Semantic Web project

The Semantic Web project is a large and long practical exercise that consists in integrating all the pieces that have been seen during the first sessions into a consolidated Web application. To make sure you can advance sufficiently fast to cover everything, you are allowed to work by pair.

Main objective

The objective of the project is to **build a Knowledge Graph (KG) from a wiki**, as DBpedia or YAGO have been built from Wikipedia. As a starting point, you will use The Tolkien Gateway, an encyclopedia about J.R.R. Tolkien's legendarium with 13k+ articles.

Requirements

- The KG must capture the wiki's content as much as possible. Every page should correspond to an entity in the KG. Infoboxes on wiki pages should translate into RDF triples. If a link exists across wiki pages, there should be an RDF triple between the corresponding entities.
- KG entities must have labels in multiple languages (wherever possible).
- The KG should be validated against schemas derived from wiki templates.
- The vocabulary of the KG should be consistent and reuse schema.org or the DBpedia ontology (but not both) as much as possible.
- The KG should have links to other Web pages, especially those of the original wiki, and to resources in other KGs.
- The KG should be accessible through a SPARQL endpoint.
- Answers to SPARQL queries on the KG should include implicit facts.
- Each entity, class or property of the KG should be accessible through a Linked Data interface, serving Turtle and (basic) HTML.
- The Linked Data interface may build upon the SPARQL endpoint (that is, the description of an entity, class or property is the result of a SPARQL query evaluated on the KG).

Pedagogical objectives

- Do a little software development, using Semantic Web programming frameworks
- Setup and interact with an RDF database
- Exploit multiple sources of heterogeneous data
- Present information online with rich metadata

Timeline

The project starts full time on Friday 19th December 2025 but some of the practical sessions already provided the building blocks for the project. You will be working on your project full time during the sessions of the 19th of December morning, 19th of December afternoon, 6th of January 2026 afternoon (after the exam), and 9th of January 2026 afternoon.

You must deliver all of your working files before the **15th of January 2026**. All files must be sent strictly before the end of the day, in Central European Time. **Every extra minute will be penalised**. You must additionally provide a written report explaining your choices, the functionalities, etc. before the 23rd of January, 2026 end of the day, Central European Time. Everything that comes after this deadline will be rejected as if nothing was delivered.

Delivery: Files must be committed to a code repository of your choice. You must send an email with the link to the repository and no attached document, with your team mate in CC of the email. The report documenting your work must be put on the repository. The knowledge graph that you produce in Turtle must also be put on the code repository.

Resources

Tolkien Gateway wiki

Tolkien Gateway is a wiki powered by MediaWiki, the wiki engine originally developed for Wikipedia. Because of that, the wiki's content can be accessed via the MediaWiki API. See the list of API calls available on the Tolkien Gateway. The main API calls to use in the project are `query` and `parse` (they are called *actions* in the documentation). With the `query` action, you can list pages and categories of the wiki. With the `parse` action, you can get the source of a given page or extract specific elements, such as links to other pages or images.

All MediaWiki-powered wikis use the same syntax to edit pages: wikitext. An introduction by example is available on MediaWiki. The most relevant feature of wikitext in the project is templating. A template is a page containing text that is meant to be reused at several places in the wiki. Invoking the template in some page has the effect of inserting the template text in that page. Templates may have parameters. See the MediaWiki documentation on templates for more details.

Many client libraries exist to ease the development of bots accessing a MediaWiki API. See the list maintained on MediaWiki. Most of these libraries ensure that bots are "well-behaved", which typically implies that they do not overload the server with many requests (throughput is limited) but also that they declare themselves as bots (by setting an appropriate `User-Agent` header in their HTTP requests). To avoid being banned by the wiki server, **we strongly recommend that you use a client library to access the Tolkien Gateway.**

Integrating external data

The collectible card game *Middle Earth: The Wizards* has cards that represent characters, items, or places of Middle Earth. Some of them are invented by the card maker company, but some cards correspond to actual enti-

ties of Tolkien's world. A data set of all METW cards is available in JSON. Use this data set to link wiki entities with the corresponding cards.

Additionally, you can a data set of Lord of the Rings characters in CSV to enrich your data.

Finally, you may use another wiki about Tolkien's legendarium to enrich your data further. In particular, this wiki is available in multiple languages (see the list and links at the bottom of the main page). You can add labels for the entities in your data in multiple languages.

Technical guidelines

Here are steps you can carry out to develop your application. You will not be evaluated on each of these steps but on the end result; you are free to plan development in a different way.

- Set up a triplestore (e.g. Fuseki) to later store data generated from Bulbapedia.
- To generate RDF from wikitext, start with an example. For instance, look up the page for Elrond. The source code of the page is in wikitext (using link "view source" at the top of the page). This wikitext should include a template invocation for the `infobox character` template: `{{infobox character|...}}`. Put the template in a file and parse it with a library. Then, generate an RDF graph encoding Elrond's infobox.
- Every character page has an infobox, whose template is documented on a dedicated page on the Tolkien Gateway. Write a procedure that generates RDF for any character infoboxes. Then, test it on a list of characters. You can use, for instance, the list from the wiki category Third Age characters that lists characters from *The Lord of the Rings*, among others. To iterate over character names, you can parse the source code of this page, as you did in the previous step, or you can call the `query` action via the MediaWiki API and find a way to list all members of the category.
- Repeat the process for as many types of infobox as possible. The list of infobox types is given in the category page about Infobox templates.
- Use the MediaWiki API to navigate the wiki. In particular, use the `query` action with parameter `list=allpages` to exhaustively list pages of the wiki (you might have to manage pagination with the `continue` parameter). For each page, build a triple stating that the page is about some entity that is distinct but has the same name. Get inspiration from DBpedia, which e.g. distinguishes `http://dbpedia.org/page/X` from `http://dbpedia.org/resource/X`, or YAGO.
- Once you have decided how to identify entities, you can start integrating external knowledge. Generate triples from the file containing METWCCG data (`cards.json`), such that the KG entities you have generated have links to the card descriptions. Make sure the generated triples have proper language tags.
- You can add labels in multiple languages using data from The Lord of the Rings wiki, or even Wikipedia, that contains a lot of information about Tolkien's work.
- For each page, use the `parse` action with parameters `page=<page title>` and `prop=wikitext` to retrieve its source code. You might also want to use other features of the action, including `prop=templates`, `prop=images` and `prop=links`. Then, if applicable, call the infobox transformation procedure you previously wrote.
- By then, you should have produced a significant amount of triples, possibly with inconsistencies. Specify a vocabulary that is aligned to `schema.org` to capture information from the Tolkien Gateway. Your vocabulary should consist of RDFS classes and properties. Get inspiration from YAGO, which also aligns with schema.org. See the design document of YAGO. To further specify classes and properties, translate in-

fobox templates into SHACL shapes that refer to your vocabulary. Each class should be the target of at least one shape.

- Wiki pages typically include lists of related items, either inside the wiki or elsewhere on the Web. Use the MediaWiki API to retrieve Tolkien Gateway links to Wikipedia (`parse` action with parameter `pop=externallinks`). Find possible alignments with DBpedia and YAGO by looking for resources in these KGs that link to the same Wikipedia pages as the entities you are creating. An alignment here is a triple stating that your entity is semantically the same as some DBpedia or YAGO resource.
- If you have already been using a triplestore to manage your KG, you already have a SPARQL interface to it. However, this interface does not perform any reasoning. Write a SPARQL query that returns all classes of an entity, including superclasses defined in schema.org or your own vocabulary. You may want to use property paths for that. Write another SPARQL query that returns all relations of an entity (incoming and outgoing links), taking `owl:sameAs` triples into account: if entity `X` is the same as entity `Y`, for every triple with `X` as subject or object in the KG, there is an equivalent triple with `Y` at the same position.
- As a final step, build a Linked Data interface based on triples stored in the triplestore and the queries you have defined in the previous step. For every entity in the KG, dereferencing its URI with a `GET` request should return a description of the entity. There is no strict definition of what a “description” is but it is common to return all direct relations of the entity. Develop a small Web server that serves Turtle or HTML, depending on the `Accept` header in the request. If the client requests HTML, build a page with a title, a short description, an illustration (if available) and a table with hyperlinks. Get inspiration from the interfaces of DBpedia and YAGO, which are very similar (see below).