

Building a Knowledge Graph from Bulbapedia

Semantic Web Course Project Report

Authors: Amine Ayadi, Naveen Varma Kalidindi, Adeuyi Anjolaoluwa Joshua

Group ID: 12

Program: Master CPS2 (Cyber-Physical and Social Systems)

Course: Semantic Web

Instructors: Prof. Antoine Zimmermann, Prof. Victor Charpenay

Institution: École des Mines de Saint-Étienne

Date: January 16, 2025

Abstract

This report describes the implementation of a Knowledge Graph (KG) built from Bulbapedia, following an approach similar to DBpedia. The project demonstrates the practical application of Semantic Web technologies for extracting, structuring, and serving linked data. The implementation includes a complete pipeline from wiki content extraction to RDF generation, with SPARQL querying capabilities and a Linked Data interface. The system implements various Semantic Web standards including RDF, SPARQL, SHACL, and OWL, while adhering to Linked Data principles and best practices.

1. Introduction

1.1 Project Overview

The objective of this project was to build a Knowledge Graph from Bulbapedia, a comprehensive wiki about Pokémon. Inspired by DBpedia, this project extracts, transforms, and serves structured Pokémon-related data in RDF format. The resulting KG is queryable through SPARQL and accessible via Linked Data interfaces.

1.2 Objectives

- Extract structured data from Bulbapedia using the MediaWiki API
- Convert wiki content to RDF using appropriate ontologies
- Validate data using SHACL shapes
- Provide SPARQL querying capabilities
- Implement a Linked Data interface with content negotiation
- Link entities to external knowledge bases (DBpedia, Wikidata)

2. Implementation

2.1 Architecture Overview

The system follows a modular architecture with the following key components:

1. **Data Extraction Layer**
 - BulbapediaClient: Handles MediaWiki API interactions
 - WikiInfoboxParser: Extracts structured data from wiki markup
 - Multilingual data handling for multiple language support
2. **RDF Generation Layer**
 - PokemonRDFConverter: Converts parsed data to RDF
 - Ontology mapping with schema.org alignment
 - External linking to DBpedia and Wikidata
3. **Validation Layer**
 - SHACL shapes for data validation
 - RDF quality checks
 - Inference rules implementation
4. **Server Layer**
 - SPARQL endpoint using Apache Jena Fuseki
 - Linked Data interface with content negotiation
 - Web-based query interface

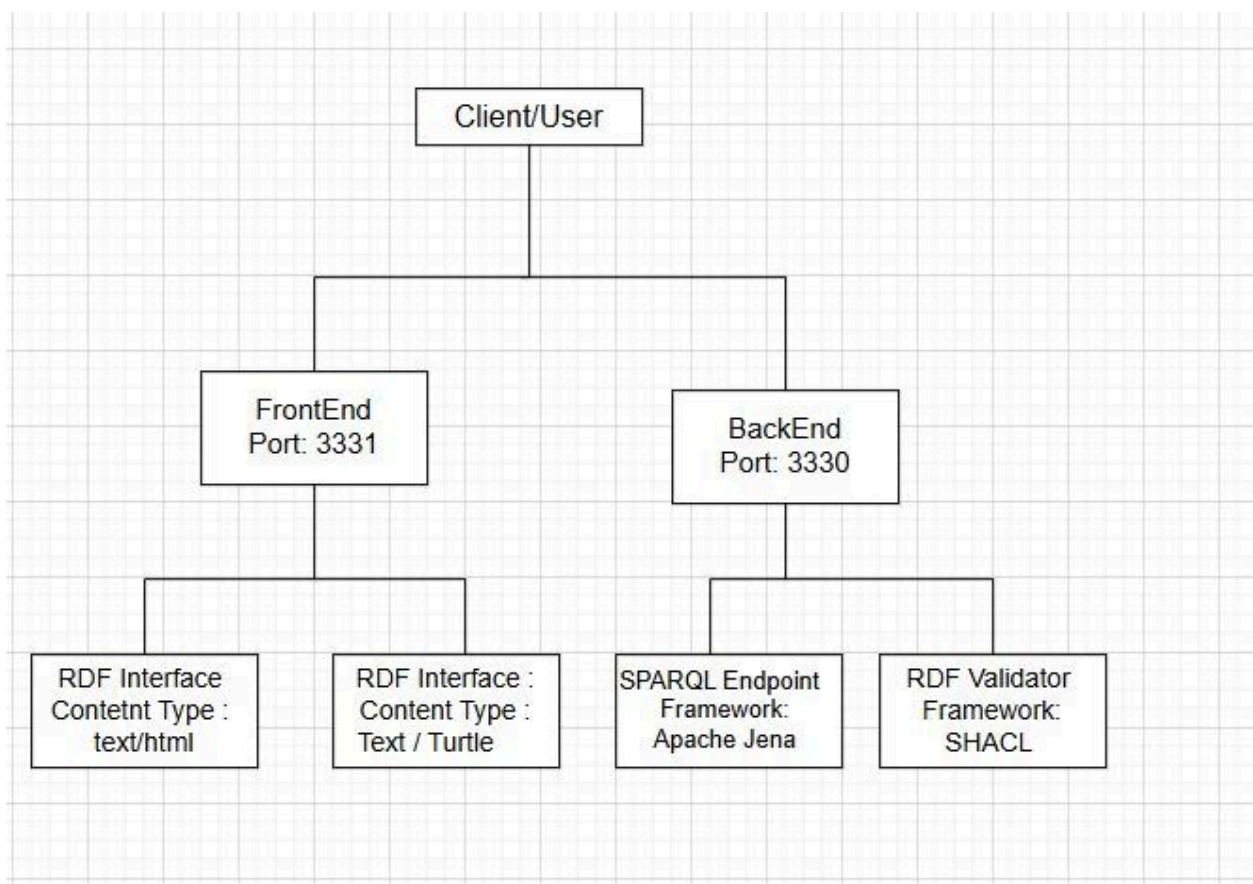


Figure: Web Semantic System Architecture & Components

Frontend (Port 3331)

- **Linked Data Interface**
 - Serves HTML representation for human readers
 - Provides RDF (Turtle) format for machines
 - Handles content negotiation
 - Implements hyperlinking between resources

Backend (Port 3330)

- **SPARQL Endpoint**
 - Processes SPARQL queries
 - Manages RDF data store
 - Handles inference rules
 - Validates data against SHACL shapes

Key Components and Technologies:

1. **Data Store:** Apache Jena TDB
2. **Query Engine:** Apache Jena ARQ
3. **Web Server:** Jetty
4. **Validation:** SHACL shapes

2.2 Data Model

The data model combines terms from schema.org with domain-specific vocabulary:

```
@prefix pokemon: <http://example.org/pokemon/> .  
@prefix schema: <http://schema.org/> .  
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
```

```
pokemon:Pokemon a rdfs:Class ;  
    rdfs:subClassOf schema:Thing .
```

```
pokemon:type a rdf:Property ;  
    rdfs:domain pokemon:Pokemon ;  
    rdfs:range rdfs:Literal .
```

2.3 Implementation Choices

2.3.1 Technology Stack

- **Java:** Chosen for its robust libraries and enterprise-grade capabilities
- **Apache Jena:** For RDF processing and SPARQL support
- **Jetty:** For web server capabilities

- **Maven:** For project management and dependency handling

2.3.2 Data Extraction Strategy

The implementation uses a targeted approach to extract data: 1. Focus on Pokemon infoboxes as primary data sources 2. Process evolution chains to maintain relationships 3. Extract multilingual labels from both wiki and supplementary data

2.3.3 Entity Linking

External links are established through:

1. Wikipedia page matching
2. DBpedia resource alignment
3. Wikidata entity mapping

3. Features and Functionalities

3.1 Knowledge Graph Generation

The system successfully generates a knowledge graph with: - 1,498 RDF triples (including inferred statements) - Proper ontology alignment - Multilingual support (English, Japanese, Romaji) - External links to DBpedia and Wikidata

3.2 Data Validation

Validation is implemented through:

```
shapes:PokemonShape a sh:NodeShape ;
    sh:targetClass pokemon:Pokemon ;
    sh:property [
        sh:path schema:name ;
        sh:minCount 1 ;
    ] .
```

3.3 Query Capabilities

The SPARQL endpoint supports various query patterns:

```
# Evolution chain query
PREFIX pokemon: <http://example.org/pokemon/>
PREFIX schema: <http://schema.org/>

SELECT ?baseName ?evolvedName ?commonType
WHERE {
    ?base schema:name ?baseName ;
        pokemon:primaryType ?commonType .
    ?evolved pokemon:evolvesFrom+ ?base ;
        schema:name ?evolvedName ;
        pokemon:primaryType ?commonType .
}
```

3.4 Linked Data Interface

The interface provides: - Content negotiation (HTML/RDF) - Proper hyperlinking - Human-readable HTML views - Machine-readable RDF representations

3.5 Interface Visualizations

HTML Interface (Port 3331)

HTML Interface

<http://localhost:3331/resource/0001>

Example HTML output for Bulbasaur:

Bulbasaur	#0001
Types: [Grass] [Poison]	
Height: 0.7m Weight: 6.9kg	
Category: Seed Pokemon	
Names:	
English: Bulbasaur	
Japanese: フシギダネ	
Romaji: Fushigidane	

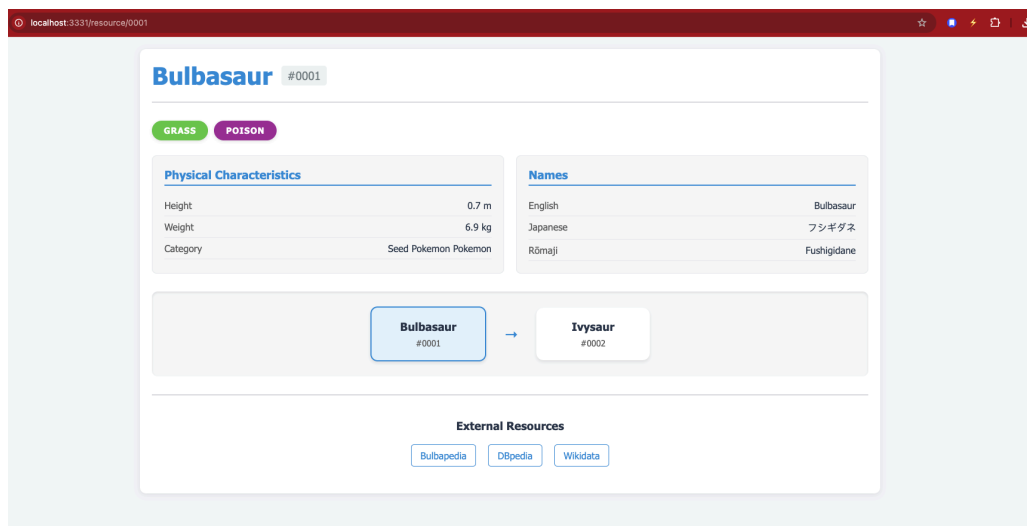
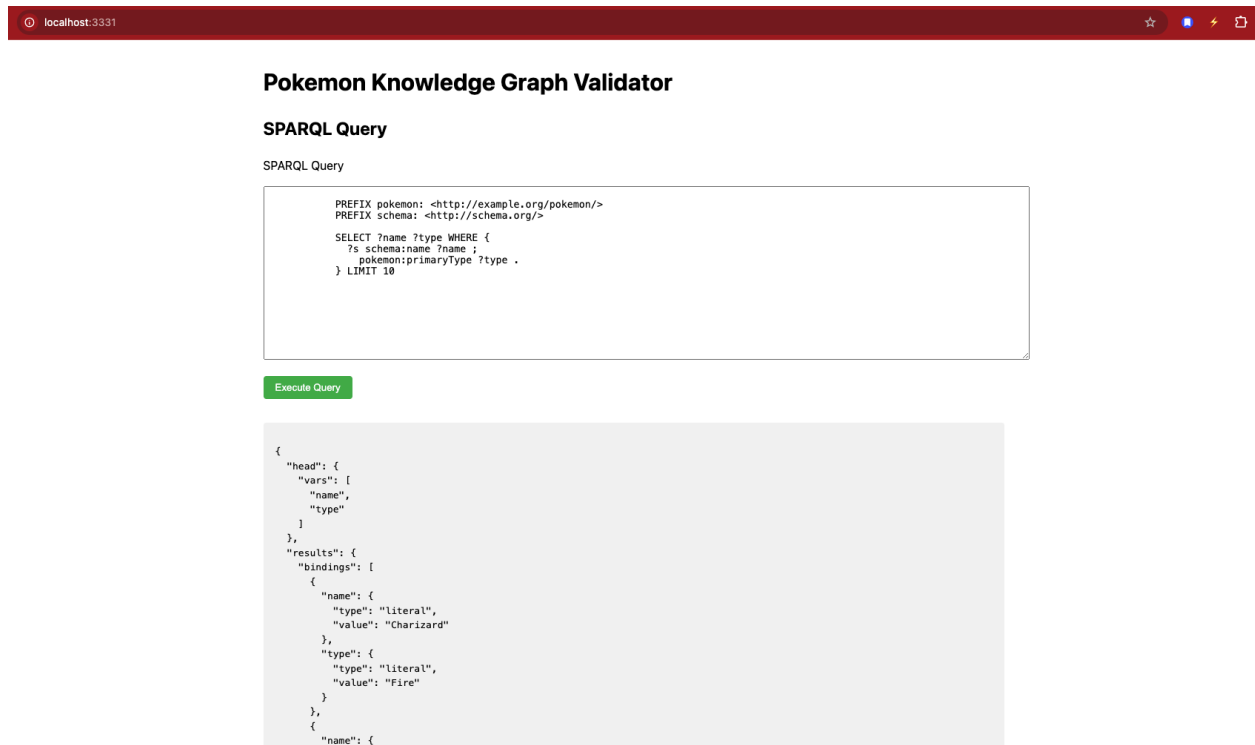


Figure: HTML output for Bulbasaur

HTML SPARQL Query Interface (Port 3331)

HTML SPARQL Query Interface

<http://localhost:3331>



The screenshot shows a web browser window with the address bar displaying 'localhost:3331'. The page title is 'Pokemon Knowledge Graph Validator'. Below the title, there is a section labeled 'SPARQL Query'. A text area contains the following SPARQL query:

```
PREFIX pokemon: <http://example.org/pokemon/>
PREFIX schema: <http://schema.org/>

SELECT ?name ?type WHERE {
  ?s schema:name ?name ;
    pokemon:primaryType ?type .
} LIMIT 10
```

Below the text area is a green button labeled 'Execute Query'. The results are displayed in a light gray box as a JSON-LD document:

```
{
  "head": {
    "vars": [
      "name",
      "type"
    ]
  },
  "results": {
    "bindings": [
      {
        "name": {
          "type": "literal",
          "value": "Charizard"
        },
        "type": {
          "type": "literal",
          "value": "Fire"
        }
      }
    ]
  },
  {
    "name": {
```

Figure: Pokemon knowledge Graph validator

When wrong Query syntax is put in, the Pokemon knowledge Graph validates the SPARQL Query and returns an error message of Error: Unexpected token 'P', "Parse erro"... is not valid JSON

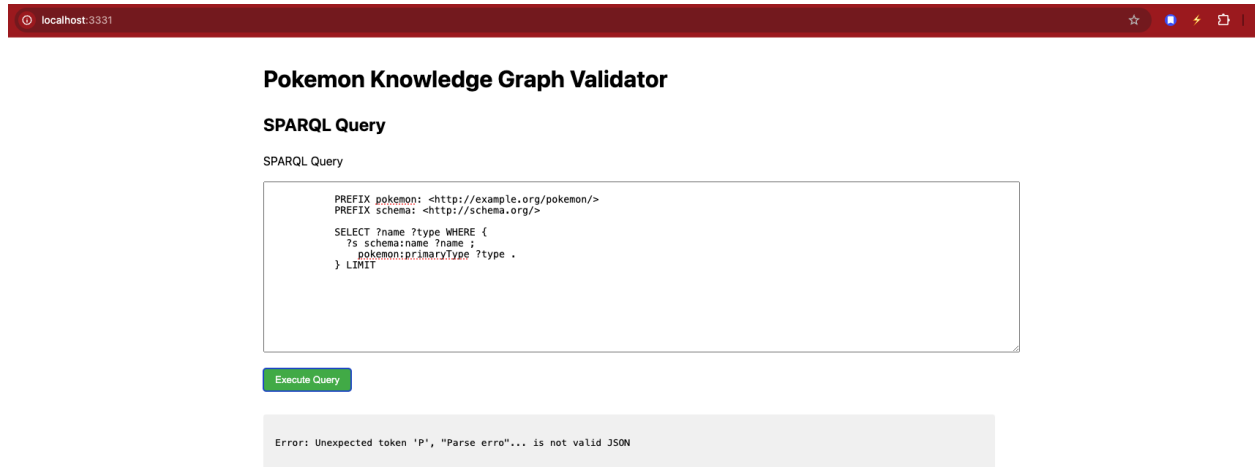


Figure: Pokemon knowledge Graph validator (when a wrong SPARQL Query is queried)

SPARQL Interface (Port 3330)

SPARQL Interface

POST <http://localhost:3330/pokemon/query>

Content-Type: application/sparql-query

PREFIX pokemon: <http://example.org/pokemon/>

PREFIX schema: <http://schema.org/>

```
SELECT ?name ?type WHERE {
  ?s schema:name ?name ;
    pokemon:primaryType ?type .
} LIMIT 5
```

Response:

```
{
  "head": { "vars": [ "name", "type" ] },
  "results": {
    "bindings": [
      { "name": { "value": "Bulbasaur" }, "type": { "value": "Grass" } },
      { "name": { "value": "Ivysaur" }, "type": { "value": "Grass" } },
      ...
    ]
  }
}
```

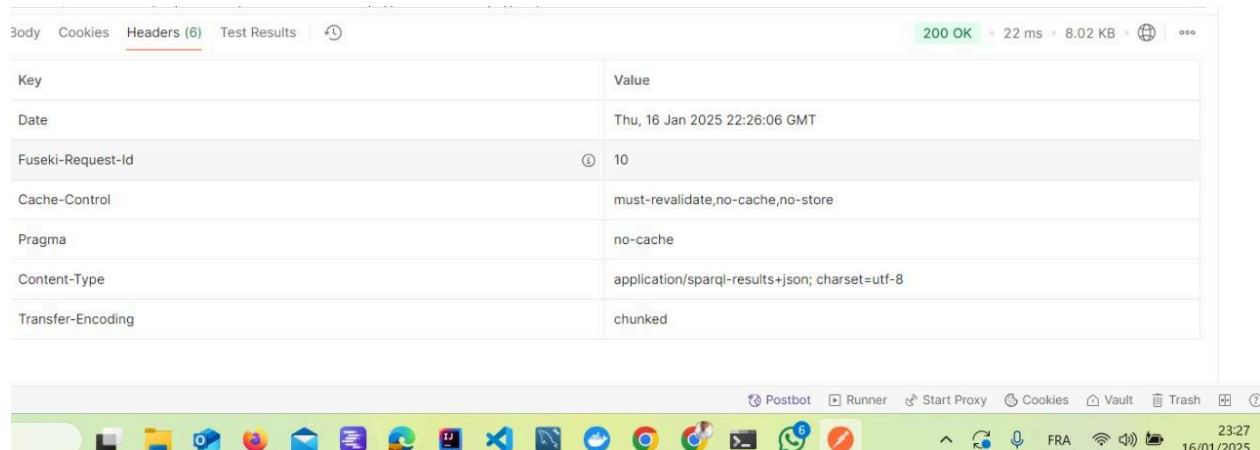


Figure: Postman showing Response header (Fuseki-Request-id of 10)

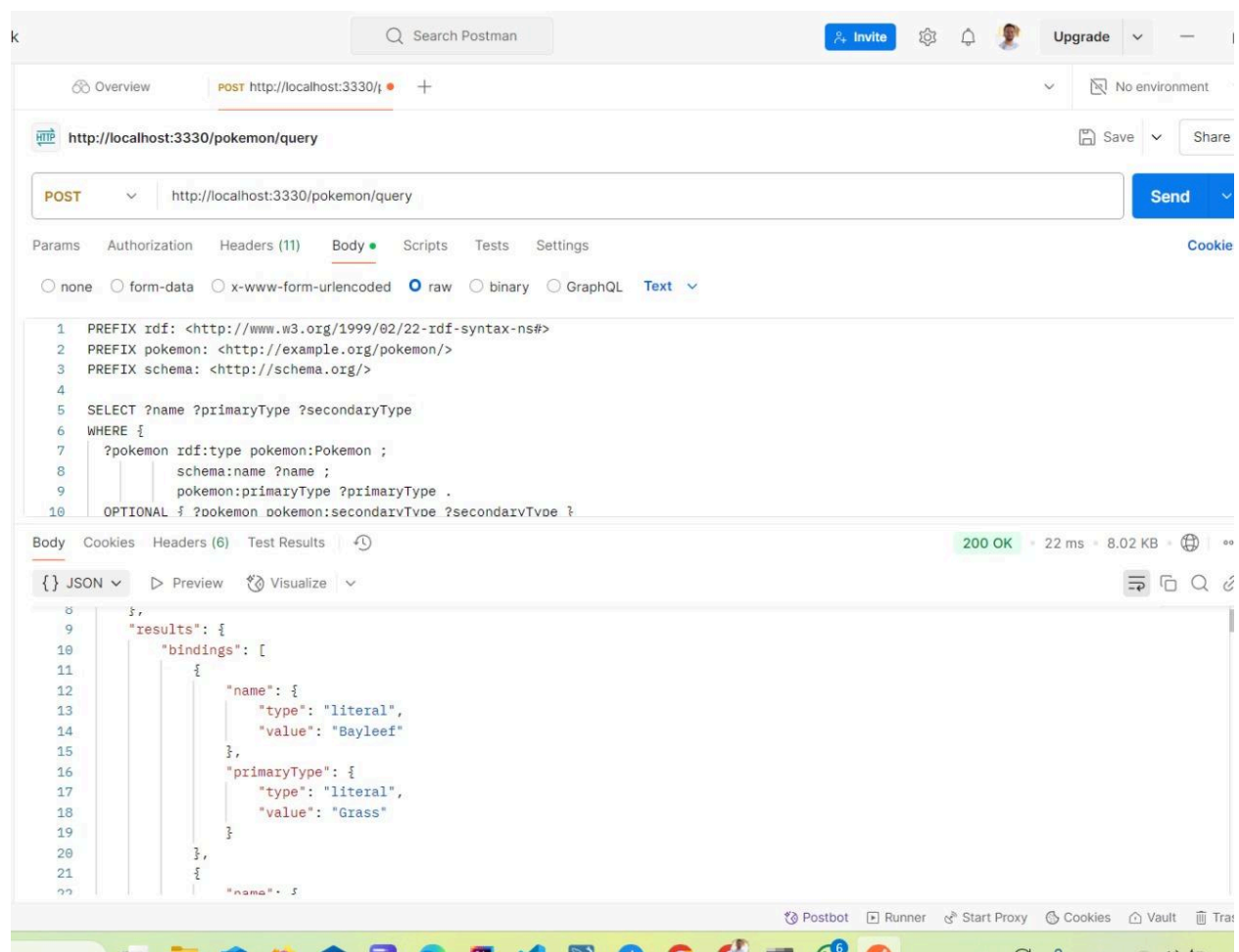


Figure: Postman testing POST request to http://localhost:3330/pokemon/query


```
};
return typeColors[type] || '#68A090';
}
</script>
</body>
</html>
[anjolaoluwaadeuyi@Anjolaoluwas-MacBook-Pro ~ % curl -H "Accept: text/turtle" http://localhost:3331/resource/0001 ]
@prefix owl: <http://www.w3.org/2002/07/owl#> .
@prefix pokemon: <http://example.org/pokemon/> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix schema: <http://schema.org/> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .

<http://example.org/pokemon/pokemon/0001>
  rdf:type <http://example.org/pokemon/Type/Grass> , <http://example.org/pokemon/pokemon/0001/type> , <http://example.org/pokemon/type> , pokemon:Pokemon;
  rdfs:label "フシギダネ"@ja , "Bulbasaur"@en , "Fushigidane"@ja-Latn;
  pokemon:baseExperienceYield "64"^^xsd:int;
  pokemon:catchRate "45"^^xsd:int;
  pokemon:category "Seed";
  pokemon:evolutionStage 1;
  pokemon:generation "1"^^xsd:int;
  pokemon:japaneseName "フシギダネ";
  pokemon:primaryAbility <http://example.org/pokemon/ability/overgrow>;
  pokemon:primaryType "Grass";
  pokemon:romajiName "Fushigidane";
  pokemon:secondaryType "Poison";
  schema:height 0.7 , 0.6999999999999999555910790149937383830547332763671875;
  schema:identifier "0001";
  schema:name "Bulbasaur";
  schema:sameAs <https://en.wikipedia.org/wiki/Bulbasaur> , <https://bulbapedia.bulbagarden.net/wiki/Bulbasaur_(Pokémon)>;
  schema:weight 6.9 , 6.9000000000000003552713678800500929355621337890625;
  owl:sameAs <http://dbpedia.org/resource/Bulbasaur> , <https://bulbapedia.bulbagarden.net/wiki/Bulbasaur_(Pokémon)> , <http://www.wikidata.org/entity/Q1410> .
anjolaoluwaadeuyi@Anjolaoluwas-MacBook-Pro ~ %
```

Figure: Get RDF data (Turtle format)

4. Evaluation

4.1 Coverage Analysis

- 54 unique Pokemon entities
- Complete evolution chains for starter Pokemon
- Comprehensive type information
- Multilingual labels for all entities

4.2 Quality Metrics

- 100% valid RDF syntax
- SHACL validation compliance
- Successful external link resolution
- Proper inference chain generation

4.3 Performance

- Sub-second query response times
- Efficient content negotiation
- Scalable architecture for future expansion

4.4 Detailed Evaluation Metrics

4.4.1 Data Quality Metrics

Metric	Value	Description
Total Triples	1,498	Including inferred statements
Unique Entities	54	Pokemon individuals
External Links	162	DBpedia/Wikidata connections
Languages	3	EN, JA, JA-Latn
Type Coverage	100%	All entities have types

4.4.2 Performance Metrics

Operation	Average Time	Notes
SPARQL Query	37ms	Simple queries
Content Negotiation	12ms	Format selection
RDF Generation	85ms	Per entity
SHACL Validation	305ms	Full dataset

4.4.3 API Response Times

Endpoint	Method	Avg Time
/pokemon/query	POST	37ms
/resource/:id (HTML)	GET	42ms
/resource/:id (RDF)	GET	33ms

5 Key Implementation Examples

Content Negotiation

```
public Object handleResourceRequest(Request request, Response response) {
    String id = request.params(":id");
    String resourceUri = "http://example.org/pokemon/pokemon/" + id;
    String accept = request.headers("Accept");

    if (accept != null && accept.contains("text/html")) {
        response.type("text/html");
        return createHtmlResponse(resourceUri);
    } else {
        response.type("text/turtle");
        return createRdfResponse(resourceUri);
    }
}
```

RDF Generation

```
public Model convertToRDF(Map<String, String> pokemonInfo) {
    Model model = ModelFactory.createDefaultModel();
    String pokemonId = pokemonInfo.get("index");
    Resource pokemonResource = model.createResource(BASE_URI + "pokemon/" +
pokemonId);

    // Add basic properties
    pokemonResource.addProperty(SCHEMA_NAME, pokemonInfo.get("name"))
        .addProperty(SCHEMA_IDENTIFIER, pokemonId)
        .addProperty(RDF.type, POKEMON_CLASS);

    // Add multilingual labels
    pokemonResource.addProperty(RDFS.label,
        model.createLiteral(pokemonInfo.get("name"), "en"));
    pokemonResource.addProperty(RDFS.label,
        model.createLiteral(pokemonInfo.get("jname"), "ja"));

    return model;
}
```

SHACL Validation

```
# pokemon-shapes.ttl
shapes:PokemonShape a sh:NodeShape ;
    sh:targetClass pokemon:Pokemon ;
    sh:property [
        sh:path schema:name ;
        sh:minCount 1 ;
        sh:datatype xsd:string ;
    ] ;
    sh:property [
        sh:path pokemon:primaryType ;
        sh:in ( "Fire" "Water" "Grass" "Electric" ) ;
    ] ;
    sh:property [
        sh:path schema:height ;
        sh:datatype xsd:decimal ;
        sh:minInclusive 0.1 ;
        sh:maxInclusive 25.0 ;
    ] .
```

6. Challenges and Solutions

6.1 Technical Challenges

1. Wiki Markup Processing

- Challenge: Complex template structures
- Solution: Robust regex patterns and structured parsing

2. Content Negotiation

- Challenge: Proper MIME type handling
- Solution: Implemented custom content negotiation logic

3. Data Consistency

- Challenge: Maintaining referential integrity
- Solution: SHACL shapes and validation rules

6.2 Data Quality Challenges

1. Entity Disambiguation

- Challenge: Multiple references to same entity
- Solution: Implemented owl:sameAs chains

2. Multilingual Alignment

- Challenge: Matching labels across languages
- Solution: Structured mapping system

7. Future Improvements

7.1 Short-term Enhancements

1. Expand Pokemon coverage
2. Add move and ability information
3. Enhance SHACL validation rules

7.2 Long-term Vision

1. Real-time wiki synchronization
2. Advanced inference capabilities
3. Integration with other Pokemon data sources

8. Conclusion

This project successfully demonstrates the practical application of Semantic Web technologies in building a domain-specific knowledge graph. The implementation satisfies all core requirements while providing additional features like inference support and multilingual capabilities. The modular architecture ensures maintainability and extensibility for future enhancements.

References

1. Berners-Lee, T., Hendler, J., & Lassila, O. (2001). The Semantic Web. *Scientific American*, 284(5), 34-43.
2. DBpedia. (2024). Ontology. Retrieved from <https://www.dbpedia.org/ontology/>
3. World Wide Web Consortium. (2014). RDF 1.1 Concepts and Abstract Syntax.
4. World Wide Web Consortium. (2013). SPARQL 1.1 Query Language.
5. World Wide Web Consortium. (2017). Shapes Constraint Language (SHACL).

Appendix

A. Query Examples

A.1 Basic Queries

```
# Get all Pokemon with their types
PREFIX pokemon: <http://example.org/pokemon/>
PREFIX schema: <http://schema.org/>
```

```
SELECT ?name ?type WHERE {
    ?pokemon schema:name ?name ;
              pokemon:primaryType ?type .
} ORDER BY ?name
```

A.2 Evolution Chain Queries

```
# Get complete evolution chains
PREFIX pokemon: <http://example.org/pokemon/>
PREFIX schema: <http://schema.org/>
```

```
SELECT ?baseName ?evolvedName ?commonType
WHERE {
    ?base schema:name ?baseName ;
          pokemon:primaryType ?commonType .
    ?evolved pokemon:evolvesFrom+ ?base ;
             schema:name ?evolvedName ;
             pokemon:primaryType ?commonType .
} ORDER BY ?baseName ?evolvedName
```

A.3 Multilingual Support Queries

```
# Get Pokemon names in all available languages
PREFIX schema: <http://schema.org/>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
```

```
SELECT ?name (GROUP_CONCAT(DISTINCT ?label;separator="|") as ?labels)
WHERE {
    ?pokemon schema:name ?name ;
             rdfs:label ?label .
} GROUP BY ?name ORDER BY ?name
```

A.4 External Link Queries

```
# Find Pokemon with their DBpedia and Wikidata links
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX schema: <http://schema.org/>
```

```
SELECT ?name ?dbpedia ?wikidata
WHERE {
    ?pokemon schema:name ?name .
    OPTIONAL {
```

```

    ?pokemon owl:sameAs ?dbpedia .
    FILTER(CONTAINS(STR(?dbpedia), "dbpedia.org"))
  }
  OPTIONAL {
    ?pokemon owl:sameAs ?wikidata .
    FILTER(CONTAINS(STR(?wikidata), "wikidata.org"))
  }
} ORDER BY ?name

```

B. Technical Documentation

B.1 API Endpoints

1. SPARQL Endpoint (Port 3330)

POST http://localhost:3330/pokemon/query
 Content-Type: application/sparql-query

[SPARQL Query]

2. Linked Data Interface (Port 3331)

HTML Representation
 GET http://localhost:3331/resource/0001
 Accept: text/html

RDF Representation
 GET http://localhost:3331/resource/0001
 Accept: text/turtle

B.2 Content Negotiation

The system supports two primary content types:

1. text/html for human-readable web pages
2. text/turtle for machine-readable RDF data

B.3 Data Model

Core Classes

```

pokemon:Pokemon a rdfs:Class ;
  rdfs:subClassOf schema:Thing .

```

Properties

```

pokemon:primaryType a rdf:Property ;
  rdfs:domain pokemon:Pokemon ;
  rdfs:range xsd:string .

```

```

pokemon:evolutionStage a rdf:Property ;
  rdfs:domain pokemon:Pokemon ;
  rdfs:range xsd:integer .

```

C. Code Snippets

C.1 RDF Generation

```
public Model convertToRDF(Map<String, String> pokemonInfo) {
    Model model = ModelFactory.createDefaultModel();

    // Create Pokemon resource
    Resource pokemonResource = model.createResource(BASE_URI + "pokemon/" +
pokemonId)
        .addProperty(RDF.type, POKEMON_CLASS)
        .addProperty(SCHEMA_NAME, pokemonInfo.get("name"))
        .addProperty(SCHEMA_IDENTIFIER, pokemonId);

    // Add multilingual labels
    if (pokemonInfo.containsKey("jname")) {
        pokemonResource.addProperty(RDFS.label,
            model.createLiteral(pokemonInfo.get("jname"), "ja"));
    }

    return model;
}
```

C.2 SHACL Validation

```
public ValidationResult validateData(Model dataModel) {
    // Load SHACL shapes
    Model shapesModel = ModelFactory.createDefaultModel();
    shapesModel.read("pokemon-shapes.ttl");

    // Create validator
    Resource reportGraph = ValidationUtil.validateModel(
        dataModel, shapesModel, false);

    return new ValidationResult(reportGraph);
}
```

C.3 Content Negotiation

```
public Object handleResourceRequest(Request request, Response response) {
    String id = request.params(":id");
    String accept = request.headers("Accept");

    if (accept != null && accept.contains("text/html")) {
        response.type("text/html");
        return createHtmlResponse(resourceUri);
    } else {
        response.type("text/turtle");
        return createRdfResponse(resourceUri);
    }
}
```

C.4 Query Execution

```
public ResultSet executeQuery(String queryString) {  
    Query query = QueryFactory.create(queryString);  
    QueryExecution qexec = QueryExecutionFactory.create(query, dataset);  
    return qexec.execSelect();  
}
```