Dheya El Hak Chiha
23720753

User Guide:

Files Submitted Explained

I have submitted a zip containing:
- ontology.py → python file generating the ontology and the reasoner (rules, axioms)
- parse.py → python file to parse a folder named 'TestData' containing json files (each json files represents a procedure) into the ontology
- TestData → folder containing 285 json files (each represents a procedure)
- queriesSparql.py → python file that converts the ontology into a knowledge graph and allows for querying (contains 4 queries)
- app.py → python file which represents the flask application
- templates → folder containing html files
- static → folder containing css file
- report.pdf →pdf file containing report
- UserGuide → ..

Dheya El Hak Chiha
23720753

Dependency

-rdflib
-java
- owlready2
-flask

Ontology Overview

The ontology.py script defines the core structure of the ontology used to model repair procedures for the iFixit dataset. The ontology captures entities such as procedures, items, steps, tools, and more, and defines relationships between these entities to enable querying and reasoning over the knowledge graph.

This script makes use of the OWLReady2 Python library to define an ontology and its classes, properties, and relationships. The resulting ontology is saved as an OWL file (cars_trucks_ontology.owl).

Classes Defined in the Ontology

1. Procedure
Represents a repair procedure.
Example: "Replacing the battery in a phone" is a procedure.

2. Step
Represents an individual step in a procedure.
Example: "Remove the back cover" is a step in a procedure.

3. Tool
Represents the tools used in a procedure or step.
Example: "Phillips #00 Screwdriver" is a tool.

4. Item
Represents the physical object in question involved in the procedure, such as car, autopart or accessory.

5. Part
Represents an autopart

6. Image
Represents images

7. Model
Represents the model of a product for which a procedure is created.

8. Make
Represents the manufacturer or make of a product.

Subclasses of Image

9. ToolImage
Represents images associated with tools (thumbnails).

10. StepImage
Represents images illustrating steps in a procedure.

Object Properties

requires_tool (Procedure → Tool)
This property links a Procedure to the Tool that is required to perform it. This property essentially forms the "toolbox" of a procedure.

used_in (Tool → Procedure)
The inverse of requires_tool. This property links a Tool to the Procedure in which it is used. It ensures a bidirectional relationship between tools and procedures.

requiresTool (Step → Tool)
Links individual Steps in a procedure to the Tools needed to complete them. This helps track which tool is needed for each specific action in the procedure.

isRequiredFor (Tool → Step)
The inverse of requiresTool. This links a Tool to the Step that requires it.

has_part (Item → Item)
Represents a transitive, hierarchical relationship between Items, where one item can be a part of another item.

part_of (Item → Item)
The inverse of has_part. This property links an item to the larger item it is part of.

has_sub_procedure (Procedure → Procedure)
Links a Procedure to its Sub-procedures. This relationship is transitive (if procedure A has sub-procedure B, and B has sub-procedure C, then A also has C as a sub-procedure) and doesnt allow for symmetry

sub_procedure_for (Procedure → Item)
This property links a Procedure to the Item it is performed for (inverse of has_sub_procedure)

has_make (Procedure → Make)
Links a Procedure to the Make of the item it applies to.

has_model (Procedure → Model)
Links a Procedure to the Model of the item.

is_for_make (Make → Procedure)
The inverse of has_make. This property links a Make to the Procedure it is related to.

is_for_model (Model → Procedure)
The inverse of has_model. This property links a Model to the Procedure it is associated with.

has_model_for_make (Make → Model)
Links a Make to its associated Model.

is_model_of (Model → Make)
The inverse of has_model_for_make. This property links a Model to its Make.

hasProcedure (Item → Procedure)
Links an Item to the Procedure that is used for repairing.

ProcedureForThis (Procedure → Item)
The inverse of hasProcedure. This property links a Procedure to the Item it is for.

has_Item (Procedure → Item)
Links a Procedure to the Item involved in the procedure.

has_image_for_step (Step → StepImage)
Links a Step to an associated Image that illustrates or explains that step.

has_image (Tool → ToolImage)
Links a Tool to its associated Image.

Data Properties

has_guideID (Procedure → string)
Links a Procedure to its unique guide identifier

guideTitle (Procedure → string)
Links a Procedure to its title.

Dheya El Hak Chiha
23720753

stepOrder (Step → int)
Specifies the order of a Step within a Procedure.

toolURL (Tool → string)
Stores the URL for a Tool, such as a link to where the tool can be purchased or viewed.

toolImageURL (ToolImage → string)
Stores the URL of an image for a tool.

stepText (Step → string)
Stores the instructional text for a Step.

has_step (Procedure → Step)
Links a Procedure to its Steps.

Reasoning

Axioms:

```
# 1. A `Procedure` must have at least one `Step`
Procedure.equivalent_to.append(Procedure & has_step.some(Step))

# 2. Any Procedure that has at least one Step that requires a Tool must itself require that Tool
Procedure.equivalent_to.append(Procedure & has_step.some(Step & requiresTool.some(Tool)) &
requires_tool.some(Tool))


# 3. A `Step` requiring a `Tool` must have that tool in the `Toolbox`
Step.equivalent_to.append(Step & requiresTool.some(Tool) & isRequiredFor.some(requires_tool.some(Tool)))

# 4. A sub-procedure of a Procedure must involve a part or the same item as the parent Procedure
Procedure.equivalent_to.append(Procedure & has_sub_procedure.some(Procedure & sub_procedure_for.some(Item &
has_part.some(Item))))

# 5. A Tool Must Be Used in At Least One Procedure
Tool.equivalent_to.append(Tool & used_in.some(Procedure))
```

SWRL rules:

```
  # Rule 1: If a Step Requires a Tool, That Tool Must Be in the Procedure's Toolbox
  rule1 = Imp()
  rule1.set_as_rule("""
  Step(?s), Procedure(?p), Tool(?t), has_step(?p, ?s), requiresTool(?s, ?t) -> requires_tool(?p, ?t)
  """)

  # Rule 2: If a Procedure Has a Sub-Procedure, Both Must Be for the Same Item or a Part of the Same Item
  rule3 = Imp()
  rule3.set_as_rule("""
    Procedure(?p1), Procedure(?p2), has_sub_procedure(?p1, ?p2),
    sub_procedure_for(?p1, ?item1), sub_procedure_for(?p2, ?item2), part_of(?item2, ?item1) -> sub_procedure_for(?p2,
?item2)
    """)

    # Rule 3: If a Procedure Requires a Tool, That Tool Must Be Used in One of Its Steps
    rule4 = Imp()
    rule4.set_as_rule("""
    Procedure(?p), Tool(?t), requires_tool(?p, ?t) -> has_step(?p, ?s), requiresTool(?s, ?t)
    """)

    # Rule 4: If a Step Is Part of a Procedure, It Must Have a Step Order
    rule5 = Imp()
    rule5.set_as_rule("""
    Step(?s), Procedure(?p), has_step(?p, ?s) -> stepOrder(?s, ?o)
    """)
```

Parsing into the ontology

- First run the ‘python3 ontology.py’ to initialise the ontology (creates ‘cars_trucks_ontology.owl’)
- Run ‘python3 parse.py’ to parse the json files (located in TestData) into the ontology, this will then create and updated ontology file called ‘updated_cars_trucks_ontology.owl’

The ‘updated_cars_trucks_ontology.owl’ will now contain the ontology with the instances and inferred information

Querying the Knowledge Graph

To convert the ontology into the knowledge graph for querying, run ‘queriesSparql.py’. The output will be the output for the 4 queries.

Query Interpretation

Find all procedures with more than 6 steps:

```
"""
PREFIX ex: <http://example.org/ifixit.owl#>
SELECT ?procedure
WHERE {
    ?procedure a ex:Procedure .
    ?procedure ex:has_step ?step .
}
```

```
GROUP BY ?procedure
HAVING (COUNT(?step) > 6)
"""
```

The output will list the URIs of all procedures that have more than six associated steps. Each URI represents a procedure in the knowledge graph.

Find all items that have more than 10 procedures written for them:

```
"""
PREFIX ex: <http://example.org/ifixit.owl#>
SELECT ?item
WHERE {
   ?procedure a ex:Procedure .
   ?procedure ex:has_Item ?item .
}
GROUP BY ?item
HAVING (COUNT(?procedure) > 10)
"""
```

The output will provide a list of items that have more than ten procedures linked to them.

Find all procedures that include a tool that is never mentioned in the procedure steps:

```
"""

PREFIX ex: <http://example.org/ifixit.owl#>
SELECT ?procedure ?tool
WHERE {
   ?procedure a ex:Procedure .
   ?procedure ex:requires_tool ?tool .

   # Find tools required by the procedure
   FILTER NOT EXISTS {
      # Match steps within the procedure
      ?procedure ex:has_step ?step .
      # Match tools used in those steps
      ?step ex:requiresTool ?tool .
   }
}
"""
```

The output will consist of pairs of procedures and tools. For each pair, it shows which tool is required for a procedure but is not referenced in the steps.

Flag potential hazards in the procedure by identifying steps with works like "careful" and "dangerous".:

```
"""
PREFIX ex: <http://example.org/ifixit.owl#>
SELECT ?procedure ?step ?text WHERE {
  ?procedure ex:has_step ?step .
  ?step ex:stepText ?text .
  FILTER (CONTAINS(LCASE(?text), "careful") || CONTAINS(LCASE(?text), "dangerous"))
}
"""
```

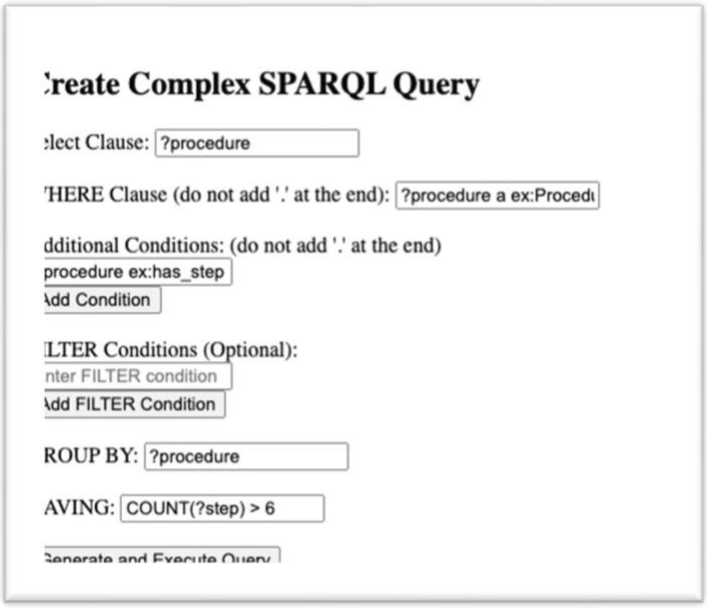
The output will include the procedures, steps, and the associated text that contains the keywords 'dangerous' and 'careful'.

Flask Application

Functionality

Here's how you can effectively use the application:

1. **Search Dataset:**
   - Functionality: Users can enter a SPARQL query in the search field. SPARQL (SPARQL Protocol and RDF Query Language) is used to retrieve data stored in the ontology.
   - Allows for customizable queries (don't add '.' at the end for conditons + additional condition)



:reate Complex SPARQL Query

:lect Clause: ?procedure

'HERE Clause (do not add '.' at the end): ?procedure a ex:Procedı

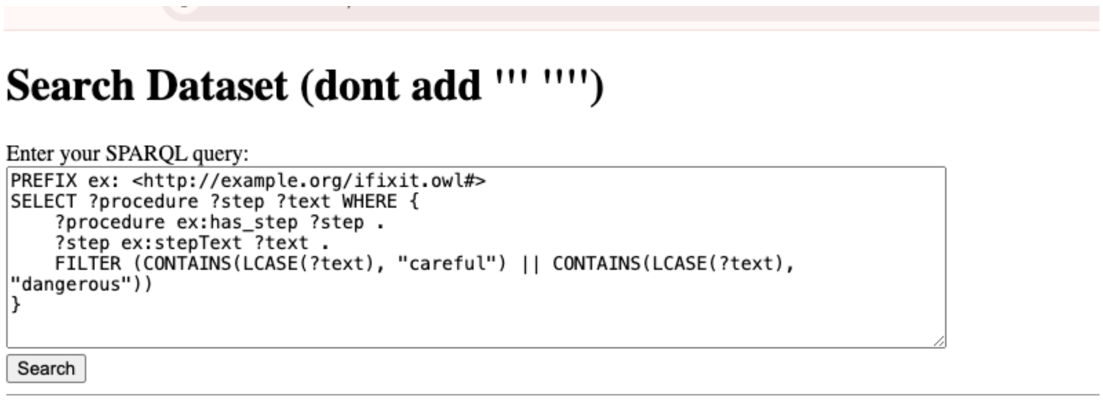
dditional Conditions: (do not add '.' at the end)
procedure ex:has_step
\dd Condition

ILTER Conditions (Optional):
nter FILTER condition
\dd FILTER Condition

ROUP BY: ?procedure

AVING: COUNT(?step) > 6

- How to Use:
  - Type your SPARQL query directly into the search box/ customise your query/ use predefined query.
  - Click the "Search" button to execute the query.
  - The application will display results based on your query, allowing you to find specific information or data patterns within the ontology.



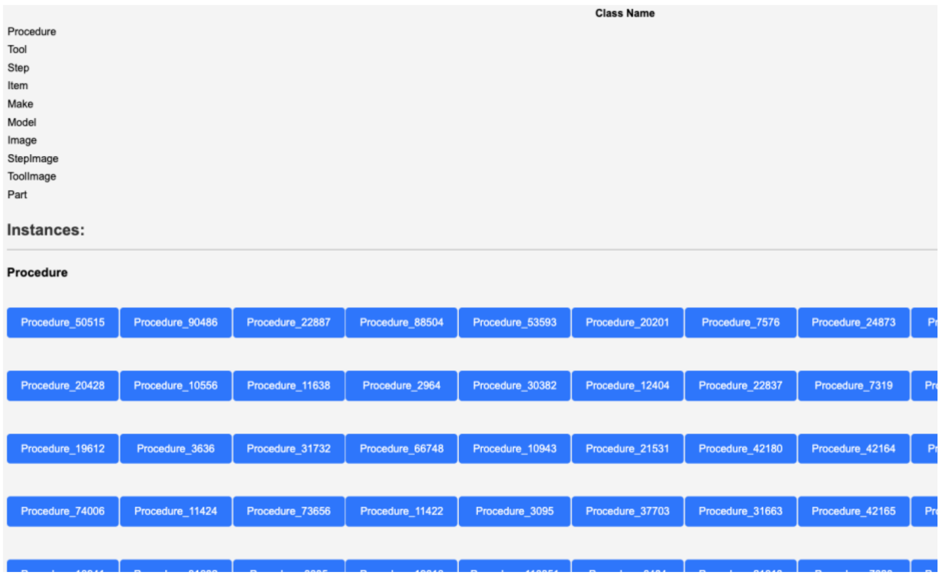
Search Dataset (dont add ''' '''')

Enter your SPARQL query:

```
PREFIX ex: <http://example.org/ifixit.owl#>
SELECT ?procedure ?step ?text WHERE {
    ?procedure ex:has_step ?step .
    ?step ex:stepText ?text .
    FILTER (CONTAINS(LCASE(?text), "careful") || CONTAINS(LCASE(?text),
"dangerous"))
}
```

2. Display Data:

- Functionality: This feature allows users to view all classes and instances defined in the ontology.
- How to Use:
  - Navigate to the relevant section of the application to see a structured list or hierarchy of classes and instances.
  - Can navigate the 'graph' by clicking on instances and viewing their properties and relations
  - This view helps users understand the structure of the ontology and the relationships between different entities.



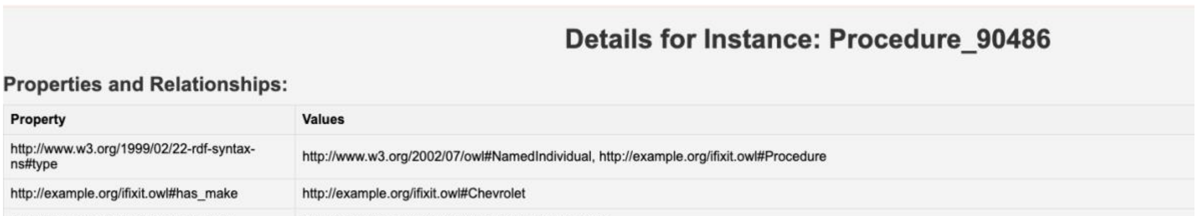
Class Name

Procedure
Tool
Step
Item
Make
Model
Image
StepImage
ToolImage
Part

Instances:

Procedure

Procedure_50515	Procedure_90486	Procedure_22887	Procedure_88504	Procedure_53593	Procedure_20201	Procedure_7576	Procedure_24873
Procedure_20428	Procedure_10556	Procedure_11638	Procedure_2964	Procedure_30382	Procedure_12404	Procedure_22837	Procedure_7319
Procedure_19612	Procedure_3636	Procedure_31732	Procedure_66748	Procedure_10943	Procedure_21531	Procedure_42180	Procedure_42164
Procedure_74006	Procedure_11424	Procedure_73656	Procedure_11422	Procedure_3095	Procedure_37703	Procedure_31663	Procedure_42165



Details for Instance: Procedure_90486

Properties and Relationships:

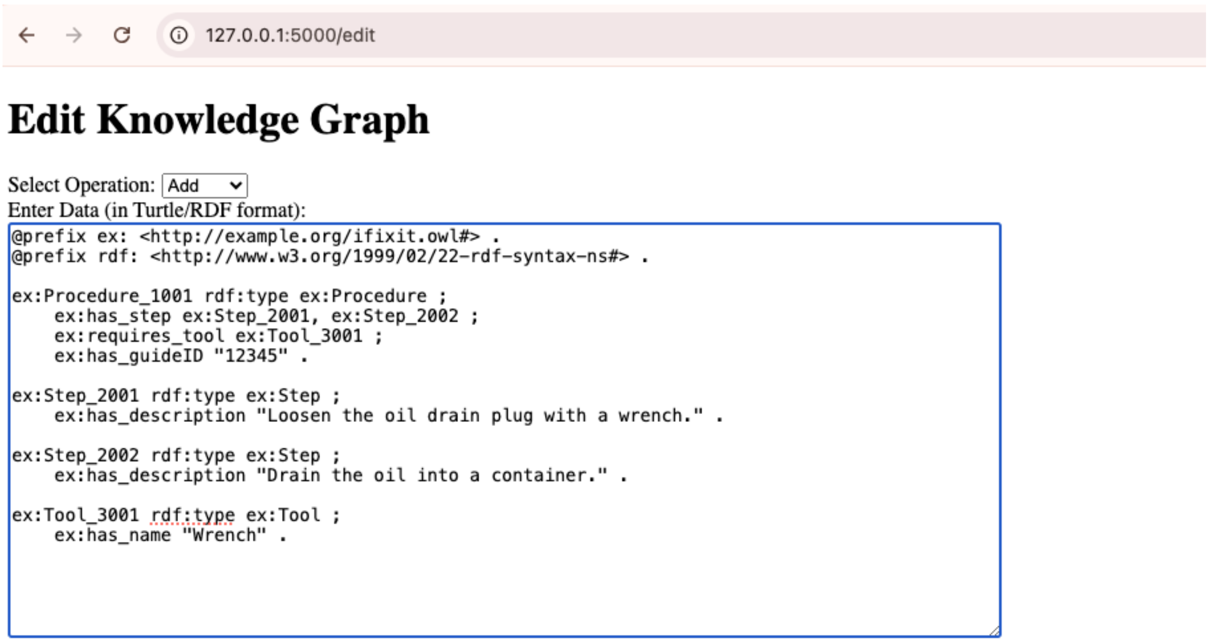
Property	Values
http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://www.w3.org/2002/07/owl#NamedIndividual, http://example.org/ifixit.owl#Procedure
http://example.org/ifixit.owl#has_make	http://example.org/ifixit.owl#Chevrolet

Dheya El Hak Chiha
23720753

3. **SPARQL Queries:**
   - Functionality: The application provides predefined SPARQL queries.
4. **Add/Edit/Update functionality**
   - Allows for adding, editing, and updating data to the ontology
   - It modifies the ontology without any wait
   - Note: you must restart the flask app (shaky sometimes, might have to restart 2) to view the new/modified instance in the display page
   - Handles and identifies any errors in the syntax and most importantly the data (in respect with rules/ ontology structure)

Tutorial

1) Write the necessary data in rdf format



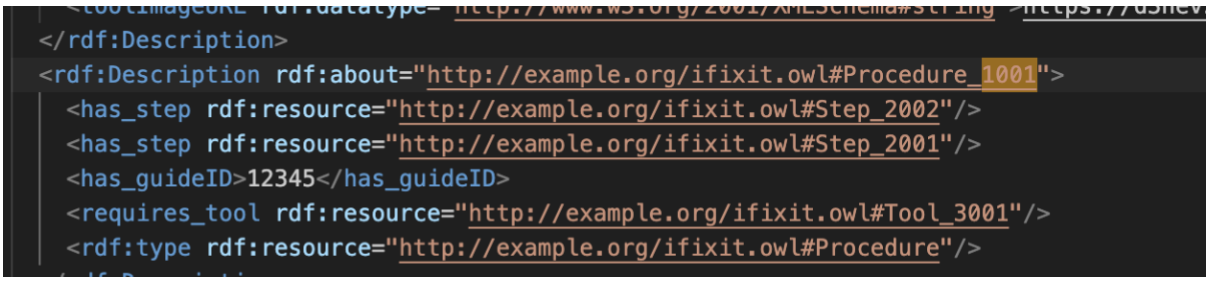
127.0.0.1:5000/edit

Edit Knowledge Graph

Select Operation: Add
Enter Data (in Turtle/RDF format):

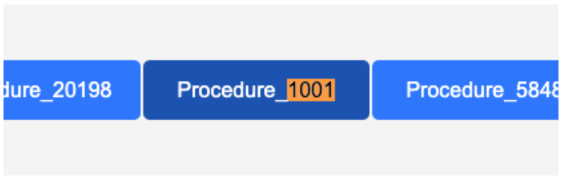
```
@prefix ex: <http://example.org/ifixit.owl#> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .

ex:Procedure_1001 rdf:type ex:Procedure ;
    ex:has_step ex:Step_2001, ex:Step_2002 ;
    ex:requires_tool ex:Tool_3001 ;
    ex:has_guideID "12345" .

ex:Step_2001 rdf:type ex:Step ;
    ex:has_description "Loosen the oil drain plug with a wrench." .

ex:Step_2002 rdf:type ex:Step ;
    ex:has_description "Drain the oil into a container." .

ex:Tool_3001 rdf:type ex:Tool ;
    ex:has_name "Wrench" .
```

2) The ontology is modified straight away

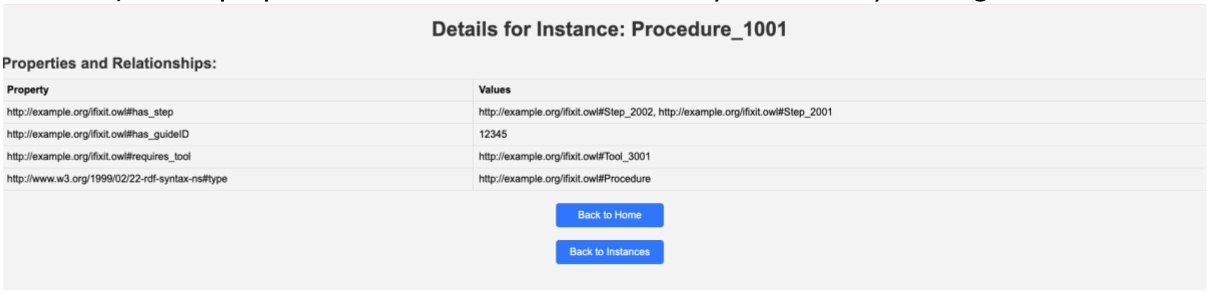


```
</rdf:Description>
<rdf:Description rdf:about="http://example.org/ifixit.owl#Procedure_1001">
  <has_step rdf:resource="http://example.org/ifixit.owl#Step_2002"/>
  <has_step rdf:resource="http://example.org/ifixit.owl#Step_2001"/>
  <has_guideID>12345</has_guideID>
  <requires_tool rdf:resource="http://example.org/ifixit.owl#Tool_3001"/>
  <rdf:type rdf:resource="http://example.org/ifixit.owl#Procedure"/>
```

3) Restart the flask app to view the new instance in display.html



4) View properties and extra information that you added by clicking on the instance



Details for Instance: Procedure_1001

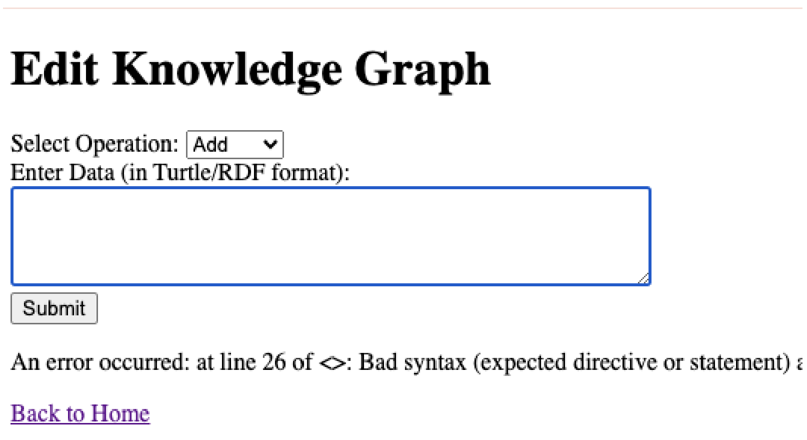
Properties and Relationships:

Property	Values
http://example.org/ifixit.owl#has_step	http://example.org/ifixit.owl#Step_2002, http://example.org/ifixit.owl#Step_2001
http://example.org/ifixit.owl#has_guideID	12345
http://example.org/ifixit.owl#requires_tool	http://example.org/ifixit.owl#Tool_3001
http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://example.org/ifixit.owl#Procedure

Example to try:

```
@prefix ex: <http://example.org/ifixit.owl#> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .

ex:Procedure_1001 rdf:type ex:Procedure ;
  ex:has_step ex:Step_2001, ex:Step_2002 ;
  ex:requires_tool ex:Tool_3001 ;
  ex:has_guideID "12345" .

ex:Step_2001 rdf:type ex:Step ;
  ex:has_description "Loosen the oil drain plug with a wrench." .

ex:Step_2002 rdf:type ex:Step ;
  ex:has_description "Drain the oil into a container." .

ex:Tool_3001 rdf:type ex:Tool ;
  ex:has_name "Wrench" .
```

Error handling:



Edit Knowledge Graph

Select Operation: Add
Enter Data (in Turtle/RDF format):

An error occurred: at line 26 of <>: Bad syntax (expected directive or statement) a

Dheya El Hak Chiha
23720753

How to run

Create and run the Virtual Environment:

- python3 -m venv venv
- source venv/bin/activate

Download dependencies:

- pip3 install flask
- export FLASK_APP=app.py (optional)
- pip install rdflib
- pip install owlready2

Run app:

- flask run
- go to ‘http://127.0.0.1:5000/’