

Cycling Tour Operator (CTO)

Radhi ALOULOU - Aymeric NURDIN

Concepts principaux

- Forfait touristique (TourPackage): Offre commerciale d'un séjour (ex: cto:GrandEuropeanTour).
- Destination (CyclingDestination): Lieu géographique d'un séjour (ex: cto:Paris, cto:FrenchAlps).
- Parcours (CyclingPath): Itinéraire cyclable spécifique (ex: cto:SeineRiverTrail).
- Client (Client): Personne réservant les services (ex: cto:Client_AymericNurdin).
- Guide (Guide): Personne accompagnant les tours (ex: cto:Guide_JulieMartin).
- Réservation (Booking): Instance d'une réservation d'un forfait par un client (ex: cto:Booking1).
- Type de vélo (BikeType): Catégorie de vélo (ex: cto:RoadBike, cto:MountainBike).
- Location de vélos (BikeRentalShop): Magasin partenaire pour la location (ex: cto:ParisBikeRentals).
- Option de tour (TourOption): Activité optionnelle (ex: cto:SeineBoatTour).
- Discipline cycliste (CyclingDiscipline): Style de cyclisme associé à une destination (ex: cto:Mountain_Climbing).

Liens créés

- Liaison des forfaits et parcours: Un TourPackage inclut un ou plusieurs CyclingPath (ctosch:includesPath).
- Localisation des parcours: Un CyclingPath est situé dans une CyclingDestination (ctosch:heldIn).
- Gestion des réservations: Une Booking est faite par un Client (ctosch:bookedBy) pour un TourPackage (ctosch:forTourPackage).
- Assiguation des guides: Un TourPackage a un Guide assigné (ctosch:assignedGuide) ; un Guide est certifié pour des destinations (ctosch:certifiedFor).
- Gestion des vélos: Liaison entre les vélos inclus dans un tour (ctosch:includesBikeType), recommandés pour un parcours (ctosch:recommendedBikeType), et offerts par un loueur (ctosch:offersBikeType).
- Options de réservation: Un TourPackage propose des TourOption (ctosch:availableOptionalActivity), qu'une Booking peut sélectionner (ctosch:selectedOptions).
- Partenariats de location: Une Booking est associée à un BikeRentalShop partenaire (ctosch:bikeRentalPartner).

Réutilisation de vocabulaire existant

Nous avons réutilisé plusieurs vocabulaires existants. [Schema.org](#) a été employé pour les propriétés descriptives générales telles que schema:name, schema:email et schema:address. Nous nous sommes appuyés sur [DBpedia](#) pour lier nos entités à des ressources externes (par exemple, dbp:France, dbp:Tunisia) et pour définir des concepts (comme dbp:Spring ou dbp:Sand). Le vocabulaire [FOAF](#) a été utilisé pour typer nos classes Client et Guide en tant que sous-classes de foaf:Person. Enfin, les vocabulaires de base [RDFS](#) et [OWL](#) ont servi à la définition même du schéma (par exemple, rdfs:Class, rdfs:label, owl:sameAs), tandis que [XSD](#) a permis de spécifier les types de données littérales pour nos propriétés (comme xsd:integer, xsd:float et xsd:date).

Vocabulaires créés

Notre schéma (préfixe ctosch:), défini dans cto_schema.ttl, étend ces vocabulaires avec des classes spécifiques au domaine du tourisme/cyclisme (ex: ctosch:TourPackage, ctosch:CyclingPath, ctosch:BikeRentalShop, ctosch:Booking) et des propriétés spécifiques pour décrire les relations métier (ex: ctosch:includesPath, ctosch:bookedBy, ctosch:difficultyLevel, ctosch:bestCyclingSeason, ctosch:bikeRentalPartner).

Fichiers

- cto_schema.ttl: L'ontologie définissant le vocabulaire (classes et propriétés) de l'opérateur de tours.
- cto_data.ttl: Le graphe RDF d'instance (en Turtle) contenant les données factuelles sur les destinations, tours, clients, réservations, etc.
- cto_shapes.ttl: Un graphe SHACL définissant les contraintes de validation pour assurer la qualité et la conformité des données dans cto_data.ttl.
- cto_query: fichier contenant des requêtes pour explorer et analyser le graphe

Organisation des données

Pour l'organisation des données, nous avons maintenu une séparation claire entre notre vocabulaire (schéma ctosch:) et les instances (données cto:), en identifiant chaque entité par une URI unique (ex: cto:GrandEuropeanTour, cto:Client_AymericNurdin). Notre approche a été de privilégier l'utilisation de ressources (URI) dès que possible, en nous liant à des vocabulaires existants comme DBpedia ou en référençant nos propres entités, afin d'éviter l'usage de littéraux là où un lien était plus approprié. Lorsque les littéraux étaient nécessaires, par exemple pour les valeurs numériques, les dates ou les descriptions textuelles, nous les avons systématiquement typés en utilisant XSD (ex: "10"^^xsd:integer, "1800"^^xsd:float, "2025-05-01"^^xsd:date) et avons fourni des métadonnées descriptives, telles que rdfs:label et rdfs:comment, avec des libellés multilingues (ex: @en, @fr).

SHACL

Pour assurer la qualité des données, nous avons mis en œuvre une validation via SHACL, dont les contraintes sont définies dans le fichier cto_shapes.ttl. Ces contraintes, exprimées sous forme de sh:NodeShape, vérifient des aspects essentiels tels que la cardinalité (par exemple, sh:minCount 1 pour la propriété ctosch:bookedBy d'une ctosch:Booking), les types de données (par exemple, sh:datatype xsd:integer pour ctosch:difficultyLevel), ainsi que les plages de valeurs autorisées (par exemple, sh:lessThanOrEqualTo 5 pour la difficulté, ou sh:in ("pending" "confirmed" "cancelled") pour la propriété ctosch:status).

Modalité de travail

Concernant nos modalités de travail nous avons eu recours à l'IA pour leur génération des données via un processus de prompts itératifs. Cela veut dire que c'est nous, qui définissons les préfixes/namespaces, les propriétés et beaucoup de ressources puis nous réadaptions les données générées. L'objectif derrière était de construire un jeu de données unique à nous mais aussi permettant de valider des requêtes SPARQL complexes.

Pour cela, nous avons intentionnellement inclus des cas de figure variés, y compris des scénarios de négation ou de complétude, comme :

- L'identification des réservations où le client a sélectionné toutes les options disponibles dans un tour
- La vérification des réservations où le partenaire de location couvre 100% des types de vélos disponibles pour un tour.

Nous avons également insisté pour intégrer des ressources de vocabulaires existants (ex: dbp:Sand). Nous avons aussi utilisé Corese pour valider la qualité des données avec SHACL, mais également pour vérifier l'exactitude de nos requêtes SPARQL et corriger, en conséquence, soit les données, soit les requêtes.

Conclusion

Notre modélisation de l'opérateur de tour de cyclisme offre une représentation riche, exploitant les ressources existantes tant pour les propriétés que pour les données, et permet une interrogation complexe du graphe.