



UNIVERSIDAD DE SEVILLA

Máster Universitario en Lógica, Computación e Inteligencia Artificial

TRABAJO FIN DE MÁSTER

Diseño e implementación de una ontología para la gestión del conocimiento en el sector turismo y hostelería.

Elvin Jesús Reyes Vargas

Dirigido por:  
Antonia María Chávez

Sevilla, Septiembre de 2022



## **Abstract**

One of the biggest challenges that humanity has constantly faced is the access to information, and to make it remain over time. The semantic web has come to provide meaning to the content on the web, thus seeking to constitute the web in a large knowledge base, where everything is related, and where the use of the web becomes more efficient.

The main objective of this work is the design and implement an ontology for knowledge management in the tourism and hospitality sector, based, on one hand on the METHONDOLOGY methodology, for the design and construction of the ontology, and on the other hand, based in the use of agile methodologies and clean code for its programming.

Through this project it has been successfully achieved the construction of a web page based on an ontology, where all the knowledge and reasoning that is generated in it, explicitly or not, is extracted. The final result demonstrates the viability of the semantic web as the future of the way in which knowledge is created and interpreted in web pages.



# Índice General

## Índice de Figuras

- 1. Introducción y objetivos**
  - 1.1. Introducción
  - 1.2. Objetivos
- 2. Conceptos de Ontologías**
  - 2.1. La web semántica
    - 2.1.1. RDF
      - 2.1.1.1. RDFS
      - 2.1.1.2. RDFa
    - 2.1.2. OWL
    - 2.1.3. Linked Open Data
    - 2.1.4. SPARQL
  - 2.2. Ontologías y léxico
    - 2.2.1. Elementos y características de una ontología
    - 2.2.2. Metodologías para el desarrollo de ontologías
    - 2.2.3. Creación de una ontología
    - 2.2.4. Definición de una ontología en Protégé
- 3. Visión global de la ontología implementada**
  - 3.1. Introducción y características del dominio
  - 3.2. Alcance
  - 3.3. Clases y jerarquías
  - 3.4. Propiedades
    - 3.4.1. Propiedades de objetos
    - 3.4.2. Propiedades de datos
  - 3.5. Uso
  - 3.6. Toma de decisiones
  - 3.7. Representación gráfica
- 4. Representación Web de la ontología**
  - 4.1. Arquitectura general del sistema
    - 4.1.1. Aspectos metodológicos de implementación
    - 4.1.2. Arquitectura lógica del proyecto (Backend)
    - 4.1.3. Arquitectura visual del proyecto (Frontend)
    - 4.1.4. Capas del proyecto
      - 4.1.4.1. Aplicación
      - 4.1.4.2. Infraestructura
      - 4.1.4.3. Modelos
      - 4.1.4.4. API
      - 4.1.4.5. Web
  - 4.2. Organización funcional del proyecto

## **Indice General**

### **5. Análisis de implementación**

#### **5.1. Código y análisis final de la implementación realizada**

##### **5.1.1. Modelo**

##### **5.1.2. Repositorio**

##### **5.1.3. Servicio**

##### **5.1.4. API**

##### **5.1.5. Cliente**

### **6. Consideraciones finales**

#### **6.1. Conclusiones**

#### **6.2. Futuras líneas de trabajo**

### **7. Referencias**

### **8. Anexos**

#### **8.1. Diagrama Gantt de planificación del proyecto**

#### **8.2. Documentación del despliegue del servicio REST**



## Índice de figuras

- 2.1 Estructura de la web semántica.
- 2.2 Visión general de Protégé.
- 2.3 Visión general de Protégé con el dominio trabajado.
- 2.4 Taxonomía de clases.
- 2.5 Visión general de Protégé con los individuos por clase.
- 3.1 Elementos principales de la ontología diseñada.
- 3.2 Clases principales de la ontología.
- 3.3 Clase de conocimiento general.
- 3.4 Clase de Rooms o habitaciones.
- 3.5 Clase de activities.
- 3.6 Clase de Facilities.
- 3.7 Clase de Destinations.
- 3.8 Clase de Accommodations.
- 4.1 Visual Studio Community 2022.
- 4.2 Sistema de control de versiones GIT.
- 4.3 Logo de la herramienta de trabajo Trello.
- 4.4 Listado de tareas creadas en Trello.
- 4.5 Lenguaje de programación C# y Plataforma .Net 6.
- 4.6 Librería dotNetRDF.
- 4.7 Servicios REST (Transferencia de Estado Representacional).
- 4.8 Estándar para transferencia de datos JSON.
- 4.9 Especificación abierta para documentación con Swagger.
- 4.10 Diagrama MVC de una aplicación.
- 4.11 Framework para el manejo de estilos Bootstrap.
- 4.12 Lenguaje de script en el navegador JavaScript.
- 4.13 Librerías jQuery y AJAX.
- 4.14 Framework para manejo de interfaces Vue.js.
- 4.15 Manejo de solicitudes con Refit
- 4.16 Arquitectura propuesta por el diseño basado en el dominio.
- 4.17 Estructura general del proyecto.
- 4.18 Capa de aplicación.
- 4.19 Capa de infraestructura.
- 4.20 Capa de modelos.
- 4.21 Capa de API (Application Programming Interface).
- 4.22 Capa web.
- 4.23 Página de inicio del sitio web.
- 4.24 Página con el listado de destinos.
- 4.25 Página con el detalle de un destino.
- 4.26 Vista detallada de un individuo.
- 4.27 Diseño de la página web de ontología.





# **1. Introducción y objetivos**

## **1.1. Introducción**

Con el nacimiento de la web semántica, o Web 3.0, como una visión futura en el año 2000 de hacia donde deben enfocarse los esfuerzos en la interacción entre datos web, máquinas y humanos, se empieza a crear toda una estructura de conceptos y prácticas sobre cómo se debería representar el conocimiento en la web, buscando evolucionar de la Web 2.0, donde hay poca relación entre los datos y el contexto en que son utilizados, y posteriormente filtrados por los diferentes buscadores.

La ontología, como elemento importante dentro de la web semántica, intenta hacer una especificación explícita y formal sobre una conceptualización compartida, partiendo de la definición anterior, las ontologías definen conceptos y relaciones de algún dominio, dicha conceptualización debe ser representada de una manera formal, legible y utilizable por los ordenadores y los humanos.

El objetivo principal de la web semántica es que la información en la web pase a ser una base de conocimientos, donde se facilite la búsqueda de datos específicos. Partiendo de esto es que se aborda la implementación de esta ontología, como un dominio específico, donde se crea una base de conocimientos común y con un lenguaje formal, donde se aplican todos los conceptos y enunciados propuestos por la W3C [5].

Este trabajo se enfoca en el desarrollo e implementación de una ontología de gestión del conocimiento en el sector turismo, tomando como eje focal todas aquellas áreas que lo conforman, y siempre tratando de realizar una ontología apegada a la realidad del dominio ya mencionado; con este informe no solo se pretende desarrollar una ontología, sino hacer su representación en una página web, donde se presente el conocimiento generado e inferido por medio de métodos de búsqueda dinámicos.

A grandes rasgos se hace una revisión de la bibliografía actual que engloba la web semántica, analizando los conceptos clave, y su posterior uso en la ontología realizada. En este contexto, el objetivo de este trabajo es la exploración de las metodologías existentes para la determinación de una metodología que permita el desarrollo de un modelo ontológico que responda a las necesidades del sector turismo, con el fin de lograr una gestión más eficiente de los recursos que maneja. Es decir, el objetivo principal es estudiar de forma preliminar la creación de una ontología para el fin mencionado.

Esta ontología nos lleva a abordar diferentes aspectos de la web 3.0, así como elementos más específicos en su ámbito de desarrollo, en donde se centra este

trabajo. Este informe se enmarca en abordar la importancia del sector turismo como gran fuente de conocimiento, y como forma de servir para que cuando estemos asentados sobre la web semántica, este dominio pueda alinearse con los estándares y requerimientos solicitados.

Realizar esta representación del dominio ya mencionado supone de la interacción entre muchas áreas del sector, que inicialmente podrían no ser abordados a plenitud, y que deja abierta la posibilidad de mejoras y cosas por agregar.

Definitivamente el conocimiento que se tiene en la web del sector turismo aumentará exponencialmente, es por medio de la web semántica que definitivamente se podrá alcanzar un nivel de acoplamiento óptimo tanto en lo que sabe el usuario, como en lo que ofrece el proveedor, y en lo que la computadora es capaz de procesar para lograr una mejor interoperabilidad entre todos los actores de este sector.

Una vez descritos los detalles de implementación de esta ontología, que fue desarrollada con Protégé, por medio del lenguaje OWL [1], se abordarán los elementos de la representación web de la misma, que ha sido implementada en .Net 6, y utilizando las mejores prácticas de programación, seguido de un breve análisis sobre los resultados obtenidos como producto de la realización de esta ontología.

Desde el punto de vista técnico, la Web es un conglomerado de tecnologías, estándares y agentes, es por ello que representar el dominio trabajado en la web supone del uso de diferentes tecnologías, metodologías, estrategias de implementación, así como buenas prácticas, que serán tratados más adelante.

Finalmente se presentan las consideraciones finales, donde se exponen los puntos más relevantes como resultado final, así como las futuras líneas para extender y mejorar lo que se ha realizado en este trabajo.

## 1.2. Objetivos

Este trabajo tiene como objetivo principal el diseño e implementación de una ontología para la gestión del conocimiento en el sector turismo y hostelería.

Se pretende demostrar que el dominio tratado es una base de conocimiento amplia y de mucho interés, porque constituye una fuente importante de información y recursos para todos los actores involucrados en las gestiones que toman lugar. Por otro lado, el dominio tratado es de alto interés por lo que significa en la actualidad, y por ende el impacto socio-económico que posee.

Otros objetivos que se persiguen por medio de este trabajo son:

- Proponer una ontología para el modelado y representación de los datos relativos a la actividad apegado a la realidad del dominio.
- Comprender e identificar el contexto tecnológico de la Web Semántica.
- Identificar los principales elementos de la propuesta por medio de clases y propiedades, que permitan desarrollar la ontología.
- Documentar y detallar la ontología propuesta de forma adecuada, para un mayor entendimiento.
- Representar la ontología diseñada utilizando las tecnologías y metodologías más adecuadas.

Para alcanzar los objetivos planteados se realizará un análisis exhaustivo de las metodologías y estrategias que mejor se adecúan al dominio y tamaño del mismo.

Con respecto a la selección del lenguaje de programación e infraestructura tecnológica, la decisión de elegir una tecnología u otra será basado, primero en los conocimientos previos de programación, y segundo en lo adecuado que pueda ser para el desarrollo del proyecto.

Alcanzar los objetivos planteados sin lugar a dudas se verá reflejado en unos resultados adecuados y de altos estándares con respecto a lo significativo que es este trabajo, así, se podrá medir si se han alcanzado las metas propuestas por medio de diversas pruebas tanto sobre la arquitectura y funcionalidades, como en el sistema resultante.

## **2. Conceptos de Ontologías**

La web semántica ha llegado para dotar de significado el contenido en la web, de esa forma se busca constituir la web en una gran base de conocimientos, donde todo se relaciona, y donde el uso de la web se convierte en más eficiente.

Es a través de la web semántica que múltiples conceptos han tomado acción, para establecer de forma clara los vínculos en todo lo que representa.

Este apartado permitirá conocer en detalle el estado del arte de los diferentes conceptos que toman escena al momento de realizar e implementar una ontología.

### **2.1. La web semántica**

Uno de los grandes retos que ha tenido de forma constante la humanidad es el acceso a la información, y en gran medida hacer que la misma permanezca a través del tiempo.

Antes de la aparición de la web, como gran base de datos y conocimiento para, el manejo de información quedaba relegado meramente a las bibliotecas, y documentos que con mucho cuidado se almacenaban con el fin de protegerlos y asegurar su permanencia en el tiempo.

Todo cambia cuando aparece la web, que llega a revolucionar la forma en que la humanidad se comunica y se informa. Si bien es cierto que en el principio la Web no era lo eficiente que es hoy día, ha servido para hacer el conocimiento global.

Con la intención de que el conocimiento sea más legible, esté interconectado, y tengo mayor significado tanto para las personas como para las máquinas, es que surge la web 3.0, o web semántica.

Tal y como la define la World Wide Web (W3C) [5], “La Web Semántica es una Web extendida, dotada de mayor significado en la que cualquier usuario en Internet podrá encontrar respuestas a sus preguntas de forma más rápida y sencilla gracias a una información mejor definida. Al dotar a la Web de más significado y, por lo tanto, de más semántica, se pueden obtener soluciones a problemas habituales en la búsqueda de información gracias a la utilización de una infraestructura común, mediante la cual, es posible compartir, procesar y transferir información de forma sencilla. Esta Web extendida y basada en el significado, se apoya en lenguajes universales que resuelven los problemas ocasionados por una Web carente de semántica en la que, en ocasiones, el acceso a la información se convierte en una tarea difícil y frustrante.” [1].

Partiendo de la definición que propone la W3C se pueden hacer varias deducciones que dan un sentido más amplio a lo que representa la web semántica; en primer lugar, esta viene a ser una extensión de la Web actual dotada de significado, de manera que pueda ser interpretada tanto por agentes humanos como por agentes computerizados.

En segundo lugar, para poder explotar la Web semántica, se necesitan lenguajes semánticos más potentes, esto es, lenguajes de marcado capaces de representar el conocimiento basándose en el uso de metadatos y ontologías. Estos lenguajes deben ser estandarizados y formalizados para que su uso sea universal, reutilizable y compartido en toda la Web. Se necesita un lenguaje común basado en web, con suficiente capacidad expresiva y de razonamiento para representar la semántica de las ontologías. De esta forma, la utilización de lenguajes tales como OWL son un paso más en la consecución de la Web Semántica.

Por otro lado, es necesario crear bibliotecas de vocabularios descriptivos/semánticos, definidos en un formato estándar y ubicados en la Web para determinar el significado contextual de una palabra por medio de la consulta a la ontología apropiada. De esta forma, agentes inteligentes y programas autónomos podrían rastrear la Web de forma automática y localizar, exclusivamente, las páginas que se refieran a la palabra buscada con el significado y concepto precisos con el que interpretemos ese término.

En tercer lugar, se expone que el objetivo de la Web Semántica es que la Web pase de ser una colección de documentos a convertirse en una base de conocimiento, donde se abren nuevas posibilidades de comunicación entre hombres y sistemas informáticos a través de servicios que unen múltiples fuentes de datos, y que son tratados para su procesamiento o ejecución de búsquedas de información. Con esto se busca mejorar el entendimiento de la información de los ordenadores y dispositivos, facilitando así la tarea de búsqueda de la información.

La web semántica es una estructura compleja compuesta por una arquitectura de capas. Cada parte de esta representa un aspecto concreto del problema de agregar y utilizar información semántica. El objetivo principal de esta estructura es brindar una fuente sólida y fiable que sirva de guía para una implementación fácil de lo que se persigue mediante la web 3.0. Partiendo de eso en la presentación “Semantic Web” de Tim Berners-Lee en el año 2000, propone lo que se convertiría en la base fundamental de la web semántica; en dicha presentación presenta un enfoque global sobre la función de cada uno de estos niveles dentro de este nuevo esquema funcional de la web.

A continuación tenemos la representación de la jerarquía de capas propuesta por Tim Berners-Lee [4].

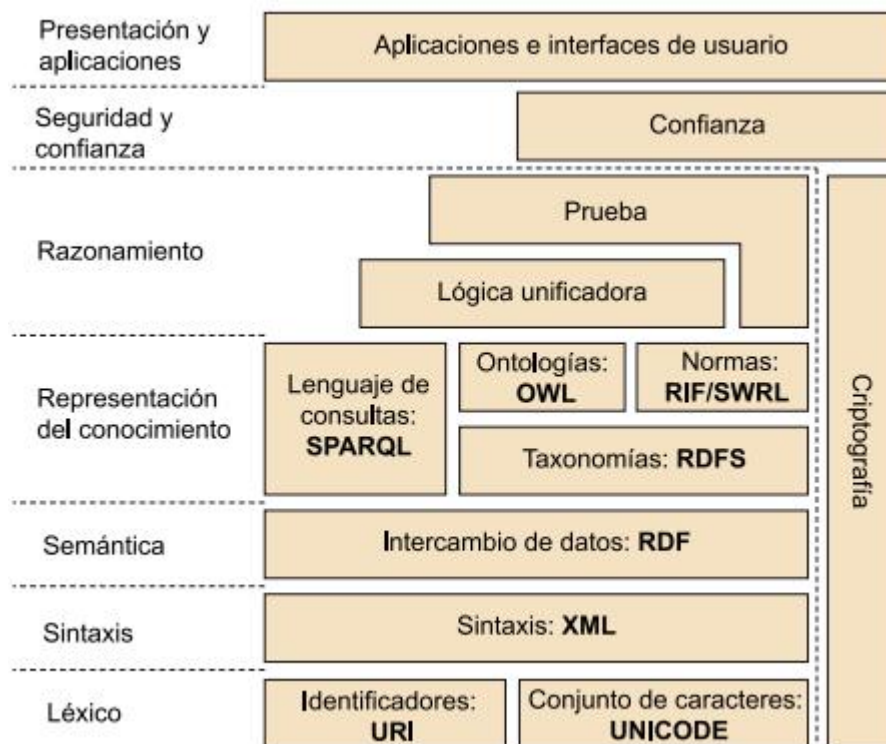


Figura 2.1: Estructura de la web semántica

Como se puede observar, la arquitectura está organizada en una jerarquía de capas:

- a) **Capa de léxico:** define cuáles son los símbolos que podemos utilizar para construir la web. Este nivel considera qué caracteres se pueden utilizar y cómo se denominan los recursos:
  - **Unicode** es el estándar que posibilita que todos los idiomas del mundo puedan ser utilizados para leer o escribir en la web de manera estándar.
  - **Uniform Resource Identifier (URI)** es un formato estandarizado que identifica de manera única recursos, como pueden ser documentos, imágenes, etc. Su uso es importante para sistemas distribuidos, porque proporciona una identificación inteligible de todos los recursos.
- b) **Capa de sintaxis:** especifica el modo como los símbolos definidos previamente se agrupan para construir una página válida. Los estándares web codifican la información mediante lenguajes basados en XML (eXtensible Mark-up Language), un lenguaje de propósito general para el marcaje de información estructurada en documentos.

- c) **Capa semántica:** asigna significado a los elementos de una página web. Posee el núcleo del formato de representación de datos para la web semántica, el Resource Description Framework (RDF). RDF es una notación estándar para la representación e intercambio de información sobre recursos, será tratado a detalle más adelante.
- d) **Capa de representación del conocimiento:** engloba un conjunto de lenguajes por medio de los que se pueden realizar un conjunto de actividades de representación del conocimiento. Por ejemplo, RDF permite añadir anotaciones semánticas. Para poder representar una ontología disponemos de otro estándar que nos ofrece extensiones respecto a RDFS, el Web Ontology Language (OWL). Otros estándares no se centran en la descripción de vocabularios sino de otro tipo de conocimiento: conocimiento inferencial en forma de reglas de producción. Aquí destacan dos estándares/especificaciones, el Rule Interchange Format (RIF) y el Semantic Web Rule Language (SWRL). También se dispone de un lenguaje para hacer consultas: Simple Protocol and RDF Query Language (SPARQL), un estándar para la consulta de información basada en RDF, como pueden ser ontologías RDFS y OWL.
- e) **Capa de razonamiento:** permite relacionar el conocimiento y hacer inferencias. Los estándares descritos previamente definen una semántica formal que permite razonar sobre ellos. Aquí destaca la lógica descriptiva, una familia de lenguajes formales que inspiran la notación OWL y que permiten hacer inferencias, como por ejemplo detectar información inconsistente. Se considera que este es uno de los talones de Aquiles de la web semántica, porque es complicado conseguir mecanismos de razonamiento que sean escalables para grandes volúmenes de información y que a la vez sean capaces de extraer conclusiones no triviales.
- f) **Capa de seguridad y confianza:** los agentes que utilicen información semántica deberán poder asegurar la integridad de la información, establecer el grado de confianza de una fuente de conocimiento o mantener la confidencialidad de los datos. Con este objetivo, será necesario utilizar técnicas criptográficas (como por ejemplo el cifrado o la firma digital) y de gestión de la reputación para asegurar la seguridad y confianza de las aplicaciones en la web semántica.
- g) **Capa de presentación y aplicaciones:** aporta un valor añadido y las interfaces que permiten a los usuarios acceder al conocimiento de la web semántica de forma amigable.



### 2.1.1. RDF

Resource Description Framework (RDF) es un estándar del W3C para el intercambio de datos y la descripción de su semántica [4].

Está diseñado para representar el conocimiento en un mundo distribuido, permitiendo integrar datos de diversas fuentes sin requerir programación personalizada, garantizando así su reutilización por otras partes.

El punto fuerte de RDF radica en su integración con diversas fuentes de datos, permitiendo así que se puedan describir las propiedades y meta-información de los documentos web, y siempre mantendrá el significado de los datos.

RDF concibe un modelo basado en enunciado o tripletas, formado por tres elementos:

- Un **sujeto**, que es la entidad origen de la relación;
- un **predicado** o propiedad, que identifica la relación;
- y un **objeto**, entidad de destino de la relación.

La representación de esta relación queda como sigue:

Sujeto-Predicado-Objeto

No todo el conocimiento web se podrá describir con una única triplete, por lo que puede ser necesario descomponerla en un conjunto de tripletas.

RDF identifica cada recurso de forma única, así, se basa en la especificación URI (Uniform Resource Identifier), de forma particular utiliza lo que se denomina URIref (URI references) para identificar cada elemento de la triplete.

#### 2.1.1.1. RDFS

RDF Schema o Esquema RDF es una extensión semántica de RDF [6].

Surge para cubrir la necesidad de especificación de tipos o clases específicas de recursos. RDFS no proporciona un vocabulario sobre aplicaciones orientadas a clases, sino que provee de mecanismos para especificar que tales clases y propiedades son parte de un vocabulario y de cómo se espera su relación [7]. Este sistema de clases y propiedades es similar al utilizado en la programación orientada a objetos. Para evitar definiciones conflictivas con el mismo término, se utilizarán espacios de nombres. De este modo se consigue asociar cada uso de una palabra al esquema donde tiene una definición determinada.

### 2.1.1.2. RDFa

Resource Description Framework in attributes (RDFa) es un mecanismo que posibilita marcar datos RDF (enunciados con sujeto-predicado-objeto) dentro de documentos HTML de manera directa. En realidad no solo se puede aplicar a documentos HTML (HTML4 y HTML5), sino también a cualquier otro documento basado en el formato XML, como son XHTML, XML y SVG. A la vez, se posibilita la extracción de esta información estructurada (tripletas) [3].

Su esencia recae en la posibilidad de agregar atributos que permiten colocar metadatos a los lenguajes de marcado, tales como HTML, así, su idea principal es marcar la entidad con un tipo de elemento.

Los atributos básicos que se utilizan para el marcado de datos con RDFa Core son los que siguen:

- **vocab:** define vocabularios sobre los elementos más comunes en la web. Antes de marcar información dentro del documento web, hay que indicar el vocabulario a utilizar.
- **typeof:** indica el tipo de sujeto, o sea, de qué se habla.
- **property:** permite especificar las propiedades del sujeto.

### 2.1.2. OWL

Ontology Web Language (OWL) o Lenguaje de Ontología Web, es un lenguaje que extiende RDF y RDFS, ofreciendo un conjunto mucho más amplio de tipos de restricciones al conjunto de tripletas definidas. Además de ello, el lenguaje ofrece una serie de diversos constructores que permiten, entre otras posibilidades, la construcción de clases complejas a partir de otras definiciones de clases, y el encadenamiento de propiedades [2].

OWL se encargará de establecer relaciones entre los conceptos y las reglas lógicas que son necesarias para entender el conocimiento en la web semántica.

Una de las principales bases del OWL es Description Logics (DLs), una familia de lenguajes de representación de conocimiento ampliamente utilizados en el modelado de ontologías. Una ontología es la especificación de un concepto dentro de un determinado dominio de interés.

OWL está diseñado para usarse cuando la información contenida en los documentos necesita ser procesada por programas o aplicaciones, en oposición a situaciones donde el contenido solamente necesita ser presentado a los seres humanos.

OWL puede usarse para representar explícitamente el significado de términos en vocabularios y las relaciones entre aquellos términos, además posee más funcionalidades para expresar el significado y semántica que XML, RDF, y RDFS, incluso va más allá que estos lenguajes pues ofrece la posibilidad de representar contenido de la Web interpretable por máquina.

El motivo del desarrollo de este lenguaje ha sido la puesta en marcha de la Web Semántica.

### **2.1.3. Linked Open Data**

Linked Open Data indica que son datos abiertos en RDF (que es el estándar W3C para Web Semántica). Esto significa que el usuario puede enlazar datos provenientes de diversas fuentes, instituciones u organizaciones, explorar y combinar estos datos de manera libre y sin restricciones de copyright para nuevos desarrollos web [4].

El objetivo principal de Linked Open Data es la reutilización de datos gestionados por diferentes entidades para su lectura, interpretación y procesado por parte de aplicaciones informáticas

Linked Open Data trata de facilitar la investigación, facilitar las contribuciones y colaboraciones, aumentar la eficiencia y calidad de los sistema y conseguir un alta transparencia de las instituciones.

Con Linked Open Data se hace necesario usar formatos abiertos y de libre uso, que están caracterizados por ofrecer toda la información sobre sus características e implementación, como por ejemplos CSV, JSON, XML y RDF.

### **2.1.4. SPARQL**

Simple Protocol and RDF Query Language (SPARQL), un estándar para la consulta de información basada en RDF, como pueden ser ontologías RDFS y OWL [4]. Es un lenguaje inspirado en SQL que utiliza tripletas RDF y recursos tanto para hacer coincidir la consulta como para devolver los resultados de esta. SPARQL también es un protocolo para acceder a datos RDF.

Las variables SPARQL comienzan con un carácter "?" y pueden ser definidas en cualquiera de las tres posiciones de una triplete (sujeto, predicado, objeto) en el conjunto de datos RDF. SPARQL admite una serie de filtros y operadores que permiten hacer consultas complejas al conjunto de tripletas almacenadas.

## 2.2. Ontologías y léxico

Una ontología o modelo del dominio posibilitan la compartición de conocimiento en la web [4]. Su objetivo es facilitar la comunicación y la transmisión de la información entre diferentes sistemas y entidades, incluyendo a las personas.

Las ontologías definen conceptos y relaciones de algún dominio, de forma compartida y consensuada; y que esta conceptualización debe ser representada de una manera formal, legible y utilizable por los ordenadores.

Las ontologías se encargan de definir los conceptos necesarios para describir y representar un determinado dominio y las relaciones existentes entre estos conceptos.

A partir de su definición, podemos identificar las tres **principales propiedades** de las ontologías. Una ontología [3]:

- a) utiliza una representación explícita; por lo tanto, suele estar escrita en un lenguaje formal y en un soporte digital que puede ser leído e interpretado por programas informáticos.
- b) es una conceptualización compartida; por lo tanto, representa la información que un conjunto de personas tienen con respecto a un dominio de discurso. Una ontología no puede representar el punto de vista de un solo individuo, como podría suceder, por ejemplo, con un esquema de base de datos.
- c) representa un dominio en particular; por lo tanto, representa el dominio de discurso relevante para un problema concreto.

La ontología ha dado lugar al resurgimiento de la disciplina de la representación del conocimiento, área de la inteligencia artificial que facilita la compartición y la reutilización del conocimiento.

Las ontologías establecen el lenguaje común en que la web semántica se basará para mantener alineados los estándares en que se fundamentará la web una vez se haya evolucionado completamente hacia ella.

Este apartado se centra en abordar los principales elementos que componen una ontología, así como el proceso a seguir para crearla, y la metodología más apropiada para alcanzar ese fin. Se entrará en el detalle de la implementación y en la elección de los formatos.

### 2.2.1. Elementos y características de una ontología

La ontología define modelos base que tendrán la definición semántica representando a una clase de objetos en la ontología.

En una ontología, los conceptos son las unidades fundamentales para la especificación. Proveen una base para la descripción de información. Cada concept consta de 3 componentes básicos: términos, atributos, relaciones.

Las ontologías están formadas por los siguientes componentes que servirán para representar el conocimiento de algún dominio específico [4]:

- **Conceptos:** son las ideas básicas que intentan formalizar, estos conceptos pueden ser clases de objetos, métodos, planes, estrategias, proceso de razonamiento, etc.
- **Roles o relaciones:** representan la interacción y el enlace entre los conceptos del dominio. Suelen formar la taxonomía del dominio. Por ejemplo: subclase-de, etc.
- **Funciones:** son un tipo concreto de relación, donde se identifica un elemento mediante el cálculo de una función que considera varios elementos de la ontología.
- **Individuos o instancias:** representan objetos determinados de un concepto.
- **Axiomas:** son teoremas que se declaran sobre relaciones que deben cumplir los elementos de la ontología.

Las ontologías se pueden ver como un conjunto de conceptos-definiciones, estos conceptos pueden ser ordenados en jerarquías de taxonomías y tener propiedades asociadas.

La ontología codifica la estructura relacional de conceptos, utilizada para describir aspectos del mundo. En el proceso de creación de una ontología, se hace uso de conocimiento previo almacenado en una ontología para la indexación y solución a problemas de búsqueda.

Por otro lado, una ontología debe presentar:

- **Claridad:** comunicar el significado intencionado de los términos definidos.
- **Coherencia:** para sancionar inferencias que son consistentes con las definiciones.
- **Extensibilidad:** para anticipar el uso de vocabulario compartido.

### **Algunas de las características más importantes de las ontologías son:**

- Se puede combinar dos o más ontologías, dependiendo de la dependencia que tenga una de otra.
- Poseen diferentes niveles de abstracción o generalización, que permite obtener una topología de la ontología. La idea es caracterizar una red de ontologías con el uso de multiplicidad y abstracción. Puesto que no podemos aspirar a tener una descripción completa del mundo, se puede pensar en una estrategia de construcción gradual de abajo hacia arriba.
- Un concepto puede ser representado de muchas formas, por lo que pueden coexistir múltiples representaciones de un mismo concepto.
- Se puede establecer relaciones entre los elementos de una o más ontologías, para generar conexiones, especializaciones y generalizaciones.

### **2.2.2. Metodologías para el desarrollo de ontologías**

La elección de una metodología para crear la ontología dependerá en gran medida del alcance del dominio, condiciones de reutilización, resultado esperado, y la disponibilidad de información sobre el dominio.

Algunas metodologías que describen procedimientos, tareas y ciclos de vida para la construcción y validación de las ontologías son las siguientes [4]:

**Methontology:** metodología propuesta por el Laboratorio de Inteligencia Artificial de la Universidad Politécnica de Madrid (UPM), permite empezar la creación de ontologías desde cero o a partir de la reutilización de otras ya existentes. Es uno de los métodos de ingeniería de ontologías más completo. Esta metodología identifica un conjunto de actividades en el proceso de desarrollo de la ontología, como por ejemplo la evaluación, la configuración, la gestión, la conceptualización y la integración.

Propone un ciclo de vida de construcción basado en prototipos evolutivos, con objeto de poder agregar, cambiar y eliminar elementos en cada nueva versión (llamado prototipo). La metodología especifica los pasos para llevar a cabo cada una de las actividades, las técnicas utilizadas, los resultados que se tendrían que obtener y el proceso para hacer la evaluación.

**On-To-Knowledge:** destinado al desarrollo de metodologías y herramientas para facilitar la gestión de conocimiento por medio de ontologías, y destinado a profesionales no especializados en ciencias de la computación.

Dentro del proyecto se creó uno de los primeros lenguajes de representación de ontologías para la infraestructura de la web semántica, denominado **Ontology Interchange Language (OIL)**, este se basa en conceptos desarrollados en la lógica descriptiva y la representación de marcos, sin abandonar la compatibilidad con RDF.

**Text2Onto:** es un entorno de trabajo para la creación de ontologías a partir técnicas de data mining.

**SENSUS-Based:** Este método es un enfoque top-down para derivar ontologías de dominio específico a partir de una ontología de alto nivel, como por ejemplo Cyc, ConceptNet, WordNet o SENSUS.

**Grüninger y Fox:** se ha desarrollado a partir de la experiencia obtenida en el proyecto Toronto Virtual Enterprise (TOVE), cuyo objetivo fue desarrollar un conjunto de ontologías para modelizar procesos y actividades de negocio. La metodología lleva el nombre de sus investigadores Michael Grüninger y Mark S. Fox.

### 2.2.3. Creación de una ontología

En el proceso de creación de una ontología, también llamado ingeniería de la ontología, debemos definir los conceptos principales y sus relaciones, así como la mejor forma de representarlas.

En este proceso aparecen cuestiones como por ejemplo qué metodología se tiene que escoger, cuál es la herramienta más apropiada o qué lenguaje hay que utilizar para describirla. Para dar respuestas a algunas de ellas, a continuación exponemos unas reglas fundamentales enfocadas al diseño y la construcción de ontologías basadas en N. F. Noy y D. L. McGuinness (2001):

- a) No existe ninguna metodología que podamos considerar perfecta o la mejor de todas.
- b) El proceso de construcción de ontologías es necesariamente iterativo.
- c) No hay una forma correcta de modelizar un dominio. Puede haber diferentes alternativas, pero todo dependerá del dominio y el uso que queramos hacer de la ontología
- d) Los conceptos en la ontología tienen que ser cercanos a los objetos (físicos o lógicos) y las relaciones en el dominio de interés. Probablemente son sustantivos (objetos) o verbos (relaciones) en las frases que describen el dominio.

A continuación listaremos unos pasos básicos que nos guiarán en el proceso de construcción de una ontología.

### **Paso 1 – Determinar el dominio y el alcance de la ontología**

Por medio de un conjunto de preguntas relacionadas con el uso de la ontología, pondremos límite al alcance del modelo:

- ¿Cuál es el dominio que se quiere cubrir?
- ¿Cuál es la finalidad de la ontología?
- ¿Quién la utilizará y quién la mantendrá?

### **Paso 2 – Considerar la reutilización de ontologías existentes**

Conviene comprobar si podemos construir una ontología a partir de una o algunas ya existentes, refinándola o ampliando sus límites para nuestro dominio o tarea en particular.

La reutilización puede venir como un requisito necesario, cuando nuestra aplicación tiene que interactuar con otras aplicaciones que estén empleando ontologías ya existentes.

Algunos sitios web donde podemos buscar ontologías ya creadas son:

- DAML
- Ontolingua
- Swoogle
- Hakia

### **Paso 3 – Enumerar los términos relevantes de la ontología**

Es importante listar todos los términos que tengan relación con nuestro dominio. Podemos quererlos utilizar para crear enunciados, o bien nos pueden interesar porque aparecen en las respuestas a las preguntas de verificación. A la hora de proponer términos, no debemos limitarnos solo a los conceptos del dominio, sino también a sus propiedades

### **Paso 4 – Definir las clases y la jerarquía**

Para realizar una jerarquía de clases podemos optar por utilizar diferentes metodologías, como explican M. Uschold y M. Gruninger (1996):

- **Top-down:** antes que nada se crearán las clases para definir los conceptos más generales del dominio, y posteriormente se irán creando las especializaciones de estos.



- **Bottom-up:** empezaremos por las clases más específicas, que serán las hojas de la jerarquía, y posteriormente las clases se irán agrupando en conceptos más generales.
- **Combinado de top-down y bottom-up:** se empieza definiendo los conceptos más destacados, y posteriormente se irán o bien especializando o bien generalizando.

El método más apropiado para cada caso concreto dependerá mucho del dominio concreto y de qué visión se tenga de él.

### **Paso 5 – Definición de las propiedades de las clases**

En este paso se definen las propiedades de cada una de las clases de la jerarquía. El paso 4 y el paso 5 están estrechamente ligados, y muchas veces es difícil realizar uno y después el otro. Lo que finalmente suele hacerse es crear unas cuantas definiciones de conceptos dentro de la jerarquía y a continuación describir sus propiedades.

### **Paso 6 – Definir las restricciones de las propiedades**

Cada propiedad puede tener diferentes restricciones que la describan o que caractericen el tipo de valor que admite la propiedad.

Algunas de las restricciones más habituales son:

- **Cardinalidad:** indica cuántos valores puede tener una propiedad.
- **Tipo de valor:** describe el tipo de valor que es admitido para asignar la propiedad.
- **Dominio y rango:** se denomina rango las clases que definen los posibles valores que puede tomar una propiedad. Se denomina dominio de una propiedad el conjunto de clases que tienen asociada aquella propiedad.

### **Paso 7 – Crear instancias**

Finalmente, el último paso consiste en crear las instancias individuales de las clases de la jerarquía. Para definir una instancia individual de una clase hace falta escoger una clase, crear una instancia individual de la clase y dar valor a las propiedades.

#### 2.2.4. Definición de una ontología en Protégé

Protégé es uno de los editores de ontologías más utilizados y apoyado por una gran comunidad de usuarios. Se trata de un editor libre de código abierto y multiplataforma desarrollado por la Universidad de Stanford, en concreto por el Centro Stanford para la Investigación Informática en Biomedicina (Center for Biomedical Informatics Research).

El editor ofrece una serie de herramientas para modelizar (crear, visualizar y manipular) el conocimiento en forma de ontologías destinadas a representar dominios o conocimiento para aplicaciones. Además, permite la exportación a una gran variedad de formatos, como RDF, RDF Schema, OWL y XML Schema.

El editor permite dos maneras de modelizar conocimiento: construyendo ontologías basadas en marcos o bien construyendo ontologías destinadas a la web semántica, particularmente con el formato OWL.

Una de las características que hay que destacar de Protégé es su escalabilidad y ampliabilidad, que ofrece la posibilidad de adaptar el editor a nuestras necesidades para trabajar con ontologías. Por ejemplo, la ampliabilidad permite desde crear interfaces específicas hasta ejecutar cualquier proceso que las manipule: fusiones, inferencias, etc. Para hacerlo, pone a disposición dos mecanismos: una arquitectura de extensiones basadas en Java y una API que puede ser llamada desde un programa externo.

Además, hace falta mencionar otras características, como son su robustez a la hora de construir y procesar de manera eficiente ontologías de gran volumen, edición colaborativa, y la gestión de versiones.

Una ontología se compone de los elementos mencionados en un apartado anterior: individuos, propiedades, clases y restricciones. Las posibilidades de desarrollo de cada uno dan lugar a distintas características y tipologías, que se detallan a continuación [9].

- a) **Individuos o instancias (individuals):** representan objetos del mundo real, en el dominio en el que estamos trabajando.
- b) **Propiedades (properties):** indican relaciones entre individuos o entre un individuo y un dato, y se detallan habitualmente a nivel de clase. Pueden crearse subpropiedades y existen tanto para un conjunto de individuos (esto es, propiedad o atributo de clase) como para individuos concretos (propiedad o atributo de instancia). Una propiedad tiene un dominio (domain) y un rango (range). En el dominio es el conjunto de individuos de una clase que se relaciona con el conjunto de individuos de otra clase, el rango. Existen tres tipos de propiedades:

- **Propiedades de anotación (annotation properties):** utilizadas para añadir información a clases, individuos y propiedades de objeto o dato. Entre estas propiedades, destacan las labels y los comments
- **Propiedades tipo dato (datatype restrictions):** vinculan individuos y valores (datos) definidos mediante un rdf literal o utilizando un esquema de datos XML. Para establecer estas relaciones se utilizan las restricciones tieneValor (hasValue restrictions), que se explican más adelante.
- **Propiedades tipo objeto (object type restrictions):** son relaciones binarias entre individuos. Características de estas propiedades son:
  - **Propiedad inversa (inverse property):** si una propiedad vincula A y B, su propiedad inversa relacionará B y A. Por ejemplo, esParteDe tiene como inversa tieneComoParte. En este caso, dominio y rango de la propiedad esParteDe serán rango y dominio, respectivamente, de tieneComoParte.
  - **Propiedad funcional (functional property):** como máximo, un individuo sólo se relaciona con un individuo. Este tipo de propiedad también se denomina de un único valor (single valued property) o feature.
  - **Propiedad inversa funcional (inverse functional property):** si una propiedad es inversa funcional, implica que su inversa también lo es.
  - **Propiedad transitiva (transitive property):** si A se relaciona con B y éste con C mediante una propiedad transitiva los individuos A y C están relacionados.
  - **Propiedad simétrica (symmetric property):** permite establecer relaciones de igualdad entre dos individuos. Si "María" esHermanaDe "Laura", "Laura" esHermanaDe "María"
  - **Propiedad asimétrica (asymmetric property):** si A se relaciona con B mediante una relación asimétrica no puede aplicarse la misma relación entre B y A
  - **Propiedad reflexiva (reflexive property):** se aplica en los casos en que un individuo se relaciona consigo mismo. Por ejemplo, una "Institución" puede investigarse a mi misma.
  - **Propiedad irreflexiva (irreflexive property):** incide en el caso contrario a la anterior. Por ejemplo, la relación "María" esHermanaDe "María" es irreflexiva (e imposible), una persona no puede ser hermana de sí misma
- c) **Clases (classes):** son sets que contienen individuos. Se trata de representaciones concretas de conceptos, que se describen mediante declaraciones que indican los requerimientos para ser miembro de éstas.

## **Tipos de clases:**

- **Clases disjuntas (disjoint classes):** son aquellas de las que un individuo no puede ser instancia simultáneamente.

Existen otros tipos de clases, definidas a partir de restricciones (restrictions), que describen el conjunto de individuos de dicha clase basándose en las relaciones en las que éstos participan.

- **Clases primitivas (primitive classes):** hace referencia a aquellas condiciones necesarias que tienen que reunir los individuos para formar parte de una clase. Estas condiciones son denominadas superclases o subclases, dependiendo de la versión de Protégé.
- **Clases equivalentes (equivalent classes):** Ahora bien, el simple hecho de cumplir las condiciones no implica obligatoriamente que un individuo sea miembro de cierta clase. Para ello, estas condiciones necesarias deben definirse como suficientes. De esta forma, cualquier individuo que satisfaga estas condiciones, será considerado miembro de dicha clase. Cuando una clase tiene, como mínimo una condición necesaria y suficiente, recibe el nombre de clase definida (defined class).
- **Clases enumeradas (enumerated classes):** se crean definiendo previamente el conjunto de individuos que forman parte de dicha clase.

## **d) Tipos de restricciones existentes**

- **Restricciones de cardinalidad (cardinality restrictions):** limitan el número de relaciones entre individuos. Existen tres tipos de restricciones de cardinalidad: al menos (at least), como máximo (at most), exactamente (exactly).
- **Restricciones tieneValor (hasValue restrictions):** permiten especificar relaciones entre individuos de una clase y valores. Es posible utilizar elementos matemáticos que concreten esta relación.
- **Restricciones de cuantificación (quantifier restrictions).** Dentro de las mismas se contemplan:
  - **Restricciones existenciales (existential restrictions ó some restrictions):** denominadas en OWL someValuesFrom, indican que los individuos del dominio de una propiedad se relacionan con al menos un (at least one) individuo del rango. Este tipo de restricción no es excluyente, los individuos del dominio pueden relacionarse con individuos de otras clases.

- **Restricciones universales (universal restrictions):** definida en OWL como `allValuesFrom`, señalan que los individuos de cierta clase sólo se relacionan con los individuos de otra clase, por lo que excluye cualquier relación con individuos de otras clases.

En cuanto a las convenciones para nombrar elementos, se recomienda la notación CamelBack, indica que todos los nombres de clases deben comenzar con una letra mayúscula y no debe contener espacios; los nombres de las propiedades comienzan en minúscula, sin espacios e inician las siguientes palabras con letra mayúscula.

Capturas de Protégé:

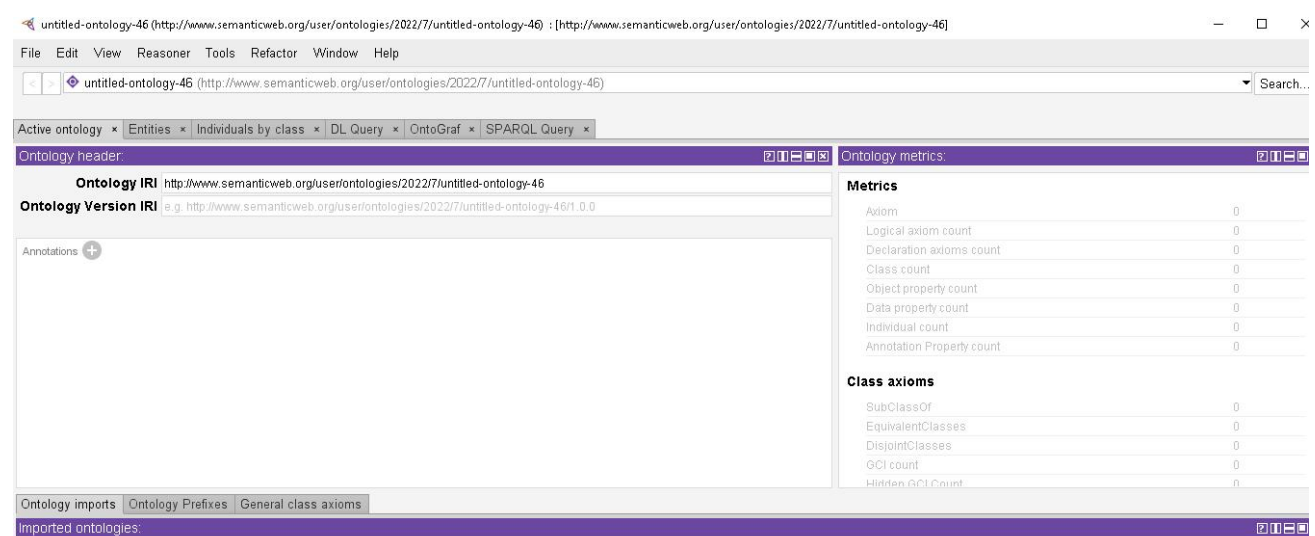


Figura 2.2: Visión general de Protégé.

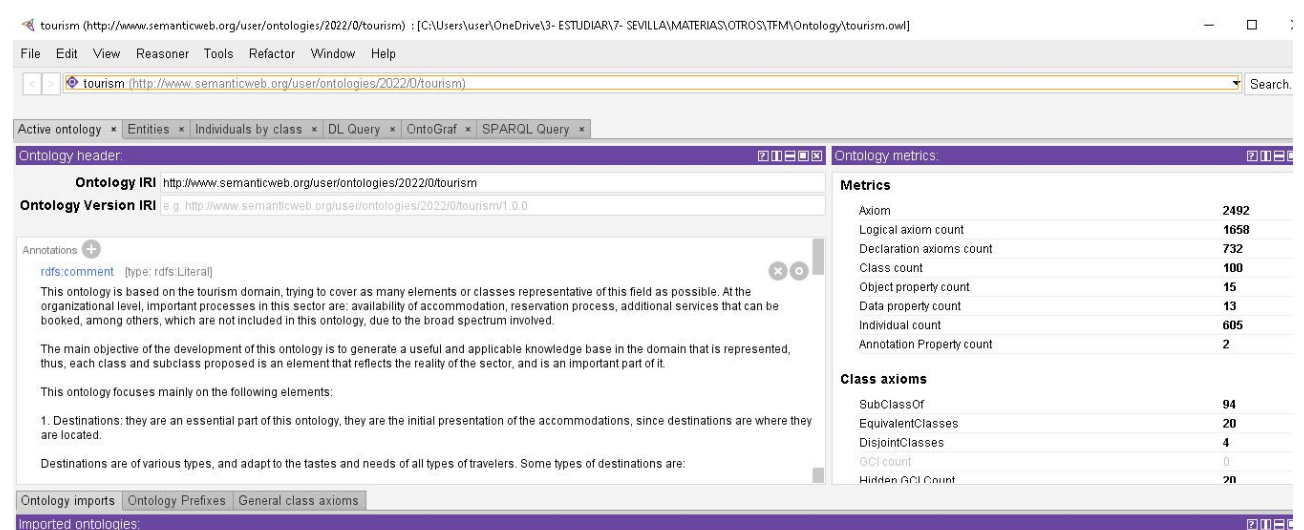


Figura 2.3: Visión general de Protégé con el dominio trabajado.

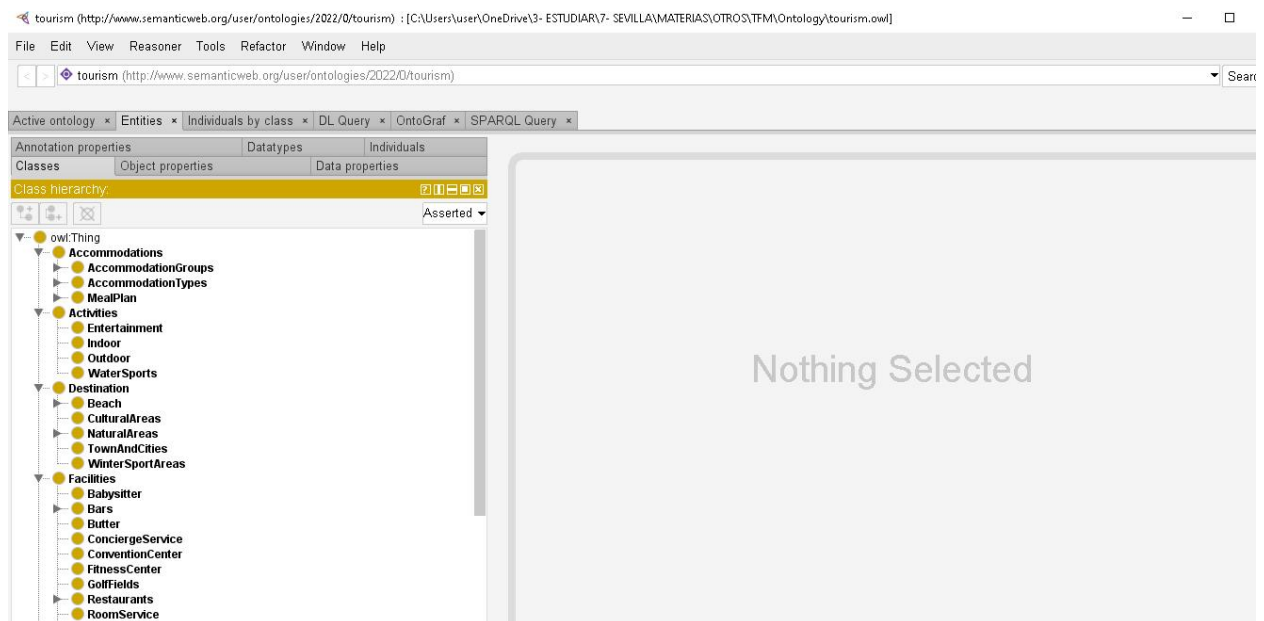


Figura 2.4: Taxonomía de clases.

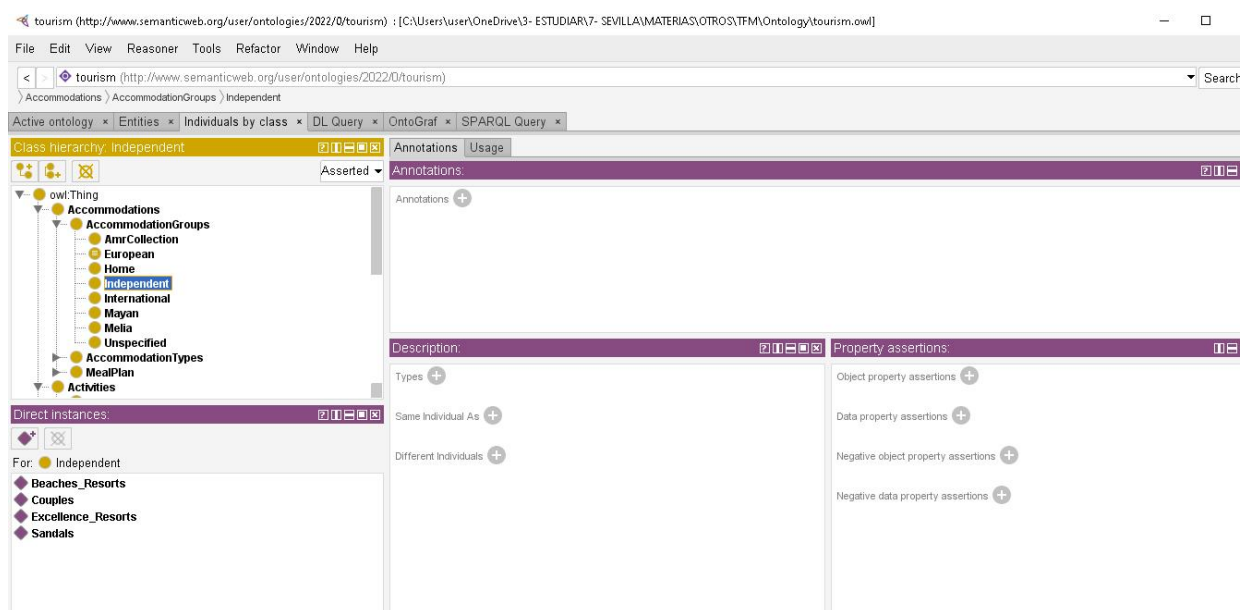


Figura 2.5: Visión general de Protégé con los individuos por clase.

### **3. Visión global de la ontología propuesta**

En este trabajo se propone el diseño e implementación de una ontología para la gestión del conocimiento en el sector turismo y hostelería.

El turismo se ha convertido en el sector que más riqueza aporta en muchos muchos países, e impacta de forma directa los demás pilares de la economía.

En la actualidad constituye un medio eficaz para la preservación de la naturaleza y los monumentos importantes de los países, de igual forma, es un motor activo en la mejora de la calidad de vida de las personas.

Se desarrolla esta ontología cubriendo los ejes centrales del dominio, y dejando la puerta abierta para que la misma pueda ser objeto de escrutinio y trabajo futuro.

#### **3.1. Introducción y características del dominio**

Esta ontología se basa en el dominio de turismo, tratando de abarcar la mayor cantidad de elementos o clases representativas de dicho ámbito. A nivel organizacional, procesos importantes de este sector lo son: disponibilidad de alojamiento, proceso de reserva, servicios adicionales que se pueden reservar, entre otros., los cuales no son incluidos en esta versión, por el espectro tan amplio que conllevan, y que indiscutiblemente llevarían a que el dominio de conocimiento sea dividido en varias ontologías, porque el alcance que tendría el mismo sobrepasaría la sencillez y claridad que se pide al momento de desarrollar una ontología.

El principal objetivo del desarrollo de esta ontología es generar una base de conocimientos útil y aplicable en el dominio que se representa, de esa forma, cada clase y subclase planteada es un elemento que refleja la realidad del sector, y que forma parte fundamental del mismo. Con una base sólida, se aspira a una mayor unificación en la información web sobre los elementos globales del área, de igual forma, lograr una mayor interoperabilidad entre cada uno de los actores.

Durante la realización de esta ontología se han consultado diferentes fuentes documentales, tales como la librería de ontologías de Protégé, donde aparece una ontología relacionada al dominio tratado, llamada '*Travel*', no se ha realizado una ampliación de la misma porque no contempla muchos de los elementos que abarca el dominio turismo, y porque no representa de forma estructurada la realidad que se pretende exponer a través de este informe.

### 3.2. Alcance

En esta ontología se abordan los principales elementos que componen el dominio, dejando fuera aspectos meramente administrativos del mismo.

Así, el diseño de esta ontología aborda desde el destino, como componente inicial, hasta los huéspedes, como consumidor final de todos los servicios y facilidades que se ofertan.

Por otro lado, la configuración de este proyecto se plantea en tres fases principales:

- **Análisis del dominio:** este proceso ha conllevado la consulta de diversas fuentes que contemplan las características que expone el sector, así como el estudio minucioso sobre la categoría y relación entre los diferentes elementos que participan. En esta etapa se utilizó la metodología “*Methontology*”, porque era la más apropiada frente a lo que se quiere lograr, además porque plantea unas fases en el ciclo de vida que ha permitido segmentar cada parte a desarrollar.
- **Diseño de la ontología en Protégé:** durante este proceso se llevaron a Protégé todas las entidades que se obtuvieron como resultado del análisis realizado, de igual forma se definieron las relaciones y restricciones existentes entre las clases y los individuos.
- **Implementación de la ontología en la web:** en esta fase se tomó el archivo generado por Protégé como principal fuente de datos, para generar un sistema que permita extraer cada elemento, y representarlo en la web. Se han seleccionado las tecnologías más adecuadas, y que mejores resultados ofrecen en el proceso de implementación, de igual forma, se han utilizado las técnicas y metodologías de programación que permitan obtener un producto de calidad, acorde a lo que se espera.



### 3.3. Clases y jerarquías

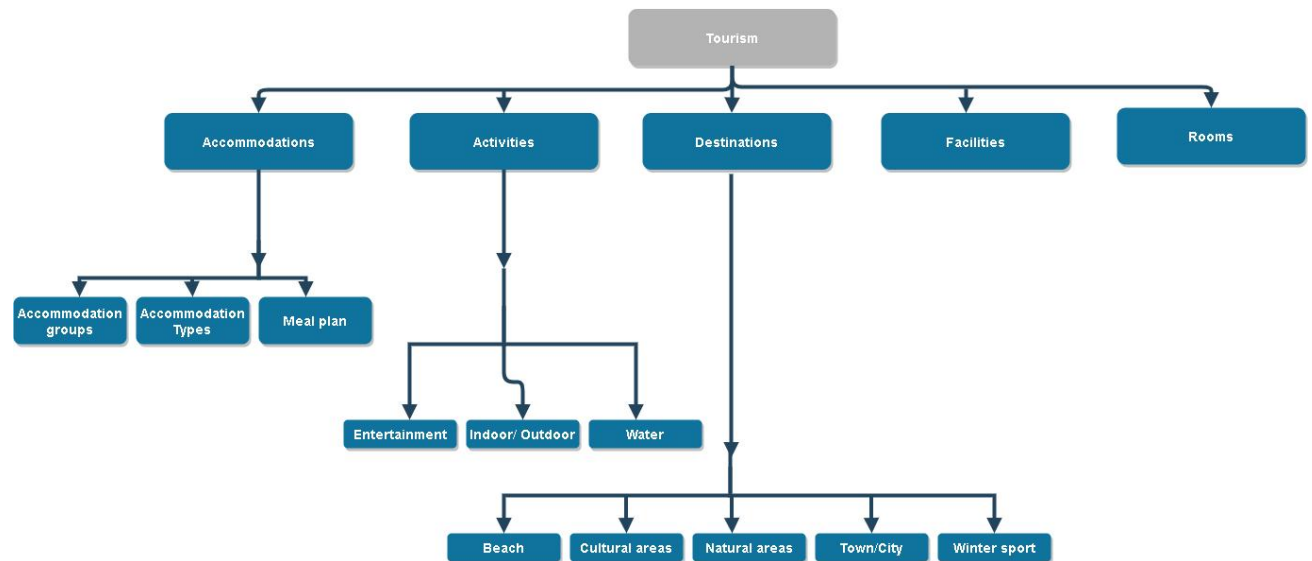


Figura 3.1: Elementos principales de la ontología diseñada.

Las clases y subclases obtenidas como resultado del análisis realizado al dominio seleccionado se detallan más adelante:

1. **Destinos:** son parte esencial del sector, es donde están ubicados los alojamientos.

Los destinos son de varios tipos, y se adaptan a los gustos y necesidades del amplio espectro de viajeros. Algunos tipos de destinos son:

- Playa: este tipo incluye alojamientos de playa, islas, playas solitarias.
- Areas culturales
- Areas naturales: contempla en sí las zonas rurales, bosques y montañas.
- Pueblos y ciudades
- Zonas de deporte de invierno

2. **Alojamiento:** esta clase contempla los grupos y tipos de alojamiento, así como el régimen de comida que ofrecen en sus diferentes opciones a los turistas.

Las subclases que se manejan son las siguiente:

- **Grupos de alojamiento:** generalmente conocidos como grupos hoteleros, cadenas hoteleras, o sencillamente grupo asociado a un alojamiento., es la entidad superior a que pertenece un alojamiento específico. A nivel de turismo es utilizado para segmentar a nivel de marca los alojamientos.

- **Tipos de alojamientos:** algo que representa el sector turismo en la actualidad es la diversificación que ha logrado, por ello la existencia de este elemento, y no es más que el formato de alojamiento a que el huésped puede acceder para alojarse. El mismo contempla los siguientes elementos o conceptos: Agro-turismo, Hoteles boutique, Bungalow, Campamentos, Casas de huéspedes, Hostales, Hoteles, Moteles, Resorts, Casa de turistas, Hostales para jóvenes.
  - **Plan de alimentación:** esto corresponde a los diferentes planes de comida que ofrece el alojamiento. Esto influye directamente en el precio del mismo, dependiendo de los servicios que ofrece, el monto a pagar por alojarse o reservar puede aumentar. Esta clase contempla; sólo alojamiento, todo incluido, servicio de cama y desayuno, privilegio infinito, pensión completa, media pensión, experiencia ilimitada.
3. **Actividades:** todos los alojamientos cuentan con diversas actividades que los hacen atractivos para los huéspedes o turistas. Las actividades ofrecidas pueden ser de tipo: entretenimiento, bajo techo, al aire libre, actividades de agua. Una actividad puede ser de varios tipos a la vez.
4. **Habitaciones:** son espacios cerrados dentro de los alojamientos, que brindan comodidad y lujo a los huéspedes durante su estadía. Las habitaciones son de diversos tipos, y para todo los tipos de gustos. Dependiendo del tipo de habitación, el precio de reserva puede variar. Cada tipo de habitación cuenta con amenidades específicas, y que son parte de ellas.

Algunos de los tipos de habitaciones más conocidos son:

- |                      |            |          |
|----------------------|------------|----------|
| ■ Deluxe             | ■ Quad     | ■ Suite  |
| ■ Double             | ■ Queen    | ■ Triple |
| ■ King               | ■ Single   | ■ Twin   |
| ■ Penthouse suite    | ■ Standard |          |
| ■ Presidential suite | ■ Studio   |          |

5. **Conocimiento general:** esta clase es utilizada para reguardar datos y conocimientos utilizados y/o generados por otras clases de la ontología.

En este apartado se cubren elementos tales como:

- **Alojamientos por continentes**
- **Alojamientos que son solo para adultos**
- **Rating de alojamientos:** esto es conocido popularmente como estrellas del alojamiento, es un indicador del prestigio y nivel de calidad del mismo. Este concepto es solo aplicado a alojamientos como hoteles, Resorts,

Bungalow, entre otros., los otros tipos de alojamiento solo manejan puntuaciones dadas por los usuarios.

Otros elementos sumamente importantes en este dominio, y que quedan como puntos de trabajo futuro para nuevas versiones, no abordados en esta ontología, para mantener una estructura y cohesión adaptada a lo solicitado, son los siguientes:

- **Turistas o huéspedes:** son las personas que ocupan y dan uso a los elementos que constituyen el sector turismo, sin ellos el sector no sería posible.
- **Disponibilidad:** se centra en presentar cuáles alojamientos poseen habitaciones disponibles para un período de tiempo determinado, y especificado por los turistas.
- **Reservación:** es el proceso en que un turista paga por una habitación en un alojamiento específico. Sin este proceso en particular no se podría realizar la relación directa entre un turista y una habitación de un alojamiento, en un periodo de tiempo determinado.

### 3.4. Propiedades

En este apartado se describen las diferentes propiedades que asocian las entidades definidas, de igual forma, se detallan las restricciones aplicadas sobre los elementos del dominio.

#### 3.4.1. Propiedades de objetos

Estas propiedades nos permiten definir las relaciones que se utilizan para asociar diferentes objetos, a continuación se detallan:

- **accommodationsUnder:** esta propiedad tiene como dominio los destinos, y como rango los tipos de alojamiento. El propósito de la misma es relacionar los tipos de alojamiento con su destino padre. Tiene como inversa *isLocated*. Tiene las características: inversa funcional, porque los destinos se definen en base a los alojamientos circunscritos a él. Mientras que *isLocated* posee la característica de ser funcional, porque un alojamiento solo puede estar ubicado en un destino.
- **actOfferedBy:** tiene como dominio las actividades, y como rango los tipos de alojamiento. Su objetivo es mostrar o listar las actividades que ofrece un alojamiento. Tiene como inversa la propiedad *offerActivity*, y esta posee la

característica de ser transitiva, porque me interesa relacionar los alojamientos que ofrecen un tipo de actividad parecida.

- **hasRoom:** tiene como dominio tipos de alojamiento y como rango las habitaciones. Muestra los tipos de habitaciones que posee un alojamiento. Tiene como inversa *actWithRoom*.
- **underBrand:** tiene como dominio los tipos de alojamiento, y como rango los grupos de alojamiento. Relacione cada tipo de alojamiento con el grupo o cadena a que pertenece. Hay unos tipos de alojamiento que no cuentan con esta propiedad. Es funcional porque un alojamiento solo puede estar bajo una marca; asimétrica, porque se relacionan mutuamente; e irreflexiva porque una marca no puede relacionarse a ella misma, son los alojamientos los que se relacionan con ella. Posee como inversa *brandAccommodation*.
- **hasFacility:** muestra las facilidades con que cuenta una habitación o tipo de alojamiento. Tiene como subpropiedad a *roomFacility*, la he colocado a este nivel por la relación existente entre ambas.
- **hasRestaurant:** usada para listar los individuos de restaurantes que tiene un tipo de alojamiento.
- **offer:** tiene como dominio los tipos de alojamiento, y como rango los planes de comida. Tal y como se ha descrito anteriormente, eso muestra los diferentes planes que se pueden seleccionar al hacer la reserva de un alojamiento. Tiene como inversa la propiedad *offeredBy*.

#### 3.4.1.1. Propiedades de datos

Estas propiedades permiten agregar los diversos componentes con que cuenta un individuo, a continuación se detallan:

- **amenities:** tiene como rango las habitaciones, y como dominio un tipo de dato cadena. Es utilizada para listar las amenidades con que cuenta una habitación.
- **awards:** tiene como dominio los tipos de alojamiento, y como rango un listado de posibles premios que puede ganar un alojamiento.
- **brandCode:** propiedad funcional, con dominio en los grupos de alojamiento, y con rango una cadena de caracteres. Designa el código del grupo.

- **code:** usada para asignar un código a los alojamientos. Tiene como dominio los tipos de alojamiento, y como rango una cadena de caracteres, es de tipo funcional.
- **continent:** tiene como rango los tipos de alojamiento, y como dominio el listado de continente. Usada para mostrar el continente donde está ubicado un alojamiento.
- **country:** propiedad con dominio en tipos de alojamiento, y con rango una cadena de caracteres., es de tipo funcional, y se usa para designar el país donde está ubicado un alojamiento.
- **hasStarts:** con dominio en algunos tipos de alojamiento (Resorts, Bungalow, Hotel, HotelRating, Boutique), y con rango en un entero. Asigna las estrellas que posee un tipo de alojamiento.
- **maxPrice:** precio máximo de un alojamiento. Tiene dominio en los tipos de alojamiento, y rango en un valor tipo double.
- **minPrice:** precio mínimo de un alojamiento. Tiene dominio en los tipos de alojamiento, y rango en un valor tipo double.
- **numberRooms:** señala la cantidad de habitaciones que posee un alojamiento. Tiene rango en los tipos de alojamiento, y dominio en un valor entero.
- **onlyAdults:** propiedad con dominio en los tipos de alojamiento, y rango en un valor booleano. Utilizada para mostrar si un hotel solo acepta adultos o no.
- **visitors:** propiedad con dominio en los tipos de alojamiento, y rango en un valor entero. Muestra la cantidad de visitantes que ha tenido un alojamiento durante el último año.

### 3.5. Uso

Esta ontología ofrece una base de conocimientos amplia y totalmente aplicable al desarrollo de estructuras en el sector turismo, sirve de referente para que un usuario pueda conocer como está fundamentado el dominio abordado de forma sencilla y rápida.

El uso de esta ontología se basa principalmente en que exhibe la realidad, hace un uso correcto del lenguaje asociado al turismo, y representa todos los conceptos del área. Más que ver de forma teórica los pilares del sector turismo,

podemos situarnos en el funcionamiento del mismo, a través de las entidades creadas, y la relación funcional entre ellos.

Un aspecto importante sobre este trabajo es la relación que se crea con la inteligencia artificial, el objetivo principal de la web semántica es que la web pueda convertirse en una base de conocimientos globales que puedan ser interpretados tanto por humano como por máquinas. En este orden, el uso de la inteligencia artificial será primordial para que la web 3.0 logre el alcance que tienen previsto sus autores. Cuando se realiza una búsqueda, las máquinas por medio de redes neuronales podrán ser capaces de entender y procesar todo el conocimiento existente en la web, de esta forma podrán satisfacer con un alto grado de acierto a la petición que realiza un usuario, y retornar solo aquellos resultados que correspondan al contexto específico de la búsqueda.

La integración de la inteligencia artificial con la web semántica permite comprender mejor a los usuarios a la vez que se les proporciona una mejor experiencia [14]. El entendimiento del lenguaje natural va a permitir comprender mucho mejor las necesidades y expectativas de los usuarios.

Partiendo de lo antes expuesto, esta ontología del sector turismo permitirá que se comprendan las necesidades de los clientes, a la vez que promueve un crecimiento del sector, y por ende mayores y mejores beneficios.

### **3.6. Toma de decisiones**

Llevar a cabo el diseño de cualquier ontología requiere de un proceso de deconstrucción del dominio, hasta el punto que el diseño de la misma se corresponda sin ambigüedades con la realidad que se intenta representar, es por esto, que al momento de hacer una definición de las clases, subclases, propiedades, y cualquier otro elemento parte del dominio que se trabaja, es necesario tener conocimiento pleno sobre el posicionamiento que debe llevar cada elemento, y sobre la importancia de tomar la decisión correcta en este proceso de diseño.

La definición de las clases ha sido en base a las entidades reales del dominio, inicialmente ha sido en base a un diagrama de como está estructurado el sector turismo, y partiendo de eso se han creado las clases necesarias para la ontología.

Algunos casos concretos donde se ha decidido si utilizar una clase o no son:

- **Actividades:** si bien esta clase tiene como subclases unos tipos concretos de actividades, cada una de estas a su vez podrían traer asociadas las actividades específicas. Se ha decidido colocar las actividades puntuales como individuos, porque los mismos no varían de nombre en cada alojamiento, y en los casos que lo hacen, el sentido de la misma continúa

siendo igual. Por otro lado, es más factible asociar a cada alojamiento la actividad como se hace en esta ontología.

- **General Knowledge (Conocimiento General):** la decisión de crear esta clase tiene como objetivo colocar debajo de ella algunas de las clases que se relacionan con otras, pero que son elementos utilitarios y de importancia en el sector. Un elemento que se adapta a lo antes descrito son las estrellas recibidas por un alojamiento; si bien es cierto que popularmente se conocen con números, a nivel representativo las mismas representan una cualidad, en este caso inferida en la ontología.
- **Restaurantes:** inicialmente se ha considerado solo crear la clase base, y luego crear los tipos de cocina como individuos. Más adelante, ha surgido la necesidad de crear los tipos de cocina como subclases, porque las mismas son genéricas, pero dependiendo de cada alojamiento puede variar el nombre del mismo, y las condiciones con las que se ofrece al huésped pueden ser diferentes.
- **Marcas de alojamientos:** las marcas son un identificador a nivel publicitario que representa el alojamiento, y que a su vez se asocia a un grupo hotelero. La decisión de colocar estas marcas como individuos recae en que son genéricas sin importar el contexto o situación, y solo es un dato asociado al alojamiento, que representa una asociación explícita al grupo de mayor jerarquía.

### 3.7. Representación gráfica

El ontogrf de esta ontología resulta muy amplio, por lo que he segmentado cada parte de la misma de acuerdo a la clase que representa.

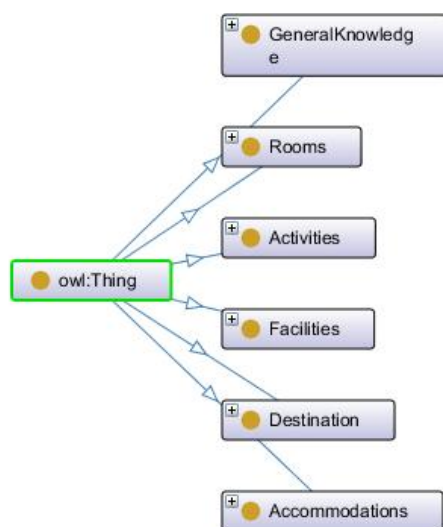


Figura 3.2: Clases principales de la ontología.

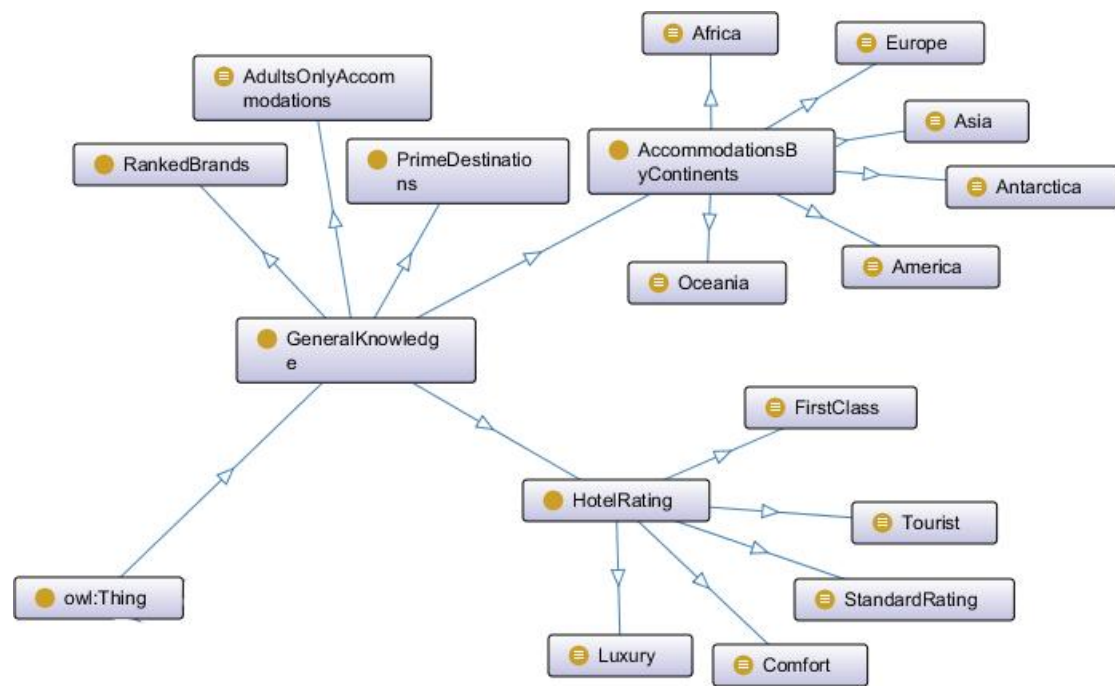


Figura 3.3: Clase de conocimiento general.

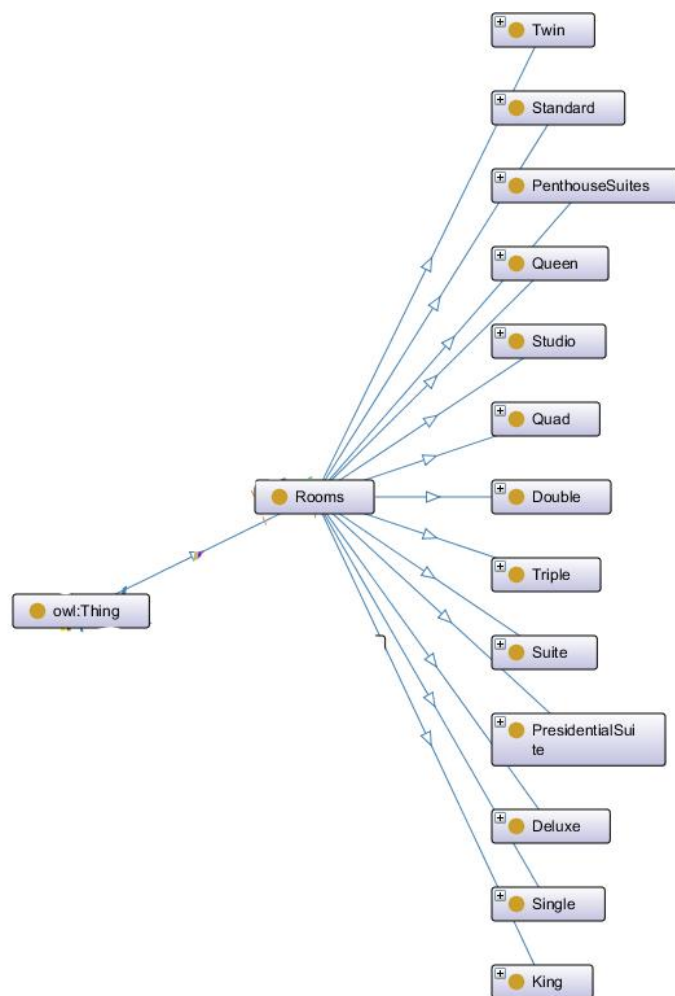


Figura 3.4: Clase de Rooms o habitaciones.



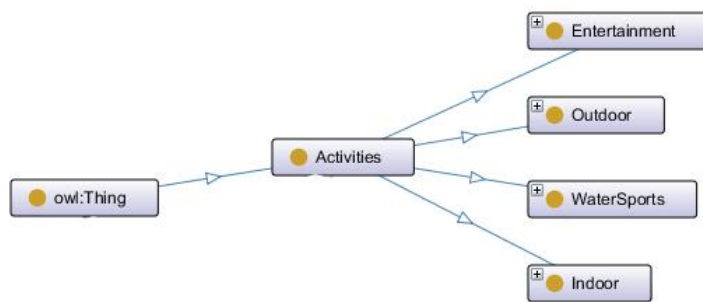


Figura 3.5: Clase de activities.

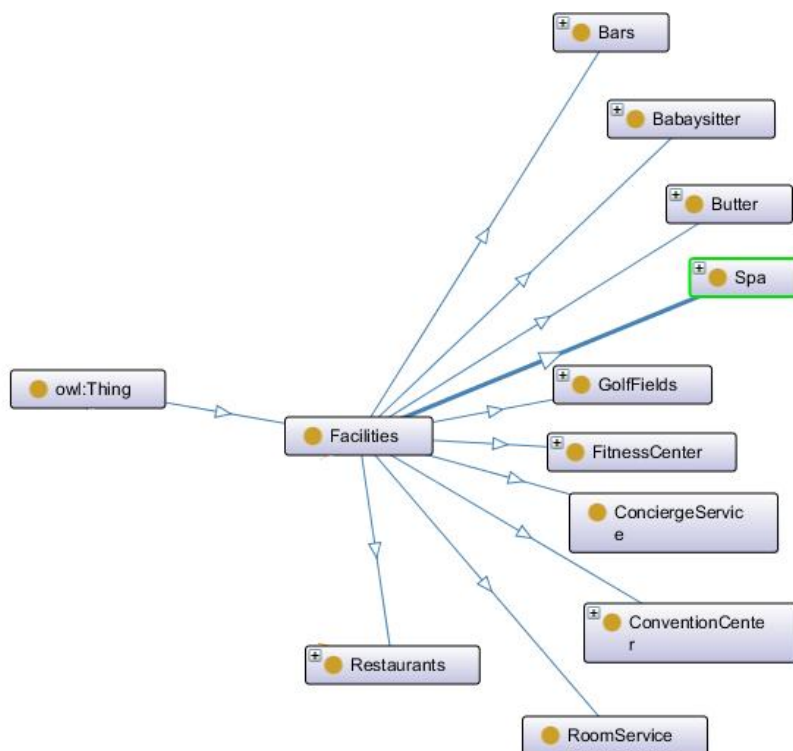


Figura 3.6: Clase de Facilities.

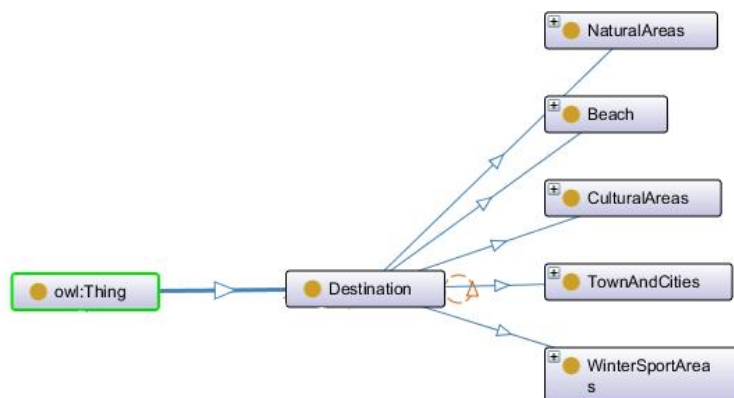


Figura 3.7: Clase de Destinations.

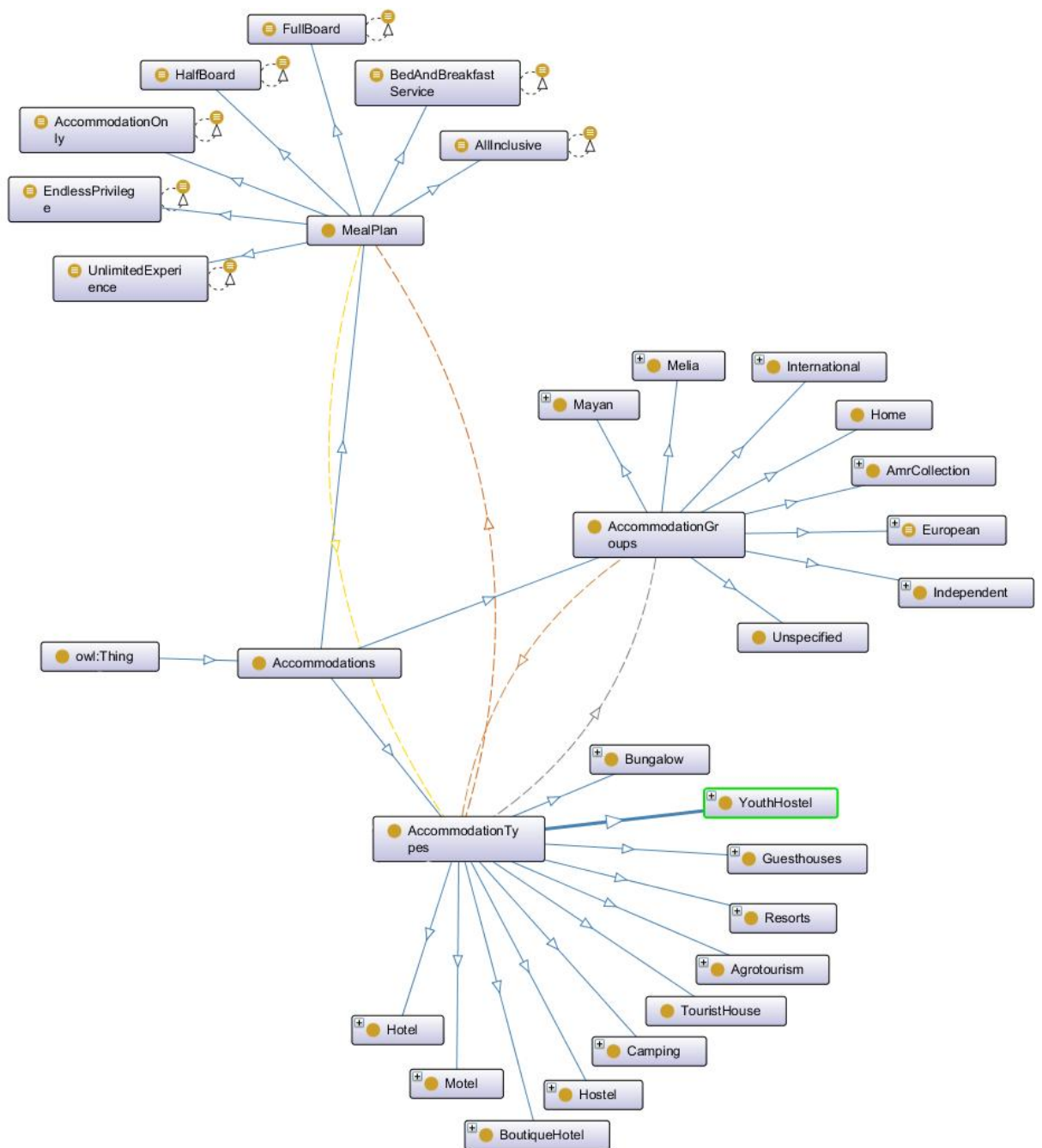


Figura 3.8: Clase de Accommodations.

## 4. Representación Web de la ontología

Esta sección trata los aspectos técnicos de la implementación de la ontología en la página web. Se estará abordando desde la arquitectura del proyecto: tecnologías, frameworks, herramientas, lenguaje de programación, principios aplicados, etc., hasta la organización y estructura de dicho proyecto, a fin de detallar los elementos que lo componen y forman parte de la web semántica.

### 4.1. Arquitectura general del sistema

La implementación de este proyecto se asienta sobre tecnologías Microsoft, utilizando así .Net como la principal tecnología, y Visual Studio Community 2022 como la plataforma de programación por defecto.



Figura 4.1: Visual Studio Community 2022.

#### 4.1.1. Aspectos metodológicos de implementación

Todo proyecto bien organizado hace uso de un sistema de versiones, con el que se pueda asegurar la total disponibilidad del código, además de un control eficaz sobre los cambios aplicados en todo el ciclo de vida del desarrollo; en ese sentido, para este proyecto se hace uso de GIT, un sistema distribuido, conectado, conectado a un repositorio en GitHub; donde se puede encontrar el código y documentación de este proyecto [26].



Figura 4.2: Sistema de control de versiones GIT.

Por otro lado, se ha hecho uso de metodologías ágiles, que han permitido llevar un control más detallado sobre cada tarea a realizar, y sobre el tiempo a dedicar a cada una de ellas. Estas metodologías promueven un trabajo organizado y escalable a través del tiempo.

La planificación de tareas de este proyecto se lleva a cabo utilizando Trello, una plataforma gratis, y que brinda características muy potentes para crear tareas, y hacer un seguimiento efectivo de estas.



Figura 4.3: Logo de la herramienta de trabajo Trello.

El listado de tareas planificadas y ejecutadas durante este proceso queda como sigue:

#### **Fase de análisis de requerimientos**

- Realizar análisis sobre cómo integrar la ontología en una página web
- Determinar tecnologías a utilizar
- Definir y hacer prototipo del diseño de las interfaces

#### **Fase de desarrollo de API**

- Crear estructura del proyecto siguiendo DDD
  - Crear capa de aplicación y definir objetos
  - Crear capa de infraestructura y objetos
  - Crear capa de modelos y objetos
  - Crear capa de web y objetos
  - Crear capa de API y objetos
- Definir y crear entidades del modelo en base a la ontología
- Crear repositorio y hacer lectura de ontología
- Crear servicio y parseo de la ontología
  - Convertir entidades al tipo de modelo a retornar
  - Procesar entidades con conocimiento por inferir
- Crear controladores del API
  - Crear controlador y retornar modelo
  - Definir método para filtrado sobre el modelo

#### **Fase de desarrollo de aplicación Web**

- Definir y crear estructura de vistas y controladores
- Agregar opción de multi-lenguaje
- Establecer comunicación con el API
- Cargar datos del modelo en la vista
- Agregar opción de filtrar sobre el modelo

## Fase de prueba y publicación

- Definir y crear pruebas unitarias
- Corrección de errores resultantes de las pruebas
- Publicar API y aplicación en Azure
- Realizar pruebas finales
- Aplicar correcciones finales sobre errores y detalles observados

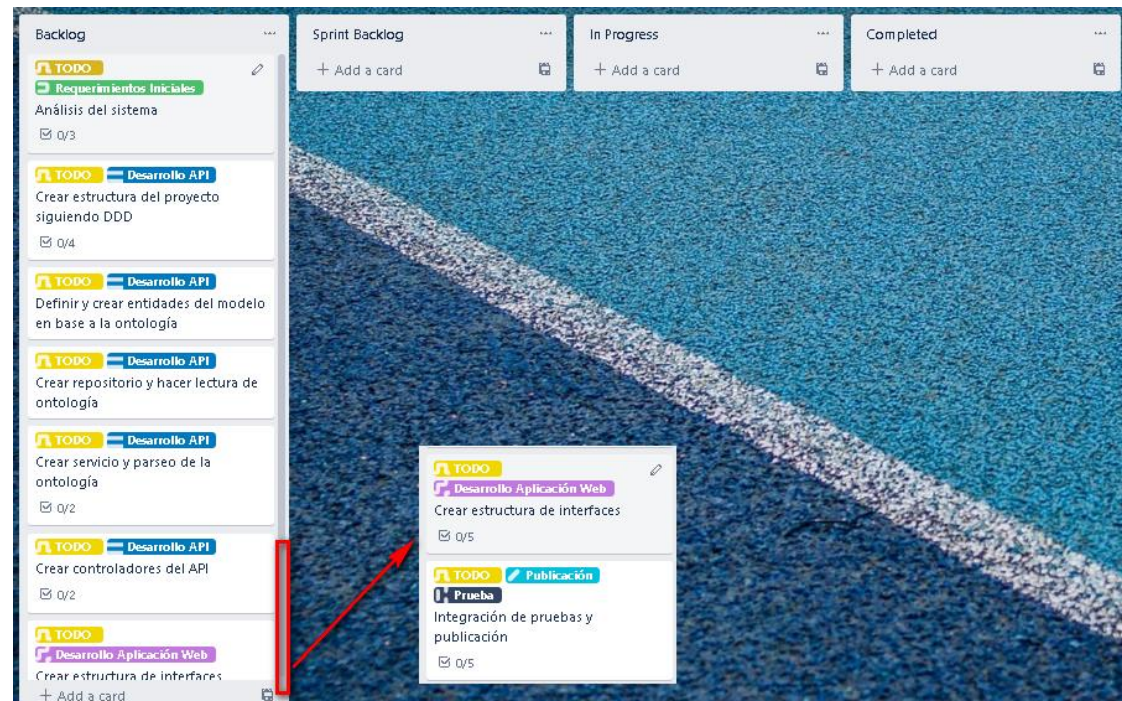


Figura 4.4: Listado de tareas creadas en Trello.

Otro de los aspectos fundamentales en que debe basarse la correcta ejecución de un sistema es el uso de estrategias, principios y metodologías que establezcan parámetros de calidad medibles sobre la forma en que se ha codificado. En la implementación de este sistema se ha hecho uso principalmente de los principios SOLID, que orientan hacia el uso de buenas prácticas que pueden ayudar a escribir un mejor código: más limpio, mantenible y escalable.

Algunas de las buenas prácticas contempladas en este desarrollo son:

- Uso de estándares, y nombres descriptivos en las variables y funciones.
- Uso de clases de utilidad para tareas comunes.
- Correcto espaciado y agrupación de código.
- segmentación de las tareas por clases o entidades.
- Control sobre las excepciones.

#### 4.1.2. Arquitectura lógica del proyecto (Backend)

Este desarrollo se lleva a cabo utilizando C# como lenguaje de programación único, como parte de la plataforma .Net, y siendo un lenguaje tipado y de alto nivel, posee la potencia y herramientas necesarias para que se pueda llegar un producto final de calidad y altamente eficiente. Este lenguaje de programación se utiliza con Net 6 como principal framework, que brinda un entorno de ejecución de aplicaciones, así como el ecosistema de bibliotecas base para la construcción de todo tipo de proyectos. Este framework se caracteriza por brindar un desarrollo simplificado, alto rendimiento, productividad bien definida y un entorno multiplataforma y de fuente abierta.



Figura 4.5: Lenguaje de programación C# y Plataforma .Net 6.

Uno de los puntos cruciales de este sistema es la lectura y carga de la ontología en entidades propias del sistema, donde se pueda extraer la información explícita y por inferir, para realizar una construcción dinámica de todas las entidades y objetos; para llevar a cabo esta tarea se está haciendo uso de dotNetRDF, una biblioteca completa para analizar, administrar, consultar y escribir RDF, manejado en .NET a través de una interfaz de aplicación común por medio del manejador de paquetes estándar. Este paquete es sumamente importante para este proyecto, sin él no hubiese sido posible dar el tratamiento correcto a la ontología.



Figura 4.6: Librería dotNetRDF.

Las clases principales de la biblioteca dotNetRDF se pueden encontrar en el espacio de nombres VDS.RDF. Todas las clases principales se basan en interfaces o clases abstractas para hacer que la biblioteca sea lo más extensible posible.



Estas interfaces clave son las siguientes [24]:

- **INode**: representa un nodo en un gráfico RDF, esto a veces se denomina término RDF. La interfaz es bastante escasa y proporciona principalmente información sobre el tipo de nodo y el gráfico al que está asociado. Como se indicó, cada término RDF se puede tratar como un nodo en un gráfico. Como tales, todos los términos RDF se modelan como implementaciones concretas de la interfaz INode y de una subinterfaz relevante de la lista anterior, p. IUriNodo.
- **IGraph**: se puede considerar que un documento RDF forma un gráfico matemático y, por lo tanto, representamos conjuntos de tripletas RDF como gráficos. Todos los gráficos de la biblioteca son implementaciones de la interfaz IGraph y generalmente se derivan de la clase BaseGraph abstracta que implementa algunos de los métodos principales de la interfaz, lo que permite que las implementaciones específicas se concentren en aspectos específicos, como la persistencia en el almacenamiento/seguridad de subprocesos. Una implementación de IGraph es una representación en memoria de un documento RDF.
- **ITripleStore**: un Triple es la unidad básica de datos RDF, los nodos por sí solos no tienen significado, pero se usan en forma de Triple como una declaración que afirma algún conocimiento. Una Triple está formada por Sujeto, Predicado y Objeto. Se interpreta como afirmando que algún Sujeto está relacionado con algún Objeto por una relación especificada por el Predicado. Los componentes de la clase Triple son Nodos, lo que significa que cualquiera de las clases de Nodos discutidas en la sección anterior se puede usar en cualquiera de las posiciones.

## Inferencia y Razonamiento con dotNetRDF

La inferencia y el razonamiento son mecanismos mediante los cuales una aplicación puede descubrir información adicional que no se establece explícitamente en los datos iniciales.

La librería dotNetRDF implementa estos conceptos por medio de una interfaz llamada IIInferenceEngine, que permite razonamiento sobre las reglas definidas, así como la creación de reglas dinámicas, en función de los datos de entrada.

La interfaz IIInferenceEngine tiene dos métodos principales que los implementadores deben implementar para integrar sus propios razonadores en dotNetRDF. El primero de ellos es el método Initialise(IGraph g) que se usa para ingresar gráficos al razonador que definen el esquema/reglas que debe seguir el razonador. El razonador puede procesar e interpretar este gráfico de la forma que desee para generar las reglas que utilizará cuando aplique la inferencia a un gráfico .

El segundo método es el método Apply() que aplica la inferencia a un gráfico que genera los triples inferidos en el mismo gráfico o en otro gráfico. Para los implementadores, este método es donde se ubicará la lógica central del razonador [25].

Otra parte importante de este proyecto es el tratamiento sobre los modelos generados a partir de la ontología previamente cargada, en este punto es donde entra el concepto de API (Application Programming Interface), o Interfaz de programación de aplicaciones, que provee de los recursos necesarios para una integración entre diferentes servicios.

Se hace uso de servicios REST (Representational State Transfer) o Transferencia de Estado Representacional, que proveen de una interfaz para conectar varios sistemas basados en el protocolo HTTP (uno de los protocolos más antiguos) y nos sirve para obtener y generar datos y operaciones, devolviendo esos datos en formatos muy específicos, como JSON (Javascript Object Notation) o notación de objetos javascript.



Figura 4.7: Servicios REST (Transferencia de Estado Representacional).

De forma principal se utiliza REST porque se está creando tanto una aplicación que administre la lógica del sistema, como un cliente encargado de consumir; se hace de esta forma siguiendo los esquemas actuales de programación, y porque eso garantiza una mejor interconectividad, y una mayor delegación de funciones entre cada aplicación.



Figura 4.8: Estándar para transferencia de datos JSON.

Con respecto a JSON, método de intercambio de información mencionado previamente, se escoge por varias ventajas que otros no ofrecen, tales como:

- Es autodescriptivo y fácil de entender.
- Su sencillez le ha permitido posicionarse como alternativa a XML.
- Es más rápido en cualquier navegador.
- Es más fácil de leer que XML.
- Es más ligero (bytes) en las transmisiones.
- Se realiza la conversión más rápido.
- Velocidad de procesamiento alta.



Con los puntos a favor mencionados previamente se obtendrá un sistema rápido y robusto.

Toda aplicación que hace uso de servicios basados en REST debe cumplir unos estándares y especificaciones mínimas, tales como: aplicar buenas estrategias de seguridad ante un cliente que intente establecer comunicación, y por otro lado, ofrecer una documentación clara y legible, que sirva como la principal guía para aquellos que intenten consumirlos. Basado en el principio de la seguridad de hace uso de JWT Bearer (JSON Web Token), que es un estándar de autorización, donde se codifica la información a enviar utilizando unas claves conocidas tanto por el cliente como proveedor, asegurando así una comunicación segura. Con respecto al principio de buena documentación, se hace uso de Swagger, una especificación abierta para definir la aplicación, además especifica la lista de recursos disponibles y las operaciones a las que se puede llamar en estos recursos.



Figura 4.9: Especificación abierta para documentación con Swagger.

### 4.1.3. Arquitectura visual del proyecto (Frontend)

La parte visual de este proyecto conjuga un conjunto de frameworks y tecnologías de vanguardia, con el objetivo de obtener una solución sencilla y agradable a la vista.

Una de arquitecturas más utilizadas en la actualidad para la construcción de sitios web es MVC (Model-View-Controller o Modelo-Vista-Controlador), utilizada para separar el código por sus distintas responsabilidades, manteniendo distintas capas que se encargan de hacer una tarea muy concreta, ayudando a crear aplicaciones con mayor calidad. Este patrón es utilizado en la parte visual de este proyecto, por todo lo que ofrece a la lógica de la aplicación.

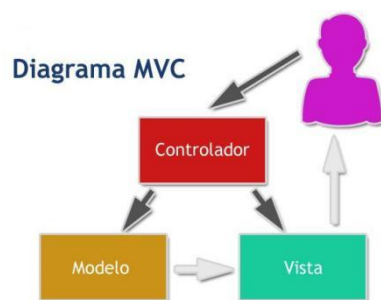


Figura 4.10: Diagrama MVC de una aplicación.

Se hace uso del framework Bootstrap, con la finalidad de brindar una interfaz adaptable a cualquier dispositivo, y que a la vez mantenga la consistencia de una página web moderna y elegante. Este framework aporta potentes funcionalidades de estilo e interactividad a la web.

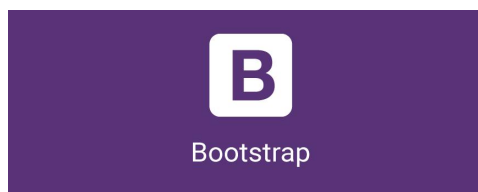


Figura 4.11: Framework para el manejo de estilos Bootstrap.

Para hacer la página más rápida e interactiva a la vista del usuario final, se utilizan tecnologías basadas en el cliente, como lo son:

- **Javascript:** es un lenguaje de programación que agrega acciones interactivas, y que funciona en los navegadores de forma nativa.



Figura 4.12: Lenguaje de script en el navegador JavaScript.

- **jQuery**: es una librería que permite añadir una capa de interacción AJAX (Asynchronous JavaScript and XML) entre la web y las aplicaciones que desarrollemos controlando eventos, creando animaciones y diferentes efectos para enriquecer la experiencia de usuario. **Ajax** permite que un usuario de la aplicación web interactúe con una página web sin la interrupción que implica volver a cargar la página web.



Figura 4.13: Librerías jQuery y AJAX.

- **Vue.js**: es un framework progresivo para construir interfaces de usuario, que permite mayor interacción entre el usuario y la aplicación.



Figura 4.14: Framework para manejo de interfaces Vue.js.

Tanto la capa lógica (expuesta mediante un servicio REST), como la capa visual deben comunicarse en un punto del proceso, para que el sistema web pueda presentar al usuario la información que se ha procesado, y de esa forma este pueda manipularla. Esta implementación hace uso de una herramienta llamada Refit, que permite convertir servicios web REST en interfaces vivas, de forma que se pueda modelar el servicio a consumir a través de atributos, así que Refit se encarga de los detalles de cómo consumirlo apropiadamente.



Figura 4.15: Manejo de solicitudes con Refit.

#### 4.1.4. Capas del proyecto

La implementación de este proyecto se basa en el diseño basado en el dominio (DDD o Domain Driven Design), con una arquitectura basada en capas, que permite una mejor organización y distribución de cada componente del mismo.

El diseño basado en el dominio (DDD) es una práctica de desarrollo de software que pone el acento en el Dominio del Negocio como faro del proyecto y en su Modelo como herramienta de comunicación entre negocio y tecnología.

El DDD provee una estructura de prácticas y terminologías para tomar decisiones de diseño que enfoquen y aceleren el manejo de dominios complejos en los proyectos de software. Nos permite el problema, de manera que podamos dividirlo en subdominios con el objetivo de maximizar el desacoplamiento entre cada uno y lograr así soluciones modulares, que premian la adaptabilidad de la solución en general [10].

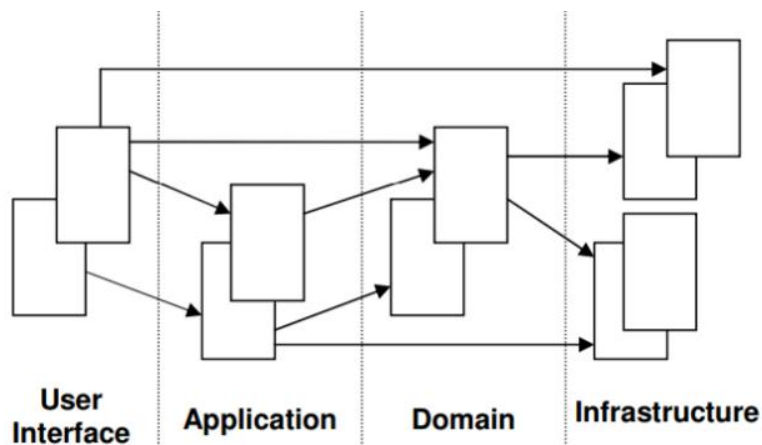


Figura 4.16: Arquitectura propuesta por el diseño basado en el dominio.

Con esta estructura se persigue diseñar un sistema de fácil entendimiento, versátil, y acorde a una arquitectura moderna y dinámica muy utilizada en la actualidad.

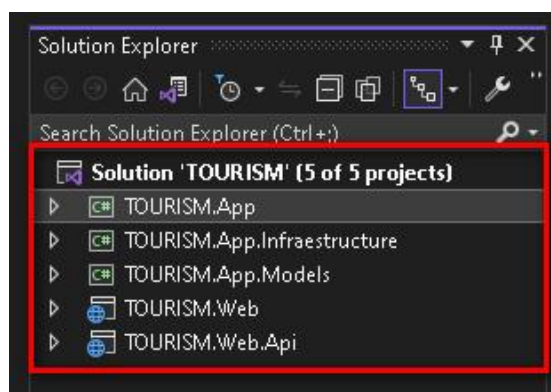


Figura 4.17: Estructura general del proyecto.

#### 4.1.4.1. Aplicación

En esta capa se concentra la lógica central del sistema. Por una parte se compone de los repositorios, este patrón consiste en separar la lógica que recupera los datos y los asigna a un modelo de entidad de la lógica de negocios que actúa sobre el modelo, esto permite que la lógica de negocios sea independiente del tipo de dato que comprende la capa de origen de datos.

Otro componente de esta capa de aplicación, es la gestión de servicios, encargado de hacer los procesos de conversión de la información recibida desde el repositorio a un tipo de dato más simplificado, que pueda retornarse posteriormente al cliente por medio de la capa de API. Adicionalmente los servicios agregan una capa más manipulación lógica, contemplando cualquier proceso adicional a realizar sobre los datos, así como el proceso de mapeo, aquí es donde toma la ontología que se leyó en el repositorio, y se convierte a un modelo donde se contemplan todas las relaciones existentes entre las entidades. La potencia de este proyecto está en la carga y procesamiento de la ontología, lo que ocurre en varias partes del sistema:

- Primero se carga y lee el archivo .owl dentro de la aplicación, cuando se realiza la lectura ese contenido es representado en un modelo previamente creado, el cual es la representación por medio de clases de la ontología.
- Lo siguiente es procesar el modelo, y convertirlo a un tipo de datos que la aplicación pueda retornar al cliente web; esto se hace en un método que procesa los datos en un ciclo que se ejecuta hasta que haya recorrido cada nodo del mismo. Este método hace uso de recursividad para ir colocando debajo de cada clase las entidades relacionadas e inferidas.
- El último paso es retornar el nuevo modelo generado a la interfaz gráfica, donde podrá ser visualizado por el usuario.

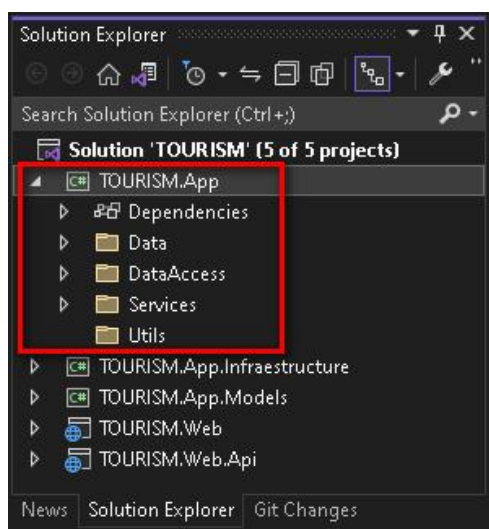


Figura 4.18: Capa de aplicación.

#### 4.1.4.2. Infraestructura

Esta es una capa de uso común dentro del sistema, la misma se compone de todas aquellas clases de uso general o que facilitan la realización de una tarea en cualquier otro módulo.

Se compone de las extensiones, que son clases que agregan funcionalidades adicionales a las clases principales, también simplifican la forma en que se hace uso de las mismas.

Otro de sus componentes son los Mappers, que funcionan como un puente entre las entidades de origen del modelo, y el tipo de dato que se retorna; de forma sencilla, los Mappers toman unos valores de entrada tipados de una forma, y los retornan convertidos en otro tipo de entidad.

Por último están los utils o utilitarios, que son las clases de ayuda más generales que posee la aplicación, y que pueden ser utilizadas por cualquier entidad que las necesite. Proveen de funcionalidades para el manejo de cadena de caracteres, así como validadores sobre los datos manejados, de igual forma contempla lo relativo al manejo de varios idiomas en el sitio web.

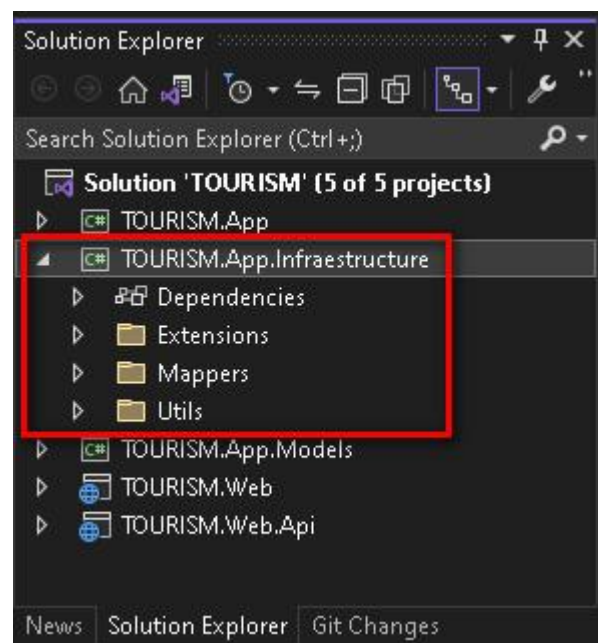


Figura 4.19: Capa de infraestructura.

#### 4.1.4.3. Modelos

La función principal de esta capa es contener las entidades o clases sobre las que se sostiene la lógica del proyecto. Aquí es donde se concentra gran parte de la estructura general del proyecto, porque representa el puente entre los datos externos y la capa de aplicación que se encarga de procesarlos.

Entre los elementos que contiene esta capa están:

- Las entidades representativas de la ontología que funge como base de datos.
- Declaración de enumeradores útiles para el manejo de tipos genéricos o respuestas de la aplicación.
- Los modelos de validación, que aseguran que los modelos cumplan con los controles establecidos.
- Los modelos de vista, son las clases o tipo de datos que se retornan desde el sistema de aplicación hacia el cliente o proyecto web.

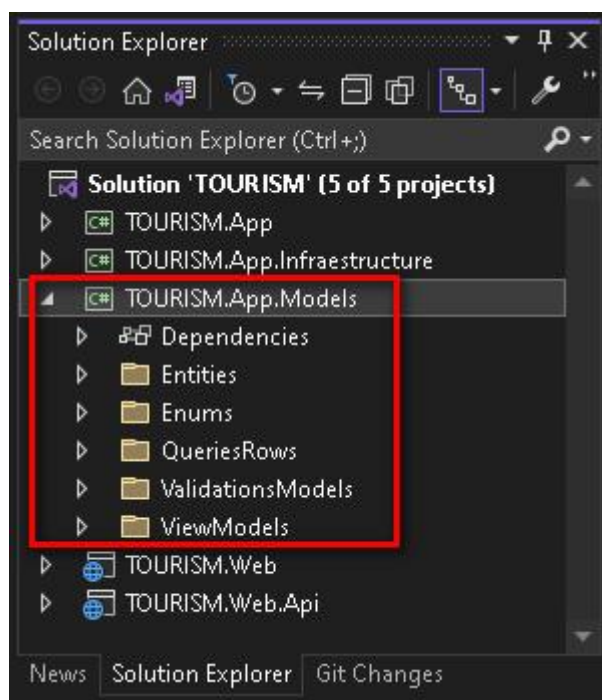


Figura 4.20: Capa de modelos.

#### 4.1.4.4. API

API significa Application Programming Interface, es interfaz de programación de aplicaciones por donde se exponen los datos procesados al cliente web que los requiere. Esta aplicación consume los datos que se han procesado en la capa de aplicación, y posteriormente los expone a través de un servicio REST, que se ha explicado en apartados anteriores. Contiene:

- Los controladores que establecen comunicación con el servicio que retorna las clases y relaciones de la ontología.
- Filtros que validan la comunicación entre el cliente y este servicio, promoviendo en todo momento una conexión estable y segura entre ambos.
- Utilidades, que aportan servicios para manejar la seguridad y los patrones de comunicación.

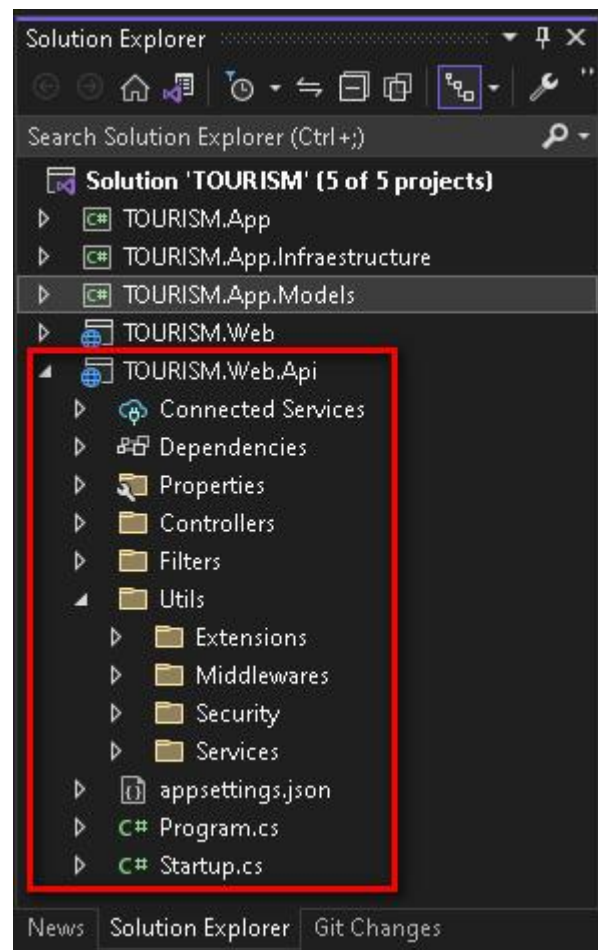


Figura 4.21: Capa de API (Application Programming Interface).



#### 4.1.4.5. Web

Es la capa que tiene exposición directa al usuario, en ella están creadas las interfaces que podemos ver a través de la web. Se basa en el uso del patrón MVC que se ha mencionado previamente, y es aquí donde se consumen los métodos que provee el servicio de aplicación que se ha tratado en el apartado anterior. Esta capa se compone principalmente de:

- **Controladores**, que se encargan de responder a las acciones que se solicitan en la aplicación, como visualizar un elemento, buscar información, aplicar un filtrado de datos. Es el proceso intermedio entre los modelos que poseen la información, y las vistas que muestran dicha información.
- **Recursos**: posee un archivo para los diferentes idiomas en que se puede visualizar la página.
- **wwwroot**: es la carpeta que contiene los recursos externos, tales como imágenes, archivos de estilos, archivos de javascript, fuentes, acceso a las librerías externas utilizadas, como jQuery y Vue.js.
- **Utilidades**: posee clases de ayuda generales en la aplicación web.
- **Vistas**: contienen el código de la aplicación que va a producir la visualización de las interfaces de usuario. Aquí es donde se muestran los datos contenidos en la ontología, las relaciones entre estos, las opciones de búsqueda y filtrado, así como el acceso directo a cada entidad.

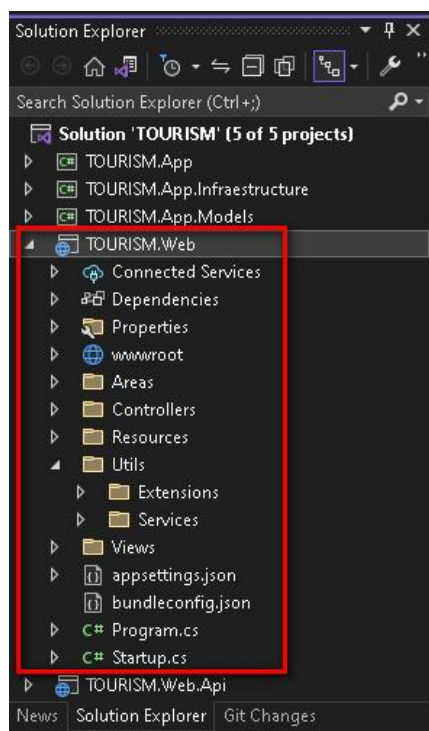


Figura 4.22: Capa web.

## 4.2. Organización funcional del proyecto

En esta sección se detalla la representación visual del proyecto, donde el usuario puede interactuar con el conocimiento extraído e inferido en la ontología.

Para acercar este proyecto los objetivos de la web semántica se hace uso de propiedades en las etiquetas Html que presentan el contenido extraído de la ontología, aportando mayor significado a cada uno de esos elementos ante los diferentes buscadores que se encargan de filtrar y renderizar. La estructura de esas etiquetas es como sigue: `property="v:url"`.

Como se puede observar, la entidad principal de la ontología son los destinos, que a su vez se relacionan con las demás clases; por esta razón la estructura actual a nivel gráfico.

- **Página principal de la aplicación:** ofrece una visión general hacia la forma de la página; en la parte superior posee un menú de navegación hacia las diferentes partes, luego los destinos, con un comentario que poseen la ontología, un enlace para el detalle de individuos y propiedades relacionadas, además de una imagen manejada en la aplicación. Los destinos son la entidad central de la ontología.

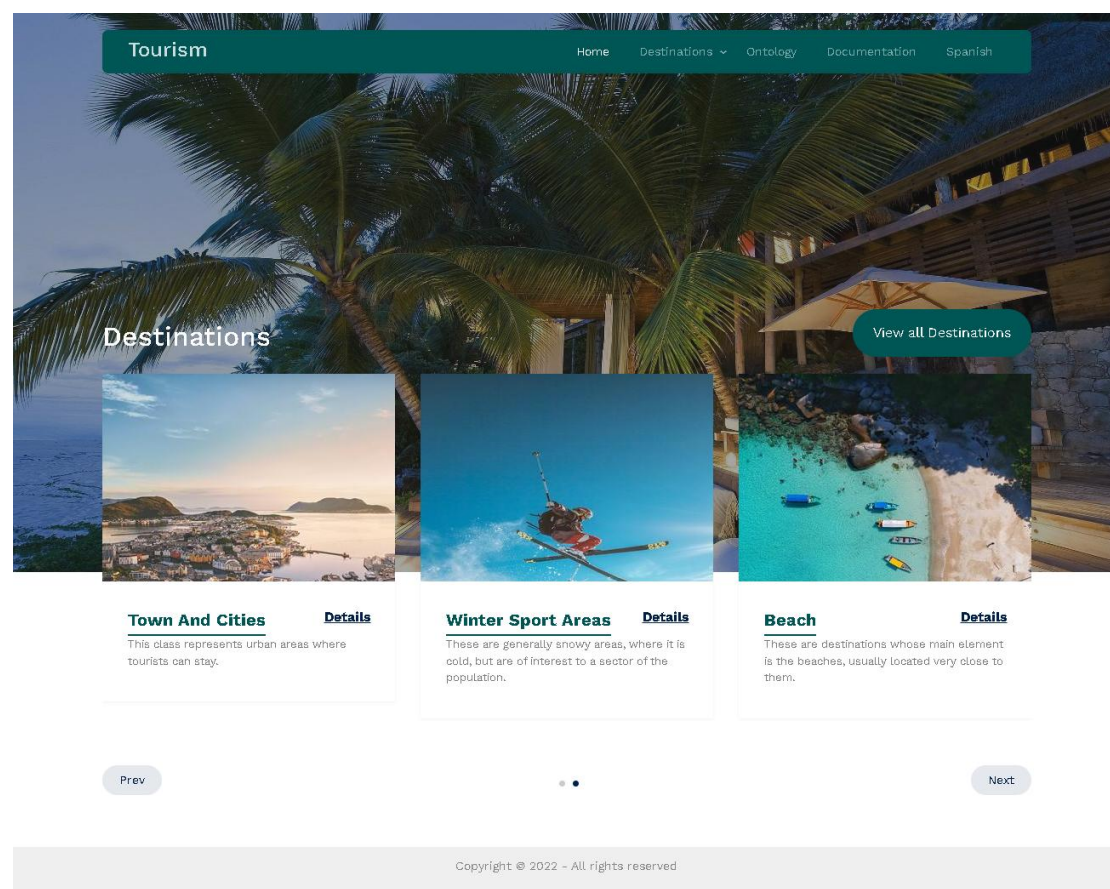


Figura 4.23: Página de inicio del sitio web.

- **Listado de destinos:** al igual que la página principal, muestra el listado de destinos, pero todos visibles al usuario. También nos permite navegar hacia los elementos particulares de cada destino.

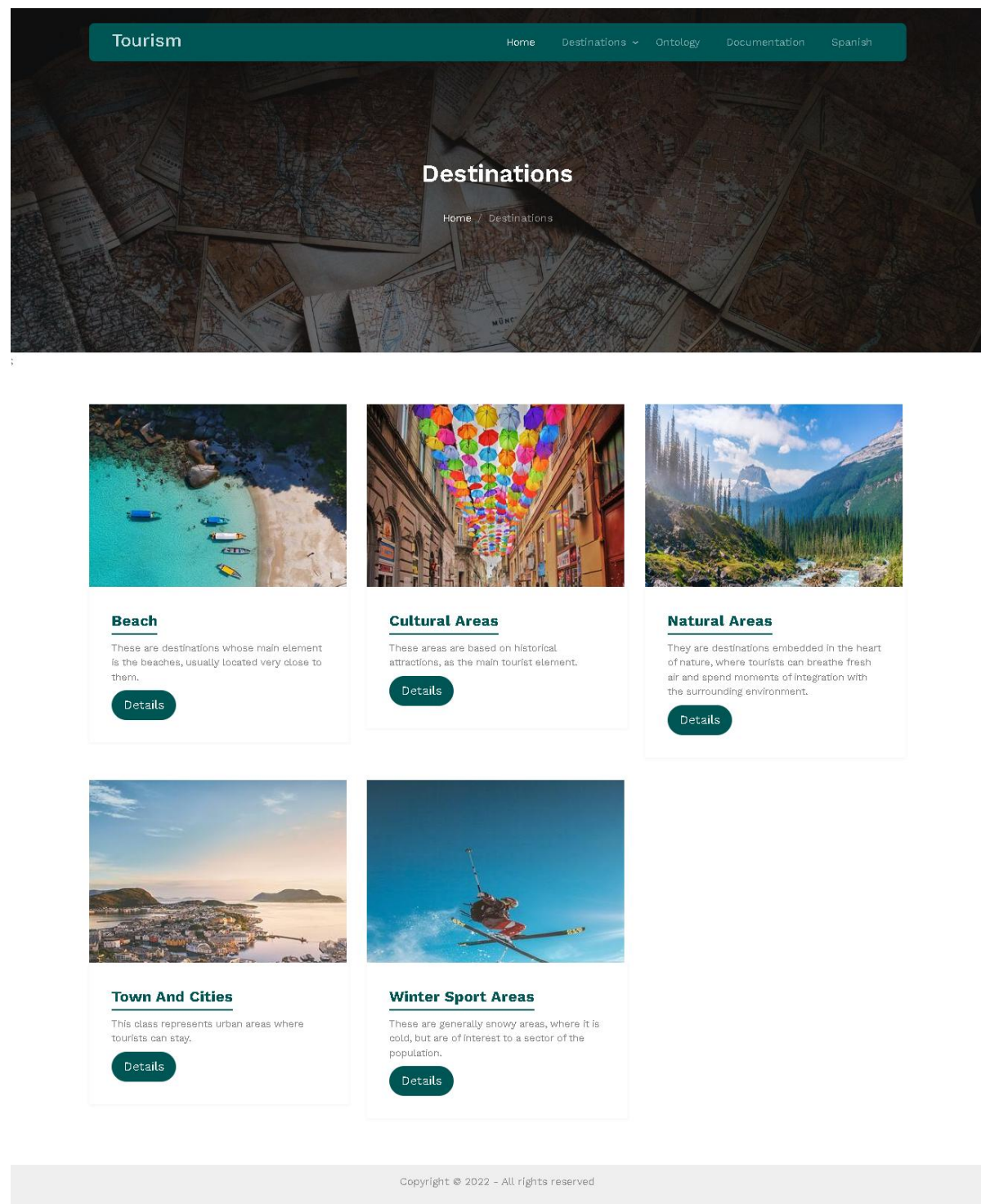


Figura 4.24: Página con el listado de destinos.

- **Detalle de destino:** esta es la vista que integra más conocimiento del extraído de la ontología. Aquí confluyen las clases relacionadas en un nivel por debajo de las clases principales. De forma general se puede apreciar la clase con sus individuos, y las propiedades de datos y de objetos asociadas. Adicional a lo mencionado anteriormente, se presenta la relación por la que se maneja el individuo o individuos, en dicha relación se integra el dominio, la propiedad, el rango, y el tipo de propiedad.

Tal y como se puede observar en la siguiente figura, esta página cuenta con:

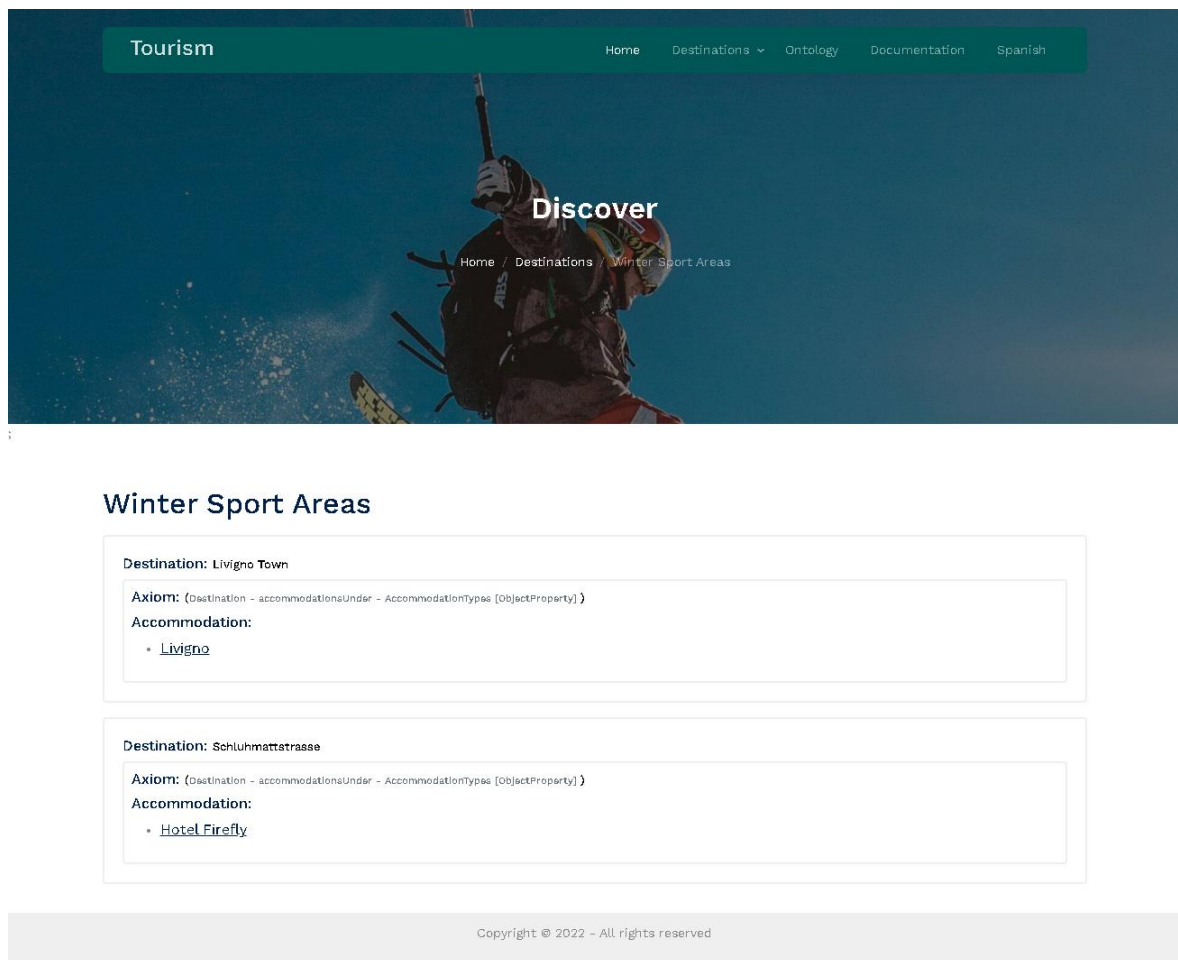
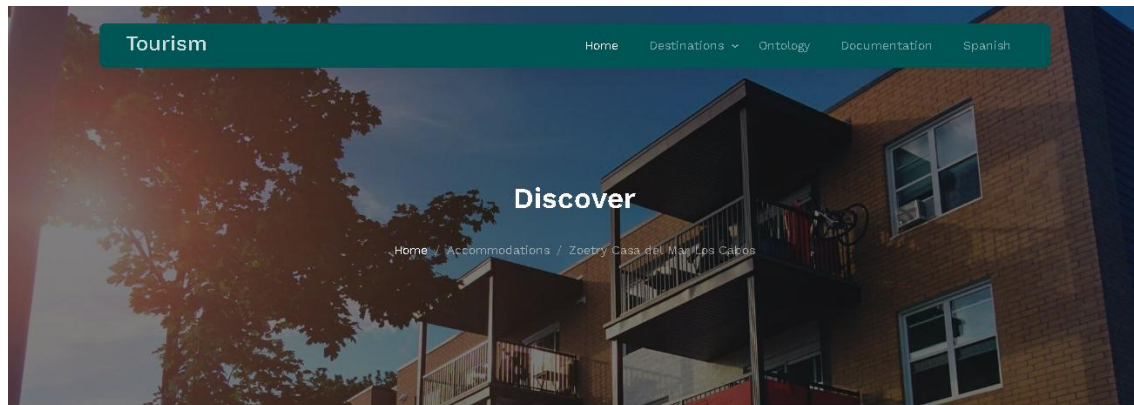


Figura 4.25: Página con el detalle de un destino.



- **Detalle de individuo de alojamiento:** esta pantalla se adentra hasta el nivel mínimo de conocimiento manejado en la ontología. Se muestra un individuo junto con sus propiedades de objetos y datos, de una forma detallada y bien estructurada.



## Zoetry Casa del Mar Los Cabos

[Go Back](#)

### Rooms:

- Presidential Suite 4
- Suite 3
- Standard 16
- Studio 7
- Triple 9
- Twin 11

• **Location:** Los Cabos

• **Brand:** Zoetry

• **Code:** ZCM

• **Country:** Mexico

• **Only Adults:** true

• **Visitors:** 20000

• **Stars:** 5

• **Continent:** America

• **Awards:** Gold Crown

### Offer Activity:

- Gala Parties
- Parties and Events
- Scuba Diving

### Facilities:

- Chinese Cuisine 3
- Beach Bar 2
- Fitness center 7
- Mini Bar 1
- Other Cuisine 10
- Pool Bar 1
- Spa 3

### Additional Relations

- **General:** AdultsOnlyAccommodations
- **Meal Plan:** AllInclusive, UnlimitedExperience
- **Continent:** America
- **Accommodation Type:** BoutiqueHotel, Resorts
- **Rating:** Luxury

Figura 4.26: Vista detallada de un individuo.

- **Página con la ontología:** muestra imágenes de las clases y subclases de la ontología de forma estructurada, presentando la ramificación que posee cada una de ellas.

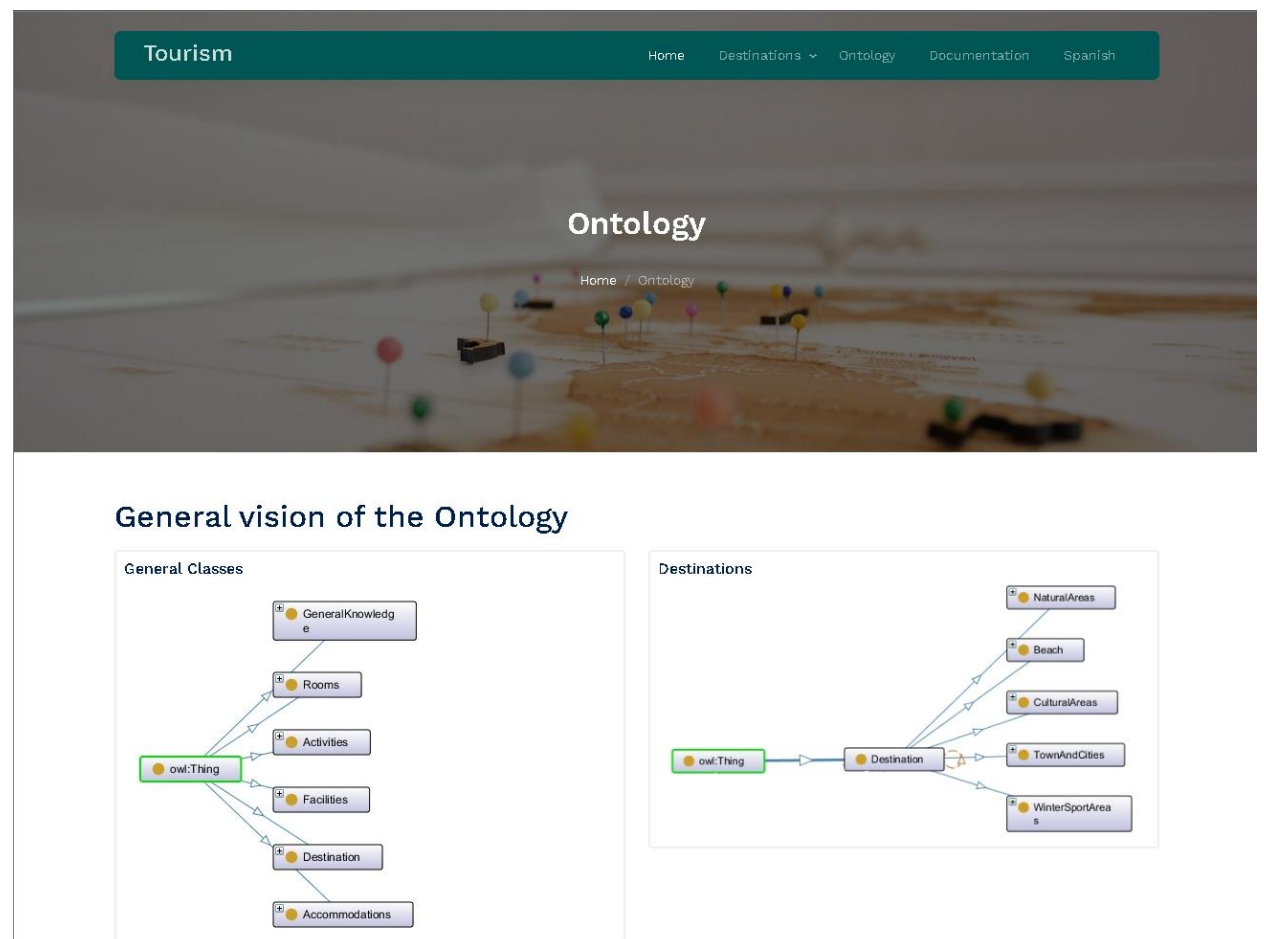


Figura 4.27: Diseño de la página web de ontología.

## **5. Análisis de implementación**

En esta sección se realiza un análisis general sobre el código más relevante en el lenguaje de programación seleccionado, con el objetivo de mostrar de cerca parte de la implementación, la calidad del código, y las posibles mejoras que se pueden realizar al mismo.

La fase de análisis y evaluación de la ontología desarrollada sirve para establecer si la misma cumple con los requisitos establecidos en la fase de planificación.

### **5.1. Código y análisis final de la implementación realizada**

La presentación de esta ontología en la web se ha basado en todo un sistema con los diferentes niveles y capas que conlleva. En este apartado se muestra el código principal en que se basa la extracción del conocimiento. La implementación se ha llevado a cabo utilizando el patrón repositorio, que promueve la definición de los métodos en una interfaz, y su posterior implementación en las clases que lo requieran.

Todo el proceso de interacción con la ontología ha sido por medio de la librería dotNetRDF, mencionada en un apartado anterior; tal y como se detallará en cada método que la utiliza, esta librería se asienta sobre la ontología y la convierte en nativa dentro de la aplicación, permitiendo que se pueda dar un tratamiento tal y como se hace en Protégé.

De manera general se ha realizado una implementación significativa y bien estructurada, por medio de la que se podría tomar otra ontología, y haciendo cambios mínimos, se puede extraer todo su conocimiento en la interfaz de aplicación del sistema.

Tal y como se señalará más adelante, el código final está sujeto a optimizaciones, que permitan que las consultas sean más eficientes, y por ende, el tiempo de interacción desde la página web sea mejor; sin embargo, esto no significa que el resultado obtenido en esta implementación no cubra e forma general los objetivos propuestos y los resultados esperados.

El desarrollo de cada método ha sido resultado de un arduo proceso de pruebas unitarias y de integración, buscando que el sistema sea lo menos susceptible a errores, y que en caso de ocurrir alguno, sea manejado de la forma adecuada sin causarle problemas al usuario durante su interacción con la página.

Por otro lado, se han respetado los patrones de diseño que rigen la programación, de igual forma se ha hecho un sistema funcional y que cumple los estándares actuales en cuestión de arquitectura informática.

### 5.1.1. Modelo

En esta capa de la aplicación se han creado las entidades o clases que administran los datos que se procesan en las fases detalladas más adelante.

Aquí las entidades creadas, y la función de cada una.

**IndividualDTO:** funciona como puente para tomar la información en un método y luego ser utilizada a modo de consulta en otro. Solo funciona dentro del repositorio, específicamente en el método **ProcessIndividualProperties**, que busca los individuos por cada clase general.

```
public class IndividualDTO
{
    public string IndividualName { get; set; }
    public string Class { get; set; }
    public string Parent { get; set; }
    public string Comment { get; set; }
    public string Image { get; set; }
}
```

**PropertyDTO:** Se utiliza en el mismo método de la clase anterior, contiene las propiedades de datos y objetos de una clase particular, así como el dominio, rango, y cualquier información relevante para hacer la representación del conocimiento.

```
public class PropertyDTO
{
    public string Class { get; set; }
    public string Parent { get; set; }
    public string Comment { get; set; }
    public string Image { get; set; }
    public List<IndividualAxiomDTO> Axioms { get; set; }
}
```

**IndividualPropertiesDTO:** Contiene los individuos consultados de una clase particular, junto al detalle de propiedades asociadas al mismo; en este mismo proceso se agrega el rango, dominio y tipo de propiedad a través de **IndividualPropertiesDTO**.

```
public class IndividualPropertiesDTO
{
    public string IndividualName { get; set; }
    public List<PropertyDTO> Properties { get; set; }
}
```



**IndividualAxiomDTO:** es la clase que maneja el dominio, rango y tipo de propiedad que están asociados a un individuo por medio de su clase padre. De igual forma, administra una colección de valores que se asocian al individuo.

```
public class IndividualAxiomDTO
{
    public string Domain { get; set; }
    public string Property { get; set; }
    public string PropertyType { get; set; }
    public string Range { get; set; }
    public List<AxiomValueDTO> Value { get; set; }
}
```

**AxiomValueDTO:** su única función es manejar los valores asociados a un individuo por medio de las propiedades de datos y objetos relacionadas.

```
public class AxiomValueDTO
{
    public string Value { get; set; }
}
```

**ValueDTO:** lleva los datos como el rango y los valores de una propiedad desde un controlador a una vista, en el cliente.

```
public class ValueDTO
{
    public string Range { get; set; }
    public List<AxiomValueDTO> Values { get; set; }
}
```

**AxiomQueryDTO:** sirve de entidad común para la estructuración de las diferentes consultas que maneja cada método.

```
public class AxiomQueryDTO
{
    public string Property { get; set; }
    public string PropertyType { get; set; }
    public string Individual { get; set; }
    public string QueryItems { get; set; }
    public string Condition { get; set; }
    public string Domain { get; set; }
}
```

**OntologyDTO:** es la clase principal del proyecto; posee todo el conocimiento extraído de la ontología, y manejado por el sistema. Por medio de esta clase se hace posible estructurar la información de: clases, subclases, propiedades, axiomas de dichas propiedades, individuos, y la información que estos poseen.

En primer lugar todos los datos consultados a través de los diferentes procesos de la aplicación que maneja la lógica, son convertidos a esta entidad, que posteriormente es retornada al cliente, o aplicación que maneja las vistas con que interactúa el usuario final.

```
public class OntologyDTO
{
    public string Class { get; set; }
    public string Parent { get; set; }
    public string Comment { get; set; }
    public List<OntologyDTO> Children { get; set; }
    public List<IndividualPropertiesDTO> Individuals { get; set; }
}
```

### 5.1.2. Repositorios

Representan el principal nivel de abstracción en la implementación del proyecto. Aquí es donde se interactúa de forma directa con el archivo de la ontología, y donde se definen los métodos y funcionalidades para el filtrado de los datos, y su posterior exposición al nivel de servicios.

A continuación se muestran los métodos creados, y la explicación sobre las funciones que realizan.

#### Interfaz con la definición de los métodos:

El objetivo principal de una interfaz en el entorno de programación es crear una definición de los métodos que posteriormente implementará una clase.

En este particular, se definen los diferentes métodos que se implementan en el repositorio de contenido de la aplicación. Cada uno de esos métodos será abordado en detalle más adelante.

```
public interface IContentRepository
{
    List<OntologyDTO> GetOntologySuperClasses();
    List<OntologyDTO> GetOntologySubClasses();
    List<OntologyDTO> GetAnyOntologyElement(string parentClass);
    List<OntologyDTO> GetOntologySubClassesByParent(string parentClass);
    List<IndividualPropertiesDTO> GetOntologyIndividuals(string parentClass, List<string> superClasses,
string additionalFilter = "");
    List<IndividualPropertiesDTO> GetOntologyIndividuals(string parentClass, List<string> superClasses);
}
```

## Implementación de los métodos de la interfaz:

Lo que se observa en el bloque a continuación es la declaración de variables y objetos a utilizar en los métodos, y su posterior inicialización en el constructor.

```
public class ContentRepository : IContentRepository
{
    private readonly RdfXmlParser rdfParser;
    private readonly Graph graph;
    private readonly LeviathanQueryProcessor processor;
    private readonly SparqlQueryParser sparqlParser;
    private readonly ISparqlDataset sparqlDataset;
    private readonly WebApiAppSettings _settings;
    private readonly SiteInfo SiteInfo = default(SiteInfo);
    private readonly string prefixes = string.Empty;
    private readonly string individualPropertyType1 = string.Empty;
    private readonly string individualPropertyType2 = string.Empty;
    private readonly string commentReplacePattern = string.Empty;
    private readonly string individualClassValidation = string.Empty;
    private readonly List<string> individualReplacePattern = new List<string>();

    public ContentRepository(IOptions<WebApiAppSettings> settings)
    {
        rdfParser = new RdfXmlParser(RdfXmlParserMode.DOM);
        graph = new Graph();
        sparqlDataset = new InMemoryDataset(graph);
        processor = new LeviathanQueryProcessor(sparqlDataset);
        sparqlParser = new SparqlQueryParser();
    }
}
```

```

        _settings = settings.Value;
        SiteInfo = _settings.SiteInfos.FirstOrDefault();
        rdfParser.Load(graph, SiteInfo.OntologyDetails.OntologyFile);
        prefixes = string.Join(' ', SiteInfo.OntologyDetails.QueryPrefixes);
        individualReplacePattern = SiteInfo.OntologyDetails.IndividualReplacePattern;
        commentReplacePattern = SiteInfo.OntologyDetails.CommentReplacePattern;
        individualClassValidation = SiteInfo.OntologyDetails.IndividualClassValidation;
        individualPropertyType1 = SiteInfo.OntologyDetails.IndividualPropertyTypes.FirstOrDefault();
        individualPropertyType2 = SiteInfo.OntologyDetails.IndividualPropertyTypes.Skip(1).FirstOrDefault();
    }

```

**GetOntologySuperClasses:** este método obtiene solo las clases de primer nivel de la aplicación. El interés detrás de esto es obtener una construcción de cada clase base, que posteriormente podrá ser parte de un proceso de iteración, para rellenar cada subclase, propiedades e individuos que forman parte de ella.

La estructura general de la mayoría de los métodos es la siguiente: primero se construye una consulta, agregando posibles parámetros adicionales que vienen desde instancias externas, luego el mismo es ejecutado a través de **dotNetRDF**, y finalmente los resultados obtenidos son convertidos a alguna de las clases locales declaradas previamente.

```

#region Process Ontology
public List<OntologyDTO> GetOntologySuperClasses()
{
    var output = Enumerable.Empty<OntologyDTO>().ToList();
    var queryString = @"SELECT DISTINCT ?class ?comment
        WHERE
        {
            {
                ?class rdf:type owl:Class .
                ?subClass rdf:type owl:Class .
            }
        }
    ";
}

```

```

        ?subClass rdfs:subClassOf ?class
        FILTER NOT EXISTS
        {
            ?class rdfs:subClassOf ?otherSup
            FILTER (?otherSup != owl:Thing)
        }
    }
    UNION
    {
        ?class rdf:type owl:Class
        FILTER NOT EXISTS { ?subClass rdfs:subClassOf ?class }
        FILTER NOT EXISTS { ?class rdfs:subClassOf ?supClass }
    }
    OPTIONAL { ?class rdfs:comment ?comment} .
}";

```

```

var query = sparqlParser.ParseFromString(string.Concat(prefixes, " ", queryString));
var results = QueryResults(query);
if (results == null || !results.Any())
    return output;
foreach (SparqlResult item in results)
{
    output.Add(new OntologyDTO()
    {
        Class = item["class"]?.ToString(),
        Comment = item["comment"]?.ToString(),
        Parent = "owl:Thing",
        Children = Enumerable.Empty<OntologyDTO>().ToList(),
        Individuals = Enumerable.Empty<IndividualPropertiesDTO>().ToList()
    });
}

```

```

    });
}
return output;
}

```

**GetOntologySubClasses:** este método retorna todas las subclases existentes dentro de la ontología, que en un próximo método es utilizado para agregar a cada clase principal las subclases que componen su árbol. La estructura del método es como se ha citado previamente, agregando al final algunas validaciones sobre los datos obtenidos.

```

public List<OntologyDTO> GetOntologySubClasses()
{
    var output = Enumerable.Empty<OntologyDTO>().ToList();
    var filterClasses = SiteInfo.OntologyDetails.OntologySubClassesPrefix;

    var queryString = @"SELECT DISTINCT ?class ?parent ?label ?comment
        WHERE {
            { ?class a owl:Class . } UNION { ?individual a ?class . } .
            OPTIONAL { ?class rdfs:subClassOf ?parent } .
            OPTIONAL { ?class rdfs:comment ?comment} .
        } ORDER BY ?class";

    var query = sparqlParser.ParseFromString(string.Concat(prefixes, " ", queryString));
    var results = QueryResults(query);

    if (results != null && results.Any())
    {
        output = ParseOntologyClassesResults(results);
    }
}

```

```

        if (output != null && output.Any())
            output=output.Where(row=> !string.IsNullOrEmpty(row.Parent)&& !row.Class.Contains(filterClasses))
                .ToList();
    }
    return output;
}

```

**GetOntologySubClassesByParent:** Como su nombre lo indica, con este método se busca obtener las subclases directas de una clase particular, que es pasada como parámetro.

```

public List<OntologyDTO> GetOntologySubClassesByParent(string parentClass)
{
    var output = Enumerable.Empty<OntologyDTO>().ToList();
    var filterClasses = SiteInfo.OntologyDetails.OntologySubClassesPrefix;

    var queryString = @"SELECT ?class ?parent ?comment
        WHERE {
            ?class rdfs:subClassOf * <MainClass> .
            OPTIONAL { ?class rdfs:subClassOf ?parent } .
            OPTIONAL { ?class rdfs:comment ?comment} .
        } ORDER BY ?class".Replace("MainClass", parentClass);

    var query = sparqlParser.ParseFromString(string.Concat(prefixes, " ", queryString));
    var results = QueryResults(query);

    if (results != null && results.Any())
    {
        output = ParseOntologyClassesResults(results);
    }
}

```



```

        if (output != null && output.Any())
            output = output.Where(row => !string.IsNullOrEmpty(row.Parent)
                && !row.Class.Contains(filterClasses)).ToList();

        if (SiteInfo.OntologyDetails.RootEntity == parentClass)
            output = output.Where(row => row.Parent == parentClass && row.Class != parentClass).ToList();
    }
    return output;
}

```

**GetOntologyIndividuals:** Obtiene todos los individuos por cada clase de primer nivel, con la posibilidad de filtrar por una clase cualquiera.

```

public List<IndividualPropertiesDTO> GetOntologyIndividuals(string parentClass, List<string> superClasses,
string additionalFilter="")
{
    var output = Enumerable.Empty<IndividualPropertiesDTO>().ToList();

    var queryString = @"SELECT DISTINCT ?class ?parent ?individual ?image ?comment
        WHERE {
            { ?class a owl:Class . } UNION { ?individual a ?class . } .
            ?class rdfs:subClassOf* <MainClass> .
            OPTIONAL { ?class rdfs:subClassOf ?parent } .
        } ORDER BY ?class".Replace("MainClass", parentClass);

    var query = sparqlParser.ParseFromString(string.Concat(prefixes, " ", queryString));
    var results = QueryResults(query);
}

```

```

        if (results == null || !results.Any())
            return output;

        output = ProcessIndividualProperties(results, superClasses);

        return output;
    }
#endregion

```

**GetIndividualAxiom:** obtiene las propiedades que se asocian a los datos contenidos por un individuo en particular. De forma principal recibe el individuo y la clase superior a este, para hacer un filtrado más exacto de lo que se busca. En una primera instancia se buscan los tipos de propiedades asociadas al individuo, para hacer un filtrado sobre las que son de interés (propiedades de objeto y propiedades de datos), ya que las demás solo generan datos repetidos. Luego de realizada la consulta y el proceso de conversión de los datos, se llama **FillAxiomValue**, que retorna los valores asociados al individuo por medio de cada propiedad de objeto y de dato.

```

#region Helpers
private List<IndividualAxiomDTO> GetIndividualAxiom(string individualClass, string individual)
{
    var individualAxioms = Enumerable.Empty<IndividualAxiomDTO>().ToList();
    var replacePattern = individualReplacePattern;
    var propertyType1 = individualPropertyType1;
    var propertyType2 = individualPropertyType2;

    var queryString = @"SELECT DISTINCT ?domain ?parent ?propertyType ?range
        WHERE {
            <IndividualClass> rdfs:subClassOf* ?parent.
            ?domain rdfs:domain ?parent.
            ?domain rdf:type ?propertyType.

```

```

        ?domain rdfs:range ?range
        FILTER ((?propertyType = <PropertyType1>)
            || (?propertyType = <PropertyType2>))
    }".Replace("IndividualClass", individualClass)
        .Replace("PropertyType1", propertyType1)
        .Replace("PropertyType2", propertyType2);

var query = sparqlParser.ParseFromString(string.Concat(prefixes, " ", queryString));
var results = QueryResults(query);

if (results != null && results.Any())
{
    individualAxioms = ParseIndividualAxiom(results);

    foreach (var item in individualAxioms)
    {
        var range = item.Range;
        var domain = item.Domain;
        var ontolProperty = item.Property;

        foreach (var rep in replacePattern)
        {
            var index = replacePattern.IndexOf(rep) + 1;
            range = range.Replace(rep, "");
            domain = domain.Replace(rep, "");

            if (index <= 2)
                ontolProperty = ontolProperty.Replace(rep, "");
        }
    }
}

```

```
        range = range.ToLower();
        domain = domain.ToLower();

        var axiomValuequery = new AxiomQueryDTO()
        {
            Condition = $" ?{domain} ontol:{ontolProperty} ?{range}.",
            QueyItems = range,
            Property = item.Property,
            PropertyType = item.PropertyType,
            Individual = individual,
            Domain = domain
        };

        var axiomValues = FillAxiomValue(axiomValuequery);

        if (axiomValues != null && axiomValues.Any())
            item.Value.AddRange(axiomValues);
    }

    individualAxioms = individualAxioms.Where(row => row.Value != null && row.Value.Any()).ToList();
}
return individualAxioms;
}
```

**FillAttributeValue:** recibe un modelo que tiene una condición de búsqueda, unos campos items, y un filtro de búsqueda. Se hace de esta forma porque cada individuo tiene asociados diferentes propiedades de datos y objetos, y no resultaría eficiente crear un método por cada tipo de propiedad.

Su intención principal es consultar y retornar los valores que están asociados a cada individuo por medio de las propiedades y restricciones.

```
Private List<AxiomValueDTO> FillAxiomValue(AxiomQueryDTO queryItems)
{
    var output = Enumerable.Empty<AxiomValueDTO>().ToList();

    var queryString = @"SELECT ?QueryItems
    WHERE {
        Condition
        FILTER ( ?Domain = <Individual>)
    }".Replace("QueryItems", queryItems.QueryItems.Trim()).Replace("Condition", queryItems.Condition.Trim())
        .Replace("Individual", queryItems.Individual.Trim()).Replace("Domain", queryItems.Domain.Trim());

    var query = sparqlParser.ParseFromString(string.Concat(prefixes, " ", queryString));
    var results = QueryResults(query);
    if (results != null && results.Any())
        output = ParseAxiomValue(results, queryItems.QueryItems);

    return output;
}
```

En lo adelante se presentan los métodos que sirven para procesar unos resultados, y convertirlos del objeto que genera dotNetRDF al objeto creado localmente. Estos métodos son considerados de ayuda, y usados solamente en el repositorio, no tienen exposición fuera de estos.

**ParseOntologyClassesResults:** convierte los datos resultantes del método **GetOntologySubClasses**, al tipo de clase que se maneja de forma local.

```
private List<OntologyDTO> ParseOntologyClassesResults(List<SparqlResult> results)
{
    var output = Enumerable.Empty<OntologyDTO>().ToList();
    var replaceComment = commentReplacePattern;

    if (results == null || !results.Any())
        return output;

    foreach (SparqlResult item in results)
    {
        output.Add(new OntologyDTO()
        {
            Class = GetFieldValue(item, "class"),
            Parent = GetFieldValue(item, "parent"),
            Comment = GetFieldValue(item, "comment").Replace(replaceComment, ""),
            Children = Enumerable.Empty<OntologyDTO>().ToList(),
            Individuals = Enumerable.Empty<IndividualPropertiesDTO>().ToList()
        });
    }
    return output;
}
```

**ParseIndividuals:** recibe los individuos resultado del método **ProcessIndividualProperties**, y los convierte al nuevo tipo de datos.

```
private List<IndividualDTO> ParseIndividuals(List<SparqlResult> results)
{
    var output = Enumerable.Empty<IndividualDTO>().ToList();

    if (results == null || !results.Any())
        return output;

    foreach (SparqlResult item in results)
    {
        var className = GetFieldValue(item, "class");
        if (className.Contains(individualClassValidation))
            continue;

        output.Add(new IndividualDTO()
        {
            Class = className,
            Comment = GetFieldValue(item, "comment"),
            Image = GetFieldValue(item, "image"),
            IndividualName = GetFieldValue(item, "individual"),
            Parent = GetFieldValue(item, "parent"),
        });
    }
    return output;
}
```

**ProcessIndividualProperties:** gestiona y procesa los resultados del método **GetOntologyIndividuals**. De igual forma procesa los resultados obtenidos para los valores asociados al individuo.

```
private List<IndividualPropertiesDTO> ProcessIndividualProperties(List<SparqlResult> results,
List<string> superclasses)
{
    var output = Enumerable.Empty<IndividualPropertiesDTO>().ToList();

    var allIndividuals = ParseIndividuals(results);
    var differentIndividuals = allIndividuals.Where(row => !string.IsNullOrEmpty(row.IndividualName))
        .GroupBy(row => row.IndividualName)
        .ToList();

    foreach (var ind in differentIndividuals)
    {
        var items = Enumerable.Empty<PropertyDTO>().ToList();
        foreach (var item in ind.ToList())
        {
            items.Add(new PropertyDTO()
            {
                Class = item.Class,
                Comment = item.Comment,
                Image = item.Image,
                Parent = item.Parent
            });
        }
    }
}
```



```
        output.Add(new IndividualPropertiesDTO()
        {
            IndividualName = ind.Key,
            Properties = items
        });
    }

    if (output == null || !output.Any())
        return output;

    foreach (var ind in output)
    {
        foreach (var item in ind.Properties)
        {
            var axioms = GetIndividualAxiom(item.Class, ind.IndividualName);

            if (axioms != null && axioms.Any())
                axioms = axioms.ToList();

            item.Axioms = axioms;
        }
    }

    foreach (var item in output)
    {
        var axioms = ((item.Properties.SelectMany(row => row.Axioms)).GroupBy(row => row.Property))
            .Select(row => new IndividualAxiomDTO()
            {
                Domain = row.FirstOrDefault().Domain,
```

```
        Property = row.FirstOrDefault().Property,  
        PropertyType = row.FirstOrDefault().PropertyType,  
        Range = row.FirstOrDefault().Range,  
        Value = row.FirstOrDefault().Value  
    }).ToList();  
  
    item.Properties.ForEach(row => row.Axioms = Enumerable.Empty<IndividualAxiomDTO>().ToList());  
  
    var parentClassProperty = item.Properties.FirstOrDefault(row => superclasses.Contains(row.Parent));  
  
    if (parentClassProperty != null)  
        parentClassProperty.Axioms = axioms;  
    else  
        item.Properties.FirstOrDefault().Axioms = axioms;  
}  
output = output.OrderBy(row => row.IndividualName).ToList();  
  
return output;  
}
```

**ParseIndividualAxiom:** es llamado por el método **GetIndividualAxiom**, para procesar los axiomas relacionados con las propiedades de un individuo en particular.

```
private List<IndividualAxiomDTO> ParseIndividualAxiom(List<SparqlResult> results)
{
    var output = Enumerable.Empty<IndividualAxiomDTO>().ToList();

    if (results == null || !results.Any())
        return output;

    foreach (SparqlResult item in results)
    {
        output.Add(new IndividualAxiomDTO()
        {
            Domain = GetFieldValue(item, "parent"),
            Property = GetFieldValue(item, "domain"),
            PropertyType = GetFieldValue(item, "propertyType"),
            Value = Enumerable.Empty<AxiomValueDTO>().ToList(),
            Range = GetFieldValue(item, "range"),
        });
    }

    return output;
}
```

**ParseAxiomValue:** es utilizado por el método **FillAxiomValue** para procesar los valores particulares de los individuos consultados.

```
private List<AxiomValueDTO> ParseAxiomValue(List<SparqlResult> results, string queryItems)
{
    var output = Enumerable.Empty<AxiomValueDTO>().ToList();
    if (results == null || !results.Any())
        return output;

    foreach (SparqlResult item in results)
        output.Add(new AxiomValueDTO() { Value = GetFieldValue(item, queryItems) });

    return output;
}
```

**GetFieldValue:** es utilizado por todos los métodos de ayuda para extraer las propiedades consultadas del dataset retornado por la consulta ejecutada a través de dotNetRDF.

```
private string GetFieldValue(SparqlResult result, string field)
{
    var value = string.Empty;
    try
    {
        value = (result[field.Trim()])?.ToString() ?? string.Empty;
    }
    catch (Exception ex) { value = string.Empty;}
    return value;
}
```

**QueryResults:** llamado por todos los métodos donde se crea una consulta, se encarga de ejecutarla y retornar los resultados.

```
private List<SparqlResult> QueryResults(SparqlQuery query)
{
    var processQuery = processor.ProcessQuery(query);

    var parsedResults = (processQuery is SparqlResultSet ? processQuery as SparqlResultSet : null)?
        .Results?.Where(row => row.IsGroundResult);

    var output = parsedResults?.ToList();

    return output;
}
#endregion
```

### 5.1.3. Servicios

Son los encargados de consumir o manipular la información retornada por el repositorio. Funcionan principalmente como un puente entre el repositorio y el controlador que expondrá los datos a través de una interfaz de programación de aplicaciones (API). Siempre que sea necesario manejar lógica de negocio adicional, se realiza en esta parte del proyecto.

A continuación se presentan los métodos creados, y su utilidad en los servicios.

#### Interfaz con definición de los métodos:

Esta interfaz define los métodos que posteriormente son implementados por el servicio (puente entre el repositorio y el controlador que expone los datos hacia la página web).

```
public interface IContentService
{
    List<OntologyDTO> GetBaseContent();
    List<OntologyDTO> GetClassContent(string itemClass);
    List<IndividualPropertiesDTO> GetIndividualContent(string individualName, string individualParent);
    List<OntologyDTO> GetFullOntology();
}
```

## Implementación de los métodos:

Aquí se está haciendo la inicialización del servicio y repositorio que se utilizan posteriormente.

```
public class ContentService : IContentService
{
    private readonly ISettingsService _settingsService;
    private readonly IContentRepository _contentRepository;
    public ContentService(IContentRepository contentRepository, ISettingsService settingsService)
    {
        _contentRepository = contentRepository;
        _settingsService = settingsService;
    }
}
```

**GetBaseContent:** este método busca las subclases de un padre ya definido. En el caso de este proyecto, la clase de **destinos** es la de mayor relevancia, y quien se asocia con las demás entidades de la ontología; por esto es el punto de entrada de la página web.

```
public List<OntologyDTO> GetBaseContent()
{
    var site = _settingsService.SiteInfo();

    var output = _contentRepository.GetOntologySubClassesByParent(site.OntologyDetails.RootEntity);
    return output;
}
```

**GetClassContent:** consulta las propiedades y elementos relacionados de una clase. Es utilizada cuando se ingresa a un destino particular.

Este método retorna un objeto que contiene la estructura completa de la clase consultada, desde los individuos asociados en un nivel superior, hasta los elementos relacionados en niveles más bajos del conjunto de entidades.

```
public List<OntologyDTO> GetClassContent(string itemClass)
{
    var output = Enumerable.Empty<OntologyDTO>().ToList();

    var superClasses = _contentRepository.GetOntologySuperClasses();
    var superClassesList = superClasses.Select(row => row.Class).ToList();

    output = _contentRepository.GetOntologySubClassesByParent(itemClass);

    foreach (var item in output)
    {
        var individuals = _contentRepository.GetOntologyIndividuals(item.Class, superClassesList);
        if (individuals != null && individuals.Any())
            item.Individuals = individuals;
    }
    return output;
}
```



**GetIndividualContent:** retorna los elementos específicos de un individuo consultado; desde la clases a donde pertenece, y clases relacionadas, hasta las propiedades y los valores que toman cada una.

```
public List<IndividualPropertiesDTO> GetIndividualContent(string individualName, string individualParent)
{
    var output = Enumerable.Empty<IndividualPropertiesDTO>().ToList();

    try
    {
        var superClasses = _contentRepository.GetOntologySuperClasses();
        var superClassesList = superClasses.Select(row => row.Class).ToList();

        output = _contentRepository.GetOntologyIndividuals(individualParent, superClassesList,
            additionalFilter:$"FILTER( ?individual=<{{individualName}}>");
    }
    catch (Exception ex)
    {
    }

    return output;
}
```

**GetFullOntology:** la intención de este método es retornar con una sola consulta el árbol completo con la estructura de la ontología. Actualmente no está siendo utilizado porque tarda un tiempo considerable haciendo el procesamiento de todas las clases y entidades.

```
public List<OntologyDTO> GetFullOntology()
{
    var output = Enumerable.Empty<OntologyDTO>().ToList();

    try
    {
        var superClasses = _contentRepository.GetOntologySuperClasses();
        var allSubClasses = _contentRepository.GetOntologySubClasses();

        output = ProcessOntologyElements(superClasses, allSubClasses);
        var superClassesList = superClasses.Select(row => row.Class).ToList();

        foreach (var item in output)
        {
            var individuals = _contentRepository.GetOntologyIndividuals(item.Class, superClassesList);
            if (individuals != null && individuals.Any())
                item.Individuals = individuals;
        }
    }
    catch (Exception ex)
    {
    }
    return output;
}
```

**ProcessOntologyElements:** es una clase de ayuda para agregar en cada subnivel del objeto principal cualquier elemento que esté asociado a el.

```
#region Helper
private static List<OntologyDTO> ProcessOntologyElements(List<OntologyDTO> classes, List<OntologyDTO> subClasses)
{
    foreach (var item in classes)
    {
        item.Children = subClasses.Where(row => row.Parent == item.Class).ToList();
        foreach (var item1 in item.Children)
        {
            item1.Children = subClasses.Where(row => row.Parent == item1.Class).ToList();
            foreach (var item2 in item1.Children)
            {
                item2.Children = subClasses.Where(row => row.Parent == item2.Class).ToList();
                foreach (var item3 in item2.Children)
                {
                    item3.Children = subClasses.Where(row => row.Parent == item3.Class).ToList();
                }
            }
        }
        return classes;
    }
}
#endregion
}
```

#### 5.1.4. Controladores

Son la fase final de la capa de programación lógica, se encargan de exponer la información extraída por el repositorio, y posteriormente procesada por el servicio. Aquí los métodos de controlador creados, y cuál es su función.

Lo primero que se realiza dentro del controlador es inicializar el servicio que se utilizará para consultar la información a retornar.

```
[Route("api/[controller]")]
[ApiController]
[TypeFilter(typeof(PublicSecurityActionFilter), Order = 1)]
[TypeFilter(typeof(PublicPostSecurityActionFilter), Order = 2)]
public class PageContentController : ControllerBase
{
    private readonly IContentService _contentService;
    public PageContentController(IContentService contentService)
    {
        _contentService = contentService;
    }
}
```

De forma general todas estas definiciones de métodos se encargan de conectar con el servicio que maneja los contenidos, para retornar dicha información.

```
[HttpGet]
[Produces(typeof(List<OntologyDTO>))]
public async Task<ActionResult<List<OntologyDTO>>> Get([FromQuery] ClientTCInternationalizationInputDTO input = null)
{
    var content = _contentService.GetBaseContent();
    return Ok(content);
}
```

```
[HttpGet("ClassContent")]
[Produces(typeof(List<OntologyDTO>))]
public async Task<ActionResult<List<OntologyDTO>>> ClassContent([FromQuery] PublicContentInputDTO input)
{
    var content = _contentService.GetClassContent(input.ItemClass);
    return Ok(content);
}
```

```
[HttpGet("IndividualContent")]
[Produces(typeof(List<IndividualPropertiesDTO>))]
public async Task<ActionResult<List<IndividualPropertiesDTO>>> IndividualContent([FromQuery]
PublicContentInputDTO input)
{
    var content = _contentService.GetIndividualContent(input.ItemClass, input.ParentClass);
    return Ok(content);
}
```

```
[HttpGet("FullOntology")]
[Produces(typeof(List<OntologyDTO>))]
public async Task<ActionResult<List<OntologyDTO>>> FullOntology([FromQuery] ClientTCInternationalizationInputDTO
input)
{
    var content = _contentService.GetFullOntology();
    return Ok(content);
}
}
```

### 5.1.5. Cliente

Es el último nivel de la cadena del proyecto, a través de esta aplicación se crean las vistas que presentan al usuario el conocimiento de la ontología, previamente consumido desde la interfaz de aplicación. A continuación se presenta los métodos que maneja.

Esta interfaz crea una representación de los métodos que se llamarán por medio del servicio REST. Cada método definido en esta sección existe dentro del controlador que maneja el contenido.

```
public interface IApiMethods
{
    #region Page Content
    [Get("/api/PageContent?SiteId={siteId}&Lang={lang}&IpAddressClient={ipAddress}&Token={token}")]
    Task<List<OntologyDTO>> GetBaseContent(string siteId, string lang, string ipAddress, string token);

    [Get("/api/PageContent/ClassContent?ItemClass={itemClass}&SiteId={siteId}&Lang={lang}&IpAddressClient={ipAddress}&Token={token}")]
    Task<List<OntologyDTO>> GetClassContent(string itemClass, string siteId, string lang, string ipAddress, string token);

    [Get("/api/PageContent/IndividualContent?ItemClass={itemClass}&ParentClass={parentClass}&SiteId={siteId}&Lang={lang}&IpAddressClient={ipAddress}&Token={token}")]
    Task<List<IndividualPropertiesDTO>> GetIndividualContent(string itemClass, string parentClass, string siteId, string lang, string ipAddress, string token);

    [Get("/api/PageContent/FullOntology?SiteId={siteId}&Lang={lang}&IpAddressClient={ipAddress}&Token={token}")]
    Task<List<OntologyDTO>> GetFullOntology(string siteId, string lang, string ipAddress, string token);
    #endregion
}
```

A continuación se muestran los métodos con que puede interactuar el cliente web, o página web a que accede el usuario. Como se puede observar, estos métodos están directamente relacionados con lo que se expone en el controlador.

```
public interface IApiClientService
{
    #region Public
    #region Content
    #region GET
    Task<List<OntologyDTO>> GetBaseContent(PublicContentInputDTO dto, string secretKey);
    Task<List<OntologyDTO>> GetClassContent(PublicContentInputDTO dto, string secretKey);
    Task<List<IndividualPropertiesDTO>> GetIndividualContent(PublicContentInputDTO dto, string secretKey);
    Task<List<OntologyDTO>> GetFullOntology(PublicContentInputDTO dto, string secretKey);
    #endregion
    #endregion
    #endregion
}
```

Esta clase implementa la interfaz anterior, y define cada método accesible desde los controladores.

```
public class ApiClientService : IApiClientService
{
    private readonly AppSettings _appSettings;

    public ApiClientService(IOptions<AppSettings> options)
    {
        _appSettings = options.Value;
    }

    public IApiMethods Rest => RestService.For<IApiMethods>(_appSettings.WebApiBaseUrl);
}
```

```

public IApiMethods RestAuth(string token) => RestService.For<IApiMethods>(_appSettings.WebApiBaseUrl,
new RefitSettings()
{
    AuthorizationHeaderValueGetter = () => Task.FromResult(token)
});
public IApiMethods RestAuthXML(string token) => RestService.For<IApiMethods>(_appSettings.WebApiBaseUrl,
new RefitSettings()
{
    AuthorizationHeaderValueGetter = () => Task.FromResult(token),
    ContentSerializer = new XmlContentSerializer()
});

#region Content
public async Task<List<OntologyDTO>> GetBaseContent(PublicContentInputDTO dto, string secretKey)
{
    dto.Token = TokenEncrypter.Encrypt(dto, secretKey);
    return await Rest.GetBaseContent(dto.SiteId, dto.Lang, dto.IpAddressClient, dto.Token);
}
public async Task<List<OntologyDTO>> GetClassContent(PublicContentInputDTO dto, string secretKey)
{
    dto.Token = TokenEncrypter.Encrypt(dto, secretKey);
    return await Rest.GetClassContent(dto.ItemClass, dto.SiteId, dto.Lang, dto.IpAddressClient, dto.Token);
}
public async Task<List<IndividualPropertiesDTO>> GetIndividualContent(PublicContentInputDTO dto,
string secretKey)
{
    dto.Token = TokenEncrypter.Encrypt(dto, secretKey);
    return await Rest.GetIndividualContent(dto.ItemClass, dto.ParentClass, dto.SiteId,
dto.Lang, dto.IpAddressClient, dto.Token);
}

```



```
}  
public async Task<List<OntologyDTO>> GetFullOntology(PublicContentInputDTO dto, string secretKey)  
{  
    dto.Token = TokenEncrypter.Encrypt(dto, secretKey);  
    return await Rest.GetFullOntology(dto.SiteId, dto.Lang, dto.IpAddressClient, dto.Token);  
}  
#endregion  
}
```

## **6. Consideraciones finales**

Con el presente trabajo de fin de master se ha propuesto el diseño e implementación de una ontología para la gestión del conocimiento en el sector turismo y hostelería. Este se considera el primer paso para desarrollar una aplicación en el sector que utilice tecnologías de Web Semántica, con el fin de promover sistemas con mayor conectividad, donde el conocimiento se mantenga fluyendo. Aquí se destacan las conclusiones principales que se han obtenido como resultado de este trabajo, de igual forma se indican una serie de trabajos futuros que pueden llevarse a cabo como continuación de este trabajo.

### **6.1. Conclusiones**

Este trabajo se fundamenta en los conceptos de la web semántica, para diseñar e implementar una ontología que responda a las necesidades del sector turismo. De esta forma, se trata de facilitar la interoperabilidad entre aplicaciones de este tipo, que permitiría a los clientes o usuarios finales disponer de herramientas de hostelería que se fundamenten en el conocimiento y en una total interacción entre cada eje del sector.

Para un mayor control sobre el dominio tratado se han recopilado textos que sirven para cubrir la fase de adquisición del conocimiento fundamental para la conceptualización del dominio en el que se desarrolla la ontología. Este trabajo presenta unas directrices claras para la creación de una ontología con las características descritas desde un punto de vista preliminar y en relación con una metodología concreta, fruto del análisis realizado.

Para llegar a la implementación de esta ontología se ha realizado un estudio sobre los principales elementos que componen el sector turismo y hostelería, identificando los actores que intervienen, así como las principales funcionalidades que realizan.

La ontología propuesta se ha desarrollado utilizando el editor de ontologías Protégé, con el lenguaje OWL, ya que es ampliamente utilizado en distintos proyectos. Se ha hecho uso de la metodología METHONTOLOGY para la construcción de la ontología por los motivos expuestos en el apartado 3.2. De forma particular:

- Se ha hecho un análisis previo de la ontología a desarrollar, definiendo aspectos como su dominio, entidades principales y alcance.
- Se ha realizado una conceptualización de la ontología, identificando los conceptos y clases principales, así como las relaciones entre estos conceptos, sus características y sus atributos.

- Se ha formalizado dicha conceptualización, definiendo una jerarquía de clases (taxonomía) para la ontología.
- Se han identificado las propiedades de objeto que relacionan los individuos o instancias de dichas clases. Asimismo, se han identificado las propiedades de tipo de datos de las mismas.
- Se han establecido axiomas sobre estas propiedades, como dominio, rango u otro tipo de restricciones (de valor, cardinalidad, etc).

Todo esto muestra la utilidad de la primera parte de este trabajo, que es el diseño de una ontología que gestione el conocimiento en el sector turismo y hostelería. Adicionalmente, se puede identificar otro tipo de contribución en este trabajo, que es la implementación de la ontología en la web, haciendo uso de tecnologías de la Web Semántica, que compone la segunda fase del proyecto realizado.

Durante la implementación de la ontología en una página web se siguió un desarrollo basado en metodologías ágiles, y haciendo uso de las tecnologías más adecuadas para el tipo de proyecto. De esta página web se pueden sacar las siguientes conclusiones:

- Se ha desarrollado acorde a los patrones de diseño actual, y haciendo uso de principios que promueven un código limpio y bien estructurado, de igual forma es una página web basada en capas y con separación de responsabilidades entre la lógica y la visualización, lo que ofrece un servicio rápido y de fácil integración.
- La extracción del conocimiento en la aplicación lógica ofrece la posibilidad de reutilizar este proyecto para otras ontologías, siempre y cuando se haga una creación previa de las entidades necesarias.
- La implementación llevada a cabo a nivel visual se basa en el uso de frameworks actuales, que permiten que sea una página moderna y a la vez de fácil interacción para el usuario final.

Lo más retador y a la vez interesante de realizar este proyecto ha sido todo el camino recorrido para manipular de forma adecuada la librería que ha permitido la integración de la ontología en la aplicación desarrollada (dotNetRDF).

Desde una perspectiva general, se considera que este trabajo aborda todos los objetivos planteados inicialmente, de igual forma cubre de forma estructurada la mayor parte de los elementos que se deben tomar en cuenta cuando se construye una ontología y cuando se desarrolla una aplicación informática.

## 6.2. Futuras líneas de trabajo

Se sugiere la posibilidad de continuar futuras líneas de investigación que aborden el uso de una metodología más adecuada para la extracción del conocimiento de la ontología en la implementación web.

A nivel de diseño de la ontología sería interés poder incluir entidades que representen los procesos de búsqueda de disponibilidad de alojamiento, el proceso de reserva, así como lo relacionado a servicios adicionales que se pueden reservar.

En otro orden, al crear una ontología existe la posibilidad de agregarle imágenes a un individuo; en esta versión del proyecto no se están extrayendo las imágenes colocadas, debido a un fallo que genera la librería dotNetRDF al ejecutar una consulta que intenta consultar alguna, sin embargo, al ejecutar la misma consulta dentro de Protégé, la misma se ejecuta sin generar errores.

Otro de los particulares para futuras mejoras, es la optimización de las consultas que extraen el conocimiento desde la ontología, puesto que actualmente el tiempo de procesamiento no es el más óptimo, aunque podría serlo si se aplican diversas mejoras en la metodología de extracción y conversión de los datos.

Resultaría de mucho interés la posibilidad de filtrar los datos que se muestran en la página web, lo que generaría una interacción más dinámica entre el usuario y el conocimiento de la ontología. Para mejoras futuras resulta conveniente poder buscar por las diferentes propiedades de objeto, de datos, o por cualquier entidad de la ontología.

Esta ontología debería evaluarse y ponerse a prueba haciendo uso de otras herramientas para la detección de problemas relacionados con la gestión del sector turismo. Por otra parte, la evaluación de la ontología se ha realizado teniendo en cuenta datos que no reflejan al 100% la realidad, sin embargo, se considera de gran utilidad plantear una evaluación más profunda de la ontología, en la que se haga uso de datos totalmente reales.

Para realizar un análisis exhaustivo sobre el desarrollo e implementación de esta ontología se considera de gran utilidad:

- Contar con el punto de vista de especialistas del sector turismo, lo que permitirá refinar el diseño de la ontología, además, contar con usuarios que utilicen la aplicación desarrollada con el fin de evaluar su utilidad y correcto funcionamiento.

Finalmente, este trabajo se ha realizado teniendo en cuenta las limitaciones explicadas y cubriendo los objetivos planteados inicialmente.



## 7. Referencias

- [1] ¿Qué es la la web Semántica?, Julio 2020. <https://www.ceupe.com/blog/que-es-la-la-web-semantica.html?dt=1658937711671>. Última visita Mayo 2022.
- [2] Guía de la Web Semántica, Enero 2015. <https://ceweb.br/guias/web-semantica/es/capitulo-4/>. Última visita Mayo 2022.
- [3] Ontologías en la Web Semántica, Mayo 2017. <http://eolo.cps.unizar.es/docencia/MasterUPV/Articulos/Ontologias%20en%20la%20Web%20Semantica.pdf>. Última visita Junio 2022.
- [4] Ontologías y web semántica, Febrero 2016. <https://www.cartagena99.com/recursos/alumnos/temarios/Ontologias%20y%20web%20semantica.pdf>. Última visita Junio 2022.
- [5] SEMANTIC WEB, Enero 2015. <https://www.w3.org/standards/semanticweb/>. Última visita Junio 2022.
- [6] RDF Schema, Diciembre 2021. [https://es.wikipedia.org/wiki/RDF\\_Schema](https://es.wikipedia.org/wiki/RDF_Schema). última visita Junio 2022.
- [7] RDF y RDF schema, Junio 2003. [https://www.matem.unam.mx/~grecia/semantic\\_web/rdf.html](https://www.matem.unam.mx/~grecia/semantic_web/rdf.html). Última visita Junio 2022.
- [8] Ontology Development 101: A Guide to Creating Your First Ontology. [https://protege.stanford.edu/publications/ontology\\_development/ontology101.pdf](https://protege.stanford.edu/publications/ontology_development/ontology101.pdf). Última visita Junio 2022.
- [9] Creación de ontologías con Protégé: conceptos básicos, Agosto 2014. <https://documentalistaparaboss.blogspot.com/2014/08/creacion-de-ontologias-con-protege.html>. Última visita Junio 2022.
- [10] Domain Driven Design o Dominio, Dominio y Dominio, Enero 2022. <https://jeronimopalacios.com/software/domain-driven-development/>. Última visita Julio 2022.
- [11] Create Gantt Project. <https://prod.teamgantt.com/>. Última visita Julio 2022.
- [12] GENERALIDADES DEL TURISMO. <https://sites.google.com/site/misitiowebelizaracely/our-story-2/about-the-bride>. Última visita Mayo 2022.

[13] Características generales del turismo. <https://www.ceupe.com/blog/caracteristicas-generales-del-turismo.html>. Última visita Mayo 2022.

[14] Inteligencia artificial aplicada a la web semántica, Mayo 2021. <https://gamco.es/inteligencia-artificial-aplicada-web-semantica/>. Última visita Julio 2022.

[15] Contreras, J., Tutorial de Ontologías. Recuperado de [http://eprints.rclis.org/11152/1/Tutorial\\_def\\_Ontologias\\_Protege.pdf](http://eprints.rclis.org/11152/1/Tutorial_def_Ontologias_Protege.pdf). Última visita Mayo 2022.

[16] Sistemas Inteligentes II. (14 Abril 2020) Ontologia OWL parte I. Youtube. <https://www.youtube.com/watch?v=WY3I4GsVAN4>. Última visita Mayo 2022.

[17] Sistemas Inteligentes. (14 Abril 2020) Ontologia OWL parte II. Youtube. [https://www.youtube.com/watch?v=bHf6I53k\\_yA](https://www.youtube.com/watch?v=bHf6I53k_yA). Última visita Mayo 2022.

[18] Sistemas Inteligentes II. (14 Abril 2020) Ontologia OWL parte III. Youtube. [https://www.youtube.com/watch?v=e\\_eKQ1SMNDg](https://www.youtube.com/watch?v=e_eKQ1SMNDg). Última visita Mayo 2022.

[19] Microsoft .NET, Enero 2022. <https://www.clarcat.com/microsoft-net/>. Última visita Junio 2022.

[20] Qué es la plataforma .NET. <https://www.campusmvp.es/recursos/post/que-es-la-plataforma-net-y-cuales-son-sus-principales-partes.aspx>. Última visita Junio 2022.

[21] Qué es MVC, Julio 2020. <https://desarrolloweb.com/articulos/que-es-mvc.html>. Última visita Junio 2022.

[22] Linked Data y Linked Open Data, Junio 2016. <https://universoabierto.org/2016/06/20/linked-data-y-linked-open-data/>. Última visita Mayo 2022.

[23] ¿Qué es el Linked Data?. <https://solvenup.com/linked-data-y-el-linked-open-data/>. Última visita Mayo 2022.

[24] Getting Started with dotNetRDF. [https://dotnetrdf.org/docs/2.7.x/user\\_guide/Getting-Started.html](https://dotnetrdf.org/docs/2.7.x/user_guide/Getting-Started.html). Última visita Agosto 2022.

[25] Inference and Reasoning.  
[https://dotnetrdf.org/docs/stable/user\\_guide/Inference-And-Reasoning.html](https://dotnetrdf.org/docs/stable/user_guide/Inference-And-Reasoning.html).  
última visita Agosto 2022.

[26] Código y documentación del proyecto.  
<https://github.com/elvinjreyesv/TFM>.




# 8. Anexos

## 8.1. Diagrama Gantt de planificación del proyecto



## 8.2. Documentación del despliegue del servicio REST


 **Swagger**  
Supported by SMARTBEAR

Select a definition API





# API

TOURISMAPI OAS3

/docs/TOURISMAPI/docs.json

Authorize 

## PageContent

GET	/api/PageContent	
GET	/api/PageContent/ClassContent	
GET	/api/PageContent/IndividualContent	
GET	/api/PageContent/FullOntology	

## Schemas



Object >

ProblemDetails >

```
AttributeValueDTO {  
  value      string  
            nullable: true  
}
```

```
IndividualAttributeDTO {  
  domain      string  
              nullable: true  
  property    string  
              nullable: true  
  propertyType string  
              nullable: true  
  range       string  
              nullable: true  
  value       {  
    nullable: true  
    AttributeValueDTO {  
      value      string  
                nullable: true  
    }  
  }  
}
```

```

PropertyDTO ▾ {
  class          string
                 nullable: true
  parent         string
                 nullable: true
  comment        string
                 nullable: true
  image          string
                 nullable: true
  attributes     ▾ [
                 nullable: true
    IndividualAttributeDTO ▾ {
      domain      string
                   nullable: true
      property    string
                   nullable: true
      propertyType string
                   nullable: true
      range       string
                   nullable: true
      value       ▾ [
                   nullable: true
        AttributeValueDTO ▾ {
          value     string
                     nullable: true
        }
      ]
    }
  ]
}

```

```

IndividualPropertiesDTO ∨ {
  individualName  string
                  nullable: true
  properties      ∨ [
                  nullable: true
    PropertyDTO ∨ {
      class      string
                  nullable: true
      parent     string
                  nullable: true
      comment    string
                  nullable: true
      image      string
                  nullable: true
      attributes  ∨ [
                  nullable: true
        IndividualAttributeDTO ∨ {
          domain    string
                      nullable: true
          property  string
                      nullable: true
          propertyType string
                      nullable: true
          range     string
                      nullable: true
          value      ∨ [
                      nullable: true
            AttributeValueDTO ∨ {
              value  string
                      nullable: true
            }
          ]
        }
      ]
    }
  ]
}
}

```

```

OntologyDTO {
  class string nullable: true
  parent string nullable: true
  comment string nullable: true
  children > [...]
  individuals {
    nullable: true
    IndividualPropertiesDTO {
      individualName string nullable: true
      properties {
        nullable: true
        PropertyDTO {
          class string nullable: true
          parent string nullable: true
          comment string nullable: true
          image string nullable: true
          attributes {
            nullable: true
            IndividualAttributeDTO {
              domain string nullable: true
              property string nullable: true
              propertyType string nullable: true
              range string nullable: true
              value {
                nullable: true
                AttributeValueDTO {
                  value string nullable: true
                }
              }
            }
          }
        }
      }
    }
  }
}

```