Universidad Politécnica

de Madrid

Escuela Técnica Superior de

Ingenieros Informáticos

Grado en Ingeniería Informática

Trabajo Fin de Grado

Diseño y Desarrollo de un Portal Web de Patrones de Diseños Ontológicos

Autora: Ariana de la Cruz Rodríguez

Tutora: María Poveda Villalón

Madrid, Julio 2024

Este Trabajo Fin de Grado se ha depositado en la ETSI Informáticos de la Universidad Politécnica de Madrid para su defensa.

Trabajo Fin de Grado

Grado en Ingeniería Informática

Título: Diseño y Desarrollo de un Portal Web de Patrones de Diseños Ontológicos

Junio 2024

Autora: Ariana de la Cruz Rodríguez

Tutora: María Poveda Villalón

Departamento de Inteligencia Artificial

ETSI Informáticos

Universidad Politécnica de Madrid

Resumen

El presente Trabajo de Fin de Grado se centra en el desarrollo de una plataforma *web* diseñada para visualizar los patrones de diseño de ontologías generados por una herramienta específica. En un contexto donde la gestión y representación del conocimiento son fundamentales, esta plataforma busca ofrecer una solución efectiva para explorar y analizar modelos ontológicos de manera eficiente y accesible.

El objetivo principal de este trabajo es proporcionar a los usuarios una herramienta interactiva que les permita explorar y comprender los patrones de diseño de ontologías de manera intuitiva. Para lograr esto, se emplearán técnicas modernas de desarrollo *web* y se integrarán funcionalidades avanzadas para mejorar la experiencia del usuario.

El desarrollo de la plataforma se ha dividido en varias fases, que incluyen el análisis de los datos generados por la herramienta, el diseño de la interfaz de usuario, la implementación del backend y frontend de la aplicación, y la realización de pruebas exhaustivas para garantizar su funcionalidad y usabilidad.

A lo largo del proceso de desarrollo, se han enfrentado diversos desafíos, como la gestión eficiente de grandes volúmenes de modelos ontológicos, la integración de tecnologías *web* para una experiencia de usuario óptima y la garantía de la seguridad y robustez de la plataforma.

El resultado final es una plataforma *web* que permite a los usuarios explorar y analizar los patrones de diseño de ontologías de manera intuitiva y eficiente. Se espera que esta herramienta contribuya significativamente al campo de la ingeniería informática y a la gestión del conocimiento en general, proporcionando una solución efectiva para la visualización de modelos ontológicos en entornos *web*.

En resumen, este Trabajo de Fin de Grado representa un esfuerzo significativo en el desarrollo de una plataforma *web* innovadora para la visualización de patrones de diseño de ontologías, con el potencial de mejorar la comprensión y el análisis de modelos ontológicos en diversos contextos de aplicación.

Abstract

The present Bachelor's Thesis focuses on the development of a web platform designed to visualize ontology design patterns generated by a specific tool. In a context where knowledge management and representation are fundamental, this platform aims to provide an effective solution for exploring and analyzing ontological models efficiently and accessibly.

The main objective of this work is to provide users with an interactive tool that allows them to explore and understand ontology design patterns intuitively. To achieve this, modern web development techniques will be employed, and advanced functionalities will be integrated to enhance the user experience.

The development of the platform has been divided into several phases, including data analysis generated by the tool, user interface design, implementation of the backend and frontend of the application, and comprehensive testing to ensure its functionality and usability.

Throughout the development process, various challenges have been encountered, such as efficiently managing large volumes of ontological models, integrating web technologies for optimal user experience, and ensuring the security and robustness of the platform.

The final result is a web platform that enables users to explore and analyze ontology design patterns intuitively and efficiently. It is expected that this tool will significantly contribute to the field of computer engineering and knowledge management in general, providing an effective solution for visualizing ontological models in web environments.

In summary, this Bachelor's Thesis represents a significant effort in the development of an innovative web platform for visualizing ontology design patterns, with the potential to enhance the understanding and analysis of ontological models in various application contexts.

Tabla de contenidos

Índice de ilustraciones

Índice de Tablas

1 Introducción

En el ámbito de la ingeniería informática, la creación y el análisis de ontologías juegan un papel crucial en la representación y organización del conocimiento. Con el creciente volumen de datos y la complejidad de los sistemas informáticos, la necesidad de herramientas efectivas para la gestión y visualización de ontologías se ha vuelto cada vez más evidente. En este contexto, el presente Trabajo de Fin de Grado se centra en el desarrollo de una plataforma *web* diseñada para visualizar los patrones de diseño de ontologías generados por una herramienta específica.

El objetivo principal de este trabajo es proporcionar una herramienta interactiva que permita a los usuarios explorar y analizar los patrones de diseño de ontologías de manera eficiente y efectiva. A través de la implementación de técnicas de desarrollo *web* modernas y la integración de funcionalidades avanzadas, se busca facilitar la comprensión y el análisis de los modelos ontológicos generados por la herramienta mencionada.

En este contexto, la presente introducción tiene como objetivo proporcionar una visión general del contexto en el que se desarrolla el TFG, así como presentar los objetivos que se pretenden alcanzar con este trabajo. A lo largo de este documento, se detallarán los procesos de análisis, diseño, implementación y evaluación de la plataforma *web*, destacando los desafíos enfrentados y las soluciones propuestas en cada etapa del desarrollo.

En esta primera versión del documento se centra en realizar una primera versión del estado del arte y presentar todas las actualizaciones y tareas realizadas hasta el momento.

2 Estado del Arte

En el actual apartado se presentan los conceptos básicos que son fundamentales para comprender el desarrollo y funcionamiento de la aplicación *web* del presente Trabajo Fin de Grado. Estos conceptos permitirán una mayor compresión del contexto en el que se sitúa el trabajo descripto en el capítulo anterior.

2.1 Ontologías

Una estructura ontológica se refiere a un mapa detallado que describe las diferentes partes, características y relaciones entre entidades que existen en un dominio específico de conocimiento [3]. Estas estructuras juegan un papel fundamental en el ámbito de la informática, ya que son creadas para simplificar la complejidad y organizar la información de una forma clara y eficiente.

Para el desarrollo de este tipo de estructuras se hace uso principalmente de los siguientes lenguajes:

- **RDF** (*Resource Description Framework*): RDF es un estándar del *World Wide Web Consortium* (W3C) utilizado para representar información en la *web* de una manera que sea comprensible tanto para humanos como para máquinas. Utiliza tripletas que consisten en sujetos, predicados y objetos para representar declaraciones simples sobre recursos *web*. Esto proporciona una base para la creación de ontologías al permitir la descripción de recursos y sus relaciones [4].

- **RDFS** (*Resource Description Framework Schema*): RDFS extiende RDF proporcionando un vocabulario básico para describir las estructuras y las relaciones en los datos RDF. Permite la definición de clases y propiedades, lo que facilita la creación de ontologías simples. Sin embargo, RDFS tiene limitaciones en la expresividad y no puede capturar todas las complejidades del mundo real [5].

- **OWL** (*Web Ontology Language*): OWL es un lenguaje de ontología más avanzado que permite una representación más rica y expresiva de conocimiento de dominio. Viene en varias variantes, como OWL Lite, OWL DL y OWL Full, que difieren en complejidad y capacidad de razonamiento. OWL permite la definición de clases, propiedades y restricciones más complejas, lo que lo hace adecuado para la creación de ontologías detalladas y sofisticadas [7].

Sin embargo, la creación de este tipo de estructuras puede ser un proceso complejo y laborioso, especialmente cuando se trata de recopilar y organizar información a partir de fuentes no estructuradas. Por ello, se hacen uso de diferentes sistemas de extracción de ontologías, los cuales permiten automatizar este proceso y extraer conocimiento semántico de manera eficiente.

Un ejemplo de sistema de extracción es el "*LCA Ontology Learning Framework*", el cual se trata de un sistema diseñado para extraer ontologías a partir de textos no estructurados. El sistema utiliza una combinación de técnicas de procesamiento de lenguaje natural (NLP) y aprendizaje automático para realizar la extracción de ontologías. Entre las herramientas utilizadas se encuentran bibliotecas de NLP como NLTK (*Natural Language Toolkit*) y spaCy para el preprocesamiento de texto, así como algoritmos de aprendizaje automático para la identificación y clasificación de entidades relevantes en los textos [2].

Otro ejemplo, es la utilización de RDFlib [6], que se trata de una librería de Python que proporciona herramientas para cargar, manipular, representar y generar archivos RDF que contienen la información ontológica necesaria para la detección de patrones. El uso de todas las herramientas que proporciona RDFlib se puede observar en la herramienta[1] que proporciona los archivos de entrada que vamos a utilizar para desarrollar la página *web* descrita en el trabajo presente.

Esta herramienta como se puede observar en la Ilustración 1, identifica y genera ficheros en los que se describen las estadísticas y relaciones de los patrones existentes en las ontologías. Para ello, extrae el contenido de cada ontología filtrando los términos que son el sujeto de una tripleta cuyo predicado es "`owl:equivalentClass`", "owl:disjointWith" o "`rdfs:subClassOf`" y cuyo objeto es un nodo en blanco. Estas estructuras se representan como árboles para resaltar los diferentes componentes de los nodos en blanco. La herramienta generará dos archivos:

1. **Structure_term_name**: Un archivo donde se especifican los URIs de los términos.

2. **Structure_term_type**: Un archivo donde se especifica el tipo de los términos en lugar de sus URIs.

A veces, el tipo de un término no se puede especificar debido a diferentes errores en la ontología. En estos casos, se aplican inferencias para mantener el máximo número de estructuras útiles y son volcadas en los ficheros **Structure_term_inferred_blank_nodes** y **Structure_term_inferred_type**.

. Algunas de estas inferencias incluyen:

- Inferencia de tipos basada en restricciones.

[1] GitHub – "Sergio-Carulli/Patrones", https://github.com/Sergio-Carulli/Patrones/tree/main

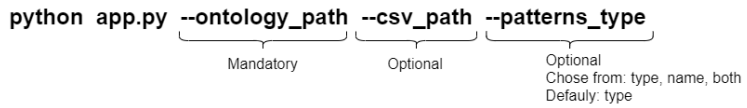
- Asignación de clases a nodos en blanco dentro de intersecciones, uniones o complementos.

- Identificación de valores de datos dentro de enumeraciones como "Data value".

Finalmente, la herramienta identifica patrones encontrando estructuras iguales. La única condición para identificar un patrón es que haya al menos dos estructuras con el mismo contenido. La herramienta generará dos archivos:

1. **Patterns_name:** Un archivo que contiene los patrones encontrados a través del archivo **Structure_term_inferred_blank_nodes**.

2. **Patterns_type:** Un archivo que contiene los patrones encontrados a través del archivo **Structure_term_inferred_type**.



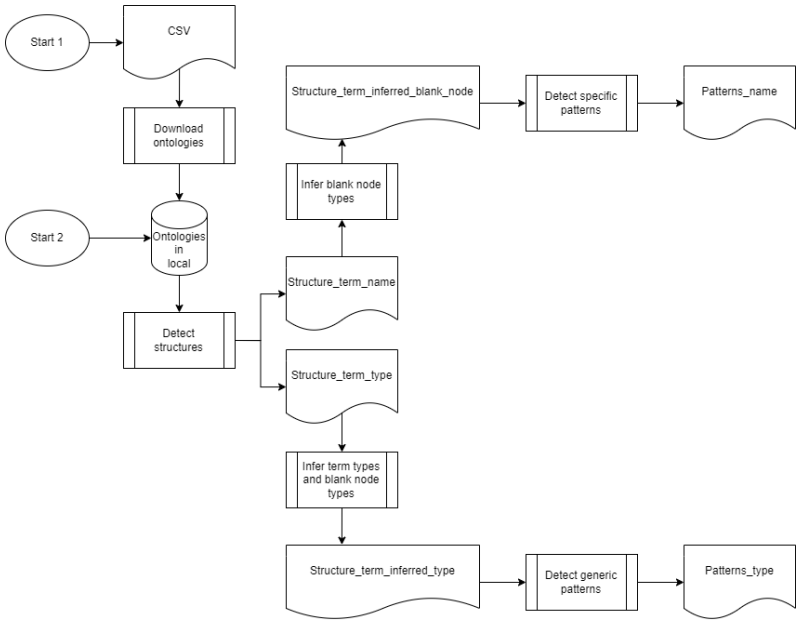


Ilustración 1. Flujo Herramienta Ficheros de Entrada[2]

[2] GitHub – "Sergio-Carulli/Flujo.drawio",

https://github.com/Sergio-Carulli/Patrones/blob/main/Diagrams/Flujo.drawio

Para facilitar la comprensión y el análisis de estos ficheros, en el presente TFG se desarrolla una página *web* dedicada específicamente a la visualización de estos documentos. Esta página *web* es capaz de leer los documentos generados por la herramienta y visualizar cada uno de los elementos más relevantes para el usuario. De esta manera, se proporciona una herramienta interactiva y accesible para explorar y comprender la estructura y las relaciones dentro de las ontologías de manera más intuitiva.

2.2 Páginas Web Estáticas

Las páginas *web* estáticas son aquellos sitios web cuyo contenido permanece constante y no cambia según la interacción del usuario. Están compuestas principalmente por archivos HTML, CSS y JavaScript, que se sirven directamente desde el servidor al navegador del usuario sin necesidad de procesamiento adicional en el servidor. Esto significa que, independientemente de cuántas veces visite un usuario una página *web* estática, siempre verá el mismo contenido, diseño y funcionalidad, a menos que se realicen cambios manuales en el código fuente [1].

Las páginas *web* estáticas son ideales para sitios que no requieren mucha interactividad o actualizaciones frecuentes. Son más rápidas de cargar y más fáciles de crear y mantener debido a su simplicidad. No requieren lenguajes de programación del lado del servidor como PHP, lo que reduce la complejidad técnica involucrada en su desarrollo.

Sin embargo, las páginas *web* estáticas tienen limitaciones en cuanto a la interactividad y la capacidad de manejar contenido dinámico. No pueden realizar operaciones como guardar datos del usuario, generar contenido basado en la sesión del usuario, o adaptarse automáticamente a diferentes regiones o idiomas. Para estos casos, se prefieren las páginas *web* dinámicas, que utilizan lenguajes de programación del lado del servidor y bases de datos para ofrecer contenido y funcionalidades más flexibles y personalizadas.

2.3 Páginas Web Dinámicas

Las páginas *web* dinámicas son aquellas capaces de cambiar su contenido en tiempo real, respondiendo a la interacción del usuario o a datos provenientes de una base de datos u otro origen dinámico. Se desarrollan utilizando tecnologías como JavaScript [3], que permite la interactividad y la personalización del contenido para cada usuario [1].

Este tipo de páginas *web* contienen las siguientes características claves:

[3] ECMA International, "JavaScript", https://262.ecma-international.org/10.0/

- **Interactividad:** Las páginas *web* dinámicas son altamente interactivas, lo que significa que los usuarios pueden participar activamente con el contenido y realizar acciones como enviar formularios, interactuar con los botones y filtrar información en tiempo real.

- **Personalización:** Estas páginas tienen la capacidad de adaptarse a las preferencias y comportamientos individuales de los usuarios, lo que permite la entrega de contenido personalizado y relevante en función de sus necesidades específicas.

- **Actualización de datos en tiempo real:** Las páginas dinámicas pueden mostrar datos actualizados en tiempo real, lo que garantiza que la información presentada esté siempre al día y sea precisa. Esto es especialmente importante en contextos donde la información cambia constantemente.

En el contexto del presente TFG, se ha desarrollado una página *web* dinámica cuyo objetivo principal es visualizar los archivos creados por la herramienta descrita anteriormente, de modo que el contenido de la página *web* se actualice dependiendo de los datos leídos en los archivos de entrada.

Para implementar esta funcionalidad, se ha utilizado Flask[4] como *framework* para el desarrollo del *backend*. Utilizar un *framework web*[5] como Flask ofrece varias ventajas, entre ellas la simplificación de la gestión de rutas, el manejo de solicitudes HTTP y la integración con bases de datos y otras herramientas. Flask, en particular, es conocido por su simplicidad y flexibilidad, lo que lo hace ideal para proyectos de pequeño a mediano tamaño. Además, Flask es un microframework de Python diseñado para simplificar el desarrollo de aplicaciones *web* siguiendo el patrón Modelo-Vista-Controlador (MVC). La característica "micro" de Flask significa que, aunque proporciona las herramientas básicas necesarias para crear una aplicación *web* funcional, no incluye funcionalidades adicionales que podrían no ser necesarias para todos los proyectos. Sin embargo, Flask es altamente extensible, permitiendo agregar funcionalidades adicionales mediante extensiones (*plugins*) según sea necesario. Esta flexibilidad, junto con su sistema de extensiones, facilita la expansión de la funcionalidad de la aplicación sin comprometer la simplicidad inicial del proyecto. Además, Flask incluye un sistema de pruebas unitarias integrado y un depurador, lo que facilita la detección y corrección de errores durante el desarrollo. También es compatible con el protocolo WSGI, lo que permite a las aplicaciones Flask interactuar con varios servidores *web* y frameworks de Python. Por último, como software libre bajo la licencia BSD,

[4] Flask, "Pallets Projects. Flask Documentation.", https://flask.palletsprojects.com/en/3.0.x/

[5] MDN Web Docs, "Mozilla Developer Network. First Steps with Web Frameworks.", https://262.ecma-international.org/10.0/

Flask es accesible para todos los desarrolladores y puede ser modificado y extendido según sea necesario, contribuyendo a su popularidad y a la disponibilidad de una amplia comunidad de desarrolladores dispuestos a compartir soluciones y mejores prácticas.

3 Desarrollo

En este capítulo se desarrolla el procedimiento llevado a cabo para el desarrollo de la plataforma *web*, detallando cada una de las fases desde el diseño hasta la implementación.

3.1 Requisitos

En el siguiente apartado se describe tanto los requisitos funcionales como los no funcionales que se han propuesto para este proyecto.

Cada uno de los requisitos se llevarán a cabo en las etapas de desarrollo del proyecto.

3.1.1 Requisitos Funcionales

1. **Visualizar Patterns_name.txt:** el sistema debe de permitir la visualización del archivo.txt mostrando las clases y los datos estadísticos de cada patrón que contiene el archivo

2. **Visualizar Patterns_name.csv:** el sistema debe de permitir mostrar las estructuras presentes en cada uno de los patrones que se encuentran en el .csv.

3. **Visualizar Patterns_type.txt:** el sistema debe de permitir la visualización del archivo.txt mostrando las clases y los datos estadísticos de cada patrón que contiene el archivo

4. **Visualizar Patterns_type.csv:** el sistema debe de permitir mostrar las estructuras presentes en cada uno de los patrones que se encuentran en el .csv.

5. **Visualizar diagramas XML:** el sistema debe de permitir la visualización de los diagramas XML en .svg de cada uno de los patrones.

6. **Visualizar Structure_term_inferred_type.txt:** el sistema debe de permitir la visualización de los datos de las diferentes estructuras que se encuentran en el archivo.

7. **Visualizar Structure_term_inferred_blank_nodes.txt**: el sistema debe de permitir la visualización de los datos de las diferentes estructuras que se encuentran en el archivo.

8. **Menú navegación:** la plataforma debe contener una navegación que permita al usuario moverse entre las diferentes vistas.

9. **Tabla de contenidos:** la plataforma debe de contener una tabla de contenidos en cada una de las vistas que permita al usuario moverse de forma rápida y ágil entre las diferentes secciones.

10. **Imágenes interactivas:** la plataforma debe de permitir al usuario interactuar con las imágenes: hacer zoom, descargar, compartir.

3.1.2 Requisitos No Funcionales

Requisitos de Rendimiento:

1. **Tiempo de Respuesta:** la aplicación debe ser capaz de cargar cualquieras de las páginas dentro de 2 segundos en condiciones de carga normal.

2. **Capacidad:** El sistema debe ser capaz de soportar hasta 1000 usuarios sin degradación del rendimiento.

Requisitos de Usabilidad:

1. **Facilidad de Uso:** la aplicación debe ser fácil de entender y cualquier función se debe poder alcanzar en no más de 4 clics desde la página inicial.

2. **Visibilidad:** la aplicación debe de tener un diseño claro y simple que haga que la experiencia del usuario sea agradable.

Requisitos de Fiabilidad:

1. **Disponibilidad:** el sistema debe de estar disponible el 99% del tiempo.

2. **Tolerancia a fallos:** el sistema debe ser capaz de continuar en modo reducido si alguno de los componentes no crítico falla, como puede ser la falta de algunas de las imágenes o el fallo de lectura de un csv.

Requisitos de Mantebilidad:

1. **Modularidad:** el código debe estar organizado de forma clara de tal forma que permita cambios independientes para futuras actualizaciones.

2. **Documentación:** Todo el código debe estar documentado de manera adecuada para facilitar el mantenimiento y las futuras actualizaciones.

3.2 Diseño

Para el diseño de las diferentes vistas de la aplicación se ha hecho uso de la aplicación Figma[6]. Figma se trata de una plataforma colaborativa y versátil para el diseño de interfaces de usuario. Su capacidad para simular interacciones y animaciones, así como su facilidad de uso, hacen de Figma una opción preferente para diseñadores que buscan eficiencia y precisión en la elaboración de prototipos de páginas *web*.

3.2.1 Vista Patrones

Para la visualización de los patrones extraídos tanto de pattern_name.txt/.csv como los extraídos de pattern_type.txt/.csv se han creado varios diseños y se han realizados diferentes pruebas de usabilidad para determinar cuál de los diseños se ajusta más a las necesidades del usuario.

3.2.1.1 Diseño 1

Como se puede ver en la Ilustración 2, en el primer diseño que se ha llevado a cabo se muestran en una misma vista tanto los pattern_type como los pattern_name de cada uno de los patrones. El usuario puede desplazarse de un patrón a otro haciendo uso de la tabla de contenidos que se encuentra en la parte izquierda de la interfaz.

Tras realizarse las pruebas de usabilidad se ha observado que la cantidad de información en una sola vista supone una sobrecarga informativa. Esto afecta negativamente la legibilidad, pues los usuarios tienen dificultades para localizar y procesar toda la información.

[6] Figma Community, "Explore templates, plugins, and widgets published by the community", https://www.figma.com/community

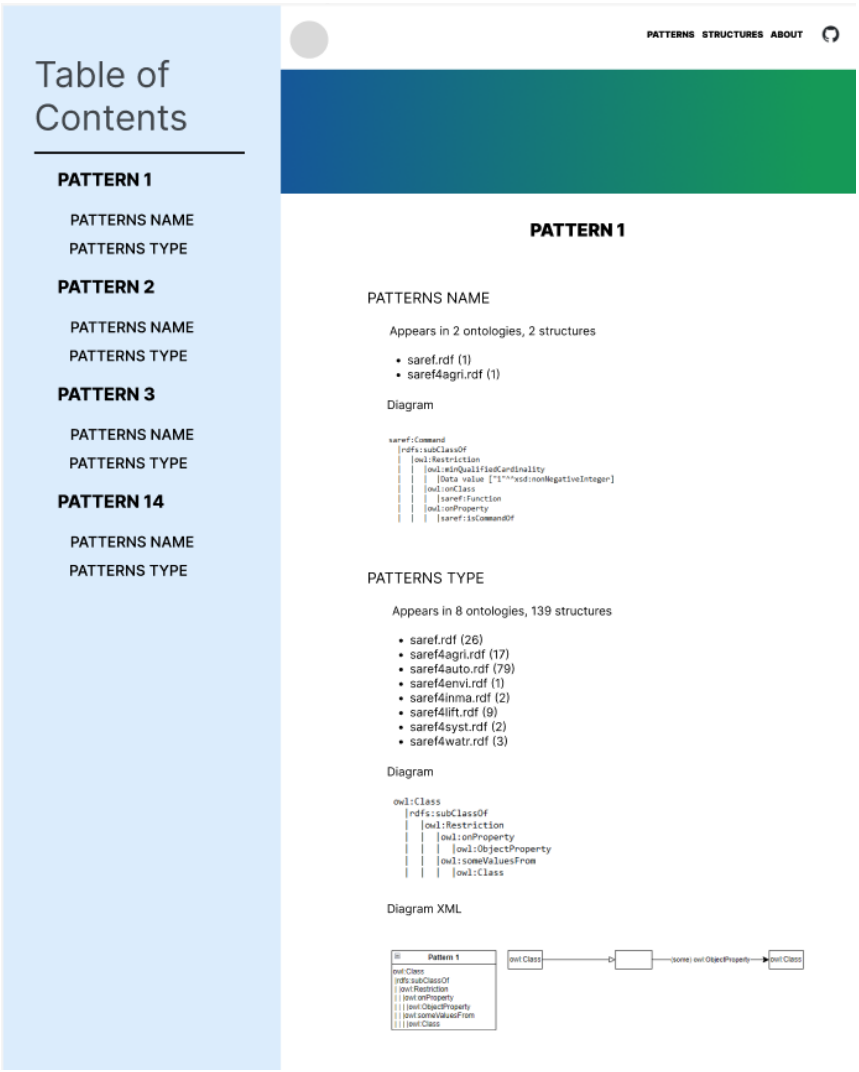


Ilustración 2. Vista Patrones: Diseño 1

3.2.1.2 Diseño 2

El segundo diseño como se muestra en la Ilustración 3 es muy similar al primero, con la única diferencia que las estructuras en las que aparece en vez de ser mostradas a través de una lista se muestran en un diagrama circular. Esta modificación visual pretende ofrecer una representación más dinámica e interactiva.

Tras realizarse las pruebas de usabilidad se ha observado que, al igual que en el diseño 1, la cantidad de información en una sola vista afecta negativamente la legibilidad.

Por otro lado, los diagramas visuales, aunque son visualmente atractivos, dificultan la navegación e interpretación rápida de las relaciones debido a su naturaleza más gráfica y menos estructuradas con las listas tradicionales. Además, se requiere de más recursos para renderizar correctamente, en

especial cuando se presentan muchas estructuras, que suele ser el caso más habitual. Esto deriva en que los tiempos de carga excedan los 2 segundos, incumpliéndose así los requisitos de rendimiento establecidos anteriormente, afectando directamente la eficiencia de la interfaz en situaciones de alto volumen.

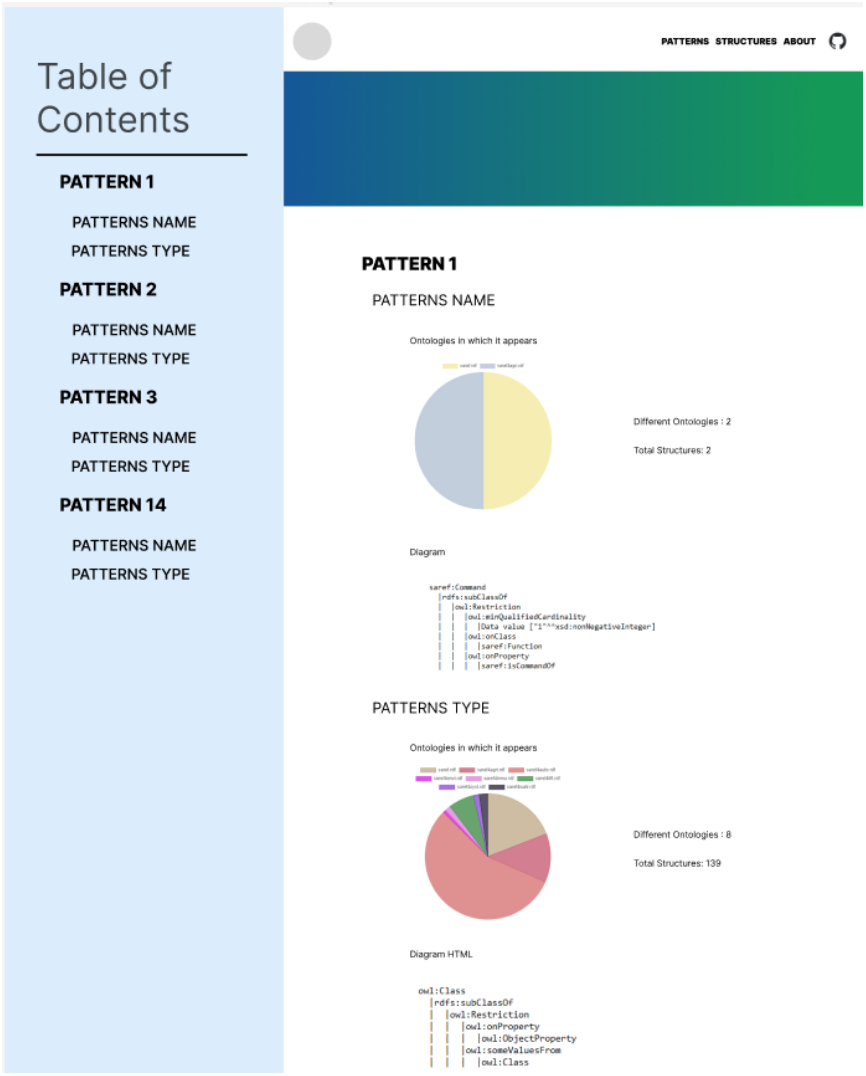


Ilustración 3. Vista Patrones: Diseño 2

3.2.1.3 Diseño 3

Como se muestra en la Ilustración 4, este diseño divide la información en dos vistas separadas, una dedicada a Pattern Name y otra a Pattern Type. En ambas vistas se muestran los patrones con los datos leídos en los .txt y.csv correspondientes y además en la vista de Pattern Type se muestra la imagen del diagrama .svg.

Tras las pruebas de usabilidad se ha observado que la separación en dos vistas diferentes facilita que los usuarios se enfoquen en un tipo de información, reduciendo así la sobrecarga cognitiva y mejorando la claridad.

Además, al dividir las vistas, cada página carga solo los recursos que necesita y como resultado se obtiene un mejor rendimiento.

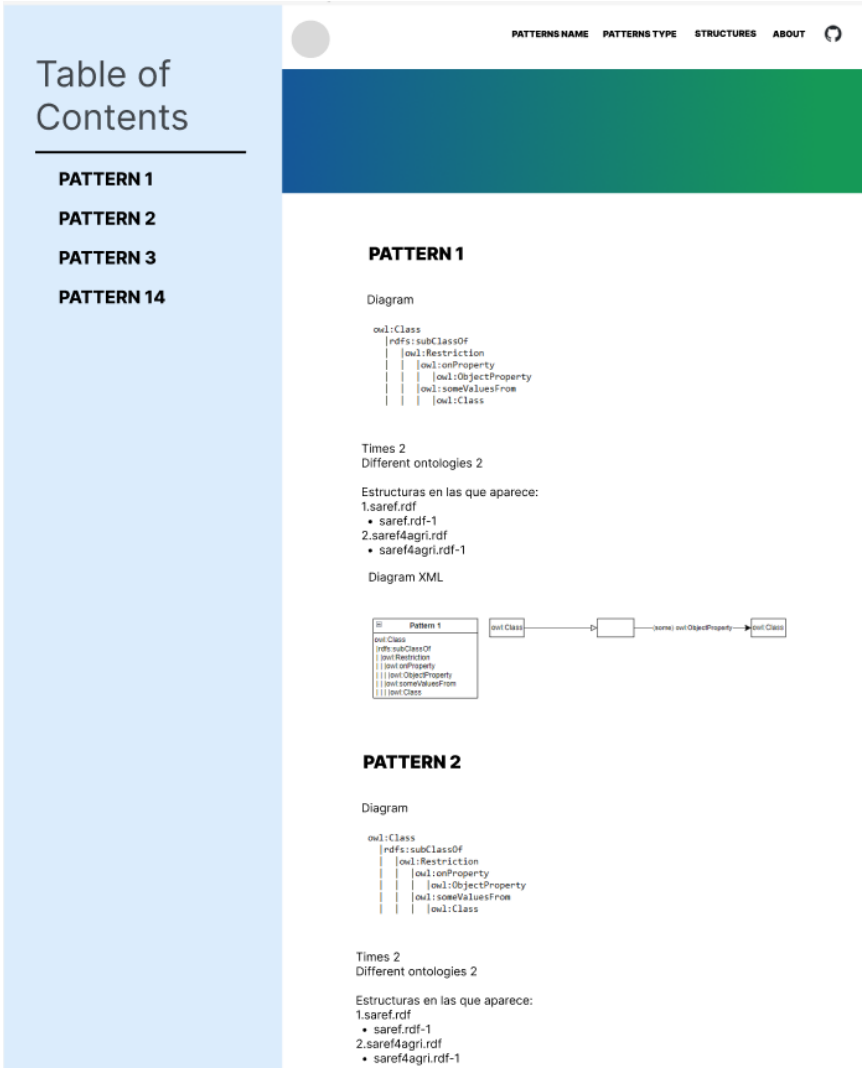


Ilustración 4. Vista Patrones: Diseño 3

3.2.1.4 Diseño Elegido

Tras analizar de forma exhaustiva todas las pruebas de usabilidad de cada uno de los diseños, el diseño 3 ha demostrado ser el más efectivo en términos de facilidad de uso, visibilidad y rendimiento. La estructura clara y la división de la información facilitan la interacción del usuario y mejoran la accesibilidad a los datos específicos.

3.2.2 Vista Estructuras

Para la visualización de las estructuras extraídas de Structure_term_inferred_type.txt y Structure_term_inferred_blank_nodes.txt también se han creado diferentes diseños.

3.2.2.1 Diseño 1

En este primer diseño, que se muestra en la Ilustración 5, se propone una visión de los datos de forma lineal y secuencial, es decir, se muestran unos debajo de otros. Al igual que en las vistas de los Pattern Name y Pattern Type también se ofrece una tabla de contenidos que permitan navegar entre las diferentes estructuras no solo para facilitar la navegación entre la información sino también para crear un diseño coherente y consistente entre las diferentes vistas de la página *web*.

Tras las pruebas de usabilidad se ha observado que, aunque la información se presenta de forma clara se muestra de una forma poca atractiva para el usuario.

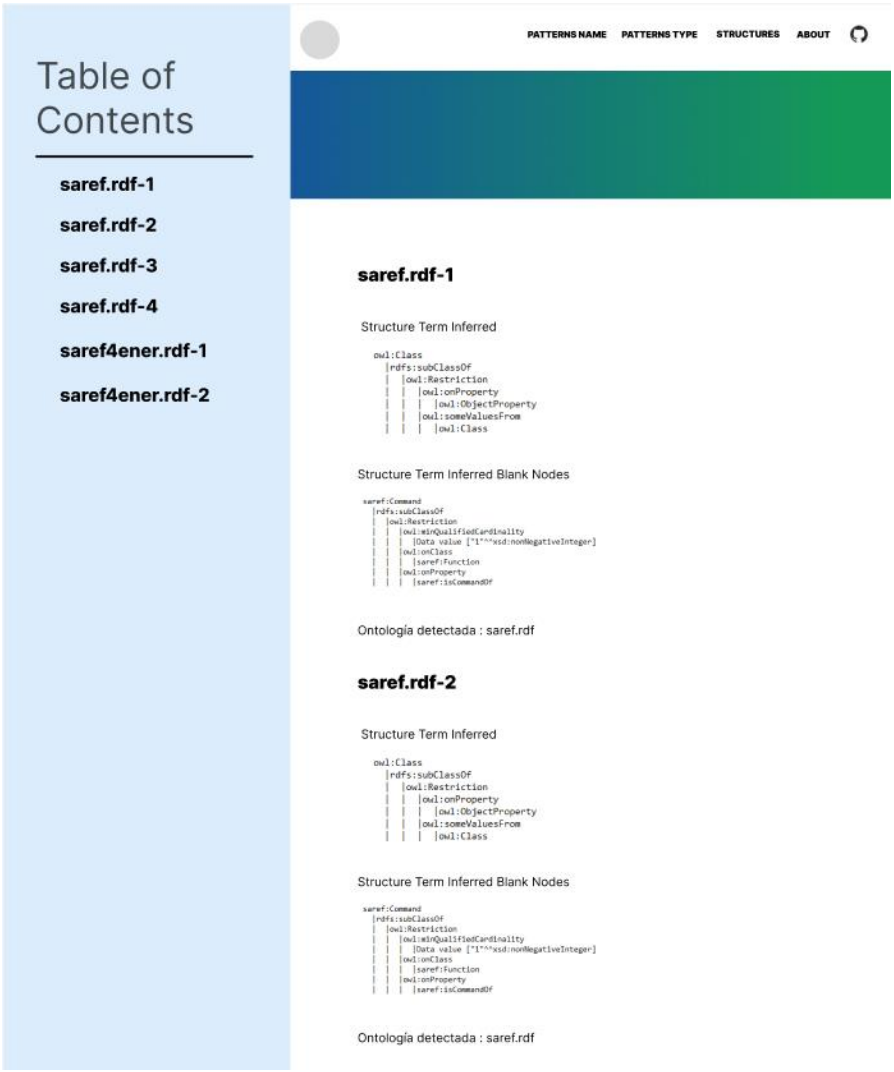


Ilustración 5. Vista Estructuras: Diseño 1

3.2.2.2 Diseño 2

Como se muestra en la Ilustración 6, en el segundo diseño, al igual que en el primero, se mantiene una tabla de contenidos para navegar entre las diferentes estructuras y asegurar un diseño coherente y consistente a través de las distintas vistas de la página *web*. La principal diferencia radica en la visualización de los diagramas de las estructuras. Para facilitar una visión conjunta, se ha optado por presentar esta información en tablas, haciéndola más atractiva y accesible para el usuario.

Tras las pruebas de usabilidad, se ha observado que el uso de tablas no solo mejora la estética de la presentación, sino que también hace la información más clara y concisa, permitiendo que los usuarios asimilen mejor los datos.

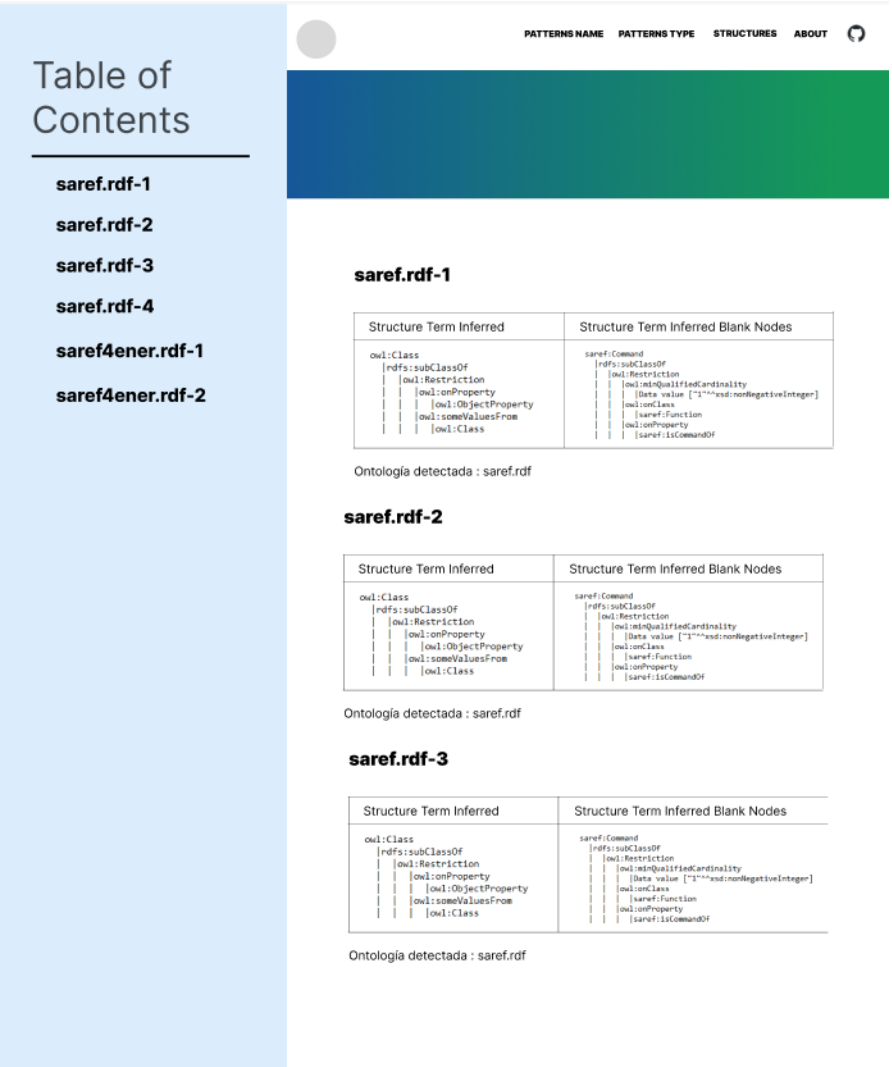


Ilustración 6. Vista Estructuras: Diseño 2

3.2.2.3 Diseño Elegido

Tras analizar de forma exhaustiva todas las pruebas de usabilidad de cada uno de los diseños, el diseño 2 ha demostrado ser el más efectivo en términos de facilidad de uso y visibilidad. La estructura clara y visual permiten tener una experiencia agradable para el usuario.

Su estructura tabular, complementada con elementos visuales claros y organizados, facilita la navegación y comprensión de los datos, lo que se traduce en una mejora significativa en la experiencia del usuario. Estas características aseguran una experiencia de usuario agradable y eficiente.

3.3 Arquitectura del Diseño

Como se puede observar en la Ilustración 7, la arquitectura de la aplicación *web* desarrollada en el presente TFG se basa en una solicitud/respuesta HTTP,

que se divide en dos componentes principales: la programación del lado del servidor (*Backend*) y la programación del lado del cliente (*Frontend*).

Programación del Lado del Servidor (Backend)

El usuario realiza una solicitud HTTP a través del navegador. Esta solicitud es recibida por la aplicación Flask en el *Backend*, que se encarga de su procesamiento. El archivo app.py dirige la solicitud a la ruta correspondiente definida en views.py. Este archivo es el encargado de llamar a las funciones definidas en utilities.py para leer y procesar los datos procedentes de los archivos de entrada.

Programación del Lado del Cliente (Frontend)

Una vez procesados los datos, se pasan a las plantillas HTML para su renderización. Estas plantillas, que utilizan el motor de plantillas Jinja2[7] de Flask, integran los datos dinámicamente en el HTML antes de enviarlo de vuelta al cliente. Finalmente, la página completa se envía de vuelta al navegador del usuario como respuesta HTTP.

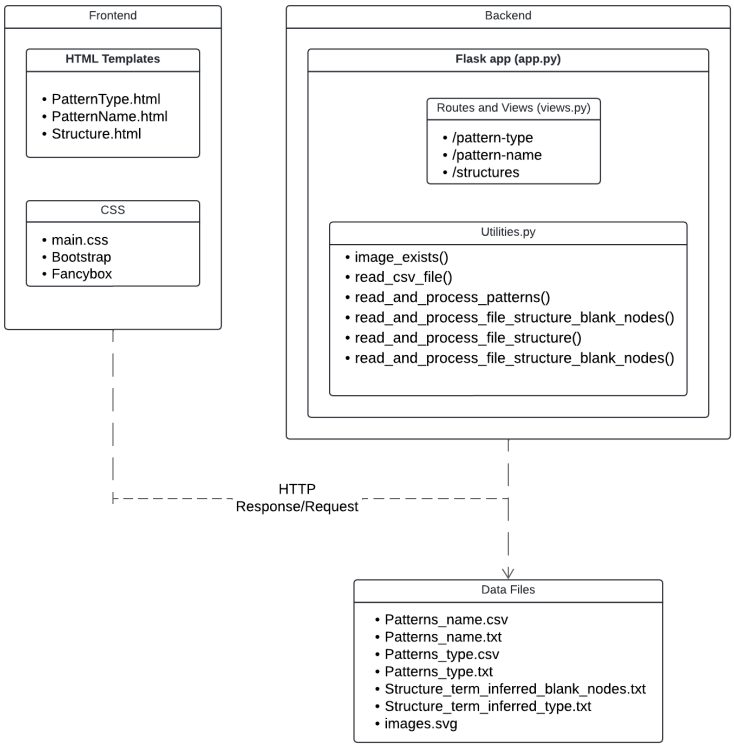


Ilustración 7: Arquitectura aplicación web

[7] Jinja, "Pallets Projects. Jinja API Documentation.", https://jinja.palletsprojects.com/en/3.1.x/api/

En la Ilustración 8, se muestra de forma más detallada cómo la aplicación *web* maneja una solicitud HTTP, procesa los datos necesarios y devuelve una respuesta al usuario. Este enfoque asegura que la aplicación sea eficiente y responda adecuadamente a las solicitudes de los usuarios, utilizando un diseño basado en rutas y funciones específicas para procesar distintos tipos de datos ontológicos.

1. **Recepción de la Solicitud**: El usuario realiza una solicitud HTTP desde su navegador, que es recibida por el archivo app.py en el Backend.

2. **Determinación de la Ruta**: app.py dirige la solicitud al archivo views.py, que determina la ruta específica de la solicitud.

3. **Evaluación de la Ruta**:

   - Si la ruta es **/pattern-name**: Se invocan las funciones read_csv_file, read_and_process_patterns y read_and_process_file_structure de utilities.py para procesar los datos de PatternName.txt y PatternName.csv. Los datos procesados se utilizan para renderizar la plantilla PatternName.html.

   - Si la ruta es **/pattern-type**: Se invocan las funciones image_exists, read_csv_file, read_and_process_file_structure_blank_nodes y read_and_process_patterns de utilities.py para procesar los datos de PatternType.txt, PatternType.csv, Pattern X.svg y Structure_term_inferred_blank_nodes.txt. Los datos procesados se utilizan para renderizar la plantilla PatternType.html.

   - Si la ruta es **/structures**: Se invocan las funciones read_and_process_file_structure y read_and_process_file_structure_blank_nodes de utilities.py para procesar los datos de Structure_term_inferred_blank_nodes.txt y Structure_term_inferred_type.txt. Los datos procesados se utilizan para renderizar la plantilla Structure.html.

4. **Procesamiento de Datos**: Los datos provenientes de los archivos de entrada son leídos y procesados por utilities.py utilizando funciones específicas para cada tipo de dato.

5. **Renderización y Respuesta**: Las plantillas HTML (PatternType.html, PatternName.html o Structure.html) se renderizan con los datos procesados. La página completa se envía de vuelta al navegador del usuario como respuesta HTTP.

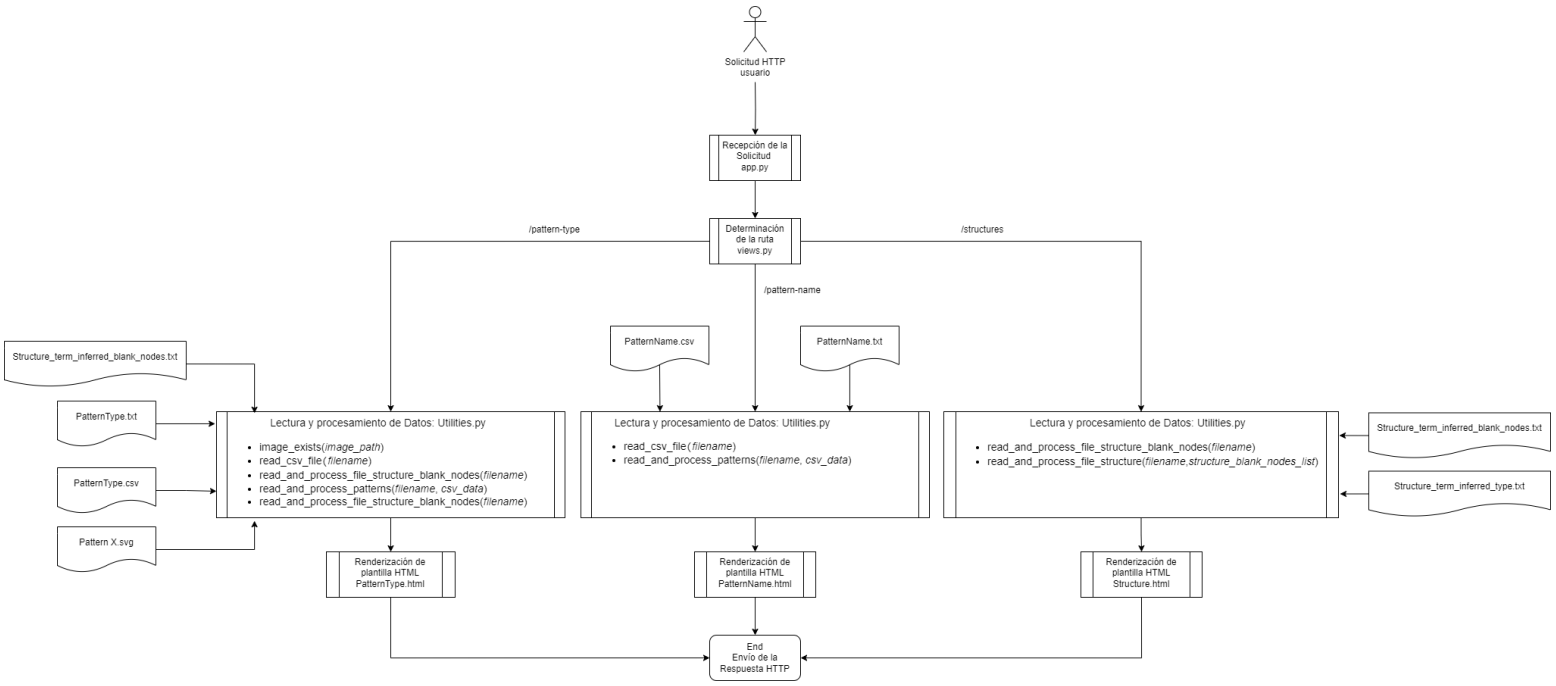


Ilustración 8: Worklow

3.4 Backend

En el siguiente apartado se describe como se ha llevado a cabo el desarrollo del *backend* de la aplicación *web*.

3.4.1 Tecnologías Utilizadas

Para el desarrollo del servidor de la aplicación, se ha seleccionado Flask[8], un *framework web* basado en Python. Flask es conocido por su simplicidad y flexibilidad, características que se alinean perfectamente con las necesidades

[8] Flask, "Quickstart — documentación de flask (3.0.x)", https://flask.palletsprojects.com/es/main/quickstart/

de nuestro proyecto. Esta elección permite un desarrollo ágil y eficiente, ideal para aplicaciones que requieren una base sólida con la capacidad de expandirse o modificarse fácilmente en el futuro.

Adicionalmente, dado que nuestra página *web* requiere la lectura y manipulación de archivos .txt y .csv, se ha optado por utilizar Python como lenguaje de programación. Python[9] destaca por su amplia colección de bibliotecas especializadas, tales como os y csv, que facilitan el manejo eficiente y efectivo de este tipo de archivos. Estas herramientas no solo simplifican las operaciones de lectura y escritura, sino que también proporcionan funcionalidades avanzadas para el procesamiento y análisis de datos.

Por otro lado, se ha hecho un uso extensivo de HTML[10] para formatear y representar la información. La utilización de HTML es crucial porque permite estructurar los datos procesados de manera que sean legibles y accesibles para los usuarios finales a través del navegador *web*. Integrar HTML en el *backend*, específicamente en combinación con Flask, facilita la creación de respuestas dinámicas. Por ejemplo, los *scripts* de Python en Flask pueden generar dinámicamente contenido HTML en función de los datos procesados, como listas de términos, tablas de contenidos, y visualizaciones de estructuras de datos que provienen de archivos .txt y .csv. Esto se logra mediante la construcción de cadenas HTML dentro del código Python, que luego se vuelcan en las plantillas de Flask o se envían directamente al cliente. Este enfoque garantiza que la presentación de los datos sea no solo funcional sino también estéticamente agradable, mejorando significativamente la experiencia del usuario al interactuar con la aplicación *web*.

3.4.2 Estructura del Código

Para optimizar la claridad y la modularidad del servidor se ha creado la siguiente estructura:

Archivos en el Directorio Raíz

- **app.py:** Es el archivo principal de la aplicación Flask. En él se inicializa la aplicación y todas las configuraciones necesarias.
- **utilities.py:** Contiene las funciones de lectura y procesamiento de los archivos para la generación del contenido HTML dinámico.
- **views.py:** En este archivo se definen las rutas y las vistas de la aplicación. Interactúa con utilities.py para procesar los datos y con las plantillas para generar las respuestas HTML.

[9] Python, “Python Software Foundation” , https://www.python.org/

[10] WHATWG, “HTML Living Standard”, https://html.spec.whatwg.org/multipage/

Directorios en el Directorio Raíz

- **/data:** En este directorio se almacenan lo archivos de entrada que la aplicación procesa para generar la respuesta HTML.
- **/env:** Este directorio contiene el entorno virtual de Python para el proyecto.
- **/static:** Este directorio contiene los archivos necesarios para el frontend de la aplicación. En él a su vez existen dos directorios:
  - **/images:** aquí se almacenan las imágenes.
  - **/styles:** aquí se almacena el archivo main.css que se utiliza para diseñar y dar estilo a la página *web*.
- **/templates:** Aquí se almacenan las plantillas HTML que se utilizan para crear la página *web*.

3.4.3 Código

A continuación, se detallan las partes más importantes de cada archivo del servidor.

3.4.3.1 app.py

En este archivo, como se puede ver en la Ilustración 9, se configura la aplicación *web* utilizando el *framework* Flask en Python. Inicialmente, se importa los siguientes módulos:

- **Flask** del módulo **flask**[11], que es núcleo del *framework* y se utiliza para crear instancias de la aplicación.

- **setup_routes** de un módulo local llamado **views**, que se utiliza para definir las rutas de la aplicación y asociar funciones a diferentes *endpoints* URL.

- **serve** del módulo **waitress**[12], que se trata de un servidor de producción recomendado para aplicaciones Flask que ofrece una alternativa más robusta al servidor de desarrollo incluido en Flask.

[11] Flask, "Pallets Projects. Flask: Patterns for packages.", https://flask.palletsprojects.com/es/main/patterns/packages/

[12] Flask, "Pallets Projects. Flask: Deploying with Waitress.", https://flask.palletsprojects.com/en/3.0.x/deploying/waitress/

Luego, se crea una instancia de la aplicación Flask con `app = Flask(__name__)`, donde __name__ es una variable de Python que indica el nombre del módulo en el que se está ejecutando el *script*.

La función `setup_routes(app)` se utiliza para configurar las rutas de la aplicación. Esta función define varios *endpoints* que la aplicación manejará para las distintas vistas.

El bloque condicional `if __name__ == '__main__':` verifica si el *script* se está ejecutando como programa principal y no siendo importado como módulo. Si es el caso, `serve(app,host='0.0.0.0',port=8000)` arranca el servidor configurado para escuchar en todas las interfaces de red (host='0.0.0.0') y en el puerto 8000. Esto indica que el servidor puede ser accesible desde cualquier dispositivo en la red, no solo localmente.

El código comentado `#app.run(debug=True)` arranca el servidor de desarrollo de Flask en el puerto predeterminado 5000, con el modo depuración activado, lo que facilita la visualización de errores y la recarga automática del servidor ante cambios en el código.

```
1  from flask import Flask
2  from views import setup_routes
3  #from waitress import serve
4
5  app = Flask(__name__)
6
7
8  setup_routes(app)
9
10 if __name__ == '__main__':
11     app.run(debug=True)
12     #serve(app, host='0.0.0.0', port=8000)
13
```

Ilustración 9. app.py

3.4.3.2 utilities.py

En este archivo se encuentran todas las funciones de procesamiento de los archivos de entrada, a continuación, se describen todas las funciones presentes en él.

Para poder procesar los diferentes archivos se han importado las siguientes librerías:

- **Lib/os.py [13] :** este módulo proporciona una manera de utilizar funcionalidades dependientes del sistema operativo, como la manipulación de rutas de archivos y directorios, y la ejecución de comandos del sistema.

- **Lib/csv.py[14]:** este módulo permite la lectura y escritura de archivos en formato CSV. El módulo csv simplifica el proceso de manipular estos archivos, permitiendo tanto la lectura como la escritura de forma que los datos pueden ser procesados o producidos sin necesidad de manejar manualmente la separación de campos y otros problemas típicos de formato.

3.4.3.2.1 Image_exists(*image_path*)

Esta función, como se observa en la Ilustración 10, es una función muy simple que tiene como entrada la ruta de una imagen y retorna un booleano, *true* si existe la imagen en la ruta o *false* en caso contrario. Esta función ha sido creada para la claridad del código.

```
6   # Función para examinar si una imagen existe en la ruta
7   def image_exists(image_path):
8       return os.path.exists(image_path)
9
```

Ilustración 10. Función image_exists

3.4.3.2.2 read_csv_file(*filename*)

Esta función, que se puede observar en la Ilustración 11, se emplea para leer archivos CSV y organizar su contenido en un diccionario en Python. La función abre el archivo y lo lee línea por línea, usando un punto y coma como delimitador (línea 16). Cada fila es almacenada en un diccionario, donde la primera columna de cada fila se utiliza como clave (líneas 19) y la fila completa como valor (línea 22). Si la clave aún no existe en el diccionario, se inicializa una nueva lista para esa clave antes de añadir la fila (líneas 20-21).

---

[13] Python Documentation, "Python Software Foundation. os — Miscellaneous operating system interfaces.", https://docs.python.org/es/3.10/library/os.html

[14] Python Documentation, "Python Software Foundation. csv — CSV File Reading and Writing.", https://docs.python.org/es/3/library/csv.html

Esto facilita la organización y el acceso a los datos basados en claves únicas, mejorando la eficiencia del manejo de los datos en las siguientes funciones.

```python
# Función para leer datos de un archivo CSV y convertirlos en una lista de diccionarios
def read_csv_file(filename):
    path = os.path.join(current_app.root_path, 'data', filename)
    data = {}
    try:
        with open(path, newline='', encoding='utf-8') as csvfile:
            reader = csv.reader(csvfile, delimiter=';')
            for row in reader:
                if row:
                    pattern_key = row[0].strip()
                    if pattern_key not in data:
                        data[pattern_key] = []
                    data[pattern_key].append(row)
    except FileNotFoundError:
        return "File not found."
    return data
```

Ilustración 11. Función read_csv_file

3.4.3.2.3 read_and_process_patterns(*filename, csv_data*)

Esta función se encarga de procesar los archivos .txt de los patrones ontológicos y devuelve un diccionario, que tiene como clave el nombre del patrón y como valor un array de los diferentes datos informativos del patrón ontológicos recogidos de los ficheros de entrada, y una lista de cabeceras con los nombres de los patrones. A continuación, se describen las partes más relevantes de la función.

Esta función, que se puede observar en la Ilustración 12, se encarga de procesar los archivos .txt de los patrones ontológicos y crear el HTML para las plantillas de las vistas de los Patterns. A continuación, se describen las partes más relevantes de la función.

Como se puede observar en la Ilustración 12, esta primera parte de la función abre el fichero .txt pasado como parámetro de entrada `filename` y divide el contenido en los diferentes patrones existentes en el fichero haciendo uso de la función `Split()` (línea 49). A continuación, por cada patrón comenzamos a guardar en el diccionario el patrón detectado y a procesar los datos leídos para extraer la información relevante y guardarla en el diccionario.

```
42      try:
43          if not os.path.exists(path):
44              return {"error": f"The file {filename} does not exist."}, []
45          with open(path, 'r', encoding='utf-8') as file:
46              content = file.read()
47              if not content:
48                  return {"error": f"The file {filename} is empty."}, []
49              patterns = content.split("Pattern ")
50              if not patterns[0] :
51                  patterns.pop(0)
52
53              # guardar los datos del txt
54              for index, pattern in enumerate(patterns, start=1):
55                  pattern_key = f"Pattern {index}"
56                  header_list.append(pattern_key)
57                  lines = pattern.split('\n')
58                  found_owl_class_section = False
59                  diagram=""
60                  for line in lines:
61                      line = line.strip()
62                      if line.startswith("Ontologies in which it appears"):
63                          ontologiesAppears = line.replace("Ontologies in which it appears","")
64                          found_owl_class_section = True
65                      elif found_owl_class_section and line:
66                          diagram += line + "<br>"
67                      elif "Times" in line:
68                          times = line;
69                      elif "Different ontologies" in line:
70                          ontologies= line
```

Ilustración 12. Función read_and_process_patterns

El siguiente fragmento de código, que se puede observar en la Ilustración 13, se encarga de integrar los datos adicionales provenientes de un archivo .csv. Para ello, primero se añade una entrada en el diccionario `value_csv` con la ontología recogida de la variable `ontologiesAppears` como clave y una lista vacía como valor (líneas 79-83).

A continuación, se verifica si existen datos para ese patrón en el diccionario `csv_data` (línea 85), que se pasa como parámetro de entrada a la función.

Si los datos están disponibles, se itera por cada clave del diccionario `csv_data` y por cada estructura contenida a partir de fila 3 de la lista del valor de la clave (líneas 87-88) y a continuación, se comprueba si existe una clave y se añade la estructura al diccionario `value_csv` (líneas 892-96).

```
72              # guardar el texto del csv
73                  #restaurar variables
74                  csv=[]
75                  value_csv={}
76
77                  #Separamos por ; y añadimos las claves al diccionario value_csv
78                  keyNameCsv = ontologiesAppears.split(";")
79                  for key in keyNameCsv:
80                      key = key.strip()
81                      if key:
82                          value_csv[key] = []
83
84                  if pattern_key in csv_data:
85                      for csv_row in csv_data[pattern_key]:
86                          for structure in csv_row[3:]:
87                              aux_structure = structure.split("-")[0]
88                              # comprobamos si la stucture:
89                              #  está contenida en el nombre de alguna key del diccionario
90                              for key in value_csv.keys():
91                                  # si está contenida metemos la structure como valor de dicha key
92                                  if aux_structure in key:
93                                      value_csv[key].append(structure)
94                                      break
95                      csv.append(value_csv)
96                  else:
97                      value_csv = {"No data": ["No CSV data found for this pattern."]}
98
```

Ilustración 13. Función read_and_process_patterns

Este fragmento de código que se observa en la Ilustración 14, verifica si el archivo procesado es 'Patterns_type.txt' y, de ser así, intenta vincular cada patrón con una imagen SVG correspondiente nombrada según el pattern_key. Utiliza la función `image_exists` (línea 104) para comprobar la existencia de la imagen en la ruta adecuada. Si la imagen está disponible, se guarda la ruta de la imagen en la variable `image_file.`

Al final, la función añade toda la información extraída del patrón al diccionario de salida (líneas 108-113), y al terminar de recorrer todos los patrones contenidos en el archivo .txt la función termina retornando el diccionario de salida y la lista de cabeceras.

```
99              # Verificar y añadir imagen
100                 if filename == "Patterns_type.txt":
101                     image_file = f"{pattern_key}.svg"
102                     image_path = os.path.join(current_app.root_path, 'static', 'images', image_file)
103                     image_file=url_for("static", filename=f"images/{image_file}")
104                     if not image_exists(image_path):
105                         image_file = 'No image available for this pattern.'
106 
107             #añadir al diccionario de salida
108                 if filename == "Patterns_type.txt":
109                     content =[diagram,times,ontologies,csv,image_file]
110                     data.update({pattern_key:content})
111                 else:
112                     content =[diagram,times,ontologies,csv]
113                     data.update({pattern_key:content})
114     except FileNotFoundError:
115         return {"error": f"The file {filename} does not exist."}, []
116 
117     return data,header_list
```

Ilustración 14. Función read_and_process_patterns

3.4.3.2.4 read_and_process_file_structure_blank_nodes(*filename*)

La función read_and_process_file_structure_blank_nodes, que se puede leer en la Ilustración 15~~Ilustración 15~~, está diseñada para leer y procesar datos desde el archivo Structure_term_inferred_blank_nodes, especificado por el parámetro `filename`. Si el archivo no se encuentra en la ruta, retorna un mensaje indicando que el archivo no fue encontrado. En caso contrario, la función busca una línea que comienza con "Ontology" (línea 135-137), marcando el inicio de la información que se va a guardar. A continuación, busca la etiqueta "Structure:", captura la clave de la estructura (línea 139) y procesa las líneas siguientes como contenido de esa estructura, almacenándolas en la variable `processed_content`.

Cada estructura procesada y su contenido asociado se almacenan en un diccionario `structure_list` haciendo uso de la función `append()` cada vez que nos encontremos una línea en blanco (líneas 144-147), puesto que esto significa que se ha terminado la sección de esa estructura en el txt, que es luego devuelto al finalizar la función.

Este método asegura una lectura ordenada y la estructuración de datos, facilitando su posterior manipulación o visualización.

```python
# Función para leer Structure Blank Nodes
def read_and_process_file_structure_blank_nodes(filename):
    path = os.path.join(current_app.root_path, 'data', filename)
    try:
        if not os.path.exists(path):
            return {"error": f"The file {filename} does not exist."}
        with open(path, 'r', encoding='utf-8') as file:
            lines = file.readlines()
            if not lines:
                return {"error": f"The file {filename} is empty."}
        processed_content = ""
        structure_key = ""
        structure_list = {}
        foundFirstParagraph = False
        foundLineHeader = False

        for line in lines:
            line = line.strip()
            if line.startswith("Ontology"):
                foundFirstParagraph = True
                processed_content = ""
            elif foundFirstParagraph:
                if line.startswith("Structure:"):
                    structure_key = line.split(":")[1].strip()
                    foundLineHeader = True
                elif foundLineHeader and line:
                    processed_content += line + "<br>"
                elif line == "":
                    structure_list.update({structure_key: processed_content})
                    foundFirstParagraph = False
                    foundLineHeader = False
    except FileNotFoundError:
        return {"error": f"The file {filename} does not exist."}
    return structure_list
```

Ilustración 15. Función read_and_process_file_structure_blank_nodes

3.4.3.2.5 read_and_process_file_structure(*filename,structure_blank_nodes_list*)

Esta función recibe como parámetros de entrada el fichero de texto Structure_term_inferred_type y un diccionario de estructuras, obtenido de la función *read_and_process_file_structure_blank_nodes*. Si el archivo no se encuentra, retorna un mensaje de error.

Si existe el fichero por cada línea leída del archivo TXT busca la definición de una estructura ontológica (línea 172), guarda el nombre de la ontología e identifica que ha encontrado el primer párrafo para comenzar a extraer los datos. Una vez que extrae todos los datos y encuentra una fila vacía guarda

los datos en el diccionario de salida `data` y pasa a buscar la siguiente estructura ontológica.

Al final, la función devuelve diccionario y la lista de encabezados que se han ido guardando en una lista `header_list` para la navegación futura. El código se puede observar en la Ilustración 16~~Ilustración 16~~.

```python
# Función leer un archivo txt y devolver un diccionario y una lista de las cabeceras
def read_and_process_file_structure(filename, structure_blank_nodes_list):
    path = os.path.join(current_app.root_path, 'data', filename)
    try:
        if not os.path.exists(path):
            return {"error": f"The file {filename} does not exist."}, []
        with open(path, 'r', encoding='utf-8') as file:
            lines = file.readlines()
            if not lines:
                return {"error": f"The file {filename} is empty."}, []

        data = {}
        content = []
        ontology_name = ""
        diagram_inferred_type = ""
        header_list = []
        foundFirstParagraph = False
        foundLineHeader = False

        for line in lines:
            line = line.strip()
            if line.startswith("Ontology"):
                ontology_name = line.split(":")[1].strip()
                foundFirstParagraph = True
            elif foundFirstParagraph:
                if line.startswith("Structure:"):
                    structure_key = line.split(":")[1].strip()
                    foundLineHeader = True
                    header_list.append(structure_key)
                elif foundLineHeader and line:
                    diagram_inferred_type += line + "<br>"
                elif line == "":
                    foundFirstParagraph = False
                    foundLineHeader = False
                    if structure_key in structure_blank_nodes_list:
                        blank_nodes_content = structure_blank_nodes_list[structure_key]
                    else:
                        blank_nodes_content = "No data"
                    content = [diagram_inferred_type, blank_nodes_content, ontology_name]
                    data.update({structure_key: content})
                    diagram_inferred_type = ""
    except FileNotFoundError:
        return {"error": f"The file {filename} does not exist."}, []
    return data, header_list
```

Ilustración 16. Función read_and_process_file_structure

3.4.3.3 views.py

En este archivo, como se puede leer en la Ilustración 17, se define la función `setup_routes` que configura las rutas de la aplicación Flask para manejar las diferentes vistas de la interfaz *web*.

En la primera línea del código se importa `render_template` del módulo flask[15], que se trata de una función proporcionada por Flask que facilita la integración del *backend* de Python con el *frontend* HTML. Permite retornar páginas HTML en respuesta a solicitudes en las aplicaciones *web*, utilizando plantillas dinámicas que pueden incorporar datos de python. Y además se importa `redirect`, que se utiliza para redirigir a los usuarios a una URL diferente, y `url_for, que` se utiliza para construir URLs dinámicamente para las rutas definidas en la aplicación Flask.

En la siguiente línea del código se importan todas las funciones que se han definido en el archivo utilities.py.

En las siguientes líneas se define la función `setup_routes`, en cada una de las rutas se hacen uso de las funciones que se necesitan para procesar la información que se va a volcar en esa página de la *web*. Para ello primero se llama a cada una de las funciones necesarias y a continuación, se comprueba que no haya ningún mensaje de error, en caso de haber algún mensaje de error se renderiza y se devuelve la plantilla con el error (líneas 15, 18, 27, 35 y 39), en el caso contrario se renderiza y se devuelve la plantilla con el contenido y la tabla de contenidos (líneas 20, 29 y 41).

[15]Flask, "Pallets Projects. Flask: Quickstart - Rendering Templates.", https://flask.palletsprojects.com/en/2.3.x/quickstart/#rendering-templates

```python
from flask import render_template, redirect, url_for
from utilities import read_csv_file, read_and_process_patterns,read_and_process_file_structure,read_and_process_file_structure_blank_nodes

def setup_routes(app):
    @app.route('/')
    def home():
        return redirect(url_for('pattern_types'))

    @app.route('/pattern-type')
    def pattern_types():
        csv_type_data = read_csv_file('Patterns_type.csv')
        pattern_content_type,header_list = read_and_process_patterns('Patterns_type.txt', csv_type_data)
        content_blank_nodes=read_and_process_file_structure_blank_nodes('Structure_term_inferred_blank_nodes.txt')
        if "error" in pattern_content_type:
            return render_template('PatternType.html', pattern_content_type={}, header_list=[],
                                   content_blank_nodes={}, error_message=pattern_content_type["error"],error_message_structure=None)
        if "error" in content_blank_nodes:
            return render_template('PatternType.html', pattern_content_type=pattern_content_type, header_list=header_list,
                                   content_blank_nodes={}, error_message=None, error_message_structure=content_blank_nodes["error"])
        return render_template('PatternType.html', pattern_content_type=pattern_content_type, header_list=header_list,
                               content_blank_nodes=content_blank_nodes, error_message=None,error_message_structure=None)

    @app.route('/pattern-name')
    def pattern_name():
        csv_name_data = read_csv_file('Patterns_name.csv')
        pattern_content_name,header_list = read_and_process_patterns('Patterns_name.txt', csv_name_data)
        if "error" in pattern_content_name:
            return render_template('PatternName.html', pattern_content_name={}, header_list=[],error_message=pattern_content_name["error"])
        return render_template('PatternName.html', pattern_content_name=pattern_content_name,header_list=header_list, error_message=None)

    @app.route('/structures')
    def structure():
        content_blank_nodes = read_and_process_file_structure_blank_nodes('Structure_term_inferred_blank_nodes.txt')
        if "error" in content_blank_nodes:
            return render_template('Structure.html', error_message=content_blank_nodes["error"], content_blank_nodes={},
                                   content_type={}, header_list=[])
        content_type, header_list = read_and_process_file_structure('Structure_term_inferred_type.txt', content_blank_nodes)
        if "error" in content_type:
            return render_template('Structure.html', error_message=content_type["error"], content_blank_nodes=content_blank_nodes,
                                   content_type={}, header_list=[])
        return render_template('Structure.html', content_type=content_type, content_blank_nodes=content_blank_nodes,
                               header_list=header_list, error_message=None)
```

Ilustración 17. views.py

3.5 Frontend

En el siguiente apartado se describe como se ha llevado a cabo el desarrollo de la interfaz de usuario.

3.5.1 Tecnologías Utilizadas

Para el desarrollo de la interfaz del usuario, se ha utilizado HTML para crear la estructura básica de las diferentes vistas que se hacen uso en la página *web*. EL lenguaje HTML proporciona el esqueleto básico para construir los elementos visuales y funcionales.

Para definir el estilo de la página *web* y mostrar la parte visual de la página de forma atractiva para el usuario se ha hecho uso del lenguaje CSS. El lenguaje CSS permite personalizar detalladamente los elementos HTML, desde el color y tamaño del texto hasta la posición y comportamiento responsivo de los diferentes elementos. Además, para dar estilo al uso de las imágenes se ha hecho uso de la librería de JQuery que se trata de una biblioteca de JavaScript.

3.5.2 Estructura del Código

Para maximizar la eficiencia y mantenibilidad del código de la interfaz del usuario se ha diseñado la siguiente estructura:

Directorios en el Directorio Raíz

- **/static:** Este directorio contiene los archivos necesarios para el *frontend* de la aplicación. En él a su vez existen dos directorios:
    - **/images:** aquí se almacenan las imágenes.
    - **/styles:** aquí se almacena el archivo **main.css** que se utiliza para diseñar y dar estilo a la página *web*.
- **/templates:** Aquí se almacenan las plantillas HTML que se utilizan para crear la página *web*.
    - **PatternType.html:** contiene la plantilla HTML de los tipos de los términos de los diferentes patrones. Incluye una tabla de contenido, secciones de navegación, y un área principal donde se visualizan los diagramas y las imágenes correspondientes a cada tipo de patrón. También presenta información detallada sobre el número de ontologías y las estructuras en las que aparecen estos patrones.
    - **PatternName.hmtl:** Similar a la anterior, contiene la plantilla HTML de los URIs de los términos de los diferentes patrones. Incluye una tabla de contenido, secciones de navegación, y un área principal donde se visualizan los diagramas correspondientes a cada tipo de patrón. También presenta información detallada sobre el número de ontologías y las estructuras en las que aparecen estos patrones.
    - **Structure.html:** Contiene la plantilla HTML de las estructuras ontológicas. Incluye una tabla de contenido, secciones de navegación, y un área principal donde se presentan tablas comparativas de términos inferidos de estructuras y nodos en blanco. Además, muestra información sobre las ontologías detectadas, proporcionando una vista detallada y organizada de las estructuras ontológicas completas

Estas plantillas se integran con los datos procesados por utilities.py, permitiendo que la información se renderice dinámicamente en el navegador del usuario. Al usar Jinja2[16] para la renderización de plantillas, se asegura

[16]Pallets Projects. (n.d.), "Jinja API Documentation",

que los datos se presenten de manera coherente y estructurada, mejorando la experiencia del usuario al interactuar con la plataforma.

3.5.3 Código

Para asegurar una experiencia de usuario coherente y facilitar el mantenimiento del sitio, las plantillas HTML de la aplicación *web* comparten una estructura básica común. A continuación, se muestra la estructura común que comparten las 3 plantillas HTML. Esta estructura estándar incluye elementos clave como la barra de navegación, los estilos CSS, que son idénticos en cada plantilla, con la única variación siendo los datos específicos que se muestran, que se explicarán de forma individual más adelante.

El código, como se puede observar en la Ilustración 18, comienza con `<!DOCTYPE html>`, necesario para asegurar que el navegador procese la página como HTML5[17]. La etiqueta `<html lang="en">` indica que el contenido del sitio está en inglés, importante para la accesibilidad y los motores de búsqueda.

La siguiente sección del código incluye etiquetas meta que son esenciales para la codificación y la responsividad del sitio *web*[18]. La etiqueta `<meta charset="UTF-8"/>` asegura que el contenido se codifique correctamente, mientras que la etiqueta `<meta name="viewport" content="width=device-width, initial-scale=1.0"/>` garantiza que el sitio sea responsivo y se vea bien en dispositivos de diferentes tamaños.

```
1   <!DOCTYPE html>
2   <html lang="en">
3     <head>
4       <meta charset="UTF-8" />
5       <meta name="viewport" content="width=device-width, initial-scale=1.0" />
```

Ilustración 18. Tipo de documento y metadatos

---

https://jinja.palletsprojects.com/en/3.1.x/api/

[17]W3Schools. (n.d.), “HTML Introduction”, https://www.w3schools.com/html/html_intro.asp

[18]W3Schools. (n.d.), “CSS RWD Viewport”, https://www.w3schools.com/css/css_rwd_viewport.asp

La siguiente sección del código se trata de las importaciones a los diferentes recursos externos que se hace a través de la etiqueta `<link>`[19] y la etiqueta `<script>`, a continuación, se detallan las importaciones más relevantes:

- **CSS de Bootsrap**[20]: Bootsatrap se trata de un *framework* de CSS que facilita el diseño de interfaces *web* responsivas y estéticas. Proporciona una amplia biblioteca de estilos predefinidos que ayudan en la rápida implementación de componentes estilizados.

  En el presente proyecto se importa para estilizar la interfaz del sitio y asegurar que sea responsiva, garantizando que se adapte a diferentes tamaños de pantalla

- **JavaScript de Bootstrap** [21] **:** Este *script* complementa el CSS de Bootstrap al añadir interactividad a los componentes que lo requieren.

  En nuestro proyecto se utiliza para habilitar las características dinámicas de los componentes de Bootstrap en el sitio *web*.

- **main.css:** Este archivo ha sido creado para contener las reglas CSS específicas del proyecto. Se utiliza para aplicar estilos adicionales que no se incluyen en Bootstrap, permitiendo una personalización más detallada del estilo de la aplicación *web*.

- **Fancybox CSS** [22] **:** Fancybox es una biblioteca ligera que permite mostrar imágenes, videos y más en un *lightbox* que se superpone sobre la página activa.

  En nuestro proyecto se utiliza para enriquecer la interacción con las imágenes que se muestran en la página Pattern Type. Al hacer clic en las imágenes de patrones, Fancybox las abre en un *lightbox* sobre la página actual, evitando la necesidad de navegar fuera de la página o abrir una nueva pestaña. Esto mantiene al usuario en el mismo contexto mientras explora los detalles visuales de los patrones.

---

[19]W3Schools. (n.d.), “HTML HTML ‘<link>’ Tag”, https://www.w3schools.com/tags/tag_link.asp

[20]Bootstrap, "Twbs Bootstrap. Introduction to CSS.", https://getbootstrap.com/docs/3.4/css/

[21]Bootstrap, “Twbs Bootstrap. Introduction to JavaScript.”, https://getbootstrap.com/docs/3.4/javascript/

[22]FancyApps, "FancyApps. Fancybox Documentation.", https://fancyapps.com/fancybox/

- **jQuery**[23]**:** jQuery es una biblioteca de JavaScript que simplifica la manipulación del Document Object Model (DOM), el manejo de eventos, y las interacciones AJAX.

  En nuestro proyecto es fundamental para facilitar el uso de Fancybox y otros plugins que dependen de jQuery.

- **Fancybox JavaScript:** Proporciona la funcionalidad JavaScript necesaria para inicializar y controlar el comportamiento de los lightboxes de Fancybox.

  En nuestro proyecto se emplea para implementar funcionalidades avanzadas de visualización de las imágenes, como descargar, compartir y ver en pantalla completa.

Las últimas 3 importaciones únicamente se realizan en la plantilla PatternType.html, puesto que es en la única página que se hace uso de datos multimedia.

El siguiente bloque de código HTML es responsable de generar la tabla de contenidos en la página *web*. Esta tabla de contenidos permite a los usuarios navegar rápidamente a diferentes secciones de la página. Como se puede observar en la Ilustración 19, para crear la tabla de contenidos se utiliza un bucle de Jinja2 que itera sobre cada elemento encontrado en la lista `header_list`.

```
47      <div id='toc' class='col-3 pt-4 d-none d-xl-block bg-light px-5'>
48        <div class="container">
49          <h4 class="options text-secundary pb-2">Table of Contents</h4>
50          {%for header in header_list %}
51          <p><a href="#{{header}}">{{header}}</a></p>
52          {% endfor %}
53        </div>
54      </div>
```

Ilustración 19. Tabla de contenidos dinámica

Tras la tabla de contenidos, como se puede observar en la Ilustración 20, se define el elemento Navbar haciendo uso de la etiqueta `<nav>`[24], que define la forma en que los usuarios navegan entre las diferentes secciones de la

[23]jQuery, "jQuery Foundation. jQuery.", https://jquery.com/

[24]W3Schools. (n.d.), "HTML '<nav>' Tag", https://www.w3schools.com/tags/tag_nav.asp

plataforma *web*. Utiliza `url_for` de Flask[25] para generar URLs dinámicamente, facilitando el mantenimiento y la reconfiguración de las rutas

```
<header>
  <nav class="navbar navbar-expand-lg navbar-light py-3 text-dark">
    <div class="container">
      <button
        class="navbar-toggler"
        type="button"
        data-bs-toggle="collapse"
        data-bs-target="#navbarNav"
        aria-controls="navbarNav"
        aria-expanded="false"
        aria-label="Toggle navigation"
      >
        <span class="navbar-toggler-icon"></span>
      </button>
      <div class="collapse navbar-collapse" id="navbarNav">
        <ul class="navbar-nav ms-auto fw-bold mx-4">
          <li class="nav-item">
            <a class="nav-link" href="{{ url_for('pattern_types') }}"
              >Patterns Type</a
            >
          </li>
          <li class="nav-item">
            <a class="nav-link" href="{{ url_for('pattern_name') }}"
              >Patterns Name</a
            >
          </li>
          <li class="nav-item">
            <a class="nav-link" href="{{ url_for('structure') }}"
              >Structures</a
            >
          </li>
        </ul>
      </div>
    </div>
  </nav>
</header>
```

Ilustración 20. Navegación

A continuación, se explicarán de forma individual la sección de contenido de cada una de las plantillas HTML.

[25]GeeksforGeeks. (n.d.), “Flask URL Helper Function: flask.url_for.”, https://www.geeksforgeeks.org/flask-url-helper-function-flask-url_for/

3.5.3.1 PatternType.html

Como se puede observar en la Ilustración 21, al igual que para la tabla de contenidos, se utiliza un bucle de Jinja2 (línea 107) para iterar sobre cada uno de los patrones y mostrar la información contenida en ellos. El filtro `|safe` (líneas 116 y 137) permite renderizar HTML seguro dentro de las variables, asegurando que el HTML generado dinámicamente se presente correctamente.

```
102     <div class="container">
103       {% if error_message %}
104       <div class="alert alert-danger" role="alert">{{ error_message }}</div>
105       {% endif %}
106       <section id="specification-ontologists" class="pt-5 px-5">
107         {% for key_pattern, pattern in pattern_content_type.items() %}{% if loop.first %}
108         <h4 id="{{ key_pattern }}">{{ key_pattern }}</h4>
109         {% else %}
110         <h4 id="{{ key_pattern }}" class="pt-5">{{ key_pattern }}</h4>
111         {% endif %}
112
113         <!--Diagram and image-->
114         <div id="container-patterntype">
115           <div id="container-diagram-ontologie">
116             <p><code>{{ pattern[0]|safe }}</code></p>
117           </div>
118           <div id="container-image-patterntype">
119             <p>
120               <!-- Image -->
121               {% if "No image available" in pattern[4] %}
122               <div class="alert alert-danger" role="alert">{{pattern[4]}}</div>
123               {% else %}
124               <a href="{{ pattern[4] }}" data-fancybox>
125                 <img src="{{ pattern[4] }}" alt="img type {{ pattern[4] }}" />
126               </a>
127               {% endif %}
128             </p>
129           </div>
130         </div>
131         <!--Times-->
132         <p>
133           {{ pattern[1] }}
134         </p>
135         <!--Number of ontologies-->
136         <p>
137           {{ pattern[2]|safe }}
138         </p>
```

Ilustración 21. Contenido 1 PatterType

Como se puede observar en la Ilustración 22, para mostrar el contenido de las estructuras ontológicas detectadas en el patrón se hace uso de elementos colapsables de Bootstrap [26]. Estos elementos permiten mostrar y ocultar información de manera dinámica, mejorando así la organización y accesibilidad del contenido. Para poder crear elementos únicos, se generan IDs únicos (líneas 147 y 157) añadiendo el número de indexación del bucle. Esto se hace porque las estructuras ontológicas pueden repetirse entre los distintos patrones, y si se utilizara únicamente el nombre de la estructura como ID del elemento, podría ocurrir que al hacer clic sobre una de las estructuras se desplegaran todas las instancias de esa estructura en todos los patrones que la contengan.

```
139     <!--List of structures-->
140     <p>Structures in which it appears:
141       {% if not pattern[3]%}
142         <div class="alert alert-danger" role="alert">No CSV data found for this pattern.</div>
143       {% else %}
144         {% for structures in pattern[3] %}
145         {% for key, value in structures.items()  %}
146             <!--Generamos un id único-->
147             {% set id = 'collapse_' ~ key_pattern.replace(' ', '_') ~ '_' ~ loop.index %}
148             <p class="d-inline-flex gap-1">
149                 <a class="btn custom-btn" data-bs-toggle="collapse" href="#{{ id }}" role="button" aria-expanded="false" aria-controls="{{ id }}">
150                     {{ key }}
151                 </a>
152             </p>
153             <div class="collapse" id="{{ id }}" style="margin-left: 20px;">
154                 {% for structure in value %}
155                   <!--<span class="badge custom-badge mb-4">{{ structure }}</span> -->
156                 {% set id_structure = 'collapse_' ~ structure.replace(".","_") ~ '_' ~ loop.index %}
157                 <p class="d-inline-flex gap-1">
158                   <a class="btn custom-btn-structure" data-bs-toggle="collapse" href="#{{ id_structure }}" role="button" aria-expanded="false" aria-controls="{{ id_structure }}">
159                     {{ structure }}
160                   </a>
161                 </p>
162                 <div class="collapse pb-4" id="{{ id_structure }}" style="margin-left: 20px;">
163                   {% if error_message_structure %}
164                   <div class="alert alert-danger" role="alert">{{ error_message_structure }}</div>
165                   {% else %}
166                   <code>{{ content_blank_nodes[structure] | safe}}</code>
167                   {% endif %}
168                 </div>
169                 {% endfor %}
170             </div>
171         {% endfor %}
172         {% endfor %}
173       {% endif %}
```

Ilustración 22. Componentes Bootstrap PatternType

Por último, en la Ilustración 23 se muestra el *script* de configuración de Fancybox, que es únicamente utilizado en PatternType.html. Este script es fundamental para personalizar la experiencia del usuario al interactuar con las imágenes de la página *web*. Se inicializa Fancybox en todos los elementos que contienen el atributo data-fancybox, aplicando una serie de opciones de configuración que mejoran la funcionalidad y seguridad de las visualizaciones de imágenes. A continuación, se detallan y explican las opciones configuradas:

- **loop:** Esta opción está configurada como *true*, lo que permite que la galería de imágenes o contenidos multimedia ciclen indefinidamente. Al llegar al final de una galería, el usuario puede continuar al principio sin

[26] Bootstrap, "Bootstrap. Collapse.", https://getbootstrap.com/docs/5.3/components/collapse/

necesidad de navegar hacia atrás, mejorando la fluidez de la experiencia de navegación.

- **width y height:** Se establecen al 60% del ancho y altura de la ventana del navegador, respectivamente. Esto asegura que el contenido se muestre de manera prominente, ocupando una cantidad significativa de espacio visual sin sobrepasar las dimensiones de la ventana, lo que ayuda a mantener un balance adecuado entre el contenido y el espacio de la interfaz.

- **buttons:** Se configura un arreglo de botones que incluye opciones como compartir (*share*), presentación de diapositivas (*slideShow*), pantalla completa (*fullScreen*), descarga (download), miniaturas (*thumbs*), y cerrar (*close*). Cada botón añade una funcionalidad específica que permite al usuario interactuar con el contenido de manera más dinámica y útil, desde compartir contenidos en redes sociales hasta descargar imágenes directamente desde el *lightbox*.

```
192     <script>
193       $("[data-fancybox]").fancybox({
194         // Opciones de configuración
195         buttons: [
196           "share",
197           "slideShow",
198           "fullScreen",
199           "download",
200           "thumbs",
201           "close",
202         ],
203         width: $(window).width() * 0.8, // 80% del ancho de la ventana
204         height: $(window).height() * 0.8, // 80% de la altura de la ventana
205       });
206     </script>
```

Ilustración 23. Configuración imágenes PatternType

3.5.3.2 PatternName.html

Esta plantilla es muy similar a la plantilla PatterType.html con la diferencia que no se muestran imágenes y que los elementos que se utilizan de bootstrap para mostrar las estructuras ontológicas detectadas en los patrones son las insignias o badges[27], como se puede observar en la Ilustración 24.

[27] Bootstrap, "Bootstrap. Badges.", https://getbootstrap.com/docs/5.3/components/badge/

```
<p>
  Structures in which it appears: <br />
  {% if not pattern[3]%}
    <div class="alert alert-danger" role="alert">No CSV data found for this pattern.</div>
  {% else %}
    {% for structures in pattern[3] %}
      {% for key, value in structures.items() %}
        {% for structure in value %}
          <span class="badge custom-badge mt-2">{{ structure }}</span>
        {% endfor %}
      {% endfor %}
    {% endfor %}
  {% endif %}
</p>
```

Ilustración 24. Componentes Bootstrap PatterName

3.5.3.3 Structure.html

Esta plantilla es creada para visualizar las estructuras ontológicas completas. Al igual que en las demás plantillas para mostrar los datos se hace uso del bucle Jinja2 para iterar sobre las distintas estructuras ontológicas y sus datos. Como se puede observar en la Ilustración 25, se hace uso de una tabla, donde se comparan términos inferidos de estructuras y nodos en blanco. La tabla está estilizada con clases de Bootstrap[28], la etiqueta `<table>` aplica el estilo básico de tabla de Bootstrap.

[28] Bootstrap, "Bootstrap. Tables.", https://getbootstrap.com/docs/4.1/content/tables/

```
92  <table class="table table-bordered align-middle pt-3">
93    <thead>
94      <tr>
95        <th scope="col" style="width: 50%">
96          Structure Term Inferred Type
97        </th>
98        <th scope="col" style="width: 50%">
99          Structure Term Inferred Blank Nodes
100       </th>
101     </tr>
102   </thead>
103   <tbody>
104     <tr>
105       <td>
106         <code> {{ value[0]|safe }} </code>
107       </td>
108       <td>
109         <code> {{ value[1]|safe }} </code>
110       </td>
111     </tr>
112   </tbody>
113 </table>
```

Ilustración 25. Tabla Estructuras Ontológicas Structure.html

Para cualquier consulta el código se encuentra disponible haciendo clic en el enlace de GitHub.

4 Pruebas

El desarrollo de cualquier aplicación de *software* implica no solo la implementación de funcionalidades diseñadas para cumplir con los requisitos específicos de un proyecto, sino también la validación de que estas funcionalidades funcionen correctamente en todos los escenarios posibles. Las pruebas de software son, por lo tanto, un componente crítico del proceso de desarrollo que asegura la calidad, la seguridad y la eficacia de la aplicación antes de su despliegue en un entorno de producción.

En este proyecto, el enfoque en las pruebas es multifacético, abarcando desde pruebas unitarias que verifican la funcionalidad a nivel de componentes individuales, hasta pruebas de sistema que evalúan la aplicación en su conjunto. Este enfoque estructurado no solo ayuda a identificar y corregir errores durante las primeras etapas del desarrollo, sino que también asegura que la aplicación cumpla con todos los requisitos funcionales y no funcionales establecidos, proporcionando una base sólida para su operación continua y mantenimiento.

El objetivo de este apartado es detallar el plan de pruebas implementado para el proyecto, describiendo las diferentes categorías de pruebas llevadas a cabo, incluyendo pruebas unitarias, de integración y de sistema.

4.1 Pruebas Unitarias

Este tipo de pruebas se centran en los componentes más pequeños de la aplicación, asegurando que cada función, método o clase funcione correctamente de manera aislada. Para este tipo se han añadido `print()`en las diferentes líneas del código para ver en consola los resultado de las funciones.

Identificador	#unit1
Descripción	Llamar a la función image_exists(image_path) con una ruta correcta de una imagen.
Resultado Esperado	La función debe devolver true.

Tabla 1. Prueba Unitaria 1

Identificador	#unit2
Descripción	Llamar a la función image_exists(image_path) con una ruta de una imagen que no existe.
Resultado Esperado	La función debe devolver false.

Tabla 2. Prueba Unitaria 2

Identificador	#unit3
Descripción	Llamar a la función read_csv_file(filename) con el nombre de un csv que no existe
Resultado Esperado	La función debe devolver “File not found”

Tabla 3. Prueba Unitaria 3

Identificador	#unit4
Descripción	Llamar a la función read_csv_file(filename) con un csv de prueba
Resultado Esperado	La función debe guardar correctamente los pares clave-valor y devolver el diccionario correctamente.

Tabla 4. Prueba Unitaria 4

Identificador	#unit5
Descripción	Llamar a la función read_and_process_patterns(filename, csv_data) con el nombre de un txt que no existe y un diccionario de datos
Resultado Esperado	La función debe devolver “File not found”

Tabla 5. Prueba Unitaria 5

Identificador	#unit6
Descripción	Llamar a la función read_and_process_patterns(filename, csv_data) con el archivo PatternName.txt existente y un diccionario de datos.
Resultado Esperado	La función debe devolver una variable de texto con todos los datos leídos del fichero de texto y del diccionario con todas las cabeceras en HTML, además debe devolver una lista con todas las cabeceras de los Patrones encontrados.

Tabla 6. Prueba Unitaria 6

Identificador	#unit7
Descripción	Llamar a la función read_and_process_patterns(filename, csv_data) con el archivo PatternType.txt existente y un diccionario de datos y además en la carpeta statics/images se encuentran todas las imágenes .svg de los patrones existen en el archivo txt.
Resultado Esperado	La función debe devolver una variable de texto con todos los datos leídos del fichero de texto y del diccionario con todas las cabeceras en HTML, incluyendo las imágenes .svg en cada uno de los patrones, además debe devolver una lista con todas las cabeceras de los Patrones encontrados.

Tabla 7. Prueba Unitaria 7

Identificador	#unit8
Descripción	Llamar a la función read_and_process_file_structure_blank_nodes(filename) con un nombre de archivo que no existe.
Resultado Esperado	La función debe devolver “File not found”

Tabla 8. Prueba Unitaria 8

Identificador	#unit9
Descripción	Llamar a la función read_and_process_file_structure_blank_nodes(filename)) con Structure_term_inferred_blank_nodes.txt. Este archivo existe en la capeta correspondiente.
Resultado Esperado	La función debe devolver un diccionario clave-valor, donde la clave son el nombre de las estructuras y el valor debe de ser el diagrama de las estructuras.

Tabla 9. Prueba Unitaria 9

Identificador	#unit10
Descripción	Llamar a la función read_and_process_file_structure(filename, structure_blank_nodes_list) con un nombre de archivo que no existe y un diccionario.
Resultado Esperado	La función debe devolver “File not found”

Tabla 10. Prueba Unitaria 10

Identificador	#unit11
Descripción	Llamar a la función read_and_process_file_structure(filename, structure_blank_nodes_list) con el nombre de archivo Structure_term_inferred_type.txt y un diccionario.
Resultado Esperado	La función debe devolver una variable de texto con todos los datos leídos del fichero de texto y del diccionario con todas las cabeceras en HTML, además debe devolver una lista con todas las cabeceras de las estructuras encontrados.

Tabla 11. Prueba Unitaria 11

Identificador	#unit12
Descripción	Llamar a la función create_table_of_contents(header_list) con una lista de nombres.
Resultado Esperado	La función debe devolver una variable de texto con las etiquetas HTML para crear la tabla de contenidos, las cabeceras de la tabla de contenidos deben ser los mismo que se han pasado en la lista de parámetro de entrada.

Tabla 12. Prueba Unitaria 12

4.2 Pruebas de Integración

Este tipo de pruebas examinan la cooperación entre varios módulos o servicios para detectar fallos en la interacción entre estos componentes integrados. Para este tipo de pruebas se ha utilizado la aplicación en modo debug, para mostrar por consola cualquier error.

Identificador	#int1
Descripción	Se ejecuta la aplicación con todas las rutas de archivos configuradas de forma correcta y se observan los datos mostrados en la página.
Resultado Esperado	La aplicación *web* debe de mostrar en cada una de las páginas toda la información correspondiente.

Tabla 13. Prueba de Integración 1

Identificador	#int2
Descripción	Se ejecuta la aplicación con todas las rutas de archivos configuradas de forma correcta y se navega entre las diferentes páginas de la aplicación.
Resultado Esperado	La aplicación *web* debe navegar entre las diferentes páginas correctamente.

Tabla 14. Prueba de Integración 2

Identificador	#int3
Descripción	Se ejecuta la aplicación con todas las rutas de archivos configuradas de forma correcta y se navega en la página a través de la tabla de contenidos.
Resultado Esperado	La aplicación *web* dentro de cada página debe navegar correctamente entre las diferentes secciones a través de la tabla de contenidos.

Tabla 15. Prueba de Integración 3

Identificador	#int4
Descripción	Se ejecuta la aplicación con todas las rutas de archivos configuradas de forma correcta, nos dirigimos a la página Pattern Type e interactuamos con las imágenes.
Resultado Esperado	Las imágenes se deben visualizar en la página y al hacer clic sobre ellas deben abrirse en un ligthbox sobre la misma página y tener los botones para compartir, descargar, ver a pantalla completa y cerra.

Tabla 16. Prueba de Integración 4

4.3 Pruebas de Sistema

Este tipo de pruebas sirven para evaluar el comportamiento de toda la aplicación, desde el frontend hasta el *backend*, asegurando que todos los requisitos se cumplen a nivel del sistema completo. Para este tipo de pruebas se ha utilizado la aplicación en modo producción habilitando el módulo waitress.

Identificador	#sist1
Descripción	Se prueba acceder a la aplicación *web* en múltiples dispositivos a la vez.
Resultado Esperado	La aplicación *web* debe ser capaz de soportar el acceso múltiple desde diferentes dispositivos con un tiempo de respuesta de carga bajo.

Tabla 17. Prueba de Sistema 1

Identificador	#sist2
Descripción	Se accede a la aplicación *web* y se cuenta el número de clics de cada unas de las funciones disponibles en la aplicación.
Resultado Esperado	El número de clic para todas las funciones no debe ser mayor a 4.

Tabla 18. Prueba de Sistema 2

Identificador	#sist3
Descripción	Se eliminan imágenes de la carpeta /statics/images
Resultado Esperado	En la página Pattern Type se debe seguir mostrando la información correctamente y en las secciones de las imágenes borradas se debe de mostrar el mensaje “No image avalaible for this pattern”

Tabla 19. Prueba de Sistema 3

Identificador	#sist4
Descripción	Se eliminan los csv de la carpeta /data
Resultado Esperado	En la página Pattern Type y Pattern Name se debe mostrar la información completa y en las secciones que corresponde a los datos del csv se debe de mostrar el mensaje “No CSV data found for this pattern”

Tabla 20. Prueba de Sistema 4

5 Resultados y Conclusiones

5.1 Navegación Final

Como se puede observar en la Ilustración 26, la navegación de la plataforma *web* se ha dividido en 3 vistas: Pattern Type, Pattern Name y Structures. Para poder ir de una vista a otra existe un menú de navegación superior que al hacer clic sobre el nombre de la vista nos dirige a esta.

Además, dentro de cada vista, se puede observar que en la parte izquierda de la ventana existe una tabla de contenidos, que permite navegar a través de las diferentes secciones existentes. Al igual que con el menú superior para poder ir de una sección a otra con ayuda de la tabla de contenidos únicamente hay que hacer clic sobre el nombre de la sección.

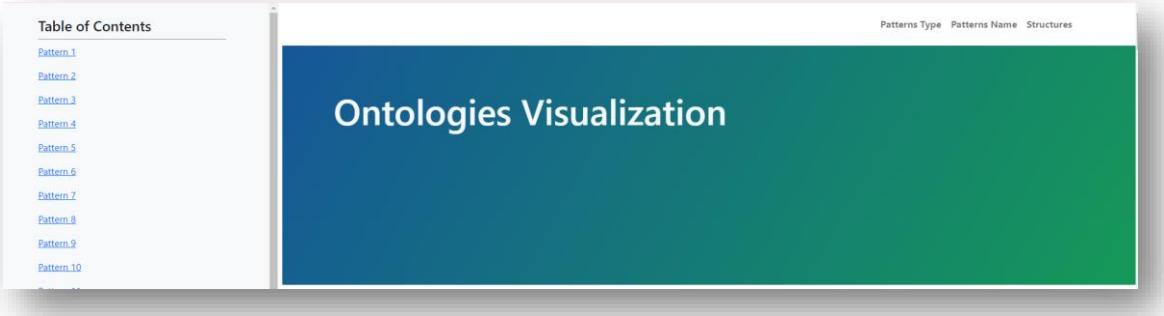


Ilustración 26. Navegación Final

5.2 Vista Final Pattern Type

En la Ilustración 27, se observa claramente cómo está organizada la vista de "Pattern Type" en la aplicación. Cada patrón se presenta inicialmente con un diagrama que muestra la estructura de este. A la derecha del diagrama se puede observar que se muestra la imagen del diagrama XML. Esta imagen está configurada para que el usuario pueda interactuar con ella. Al hacer clic sobre la imagen del diagrama XML, como se puede ver en la Ilustración 28, la imagen se abre en un lightbox sobre la página actual. Este *lightbox* no solo facilita una visualización más detallada del diagrama, sino que también integra funcionalidades interactivas accesibles mediante botones situados en la parte superior derecha del *lightbox*. Estos botones permiten a los usuarios compartir la imagen, abrir el diagrama en pantalla completa para una inspección más detallada o descargar la imagen para su uso fuera de la plataforma. Para salir del modo de visualización ampliada, los usuarios pueden hacer clic en el botón con una "X" situado también en la parte superior derecha, o simplemente haciendo clic fuera de la imagen en el fondo oscurecido.

Justo debajo del diagrama, se proporcionan datos estadísticos relevantes del patrón. Además, se incluye una lista que detalla las ontologías detectadas en el patrón, lo cual es esencial para los usuarios que buscan entender la aplicabilidad y el contexto de uso de cada patrón.

La lista de ontologías está organizada utilizando el componente de colapsables de Bootstrap. Este componente permite que las secciones de la lista se expandan o contraigan al hacer clic en un botón asociado. Cada ontología de la lista es un botón de colapso, que al ser presionado, despliega o contrae las estructuras concretas que aparecen en el patrón, además cada una de las estructuras es otro botón de colapso que al presionarlo, como se puede observar en la Ilustración 27, muestra el diagrama de la representación detallada de la estructura ontológicas enfocada en los nodos en blanco. Esta funcionalidad no solo mejora la organización visual de la información, sino que también proporciona una experiencia de usuario más interactiva y manejable, permitiendo a los usuarios explorar solo las secciones que les interesan sin abrumarse con demasiada información de una sola vez.

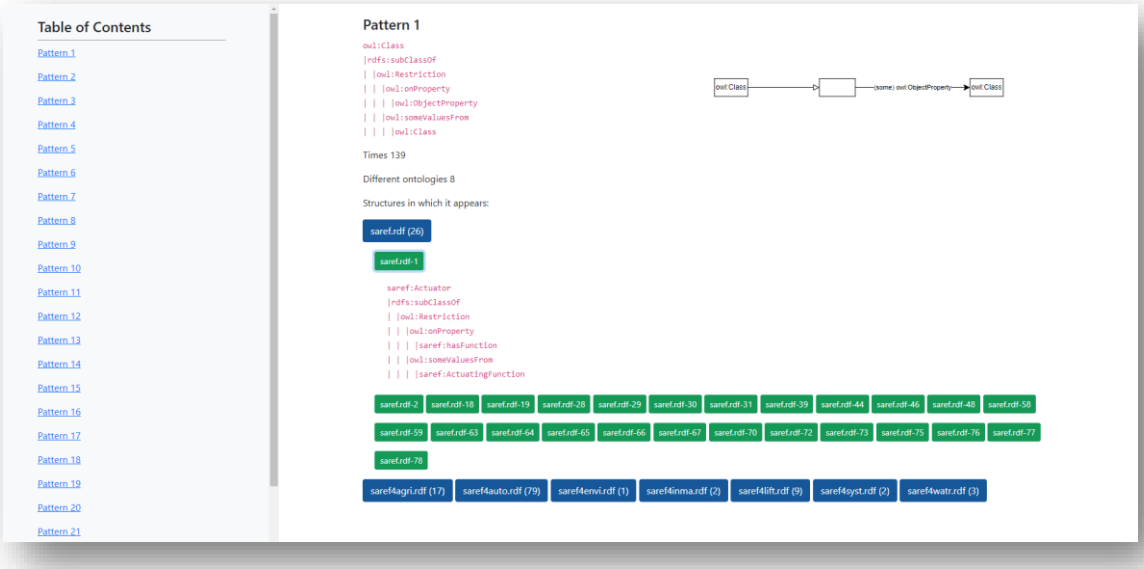


Ilustración 27. Vista Final de Pattern Type

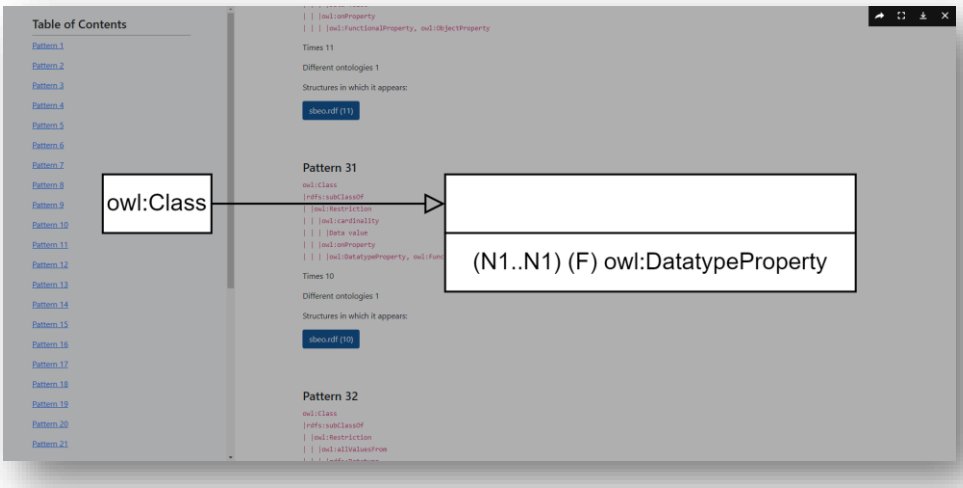


Ilustración 28. Vista Interacción Imagen Pattern Type

5.3 Vista Final Pattern Name

En la Ilustración 29, se puede observar que la organización de la vista “Pattern Name” es muy similar a la vista “Pattern Type”, con la excepción de la falta de la imagen XML y las listas de las estructuras de las ontologías. A diferencia de la vista “Pattern Type”, en la vista “Pattern Name” las ontologías detectadas no se muestran como botones de colapso, sino como badges. Esto se debe a que, en esta vista, se presentan los patrones de diseño ontológicos basados en los nombres de los términos en lugar de sus tipos, lo que evita así mostrar información repetida.

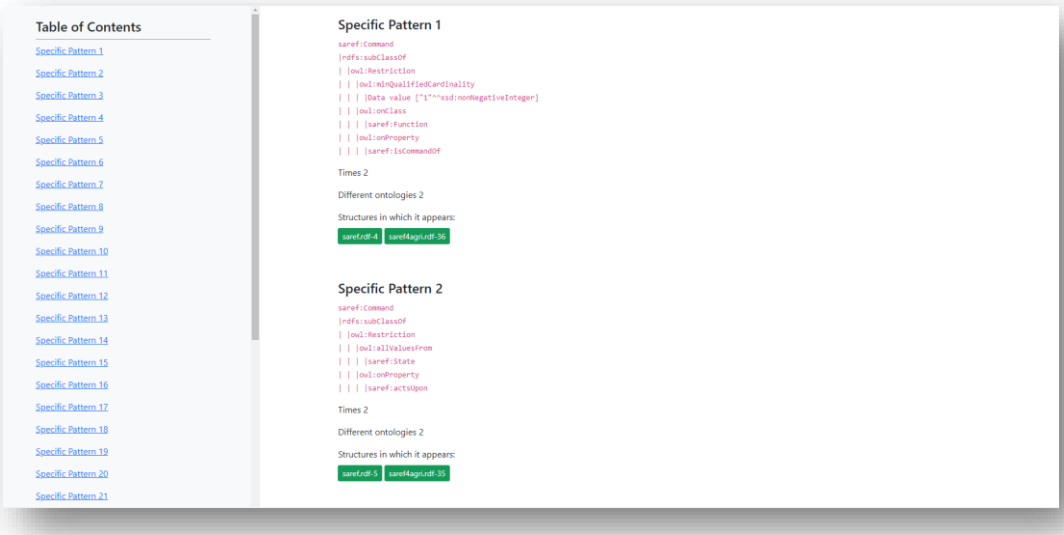


Ilustración 29. Vista Final de Pattern Name

5.4 Vista Final Structure

En la Ilustración 30, se observa claramente cómo está organizada la vista de "Structure" en la aplicación. Cada estructura presenta una tabla en la que se pueden observar los términos de las estructuras inferidas y los términos de las estructuras inferidas con nodos en blanco. Finalmente se muestra la ontología detectada en la estructura.

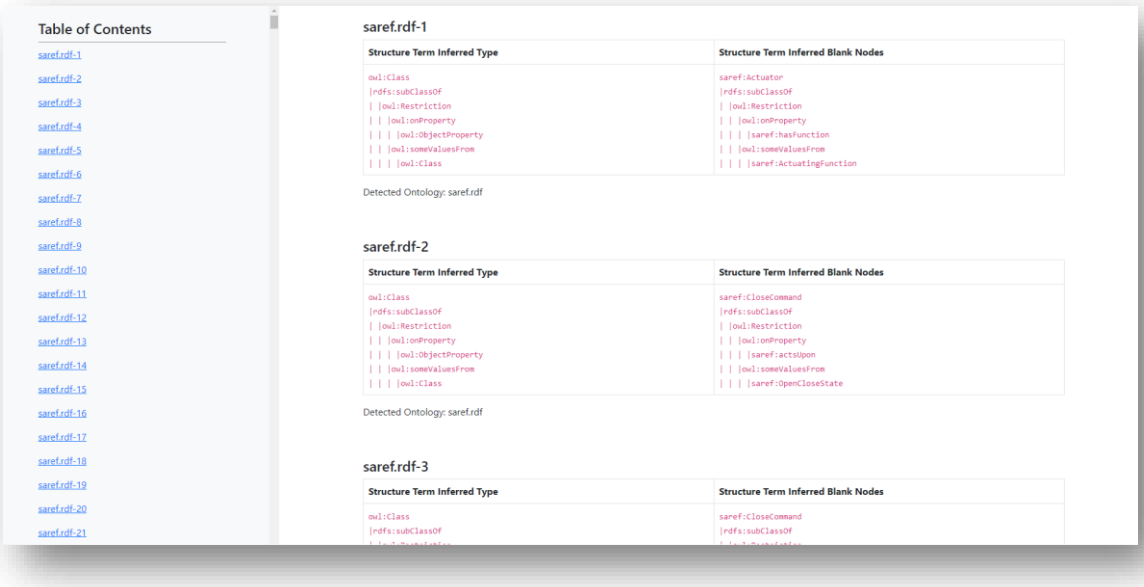


Ilustración 30. Vista final de Structure

5.5 Conclusiones

El análisis de las funcionalidades y la interfaz de usuario de la plataforma *web* revela un diseño cuidadosamente pensado que mejora significativamente la experiencia del usuario y facilita el acceso a la información crucial. Las ilustraciones discutidas evidencian una navegación intuitiva y una estructura de contenido bien organizada, aspectos que son fundamentales para el éxito de cualquier herramienta digital moderna. A continuación, se detallan cada una de las conclusiones que se han sacado durante el desarrollo del presente TFG.

- **Eficacia de la Navegación:** La inclusión de un menú de navegación superior y una tabla de contenidos en cada vista demuestra un enfoque centrado en el usuario que permite transiciones fluidas entre diferentes secciones y vistas. Esta organización no solo mejora la usabilidad, sino que también minimiza el tiempo de aprendizaje necesario para nuevos usuarios, facilitando un acceso rápido y directo a la información deseada.

- **Interactividad y Accesibilidad:** La capacidad de interactuar con los elementos visuales, como los diagramas XML, a través de un *lightbox*, proporciona una experiencia inmersiva que permite a los usuarios explorar detalles complejos sin salir de su contexto. Las funcionalidades integradas para compartir, descargar y visualizar en pantalla completa son críticas para usuarios que dependen de esta plataforma para su investigación o trabajo diario, ofreciendo flexibilidad en cómo consumen y utilizan la información.

- **Apoyo al Proceso de Aprendizaje y Análisis:** Al permitir que los usuarios visualicen estructuras de patrones y accedan a datos estadísticos relevantes directamente desde la interfaz, la plataforma no solo sirve como una herramienta de consulta, sino también como un recurso educativo que puede potenciar el entendimiento y la aplicación de estos patrones en diversos contextos profesionales o académicos.

- **Contribuciones a la Comunidad de Usuarios:** Al proporcionar un sistema que integra de manera efectiva la visualización de datos complejos con una interacción usuario amigable, el proyecto no solo cumple con las expectativas técnicas, sino que también enriquece la comunidad de usuarios al proporcionar una herramienta valiosa para el avance de sus propios objetivos y proyectos.

En conclusión, este proyecto demuestra cómo un diseño reflexivo y la implementación de funcionalidades interactivas pueden transformar la manera en que los usuarios acceden y interactúan con información compleja. La plataforma no solo mejora la accesibilidad y la comprensión de patrones complejos, sino que también establece un nuevo estándar en la presentación de este tipo de información, apuntando a una mejora continua en la interacción digital y el análisis de datos en el futuro.

5.6 Trabajo Futuro

Aunque el diseño actual de la plataforma y sus funcionalidades han demostrado ser eficaces, únicamente se han hecho pruebas con la tutora del presente TFG, María Poveda Villalón, y con el desarrollador de la herramienta de los ficheros de entrada, Sergio Mario Carulli Pérez. Por tanto, una línea futura de trabajo será la realización de pruebas con más usuarios nuevos para validar cuánto mejora la interacción con la plataforma. Este paso permitirá obtener retroalimentación directa de los nuevos usuarios, identificar posibles problemas de usabilidad y confirmar las mejoras en la experiencia del usuario que se anticipan con la integración de nuevas funcionalidades y optimizaciones. Probar la plataforma con nuevos usuarios ayudará a recopilar más información y asegurar que las mejoras técnicas también se traduzcan en una mejor interacción y satisfacción del usuario final.

Por otro lado, con la información compleja, las pruebas de rendimiento han revelado áreas potenciales para la mejora en la escalabilidad y eficiencia del sistema. Considerando esto, otra línea para el trabajo futuro en este proyecto será la integración de una base de datos. Esta evolución no solo se anticipa como una solución a los desafíos de rendimiento observados, como la acumulación de tareas en la cola del servidor bajo cargas elevadas, sino también como un paso esencial para soportar un mayor número de usuarios y gestionar de manera más eficiente grandes volúmenes de datos.

La transición a una arquitectura basada en base de datos permitirá gestionar eficientemente los datos ontológicos almacenados en archivos TXT y CSV, y las imágenes asociadas y permitirá pasar de una web estática a una *web* dinámica, lo que aumentará significativamente la interacción del usuario con la aplicación. A medida que el volumen de datos crece, acceder directamente a estos archivos se vuelve lento e ineficiente. La base de datos mejora significativamente el rendimiento al permitir indexar los datos para búsquedas rápidas y gestionar relaciones complejas entre los patrones y estructuras ontológicas. Además, facilita la personalización al almacenar preferencias y configuraciones del usuario, y mejora la gestión de sesiones. Esto no solo optimiza las operaciones de lectura y escritura, sino que también asegura que la plataforma pueda escalar y adaptarse a un mayor número de usuarios y volúmenes de datos, proporcionando una experiencia más robusta y eficiente para investigadores y académicos que utilizan la plataforma para analizar y visualizar modelos ontológicos.

Por último, un paso futuro muy prometedor sería la conexión directa entre la ejecución del programa de patrones, desarrollado por Sergio Mario Carulli Pérez, y la generación de la página *web*. Esta integración permitiría que un usuario envíe un conjunto de ontologías y obtenga automáticamente una página *web* generada a partir de esos datos. Este enfoque automatizaría completamente el proceso de creación de la *web*, eliminando la necesidad de intervenciones manuales para cargar y procesar los archivos de entrada.

Esta progresión hacia una solución más sofisticada y escalable refleja el compromiso continuo del proyecto con la mejora y la adaptación, asegurando que la plataforma no solo atienda las demandas actuales, sino que también

esté preparada para los desafíos futuros. Al hacerlo, el proyecto sigue estableciendo un nuevo estándar en la presentación y análisis de datos complejos, y promete contribuir aún más al avance tecnológico y al enriquecimiento de su comunidad de usuarios.

6 Análisis de Impacto

En este capítulo, se realiza un análisis exhaustivo del impacto potencial que los resultados obtenidos durante la realización del Trabajo de Fin de Grado (TFG) pueden tener en diversos contextos. Este análisis abarca desde el ámbito personal hasta el empresarial, social, económico, medioambiental y cultural, considerando tanto los beneficios esperados como los posibles efectos perjudiciales. Además, se evalúa cómo este proyecto se alinea con los Objetivos de Desarrollo Sostenible (ODS) de la Agenda 2030.

A nivel personal, el desarrollo de este proyecto ha significado un considerable avance en habilidades técnicas y analíticas, fundamentales para enfrentar desafíos profesionales futuros. El conocimiento adquirido y la experiencia ganada proporcionan una base sólida para futuras investigaciones o proyectos en ámbitos relacionados con la tecnología y el desarrollo de software.

Desde una perspectiva empresarial, el proyecto aporta un valor significativo al mejorar la eficiencia en el análisis y visualización de modelos ontológicos a través de la automatización y optimización del manejo de datos. Implementar esta solución permitirá a las empresas gestionar grandes volúmenes de datos ontológicos de manera más eficiente, facilitando la realización de consultas complejas y mejorando la precisión en el análisis de patrones y estructuras ontológicas. En el ámbito social, el proyecto potencia la accesibilidad y la comprensión de información compleja, contribuyendo así a la democratización del conocimiento. Esto es especialmente relevante en contextos educativos y de investigación, donde facilitar el acceso a herramientas avanzadas puede fomentar una sociedad más informada y preparada.

Económicamente, la implementación de este proyecto puede incentivar el crecimiento económico al permitir a las empresas redistribuir recursos hacia actividades de mayor valor añadido. Al automatizar el análisis de datos ontológicos y mejorar la eficiencia operativa, las empresas pueden reducir los costos asociados con el procesamiento manual de datos y aumentar su productividad. Esto, a su vez, puede liberar recursos para inversiones en innovación y desarrollo, fomentando un entorno económico más dinámico y competitivo.

En términos medioambientales, aunque el impacto directo del proyecto pueda ser neutral, las mejoras en la eficiencia operativa y la reducción en el uso de recursos físicos, como el papel y otros materiales, contribuyen indirectamente a la sostenibilidad ambiental.

Culturalmente, el proyecto promueve la adopción de tecnologías digitales en la gestión de información, lo que puede ayudar a cultivar una cultura que valore la eficiencia y la innovación tecnológica.

Este proyecto también se alinea con varios ODS, como el ODS 4, que promueve una educación de calidad a través de un mejor acceso a recursos educativos; el ODS 9, que apoya la industria, la innovación y la infraestructura; y el ODS 12, que aboga por una producción y un consumo responsables. Todas las decisiones tomadas a lo largo del proyecto han

considerado estos impactos, eligiendo tecnologías y métodos que maximizan los beneficios mientras minimizan cualquier efecto adverso.

En resumen, este análisis no solo destaca la importancia del proyecto en términos de su aplicación práctica y contribución académica, sino que también subraya su relevancia en el impulso hacia un futuro más sostenible y equitativo. Las decisiones conscientes tomadas durante el desarrollo del proyecto reflejan un compromiso con la responsabilidad social y medioambiental, asegurando que el proyecto no solo cumple con los requisitos técnicos, sino que también contribuye positivamente a la comunidad y al medio ambiente.

7 Bibliografía

[1] E. Vértice, *Técnicas avanzadas de diseño web*. Editorial Vértice, 2009.

[2] L. Asprino, V.A. Carreiro, V.Presutti, *Extraction of common conceptual components from multiple ontologies*, Univ. Bologna, 2021.

[3] Lozano-Tello, Adolfo. *Ontologías en la Web Semántica*, 2001. Accedido mayo de 2024. [En línea]. Disponible: http://eolo.cps.unizar.es/docencia/MasterUPV/Articulos/Ontologias%20en%20la%20Web%20Semantica.pdf

[4] "RDF - semantic web standards". W3C. Accedido mayo de 2024. [En línea]. Disponible: https://www.w3.org/RDF

[5] "RDF schema 1.1". W3C. Accedido mayo de 2024. [En línea]. Disponible: https://www.w3.org/TR/rdf-schema/

[6] "RDF terms in rdflib — rdflib 7.0.0 documentation". rdflib 7.0.0 — rdflib 7.0.0 documentation. Accedido mayo de 2024. [En línea]. Disponible: https://rdflib.readthedocs.io/en/stable/rdf_terms.html

[7] "Vista general del lenguaje de ontologías web (OWL)". W3C. Accedido mayo de 2024. [En línea]. Disponible: https://www.w3.org/2007/09/OWL-Overview-es.html

8 Anexo: Resultado de Pruebas

Identificador	Iteración	Resultado
#unit1	1	Test Superado
#unit2	1	Test Superado
#unit3	1	Test Superado
#unit4	1	Test Superado
#unit5	1	Test Superado
#unit6	1	Test Superado
#unit7	1	Test Superado
#unit8	1	Test Superado
#unit9	1	Test Superado
#unit10	1	Test Superado
#unit11	1	Test Superado
#unit12	1	Test Superado
#int1	3	Test Superado
#int2	1	Test Superado
#int3	1	Test Superado
#int4	1	Test Superado
#sist1	1	Test No Superado
#sist2	1	Test Superado
#sist3	1	Test Superado
#sist4	1	Test Superado

Este documento esta firmado por

	**Firmante**	CN=tfgm.fi.upm.es, OU=CCFI, O=ETS Ingenieros Informaticos - UPM, C=ES
	Fecha/Hora	Sun Jun 30 22:31:44 CEST 2024
	Emisor del Certificado	EMAILADDRESS=camanager@etsiinf.upm.es, CN=CA ETS Ingenieros Informaticos, O=ETS Ingenieros Informaticos - UPM, C=ES
	Numero de Serie	561
	Metodo	urn:adobe.com:Adobe.PPKLite:adbe.pkcs7.sha1 (Adobe Signature)