

Makina Corpus évolue dans le secteur du numérique, plus précisément dans le domaine du développement d'applications métiers complexes à forte valeur ajoutée. Elle a construit son expertise sur plusieurs axes majeurs : la cartographie interactive, le traitement d'informations spatiales, les systèmes d'information géographique (SIG), la data science, l'éco-informatique, ainsi que la conception de portails et plateformes collaboratives. Ces compétences couvrent un large spectre, tant sur le plan technique que fonctionnel. Le cœur de métier de l'entreprise repose sur un accompagnement global, depuis la conception et l'architecture logicielle, jusqu'au développement, à l'ergonomie, au graphisme, à la maintenance, à l'hébergement, aux audits techniques, sans oublier une offre de formation structurée. À ce titre, Makina Corpus est un organisme de formation certifié Qualiopi, ce qui atteste du sérieux et de la qualité de ses prestations pédagogiques. Son catalogue est dense et couvre aussi bien le développement backend et frontend, la sécurité des applications, que les outils de cartographie et l'analyse de données.

La diversité des compétences internes permet à Makina Corpus de répondre à des besoins variés et exigeants. Son portefeuille client comprend de grands groupes industriels et de services, des administrations, des établissements publics, des collectivités territoriales, mais aussi des PME et des start-ups. On y retrouve des noms comme Orange, Airbus, La Poste, Météo France, le Ministère de l'Éducation nationale ou encore la SNCF. L'entreprise fonde sa relation client sur un principe simple : la co-construction. Elle s'appuie sur une méthode de travail agile, mêlant rigueur technique et souplesse organisationnelle. Les pratiques Scrum et Kanban sont largement répandues dans ses projets, avec une attention particulière portée à l'écoute du client, à la transparence dans les processus, et à l'amélioration continue. Chaque projet s'inscrit dans une logique de collaboration active et d'adaptation permanente aux besoins réels du terrain.

Makina Corpus revendique une culture d'entreprise solidement ancrée dans les valeurs du logiciel libre. L'ouverture, le partage, la transparence, la contribution font partie intégrante de son identité. Cette culture se traduit au quotidien : les équipes publient des articles techniques, participent à des conférences, s'impliquent dans des communautés, encadrent des étudiants dans le cadre de programmes comme le Google Summer of Code, et contribuent activement à des projets open source sur des plateformes comme GitHub. En interne, le développement

collaboratif, le pair programming, les revues de code et la veille technologique sont des pratiques encouragées et structurantes. Cette dynamique collective renforce la qualité du code produit, tout en assurant une montée en compétence continue des collaborateurs.

Cette approche ouverte et responsable s'inscrit également dans une démarche plus globale de numérique éthique et durable. Consciente des enjeux environnementaux liés aux usages numériques, Makina Corpus a engagé un travail de fond sur l'éco-conception logicielle. L'entreprise s'appuie sur les principes de l'analyse du cycle de vie pour essayer de concevoir des services numériques sobres, optimisés non seulement lors de leur utilisation, mais aussi dès la phase de développement. Cela signifie que chaque étape, du codage aux outils de tests, en passant par les choix techniques d'hébergement ou de structure de données, est pensée pour réduire l'empreinte environnementale globale. Makina Corpus utilise pour cela des outils comme LightHouse ou GreenFrame, qui permettent de mesurer concrètement la performance énergétique et les émissions de gaz à effet de serre d'une application.

En parallèle de ses activités de service, Makina Corpus s'investit fortement dans la recherche et développement. Chaque année, plus de 15 % de son chiffre d'affaires est consacré à l'innovation. L'entreprise entretient des partenariats étroits avec des laboratoires publics tels que le CNRS, l'Institut de Recherche en Informatique de Toulouse, l'École des Mines de Nantes ou encore le Co-Design Lab de Télécom ParisTech. Ces collaborations donnent naissance à des projets de grande valeur scientifique et sociale, comme Accessimap, qui vise à rendre la cartographie accessible aux personnes déficientes visuelles, ou encore GéoTopia, axé sur la représentation cartographique des territoires. Cette implication dans la recherche illustre une volonté constante de repousser les limites techniques, tout en apportant des réponses concrètes à des enjeux sociétaux.

Sur le plan financier, Makina Corpus repose sur un modèle clair et maîtrisé, exclusivement fondé sur la prestation de services. Elle ne vend aucune licence logicielle, mais valorise son savoir-faire à travers le développement sur mesure, la maintenance, l'hébergement, le conseil et la formation. Ce modèle économique permet à ses clients de conserver une indépendance technologique totale, tout en bénéficiant de prestations expertes et adaptées.

En somme, Makina Corpus est bien plus qu'un prestataire technique. C'est une entreprise engagée, libre et visionnaire, qui place l'humain, la durabilité et l'excellence technologique au cœur de son fonctionnement. Elle incarne une alternative crédible et responsable dans l'écosystème numérique, et s'affirme comme un partenaire stratégique pour les acteurs publics et privés à la recherche de solutions fiables, éthiques et pérennes.

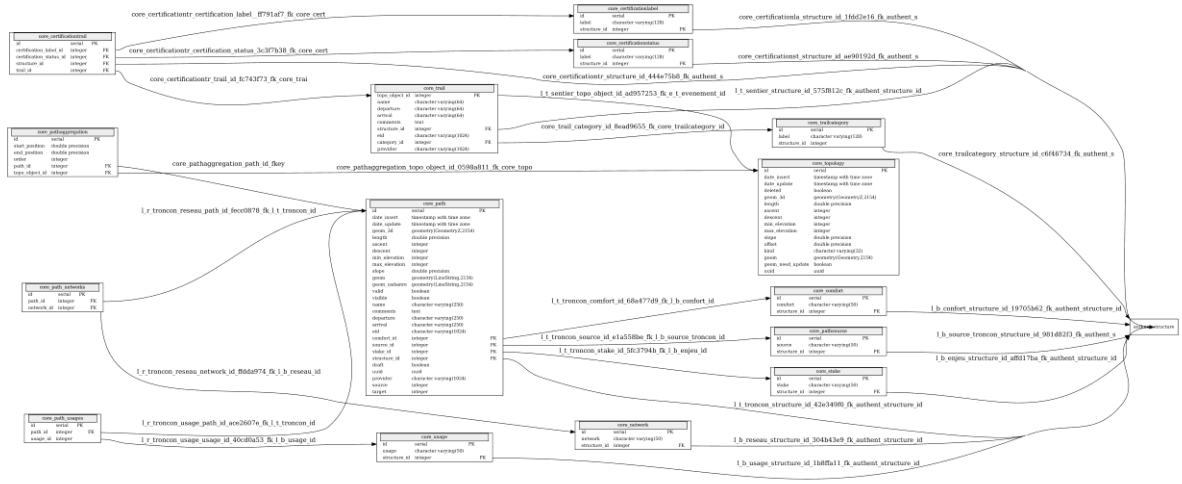
Annexe A – Dépendence de l'application OSM-path

```
asgiref==3.8.1
    # via django
certifi==2025.1.31
    # via pyogrio, pyproj, requests
charset-normalizer==3.4.1
    # via requests
django==5.1.7
    # via osm-paths (pyproject.toml),  djangorestframework
djangorestframework==3.16.0
    # via osm-paths (pyproject.toml)
geopandas==1.0.1
    # via osmnx
gunicorn==23.0.0
    # via osm-paths (pyproject.toml)
idna==3.10
    # via requests
networkx==3.4.2
    # via osmnx
numpy==2.2.4
    # via geopandas, osmnx, pandas, pyogrio, shapely
osmnx==2.0.2
    # via osm-paths (pyproject.toml)
packaging==24.2
    # via geopandas, gunicorn, pyogrio
pandas==2.2.3
    # via geopandas, osmnx
pyogrio==0.10.0
    # via geopandas
pyproj==3.7.1
    # via geopandas
python-dateutil==2.9.0.post0
```

```
    # via pandas
pytz==2025.2
    # via pandas
requests==2.32.3
    # via osmnx
shapely==2.0.7
    # via osm-paths (pyproject.toml), geopandas, osmnx
six==1.17.0
    # via python-dateutil
sqlparse==0.5.3
    # via django
tzdata==2025.2
    # via pandas
urllib3==2.3.0
    # via request
```

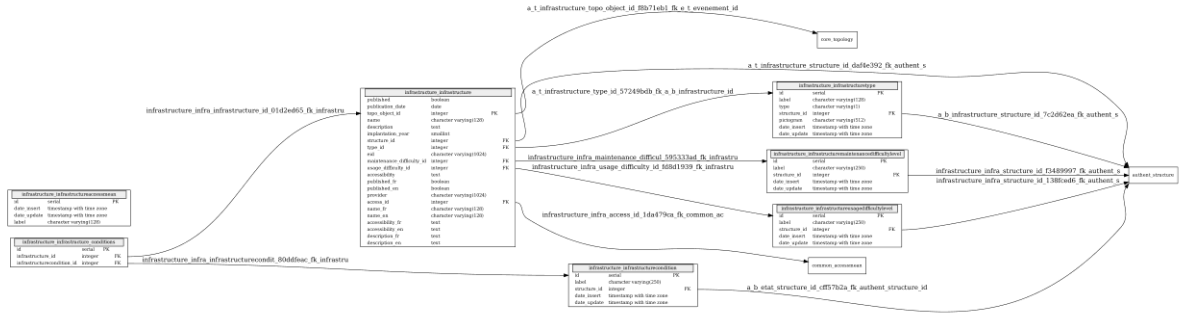
Annexe B – Diagramme de classe des modèles de Geotrek

Modèles de l'application core



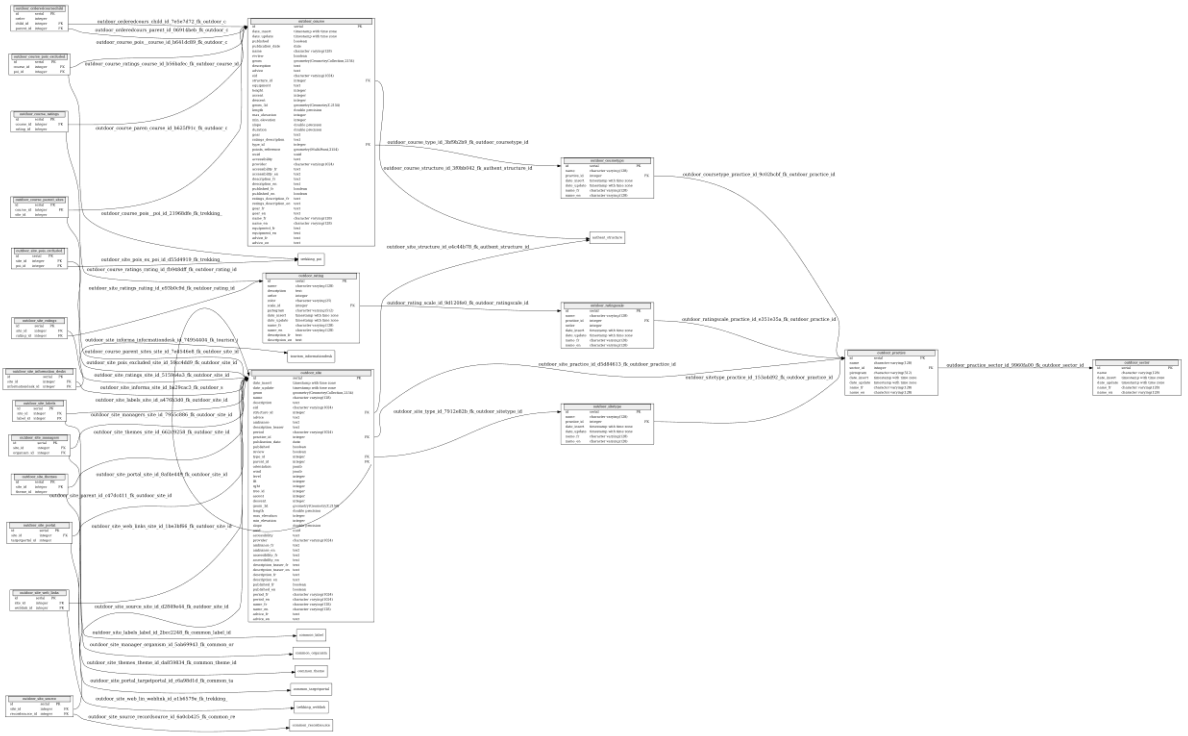
Source : https://github.com/GeotrekCE/Geotrek-admin/blob/master/docs/data-model/core.pdf

Modèles de l'application infrastructure



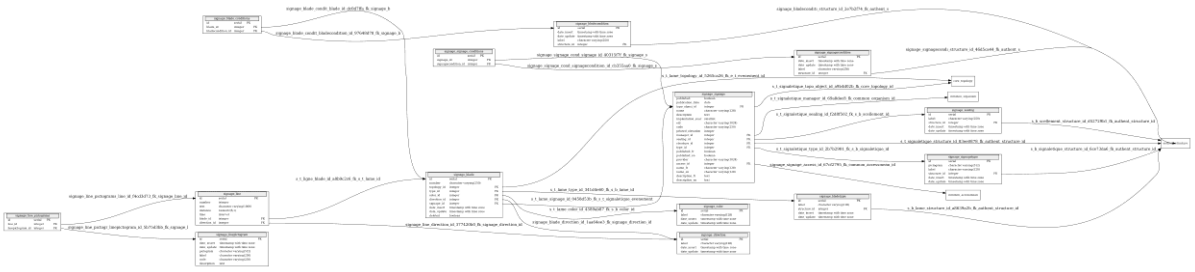
Source : https://github.com/GeotrekCE/Geotrek-admin/blob/master/docs/data-model/infrastructure.pdf

Modèles de l'application outdoor



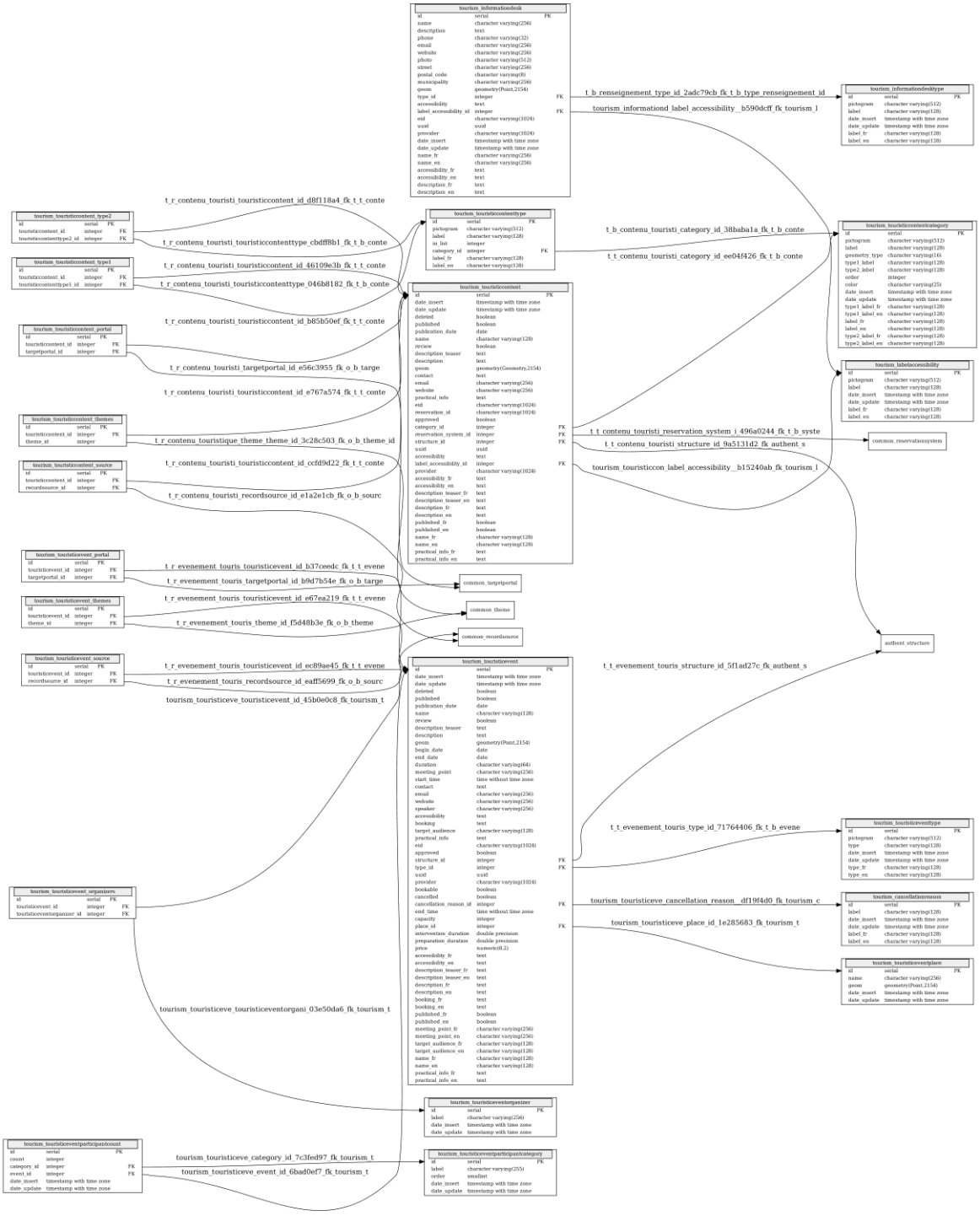
Source : https://github.com/GeotrekCE/Geotrek-admin/blob/master/docs/data-model/outdoor.pdf

Modèles de l'application signage



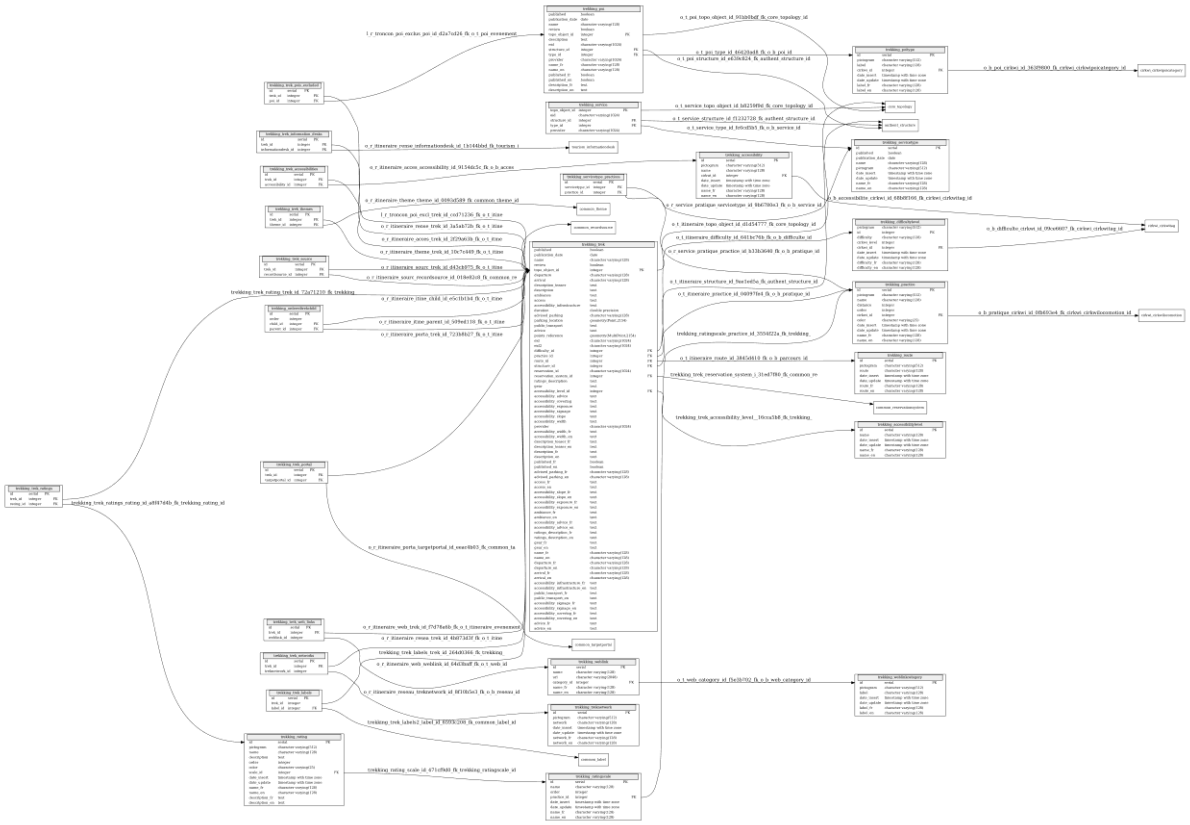
Source : https://github.com/GeotrekCE/Geotrek-admin/blob/master/docs/data-model/signage.pdf

Modèles de l'application tourism



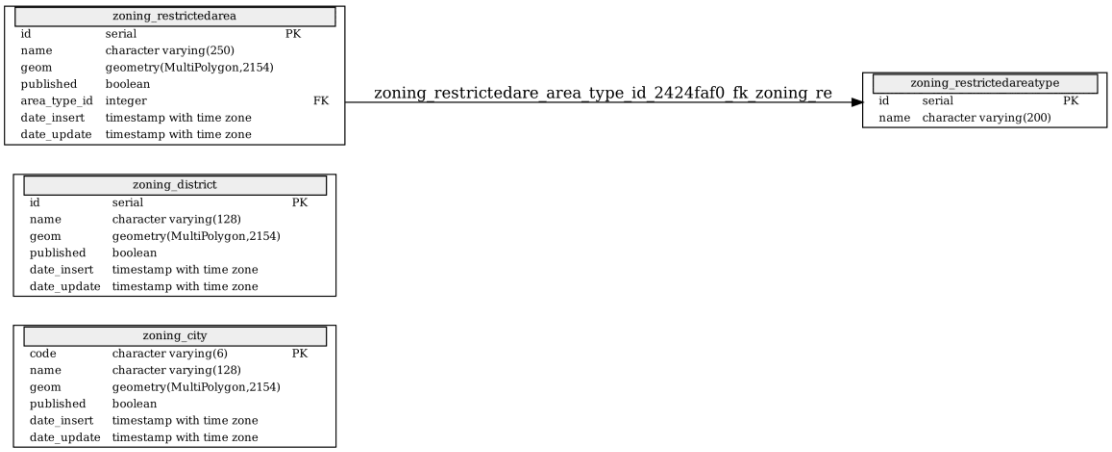
Source : https://github.com/GeotrekCE/Geotrek-admin/blob/master/docs/data-model/tourism.pdf

Modèles de l'application trekking



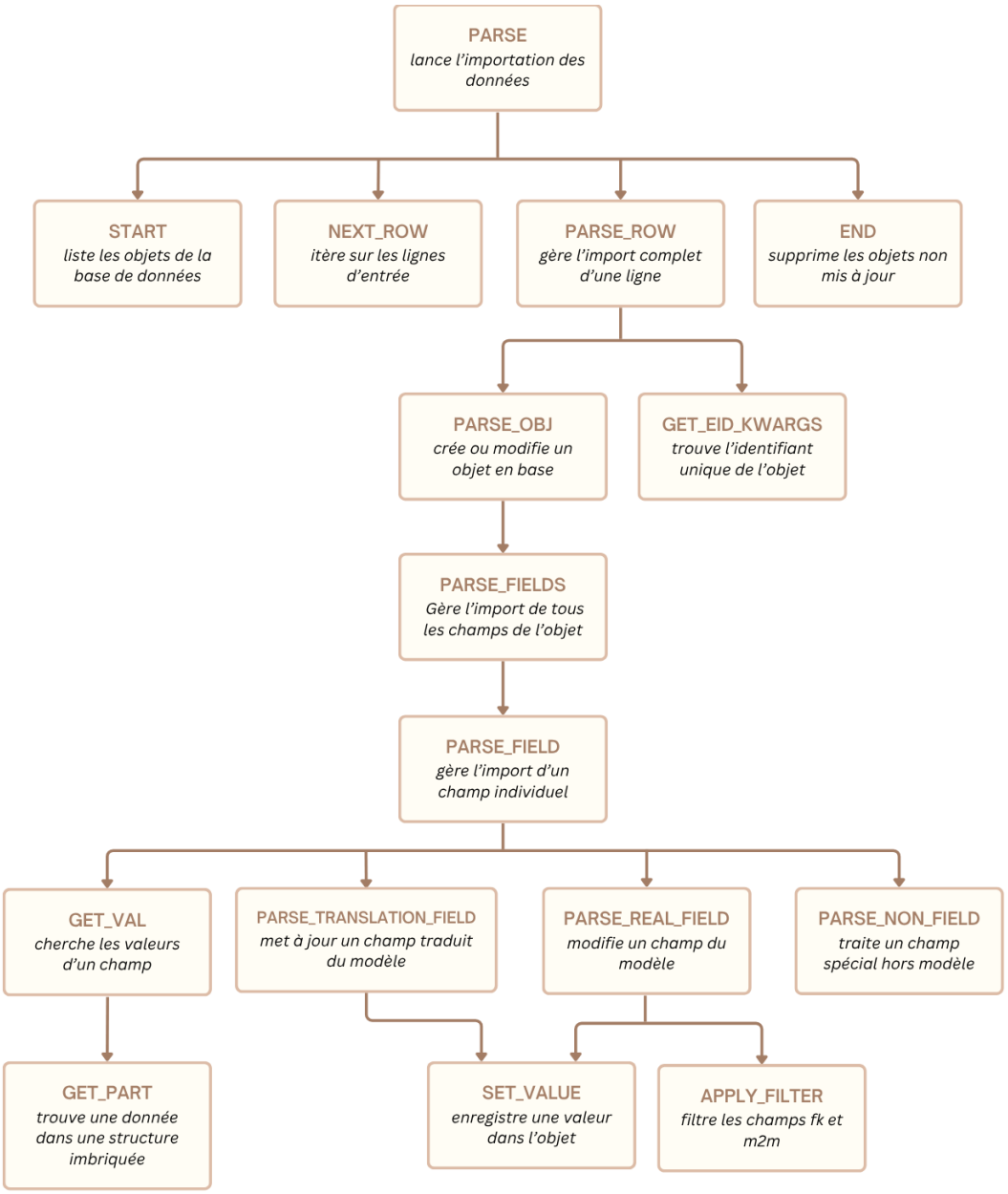
Source : https://github.com/GeotrekCE/Geotrek-admin/blob/master/docs/data-model/trekking.pdf

Modèles de l'application zoning



Source : https://github.com/GeotrekCE/Geotrek-admin/blob/master/docs/data-model/zoning.pdf

Annexe C – Arbre d'appel des fonctions de la classe Parser



Annexe D – Table des paramètres de configuration la classe Parser

Paramètre	Type	Description reformulée
label	str	Nom affiché dans l'interface pour identifier ce parser.
model	str	Nom du modèle de données dans lequel les informations doivent être enregistrées.
filename	str	Nom du fichier source contenant les données à importer.
url	str	Adresse de l'API d'où proviennent les données.
simplify_tolerance	float	Distance minimale (en mètres) entre deux points d'une forme géographique : les points trop proches peuvent être supprimés pour simplifier la forme.
update_only	bool	Si activé, seules les données déjà existantes peuvent être mises à jour. Aucun nouvel objet ne sera créé.
delete	bool	Si activé, les objets qui ne sont plus présents dans les nouvelles données seront supprimés.
duplicate_eid_allowed	bool	Si activé, plusieurs objets peuvent avoir le même identifiant externe.
fill_empty_translated_fields	bool	(Description manquante à compléter selon besoin, probablement : permet de remplir automatiquement les champs de traduction vides.)
warn_on_missing_fields	bool	Affiche un message d'avertissement si des champs attendus ne sont pas trouvés dans les données importées.
warn_on_missing_objects	bool	Affiche un message d'avertissement si des objets attendus ne sont pas présents dans les données importées.
separator	str	Caractère utilisé pour séparer plusieurs valeurs dans une même cellule (par exemple une virgule).
eid	str	Nom du champ qui contient l'identifiant externe unique pour chaque objet.
provider	str	Nom de la source d'où viennent les données (ex : nom d'un partenaire ou d'une API).
fields	dict	Correspondance entre les champs du modèle et ceux de la source. Un champ peut être lié à plusieurs champs de la source.

constant_fields	dict	Donne des valeurs fixes à certains champs du modèle, pour chaque objet importé.
m2m_fields	dict	Correspondance entre les champs « many-to-many » du modèle (liens multiples avec d'autres objets) et ceux de la source.
m2m_constant_fields	dict	Valeurs fixes pour les champs « many-to-many » à attribuer à chaque objet.
m2m_aggregate_fields	list	Liste des champs « many-to-many » pour lesquels les nouvelles données seront ajoutées sans supprimer les anciennes.
non_fields	dict	Correspondance entre des données de la source et des champs qui ne font pas partie du modèle principal (par exemple, des données annexes).
natural_keys	dict	Indique pour chaque relation (comme les liens vers d'autres modèles) quel champ utiliser pour identifier l'objet lié.
field_options	dict	Paramètres supplémentaires pour chaque champ, comme "obligatoire" ou "créer s'il n'existe pas".
default_language	str	Langue par défaut utilisée pour les données importées.

Annexe E – Parsers personnalisé pour les parc du Triglav

```
from geotrek.tourism.parsers import InformationDeskOpenStreetMapParser
from geotrek.trekking.parsers import OpenStreetMapPOIParser
from geotrek.zoning.parsers import OpenStreetMapDistrictParser
from geotrek.signage.parsers import OpenStreetMapSignageParser
from geotrek.infrastructure.parsers import OpenStreetMapInfrastructureParser
from geotrek.zoning.parsers import OpenStreetMapRestrictedAreaParser
from geotrek.tourism.parsers import OpenStreetMapTouristicContentParser
from geotrek.outdoor.parsers import OpenStreetMapOutdoorSiteParser

REFERENCE_GEOM = {
   "model": "restrictedarea",
   "app_label": "zoning",
   "geom_field": "geom",
   "object_filter": {"name": "Triglavski narodni park"}
}



# InformationDesk
class RangerStationParser():
   label = "Information desk - ranger station"
   provider = "OpenStreetMap"
   intersection_geom = REFERENCE_GEOM
   tags = [{"amenity" : "ranger_station"}]
   default_fields_values = {
      "name": "Hiša v parku",
      "name_en": "Ranger station",
      "name_fr": "Maison du parc"
      }
   type = "Maisons du parc"
```

```
class TourismOfficeParser(InformationDeskOpenStreetMapParser):
    label = "Information desk - tourism office"
    provider = "OpenStreetMap"
    intersection_geom = REFERENCE_GEOM
    tags = [{"information" : "office"}]
    default_fields_values = {
        "name": "Turistični urad",
        "name_en": "Tourism office",
        "name_fr": "Office du tourisme"
    }

# POI
class ViewPointParser(OpenStreetMapPOIParser):
    label = "POI - viewpoint"
    provider = "OpenStreetMap"
    intersection_geom = REFERENCE_GEOM
    tags = [{"tourism" : "viewpoint"}]
    default_fields_values = {
        "name": "Stališče",
        "name_en": "Point of view",
        "name_fr": "Point de vue"
        }
    type = "Point de vue"

class LakeParser(OpenStreetMapPOIParser):
    label = "POI - lake"
    provider = "OpenStreetMap"
    intersection_geom = REFERENCE_GEOM
    tags = [{"water" : "lake"}]
    default_fields_values = {
        "name": "Jezero",
        "name_en": "Lake",
```

```
        "name_fr": "Lac"
        }
    type = "Lac"

class GeologyParser(OpenStreetMapPOIParser):
    label = "POI - geology"
    provider = "OpenStreetMap"
    intersection_geom = REFERENCE_GEOM
    tags = [{"natural" : "arete"}]
    default_fields_values = {
        "name": "Geološka lokacija",
        "name_en": "Geology spot",
        "name_fr": "Lieu géologique"
        }
    type = "Géologie"

class HistoryParser(OpenStreetMapPOIParser):
    label = "POI - history"
    provider = "OpenStreetMap"
    intersection_geom = REFERENCE_GEOM
    tags = [
        {"historic": "yes"},
        {"historic": "castel"},
        {"historic": "memorial"},
        {"historic": "fort"},
        {"historic": "bunker"},
        {"building": "chapel"},
        {"building": "church"}
    ]
    default_fields_values = {
        "name": "Zgodovinsko območje",
        "name_en": "Historic spot",
```

```
        "name_fr": "Lieu historique"
        }
    type = "Histoire"

class ArchitectureParser(OpenStreetMapPOIParser):
    label = "POI - architecture"
    provider = "OpenStreetMap"
    intersection_geom = REFERENCE_GEOM
    tags = [
        {"historic" : "monestary"},
        {"building": "cathedral"},
    ]
    default_fields_values = {
        "name": "Arhitektura",
        "name_en": "Architecture",
        "name_fr": "Architecture"
        }
    type = "Architecture"

class SaddleParser(OpenStreetMapPOIParser):
    label = "POI - saddle"
    provider = "OpenStreetMap"
    intersection_geom = REFERENCE_GEOM
    tags = [
        {"mountain_pass": "yes"},
        {"natural": "saddle"},
    ]
    default_fields_values = {
        "name": "Ovratnik",
        "name_en": "Saddle",
        "name_fr": "Col"
        }
```

```
    type = "Col"

class PeakParser(OpenStreetMapPOIParser):
    label = "POI - peak"
    provider = "OpenStreetMap"
    intersection_geom = REFERENCE_GEOM
    tags = [{"natural": "peak"}]
    default_fields_values = {
        "name": "Vrh",
        "name_en": "Peak",
        "name_fr": "Sommet"
        }
    type = "Sommet"


class AlpineHutParser(OpenStreetMapPOIParser):
    label = "POI - alpine hut"
    provider = "OpenStreetMap"
    intersection_geom = REFERENCE_GEOM
    tags = [{"tourism": "alpine_hut"}]
    default_fields_values = {
        "name": "Zatočišče",
        "name_en": "Alpine hut",
        "name_fr": "Refuge"
        }
    type = "Refuge"

# District
class AdminAreaParser(OpenStreetMapDistrictParser):
    label = "District - district"
    provider = "OpenStreetMap"
    intersection_geom = REFERENCE_GEOM
```

```
    tags = [
        [{"boundary": "administrative"}, {"admin_level": "8"}],
        {"boundary": "national_park"}
    ]


# Signage
class DirectionalParser(OpenStreetMapSignageParser):
    label = "Signage - guidepost"
    provider = "OpenStreetMap"
    intersection_geom = REFERENCE_GEOM
    tags = [{"information": "guidepost"}]
    default_fields_values = {
        "name": "smerokaz",
        "name_en": "guidepost",
        "name_fr": "panneau directionnel"
        }
    type = "Directionelle"

class InformationParser(OpenStreetMapSignageParser):
    label = "Signage - information board"
    provider = "OpenStreetMap"
    intersection_geom = REFERENCE_GEOM
    tags = [{"information": "board"}]
    default_fields_values = {
        "name": "smerokaz",
        "name_en": "informacijska tabla",
        "name_fr": "panneau d'information"
        }
    type = "Information"

"""
```

```
class BenchParser(OpenStreetMapInfrastructureParser):
    label = "Infrastructure - bench"
    provider = "OpenStreetMap"
    intersection_geom = REFERENCE_GEOM
    tags = [{"amenity": "bench"}]
    default_fields_values = {
        "name": "klop",
        "name_en": "bench",
        "name_fr": "banc"
        }
    type = "Banc"

class FootbridgeParser(OpenStreetMapInfrastructureParser):
    label = "Infrastructure - footbridge"
    provider = "OpenStreetMap"
    intersection_geom = REFERENCE_GEOM
    tags = [
        [{"highway": "path"}, {"bridge":"yes"}],
        [{"highway": "footway"}, {"bridge":"yes"}]
    ]
    default_fields_values = {
        "name": "most za pešce",
        "name_en": "footbridge",
        "name_fr": "passerelle"
        }
    type = "Passerelle"

class ShelterParser(OpenStreetMapInfrastructureParser):
    label = "Infrastructure - shelter"
    provider = "OpenStreetMap"
    intersection_geom = REFERENCE_GEOM
    tags = [{"amenity": "shelter"}]
```

```
    default_fields_values = {
        "name": "zavetišče",
        "name_en": "shelter",
        "name_fr": "abri"
        }
    type = "Abri"

class PlaygroundParser(OpenStreetMapInfrastructureParser):
    label = "Infrastructure - playground"
    provider = "OpenStreetMap"
    intersection_geom = REFERENCE_GEOM
    tags = [
        {"leisure": "playground"},
        {"landuse": "recreation_ground"}
    ]
    default_fields_values = {
        "name": "otroško igrišče",
        "name_en": "playground",
        "name_fr": "aire de jeux"
        }
    type = "Aire de jeux"

class ToiletsParser(OpenStreetMapInfrastructureParser):
    label = "Infrastructure - toilets"
    provider = "OpenStreetMap"
    intersection_geom = REFERENCE_GEOM
    tags = [{"amenity": "toilets"}]
    default_fields_values = {
        "name": "stranišča",
        "name_en": "toilets",
        "name_fr": "toilettes"
        }
```

```
    type = "Toilettes"

class BikeParkingParser(OpenStreetMapInfrastructureParser):
    label = "Infrastructure - bicycle parking"
    provider = "OpenStreetMap"
    intersection_geom = REFERENCE_GEOM
    tags = [{"amenity": "bicycle_parking"}]
    default_fields_values = {
        "name": "parkiranje koles",
        "name_en": "bicycle parking",
        "name_fr": "parking à vélo"
        }
    type = "Parc à vélo"

class TableParser(OpenStreetMapInfrastructureParser):
    label = "Infrastructure - picnic table"
    provider = "OpenStreetMap"
    intersection_geom = REFERENCE_GEOM
    tags = [
        {"leisure": "picnic_table"},
        {"tourism": "picnic_table"}
    ]
    default_fields_values = {
        "name": "miza za piknik",
        "name_en": "picnic table",
        "name_fr": "table de pique-nique"
        }
    type = "Table"

class BarrierParser(OpenStreetMapInfrastructureParser):
    label = "Infrastructure - barrier"
    provider = "OpenStreetMap"
```

```
    intersection_geom = REFERENCE_GEOM
    tags = [
        {"barrier": "cycle_barrier"},
        {"barrier": "swing_gate"},
        {"barrier": "gate"},
        {"barrier": "yes"}
    ]
    default_fields_values = {
        "name": "zapora",
        "name_en": "barrier",
        "name_fr": "barrière"
        }
    type = "Barrière"

class ChargingStationParser(OpenStreetMapInfrastructureParser):
    label "Infrastructure - charging station"
    provider = "OpenStreetMap"
    intersection_geom = REFERENCE_GEOM
    tags = [
        {"amenity": "charging_station"},
    ]
    default_fields_values = {
        "name": "polnilna postaja",
        "name_en": "charging station",
        "name_fr": "station de recharge"
        }
    type = "Station de recharge"



# Restricted Area
class NatureReserveParser(OpenStreetMapRestrictedAreaParser):
    label = "Restricted area - nature reserve"
```

```
    provider = "OpenStreetMap"
    intersection_geom = REFERENCE_GEOM
    tags = [{"leisure": "nature_reserve"}]
    default_fields_values = {"name_en": "nature reserve"}

class NationalParkParser(OpenStreetMapRestrictedAreaParser):
    label = "Restricted area - national park"
    provider = "OpenStreetMap"
    intersection_geom = REFERENCE_GEOM
    tags = [{"boundary": "national_park"}]
    default_fields_values = {"name_en": "national parc"}
    area_type = "Inconnu"

# Touristic content
class HotelParser(OpenStreetMapTouristicContentParser):
    provider = "OpenStreetMap"
    label = "Touristic content - hotel"
    intersection_geom = REFERENCE_GEOM
    tags = [{"tourism": "hotel"}]
    default_fields_values = {
        "name": "hotel",
        "name_en": "hotel",
        "name_fr": "hôtel"
    }
    category = "Hébergement"
    type1 = "Hôtel"

class CampingParser(OpenStreetMapTouristicContentParser):
    provider = "OpenStreetMap"
    label = "Touristic content - camping"
    intersection_geom = REFERENCE_GEOM
    tags = [{"tourism": "camp_site"}]
```

```
    default_fields_values = {
        "name": "kamp",
        "name_en": "camp site",
        "name_fr": "camping"
    }
    category = "Hébergement"
    type1 = "Camping"

class GuestHouseParser(OpenStreetMapTouristicContentParser):
    provider = "OpenStreetMap"
    label = "Touristic content - guest house"
    intersection_geom = REFERENCE_GEOM
    tags = [{"tourism": "guest_house"}]
    default_fields_values = {
        "name": "hiša za goste",
        "name_en": "guest house",
        "name_fr": "gîte"
    }
    category = "Hébergement"
    type1 = "Chambre d'hôte"

class RestaurantParser(OpenStreetMapTouristicContentParser):
    provider = "OpenStreetMap"
    label = "Touristic content - restaurants"
    intersection_geom = REFERENCE_GEOM
    tags = [{"amenity": "restaurant"}]
    default_fields_values = {
        "name": "restavracija",
        "name_en": "restaurant",
        "name_fr": "restaurant"
    }
    category = "Restaurants"
```

```
    type1 = "Restaurant"

class MuseumParser(OpenStreetMapTouristicContentParser):
    provider = "OpenStreetMap"
    label = "Touristic content - museum"
    intersection_geom = REFERENCE_GEOM
    tags = [{"tourism": "museum"}]
    default_fields_values = {
        "name": "muzej",
        "name_en": "museum",
        "name_fr": "musée"
    }
    category = "Musée"
    type1 = "Musée"

# Outdoor site
class ClimbingSiteParser(OpenStreetMapOutdoorSiteParser):
    provider = "OpenStreetMap"
    label = "Outdoor site - climbing site"
    tags = [{"sport": "climbing"}]
    default_fields_values = {
        "name": "plezališče",
        "name_en": "climbing site",
        "name_fr": "site d'escalade",
        }
    practice = "Escalade"

class CanoeSiteParser(OpenStreetMapOutdoorSiteParser):
    provider = "OpenStreetMap"
    label = "Outdoor site - canoe site"
    tags = [{"sport": "canoe"}]
    default_fields_values = {
```

```
        "name": "spletno mesto za kanu",
        "name_en": "canoe site",
        "name_fr": "site de canoe-kayak",
        }
    practice = "Canoë-kayak"

class ViaFerrataParser(OpenStreetMapOutdoorSiteParser):
    provider = "OpenStreetMap"
    label = "Outdoor site - via ferrata"
    tags = [{"highway": "via_ferrata"}]
    default_fields_values = {
        "name": "via ferrata",
        "name_en": "via ferrata",
        "name_fr": "via ferrata",
        }
    practice = "Via ferrata"

class FreeFlyingSiteParser(OpenStreetMapOutdoorSiteParser):
    provider = "OpenStreetMap"
    label = "Outdoor site - free flying"
    tags = [{"sport": "free_flying"}]
    default_fields_values = {
        "name": "brezplačna spletna stran za letenje",
        "name_en": "free flying site",
        "name_fr": "site de vol libre",
        }
    practice = "Vol libre"
```

BIBLIOGRAPHIE

Privilégier la norme APA

Duchenne, N. (s. d.). *Partenariat avec l'application Komoot - Pays-Bas | Atout France*. https://www.atout-france.fr/fr/catalogue/operations-carte/partenariat-avec-lapplication-komoot

Geotrek. (s. d.). *Geotrek API v2*. https://demo-admin.geotrek.fr/api/v2/

Geotrek community. (s. d.-a). *Geotrek-admin documentation*. https://geotrek.readthedocs.io/en/2.115.1/

Geotrek community. (s. d.-b). *GeotrekCE/Geotrek-admin: Paths management for National Parks and Tourism organizations*. https://github.com/GeotrekCE/Geotrek-admin

Leplatre, M. (s. d.). *La segmentation dynamique | Makina Corpus*. https://makina-corpus.com/sig-cartographie/la-segmentation-dynamique

OpenStreetMap community. (s. d.-a). *OpenStreetMap Statistics*. https://planet.openstreetmap.org/statistics/data_stats.html

OpenStreetMap community. (s. d.-b). *OpenStreetMap Wiki*. https://wiki.openstreetmap.org/wiki/Main_Page

Topf, J. (s. d.). Evolution of the OSM Data Model.

INSA TOULOUSE
Institut National des Sciences Appliquées de Toulouse
135, avenue de Rangueil, 31077 Toulouse Cedex 4 - France
Tél +33 (0)5 61 55 95 13 - Fax +33 (0)5 61 55 95 00

www.insa-toulouse.fr