MINES Saint-Étienne

Master 2 DSC

2025 - 2026

Project Report

Building a Knowledge Graph from Tolkien Gateway Wiki

Semantic Web

January 16, 2026

Authors :

Alpha DIALLO
Ana Salimata SANOU

Supervisors :

Mr. Antoine Zimmermann
Mr. Victor Charpenay

Table of Contents

1. Introduction

1.1 Project Context

The Semantic Web, or Web of Data, envisions a web where information is not only human-readable but also machine-understandable and interconnected. Knowledge graphs, materialized through the RDF (Resource Description Framework) framework, are a cornerstone of this vision. Projects like DBpedia, which extracts structured content from Wikipedia for publication as Linked Data, have demonstrated the power of this approach for integrating and querying vast knowledge bases. Our project follows this lineage by applying it to a niche but rich domain: the literary and mythological universe created by J.R.R. Tolkien, via the collaborative Tolkien Gateway platform.

1.2 Problem Statement

Tolkien Gateway is a valuable information source, but its content remains locked in the semi-structured format of a wiki. How can we transform these thousands of articles—with their infoboxes, categories, and internal links—into a true, queryable knowledge graph? The challenges are manifold: cleanly extracting semantics from WikiText, choosing an appropriate vocabulary (balancing standardization and expressiveness), integrating complementary data, and finally publishing the result in an interoperable way. The goal is to create a resource that preserves the richness of the original wiki while offering the querying and inference capabilities of the Semantic Web.

1.3 Objectives

The objectives of this project are clear and measurable:

1. **Extract** relevant content from over 13,000 Tolkien Gateway articles.
2. **Transform** this data into an RDF graph primarily aligned with the widely adopted schema.org vocabulary, while defining a specialized vocabulary (tgo) for Tolkien-specific concepts.
3. **Enrich** this graph by integrating external data, notably illustrations and metadata from the *Middle-earth: The Wizards (METW)* card game, as well as alignments with external knowledge bases (DBpedia, YAGO).
4. **Validate** the consistency and quality of the generated data using the SHACL constraint language.
5. **Publish** the graph as Linked Data, implementing HTTP content negotiation to serve both human-readable (HTML) and machine-readable (RDF/Turtle, JSON-LD) representations of the same resources.
6. **Demonstrate** the added value through exploratory queries and the discovery of implicit relations via inference.

2. Project organization and Task distribution

2.1 Approach to Work Division

We adopted a complementary skills-based approach rather than a modular division of work. Each team member was assigned tasks with equivalent technical complexity, but in different specialized domains that leveraged our individual strengths while ensuring balanced workload distribution. For greater efficiency, we used these tools :

- **Git :** for version control;
- **Discord** and **whatsApp :** for communication;
- **Google Docs :** for writing the report.

2.2 Individual Responsibilities

- ### Ana Salimata SANOU – Data & Semantics Focus

Ana focused on the core data processing pipeline, handling the extraction of content from Tolkien Gateway via the MediaWiki API, transforming the semi-structured wiki data into RDF, and establishing alignments with external knowledge bases like DBpedia. Her work formed the foundational semantic layer of the project.

- ### Alpha DIALLO – Systems & Quality Focus

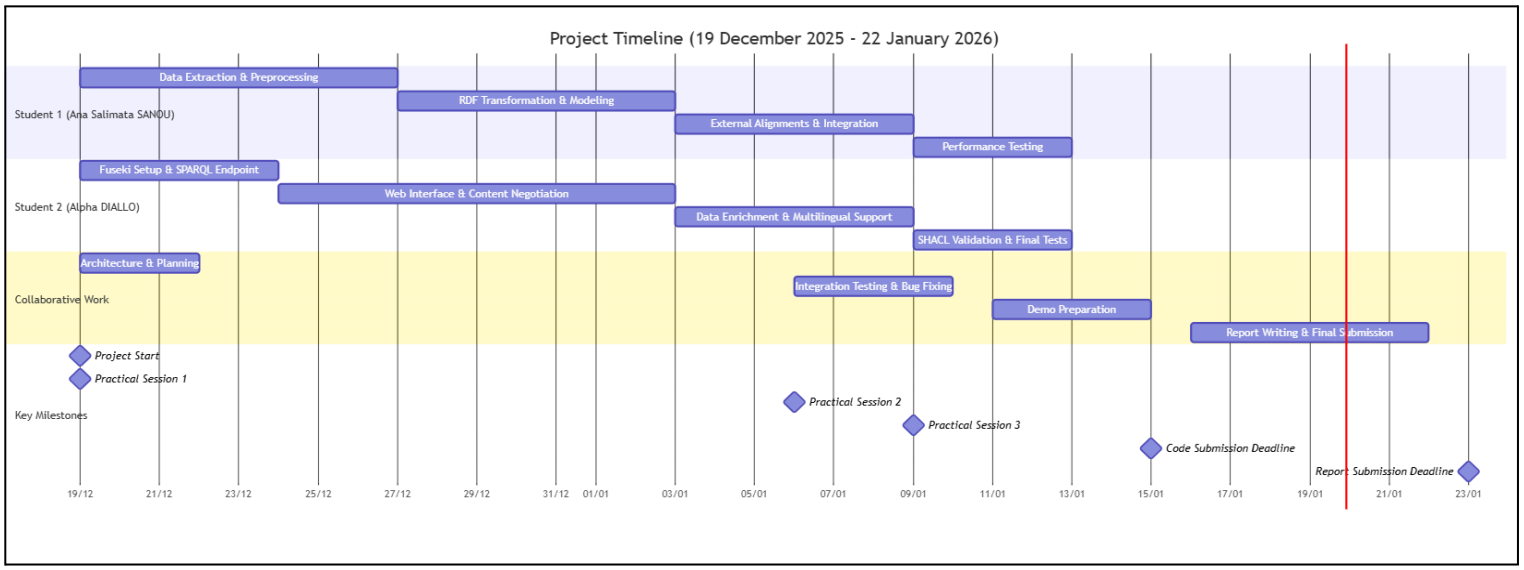
Alpha managed the system infrastructure and quality assurance, setting up the Fuseki triplestore and SPARQL endpoint, developing the web interface with content negotiation, and implementing SHACL validation to ensure the consistency and integrity of the final knowledge graph.

2.3 Collaborative Work

Key phases of the project required joint effort. We collaborated on the initial system architecture, conducted integration testing to ensure all components worked seamlessly, prepared the live demonstration, and co-authored the final report and documentation.

2.4 Project Timeline Rationale

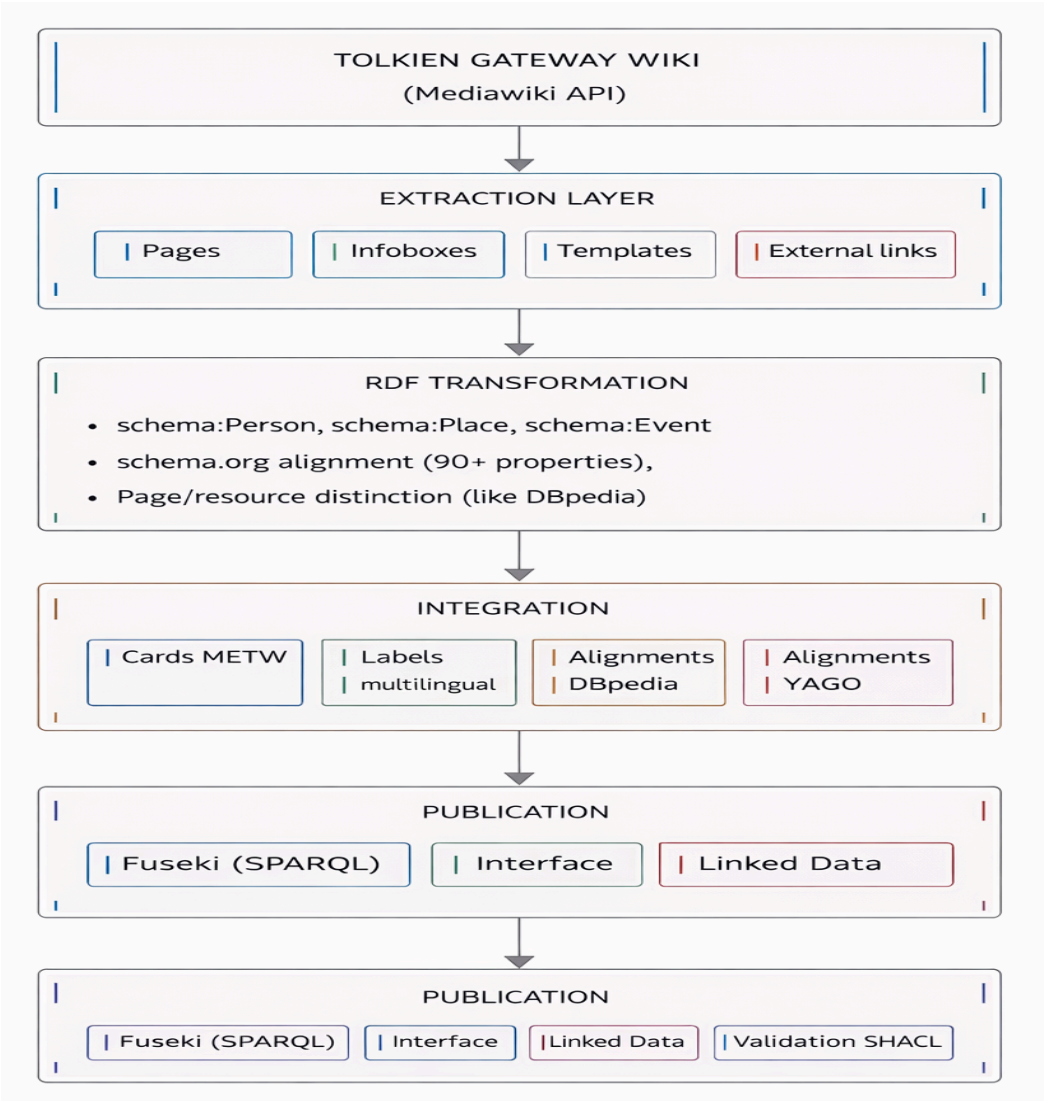
The following Gantt chart illustrates our **phased approach** to project execution:



3. Methodology

3.1 Overall System Architecture

Our architecture follows a classic ETL (Extract, Transform, Load) pipeline adapted for the Semantic Web:



3.2 Data Collection and Extraction

- **Source**: All articles from the main namespace of Tolkien Gateway.
- **Method**: Use of the action=parse API to retrieve the HTML and WikiText of each page. A local caching system was implemented to respect rate limits and speed up development.

- **Focus on Structure**: Systematic identification of infobox templates (character, location, item) to extract key-value pairs, which form the raw material for our graph.

3.3 Transformation and RDF Modeling

- **Vocabulary Choice**: schema.org as the foundation for generic concepts (Person, Place, CreativeWork).

  A custom Tolkien Gateway Ontology (TGO) was created for specific concepts (**tgo**:hasRace, **tgo**:hasCardRepresentation).

- **Mapping**: Definition of rules to transform each infobox field into one or more RDF triples.

  Example: | name = Aragorn becomes:

  ```
  Aragorn a schema:Person ; schema:name "Aragorn"@en.
  ```

- **Multilingualism Handling**: Preservation of labels in all available languages (rdfs:label), with language indication via tags **@en**, **@fr**, etc.

3.4 Data Enrichment and Alignment

- **METW Card Integration**: An external JSON dataset listing cards from the *Middle-earth: The Wizards* card game was merged with the graph. A card is linked to a character or place via the property **`tgo:hasCardRepresentation`**.

- **External Alignments**: Search and creation of owl:sameAs links between entities in our graph and their equivalents in DBpedia and YAGO where possible, fostering integration into the *Web of Data*.

- **Simple Inference**: A few RDFS/OWL rules were defined to deduce relations not explicitly stated in the wiki (e.g., enmity between factions).

3.5 Validation and Quality Assurance

- **Constraint Validation (SHACL)**: SHACL shapes were written to ensure data integrity.

  For example: Every instance of **schema:Person** must have exactly one **schema:name** property.

- **Logical Consistency**: Basic checks for type consistency and proper usage of properties.

3.6 Publication and Data Exposure

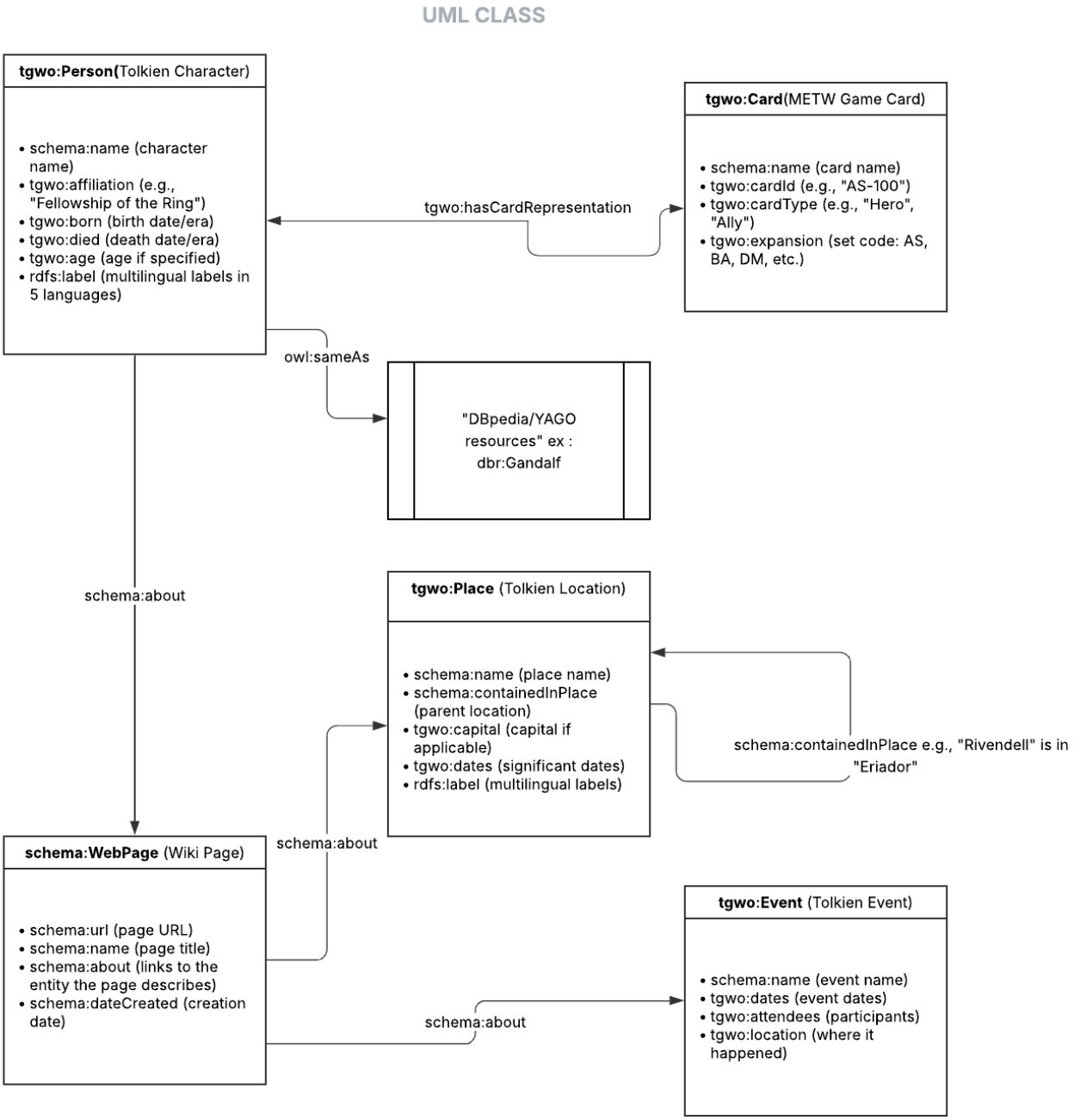
- **Triplestore**: The final graph is loaded into an **Apache Jena Fuseki** server, which provides a standard SPARQL endpoint.
- **Web Interface**: A lightweight **Flask** application offers a user-friendly HTML interface for browsing entities.
- **Linked Data & Content Negotiation**: The Flask server is configured to support HTTP content negotiation. A single URI (e.g., **/resource/Gandalf**) returns HTML if requested by the client (Accept: **text/html**) or RDF/Turtle (Accept: **text/turtle**), thereby implementing the principle of *URI dereferencing*.

4. Implementation

4.1 Technology Stack

- **Languages**: Python 3.10+ (for extraction, transformation, and the web app), SPARQL 1.1 (for querying).
- **Key Python Libraries**:
    - rdflib: For RDF graph manipulation and generation.
    - Flask: Web framework for creating the interface and managing content negotiation.
    - BeautifulSoup4 & regex: For parsing HTML and WikiText.
    - pyshacl: For executing SHACL validation.
    - requests: For HTTP calls to the MediaWiki API and external endpoints.
- **Infrastructure**:
    - **Apache Jena Fuseki**: RDF database server (triplestore) with SPARQL endpoint.
    - **Git**: For version control.
    - **Primary Serialization Format**: Turtle (.ttl) for its readability.

4.2 UML Class Diagram



Our `tgwo:` ontology specialized for Tolkien's universe extends the standard `schema.org` vocabulary by adding domain-specific properties such as affiliation, Tolkien-era dates, and game card relationships, while maintaining interoperability with the global semantic web.

4.3 Application Interfaces

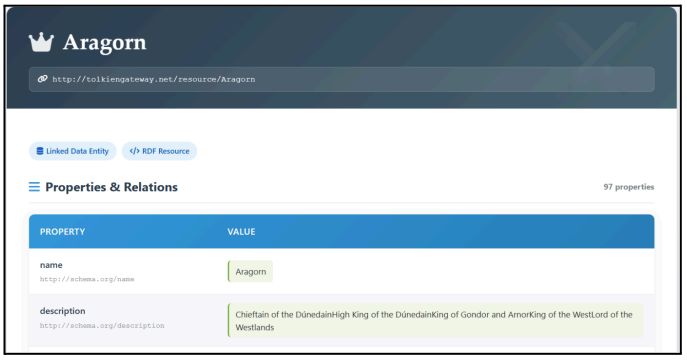
4.3.1 Web Application Interface (Flask)

A. Main Dashboard

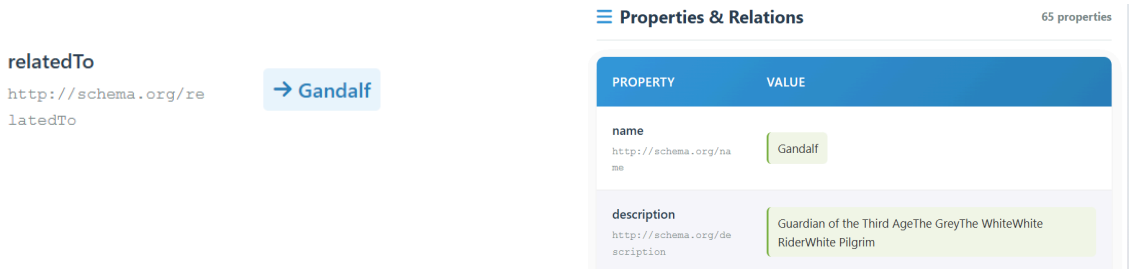
In the main page, we can have categories, like: **Most Described Characters, Important Locations, Creative Works, Events, Implicit Facts Discovery.**

By choosing entities, we can have the description of his properties and relationships.

- For example, if we the Character **Aragorn**, we can have :

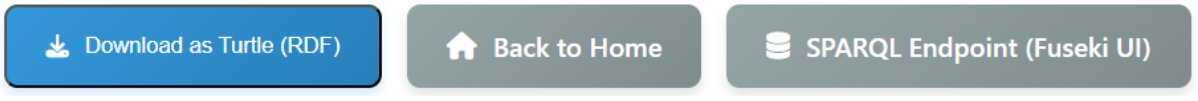


With internal relationships : on peut voir les propriétés de Gandalf

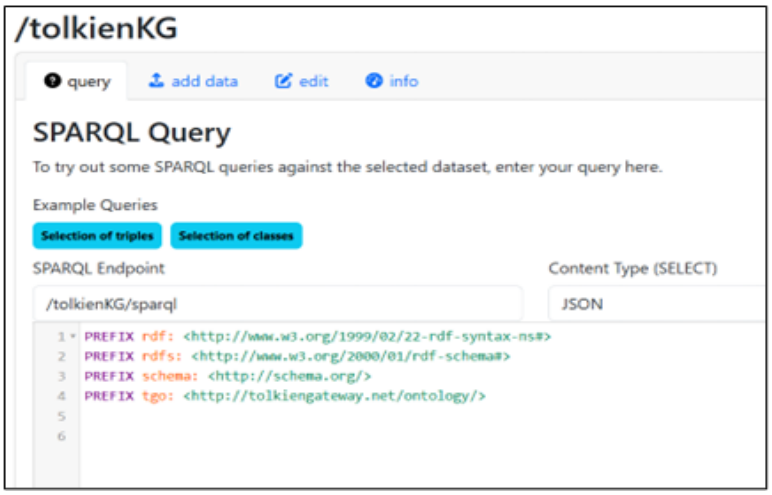


With external relationships :

We can also Download the **turtle file** of an entity or use **SPARQL endpoint** :



*B. Testing queries with **SPARQL endpoint** :*



For example :

- Query 1: Get first 5 persons with their names :

```
SELECT ?s ?name WHERE {
  ?s a schema:Person ; schema:name ?name
} LIMIT 5
```

s	name
1<http://tolkiengateway.net/resource/Herubrand>	Herubrand
2<http://tolkiengateway.net/resource/Alan_Gutierrez>	Alan Gutierrez
3<http://tolkiengateway.net/resource/Hannar>	Hannar
4<http://tolkiengateway.net/resource/Tom_Pickthorn>	Tom Pickthorn
5<http://tolkiengateway.net/resource/Glorfindel>	Glorfindel

- Query 2: Count total number of persons :

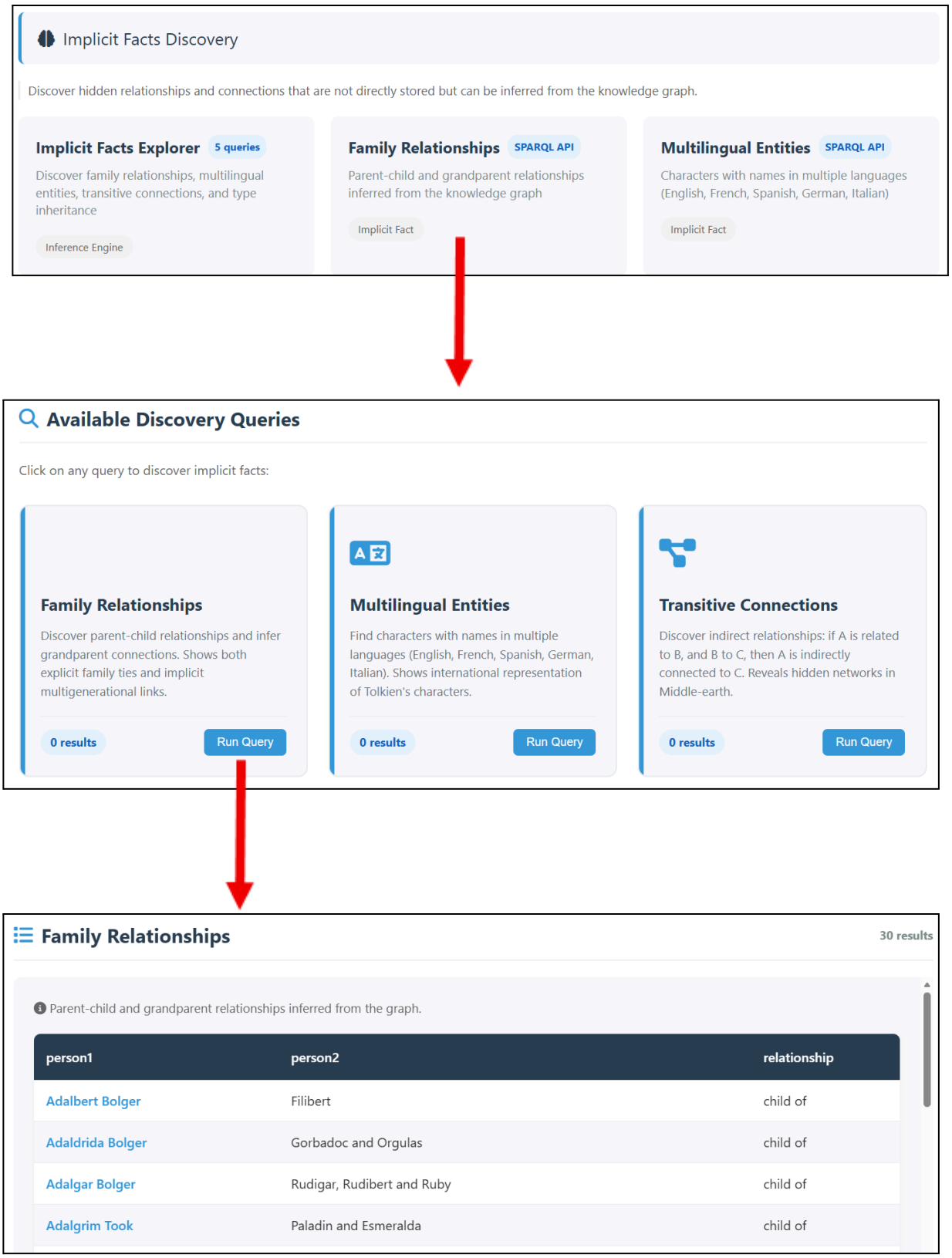
```
SELECT (COUNT(DISTINCT ?s) as ?people) WHERE {
  ?s a schema:Person
}
```

- Query 3: Count total triples in the knowledge graph

```
SELECT (COUNT(*) AS ?totalTriples) WHERE {
  ?s ?p ?o .
}
```

C. Discovering implicit facts :

We can discover many implicit facts relationships, as can see in the following figures :



4.3.2 Using CURL :

For example, in Windows :

- Query : Labels multilingues de Gandalf :

curl -X POST http://localhost:3030/tolkienKG/query -H "Content-Type: application/sparql-query" --data "PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#> SELECT ?label WHERE { <http://tolkiengateway.net/resource/Gandalf> rdfs:label ?label }"


```
{ "head": {
    "vars": [ "label" ]
  } ,
  "results": {
    "bindings": [
      {
        "label": { "type": "literal" , "xml:lang": "fr" , "value": "Gandalf" }
      } ,
      {
        "label": { "type": "literal" , "xml:lang": "en" , "value": "Gandalf" }
      } ,
      {
        "label": { "type": "literal" , "xml:lang": "es" , "value": "Gandalf" }
      } ,
      {
        "label": { "type": "literal" , "xml:lang": "it" , "value": "Gandalf" }
      } ,
      {
        "label": { "type": "literal" , "xml:lang": "de" , "value": "Gandalf" }
      }
    ]
  }
}
```


4.3.3 Using SHACL Validation :

Example : Validation with RDFS Inference

- Query : SHACL validation with RDFS reasoning enabled

```
pyshacl kg/final_knowledge_graph.ttl -s Data_Requetes/shapes.ttl
-i rdfs -f human
```

Explanation: Enables RDFS reasoning before validation (e.g., rdfs:subClassOf, rdfs:range).

Result:

- Validation Report
- Conforms: True

5. Discussion

5.1 Limitations and Ongoing Difficulties

- **Coverage**: Some complex articles or those without standard infoboxes may have been processed less effectively.
- **Maintenance**: The graph is a snapshot. Updating it with changes from the source wiki is not automated.
- **Inference Complexity**: The implemented inference rules are basic. A richer OWL ontology would allow deeper reasoning.
- **Large-scale Performance**: For a wiki comparable in size to Wikipedia, tools like the DBpedia Framework would be more suitable than our ad-hoc Python scripts.

5.2 Improvement Perspectives and Future Work

- **Administration Interface**: A backend for manual updates or triggering re-extractions.
- **Advanced Dashboard**: Interactive visualizations (relationship graph, map of places, timeline).
- **Recommendation System**: Using semantic paths to suggest related characters, places, or articles.
- **Export to Other Formats**: Automatic generation of CSV dumps, GraphQL API, or RDFa to enrich the original website.
- **Engaging a Contributor Community**: Enabling Tolkien experts to enrich or correct the graph via a simplified interface.

6. Conclusion

This project achieved its primary objective: transforming Tolkien Gateway, a static wiki resource, into a **dynamic and interoperable knowledge graph**. By traversing all stages of the Linked Data lifecycle—extraction, modeling, enrichment, validation, and publication—we have created a resource that enhances existing content while opening new possibilities for exploration and analysis.

Semantic Web technologies (RDF, Schema.org, SPARQL, SHACL) proved perfectly suited for structuring and giving meaning to a complex narrative and descriptive corpus. The successful implementation of content negotiation underscores our commitment to Linked Data principles.

Beyond Tolkien's universe, this work demonstrates the feasibility and utility of applying these methods to specialized knowledge domains. It provides a solid foundation for future applications in information retrieval, recommendation, or visualization, and offers a tangible example of how the Semantic Web can forge deeper links between information and its users.

7. References

1. MediaWiki API Documentation - https://www.mediawiki.org/wiki/API:Main_page
2. RDF 1.1 Primer - W3C Recommendation
3. Schema.org Vocabulary - https://schema.org/
4. SHACL Specification - W3C Recommendation
5. DBpedia Project - https://www.dbpedia.org/
6. YAGO Knowledge Base - https://yago-knowledge.org/
7. Apache Jena Documentation - https://jena.apache.org/
8. Python RDFLib Library - https://rdflib.readthedocs.io/

8. Appendice

8.1 Source Code Repository

Link to Git repository with complete implementation :
https://github.com/alphambd/tolkien_knowledge-graph