Deadline: upload on 11th of January 2026, 16:00, on Moodle - Hard deadline

This phase will be graded from 0 to 10.

Attention: To receive the grade, you must present the project during the laboratory on 12th January, regardless of your semi-group. So all the groups will present ONLY during that laboratory.

Do not copy code from your colleagues or share your solutions with others. You will be asked to explain your code and justify your choices. If you cannot explain how your code works, you will receive a lower grade. In the readme you must at least write the name of the students that are part of the team and their contribution and add a link to the public GitHub repository.

Maximum 3 students can work on the assignment, only one uploads the .zip archive.

1. Write a RDF/XML for the following scenario (1 pt):
To be a recommended book for a user, a book should: Belong to the user's preferred theme, match the user's reading level.

The user Alice has a reading level of Intermediate and prefers Science Fiction. The user Bob has a reading level of Beginner and prefers Mystery.

The book "Dune" is a Science Fiction and Fantasy novel suitable for Advanced readers. The book "The Silent Patient" is a Mystery and Murder novel suitable for Intermediate readers. The book "Hunger Games" is a Science Fiction and Fantasy novel for Beginner readers only.

2. Add a feature to your web application that allows the user to upload a RDF/XML file and visualize its RDF graph. You may use Jung (http://jung.sourceforge.net/) or a similar API. Test the feature with the file created at point 1. (1pt)

3. In the web application, add a feature to let you modify or add a book. Test the features for the book "Harry Potter" (add book) and "Hunger Games" (change reading level). You must use RDF and JENA API / other RDF API in order to write, read, query and perform operations (https://jena.apache.org/tutorials/). (0.5pt)

4. In the web application, list all the available books and provide a dedicated page for each book. You have to use RDF and JENA API/ other RDF API in order to write, read, query and perform operations (https://jena.apache.org/tutorials/) (0.5pt for listing, 0.5pt for book info page).

5. Create an OWL ontology for the book recommendation system presented in the first exercise. (1.5pt)

Visualize it with https://graphdb.ontotext.com and put the ontology graph exported from there into a figure or use and add-on for that in Protégé.

You can use Protégé for this exercise. Include in your zip file the screenshots taken.

6. Make 5 SPARQL queries for your ontology and save them in a txt file with the name "sparql_owl". Executed in Protégé or Graphdb and put a print screen with the results. (1 pt).

7. Build a chatbot that allows the user to perform the following operations:

1) The chat window should float over all the pages of the website. **(0.5 pt)**
2) The chat window should provide 3 conversation starters that should be context-aware (if I am viewing a Harry Potter book, the conversation starters should be about that Harry Potter book, its author or its genre | if for example I am on the books list, I could have as conversation starter 'What is a book that I am most likely to enjoy from this list?') **(1 pt)**
3) The responses of the LLM should be enhanced with information from the books that are present in the XML as well as user data (preferred books / genre, etc). **THIS SHOULD NOT BE DONE BY PROVIDING THE ENTIRE XML TO THE LLM IN EACH REQUEST YOU SHOULD BASE THIS ON YOUR VECTOR DATABASE. (1.25 pt)**
4) The chat should allow me to find a book by a theme and author. For example if I type "What book has the author Frank Herbert and the theme Science Fiction" the chat should respond with "Dune" (it could also respond with more books, depending on what is in your database). **(1.25 pt)**

Acceptance criteria:

1. You should build a vector database from the RDF/XML or XML from HW1, and the operations above should be performed
2. You should use an LLM (probably Llama 3 or Llama 4 on Ollama - but if you do not have enough RAM you can use a model that is hosted in the cloud (Check Google AI Studio / OpenAI API or OpenRouter.ai)
3. The information in the chat responses should be based on the data from your vector database rather than model knowledge (e.g. If in your database "Harry Potter" was written by "Gigel", the LLM should respond with "Gigel" to the question "Who wrote Harry Potter", rather than "J. K. Rowling") - basically you should build a RAG.