COMP 6741 - Intelligent Systems

Project Report

ROBOPROF

Instructor
Dr. René Witte

Group Name: AK_G_6

Student Name	ID	Specialization
Nilesh Suryawanshi	40229633	Knowledge Graph Engineer
Vidhi Sagathiya	40232374	Data Specialist

We certify that this submission is the original work of members of the group and meets the Faculty's Expectations of Originality.

Date: 15th April, 2024

GitHub Link: Roboprof

Contents

1 Vocabulary

Our vocabulary comprises multiple classes and properties, extending or reusing existing vocabularies wherever feasible and logically sensible. We rely on several external vocabularies and ontologies to enhance the expressiveness and interoperability of our schema:

1.1 External Vocabularies Used

- RDF (`http://www.w3.org/1999/02/22-rdf-syntax-ns#`)
- RDFS (`http://www.w3.org/2000/01/rdf-schema#`)
- FOAF (`<http://xmlns.com/foaf/spec/`)
- TEACH (`http://linkedscience.org/teach/ns#`)
- DBpedia Resource (`http://dbpedia.org/resource/`)
- DBpedia Ontology (`http://dbpedia.org/ontology/`)
- DBpedia Property (`http://dbpedia.org/property/`)

The classes of our schema and their respective links to common vocabularies are summarized in Table 1 below. This schema aims to leverage the flexibility of defining custom properties and classes while establishing connections with mainstream vocabularies for broader semantic integration.

Class	**SubClassOf**
University	foaf:Organization
Course	
Lecture	
LectureContent	
Slide	LectureContent
Worksheet	LectureContent
Topic	
Student	foaf:Person

Table 1: Knowledge-Base Classes

1.2 Schema Justification

- **Hierarchical Structure:** We employ abstract classes such as `cu:LectureContent` to structure related entities and define properties with appropriate ranges, facilitating a more organized representation of course materials. Additionally, subclasses such as cu:Worksheet and `cu:Slide` further refine the hierarchy, allowing for specific categorization of lecture content.
- **Extension of Existing Vocabulary:** While we primarily utilize existing vocabularies like RDF and RDFS, we introduce custom properties such as `cu:hasCourseOutline` to address specific requirements within our domain, ensuring semantic clarity and compatibility with our schema.

1.3 Schema Properties

While we prioritized the use of existing properties, we occasionally opted to create bespoke properties to ensure clarity and specificity in our schema. For instance, we differentiated between a course's number and a lecture number to accurately represent sequential ordering. Table 2 below outlines all the properties defined in our schema along with their respective domains and ranges.

Property	**Domain**	**Range**
hasCourseSubject	cu:Course	rdfs:Literal
hasCourseNumber	cu:Course	rdfs:Literal
hasCourseDescription	cu:Course	rdfs:Literal
hasCourseOutline	cu:Course	rdfs:Literal
hasCredits	cu:Course	rdfs:Literal
hasLecture	cu:Course	rdfs:Literal
hasLectureNumber	cu:Lecture	rdfs:Literal
hasLectureContent	cu:Lecture	cu:LectureContent
topicsCovered	cu:Lecture	cu:Topic
hasTopic	cu:Topic	rdfs:Literal
hasProvenance	cu:Topic	rdfs:Literal
hasMaterials	cu:Topic	cu:LectureContent
hasStudentID	cu:Student	rdfs:Literal
hasStudentEmail	cu:Student	rdfs:Literal
isEnrolledIn	cu:Student	cu:Course

Table 2: Property Domain and Range

Our schema is designed to balance flexibility with specificity, ensuring a comprehensive representation of university courses and student-related information within the knowledge base.

2 Knowledge Base Construction

2.1 Dataset

To construct our knowledge base, we utilized two primary datasets:

- `CATALOG_2023_09_19` and `CU_SR_OPEN_DATA_CATALOG_UTF8 CSV files` - obtained from the Concordia University's open data portal. These datasets contain information about Concordia's courses, including course numbers, codes, names, descriptions, and other relevant details.
- **Course Materials** - For our two special courses, COMP 6231 and COMP 6741, Course materials such as slides, worksheets, and readings were sourced directly from the 2024 course Moodle page.
- **vocabulary.ttl** - Created turtle file containing the vocabulary schema for the knowledge base, defining classes and properties used in the construction process.

2.2 Process and Developed Tools for Populating the Knowledge Base from the Dataset

- **Data Cleaning:** We wrote a Python script to clean the CSV files by removing entries with empty records. We merged them by adding a class unit column from the `CU_SR_OPEN_DATA _CATALOG_UTF8` file to the `CATALOG_2023_09_19` file, enabling us to use one consolidated dataset. Additionally, we converted `CU_SR_OPEN_DATA_CATALOG.csv` to UTF-8 using *chatGPT-4* , as it was initially in UTF-16 format which was not recognized by Python.
- **Dataset Limitations:** The dataset provided limited information, particularly in terms of accurate descriptions and web links. Many descriptions were missing or inaccurate, and the web link column was sparsely populated. We addressed this limitation by conditionally adding relevant triples only if the web link column was populated for a particular row.
- **Course Materials:** We manually sourced accurate descriptions, course topics, and content pieces to fill the gaps in the dataset. Since these materials are only accessible to students enrolled in the course, we provided local URIs for the resources and example URIs for some web-based resources in the knowledge graph.

2.3 How to Run the Tools to Create the Knowledge Base

- **Step 1:** Unzip the submitted file containing the project materials.
- **Step 2:** Ensure *Python* is installed on your system and Open a terminal or command prompt.
- **Step 3:** Navigate to the Roboprof directory to ensure that local URIs can resolve correctly.
- **Step 4:** Run the Python script `mergeCSV.py` to regenerate the `CLEANED_DATA` file, ensuring that the dataset is up to date and properly merged.
- **Step 5:** Execute the `main.py` file to generate the knowledge base. This process will produce two output files: `kb.ttl`, which contains the knowledge base in Turtle format, and `kb_ntriples.rdf`, which contains the knowledge base in N-Triples format. These files will be generated anew with each execution of the script, overwriting previously generated versions.

3 Triplestore and SPARQL Endpoint Setup

3.1 Selection of Apache Fuseki Server

We utilized Apache Fuseki Server as a central component for managing and querying RDF data. Apache Fuseki Server was chosen as the Triplestore solution for the RoboProf project due to several key reasons:

- **Simplicity:** Apache Fuseki Server offers a straightforward and user-friendly interface for managing RDF data and querying via SPARQL. Its intuitive design makes it accessible even for users with minimal experience in Semantic Web technologies.
- **SPARQL Support:** Fuseki provides robust support for the SPARQL query language, allowing for efficient querying and retrieval of RDF data stored within the Triplestore. This was crucial for the project's requirement of performing complex queries to extract insights from educational data.

- **Community Support:** Apache Fuseki is backed by a vibrant open-source community, providing access to extensive documentation, tutorials, and community forums. This ensures ongoing support and assistance in case of any issues or challenges encountered during setup and usage.

3.2 Installation and Configuration

This section outlines the step-by-step process we followed to install and configure Apache Fuseki Server to suit our project requirements:

1. **Download and Extraction:**

   We downloaded the latest distribution package of Apache *Fuseki Server* from the official Apache website or from a trusted mirror. Upon downloading, we used common archive utilities like *'tar'* to extract the package. After extraction, we obtained a directory containing all the necessary files and directories.

2. **Installation:**

   We navigated to the directory containing the *Fuseki* server script (*typically named 'fuseki-server'*) and made it executable using the `'chmod'` command in the terminal. This step ensured that the server script could be executed to start the *Fuseki* server instance..

3. **Accessing Web Interface:**

   To access Fuseki's web interface, we opened a web browser and navigated to the `URL` where *Fuseki* was running. By default, Fuseki runs on port `3030`, so the URL was typically `http://localhost:3030/`. Upon accessing the web interface, we were able to interact with the *Fuseki* server.

4. **Creating Dataset and Populating Data:**

   Within Fuseki's web interface, we created new datasets by specifying the dataset name. Once the dataset was created, we uploaded the `'kb.ttl'` file in Turtle format to populate the dataset with semantic data. After uploading the data, we navigated to the query tab to execute SPARQL queries against the dataset and retrieve relevant results.

3.3 Challenges Encountered and Addressed

1. **Compatibility Issues:** Compatibility issues arose when attempting to run Apache Fuseki Server on certain operating systems and Java Runtime Environments (JREs). This was primarily due to conflicting dependencies and runtime environments.

   Solution: To address compatibility issues, thorough research was conducted to identify the optimal combination of operating system and JRE version supported by Apache Fuseki. Additionally, compatibility testing was performed on multiple environments to ensure seamless deployment.

2. **Security Configuration:** Configuring secure access to the SPARQL endpoint posed challenges, particularly in implementing robust authentication mechanisms and access control policies to safeguard sensitive data.

   Solution: To enhance security, Apache Fuseki's built-in authentication features, such as HTTP basic authentication and OAuth2.0, were leveraged to authenticate users accessing the SPARQL endpoint. Access control policies were established to restrict unauthorized access to the Triplestore.

4 Knowledge Base Population

The Knowledge Base (KB) processing involves several stages to systematically construct, refine, and maintain the knowledge base for a comprehensive and accurate representation of course content. The following steps outline the KB processing workflow:

1. **Preprocessing Course Materials**:
   - The Python code (main.py) scans through each course directory to locate PDF files containing materials such as slides, worksheets, labs, and readings.
   - PDF files are converted into text files using Apache Tika while preserving the directory structure (COMP6231_TXT, COMP6741_TXT).
   - Resulting text files are stored in separate directories for each course, ensuring easy access for analysis.
2. **Topic Triples Creation Process**:
   - The code iterates through each course material directory, including slides, worksheets, and labs.
   - Text content from each file is extracted and processed.
   - Named entities are identified using spaCy's named entity recognition (NER).
   - Entities with confidence scores above a threshold (e.g., 0.75) are considered topics.
3. **Entity Linking to DBpedia**:
   - Entities with confidence scores above a threshold are considered for linking.
   - DBpedia Spotlight is employed to link entities to corresponding DBpedia resources.
   - Resulting triples include identified entities, their DBpedia links, and the course materials they are extracted from.
4. **Filtering of Linked Entities**:
   - **Relevance Assessment**: Entities are evaluated for relevance to course topics and objectives, filtering out non-relevant entities.
   - **Confidence Score Thresholding**: Only entities with high confidence scores are retained to ensure quality links to DBpedia.
   - **Domain-Specific Filtering**: Course-specific criteria are applied to retain only relevant entities. For example, in "Construction and Analysis of Distributed Processes," only distributed computing-related entities are retained.
   - **Redundancy Removal**: Duplicate or redundant entities are filtered out to enhance clarity and streamline the knowledge graph. Entities are evaluated within the course context to ensure accurate representation of covered topics, maintaining coherence in the knowledge graph.

4.1 Population of Triples into Knowledge Base (KB)

The generated triples are then added to the RDF graph using RDFLib corresponding to the appropriate course within the knowledge base. For courses COMP6231 and COMP6741, RDF graphs are populated with relevant triples based on the extracted topics from their associated PDFs. Once generated, they are seamlessly integrated into the existing knowledge base hosted on Fuseki. These statistics offer insights into the scale and content coverage of the knowledge base, facilitating analysis and evaluation of its comprehensiveness.

Metric	Value
Total Number of Triples	28,513
Number of Distinct Topics	1,564
Number of Topic Instances COMP6741	882
Number of Topic Instances COMP6721	672

Table 3: Triple Population Statistics

5 Chatbot Design

5.1 Utilizing *Rasa* for the Natural Language Conversation Interface

In developing our conversational interface, we opted for **Rasa**, a premier open-source conversational AI framework. *Rasa* empowers the creation of advanced assistants adept at understanding natural language and providing contextually relevant responses. Our implementation focuses on configuring intents, entities, and dialogue management to ensure an intuitive and effective experience for users interacting with our educational chatbot. Rasa setup and configuration involve several steps to establish the conversational AI framework.

1. **Installation**: Rasa is installed by executing the command: `pip install rasa`.
2. **Project Initialization**: A dedicated directory for the Rasa project is created and initialized using the command `rasa init`.
3. **Training Data Preparation**: Training data, including custom stories, NLU data, and response templates, is prepared and organized into specific files (`data/nlu.yml`, `data/stories.yml`, and `domain.yml`).
4. **Model Training**: The Rasa model is trained using the command `rasa train`, which trains both the NLU and dialogue management models.
5. **Testing the Assistant**: Functionality and efficacy of the assistant are validated through an interactive session initiated with the command `rasa shell`, enabling real-time interaction for testing and refinement of responses.

5.2 Intents

The following table lists the intents defined to handle various user queries and requests in the chatbot:

Intent	Description
greet	Represents various ways users may greet the chatbot.
goodbye	Indicates the user is ending the conversation.
affirm	Denotes agreement or confirmation from the user.
deny	Signifies disagreement or negation from the user.
mood_great	Indicates the user is feeling positive or great.
mood_unhappy	Indicates the user is feeling negative or unhappy.
bot_challenge	User questions whether they are interacting with a human or a bot.
about_course	User queries information about a specific course.
about_courses_offered_by_university	User asks about courses offered by a particular university.
about_course_lecture_topics	User seeks information about topics covered in a specific course lecture.
about_course_uni_topic	User inquires about courses at a university related to a specific topic.
about_courses_within_subject	User requests a list of courses within a specific subject.
about_credits	User asks about the credit value of a course.
about_materials	User seeks recommended materials for a specific topic in a course.
about_additional_resources	User inquires about additional resources available for a course.
about_detail_content	User requests detailed content information for a course lecture.
about_reading_material	User asks about recommended reading materials for a course.
about_competencies_of_student	User wants to know what competencies a student gains from completing a course.
about_student_transcript	User requests a transcript listing courses taken with grades for a specific student.
about_student_courses_completed	User asks for a list of students who have completed a specific course.
about_student_grades	User seeks grades achieved by a specific student in a course.
about_topics_covered	User inquires about topics covered in a specific course event.

Table 4: Intents

These intents are designed to facilitate a comprehensive understanding and management of course-related inquiries from users interacting with the chatbot.

5.3 Stories Defined for Conversation Flows

Stories	**Example Conversation**
get_course_info	User: "What is course COMP 6741 about?" Bot: Action - Retrieve course information.
get_course_lecture_topics	User: "Which topics are covered in COMP 6741 lecture 3?" Bot: Action - Retrieve topics covered in the specified lecture of the course.
get_course_uni_topic	User: "Which courses at Concordia University teach Computer science?" Bot: Action - Retrieve courses at Concordia University that cover the specified topic.
fetch_all_courses	User: "What courses are offered by Concordia University?" Bot: Action - Fetch and display a list of all courses offered by Concordia University.
get_course_topics	User: "Which courses cover google?" Bot: Action - Retrieve courses covering the specified topic.
get_courses_within_subject	User: "List all courses offered within COMP." Bot: Action - Retrieve and list all courses offered within the specified subject.
get_credits	User: "How many credits is COMP 6741 worth?" Bot: Action - Retrieve the number of credits for the specified course.
get_materials	User: "What materials are recommended for convolution in COMP 6741?" Bot: Action - Retrieve recommended materials for the specified topic in the course.
get_additional_resources	User: "What additional resources are available for EDUC 301?" Bot: Action - Retrieve additional resources available for the specified course.
get_detail_content	User: "Detail the content available for LEC 3 in COMP 6231." Bot: Action - Retrieve detailed content available for the specified lecture in the course.
get_reading_material	User: "What materials are recommended for convolution in COMP 6741?" Bot: Action - Retrieve recommended reading materials for the specified topics.
get_competencies_of_student	User: "What competencies does a student gain after completing COMP 6231?" Bot: Action - Retrieve competencies gained by a student.
get_student_transcript	User: "Print a transcript for 40229633." Bot: Action - Generate and print a transcript for the specified student.
get_student_courses_completed	User: "Which students have completed COMP 6741?" Bot: Action - Retrieve a list of students who have completed the specified course.
get_student_grades	User: "What grades did 40229633 achieve in COMP 6741?" Bot: Action - Retrieve grades achieved by the specified student in the specified course.
get_topics_covered	User: "Which topics are covered in lec 5 of COMP 6741?" Bot: Action - Retrieve topics covered in the specified course event.

Table 5: Stories Defined for Conversation Flows

5.4 Entities for Chatbot

Table 6: Entities and Descriptions

Entity	**Description**
course	Represents a specific course, such as COMP 6741.
lecture	Represents a lecture within a course, identified by a lecture number or name.
topic	Represents a topic covered in a course or lecture.
university	Represents a university where courses are offered.
department	Represents a department within a university.
courseEvent	Represents an event associated with a course, such as a lecture or seminar.
subject	Represents a subject area or field of study, such as Computer Science or Mathematics.
student	Represents a student enrolled in courses.

5.5 Actions for Chatbot

Action	Description
action_course_info	Retrieves information about a specific course.
action_about_reading_material	Retrieves recommended reading materials for a course topic.
action_about_competencies_of_student	Retrieves competencies gained by a student after completing a course.
action_courses_within_subject	Retrieves a list of courses within a specific subject area.
action_about_student_grades	Retrieves grades achieved by a student in a specific course.
action_about_student_courses_completed	Retrieves a list of students who have completed a specific course.
action_topics_covered	Retrieves topics covered in a specific course event.
action_course_lecture_topics	Retrieves topics covered in a specific lecture of a course.
action_course_uni_topic	Retrieves courses at a university related to a specific topic.
action_about_detail_content	Retrieves detailed content information for a course lecture.
action_about_additional_resources	Retrieves additional resources available for a course.
action_courses_offered_by_university	Retrieves courses offered by a particular university.
action_about_credits	Retrieves the credit value of a course.
action_about_materials	Retrieves recommended materials for a course topic.
action_about_student_transcript	Generates and prints a transcript for a specific student listing courses taken with grades.

Table 7: Actions and Descriptions

5.6 Integration Process with Fuseki Server and SPARQL Queries

1. **Setup and Configuration:**

   Our chatbot seamlessly integrates with a locally hosted *Fuseki* server, accessible at `http://localhost:3030`. Employing HTTP requests, specifically `requests.post`, the chatbot efficiently communicates with *Fuseki* to execute SPARQL queries and retrieve pertinent information.

2. **SPARQL Query Definition:**

   Each action within our chatbot corresponds to a meticulously crafted SPARQL query, intricately defined within our Python code as multi-line strings. To streamline URI representation and enhance readability, RDF prefixes (`PREFIX`) are judiciously employed. Python's string formatting capabilities, such as `%s` for string variables and `%d` for integers, facilitate dynamic interpolation of variables within our queries.

3. **User Input Management:**

   Throughout the conversation, user inputs, including course names and topics of interest, are methodically captured as slots (`tracker.slots`). These slots serve as essential parameters in our SPARQL queries, ensuring personalized and contextually relevant responses to user queries.

4. **SPARQL Query Execution:**

Our chatbot adeptly sends HTTP POST requests to the designated Fuseki SPARQL endpoint (`http://localhost:3030/play/sparql`), meticulously transmitting the formulated SPARQL query as form data. Upon receiving a response, our system proficiently parses the JSON format (`response.json()`) to extract and process the retrieved query results.

5. **Error Handling:**

   In the event of exceptions, like `requests.exceptions.RequestException`, diligent error management strategies, including the implementation of try-except blocks, ensure robust handling of encountered errors. Any detected anomalies or exceptions are promptly relayed back to the user interface through `dispatcher.utter_message`, maintaining a seamless user experience.

5.7 Addressing Out-of-Vocabulary (OOV) Words

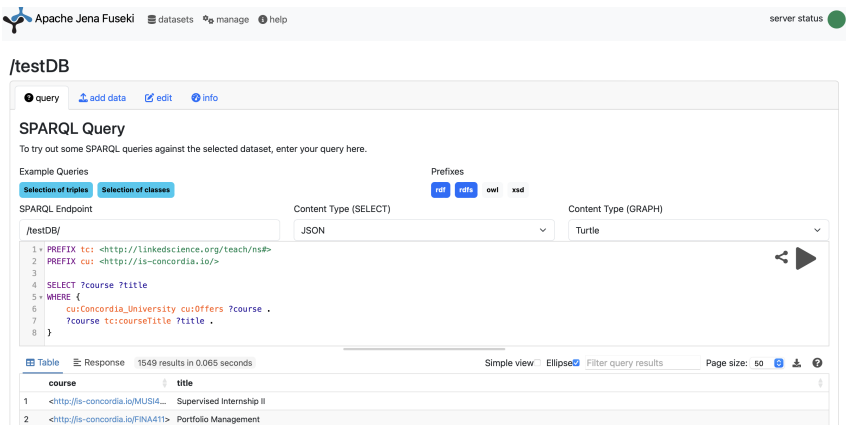
Our chatbot employs sophisticated mechanisms to handle out-of-vocabulary words effectively. Leveraging meticulously trained data, our system adeptly captures user intents and entities. In instances where inputs cannot be mapped to predefined intents or entities, our chatbot gracefully prompts users for clarification or delivers a tailored default response.

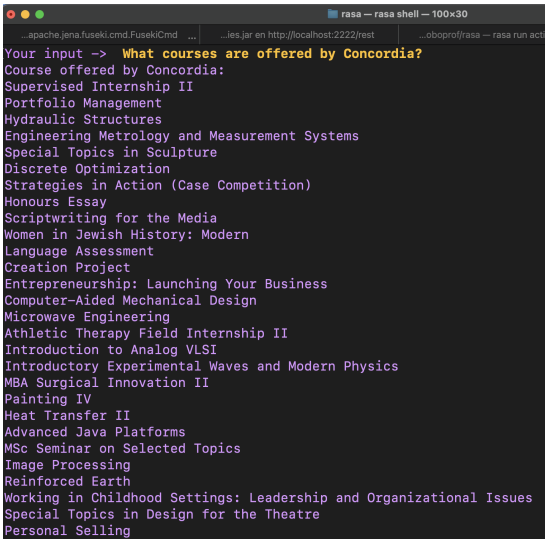
5.8 Crafting Natural Language Responses

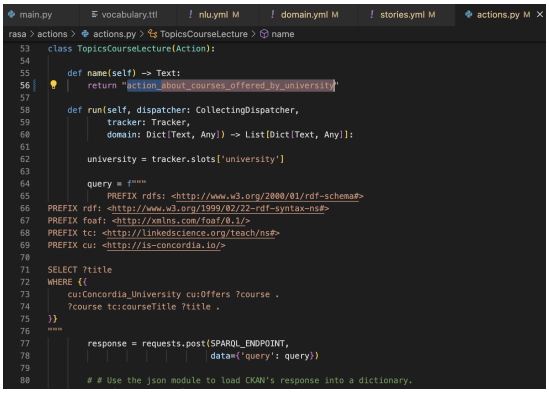
To foster a user-centric and engaging interaction experience, our chatbot meticulously generates responses in natural language. Utilizing response templates, our system expertly formats and presents query results, including comprehensive course details, covered topics, and recommended materials. Leveraging dynamic string formatting (`f"..."`), our chatbot seamlessly composes response messages, enhancing fluidity and relevance in interaction.

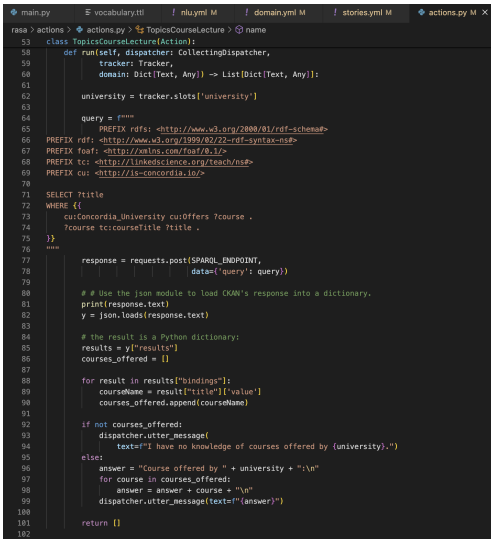
5.9 Output Samples

- **Question 1**: List all courses offered by `[university]`

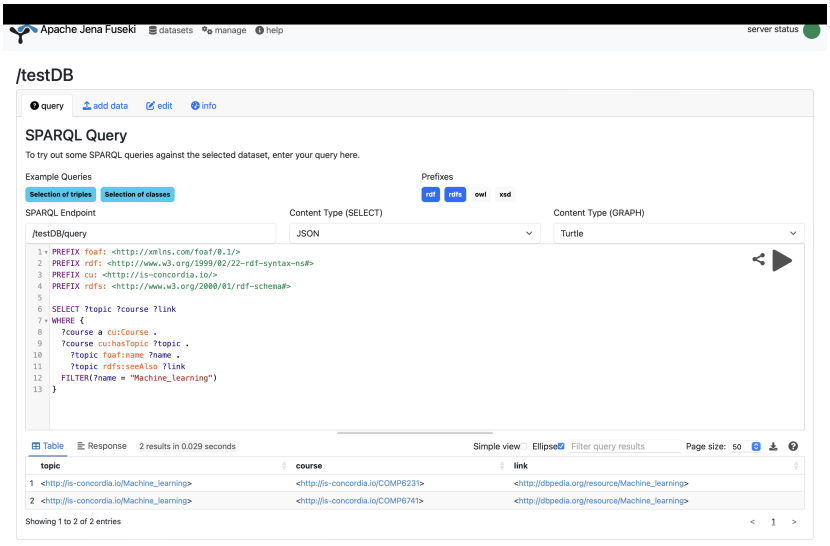




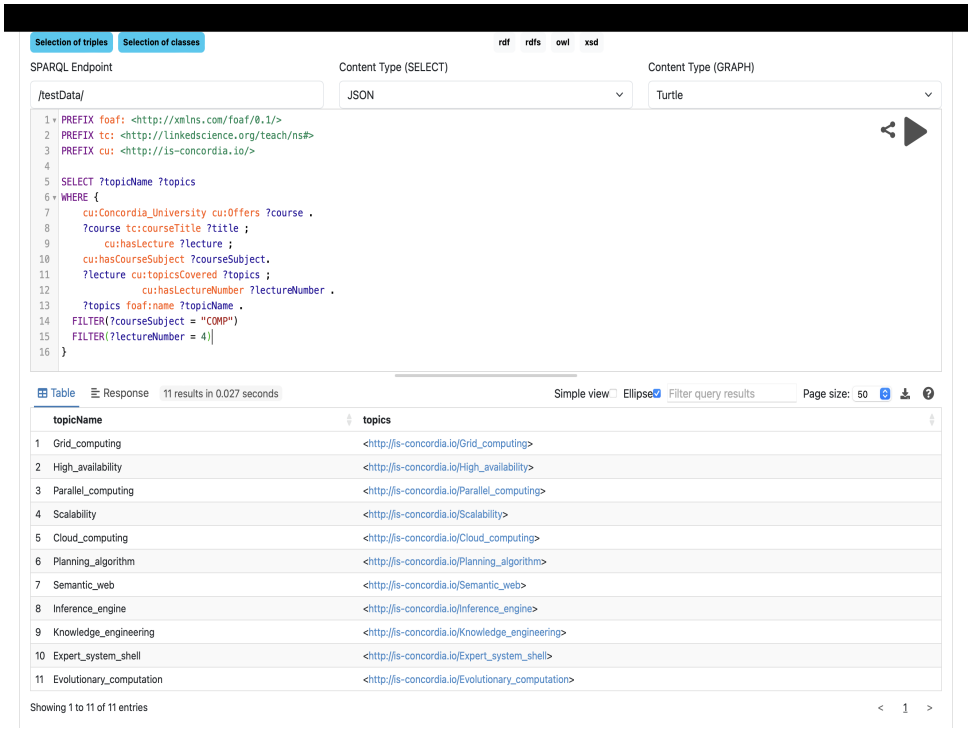


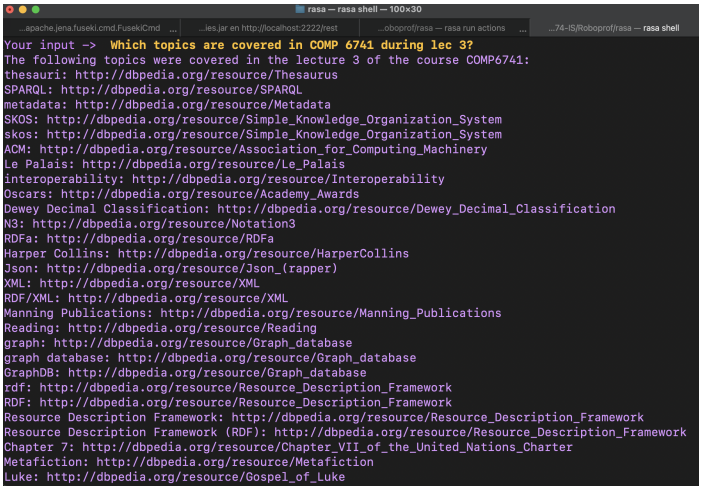


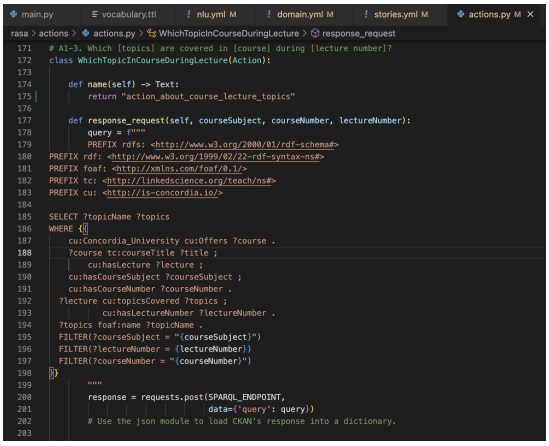
- **Question 2**: In which courses is `[topic]` discussed?

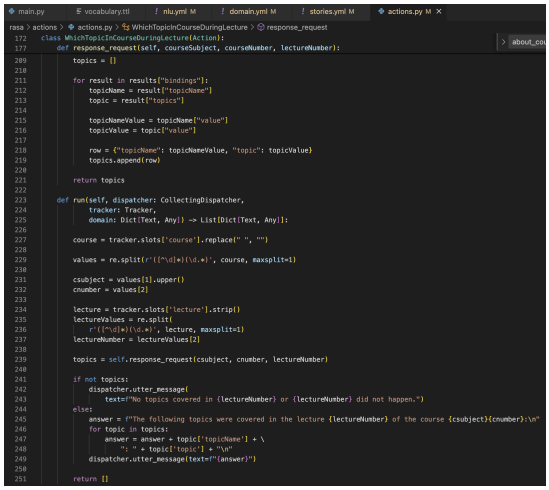


- **Question 3**: Which `[topics]` are covered in `[course]` during `[lecture number]`?

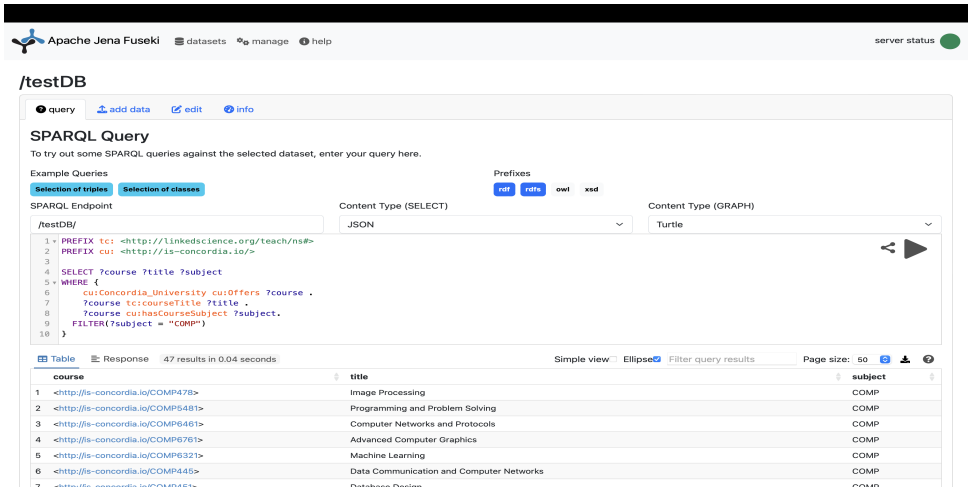


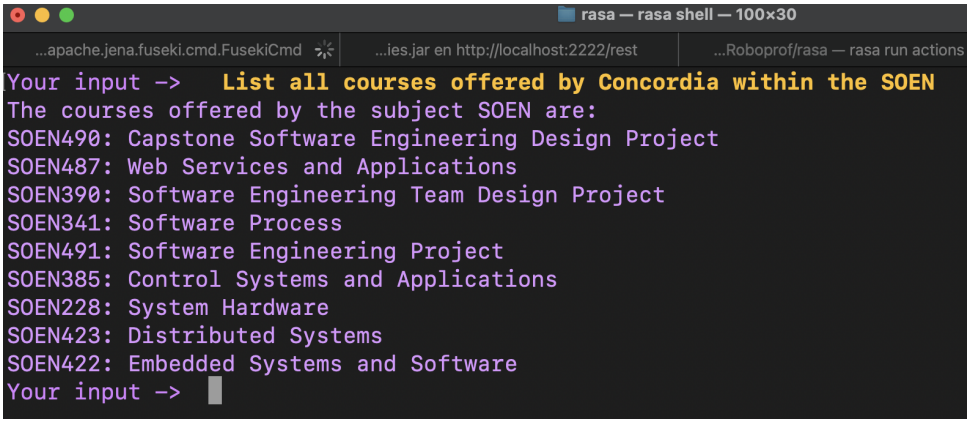




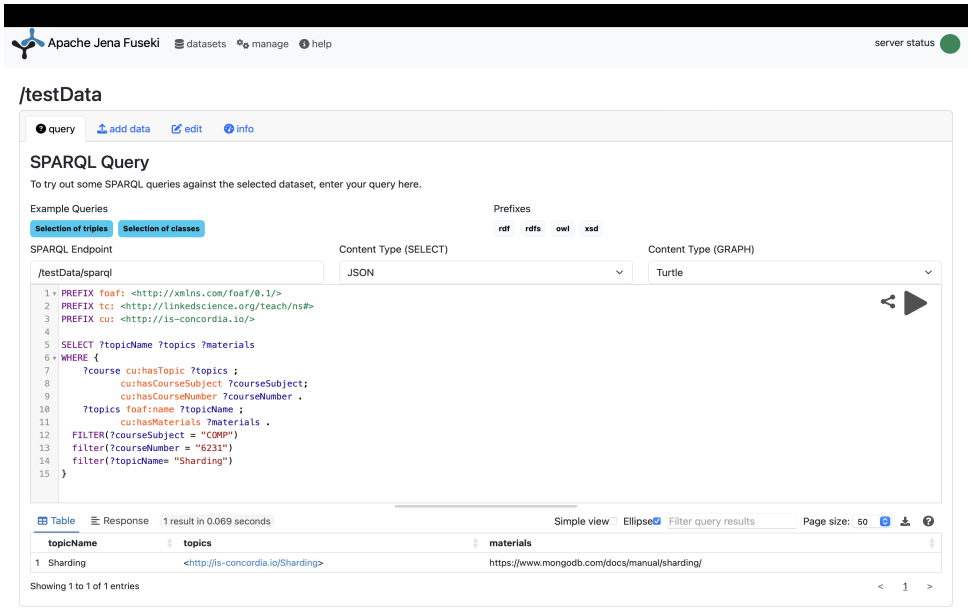


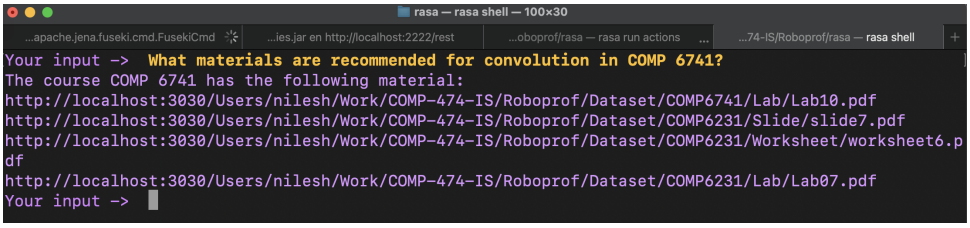
- **Question 4**: List all `[courses]` offered by `[university]` within the `[subject]` (e.g., "COMP", "SOEN").



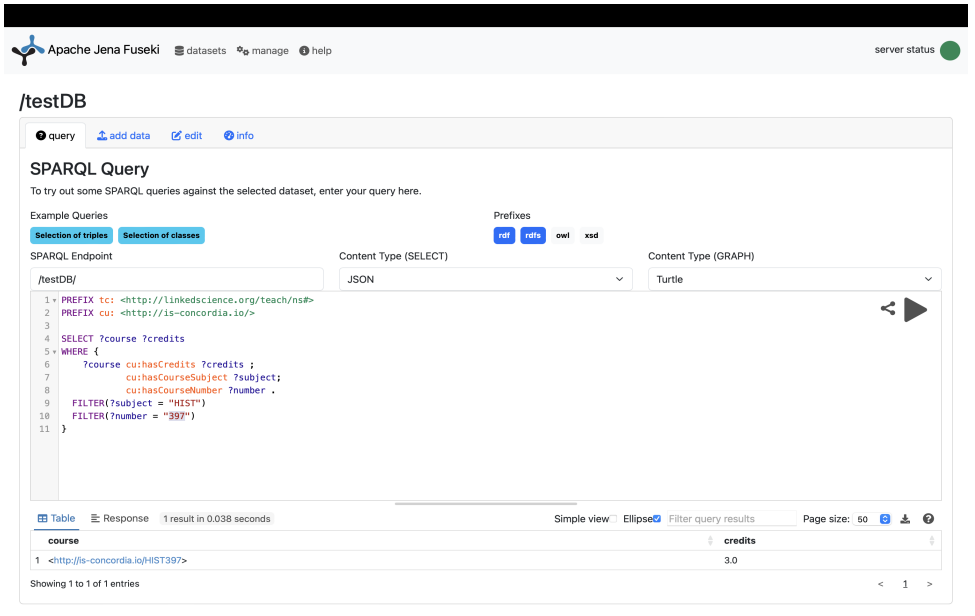


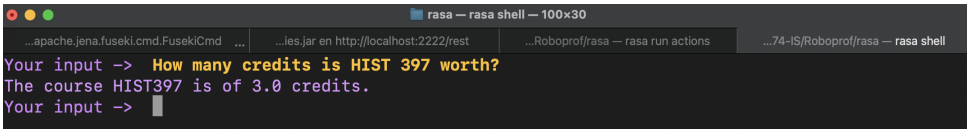
- **Question 5**: What `[materials]` (slides, readings) are recommended for `[topic]` in `[course] [number]`?

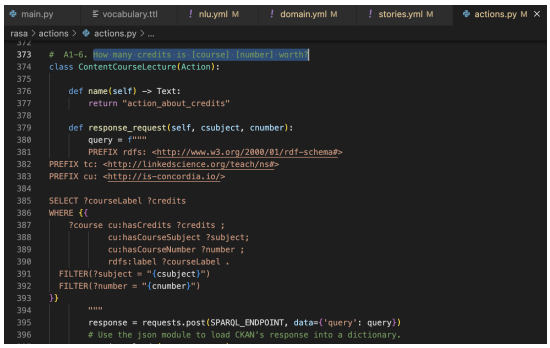


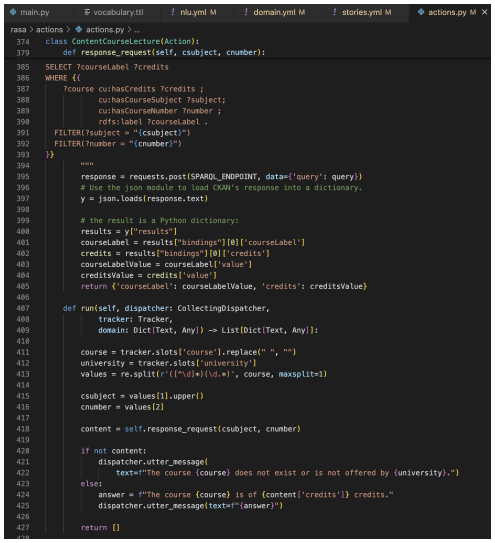


- **Question 6**: How many credits is `[course] [number]` worth?

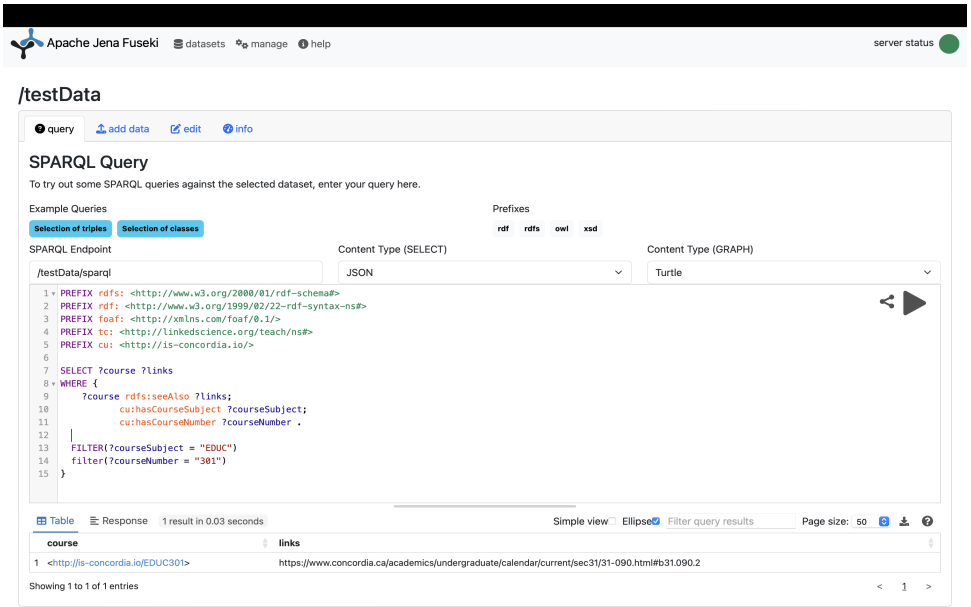


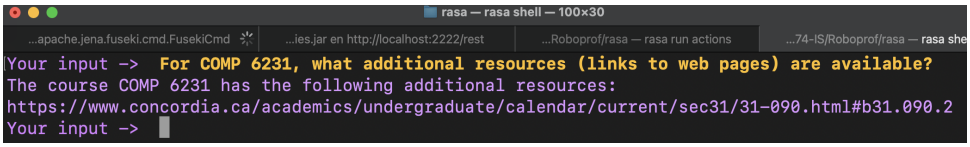


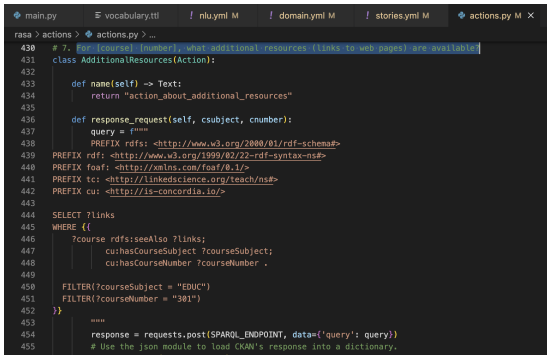


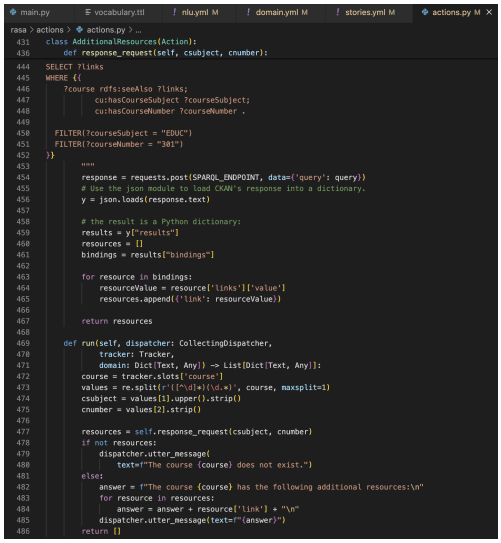


- **Question 7**: For `[course] [number]`, what additional resources (links to web pages) are available?

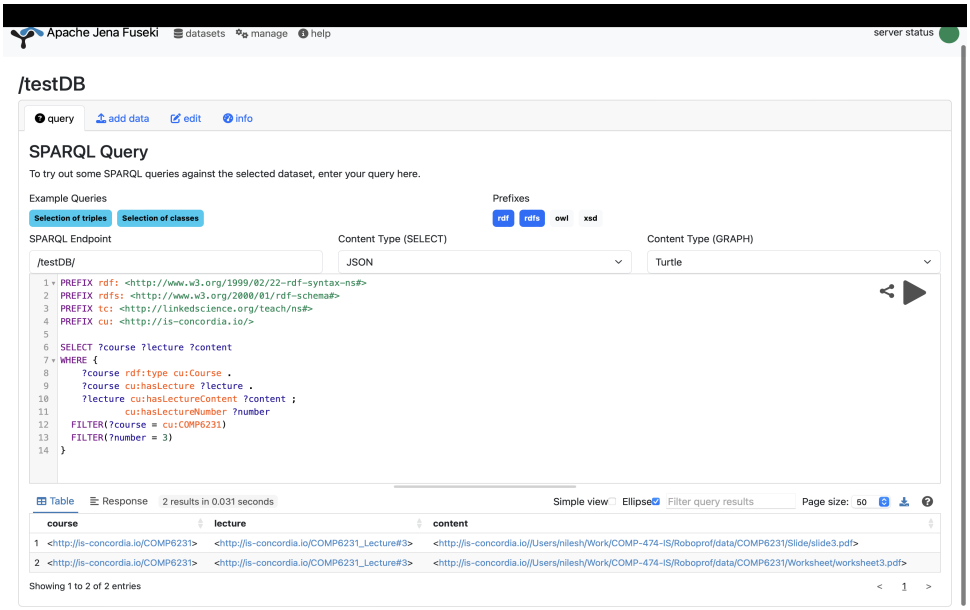


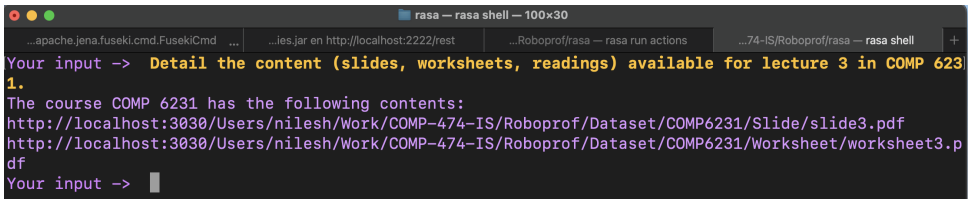


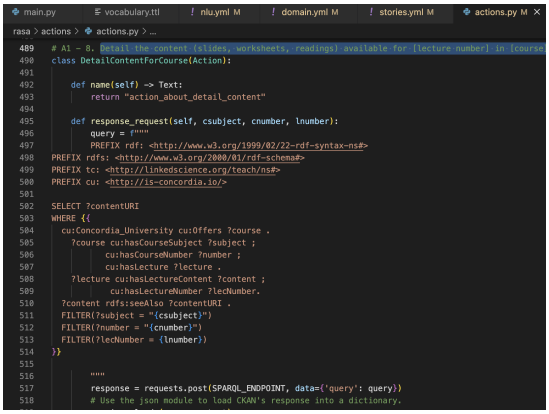




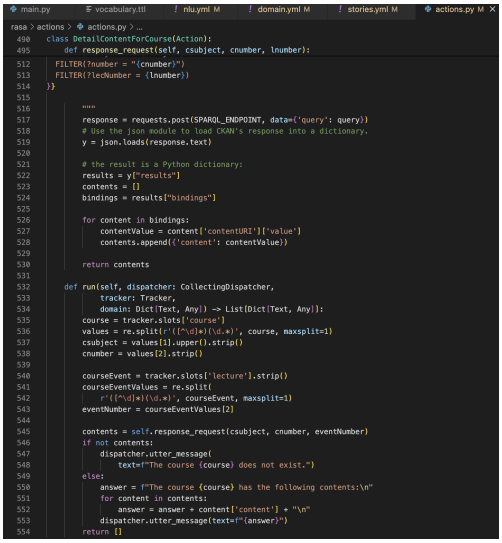
- **Question 8**: Detail the content (slides, worksheets, readings) available for `[lecture number]` in `[course] [number]`.





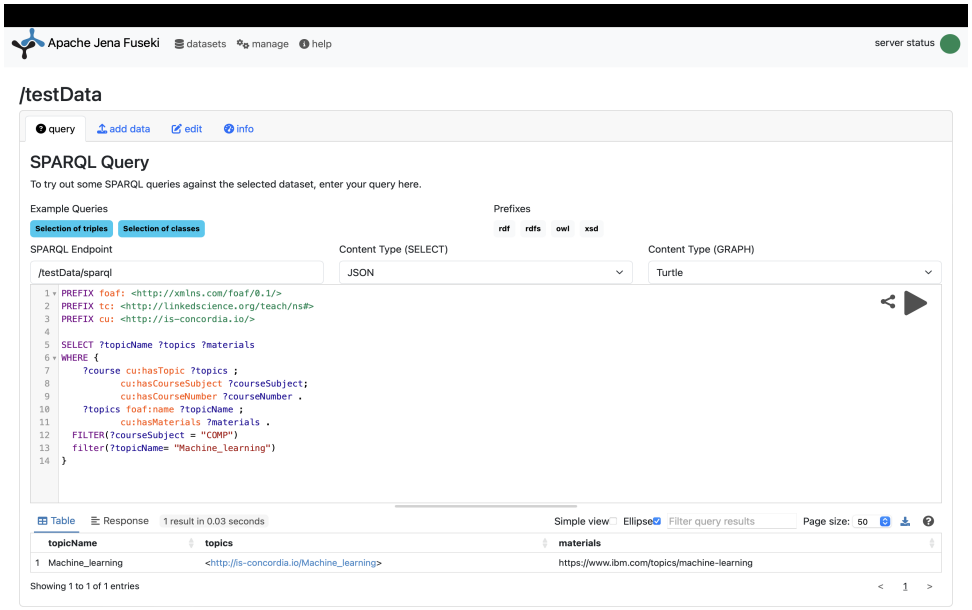


```
main.py  vocabulary.ttl  nlu.yml M  domain.yml M  stories.yml M  actions.py M
rasa > actions > actions.py > ...
489  # A1 - 8. Detail the content (slides, worksheets, readings) available for [lecture number] in [course
490  class DetailContentForCourse(Action):
491
492      def name(self) -> Text:
493          return "action_about_detail_content"
494
495      def response_request(self, csubject, cnumber, lnumber):
496          query = f"""
497          PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
498  PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
499  PREFIX tc: <http://linkedscience.org/teach/ns#>
500  PREFIX cu: <http://is-concordia.io/>
501
502  SELECT ?contentURI
503  WHERE {{
504    cu:Concordia_University cu:Offers ?course .
505      ?course cu:hasCourseSubject ?subject ;
506              cu:hasCourseNumber ?number ;
507              cu:hasLecture ?lecture .
508    ?lecture cu:hasLectureContent ?content ;
509             cu:hasLectureNumber ?lecNumber.
510    ?content rdfs:seeAlso ?contentURI .
511    FILTER(?subject = "{csubject}")
512    FILTER(?number = "{cnumber}")
513    FILTER(?lecNumber = {lnumber})
514  }}
515
516          """
517          response = requests.post(SPARQL_ENDPOINT, data={'query': query})
518          # Use the json module to load CKAN's response into a dictionary.
```



```
main.py  vocabulary.ttl  nlu.yml M  domain.yml M  stories.yml M  actions.py M
rasa > actions > actions.py > ...
490  class DetailContentForCourse(Action):
495      def response_request(self, csubject, cnumber, lnumber):
512    FILTER(?number = "{cnumber}")
513    FILTER(?lecNumber = {lnumber})
514  }}
515
516          """
517          response = requests.post(SPARQL_ENDPOINT, data={'query': query})
518          # Use the json module to load CKAN's response into a dictionary.
519          y = json.loads(response.text)
520
521          # the result is a Python dictionary:
522          results = y["results"]
523          contents = []
524          bindings = results["bindings"]
525
526          for content in bindings:
527              contentValue = content['contentURI']['value']
528              contents.append({'content': contentValue})
529
530          return contents
531
532      def run(self, dispatcher: CollectingDispatcher,
533              tracker: Tracker,
534              domain: Dict[Text, Any]) -> List[Dict[Text, Any]]:
535          course = tracker.slots['course']
536          values = re.split(r'([^\d]*)(\d.*)', course, maxsplit=1)
537          csubject = values[1].upper().strip()
538          cnumber = values[2].strip()
539
540          courseEvent = tracker.slots['lecture'].strip()
541          courseEventValues = re.split(
542              r'([^\d]*)(\d.*)', courseEvent, maxsplit=1)
543          eventNumber = courseEventValues[2]
544
545          contents = self.response_request(csubject, cnumber, eventNumber)
546          if not contents:
547              dispatcher.utter_message(
548                  text=f"The course {course} does not exist.")
549          else:
550              answer = f"The course {course} has the following contents:\n"
551              for content in contents:
552                  answer = answer + content['content'] + "\n"
553              dispatcher.utter_message(text=f"{answer}")
554          return []
```

- **Question 9**: What reading materials are recommended for studying `[topic]` in `[course]`?



Apache Jena Fuseki datasets manage help server status

/testData

query add data edit info

SPARQL Query

To try out some SPARQL queries against the selected dataset, enter your query here.

Example Queries

Selection of triples Selection of classes

Prefixes

rdf rdfs owl xsd

SPARQL Endpoint: /testData/sparql

Content Type (SELECT): JSON

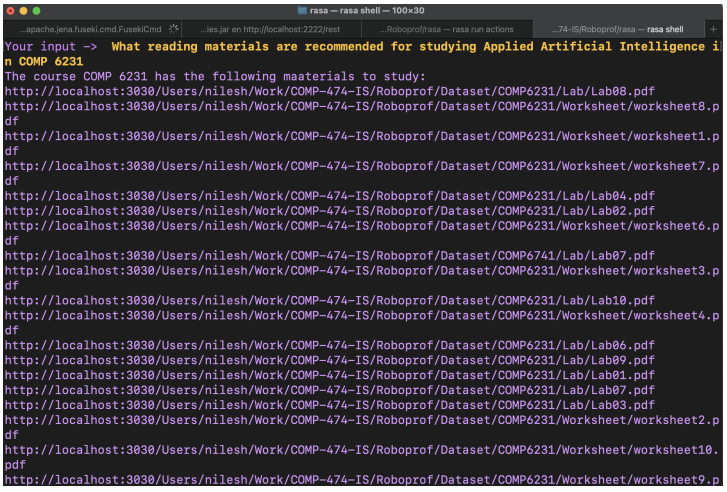
Content Type (GRAPH): Turtle

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX tc: <http://linkedscience.org/teach/ns#>
PREFIX cu: <http://is-concordia.io/>

SELECT ?topicName ?topics ?materials
WHERE {
    ?course cu:hasTopic ?topics ;
            cu:hasCourseSubject ?courseSubject;
            cu:hasCourseNumber ?courseNumber .
   ?topics foaf:name ?topicName ;
            cu:hasMaterials ?materials .
  FILTER(?courseSubject = "COMP")
  filter(?topicName= "Machine_learning")
}
```

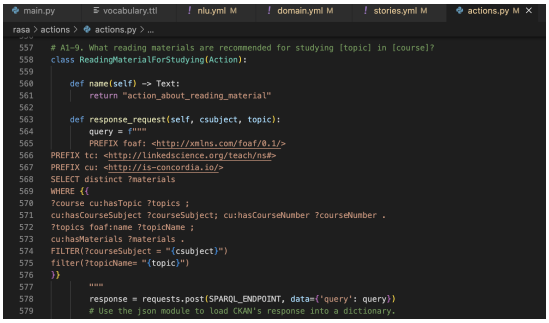
Table Response 1 result in 0.03 seconds Simple view Ellipse Filter query results Page size: 50

	topicName	topics	materials
1	Machine_learning	<http://is-concordia.io/Machine_learning>	https://www.ibm.com/topics/machine-learning

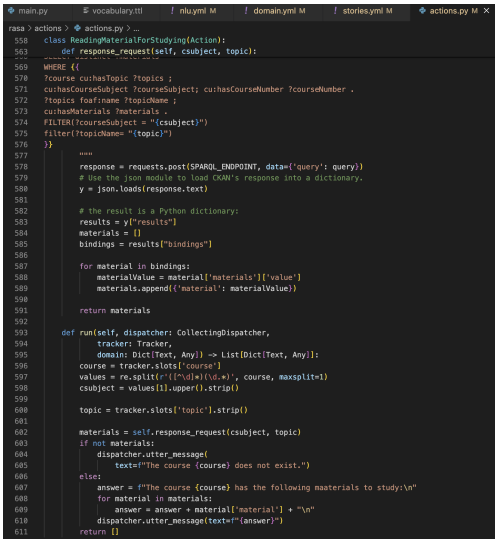
Showing 1 to 1 of 1 entries < 1 >



```
rasa — rasa shell — 100×30
...apache.jena.fuseki.cmd.FusekiCmd  ...ies.jar en http://localhost:2222/rest  ...Roboprof/rasa — rasa run actions  ...74-IS/Roboprof/rasa — rasa shell
Your input ->  What reading materials are recommended for studying Applied Artificial Intelligence i
n COMP 6231
The course COMP 6231 has the following maaterials to study:
http://localhost:3030/Users/nilesh/Work/COMP-474-IS/Roboprof/Dataset/COMP6231/Lab/Lab08.pdf
http://localhost:3030/Users/nilesh/Work/COMP-474-IS/Roboprof/Dataset/COMP6231/Worksheet/worksheet8.p
df
http://localhost:3030/Users/nilesh/Work/COMP-474-IS/Roboprof/Dataset/COMP6231/Worksheet/worksheet1.p
df
http://localhost:3030/Users/nilesh/Work/COMP-474-IS/Roboprof/Dataset/COMP6231/Worksheet/worksheet7.p
df
http://localhost:3030/Users/nilesh/Work/COMP-474-IS/Roboprof/Dataset/COMP6231/Lab/Lab04.pdf
http://localhost:3030/Users/nilesh/Work/COMP-474-IS/Roboprof/Dataset/COMP6231/Lab/Lab02.pdf
http://localhost:3030/Users/nilesh/Work/COMP-474-IS/Roboprof/Dataset/COMP6231/Worksheet/worksheet6.p
df
http://localhost:3030/Users/nilesh/Work/COMP-474-IS/Roboprof/Dataset/COMP6741/Lab/Lab07.pdf
http://localhost:3030/Users/nilesh/Work/COMP-474-IS/Roboprof/Dataset/COMP6231/Worksheet/worksheet3.p
df
http://localhost:3030/Users/nilesh/Work/COMP-474-IS/Roboprof/Dataset/COMP6231/Lab/Lab10.pdf
http://localhost:3030/Users/nilesh/Work/COMP-474-IS/Roboprof/Dataset/COMP6231/Worksheet/worksheet4.p
df
http://localhost:3030/Users/nilesh/Work/COMP-474-IS/Roboprof/Dataset/COMP6231/Lab/Lab06.pdf
http://localhost:3030/Users/nilesh/Work/COMP-474-IS/Roboprof/Dataset/COMP6231/Lab/Lab09.pdf
http://localhost:3030/Users/nilesh/Work/COMP-474-IS/Roboprof/Dataset/COMP6231/Lab/Lab01.pdf
http://localhost:3030/Users/nilesh/Work/COMP-474-IS/Roboprof/Dataset/COMP6231/Lab/Lab07.pdf
http://localhost:3030/Users/nilesh/Work/COMP-474-IS/Roboprof/Dataset/COMP6231/Lab/Lab03.pdf
http://localhost:3030/Users/nilesh/Work/COMP-474-IS/Roboprof/Dataset/COMP6231/Worksheet/worksheet2.p
df
http://localhost:3030/Users/nilesh/Work/COMP-474-IS/Roboprof/Dataset/COMP6231/Worksheet/worksheet10.
pdf
http://localhost:3030/Users/nilesh/Work/COMP-474-IS/Roboprof/Dataset/COMP6231/Worksheet/worksheet9.p
```

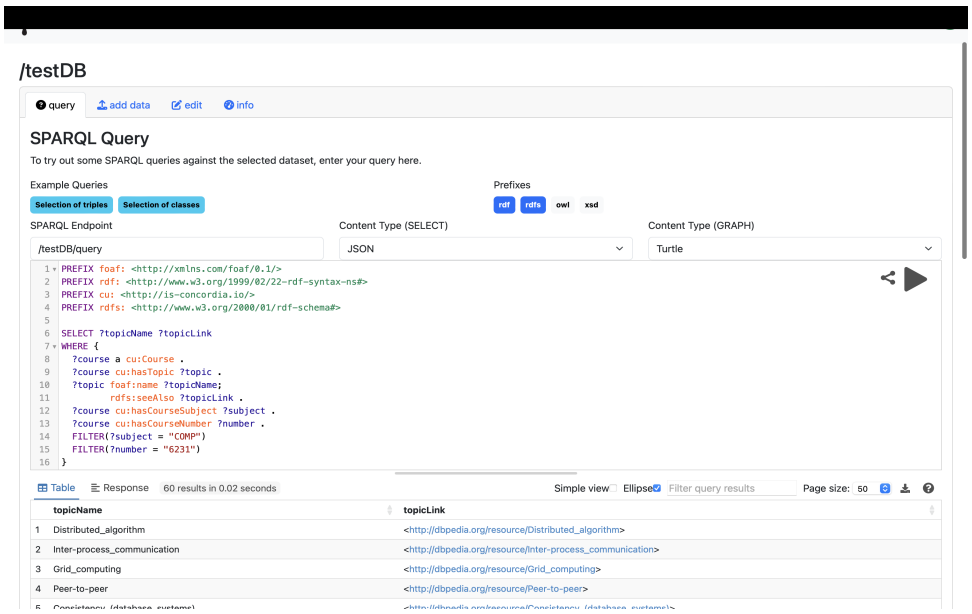


```
main.py  vocabulary.ttl  nlu.yml M  domain.yml M  stories.yml M  actions.py M
rasa > actions > actions.py > ...
557   # A1-9. What reading materials are recommended for studying [topic] in [course]?
558   class ReadingMaterialForStudying(Action):
559
560       def name(self) -> Text:
561           return "action_about_reading_material"
562
563       def response_request(self, csubject, topic):
564           query = f"""
565           PREFIX foaf: <http://xmlns.com/foaf/0.1/>
566   PREFIX tc: <http://linkedscience.org/teach/ns#>
567   PREFIX cu: <http://is-concordia.io/>
568   SELECT distinct ?materials
569   WHERE {{
570   ?course cu:hasTopic ?topics ;
571   cu:hasCourseSubject ?courseSubject; cu:hasCourseNumber ?courseNumber .
572   ?topics foaf:name ?topicName ;
573   cu:hasMaterials ?materials .
574   FILTER(?courseSubject = "{csubject}")
575   filter(?topicName= "{topic}")
576   }}
577           """
578           response = requests.post(SPARQL_ENDPOINT, data={'query': query})
579           # Use the json module to load CKAN's response into a dictionary.
```



```
main.py  vocabulary.ttl  nlu.yml M  domain.yml M  stories.yml M  actions.py M
rasa > actions > actions.py > ...
558   class ReadingMaterialForStudying(Action):
563       def response_request(self, csubject, topic):
569   WHERE {{
570   ?course cu:hasTopic ?topics ;
571   cu:hasCourseSubject ?courseSubject; cu:hasCourseNumber ?courseNumber .
572   ?topics foaf:name ?topicName ;
573   cu:hasMaterials ?materials .
574   FILTER(?courseSubject = "{csubject}")
575   filter(?topicName= "{topic}")
576   }}
577           """
578           response = requests.post(SPARQL_ENDPOINT, data={'query': query})
579           # Use the json module to load CKAN's response into a dictionary.
580           y = json.loads(response.text)
581
582           # the result is a Python dictionary:
583           results = y["results"]
584           materials = []
585           bindings = results["bindings"]
586
587           for material in bindings:
588               materialValue = material['materials']['value']
589               materials.append({'material': materialValue})
590
591           return materials
592
593       def run(self, dispatcher: CollectingDispatcher,
594               tracker: Tracker,
595               domain: Dict[Text, Any]) -> List[Dict[Text, Any]]:
596           course = tracker.slots['course']
597           values = re.split(r'([^\d]*)(\d.*)', course, maxsplit=1)
598           csubject = values[1].upper().strip()
599
600           topic = tracker.slots['topic'].strip()
601
602           materials = self.response_request(csubject, topic)
603           if not materials:
604               dispatcher.utter_message(
605                   text=f"The course {course} does not exist.")
606           else:
607               answer = f"The course {course} has the following maaterials to study:\n"
608               for material in materials:
609                   answer = answer + material['material'] + "\n"
610               dispatcher.utter_message(text=f"{answer}")
611           return []
```

- **Question 10**: What competencies [topics] does a student gain after completing `[course] [number]`?



/testDB

query | add data | edit | info

SPARQL Query

To try out some SPARQL queries against the selected dataset, enter your query here.

Example Queries: Selection of triples | Selection of classes

Prefixes: rdf | rdfs | owl | xsd

SPARQL Endpoint: /testDB/query

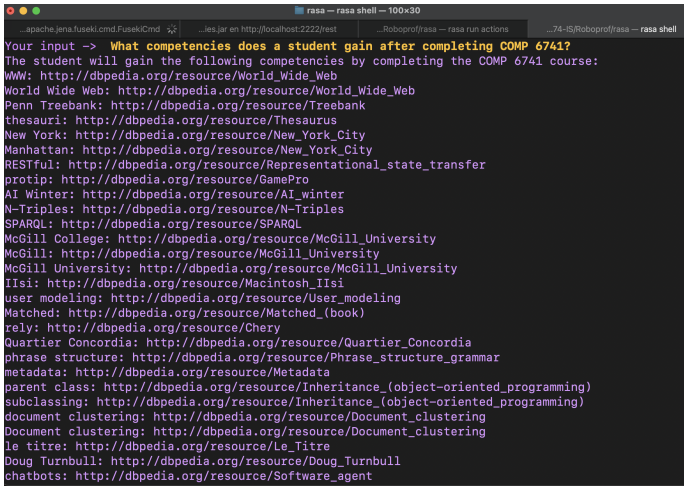
Content Type (SELECT): JSON

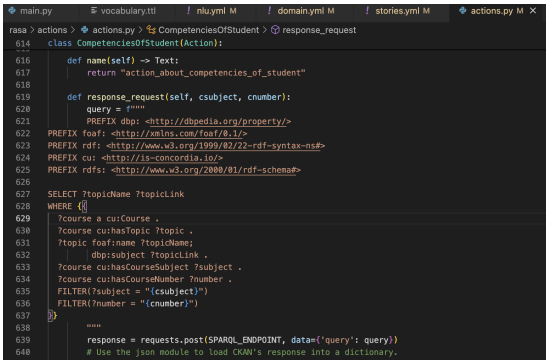
Content Type (GRAPH): Turtle

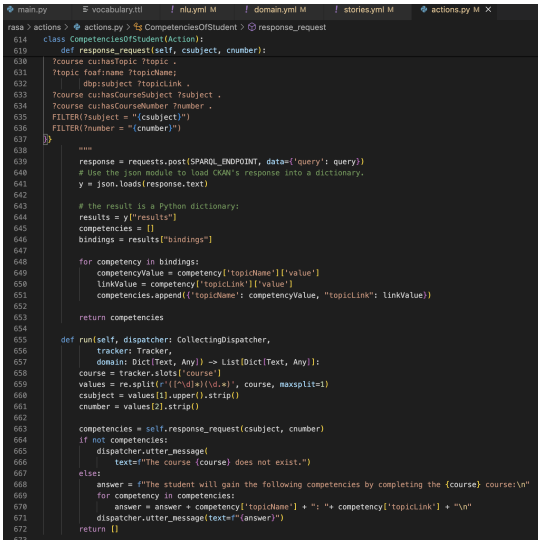
```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX cu: <http://is-concordia.io/>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>

SELECT ?topicName ?topicLink
WHERE {
  ?course a cu:Course .
  ?course cu:hasTopic ?topic .
  ?topic foaf:name ?topicName;
         rdfs:seeAlso ?topicLink .
  ?course cu:hasCourseSubject ?subject .
  ?course cu:hasCourseNumber ?number .
  FILTER(?subject = "COMP")
  FILTER(?number = "6231")
}
```

Table | Response | 60 results in 0.02 seconds | Simple view | Ellipse | Filter query results | Page size: 50

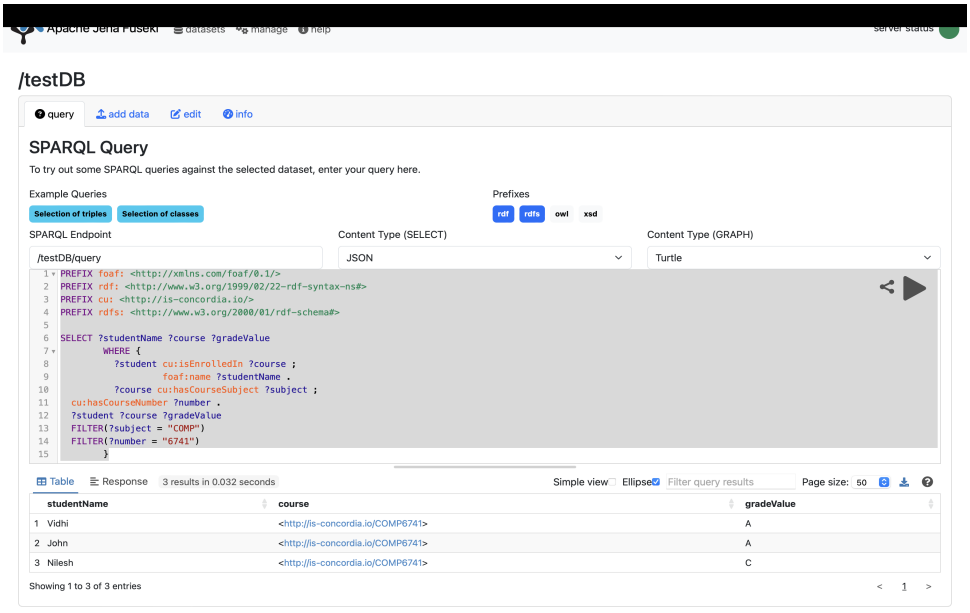
	topicName	topicLink
1	Distributed_algorithm	<http://dbpedia.org/resource/Distributed_algorithm>
2	Inter-process_communication	<http://dbpedia.org/resource/Inter-process_communication>
3	Grid_computing	<http://dbpedia.org/resource/Grid_computing>
4	Peer-to-peer	<http://dbpedia.org/resource/Peer-to-peer>
5	Consistency_(database_systems)	<http://dbpedia.org/resource/Consistency_(database_systems)>

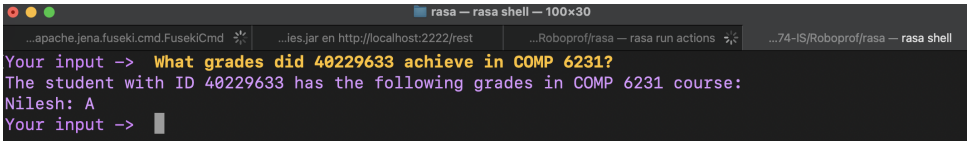




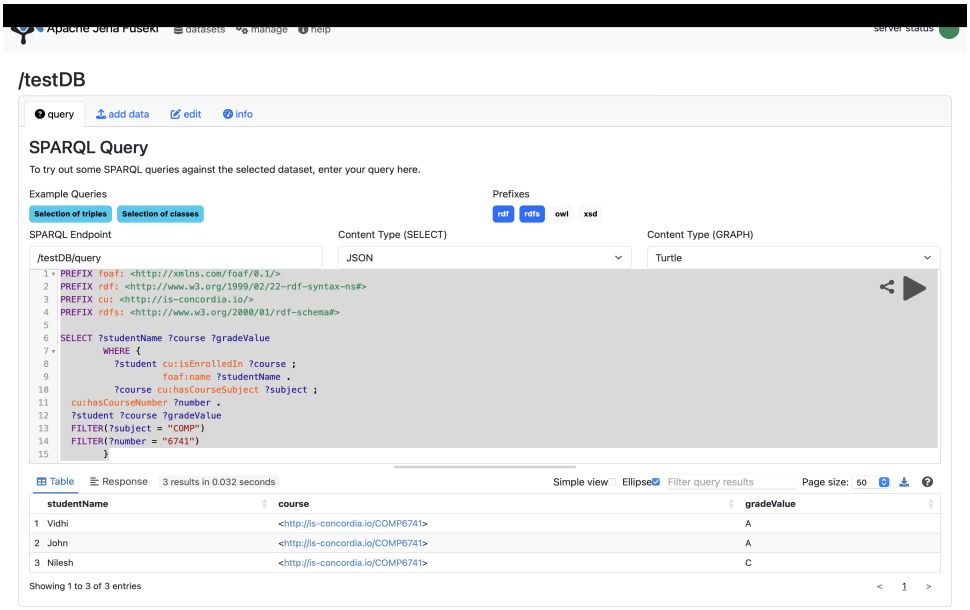


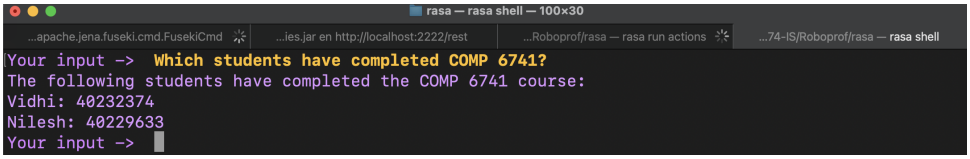
- **Question 11**: What grades did `[student]` achieve in `[course]` `[number]`?

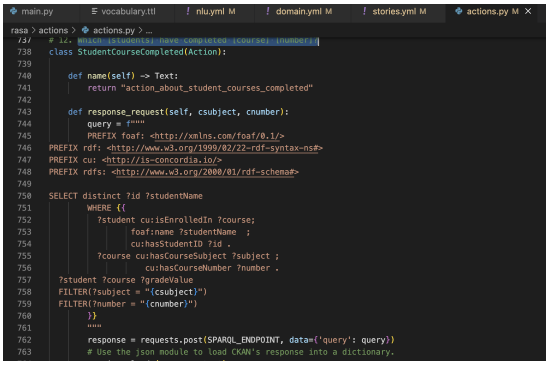


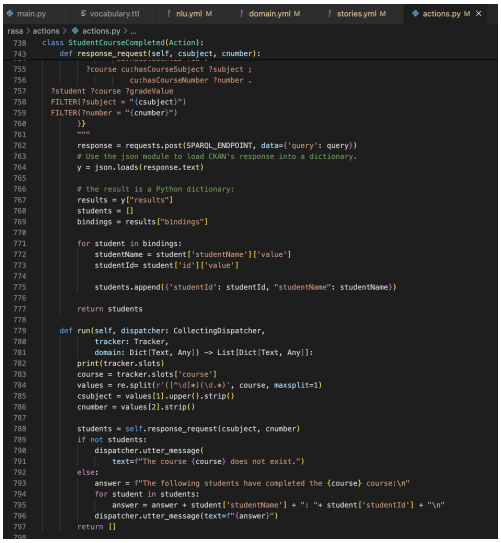


- **Question 12**: Which `[students]` have completed `[course]` `[number]`?

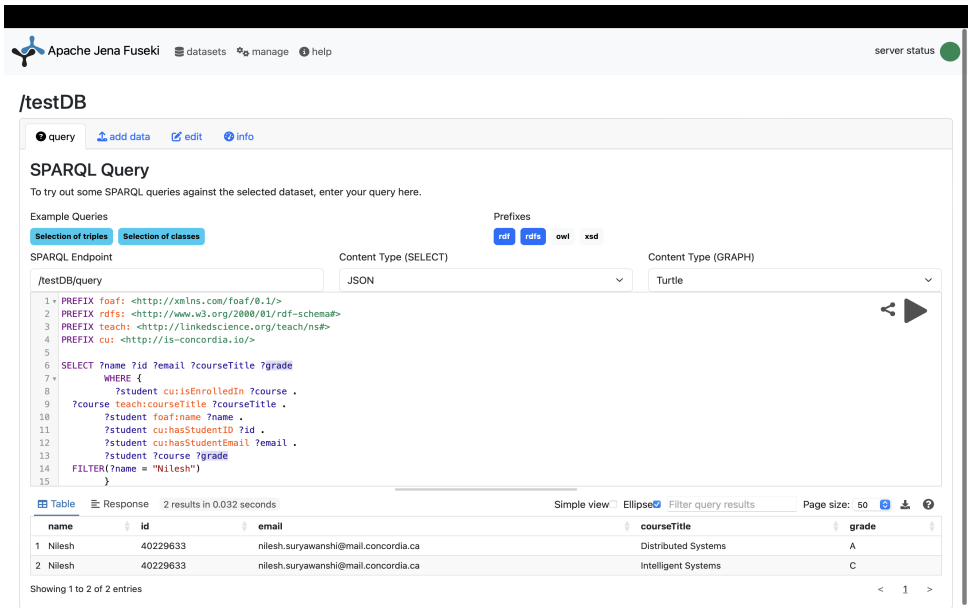


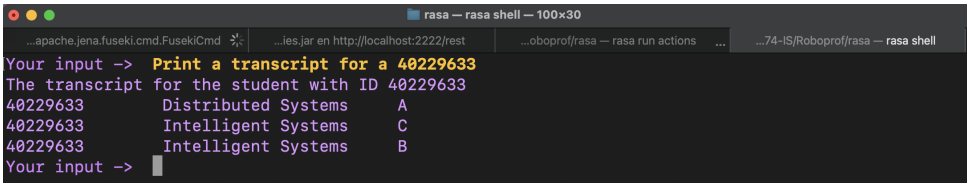


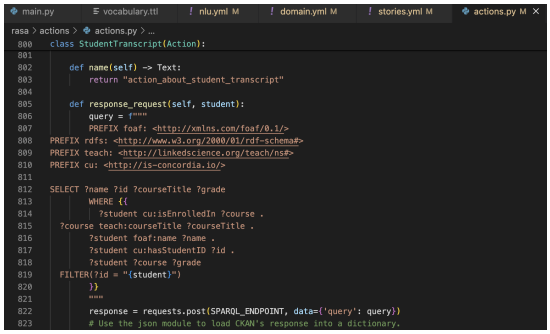


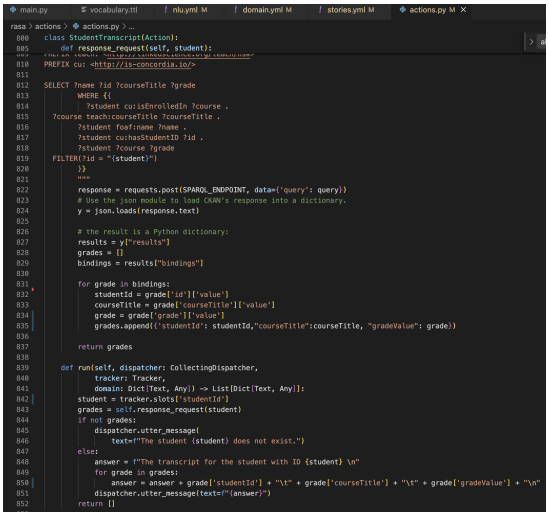


- **Question 13**: Print a transcript for a `[student]`, listing all the course taken with their grades.

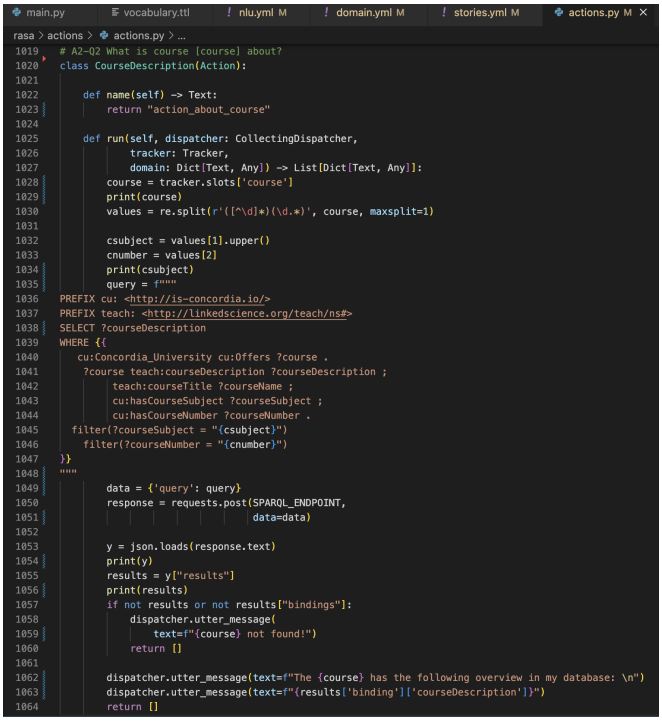








- **Question 14**: What is the `[course]` about?



- **Question 15**: Which topics are covered in `[course event]`?

```
rasa — rasa shell — 100×30
...apache.jena.fuseki.cmd.FusekiCmd | ...ies.jar en http://localhost:2222/rest | ...Roboprof/rasa — rasa run actions | ...74-IS/Roboprof/rasa — rasa shell
Your input ->  Which topics are covered in lab 3 of COMP 6741?
The following topics were covered in the lecture 3 of the course COMP6741:
SPARQL: http://dbpedia.org/resource/SPARQL
McGill College: http://dbpedia.org/resource/McGill_University
McGill: http://dbpedia.org/resource/McGill_University
McGill University: http://dbpedia.org/resource/McGill_University
chatbots: http://dbpedia.org/resource/Software_agent
data science: http://dbpedia.org/resource/Data_science
DBP: http://dbpedia.org/resource/Deutsche_Bundespost
Wikidata: http://dbpedia.org/resource/Wikidata
social media: http://dbpedia.org/resource/Social_network
social networking: http://dbpedia.org/resource/Social_network
social network: http://dbpedia.org/resource/Social_network
basics: http://dbpedia.org/resource/British_Association_for_Immediate_Care
Loyola: http://dbpedia.org/resource/Ignatius_of_Loyola
interoperability: http://dbpedia.org/resource/Interoperability
relationship extraction: http://dbpedia.org/resource/Relationship_extraction
graph: http://dbpedia.org/resource/Graph_database
graph database: http://dbpedia.org/resource/Graph_database
GraphDB: http://dbpedia.org/resource/Graph_database
rdf: http://dbpedia.org/resource/Resource_Description_Framework
RDF: http://dbpedia.org/resource/Resource_Description_Framework
Resource Description Framework: http://dbpedia.org/resource/Resource_Description_Framework
Resource Description Framework (RDF): http://dbpedia.org/resource/Resource_Description_Framework
wildcard: http://dbpedia.org/resource/Wildcard_character
English language: http://dbpedia.org/resource/English_language
English: http://dbpedia.org/resource/English_language
uniform structure: http://dbpedia.org/resource/Uniform_space
embark: http://dbpedia.org/resource/Embark_(transit_authority)
discoverability: http://dbpedia.org/resource/Discoverability
```

main.py | vocabulary.ttl | nlu.yml M | domain.yml M | stories.yml M | actions.py M

rasa > actions > actions.py > ActionTopicsCovered > run

```python
#Which topics are covered in <course event>?"
# A2-Q1. For a course c, list all covered topics t, printing out their English labels and their DBpedia
# together with the course event URI (e.g., 'lab3') and resource URI (e.g., 'slides10') where they appe
class ActionTopicsCovered(Action):

    def name(self) -> Text:
        return "action_about_topics_covered"

    def response_request(self, courseSubject, courseNumber, eventNumber, event):

        if event.lower() == "lab" or event.lower() == "laboratory":
            query = f"""
        PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX tc: <http://linkedscience.org/teach/ns#>
PREFIX cu: <http://is-concordia.io/>

SELECT ?topicName ?topics
WHERE {{
    cu:Concordia_University cu:Offers ?course .
    ?course tc:courseTitle ?title ;
        cu:hasLecture ?lecture ;
    cu:hasCourseSubject ?courseSubject ;
    cu:hasCourseNumber ?courseNumber .
  ?lecture cu:topicsCovered ?topics ;
            cu:hasLectureNumber ?lectureNumber .
  ?topics foaf:name ?topicName .
  FILTER(?courseSubject = "{courseSubject}")
  FILTER(?lectureNumber = {eventNumber})
  FILTER(?courseNumber = "{courseNumber}")
}}
        """
        else:
            query = f"""
        PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX tc: <http://linkedscience.org/teach/ns#>
PREFIX cu: <http://is-concordia.io/>

SELECT ?topicName ?topics
WHERE {{
    cu:Concordia_University cu:Offers ?course .
    ?course tc:courseTitle ?title ;
        cu:hasLab ?lab ;
```

```
main.py  vocabulary.ttl  nlu.yml M  domain.yml M  stories.yml M  actions.py M
rasa > actions > actions.py > ActionTopicsCovered > run
858  class ActionTopicsCovered(Action):
863      def response_request(self, courseSubject, courseNumber, eventNumber, event):
891  PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
892  PREFIX foaf: <http://xmlns.com/foaf/0.1/>
893  PREFIX tc: <http://linkedscience.org/teach/ns#>
894  PREFIX cu: <http://is-concordia.io/>
895
896  SELECT ?topicName ?topics
897  WHERE {{
898      cu:Concordia_University cu:Offers ?course .
899      ?course tc:courseTitle ?title ;
900          cu:hasLab ?lab ;
901      cu:hasCourseSubject ?courseSubject ;
902      cu:hasCourseNumber ?courseNumber .
903    ?lab cu:topicsCovered ?topics ;
904            cu:hasLabNumber ?labNumber .
905    ?topics foaf:name ?topicName .
906    FILTER(?courseSubject = "{courseSubject}")
907    FILTER(?labNumber = {eventNumber})
908    FILTER(?courseNumber = "{courseNumber}")
909  }}
910          """
911
912          response = requests.post(SPARQL_ENDPOINT,
913                                   data={'query': query})
914          # Use the json module to load CKAN's response into a dictionary.
```

- **Question 16**: Which course events cover `[Topic]`?

```
rasa — rasa shell — 100×30
...apache.jena.fuseki.cmd.FusekiCmd  ...ies.jar en http://localhost:2222/rest  ...Roboprof/rasa — rasa run actions  ...74-IS/Roboprof/rasa — rasa shell
Your input ->  Which topics are covered in lab 3 of COMP 6741?
The following topics were covered in the lecture 3 of the course COMP6741:
SPARQL: http://dbpedia.org/resource/SPARQL
McGill College: http://dbpedia.org/resource/McGill_University
McGill: http://dbpedia.org/resource/McGill_University
McGill University: http://dbpedia.org/resource/McGill_University
chatbots: http://dbpedia.org/resource/Software_agent
data science: http://dbpedia.org/resource/Data_science
DBP: http://dbpedia.org/resource/Deutsche_Bundespost
Wikidata: http://dbpedia.org/resource/Wikidata
social media: http://dbpedia.org/resource/Social_network
social networking: http://dbpedia.org/resource/Social_network
social network: http://dbpedia.org/resource/Social_network
basics: http://dbpedia.org/resource/British_Association_for_Immediate_Care
Loyola: http://dbpedia.org/resource/Ignatius_of_Loyola
interoperability: http://dbpedia.org/resource/Interoperability
relationship extraction: http://dbpedia.org/resource/Relationship_extraction
graph: http://dbpedia.org/resource/Graph_database
graph database: http://dbpedia.org/resource/Graph_database
GraphDB: http://dbpedia.org/resource/Graph_database
rdf: http://dbpedia.org/resource/Resource_Description_Framework
RDF: http://dbpedia.org/resource/Resource_Description_Framework
Resource Description Framework: http://dbpedia.org/resource/Resource_Description_Framework
Resource Description Framework (RDF): http://dbpedia.org/resource/Resource_Description_Framework
wildcard: http://dbpedia.org/resource/Wildcard_character
English language: http://dbpedia.org/resource/English_language
English: http://dbpedia.org/resource/English_language
uniform structure: http://dbpedia.org/resource/Uniform_space
embark: http://dbpedia.org/resource/Embark_(transit_authority)
discoverability: http://dbpedia.org/resource/Discoverability
```

```
main.py  vocabulary.ttl  nlu.yml M  domain.yml M  stories.yml M  actions.py M
rasa > actions > actions.py > ActionTopicsCovered > run
858  class ActionTopicsCovered(Action):
863      def response_request(self, courseSubject, courseNumber, eventNumber, event):
921          topics = []
922
923          for result in results["bindings"]:
924              topicName = result["topicName"]
925              topic = result["topics"]
926
927              topicNameValue = topicName["value"]
928              topicValue = topic["value"]
929
930              row = {"topicName": topicNameValue, "topic": topicValue}
931              topics.append(row)
932
933          return topics
934
935      def run(self, dispatcher: CollectingDispatcher,
936              tracker: Tracker,
937              domain: Dict[Text, Any]) -> List[Dict[Text, Any]]:
938          print(tracker.slots)
939          course = tracker.slots['course'].replace(" ", "")
940
941          values = re.split(r'([^\d]*)(\d.*)', course, maxsplit=1)
942
943          csubject = values[1].upper()
944          cnumber = values[2]
945
946          event = tracker.slots['courseEvent'].strip()
947          eventValues = re.split(
948              r'([^\d]*)(\d.*)', event, maxsplit=1)
949          eventNumber = eventValues[2]
950          event = eventValues[1]
951          topics = self.response_request(csubject, cnumber, eventNumber, event)
952
953          if not topics:
954              dispatcher.utter_message(
955                  text=f"No topics covered in {eventNumber} or {eventNumber} did not happen.")
956          else:
957              answer = f"The following topics were covered in the lecture {eventNumber} of the course {csubject}{cnumber}:\n"
958              for topic in topics:
959                  answer = answer + topic['topicName'] + \
960                      ": " + topic['topic'] + "\n"
961              dispatcher.utter_message(text=f"{answer}")
962
963          return []
```

References

[1] W3C. (n.d.). RDF 1.1 Concepts and Abstract Syntax. Retrieved from https://www.w3.org/TR/rdf11-concepts/

[2] W3C. (n.d.). RDF Schema 1.1. Retrieved from https://www.w3.org/TR/rdf-schema/

[3] W3C. (n.d.). OWL 2 Web Ontology Language Document Overview. Retrieved from https://www.w3.org/TR/owl2-overview/

[4] Linked Science. (n.d.). Teach: An ontology for teaching courses on the Web of Data. Retrieved from http://linkedscience.org/teach/ns#

[5] DBpedia. (n.d.). DBpedia: A Multilingual Cross-domain Knowledge Base. Retrieved from https://wiki.dbpedia.org/

[6] Apache Jena. (n.d.). Apache Jena Fuseki. Retrieved from https://jena.apache.org/documentation/fuseki2/

[7] Accessed on 22nd March 2024, GPT-4 from OpenAI. https://openai.com/gpt-4