



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	2
1.1	External Vocabularies Used	2
1.2	Schema Justification	2
1.3	Schema Properties	3
2	Knowledge Base Construction	3
2.1	Dataset	3
2.2	Process and Developed Tools for Populating the Knowledge Base from the Dataset	4
2.3	How to Run the Tools to Create the Knowledge Base	4
3	Triplestore and SPARQL Endpoint Setup	4
3.1	Selection of Apache Fuseki Server	4
3.2	Installation and Configuration	5
3.3	Challenges Encountered and Addressed	5
4	Knowledge Base Population	6
4.1	Population of Triples into Knowledge Base (KB)	6
5	Chatbot Design	7
5.1	Utilizing <i>Rasa</i> for the Natural Language Conversation Interface	7
5.2	Intents	8
5.3	Stories Defined for Conversation Flows	9
5.4	Entities for Chatbot	9
5.5	Actions for Chatbot	10
5.6	Integration Process with Fuseki Server and SPARQL Queries	10
5.7	Addressing Out-of-Vocabulary (OOV) Words	11
5.8	Crafting Natural Language Responses	11
5.9	Output Samples	11
	References	27

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]

The screenshot shows the Apache Jena Fuseki interface for a dataset named '/testDB'. The top navigation bar includes links for 'query', 'add data', 'edit', and 'info'. On the right, there's a 'server status' indicator. Below the navigation, the page title is '/testDB'. A 'SPARQL Query' section contains a text input field with the following SPARQL query:

```

PREFIX tc: <http://linkedscience.org/teach/ns#>
PREFIX cu: <http://is-concordia.io/>
SELECT ?course ?title
WHERE {
  cu:Concordia_University cu:offers ?course .
  ?course tc:courseTitle ?title .
}

```

Below the query, the results are displayed in a table:

course	title
<http://is-concordia.io/MUS14...>	Supervised Internship II
<http://is-concordia.io/FNA411>	Portfolio Management

At the bottom of the results table, there are buttons for 'Table', 'Response', and 'Simple view'. There are also filters for 'Ellipse' and 'Filter query results', and a 'Page size' dropdown set to 50. A green circular button with a checkmark is visible in the top right corner of the main content area.

```

Your input -> What courses are offered by Concordia?
Course offered by Concordia:
Supervised Internship II
Portfolio Management
Hydraulic Structures
Engineering Metrology and Measurement Systems
Special Topics in Sculpture
Discrete Optimization
Strategies in Action (Case Competition)
Honours Essay
Scriptwriting for the Media
Women in Jewish History: Modern
Language Assessment
Creation Project
Entrepreneurship: Launching Your Business
Computer-Aided Mechanical Design
Microwave Engineering
Athletic Therapy Field Internship II
Introduction to Analog VLSI
Introductory Experimental Waves and Modern Physics
MBA Surgical Innovation II
Painting IV
Heat Transfer II
Advanced Java Platforms
MSc Seminar on Selected Topics
Image Processing
Reinforced Earth
Working in Childhood Settings: Leadership and Organizational Issues
Special Topics in Design for the Theatre
Personal Selling

```

```

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

rasa > actions > actions.py > TopicsCourseLecture > name
rasa > class TopicsCourseLecture > @name
rasa > actions > actions.py > TopicsCourseLecture > @name
53     class TopicsCourseLecture(Action):
54         def __init__(self) -> Text:
55             self.name = "action_about_courses_offered_by_university"
56         def run(self, dispatcher: CollectingDispatcher,
57               tracker: Tracker,
58               domain: Dict[Text, Any]) -> List[Dict[Text, Any]]:
59             university = tracker.slots['university']
60             query = f"""
61             PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
62             PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
63             PREFIX foaf: <http://xmlns.com/foaf/0.1/>
64             PREFIX tc: <http://linkedscience.org/teach/ns#>
65             PREFIX cu: <http://is-concordia.io/>
66             SELECT ?title
67             WHERE {
68               cu:Concordia_University cu:offers ?course .
69               ?course tc:courseTitle ?title .
70             }
71             """
72             response = requests.post(SPARQL_ENDPOINT,
73                                     data={"query": query})
74             # Use the json module to load OKWN's response into a dictionary.
75             results = response.json()
76             courses_offered = []
77             for result in results["bindings"]:
78               course_name = result["?title"]["value"]
79               courses_offered.append(course_name)
80             dispatcher.utter_message(text=f"I have no knowledge of courses offered by {university}.")
81             else:
82               answer = f"Course offered by {university} are:\n"
83               for course in courses_offered:
84                 answer += course + "\n"
85             dispatcher.utter_message(text=f"({answer})")
86             return []
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192

```

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

topic	course	link
1 <http://is-concordia.io/Machine_learning>	<http://is-concordia.io/COMP6231>	<http://dbpedia.org/resource/Machine_learning>
2 <http://is-concordia.io/Machine_learning>	<http://is-concordia.io/COMP6741>	<http://dbpedia.org/resource/Machine_learning>

```
Your input -> In which courses is RDF discussed?
The following courses cover the topic RDF:
COMP6231: http://dbpedia.org/resource/Resource_Description_Framework
COMP6741: http://dbpedia.org/resource/Resource_Description_Framework
Your input ->
```

```
actions.py
class WhichCourseHasTopic(Action):
    def name(self) -> Text:
        return "action_about_course_uni_topic"

    def response(self, topicName):
        query = """
        PREFIX dbp: <http://dbpedia.org/property/>
        PREFIX foaf: <http://xmlns.com/foaf/0.1/>
        PREFIX rdfs: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
        PREFIX cu: <http://is-concordia.io/>
        PREFIX rdf: <http://www.w3.org/2000/01/rdf-schema#>

        SELECT ?courseName ?link
        WHERE {
            ?course a cu:Course .
            ?course rdfs:label ?courseName .
            ?course cu:hasTopic ?topic .
            ?topic foaf:name ?name .
            ?topic cu:topicSubject ?link .
        }
        FILTER(?name = '{topicName}')
    }

    response = requests.post(SPARQL_ENDPOINT,
                            data={"query": query})
    # Use the json module to load CKAN's response into a dictionary.
    y = json.loads(response.text)

    # The result is a Python dictionary.
    results = y["results"]

    courses = []
    for result in results["bindings"]:
        course = result["courseName"]
        topicLink = result["link"]
        courseValue = course["value"]
        topicLinkValue = topicLink["value"]
        row = {"courseName": courseValue, "link": topicLinkValue}
        courses.append(row)

    return courses
```

```
main.py
class WhichCourseHasTopic(Action):
    def response(self, topicName):
        response = requests.post(SPARQL_ENDPOINT,
                                data={"query": query})
        # Use the json module to load CKAN's response into a dictionary.
        y = json.loads(response.text)

        # The result is a Python dictionary.
        results = y["results"]

        courses = []

        for result in results["bindings"]:
            course = result["courseName"]
            topicLink = result["link"]
            courseValue = course["value"]
            topicLinkValue = topicLink["value"]
            row = {"courseName": courseValue, "link": topicLinkValue}
            courses.append(row)

        return courses

    def run(self, dispatcher: CollectingDispatcher,
           tracker: Tracker,
           domain: Dict[Text, Any]) -> List[Dict[Text, Any]]:
        topic = tracker.slots['topic']

        courses = self.response(topic)

        if not courses:
            dispatcher.utter_message(
                text="No course covers the topic {topic}.")

        else:
            answer = "The following courses cover the topic {topic}:\n"
            for course in courses:
                answer += course["courseName"] + \
                          ": " + course["link"] + "\n"
            dispatcher.utter_message(text=answer)
```

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

Selection of triples Selection of classes

SPARQL Endpoint Content Type (SELECT) Content Type (GRAPH)

/testData/ JSON Turtle

```
1 v PREFIX foaf: <http://xmlns.com/foaf/0.1/>
2 v PREFIX tci: <http://linkedscience.org/teach/ns#>
3 v PREFIX cu: <http://is-concordia.io/>
4 v
5 v SELECT ?topicName ?topics
6 v WHERE {
7 v     cu:Concordia_University cu:Offers ?course .
8 v     ?course tci:courseTitle ?title ;
9 v         cu:hasLecture ?lecture ;
10 v        cu:hasCourseSubject ?courseSubject .
11 v        ?lecture cu:topicsCovered ?topics ;
12 v        cu:hasLectureNumber ?lectureNumber .
13 v        ?topics foaf:name ?topicName .
14 v        FILTER(?courseSubject = "COMP")
15 v        FILTER(?lectureNumber = 4)
16 }
```

Table Response 11 results in 0.027 seconds

topicName	topics
1 Grid_computing	<http://is-concordia.io/Grid_computing>
2 High_availability	<http://is-concordia.io/High_availability>
3 Parallel_computing	<http://is-concordia.io/Parallel_computing>
4 Scalability	<http://is-concordia.io/Scalability>
5 Cloud_computing	<http://is-concordia.io/Cloud_computing>
6 Planning_algorithm	<http://is-concordia.io/Planning_algorithm>
7 Semantic_web	<http://is-concordia.io/Semantic_web>
8 Inference_engine	<http://is-concordia.io/Inference_engine>
9 Knowledge_engineering	<http://is-concordia.io/Knowledge_engineering>
10 Expert_system_shell	<http://is-concordia.io/Expert_system_shell>
11 Evolutionary_computation	<http://is-concordia.io/Evolutionary_computation>

Showing 1 to 11 of 11 entries

```

Your input -> Which topics are covered in COMP 6741 during lec 3?
The following topics were covered in the lecture 3 of the course COMP6741:
thesauri: http://dbpedia.org/resource/TheSaurus
SPARQL: http://dbpedia.org/resource/SPARQL
metadata: http://dbpedia.org/resource/Metadata
SKOS: http://dbpedia.org/resource/Simple_Knowledge_Organization_System
skos: http://dbpedia.org/resource/Simple_Knowledge_Organization_System
ACM: http://dbpedia.org/resource/Association_for_Computing_Machinery
Le Palais: http://dbpedia.org/resource/Le_Palais
interoperability: http://dbpedia.org/resource/Interoperability
Oscars: http://dbpedia.org/resource/Academy_Awards
Dewey Decimal Classification: http://dbpedia.org/resource/Dewey_Decimal_Classification
N3: http://dbpedia.org/resource/Notation3
RDFa: http://dbpedia.org/resource/RDFa
Harper Collins: http://dbpedia.org/resource/HarperCollins
Json: http://dbpedia.org/resource/Json_(rapper)
XML: http://dbpedia.org/resource/XML
RDF/XML: http://dbpedia.org/resource/XML
Manning Publications: http://dbpedia.org/resource/Manning_Publications
Reading: http://dbpedia.org/resource/Reading
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
Chapter 7: http://dbpedia.org/resource/Chapter_VII_of_the_United_Nations_Charter
Metafiction: http://dbpedia.org/resource/Metafiction
Luke: http://dbpedia.org/resource/Gospel_of_Luke

```

The terminal window shows the Rasa shell output:

```

Your input -> Which topics are covered in COMP 6741 during lec 3?
The following topics were covered in the lecture 3 of the course COMP6741:
thesauri: http://dbpedia.org/resource/TheSaurus
SPARQL: http://dbpedia.org/resource/SPARQL
metadata: http://dbpedia.org/resource/Metadata
SKOS: http://dbpedia.org/resource/Simple_Knowledge_Organization_System
skos: http://dbpedia.org/resource/Simple_Knowledge_Organization_System
ACM: http://dbpedia.org/resource/Association_for_Computing_Machinery
Le Palais: http://dbpedia.org/resource/Le_Palais
interoperability: http://dbpedia.org/resource/Interoperability
Oscars: http://dbpedia.org/resource/Academy_Awards
Dewey Decimal Classification: http://dbpedia.org/resource/Dewey_Decimal_Classification
N3: http://dbpedia.org/resource/Notation3
RDFa: http://dbpedia.org/resource/RDFa
Harper Collins: http://dbpedia.org/resource/HarperCollins
Json: http://dbpedia.org/resource/Json_(rapper)
XML: http://dbpedia.org/resource/XML
RDF/XML: http://dbpedia.org/resource/XML
Manning Publications: http://dbpedia.org/resource/Manning_Publications
Reading: http://dbpedia.org/resource/Reading
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
Chapter 7: http://dbpedia.org/resource/Chapter_VII_of_the_United_Nations_Charter
Metafiction: http://dbpedia.org/resource/Metafiction
Luke: http://dbpedia.org/resource/Gospel_of_Luke

```

The code editor windows show the following files:

- main.py**: Contains logic for handling actions related to topics covered in lectures.
- vocabulary.ttl**: RDF triples defining prefixes and namespaces.
- actions.py**: Contains the definition of the `WhichTopicInCourseDuringLecture` action.
- actions.py**: Contains the implementation of the `response_request` method for the `WhichTopicInCourseDuringLecture` action.

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

The Apache Jena Fuseki interface shows the following SPARQL query results:

```

PREFIX tcc: <http://LinkedScience.org/teach/ns#>
PREFIX cui: <http://is-concordia.io/>
SELECT ?course ?title ?subject
WHERE {
  ?course tcc:courseTitle ?title .
  ?course cui:courseSubject ?subject .
  FILTER(?subject = "COMP")
}

```

The results table displays the following data:

course	title	subject
<http://is-concordia.io/COMP478>	Image Processing	COMP
<http://is-concordia.io/COMP5481>	Programming and Problem Solving	COMP
<http://is-concordia.io/COMP6461>	Computer Networks and Protocols	COMP
<http://is-concordia.io/COMP6761>	Advanced Computer Graphics	COMP
<http://is-concordia.io/COMP6321>	Machine Learning	COMP
<http://is-concordia.io/COMP445>	Data Communication and Computer Networks	COMP
<http://is-concordia.io/COMP445>	Database Design	COMP

```
Your input -> List all courses offered by Concordia within the SOEN
The courses offered by the subject SOEN are:
SOEN490: Capstone Software Engineering Design Project
SOEN487: Web Services and Applications
SOEN390: Software Engineering Team Design Project
SOEN341: Software Process
SOEN491: Software Engineering Project
SOEN385: Control Systems and Applications
SOEN228: System Hardware
SOEN423: Distributed Systems
SOEN422: Embedded Systems and Software
Your input ->
```

```
main.py actions.py vocabulary.ttl domain.yml stories.yml actions.py

254 # AI-4. [list all courses] offered by [university] within the [subject] (e.g., "COMP", "SOEN").
255 class WhichCourseWithinSubject(Action):
256     def name(self) -> Text:
257         return "action_about_courses_within_subject"
258
259     def response_request(self, cssubject: Text):
260         PREFIX foaf: <http://xmlns.com/foaf/0.1/>
261         PREFIX tc: <https://linkedsience.org/teach/ns#>
262         PREFIX cu: <http://is-concordia.io/>
263
264         SELECT ?courseLabel ?title
265         WHERE {
266             cu:Concordia University . cu:Offers ?course .
267             ?course ?courseTitle ?title .
268             rdfs:label ?courseLabel .
269             ?course cu:hasCourseSubject ?subject .
270             FILTER(?subject = "(cssubject)")
```

The code continues with the SPARQL query and the Python logic to handle the response from the CKAN endpoint:

```
271         }
272         """
273         response = requests.post(SPARQL_ENDPOINT,
274             data={"query": query})
275         # Use the json module to load CKAN's response into a dictionary.
276         y = json.loads(response.text)
277
278         # the result is a Python dictionary:
279         results = y["results"]
280
281         courses = []
282
283         for result in results["bindings"]:
284             courseLabel = result["courseLabel"]["value"]
285             courseTitle = result["title"]["value"]
286
287             courseLabelValue = courseLabel["value"]
288             courseTitleValue = courseTitle["value"]
289             courses.append({"courseLabel": courseLabelValue,
290                             "title": courseTitleValue})
291
292         return courses
293
294
295         def run(self, dispatcher: CollectingDispatcher,
296                 tracker: Tracker,
297                 domain: Dict[Text, Any]) -> List[Dict[Text, Any]]:
298             print(tracker.slots)
299             subject = tracker.slots["subject"]
300
301             courses = self.response_request(subject)
302
303             if not courses:
304                 dispatcher.utter_message(
305                     text=f"(subject) does not offer any courses.")
306             else:
307                 answer = f"The courses offered by the subject {subject} are:\n"
308                 for course in courses:
309                     answer = answer + course['courseLabel'] + \
310                         " " + course['title'] + "\n"
311
312                 dispatcher.utter_message(text=answer)
313
314         return []
```

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

The screenshot shows the Apache Jena Fuseki interface with a SPARQL query results page. The query is:

```
1+ PREFIX foaf: <http://xmlns.com/foaf/0.1/>
2+ PREFIX tc: <https://linkedsience.org/teach/ns#>
3+ PREFIX cu: <http://is-concordia.io/>
4+
5+ SELECT ?topicName ?topics ?materials
6+ WHERE {
7+   ?course cu:hasTopic ?topics ;
8+   cu:hasCourseSubject ?courseSubject;
9+   cu:hasCourseNumber ?courseNumber .
10+  ?topics foaf:name ?topicName ;
11+    cu:hasMaterials ?materials .
12+  FILTER(?courseSubject = "COMP")
13+  filter(?courseNumber = "6231")
14+  filter(?topicName = "Sharding")
15+ }
```

The results table shows one entry:

topicName	topics	materials
1 Sharding	<http://is-concordia.io/Sharding>	https://www.mongodb.com/docs/manual/sharding/

```

...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 materials are recommended for convolution in COMP 6741?
The course COMP 6741 has the following material:
http://localhost:3030/Users/nilesh/Work/COMP-474-IS/Roboprof/Dataset/COMP6741/Lab/Lab10.pdf
http://localhost:3030/Users/nilesh/Work/COMP-474-IS/Roboprof/Dataset/COMP6231/Slide/slide7.pdf
http://localhost:3030/Users/nilesh/Work/COMP-474-IS/Roboprof/Dataset/COMP6231/Worksheet/worksheet6.pdf
http://localhost:3030/Users/nilesh/Work/COMP-474-IS/Roboprof/Dataset/COMP6231/Lab/Lab07.pdf
Your input ->

```

```

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

rasa > actions > actions.py > ...
316 # AI-5. What [materials] (slides, readings) are recommended for [topic] in [course] [number]?
317 class MaterialForCourse(Action):
318     def name(self) -> Text:
319         return "action_about_materials"
320
321     def response(self, cssubject, cnumber, topic):
322         query = """
323             PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
324             PREFIX foaf: <http://linkedscience.org/teach/ns#>
325             PREFIX tci: <http://linkedscience.org/teach/ns#>
326             PREFIX cu: <http://is-concordia.io/>
327             SELECT ?material
328             WHERE {
329                 ?course cu:hasTopic ?topic .
330                 ?course cu:hasCourseSubject ?courseSubject; cu:hasCourseNumber ?courseNumber .
331                 ?topic cu:hasMaterial ?material .
332                 FILTER(?courseNumber = "(cnumber)" )
333                 FILTER(?topicName = "(topic)" )
334             }
335             """
336             response = requests.post(SPARQL_ENDPOINT, data={'query': query})
337             # Use the json module to load CKAN's response into a dictionary.
338
339             response = requests.get(SPARQL_ENDPOINT, params={'query': query})
340             # Use the json module to load CKAN's response into a dictionary.
341
342             materials = []
343             for material in response['bindings']:
344                 materials.append(material['material']['value'])
345             return materials
346
347         def run(self, dispatcher: CollectingDispatcher,
348               tracker: Tracker,
349               domain: Dict[Text, Any] -> List[Dict[Text, Any]]:
350                 course = tracker.slots['course']
351                 values = re.split('([0-9]+)', course, maxsplit=1)
352                 if len(values) > 1:
353                     course = values[0].strip()
354                 else:
355                     course = values[1].strip()
356                 topic = tracker.slots['topic'].strip()
357                 materials = self.response_request(cssubject, cnumber, topic)
358                 if not materials:
359                     dispatcher.utter_message(
360                         text=f"The course {course} does not exist or is not offered by {university}.")
361                 else:
362                     answer = f"The course {course} has the following material:\n"
363                     for material in materials:
364                         answer += material['material'] + "\n"
365                     dispatcher.utter_message(text=f"{answer}")
366             return []
367
368
369
370
371

```

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

The screenshot shows the Apache Jena Fuseki interface with a SPARQL query entered. The query is:

```

PREFIX tci: <http://linkedscience.org/teach/ns#>
PREFIX cu: <http://is-concordia.io/>
SELECT ?course ?credits
WHERE {
  ?course cu:hasCredits ?credits ;
  cu:hasCourseSubject ?subject;
  cu:hasCourseNumber ?number .
  FILTER(?subject = "HIST")
  FILTER(?number = "397")
}

```

The results table shows one entry: course <http://is-concordia.io/HIST397> with credits 3.0.

```

...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 -> How many credits is HIST 397 worth?
The course HIST397 is of 3.0 credits.
Your input ->

```

```

main.py          vocabulary.ttl      nlu.yml M    domain.yml M    stories.yml M    actions.py M X
rasa > actions > actions.py > ...
374 # AI-6. How many credits is [course] [number] worth?
375 class ContentCourseLectureAction:
376     def name(self) -> Text:
377         return "action_about_credits"
378
379     def response_request(self, cssubject, cnumber):
380         query = """
381             PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
382             PREFIX tci: <http://LinkedScience.org/teach/ns#>
383             PREFIX cu: <http://is-concordia.io/>
384
385             SELECT ?courseLabel ?credits
386             WHERE {
387                 ?course cu:hasCredits ?credits ;
388                 cu:courseNumber ?cssubject;
389                 cu:courseLabel ?courseLabel .
390
391                 FILTER(?subject = "(cssubject)")
392                 FILTER(?number = "(cnumber)")
393             }
394             """
395
396         response = requests.post(SPARQL_ENDPOINT, data={"query": query})
397         # Use the json module to load CKAN's response into a dictionary,
398
399         results = json.loads(response.text)
400
401         courseLabel = results["bindings"][0]["courseLabel"]
402         credits = results["bindings"][0]["credits"]
403         courseLabelValue = courseLabel["value"]
404         creditsValue = credits["value"]
405
406         return {
407             "courseLabel": courseLabelValue,
408             "credits": creditsValue
409         }
410
411     def run(self, dispatcher: CollectingDispatcher,
412           tracker: Tracker,
413           domain: Dict[Text, Any]) -> List[Dict[Text, Any]]:
414
415         course = tracker.slots["course"].replace(" ", "")
416         university = tracker.slots["university"]
417
418         values = re.split("(\\(|\\))|\\(|\\)|\\(|\\)", course, maxsplit=1)
419
420         cssubject = values[1].upper()
421         cnumber = values[2]
422
423         content = self.response_request(cssubject, cnumber)
424
425         if not content:
426             dispatcher.utter_message(
427                 text=f"The course {course} does not exist or is not offered by {university}.")
428         else:
429             answer = f"The course {course} is of {content['credits']} credits."
430             dispatcher.utter_message(text=f"{answer}")
431
432     return []

```

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

The screenshot shows the Apache Jena Fuseki interface at the URL [/testData](#). The top navigation bar includes links for datasets, manage, and help, with a green server status indicator.

The main area is titled "SPARQL Query" and contains the following content:

Example Queries

Selection of triples (selected tab) | **Selection of classes**

SPARQL Endpoint: /testData/sparql

Content Type (SELECT): JSON

Content Type (GRAPH): Turtle

Prefixes: rdf, rdfs, owl, xsd

SPARQL Query:

```

1+ PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
2+ PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
3+ PREFIX foaf: <http://xmlns.com/foaf/0.1/>
4+ PREFIX tci: <http://LinkedScience.org/teach/ns#>
5+ PREFIX cu: <http://is-concordia.io/>
6
7 SELECT ?course ?links
8 WHERE {
9   ?course rdfs:seeAlso ?links;
10  cu:hasCourseSubject ?courseSubject;
11  cu:hasCourseNumber ?courseNumber .
12
13 FILTER(?courseSubject = "EDUC")
14 filter(?courseNumber = "301")
15 }

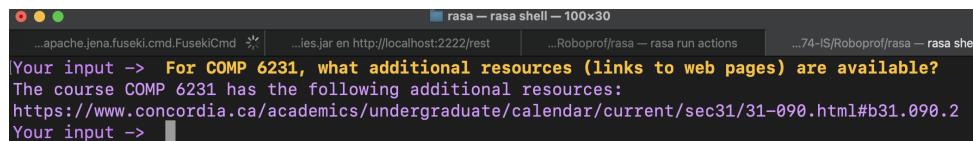
```

Table Response: 1 result in 0.03 seconds

course	links
1 <http://is-concordia.io/EDUC301>	https://www.concordia.ca/academics/undergraduate/calendar/current/sec31/31-090.html#b31.090.2

Simple view | Ellipse | Filter query results | Page size: 50 | ↻ | ?

Showing 1 to 1 of 1 entries



```

main.py          vocabulary.ttl      nlu.yml M      domain.yml M      stories.yml M      actions.py M
rasa > actions > actions.py ...
430 # 7. For [course] [number], what additional resources (links to web pages) are available?
431     class AdditionalResources(Action):
432         def name(self) -> Text:
433             return "action_about_additional_resources"
434
435         def response_request(self, csSubject, cNumber):
436             query = f"""
437                 SELECT ?Links
438                 WHERE {
439                     ?course rdf:type ?Links;
440                     cuhasCourseSubject ?courseSubject;
441                     cuhasCourseNumber ?courseNumber .
442
443                     FILTER(?courseSubject = "EDUC")
444                     FILTER(?courseNumber = "301")
445                 }
446             """
447             response = requests.post(SPARQL_ENDPOINT, data={"query": query})
448             # Use the json module to load CKAN's response into a dictionary.
449
450             results = response.json()
451             resources = []
452             for result in results["results"]["bindings"]:
453                 resources.append(result["link"]["value"])
454
455             return resources
456
457         def run(self, dispatcher: CollectingDispatcher,
458               tracker: Tracker,
459               domain: Dict[Text, Any]) -> List[Dict[Text, Any]]:
460             course = tracker.get_slot("course")
461             values = values(course)
462             course = values[0].upper().strip()
463             number = values[1].strip()
464
465             resources = self.response_request(subject=course, number=number)
466             if not resources:
467                 dispatcher.utter_message(text="The course {course} does not exist.")
468             else:
469                 answer = f"The course {course} has the following additional resources:\n"
470                 for resource in resources:
471                     answer += resource + "\n"
472                 dispatcher.utter_message(text=f"{answer}")
473
474             return []
475
476
477         def __init__(self) -> None:
478             super().__init__()
479             self.name = "AdditionalResources"
480             self.description = "Provides additional resources for a specific course."
481
482
483
484
485
486

```

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

The screenshot shows the Apache Jena Fuseki interface for a dataset named '/testDB'. The user is performing a SPARQL query to find content for a specific lecture. The query is:

```

PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX tci: <http://linkedscience.org/teach/ns#>
PREFIX cu: <http://is-concordia.io/>

SELECT ?course ?lecture ?content
WHERE {
    ?course rdfs:type cu:Course .
    ?course cuhasLecture ?lecture .
    ?lecture cuhasLectureContent ?content ;
        cuhasLectureNumber ?number
    FILTER(?course = cu:COMP6231)
    FILTER(?number = 3)
}

```

The results table shows two entries:

course	lecture	content
<http://is-concordia.io/COMP6231>	<http://is-concordia.io/COMP6231_Lecture#3>	<http://is-concordia.io/Users/nilesh/Work/COMP-474-IS/Roboprof/data/COMP6231/Slide/slide3.pdf>
<http://is-concordia.io/COMP6231>	<http://is-concordia.io/COMP6231_Lecture#3>	<http://is-concordia.io/Users/nilesh/Work/COMP-474-IS/Roboprof/data/COMP6231/Worksheet/worksheet3.pdf>

```

rasa -- rasa shell -- 100x30
...apache.jena.fuseki.cmd.FusekiCmd ... ...ies.jar on http://localhost:2222/rest ...Roboprof/rasa -- rasa run actions ...74-IS/Roboprof/rasa -- rasa shell +
Your input -> Detail the content (slides, worksheets, readings) available for lecture 3 in COMP 6231.
1.
The course COMP 6231 has the following contents:
http://localhost:3030/Users/nilesh/Work/COMP-474-IS/Roboprof/Dataset/COMP6231/Slide/slide3.pdf
http://localhost:3030/Users/nilesh/Work/COMP-474-IS/Roboprof/Dataset/COMP6231/Worksheet/worksheet3.pdf
Your input -> 

```

```

main.py
actions > actions.py > ...
489 # At - 8. Detail the content (slides, worksheets, readings) available for [lecture number] in [course]
490 class DetailContentForCourse(Action):
491     def name(self) -> Text:
492         return "action_about_detail_content"
493     def response_request(self, csubject, cnumber, lnumber):
494         query = """
495             PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
496             PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
497             PREFIX tci: <http://linkedscience.org/teach/ns#>
498             PREFIX cui: <http://is-concordia.io/>
499             SELECT ?contentURI
500             WHERE {
501                 cui:Concordia_University cuiters ?course .
502                 ?course cuhasTopic ?subject ;
503                     cuhasCourseNumber ?eventNumber ;
504                     cuhasLecture ?lecture .
505                 ?lecture cuhasLectureContent ?content ;
506                     cuhasLectureNumber ?lNumber .
507                 ?content rdfs:seeAlso ?contentURI .
508                 FILTER(?subject = ?csubject)
509                 FILTER(?number = ?cnumber)
510                 FILTER(?lNumber = ?lnumber)
511             }
512             """
513             response = requests.post(SPARQL_ENDPOINT, data={"query": query})
514             # Use the json module to load CKAN's response into a dictionary.
515             y = json.loads(response.text)
516             contents = []
517             for content in y["results"]["bindings"]:
518                 contents.append({"content": content["content"]["value"]})
519             return contents
520         def run(self, dispatcher: CollectingDispatcher,
521                 tracker: Tracker,
522                 domain: Dict[Text, Any]) -> List[Dict[Text, Any]]:
523             courseEvent = tracker.get_slot("lecture").strip()
524             courseEventValue = re.split(r"([^\d]\d)(\d)", courseEvent, maxsplit=1)[0]
525             eventNumber = courseEventValue[2].strip()
526             contents = self.response_request(subject, cnumber, eventNumber)
527             if not contents:
528                 dispatcher.utter_message(text="The course (course) does not exist.")
529             else:
530                 answer = f"The course {course} has the following contents:\n"
531                 for content in contents:
532                     answer += content["content"] + "\n"
533             dispatcher.utter_message(text=f"{answer}")
534             return []

```

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

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

```

...apache.jena.fuseki.cmd.FusekiCmd % ...les.jar en http://localhost:2222/rest ...Roboprof/rasa - rasa run actions ...74-SRoboprof/rasa - rasa shell +
Your input -> What reading materials are recommended for studying Applied Artificial Intelligence in COMP 6231
The course COMP 6231 has the following materials 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.pdf
df
http://localhost:3030/Users/nilesh/Work/COMP-474-IS/Roboprof/Dataset/COMP6231/Worksheet/worksheet1.pdf
df
http://localhost:3030/Users/nilesh/Work/COMP-474-IS/Roboprof/Dataset/COMP6231/Worksheet/worksheet7.pdf
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.pdf
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.pdf
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.pdf
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.pdf
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.pdf

```

```

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     def __init__(self, csubject, topic):
560         self.csubject = csubject
561         self.topic = topic
562     def name(self) -> Text:
563         return "action_about_reading_material"
564     def response(self, csubject, topic):
565         query = """
566             PREFIX foaf: <http://xmlns.com/foaf/0.1/>
567             PREFIX tci: <http://linkedscience.org/teach/ns#>
568             PREFIX cu: <http://is-concordia.io/>
569             SELECT distinct ?materials
570             WHERE {
571                 ?course cu:hasTopic ?topics ;
572                 cu:hasCourseSubject ?courseSubject; cu:hasCourseNumber ?courseNumber .
573                 ?topics foaf:name ?topicName ;
574                 cu:hasMaterials ?materials .
575                 FILTER(?courseSubject = "<"+csubject+">")
576                 FILTER(?topicName = "<"+topic+">")
577             }
578             response = requests.post(SPARQL_ENDPOINT, data={"query": query})
579             # Use the json module to load CKAN's response into a dictionary.
580         
```

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

The screenshot shows a SPARQL query interface with the following details:

- SPARQL Query:** To try out some SPARQL queries against the selected dataset, enter your query here.
- Example Queries:** Selection of triples (selected), Selection of classes.
- SPARQL Endpoint:** /testDB/query
- Content Type (SELECT):** JSON
- Content Type (GRAPH):** Turtle
- Query:**

```

1 PREFIX foaf: <http://xmlns.com/foaf/0.1/>
2 PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
3 PREFIX cu: <http://is-concordia.io/>
4 PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
5
6 SELECT ?topicName ?topicLink
7 WHERE {
8   ?course a cu:Course .
9   ?course cu:hasTopic ?topic .
10  ?topic foaf:name ?topicName ;
11    rdfs:seeAlso ?topicLink .
12  ?course cu:hasCourseSubject ?subject .
13  ?course cu:hasCourseNumber ?number .
14  FILTER(?subject = "COMP")
15  FILTER(?number = "6231")
16 }

```
- Results:**

topicName	topicLink
Distributed_algorithm	<http://dbpedia.org/resource/Distributed_algorithm>
Inter-process_communication	<http://dbpedia.org/resource/Inter-process_communication>
Grid_computing	<http://dbpedia.org/resource/Grid_computing>
Peer-to-peer	<http://dbpedia.org/resource/Peer-to-peer>
Congestion (database systems)	<http://dbpedia.org/resource/Congestion_(database_systems)>

```

rasa > actions > actions.py > CompetenciesOfStudent(Action):
    def name(self):
        return "action_about_competencies_of_student"
    def response_request(self, csubject, cnumber):
        query = """
PREFIX dbp: <http://dbpedia.org/resource/>
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 ;
           dbp:topicLink ?topicLink .
    ?course cu:hasCourseSubject ?subject .
    ?course cu:hasCourseNumber ?number .
    FILTER(?subject = "(csubject)") .
    FILTER(?number = "(cnumber)") .
}
        """
        response = requests.post(SPARQL_ENDPOINT, data={'query': query})
        # Use the json module to load JSON response into a dictionary.
        y = json.loads(response.text)
        # The results is a Python dictionary.
        results = y['results']['bindings']
        competencies = []
        bindings = results['bindings']
        for competency in bindings:
            competencyValue = competency['topicName']['value']
            linkValue = competency['topicLink']['value']
            competencies.append((topicName: competencyValue, topicLink: linkValue))
        return competencies
    def run(self, dispatcher: CollectingDispatcher,
           tracker: Tracker,
           domain: Dict[Text, Any]) -> List[Dict[Text, Any]]:
        course = tracker.get_slot('course')
        values = valuesplit(course)
        course = values[0].strip()
        maxsplit=1
        subject = values[1].upper().strip()
        number = values[2].strip()
        competencies = self.response_request(csubject, cnumber)
        if not competencies:
            dispatcher.utter_message(text="The course (course) does not exist.")
        else:
            answer = f"The student will gain the following competencies by completing the {course} course:\n"
            for competency in competencies:
                answer = answer + competency['topicName'] + ": " + competency['topicLink'] + "\n"
        dispatcher.utter_message(text=f"(answer)")
        return []

```

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

studentName	course	gradeValue
1 Vidhi	<http://is-concordia.io/COMP6741>	A
2 John	<http://is-concordia.io/COMP6741>	A
3 Nilesh	<http://is-concordia.io/COMP6741>	C

```

...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 grades did 40229633 achieve in COMP 6231?
The student with ID 40229633 has the following grades in COMP 6231 course:
Nilesh: A
Your input ->

```

```

# main.py          E vocabulary.ttl   E muymt.M   E domainymt.M   E storiesyml.M   E actions.py.M
rasa > actions > E actions.py > ...
675  class GradesOfStudent(Action):
676      def __init__(self, cnumber, crumber, student):
677          self.cnumber = cnumber
678          self.crumber = crumber
679          self.student = student
680
681      def name(self) -> Text:
682          return "action_about_student_grades"
683
684      def response_request(self, subject, cnumber, student):
685          query = """
686              PREFIX foaf: <http://xmlns.com/foaf/0.1/>
687              PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
688              PREFIX cui: <http://is-concordia.io/>
689              PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
690
691              SELECT ?studentName ?gradeValue
692                  WHERE {
693                      ?student cui:hasCourseSubject ?course ;
694                          cui:hasStudentID ?studentID ;
695                          cui:hasCourseNumber ?cnumber ;
696                          cui:hasCourseSubject ?subject ;
697                          cui:hasCourseNumber ?number .
698
699                      ?student ?course ?gradeValue
700                      FILTER(?subject = ?csubject)
701                      FILTER(?number = ?cnumber)
702                      FILTER(?studentID = ?student)
703
704                      FILTER(?studentName = ?name)
705                      FILTER(?gradeValue = ?value)
706
707                      ""
708
709                      response = requests.post(SPARQL_ENDPOINT, data={'query': query})
710
711                      # Use the json module to load (KAN's response into a dictionary.
712
713                  }
714
715          
```

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

The screenshot shows the Apache Jena Fuseki interface with a dataset named 'testDB'. The 'SPARQL Query' section contains the following SPARQL query:

```

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

SELECT ?studentName ?course ?gradeValue
WHERE {
    ?student cui:hasRolledIn ?course ;
        cui:hasName ?studentName .
    ?course cui:hasCourseSubject ?subject ;
        cui:hasCourseNumber ?number .
    ?student ?course ?gradeValue
    FILTER(?subject = "COMP")
    FILTER(?number = "6741")
}

```

The results table shows three entries:

studentName	course	gradeValue
Vidhi	<http://is-concordia.io/COMP6741>	A
John	<http://is-concordia.io/COMP6741>	A
Nilesh	<http://is-concordia.io/COMP6741>	C

```

...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 students have completed COMP 6741?
The following students have completed the COMP 6741 course:
Vidhi: 40232374
Nilesh: 40229633
Your input ->

```

```

    #!/usr/bin/env python
    # encoding: utf-8
    import requests
    from rdflib import Graph, URIRef, Literal
    from rdflib.namespace import FOAF, RDF, RDFS, OWL, XSD
    from SPARQLWrapper import SPARQLWrapper, JSON
    from collections import defaultdict
    import json

    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 DISTINCT ?id ?studentName
    WHERE {
      ?student cu:isEnrolledIn ?course .
      ?course rdfs:label ?courseName ;
              cu:hasStudentID ?id .
      ?course cu:hasCourseSubject ?subject ;
              cu:hasCourseNumber ?number .
      FILTER(?subject = "({subject})")
      FILTER(?number = "({number})")
    }
  
```

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

name	id	email	courseTitle	grade
1 Nilesh	40229633	nilesh.suryawanshi@mail.concordia.ca	Distributed Systems	A
2 Nilesh	40229633	nilesh.suryawanshi@mail.concordia.ca	Intelligent Systems	C

```

Your input -> Print a transcript for a 40229633
The transcript for the student with ID 40229633
40229633      Distributed Systems      A
40229633      Intelligent Systems     C
40229633      Intelligent Systems     B
Your input ->
  
```

```

main.py      vocabulary.ttl      nlu.yml M      domain.yml M      stories.yml M      actions.py M X
rasa > actions > actions.py > ...
800     class StudentTranscript(Action):
801
802         def name(self) -> Text:
803             return "action_about_student_transcript"
804
805         def response(self, student):
806             query = """
807                 PREFIX foaf: <http://xmlns.com/foaf/0.1/>
808                 PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
809                 PREFIX teach: <http://linkedscience.org/teach/ns#>
810                 PREFIX cu: <http://is-concordia.io/>
811
812                 SELECT ?name ?id ?courseTitle ?grade
813                 WHERE
814                     ?student cu:isEnrolledIn ?course .
815                     ?course teach:courseTitle ?courseTitle .
816                     ?student foaf:fname ?name
817                     ?student cu:hasStudentID ?id .
818                     ?student ?course ?grade
819                     FILTER(?id = "{student}")
820                 """
821
822             response = requests.post(SPARQL_ENDPOINT, data=query)
823             # 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 X
rasa > actions > actions.py > ...
804         class StudentTranscript(Action):
805
806             def response(self, student):
807                 query = """
808                     PREFIX foaf: <http://xmlns.com/foaf/0.1/>
809                     PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
810                     PREFIX teach: <http://linkedscience.org/teach/ns#>
811                     PREFIX cu: <http://is-concordia.io/>
812
813                     SELECT ?name ?id ?courseTitle ?grade
814                     WHERE
815                         ?student cu:isEnrolledIn ?course .
816                         ?course teach:courseTitle ?courseTitle .
817                         ?student foaf:fname ?name
818                         ?student cu:hasStudentID ?id .
819                         ?student ?course ?grade
820                         FILTER(?id = "{student}")
821
822                     response = requests.post(SPARQL_ENDPOINT, data=query)
823                     # Use the json module to load CKAN's response into a dictionary.
824
825                     # Load the JSON response into a Python dictionary.
826                     results = y["results"]
827                     grades = []
828                     for result in results["bindings"]:
829                         studentId = result["student"]["value"]
830                         courseTitle = result["courseTitle"]["value"]
831                         gradeValue = result["grade"]["value"]
832                         grades.append((studentId, {"courseTitle": courseTitle, "gradeValue": gradeValue}))
833
834                     return grades
835
836             def run(self, dispatcher: CollectingDispatcher,
837                   tracker: Tracker,
838                   domain: Dict[Text, Any]) -> List[Dict[Text, Any]]:
839                 student = tracker.slots["studentId"]
840                 grades = self.response(student)
841                 if not grades:
842                     dispatcher.utter_message(
843                         f"Sorry, the student {student} does not exist.")
844                 else:
845                     answer = f"\nTranscript for the student with ID {student}. \n"
846                     for grade in grades:
847                         answer += grade["studentId"] + "\t" + grade["courseTitle"] + "\t" + grade["gradeValue"] + "\n"
848
849             dispatcher.utter_message(text=answer)
850
851             return []

```

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

```

main.py      vocabulary.ttl      nlu.yml M      domain.yml M      stories.yml M      actions.py M X
rasa > actions > actions.py > ...
1019     # A2-02 What is course [course] about?
1020     class CourseDescription(Action):
1021
1022         def name(self) -> Text:
1023             return "action_about_course"
1024
1025         def run(self, dispatcher: CollectingDispatcher,
1026               tracker: Tracker,
1027               domain: Dict[Text, Any]) -> List[Dict[Text, Any]]:
1028             course = tracker.slots['course']
1029             print(course)
1030             values = re.split(r'(\d+)(\.\d+)', course, maxsplit=1)
1031
1032             csubject = values[1].upper()
1033             cnumber = values[2]
1034             print(csubject)
1035             query = f"""
1036                 PREFIX cu: <http://is-concordia.io/>
1037                 PREFIX teach: <http://linkedscience.org/teach/ns#>
1038                 SELECT ?courseDescription
1039                 WHERE {
1040                     cu:Concordia_University cu:Offers ?course .
1041                     ?course teach:courseDescription ?courseDescription ;
1042                         teach:courseTitle ?courseName ;
1043                         cu:hasCourseSubject ?courseSubject ;
1044                         cu:hasCourseNumber ?courseNumber .
1045                     filter(?courseSubject = "{csubject}")
1046                     filter(?courseNumber = "{cnumber}")
1047                 }
1048             """
1049             data = {'query': query}
1050             response = requests.post(SPARQL_ENDPOINT,
1051                                     data=data)
1052
1053             y = json.loads(response.text)
1054             print(y)
1055             results = y["results"]
1056             print(results)
1057             if not results or not results["bindings"]:
1058                 dispatcher.utter_message(
1059                     f"Sorry, the course {course} not found!")
1060
1061             dispatcher.utter_message(text=f"The {course} has the following overview in my database: \n")
1062             dispatcher.utter_message(text=f"{results['bindings'][0]['courseDescription']}")
1063
1064             return []

```

```

rasa > actions > actions.py > ...
Your input => What is COMP 474 about?
00:00:00.000 [INFO] Rasa: Initialization for COMP474(Rule-based expert systems, blackboard architecture, and agent-based, Knowledge acquisition and representation, Uncertainty and conflict resolution, Reasoning and planning, Design of intelligent systems).
Your input => []

```

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

```

rasa > rasa shell - 100x30
...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 X
rasa > actions > actions.py > ActionTopicsCovered > run
855 | #which topics are covered in <course events>?
856 | # A2-Q1. For a course c, list all covered topics t, printing out their English labels and their DBpedia,
857 | # together with the course event URI (e.g., 'lab3') and resource URI (e.g., 'slides10') where they appear
858 class ActionTopicsCovered(Action):
859
860     def name(self) -> Text:
861         return "action_bbout_topics_covered"
862
863     def response_request(self, courseSubject, courseNumber, eventNumber, event):
864
865         if event.lower() == "lab" or event.lower() == "laboratory":
866             query = """
867             PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
868             PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
869             PREFIX foaf: <http://xmlns.com/foaf/0.1/>
870             PREFIX tc: <http://linkedservice.org/teach/ns#>
871             PREFIX cu: <http://is-concordia.io/>
872
873             SELECT ?topicName ?topics
874             WHERE {
875                 cu:Concordia_University cu:Offers ?course .
876                 ?course tc:courseTitle ?title ;
877                     cu:hasLecture ?lecture ;
878                     cu:hasCourseSubject ?courseSubject ;
879                     cu:hasCourseNumber ?courseNumber .
880                 ?lecture cu:topicsCovered ?topics ;
881                     cu:hasLectureNumber ?lectureNumber .
882                 ?topics foaf:name ?topicName .
883                 FILTER(?courseSubject = "(courseSubject")"
884                 FILTER(?lectureNumber = "(eventNumber")
885                 FILTER(?courseNumber = "(courseNumber")
886             }
887             """
888         else:
889             query = """
890             PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
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://linkedservice.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 ;

```

```

main.py   vocabulary.ttl   ! nluyml M   ! domain.yml M   ! stories.yml M   actions.py M X
rasa > actions > actions.py > ActionTopicsCovered > run
858     class ActionTopicsCovered(Action):
859         def response_request(self, courseSubject, courseNumber, eventNumber, event):
860             """
861             SELECT ?topicName ?topics
862             WHERE {
863                 cu:Concordia_University cu:Offers ?course .
864                 ?course tc:courseTitle ?title ;
865                 cu:hasLab ?lab ;
866                 cu:hasCourseSubject ?courseSubject ;
867                 cu:hasCourseNumber ?courseNumber .
868                 rlab cu:topicsCovered ?topics ;
869                 cu:hasLabNumber ?labNumber .
870                 ?topics foaf:name ?topicName .
871                 FILTER(?courseSubject = "{courseSubject}")
872                 FILTER(?labNumber = {eventNumber})
873                 FILTER(?courseNumber = "{courseNumber}")
874             }
875             """
876
877             response = requests.post(SPARQL_ENDPOINT,
878                                     data={'query': query})
879             # Use the json module to load CKAN's response into a dictionary.
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914

```

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

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 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   ! nluyml M   ! domain.yml M   ! stories.yml M   actions.py M X
rasa > actions > actions.py > ActionTopicsCovered > run
858     class ActionTopicsCovered(Action):
859         def response_request(self, courseSubject, courseNumber, eventNumber, event):
860             topics = []
861
862             for result in results["bindings"]:
863                 topicName = result["topicName"]
864                 topic = result["topics"]
865
866                 topicNameValue = topicName["value"]
867                 topicValue = topic["value"]
868
869                 row = ("topicName", topicNameValue, "topic": topicValue)
870                 topics.append(row)
871
872
873             return topics
874
875
876             def runself, dispatcher: CollectingDispatcher,
877                 tracker: Tracker,
878                 domain: Dict[Text, Any]) -> List[Dict[Text, Any]]:
879                 print(tracker.slots)
880                 course = tracker.slots['course'].replace(" ", "")
881
882                 values = re.split(r"([^\d]*)(\d+)", course, maxsplit=1)
883
884                 csubject = values[1].upper()
885                 cnumber = values[2]
886
887                 event = tracker.slots['courseEvent'].strip()
888                 event = event.split(split)
889                 r'([^\d]*)(\d+)', event, maxsplit=1)
890                 eventNumber = eventValues[2]
891                 event = eventValues[1]
892                 topics = self.response_request(csubject, cnumber, eventNumber, event)
893
894
895                 if not topics:
896                     dispatcher.utter_message(
897                         text="# No topics covered in {eventNumber} or {eventNumber} did not happen.")
898                 else:
899                     answer = f"The following topics were covered in the lecture {eventNumber} of the course {csubject}{cnumber}:\n"
900                     for topic in topics:
901                         answer = answer + topic['topicName'] + '\n'
902                         ". " + topic['topic'] + "\n"
903                     dispatcher.utter_message(text=(answer))
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964

```

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>