Proyecto Final Web Semantica (Recomendador de Articulos Científicos)

Juan Esteban Rodríguez Ospino Julian Camilo Mora Valvuena

May 27, 2024

Abstract

El proyecto que se presenta tiene como objetivo principal aplicar los conceptos y tecnologías de la Web Semántica para el modelamiento, integración y publicación de información proveniente de fuentes de datos semi estructurados. La finalidad es desarrollar una aplicación web que despliegue información sobre papers y sus respectivas citas a partir de un set de datos vinculado (linked data set) construido, además de realizar recomendaciones sobre dichos papers.

1. Definición de la Ontología

1.1. Implementación y Razonamiento de la Ontología

La ontología fue creada utilizando el módulo `rdflib` de Python. `rdflib` es una biblioteca que permite trabajar con datos RDF (Resource Description Framework), facilitando la creación y manipulación de grafos RDF. Se utilizó este módulo para definir las clases, propiedades y relaciones descritas anteriormente.

Posteriormente, se utilizó un razonador de inferencias OWL (Web Ontology Language) para inferir más tripletas a partir de las definiciones y reglas existentes en nuestra ontología. El razonador permite deducir información implícita basada en las reglas y axiomas definidos, enriqueciendo así el conjunto de datos y mejorando la capacidad de respuesta de nuestra aplicación.

1.2. Clases

- **Anotacion**: Representa una anotación asociada a un documento de investigación.
- **Paper**: Clase que representa un documento de investigación científica.
- **Persona**: Clase que representa a una persona involucrada en la investigación científica.

1.3. Propiedades de Datos

- **authorId**: Propiedad que representa el identificador único de una persona.
- **citationVelocity**: Propiedad que indica la velocidad de citación de un documento de investigación.
- **conceptAnotation**: Propiedad que indica las anotaciones asociadas a un documento de investigación.
- **doi**: Propiedad que representa el DOI (Digital Object Identifier) de un documento.
- **hasAbstract**: Propiedad que indica el resumen o abstract de un documento de investigación.
- **hasAñoPublicacion**: Propiedad que representa el año de publicación de un documento.
- **hasCitationVelocity**: Propiedad que representa la velocidad de citación de un documento.
- **hasDOI**: Propiedad que representa el DOI de un documento.
- **hasInfluentialCitationCount**: Propiedad que indica el conteo de citas influyentes de un documento.
- **hasIntroduccion**: Propiedad que indica la introducción de un documento de investigación.
- **hasIsOpenAccess**: Propiedad que indica si un documento es de acceso abierto.
- **hasIsPublisherLicensed**: Propiedad que indica si un documento está licenciado por un editor.
- **hasKeyWords**: Propiedad que indica las palabras clave de un documento.
- **hasPaperId**: Propiedad que representa el ID de un documento.
- **hasVenue**: Propiedad que representa el lugar de publicación de un documento.
- **semanticUrl**: Propiedad que representa la URL semántica asociada a una persona.

1.4. Propiedades de Objetos

- **autor**: Propiedad que relaciona un documento con su(s) autor(es) o creador(es).
- **references**: Propiedad que representa las referencias o citas de un documento a otros papers.

2. Reglas SHACL

Las siguientes restricciones en SHACL (Shapes Constraint Language) están diseñadas para asegurar la integridad y validez de los datos en una ontología. Estas restricciones son particularmente importantes para las propiedades relacionadas con los títulos, nombres y fechas en la ontología de papers y autores. Las restricciones garantizan que los datos sean strings no vacíos, únicos donde sea necesario, y que cumplan con ciertos requisitos de formato y longitud. Esto ayuda a mantener la coherencia y la calidad de los datos, lo cual es crucial para cualquier sistema que dependa de la exactitud y completitud de la información.

```
1 # Restricción para asegurar que cada Paper tenga exactamente un abstract.

ns:PaperShape
    a sh:NodeShape ;
    sh:targetClass ns:Paper ;
    sh:property [
        sh:path ns:hasAbstract ;
        sh:maxCount 1 ;
        sh:minCount 1 ;
    ] .
```

Explicación: Se agrega esta regla para garantizar que cada paper tenga un abstract, ya que es un componente esencial para describir el contenido del documento.

```
2 # Restricción para asegurar que cada Paper tenga un DOI único.

ns:UniqueDOIShape

    a sh:NodeShape ;
    sh:targetClass ns:Paper ;
    sh:property [
        sh:path ns:hasDOI ;
        sh:maxCount 1 ;
        sh:minCount 1 ;
        sh:uniqueLang true ;
    ] .
```

Explicación: Esta regla se añade para asegurar que cada paper tenga un DOI único, evitando duplicados y facilitando la identificación y referencia precisa de cada documento.

```
3 # Restricción para asegurar que cada Paper tenga al menos una keyword.

ns:KeywordsShape
```

```
    a sh:NodeShape ;
    sh:targetClass ns:Paper ;
    sh:property [
        sh:path ns:hasKeyWords ;
        sh:minCount 1 ;
    ] .
```

Explicación: Se incluye esta regla para garantizar que cada paper tenga al menos una keyword, lo que mejora la indexación y recuperación de información relacionada con el documento.

```
4 # Restricción para asegurar que la velocidad de citación de un Paper sea un
número entero positivo.

ns:CitationVelocityShape
    a sh:NodeShape ;
    sh:targetClass ns:Paper ;
    sh:property [
        sh:path ns:hasCitationVelocity ;
        sh:minInclusive 0 ;
        sh:datatype xsd:integer ;
    ] .
```

Explicación: Esta regla se agrega para asegurar que la velocidad de citación de un paper sea un número entero positivo, lo que garantiza la coherencia de los datos y su interpretación adecuada.

```
5 # Restricción para asegurar que el año de publicación de un Paper sea un número
entero positivo.

ns:PublicationYearShape
    a sh:NodeShape ;
    sh:targetClass ns:Paper ;
    sh:property [
        sh:path ns:hasAñoPublicacion ;
        sh:minInclusive 0 ;
        sh:datatype xsd:integer ;
    ] .
```

Explicación: Se añade esta regla para garantizar que el año de publicación de un paper sea un número entero positivo, evitando valores negativos o no numéricos que puedan causar confusiones en los análisis temporales.

```
6 # Restricción para asegurar que el identificador de un Paper sea único.

ns:UniquePaperIdShape
    a sh:NodeShape ;
    sh:targetClass ns:Paper ;
    sh:property [
        sh:path ns:hasPaperId ;
        sh:maxCount 1 ;
        sh:minCount 1 ;
        sh:uniqueLang true ;
    ] .
```

Explicación: Esta regla se agrega para garantizar que cada paper tenga un identificador único, lo que facilita la identificación y referenciación precisa de cada documento en la ontología.

Restricciones acerca de las cadenas de caracteres ingresadas

1 Descripción: Esta restricción asegura que cada paper tenga un título que sea un string no vacío. Los títulos de los papers son esenciales para su identificación y referencia. Asegurar que cada título no sea vacío y esté correctamente formateado garantiza que los papers puedan ser reconocidos y citados adecuadamente.

```
# Restricción para asegurar que el título de un Paper sea un string no vacío.
ns:TitleShape
    a sh:NodeShape ;
    sh:targetClass ns:Paper ;
    sh:property [
        sh:path ns:titulo ;
        sh:minCount 1 ;
        sh:datatype xsd:string ;
        sh:minLength 1 ;
    ] .
```

2 Descripción: Esta restricción asegura que cada autor tenga un nombre que sea un string no vacío. Los nombres de los autores son cruciales para la atribución de trabajos y colaboraciones. Garantizar que los nombres no sean vacíos asegura que cada autor pueda ser identificado correctamente.

```
# Restricción para asegurar que el nombre de un autor sea un string no vacío.
ns:AuthorNameShape
    a sh:NodeShape ;
    sh:targetClass ns:Persona ;
```

```
    sh:property [
        sh:path ns.nombre ;
        sh:minCount 1 ;
        sh:datatype xsd:string ;
        sh:minLength 1 ;
    ] .
```

3 Descripción: Esta restricción asegura que cada autor tenga un identificador que sea un string no vacío. Los identificadores de los autores son importantes para distinguir entre diferentes individuos, especialmente cuando pueden compartir el mismo nombre. Asegurar que estos identificadores no sean vacíos y estén formateados correctamente ayuda a mantener la unicidad y claridad en la identificación.

```
# Restricción para asegurar que el identificador de un autor sea un string no vacío.
ns:AuthorIdShape
    a sh:NodeShape ;
    sh:targetClass ns:Persona ;
    sh:property [
        sh:path ns.authorId ;
        sh:minCount 1 ;
        sh:datatype xsd:string ;
        sh:minLength 1 ;
    ] .
```

4 Descripción: Esta restricción asegura que cada autor tenga una URL semántica que sea un string no vacío. Las URLs semánticas permiten la vinculación y referencia a los perfiles de los autores en la web. Asegurar que estas URLs sean strings no vacíos ayuda a garantizar que los enlaces sean válidos y funcionales.

```
# Restricción para asegurar que la URL semántica de un autor sea un string no vacío.
ns:SemanticUrlShape
    a sh:NodeShape ;
    sh:targetClass ns:Persona ;
    sh:property [
        sh:path ns.semanticUrl ;
        sh:minCount 1 ;
        sh:datatype xsd:string ;
        sh:minLength 1 ;
    ] .
```

5 Descripción: Esta restricción asegura que la longitud del título de un paper no exceda los 255 caracteres. Limitar la longitud de los títulos garantiza que los títulos sean concisos y manejables, facilitando su uso en bases de datos y sistemas de referencia.

```
# Restricción para asegurar que la longitud del título de un Paper no exceda los 255 car
ns:TitleLengthShape
    a sh:NodeShape ;
    sh:targetClass ns:Paper ;
    sh:property [
        sh:path ns.titulo ;
        sh:maxLength 255 ;
    ] .
```

3. Desarrollo del API

En esta sección se detalla el diseño y funcionamiento del API desarrollado para el proyecto, el cual tiene como objetivo principal facilitar la creación, exploración y recomendación de papers a partir de la ontología generada en etapas anteriores del proyecto. El API se ha implementado utilizando Angular para el front-end y Flask para el back-end, aprovechando las capacidades de cada tecnología para proporcionar una interfaz amigable y funcional.

3.1. Arquitectura del API

El API sigue una arquitectura cliente-servidor, donde el cliente, que en este caso es la interfaz de usuario desarrollada con Angular, realiza solicitudes al servidor implementado con Flask. Estas solicitudes pueden ser de varios tipos, como obtener la lista de todos los papers, filtrar los papers por ciertos criterios, ver detalles de un paper específico, agregar un nuevo paper, entre otros.

3.2. Autenticación y Seguridad

La decisión de implementar un sistema de autenticación se basa en la necesidad de garantizar la integridad y la calidad de los datos en la ontología. Se optó por un sistema de autenticación para evitar que usuarios no autorizados realicen modificaciones en la ontología, lo que podría comprometer la calidad de los datos y afectar la fiabilidad de las recomendaciones generadas.

Además, se consideró importante adoptar un enfoque de Linked Open Data para facilitar el intercambio y la reutilización de datos, siguiendo las mejores prácticas de la Web Semántica. Al restringir las modificaciones a usuarios verificados, se promueve la confianza en la ontología y se fomenta su uso por parte de la comunidad académica y científica.

3.3. Funcionalidades del API

El API proporciona las siguientes funcionalidades principales:

1. **Obtención de Lista de Papers:** Permite a los usuarios obtener una lista completa de todos los papers disponibles en la ontología.

2. **Filtrado de Papers:** Los usuarios pueden filtrar los papers según diferentes criterios, como similitud semántica en sus nombres, año de publicación, autores, entre otros. Esto se logra mediante consultas SPARQL realizadas en el backend.

3. **Detalles de Paper:** Proporciona información detallada sobre un paper específico, incluyendo su abstract, DOI, palabras clave, velocidad de citación, entre otros.

4. **Agregar Nuevo Paper:** Solo usuarios autenticados y verificados tienen permiso para agregar nuevos papers a la ontología. Esto se realiza mediante un formulario en el front-end que recopila la información necesaria y la envía al backend para su procesamiento y validación.

3.4. Implementación Técnica

En el lado del servidor, el API se implementa utilizando Flask, un framework de Python para el desarrollo de aplicaciones web. Flask proporciona una estructura modular y flexible que facilita la creación de endpoints para manejar las solicitudes HTTP entrantes.

En el lado del cliente, Angular se utiliza para desarrollar la interfaz de usuario interactiva. Angular ofrece un enfoque basado en componentes para construir aplicaciones web, lo que facilita la creación de interfaces de usuario dinámicas y receptivas.

3.5. Consultas SPARQL en el Backend

Para la funcionalidad de filtrado de papers, el backend realiza consultas SPARQL a la ontología para recuperar los papers que coincidan con los criterios especificados por el usuario. Estas consultas se ejecutan de manera eficiente para garantizar un rendimiento óptimo incluso cuando se trabaja con grandes conjuntos de datos.

3.6. Interfaz de Usuario

La interfaz de usuario desarrollada con Angular proporciona una experiencia intuitiva y amigable para los usuarios. Permite navegar fácilmente entre los diferentes papers, ver detalles específicos, filtrarlos según diferentes criterios y agregar nuevos papers cuando sea necesario.

Consideraciones

Para este proyecto falta hacer una correcta conexión de nuestra base de datos con nuestra API. Debido a problemas de carga tuvimos que hacer las pruebas con una muestra de los datos