Web sémantique et Ingénierie des connaissances

Cycling Tour

Auteurs : FALCOZ Alban, GRIPARI Alexandre

Table des matières

I. Cas d'usage

Le projet « Cycling Tour Operator » répond aux besoins d'une agence de cyclotourisme souhaitant moderniser la gestion de son catalogue et de sa relation client grâce aux technologies du Web Sémantique et de l'Intelligence Artificielle.

Les principaux cas d'usage sont :

- **Recherche intelligente de séjours :** Permettre aux utilisateurs de trouver des tours non seulement par nom, mais par critères complexes (difficulté, dénivelé, massif montagneux, distance, ville) grâce à la structuration des données en graphe.
- **Gestion de flotte et maintenance :** Suivi précis de l'état des vélos (opérationnel, en réparation) et de leur assignation aux clients, en utilisant des vocabulaires contrôlés (SKOS).
- **Assistant virtuel (Chatbot) :** Un agent conversationnel capable de répondre en langage naturel à des questions sur les tours, les vélos et les avis, en s'appuyant sur la base de connaissances interne.
- **Système de Recommandation :** Suggestion de nouveaux tours aux clients existants en se basant sur l'historique des réservations d'autres clients similaires.

II. Spécification de l'application

1. Backend

Le backend expose une API RESTful (`/api`) composée des services suivants :

- **SparqlService :** Ce service charge les fichiers Turtle dans un graphe en mémoire via la librairie `rdflib` et applique un raisonneur RDFS (`owlrl`) pour matérialiser les inférences. Il supporte l'endpoint `/api/query` pour l'exécution directe de requêtes et héberge l'algorithme de recommandation accessible via `/api/prediction`.
- **DbpediaService :** Ce service assure l'interopérabilité avec le Web Sémantique. Utilisé par l'endpoint `/api/enrich`.
- **TextToSparqlService :** Accessible via `/api/text-to-sparql`, ce service fait le pont entre le langage naturel et le graphe.
- **ChatbotService :** Ce service implémente une architecture RAG pour l'endpoint `/api/ask`.

2. Interface

L'interface de l'application est structurée en trois onglets distincts :

- Assistant conversationnel : Cet onglet permet d'interroger le système en langage naturel via une interface de chat. Selon le mode sélectionné, la question de l'utilisateur est soit traduite

en requête SPARQL avec affichage de la requête et de son résultat, ou bien il obtient une réponse conversationnelle générée par l'IA.
- Data Explorer: Cet onglet permet aux utilisateurs d'exécuter des requêtes SPARQL sur le graphe. Il intègre une interface de contrôle pour l'enrichissement sémantique : lors de requêtes sur les montagnes, l'utilisateur peut sélectionner dynamiquement quels champs supplémentaires récupérer depuis DBpedia (description, image, site web).
- Prédiction: Cette interface expose le système de recommandation. À partir de l'URI d'un client, le système identifie des profils utilisateurs similaires dans la base de données. Il suggère ensuite des séjours effectués par ces « voisins » mais pas par le client cible. Chaque proposition inclut un score de confiance basé sur la similarité des historiques.

III. Modélisation du Graph de connaissance

Le modèle de données repose sur une ontologie personnalisée (`http://data.cyclingtour.fr/schema#`) qui étend des standards existants pour assurer l'interopérabilité.

1. Modélisation

a. Vocabulaire

- **RDF/RDFS/OWL :** Pour la structure des classes et propriétés (héritage, contraintes).
- **SKOS :** Pour définir les systèmes d'organisation de connaissances. Il est utilisé pour les niveaux de difficulté (`Easy`, `Moderate`, `Hard`, `VeryHard`) et les statuts de maintenance. On utilise `cs:MaintenanceUnavailable` pour la hiérarchie des statuts problématiques, avec des statuts `skos:broader` comme `MaintenanceNeedsService` et `MaintenanceUnderRepair`.
- **FOAF :** Pour modéliser les personnes (`Client`, `Guide`).
- **Schema.org :** Alignement pour les avis (`Review`).
- **DBpedia :** Lien DBpedia pour enrichissement (villes, montagnes) si disponible

b. Classes

Nous avons défini une hiérarchie de classes permettant de couvrir l'ensemble de l'activité du tour opérateur. Voici le détail des classes principales :

- `TourPackage` et `TourStage` : Ces classes nous permettent de structurer l'offre touristique. Un `TourPackage` représente un séjour complet (ex: « Tour du Mont Blanc ») et est composé de plusieurs `TourStage` (étapes), permettant une meilleure flexibilité dans la gestion des itinéraires.
- `Bike` : Représente les vélos utilisés par les clients. La classe est divisée en sous-classes disjointes (`RoadBike`, `MountainBike`) pour distinguer les usages. L'utilisation de `owl:intersectionOf` permet de définir des vélos électriques (`ElectricMountainBike`) comme étant à la fois des vélos de type spécifique et des vélos électriques.

- `Path`, `Mountain` : Ces classes modélisent les données géographiques. `Path` représente un tronçon avec ses caractéristiques physiques (longueur, dénivelé), tandis que `Mountain` sont des points d'intérêt enrichis potentiellement par des données externes (DBpedia).
- `Client` : Sous-classe de `foaf:Person`. Elle représente les utilisateurs finaux de l'application. Elle est enrichie par des propriétés spécifiques comme le numéro de téléphone (`cs:phone`) et le statut de leur tour actuel (`cs:tourStatus`).
- `Guide` : Également sous-classe de `foaf:Person`, le Guide se distingue par ses informations de contact professionnel et son assignation aux tours (`cs:guideAssigned`).
- `TourBooking` : Sous-classe de `Booking`, elle représente la réservation d'un séjour complet (`TourPackage`) et lie un client à un package, une date, et déclenche l'assignation d'un guide.
- `BikeBooking` : Sous-classe de `Booking`. Elle lie un client à une instance spécifique de `Bike` pour une durée donnée.
- `Review` : Alignée avec `schema:Review` pour favoriser le référencement. Cette classe permet aux clients de noter et commenter soit un `TourPackage`, soit un `Bike`. La propriété `cs:reviewsItem` utilise une union de classes pour permettre cette flexibilité.

c. Propriété Algébrique

- **Transitivité :** La propriété `cs:isHigherThan` est transitive. Si la montagne A est plus haute que la B, et la B plus haute que la C, alors A est plus haute que C.

- **Symétrie :** La propriété `cs:isNear` est symétrique. Si une montagne est proche de B, alors B est proche de A.

- **Inversion :**
  - `cs:bookedBy` est l'inverse de `cs:hasBooking`.
  - `cs:reviewedBy` est l'inverse de `cs:hasReview`.
  - `cs:isSmallerThan` est l'inverse de `cs:isHigherThan`.

- **Chaîne de propriétés (Property Chain Axiom) :** La propriété `cs:includesPath` est définie par une chaîne : elle relie un `TourPackage` à un `Path` via la composition de `cs:includesStage` et `cs:stagePath`.

2. Construction du graphe de connaissance

a. RML Mapping

Nous partons d'un dataset disponible sur kaggle qui contient les différentes étapes du tour de France de 1903 jusqu'à 2022.

i. Enrichissement des données

Le jeu de données initial, restreint aux villes de départ et d'arrivée ainsi qu'à la distance, a fait l'objet d'un enrichissement sémantique ciblé sur les étapes postérieures à 2008, la documentation antérieure s'avérant lacunaire. Face au manque d'exhaustivité de DBpedia pour

les données topographiques spécifiques, la stratégie d'enrichissement a privilégié Wikipédia comme source primaire : à partir de la page de chaque étape, identifiée par sa date, nous avons extrait le dénivelé positif et la liste des montagnes traversées. Pour chaque col ou montagne recensé, un processus de réconciliation a permis de vérifier son existence sur DBpedia afin d'établir un lien dans la mesure du possible (`owl:sameAs`), tandis que les données détaillées (altitude, massif, coordonnées géographiques, pays) ont été récupérées via les info boxes Wikipédia.

ii. Préparation et Transformation des Données

Avant l'exécution du moteur RML, les données brutes issues du scraping sont normalisées pour s'adapter aux contraintes du Web Sémantique.

- **Normalisation des coordonnées :** Les coordonnées géographiques récupérées sur Wikipédia sont converties en format décimal pour être compatibles avec l'ontologie `wgs84_pos` (propriétés `geo:lat` et `geo:long`).
- **Calcul de difficulté :** La difficulté d'une étape est pré-calculée selon le dénivelé positif (`Elevation_Gain`). Le script assigne une catégorie (`Easy`, `Moderate`, `Hard`, `VeryHard`) qui est ensuite directement mappée vers l'URI correspondante dans le schéma.
- **Hiérarchie topologique :** Une logique spécifique a été implémentée pour peupler la propriété `cs:isHigherThan`. Le script trie les montagnes par altitude et assigne à chacune l'URI de la montagne immédiatement inférieure, créant ainsi une chaîne ordonnée directement exploitable.

iii. Stratégie de Mapping (Mountain & Path)

Le fichier `mapping.ttl` définit les règles de transformation pour générer un fichier ttl, contenant les chemins et les montagnes concernés et qui sont conformes à nos schémas.

iv. Gestion des Listes avec FnO

Une complexité majeure réside dans le traitement de la relation `cs:includesMountain`. Dans le fichier CSV, une étape peut contenir plusieurs montagnes séparées par des virgules. Le lien est donc effectué grâce à une fonction fnO qui parse la chaîne de caractère.

b. Unstructured data to RDF

Pour pallier l'absence de données réelles pour les vélos et les clients, nous avons mis en place un pipeline de Web Scraping pour les obtenir.

i. Acquisition des vélos

Le script `bike_scraping.py` exploite le catalogue en ligne de Decathlon pour constituer la flotte de vélos.

Pour cela, on récupère des métadonnées comme le nom, la description, le prix d'achat et on classe automatiquement les vélos dans l'ontologie (`cs:MountainBike`, `cs:ElectricRoadBike`, etc.) en détectant des mots-clés spécifiques (ex: « Rockrider », « VAE », « Route ») dans la

description. Le prix de location journalier (`cs:pricePerDayBike`) est calculé dynamiquement (1.5% du prix de vente).

ii. Création des clients et des avis

Ensuite, pour peupler le graphe avec des données sociales réalistes, nous récupérons les avis clients réels via l'API de Decathlon. Chaque auteur d'avis devient une instance `cs:Client` et les notes et commentaires sont modélisés en entités `cs:Review`, liées au vélo concerné et au client.

iii. Génération des packages

La logique de constitution des offres touristiques est implémentée dans le script `tour_generator.py`.

Le script procède en deux phases distinctes :

1. Le système parcourt l'ensemble des chemins (`cs:Path`) présents dans le graphe. Pour chaque chemin, une entité `cs:TourStage` est créée. À cette étape, un guide est assigné aléatoirement parmi la liste des guides disponibles et la durée de l'étape est estimée, en se basant sur une vitesse moyenne de 22 km/h. Une liste d'adjacence est construite dynamiquement pour modéliser la topologie du réseau (Ville de départ *to* Étape *to* Ville d'arrivée).

2. Pour créer un séjour complet, l'algorithme effectue des « marches aléatoires » sur le graphe des connexions :
   - Il itère sur chaque ville de départ disponible.
   - Il construit une chaîne de 3 à 5 étapes consécutives en s'assurant que la ville d'arrivée de l'étape N correspond à la ville de départ de l'étape $N + 1$.
   - Si une chaîne valide (au moins 3 étapes) est formée, un `TourPackage` est généré.

   Le tarif du `TourPackage` est ensuite fixé à 120€ par jour multiplié par le nombre d'étapes tandis que la durée sur la base de la distance totale divisée par une vitesse touristique de 18 km/h.

iv. Génération des bookings

Afin de densifier le graphe, nous avons simulé un historique de réservations riche pour chaque client identifié via les avis.

Le script `tour_booking_creation.py` récupère les `TourPackage` créés pour associer aléatoirement 4 à 8 `TourPackage` à chaque client, en s'assurant que les dates des `TourBooking` ne se chevauchent pas et en assignant le bon guide. La comparaison entre les dates de réservation et la date courante permet de déduire le statut du client (`In Progress`, `Finished`, `Not Started`).

Parallèlement, lorsqu'un avis est récupéré sur un vélo, une réservation correspondante est créée (`cs:BikeBooking`) pour lier le client à l'instance du vélo (`cs:Bike`) sur la période des `TourBooking`.

IV. Fonctionnalités

1. Questions de compétences

a. Quel est le dénivelé total cumulé pour chaque "Tour Package" et quel est le guide responsable de ce tour ?

```SPARQL
PREFIX cs: <http://data.cyclingtour.fr/schema#>
PREFIX cto: <http://data.cyclingtour.fr/data#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>

SELECT ?tourLabel ?guideName (SUM(?elevation) AS ?totalElevation)
WHERE {
  ?tour a cs:TourPackage ;
        rdfs:label ?tourLabel ;
        cs:guideAssigned ?guide ;
        cs:includesStage ?stage .

  ?guide foaf:name ?guideName .

  ?stage cs:stagePath ?path .
  ?path cs:elevationGain ?elevation .

FILTER (?elevation > 0)
}
GROUP BY ?tourLabel ?guideName
ORDER BY DESC(?totalElevation)
```

b. Quels "Tour Packages" ont des étapes avec une difficulté "Hard" ou "Very Hard", mais sans étapes "Easy", et quel est le prix moyen par jour ainsi que le nombre de clients ayant réservé ces tours ?

```SPARQL
PREFIX cs: <http://data.cyclingtour.fr/schema#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>

SELECT ?tourLabel ?price (AVG(?elevation) AS ?averageElevation)
(COUNT(DISTINCT ?client) AS ?nbClients)
WHERE {
```

```
6     ?tour a cs:TourPackage ;
7           rdfs:label ?tourLabel ;
8           cs:pricePerDayTour ?price .
9
10    ?tour cs:includesStage/cs:stagePath/cs:difficulty ?difficulty .
11    FILTER (?difficulty IN (cs:Hard, cs:VeryHard))
12
13    FILTER NOT EXISTS {
14      ?tour cs:includesStage ?stageEasy .
15      ?stageEasy cs:stagePath ?pathEasy .
16      ?pathEasy cs:difficulty cs:Easy .
17    }
18
19    ?tour cs:includesStage ?stage .
20    ?stage cs:stagePath ?path .
21    ?path cs:elevationGain ?elevation .
22
23    FILTER (?elevation > 0)
24
25    OPTIONAL {
26      ?booking cs:tourPackageBooked ?tour ;
27               cs:bookedBy ?client .
28    }
29  }
30  GROUP BY ?tourLabel ?price
```

c. Quels sont les « Couples de Géants », des montagnes de haute altitude (> 1000m) qui sont directement reliées entre elles, et quel est le dénivelé qu'il faudrait pour passer de l'une à l'autre ?

```SPARQL
1   PREFIX cs: <http://data.cyclingtour.fr/schema#>
2   PREFIX cto: <http://data.cyclingtour.fr/data#>
3   PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
4   PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
5
6   SELECT
7     ?massif
8     ?sommet1
9     ?alt1
10    ?sommet2
11    ?alt2
12    ?ecartAltitude
13    ?statut
```

```
14 WHERE {
15   ?m1 a cs:Mountain ;
16       rdfs:label ?sommet1 ;
17       cs:elevation ?alt1 ;
18       cs:mountainRange ?rangeUri .
19
20   ?m1 cs:isNear ?m2 .
21   ?m2 a cs:Mountain ;
22       rdfs:label ?sommet2 ;
23       cs:elevation ?alt2 .
24
25   FILTER (?alt1 > 1000 && ?alt2 > 1000 && ?alt1 > ?alt2)
26   BIND ((?alt1 - ?alt2) AS ?ecartAltitude)
27   BIND (
28     IF(?ecartAltitude < 100, "Traversée Plane",
29     IF(?ecartAltitude < 500, "Vallonné", "Montagneux"))
30     AS ?statut
31   )
32   BIND (REPLACE(STR(?rangeUri), "^.*/", "") AS ?massif)
33 }
34 ORDER BY DESC(?alt1)
35 LIMIT 15
```

2. Link Prediction

Cette fonctionnalité est conçue pour apporter une valeur ajoutée aux clients en leur suggérant des tours de vélo susceptibles de leur plaire.

Le mécanisme de recommandation repose sur une analyse comparative de l'historique des réservations (**bookings**). Le système récupère les données du client cible (identifié par son URI) et les confronte aux réservations des autres utilisateurs de la base. En appliquant la mesure de **similarité de Jaccard** (Intersection over Union), l'algorithme calcule la proximité entre les profils pour en déduire de nouvelles suggestions de réservation pertinentes.

3. Query

Ce module constitue le moteur d'interrogation direct du système. Il offre une interface brute permettant d'exécuter des requêtes SPARQL sur le graphe de connaissances local. Cette fonctionnalité est essentielle pour les administrateurs ou les composants logiciels nécessitant une extraction de données précise et flexible, sans passer par des filtres prédéfinis. Elle garantit un accès complet à la structure sémantique des données stockées.

4. Enrichment

La fonction d'enrichissement vise à augmenter la base de connaissances locale avec des informations provenant du Web sémantique ouvert.

Le processus se déroule en trois étapes :

- **Exécution locale :** Le système récupère d'abord les données internes correspondant à la demande.
- **Détection de liens :** Il identifie les propriétés `sameAs` qui relient les entités locales à des ressources externes (principalement DBpedia).
- **Fusion de données :** Le service interroge les sources distantes pour récupérer des méta-données supplémentaires (descriptions, images, résumés) et les fusionne dynamiquement avec les résultats locaux. Cela permet de présenter à l'utilisateur une information plus riche que celle stockée localement.

5. Chatbot

Le module Chatbot agit comme une interface conversationnelle intelligente, facilitant l'interaction entre l'utilisateur humain et la base de données technique. Il se décline en deux modes de fonctionnement distincts.

a. Text to SPARQL

Cette sous-fonctionnalité comble le fossé sémantique entre le langage naturel et le langage de requête formel. En donnant dans le contexte le schéma de la base de données avec les classes, leurs propriétés et 3 exemples tirés des données à un modèle de langage (Gemini), le système traduit une question posée en français ou en anglais (ex: « Quels sont les tours difficiles ? ») en une requête SPARQL syntaxiquement correcte et exécutable. Cela permet aux utilisateurs non-techniques d'interroger la base de données sans connaître le langage SPARQL.

b. Text to Text

Ce mode correspond à l'interaction conversationnelle classique, permettant à l'utilisateur de poser des questions ouvertes au système.

Le mécanisme de réponse repose sur une contextualisation dynamique via une recherche vectorielle. Le système calcule la **similarité cosinus** entre la représentation vectorielle (**embedding**) de la requête utilisateur et celle des données du graphe de connaissances. Cette méthode permet d'extraire le contexte le plus pertinent, qui est ensuite transmis au LLM pour générer une réponse en langage naturel précise et fidèle aux données.

V. Utilisation de L'IA

Nous nous sommes aidés de chat GPT pour améliorer la tournure des phrases de notre rapport et pour faciliter dans la conception de l'interface.